Data Storytelling with Generative AI

using Altair and Python

Angelica Lo Duca

MEAP

MANNING

Data Storytelling with Generative AI

using Altair and Python

Angelica Lo Duca

MEAP

MANNING

Data Storytelling with Generative AI

1. welcome
2. 1_Introducing_Data_Storytelling
3. 2_Running_your_first_data_story_in_Altair_and_GitHub_Copilot
4. 3_Reviewing_the_basic_concepts_of_Altair
5. 4_Generative_AI_Tools_for_Data_Storytelling
6. 5_Crafting_a_data_story_through_the_DIKW_Pyramid
7. 6_From_Data_to_Information:_Extracting_insights
8. 7_From_Information_to_Knowledge:_Building_textual_context
9. 8_From_Information_to_Knowledge:_Building_the_visual_context
10. 9_From_Knowledge_to_Wisdom:_Adding_Next_Steps
11. 10_Publishing_the_Data_Story
12. Appendix_A._Technical_Requirements
13. Appendix_B._Python_Pandas_DataFrame

welcome

Thank you for purchasing the MEAP for *AI-Powered Data Storytelling*. This book combines three fundamental elements: Data Storytelling, Python, and Generative AI to guide you toward building engaging data stories.

Data Storytelling is the main focus of this book. Many good books exist on the market about this topic, especially focusing on the theory of data storytelling. This book, instead, describes a practical approach to elevate your basic data visualization skills and transform you into a data storyteller. In this book, you will use the Data-Information-Knowledge-Wisdom (DIKW) pyramid as a practical method to transform simple data visualization into data stories. When I met the DIKW pyramid some years ago, I fell in love with it, and since then, I have been using it as a teaching method for my students.

Python is the second ingredient of this book. As a data journalism professor, I have experimented with many Python libraries for data visualization, such as Matplotlib, Plotly, and Seaborn. However, at the end of my experiments, I realized that the simplest library for data visualization is Altair. Unlike other libraries, Altair is declarative, thus enabling you only to focus on the output of your data visualization. Thus, this book will focus on Altair to build data visualization. Although you can find many materials on the web about Altair, they focus only on how to build raw charts. In this book, you'll learn how to build data stories with Altair and not simply raw charts.

Last but not least, you'll learn how to use Generative AI tools for data storytelling. This book won't focus on Generative AI concepts and theory. Instead, you will learn how to apply Generative AI tools, including GitHub Copilot, ChatGPT, and DALL-E, to data storytelling.

Finally, throughout *AI-Powered Data Storytelling*, you'll implement many practical examples that will ease your learning process and make your journey more fun.

To benefit most from this book, you'll want to have some basic experience in

Python and basic knowledge about Pandas and data visualization. Even if you don’t have such basic knowledge, you can still read this book, assimilating the basic concepts of data storytelling and then using them with other tools or programming languages.

I hope you find it as useful to read as I did to write it. Please be sure to post any questions, comments, or suggestions you have about the book in the liveBook discussion forum. Your feedback is essential in developing the best book possible.

—Angelica Lo Duca

In this book

welcome 1 Introducing Data Storytelling 2 Running your first data story in Altair and GitHub Copilot 3 Reviewing the basic concepts of Altair 4 Generative AI Tools for Data Storytelling 5 Crafting a data story through the DIKW Pyramid 6 From Data to Information: Extracting insights 7 From Information to Knowledge: Building textual context 8 From Information to Knowledge: Building the visual context 9 From Knowledge to Wisdom: Adding Next Steps 10 Publishing the Data Story
Appendix A. Technical Requirements Appendix B. Python Pandas DataFrame

1 Introducing Data Storytelling

This chapter covers

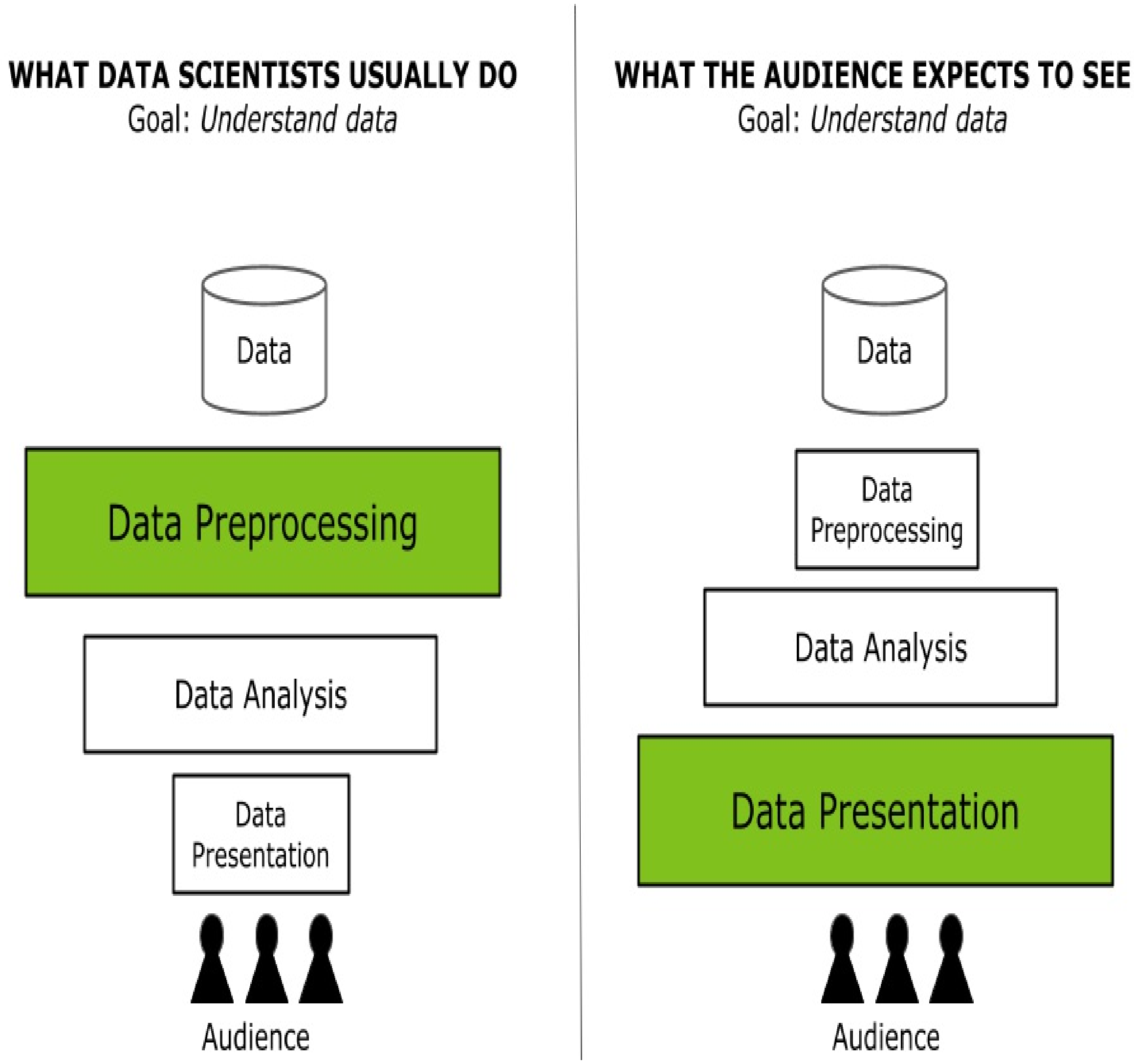
- Overview of data storytelling
- The importance of data storytelling
- Why you should use Python Altair and Generative AI tools for data storytelling
- When Altair and Generative AI tools are not useful for data storytelling
- How to read this book
- The DIKW pyramid

Data storytelling is a powerful way to share insights by transforming them into narrative stories. Data storytelling is an art you can use for any industry, such as government, education, finance, entertainment, and healthcare. Data storytelling is not only for data scientists and analysts; it’s for anyone who has ever wanted to tell a story with data.

1.1 The Art of Data Storytelling

Data scientists and analysts often allocate little time and attention to data presentation. Frequently, throughout a data science project, we devote most of the time and resources to preprocessing the data, followed by data analysis, and lastly, data presentation. Unfortunately, this often entails presenting graphs generated during the analytical phase to the audience, resulting in limited engagement and understanding. Figure 1.1 shows the journey from data to the intended audience, as viewed through the eyes of both the skilled data scientist (represented on the left) and the eagerly awaiting audience (represented on the right). The flow consists of three major phases: data preprocessing, analysis, and presentation. The size of each box reflects the amount of time devoted to its respective phase. While data scientists and audiences share a common goal (that is, to grasp the essence of the data truly), how they achieve said goal varies. Data scientists understand data during the data analysis phase, whereas audiences take center stage during data presentation.

Figure 1.1 The data science flow from the data scientist's perspective (on the left) and the audience's perspective (on the right)



Data storytelling can help data scientists and analysts to present and communicate data to an audience. You can think of data storytelling as the grand finale of the data science life cycle. It entails taking the results of the previous phases and transforming them into a narrative that effectively communicates the results of data analysis to the audience. Rather than relying on dull graphs and charts, data storytelling enables you to bring your data to life and communicate insights compellingly and persuasively.

More formally, data storytelling builds compelling stories supported by data, allowing analysts and data scientists to present and share their insights interestingly and interactively. The ultimate goal of data storytelling is to engage the audience and inspire them to make decisions. In some cases, including business scenarios, you do not start from data but from a narrative or a hypothesis you have in mind and then search for data that confirm or negate it. In this case, you can still have data storytelling, but you must pay attention not to alter your data to support your hypothesis. Brent Dykes, a well-known consultant in storytelling training, suggests: *Whenever you start with the narrative and not the data, it requires discipline and open-mindedness. In these scenarios, one source of risk will be confirmation bias. You will be tempted to cherry-pick data that confirms your viewpoint and ignore conflicting data that doesn’t* (Dykes 2023). Remember to build your data stories on accurate and unbiased data analysis. In addition, always consider the data you are analyzing.

Some time ago, I had the opportunity to work on a cultural heritage project where I had to automatically extract entities from the transcripts of a registry of names dating back to around 1700-1800. The goal was to build a database with information about the people in the register, such as name, surname, date of birth, date of death, etc. Sitting at my computer, I implemented my software, built the table with all the names, and calculated and visualized data statistics, such as the most frequent names, the number of births per year, etc. The project also involved linking these people to their graves to build an interactive cemetery map. At some point in the project, I had the opportunity to visit the cemetery. As I walked through the cemetery, the rows upon rows of headstones made me stop. It hit me like a ton of bricks - every name etched into those stones represented a life. Suddenly, the numbers and statistics I had been poring over in my data sets became more than just data points - they were the stories of real people. It was a powerful realization that changed the way I approached my work. That's when I discovered the true power of data storytelling. It's not just about creating fancy graphs and charts - it's about bringing the people behind the data to life. We have a mission to give these people a voice, to ensure that their stories are heard. And that's exactly what data storytelling does - it gives a voice to people often buried deep within the numbers. Our mission as data storytellers is to bring these stories to the forefront and ensure they are heard loud and clear.

In this book, you’ll learn two technologies to transform data into stories: Python Altair and Generative AI tools. Python Altair is a Python library for data visualization. Unlike the most known Python libraries, such as Matplotlib and Seaborn, Altair is a declarative library where you specify only what you want to see in your visualization. This aspect is beneficial for quickly building data stories without caring about how to build a visualization. Altair also supports chart interactivity so that users can explore data and interact with it directly.

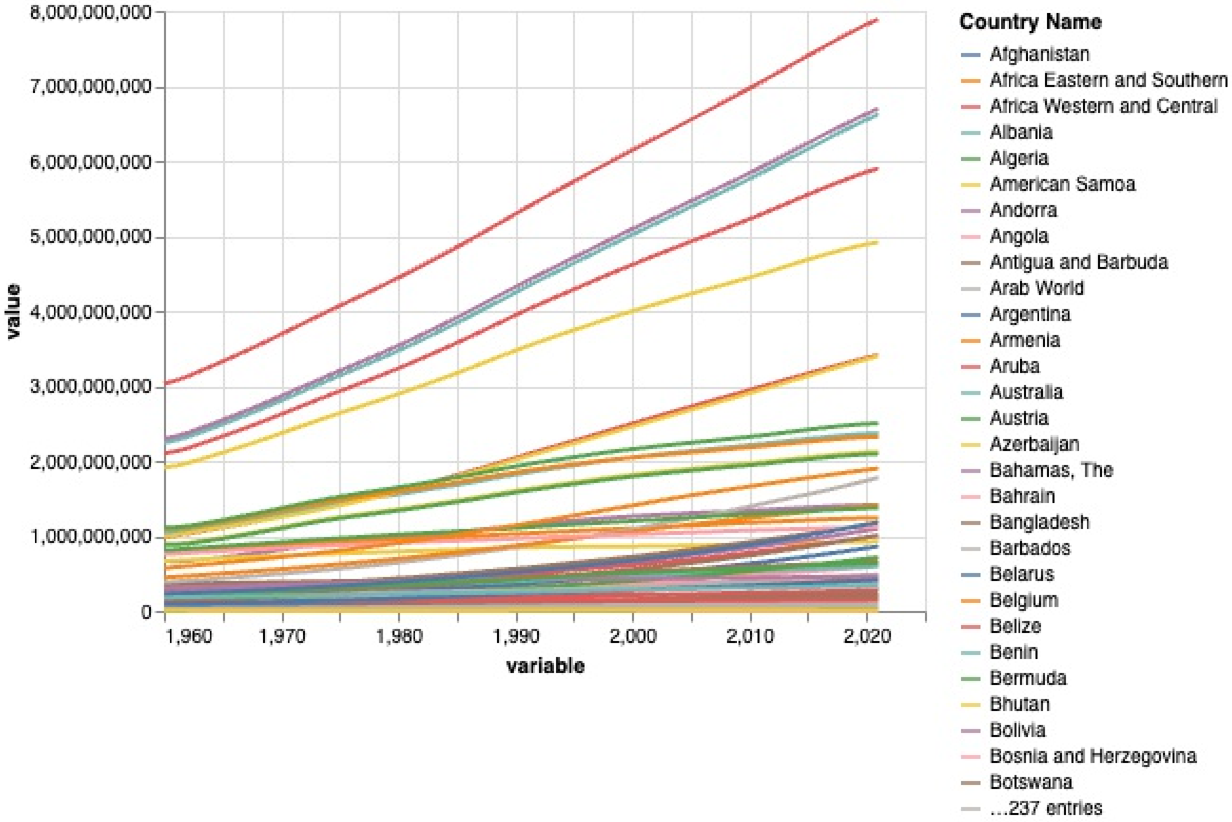
Generative AI is the second technology you’ll use to build data stories in this book. We will focus on ChatGPT to generate text, DALL-E to generate images, and GitHub Copilot to generate the Altair code automatically. We chose to use GitHub Copilot to generate code and not ChatGPT because Copilot was trained with domain-specific texts, including GitHub and StackOverflow codes. Instead, ChatGPT is more general purpose. At the time of writing, Generative AI is a very recent technology, still in progress, that translates a description of specifications or actions into text.1.2 Why Data Visualization Alone Is Not Enough

Data storytelling is about more than just communicating data. It's about inspiring your audience and inviting them to take action. Good data storytelling requires a mix of art and science. Art comes in finding the right story, while science understands how to use data to support that story. When done well, data storytelling can be a potent tool for change. In this section, we’ll briefly cover three fundamental questions about data storytelling.

1.1.1 Why Should You Use Data Storytelling?

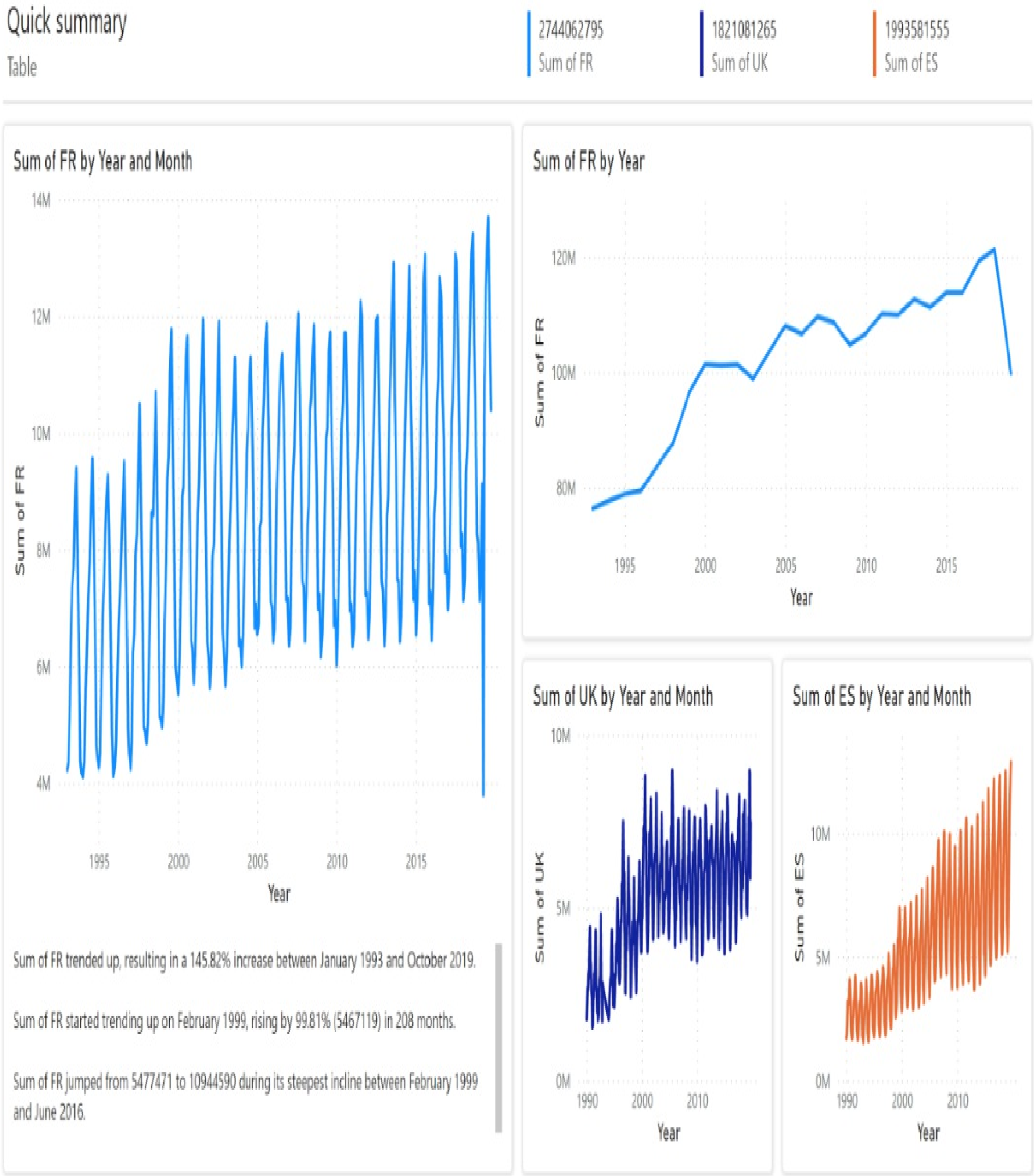
More often than not, we delay data visualization until the last stage of our data analysis process, when we're tired and exhausted from the other stages of our project. As a result, we produce running graphs, which more or less represent our data. Figure 1.2 shows an example of such a running chart, where it is really difficult to extract a meaning.

Figure 1.2 An example of a bad chart



In other cases, we build dashboards that simply juxtapose charts next to each other without any sense or logic (Figure 1.3).

Figure 1.3 An example of a bad dashboard



Sum of FR trended up, resulting in a 145.82% increase between January 1993 and October 2019.

Sum of FR started trending up on February 1999, rising by 99.81% (5467119) in 208 months.

Sum of FR jumped from 5477471 to 10944590 during its steepest incline between February 1999 and June 2016.

There is often the erroneous belief that if a chart is aesthetically beautiful, then it is certainly also useful. This isn't exactly true. In his book *The Functional Art*, Alberto Cairo says: *If you accept that a visualization is, above all, a tool, you are implicitly accepting that the discipline it belongs to is not just art, but functional art, something that achieves beauty not through* *the subjective, freely wandering self-expression of the painter or sculptor, but through the careful and restrained tinkering of the engineer* (Cairo 2012).

Data Storytelling allows you to fill the gap between simply visualizing data and communicating it to an audience. Data storytelling improves your communication skills and standardizes and simplifies the process of communicating results, making it easier for people to understand and remember information. Data storytelling also helps you learn to communicate more effectively with others, improving personal and professional relationships.

Use data storytelling if you want to:

- Focus on the message you want to communicate and make data more understandable and relatable
- Communicate your findings to others in a way that is clear and convincing
- Connect with your audience on an emotional level, which makes them more likely to take action
- Make better decisions by helping you understand your data more deeply.

1.1.2 What Problems Can Data Storytelling Solve?

Use data storytelling if you want to communicate something to an audience in the form of writing reports, doing presentations, and building dashboards.

Writing Reports

Imagine you must write a sales report for a retail company. Instead of presenting raw numbers and figures, you can weave a data story around the performance of different product categories. Start by identifying the most crucial aspects of the data, such as the top-selling products, emerging trends, or seasonal fluctuations. Then, use a combination of visualizations, anecdotes, and a logical narrative flow to present the information.

You can build a story around your data, such as introducing a problem, building suspense, and concluding with actionable recommendations.

When writing reports, use data storytelling to highlight the most important parts of your data and make your reports more engaging and easier to understand.

Doing Presentations

Consider a marketing presentation where you must demonstrate various marketing campaigns' effectiveness. Instead of bombarding the audience with numerous charts and statistics, focus on creating a compelling narrative that guides them through how the campaigns unfolded and their impact on the target audience. Finally, show the possible next steps to follow.

When doing presentations, use data storytelling to engage your audience and help them understand your message better.

Building Dashboards

Let's imagine you are developing a sales performance dashboard for a retail company. Instead of presenting a cluttered interface with overwhelming data, focus on guiding users through a narrative highlighting key insights.

When building dashboards, use data storytelling to build more user-friendly and informative dashboards.

1.1.3 How Is Data Storytelling Different From Data Visualization?

Data visualization is primarily concerned with presenting data in a way that is easy to understand, usually through charts and graphs. On the other hand, data storytelling is more concerned with using data to tell a story that will engage the audience and help them understand the problem. In addition, data storytelling often uses narratives to explain the data, while data visualization focuses on presenting the data itself. Anyway, good data storytelling includes visualization elements.

Data storytelling helps analysts to communicate their findings in a way that is both engaging and informative while also helping the audience understand a problem in a new way and see the potential solutions more clearly.

Having covered the questions of when to use data storytelling, what problems it can solve, and what makes it unique, we're now ready to consider questions relating to our two tools: Python Altair and Generative AI tools. We will do that in the next section.

1.1.4 What are the challenges of data storytelling?

Crafting compelling data stories is no easy feat. It demands both time and financial investment to ensure engaging narratives packed with valuable information. Moreover, it involves bringing together individuals from diverse backgrounds, each with their own expertise and perspectives, to work collaboratively. This collaboration can be challenging, but it's essential for weaving together a cohesive and impactful data story.

The challenges of creating data stories involve two key factors: time and teamwork. Investing in these areas is crucial to captivate audiences and effectively communicate insights.

1.2 Why Should You Use Python Altair and Generative AI for Data Storytelling?

Python provides you with many libraries for data visualization. Many of them, including Matplotlib and Seaborn, are *imperative libraries*, meaning you must define exactly how you want to build a visualization. Python Altair, instead, is a *declarative library* where you specify only what to visualize. Using Python Altair for data storytelling instead of other imperative libraries lets you quickly build your visualizations.

For example, to plot a line chart using Matplotlib, you must specify explicitly the x and y coordinates, set the plot title, labels, and customize the appearance, as shown in Listing 1.1.

Listing 1.1 Imperative library

```
import matplotlib.pyplot as plt
```

```
x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]

plt.plot(x, y)
plt.title('Square Numbers')
plt.xlabel('X')
plt.ylabel('Y')

plt.show()
```

An example of an imperative library

The chart builds a line chart in Matplotlib. You must define the single steps to build the chart: 1) set the title, 2) set the x-axis, 3) set the y-axis.

Instead, declarative visualization libraries, like Altair, enable you to define the desired outcome without specifying the exact steps to achieve it. For instance, using Altair, you can simply define the data, x and y variables, and let the library handle the rest, including axes, labels, and styling, resulting in a more concise and intuitive code (Listing 1.2).

Listing 1.2 Imperative library

```
import altair as alt
import pandas as pd

df = pd.DataFrame({'x': [1, 2, 3, 4, 5], 'y': [1, 4, 9, 16, 25]})

chart = alt.Chart(df).mark_line().encode(
    x='x',
    y='y'
).properties(
    title='Square Numbers'
)
```

An example of a declarative library

The chart builds a line chart in Altair. You must define the chart type (`mark_line`), the variables, and the title.

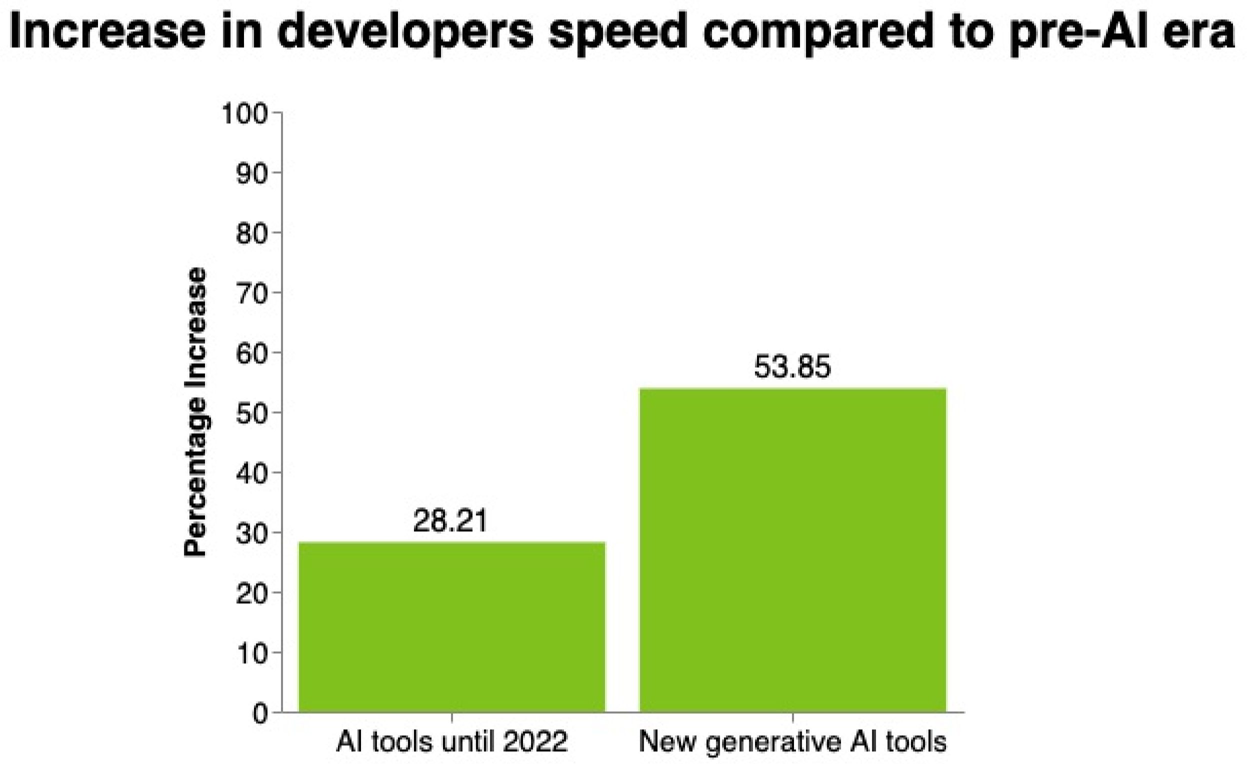
Generative AI is a subset of artificial intelligence techniques involving creating new, original content based on patterns and examples from existing data. It enables computers to generate realistic and meaningful outputs, such as text, images, or even code. In this book, we will focus on ChatGPT to generate text, DALL-E to generate images, and GitHub Copilot, to assist you while coding.

- *ChatGPT* is an advanced language model developed by OpenAI. Powered by the GPT-3.5 architecture, it is designed to engage in human-like conversations and provide intelligent responses.
- *DALL-E* is a generative AI model created by OpenAI. It combines the power of GPT-3 with image generation capabilities, allowing it to create unique and realistic images from textual descriptions.
- *GitHub Copilot* is a recent tool powered by OpenAI Codex, assisting you while writing your code. In GitHub Copilot, you describe the sequence of actions that your software must run, and GitHub Copilot transforms it into a runnable code in your preferred programming language. The ability to use GitHub Copilot consists in learning how to describe the sequence of actions.

Combining Python Altair and Generative AI tools will enable you to write compelling data stories more quickly directly in Python. For example, we can use Copilot to assist us in generating the necessary code snippets, such as importing the required libraries, setting up the plot, and labeling the axes. In addition, Copilot's contextual understanding helps it propose relevant customization options, such as adding a legend or changing the color scheme, saving us time and effort in searching for documentation or examples.

While generative AI tools are still at their early stages, some promising statistics reveal they increase workers' productivity. A study by Duolingo, one of the largest language learning apps, reveals that the use of Copilot in their company has increased developers’ speed of 25% (Duolingo and GitHub Enterprise 2022). Compass UOL, a digital media and technology company, run another study by asking experienced developers to measure the time it took to complete the use-case task (analysis, design, implementation, testing, and deployment) during three distinct periods: before AI, utilizing AI tools available until 2022, and employing new generative AI tools such as ChatGPT (Compass UOL 2023). Results demonstrated that developers completed the tasks in 78 hours before AI; 56 hours with the AI used through 2022, and 36 hours with the new Generative AI. Compared to the Pre-AI era, there is an increase in speed of 53.85% with the new Generative AI (Figure 1.4).

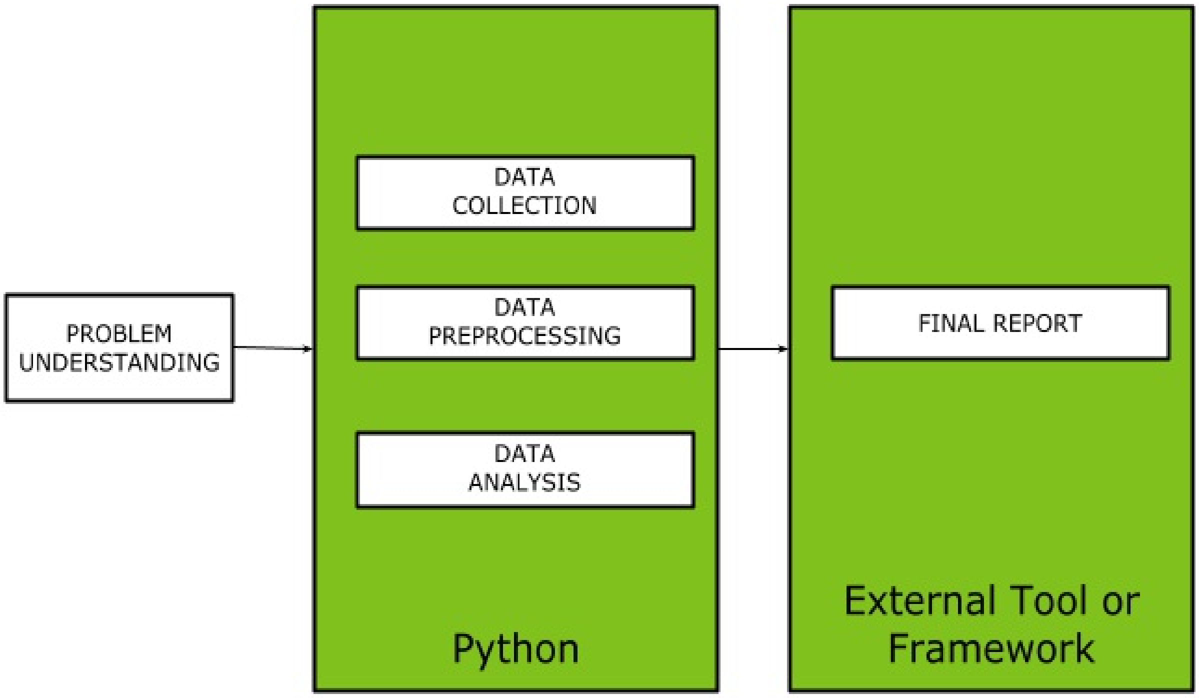
Figure 1.4 The results of the tests conducted by Compass UOL



1.2.1 The Benefits of Using Python in All the Steps of the Data Science Project Life Cycle

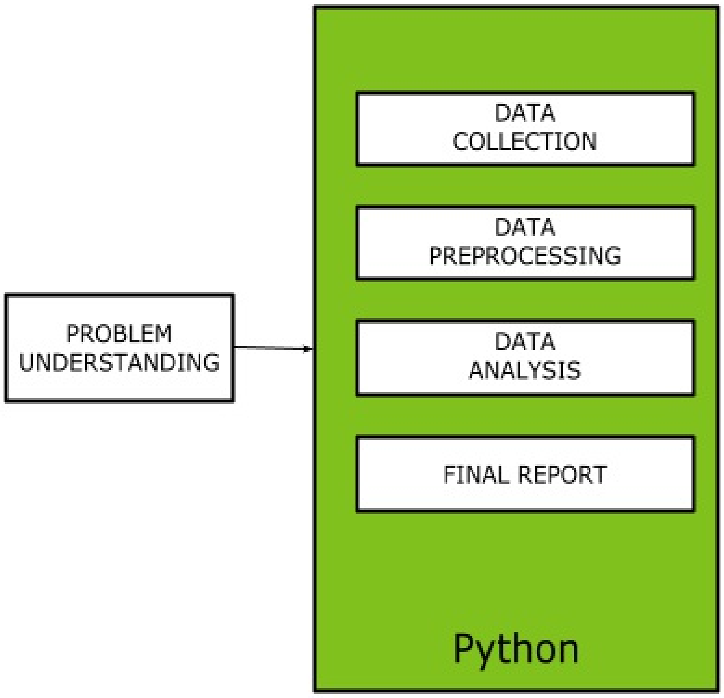
Many data scientists and analysts use Python to analyze their data. Thus it should be natural to build the final report on the analyzed data still in Python. However, data scientists and analysts often use Python only during the central phases of the data science project life cycle. Then they move to other tools, such as Tableau and Power BI, to build the final report, as shown in Figure 1.5. This means adding other work, which includes exporting data from Python and importing them into the external application. This export/import operation in itself is not expensive, but if, while building the report, you realize that you have made a mistake, you need to modify the data in Python and then export the data again. If this process is repeated many times, there is a risk of significantly increasing the overhead until it becomes unmanageable.

Figure 1.5 In the traditional approach, data scientists use different technologies during the phases of the data science project life cycle



This book enables data scientists and analysts to run each step of the data science project life cycle in Python, filling the gap of exporting data to an external tool or framework in the last phase of the project life cycle, as shown in Figure 1.6.

Figure 1.6 In the approach proposed in this book, data scientists use the same technology during all the phases of the data science project life cycle



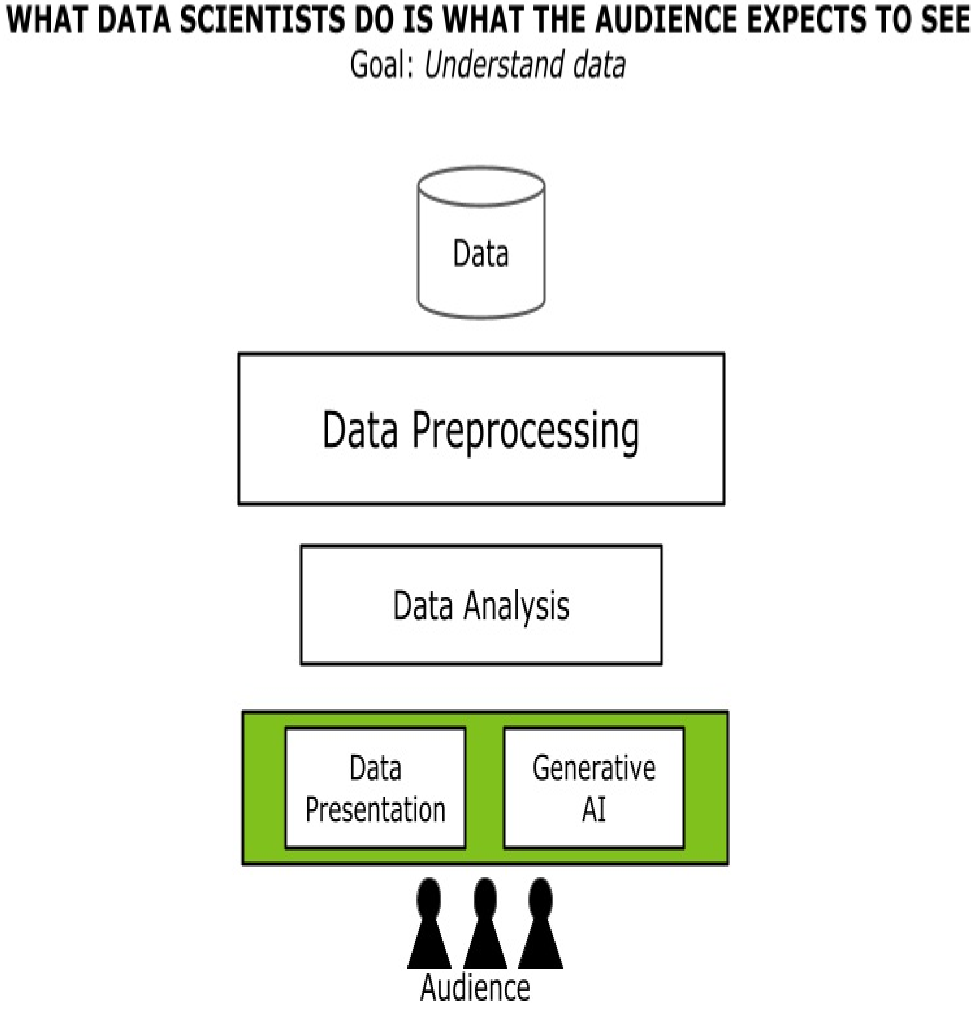
The advantage of using Python Altair and Generative AI for data storytelling is that programmers can build their reports even during the intermediate stages of their experiments without wasting time transferring data to other tools such as Tableau or Power BI.

1.2.2 The Benefits of Using Generative AI for Data Storytelling

In general, you can use Generative AI as an aid throughout all the life cycle of a data science project. However, in this book, we will focus only on generative AI in the data presentation phase, which corresponds to the data storytelling phase.

The introduction of Generative AI tools to aid the data presentation phase allows you to devote the same effort and time as you did without to the data presentation phase, obtaining better results. Thanks to Generative AI tools, you can make the audience understand your data (Figure 1.7).

Figure 1.7 Introducing Generative AI in the data presentation phase helps you to build better charts in less time, enabling the audience to understand your message



Having discussed the benefits of the tools of choice in this book, we should also briefly discuss contexts in which these tools are not as effective.

1.3 When Altair and Generative AI Tools are Not Useful for Data Storytelling

While Python Altair and Generative AI tools are handy for building data stories very quickly, they are not useful when analyzing big data, such as gigabytes of data. You should not use them if you want to:

- Do complex exploratory data analysis: exploratory data analysis helps data analysts summarize a dataset's main features, identify relationships between variables, and detect outliers. This approach is often used when working with large datasets or datasets with many variables.
- Do big data analytics: big data analytics analyzes large datasets to uncover patterns and trends. To be effective, big data analytics requires access to large amounts of data, powerful computers for processing that data, and specialized software for analyzing it.
- Write complex reports that summarize big data.

Altair enables you to build charts with datasets with 5,000 rows at maximum quickly. If the number of rows exceeds 5,000, Altair still builds the chart, but it’s slower.

For complex data analytics, use more sophisticated analytics platforms, such as Tableau and Power BI.

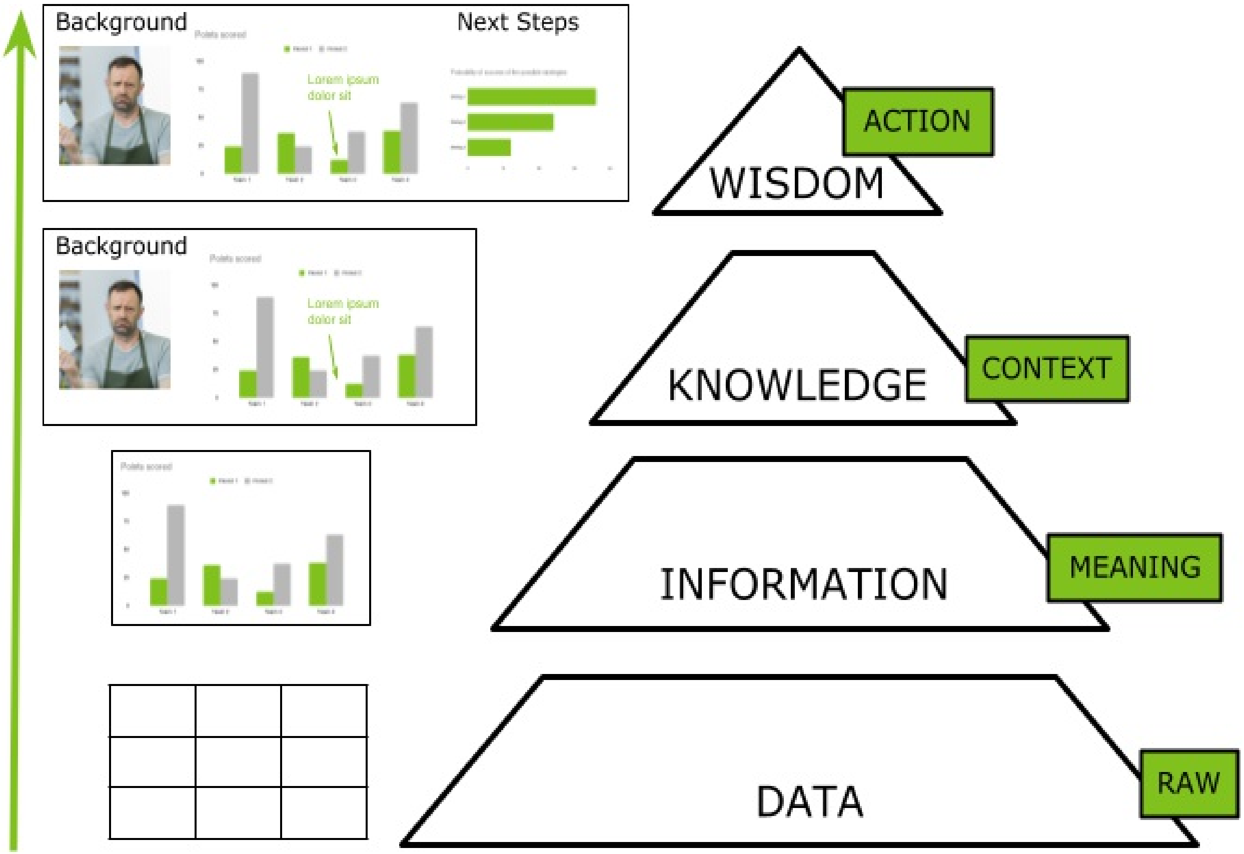
In addition, consider that you should pay a fee to use Generative AI tools, so do not use them if you do not have a sufficient budget.

1.4 Using the DIKW Pyramid for Data Storytelling

The primary focus of this book revolves around a significant concept known as **Data Information Knowledge Wisdom** (DIKW) pyramid (Figure 1.8), which we believe gives data scientists and analysts the macro steps to build data stories. We’ll cover the DIKW pyramid more extensively in Chapter 5, *Crafting a Data Story through the DIKW Pyramid*. Here, we introduce it because it is fundamental to understand how we can use it to build data stories.

The DIKW pyramid provides macro steps to transform data into knowledge, following other intermediate steps, which include information and knowledge.

Figure 1.8 The DIKW pyramid



The DIKW pyramid comprises the following elements:

- **Data** is the building block at the bottom of the pyramid. Usually, we start from a significant amount of data, more or less cleaned
- **Information** involves extracting meaning from data. The information represents organized and processed data that are easy to understand. **Knowledge** is information interpreted and understood through a context that defines the data background
- **Wisdom** is the knowledge enriched with specific ethics that invites you to act in some way. Wisdom also proposes the next steps to follow after we have understood the data.

This book describes how to use the elements of the DIKW pyramid as progressive steps to transform your data into compelling data stories. This idea is not new in data storytelling; Berengueres et al. proposed this approach (Berengueres 2019). The novelty of this book is the combination of the use of the DIKW pyramid, Python Altair, and Generative AI. In this section, we will introduce the fundamentals of DIKW that will be applied throughout this book, and we'll see how to climb each level of the pyramid."

1.4.1 From Data to Information

To turn data into information, extract meaning from data. Consider the following scenario:

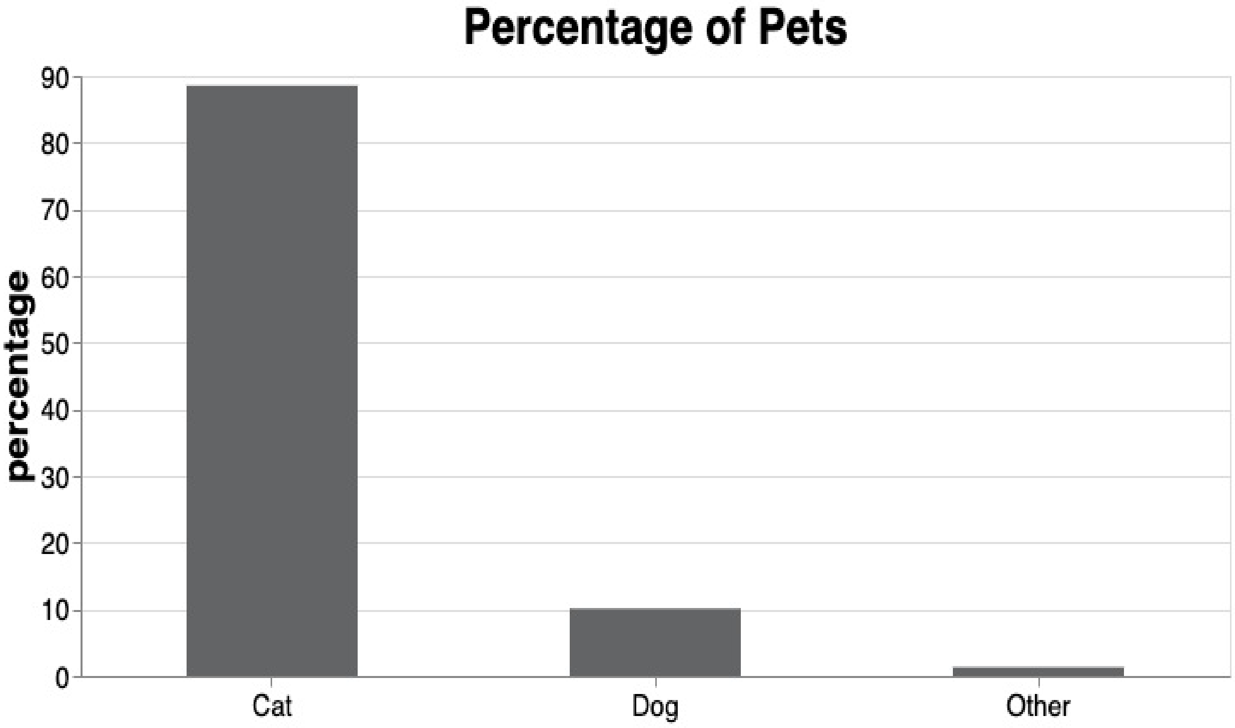
The organizers of an event dedicated to pets are collecting the type of pets that will participate. For each pet category, the organizers advertise the event on specific websites dedicated to that category. The organizers ask you to build a quick report about the current situation. The following table shows the number of participants and the number of advertised websites divided by type.

Table 1.1 Data related to pets involved in the event

Pet	Number of participants	Number of advertised websites
Cat	1138	150
Dog	130	28
Other	17	147

First, you focus on the number of participants. and build the following bar chart.

Figure 1.9 A bar chart showing the number of pets participating in the event



With respect to Table 1.1, Figure 1.9 does not add any meaning. It's simply a visual representation of the table. For some aspects, the table is even clearer than the bar chart.

To turn data into information, start understanding data. Ask the following questions:

- Who is the audience of my story?
- What information do they want?
- Are all the data relevant to answer the previous questions?

Let's answer those questions. The organizers of the event are the audience for our story. They want to know the current situation of animals involved in the event. We could suppose that they want to elaborate a promotion plan to increase the number of pets participating in the event.

Looking at the raw data in the table, you notice that the cat category has the highest number of participants by some margin, followed by a relatively low number of dogs. It seems that owners of other pets, judging by the almost total lack of participants, are not interested in the event.

You may focus on one of the following options:

- Remove the Other category because it is irrelevant
- Focus on the Other category and propose a strategy to increase their participation.

Let’s focus on the first possible course of action, removing the Other category. Now, calculate the percentage of participants for each category, as shown in the following table:

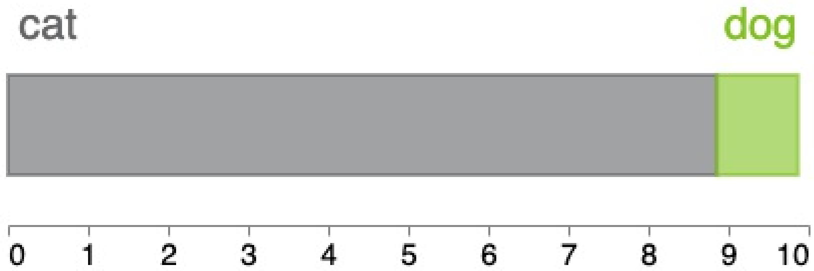
Table 1.2 Data related to pets involved in the event, with a focus on the percentage of the number of participants

Pet	Number of participants	Percentage
Cat	1138	88.56
Dog	130	10.12

Notice that 88.56% of pets are cats and 10.12% are dogs. Usually, the final audience may not be interested in the details of numbers, so approximate cats at 90% and dogs at 10%. The extracted information is that one pet out of 10 is a dog, and 9 out of 10 is a cat. This is the meaning of your data.

Consider the following stacked bar chart describing the situation.

Figure 1.10 A stacked bar chart showing the current percentage of pets participating in the event



The figure is self-explanatory because the audience can understand the situation immediately. You have extracted *information* (and thus, meaning) from the data.

1.4.2 From Information to Knowledge

To turn information into knowledge, add context. Context involves all circumstances around data, such as events, situations, etc. Adding context helps the audience to understand data better.

Consider the previous example of dogs and cats. You already know that the percentage of cats is greater than that of dogs. For example, adding context means describing the events and situations in which motivated dog owners do not participate. Let’s focus on the third column of the dataset described back in Table 1.1, and which is recalled in Table 1.3: the *number of advertised websites*. The number of advertised websites for cats is 150, and for dogs, 28. This information is the context because it helps the audience to understand why the number of cats is higher than the number of dogs.

Calculate the ratio between the number of participants and the number of advertised websites to understand the participant's rate. Table 1.3 shows the calculated values.

Table 1.3 Data related to pets involved in the event, with a focus on the participant's rate

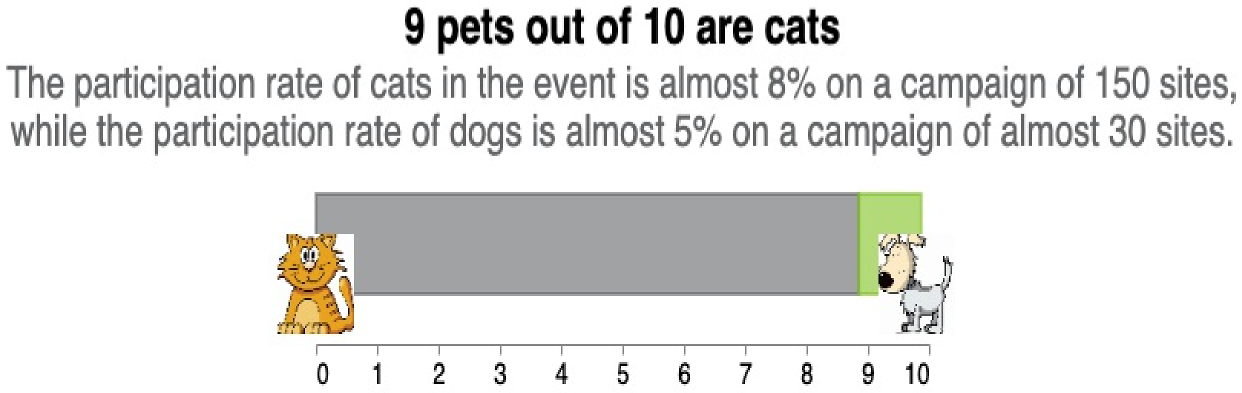
	Number of		Number of advertised	Participant’s rate

Pet	participants	Percentage	websites	#participants / #websites
Cat	1138	88.56	150	7.59
Dog	130	10.12	28	4.64

Cat’s participant rate is 7.59, meaning almost 8. Dog’s participation rate is 4.64, meaning almost 5. The participant rate helps the audience to understand the context related to the number of participants in the event.

The following figure shows the stacked bar chart of Figure 1.10 enriched with context.

Figure 1.11 A stacked bar chart showing the current percentage of pets participating in the event, enriched with context



The context involves the following elements:

- The title, which summarizes the chart content
- The header, which describes the number of advertised websites
- Two images, one for cats and the other for dogs, to facilitate reading.

The content described by the chart in Figure 1.6 is easier to read and understand than that shown in the chart in Figure 1.4.

Adding context to the chart has enabled us to turn information into knowledge.

1.4.3 From Knowledge to Wisdom

Turning knowledge into wisdom means adding a call to action, which invites the audience to do something with the discovered knowledge.

A call to action is a very effective way to help drive conversions. Here are some examples:

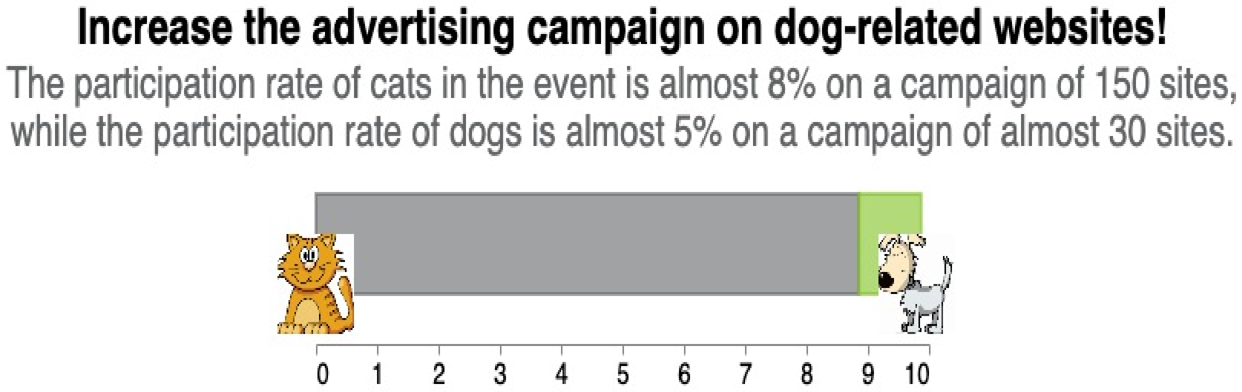
- What changes can we make?
- What opportunities do we have?
- What advantages does our program offer?
- What scenarios can we outline?
- What are some examples of scenarios?

Alternatively, you may propose a possible solution to the questions or you may invite the audience to discuss and listen to their proposals and answers to our questions. It's here where the discussion takes place.

Your audience often needs the opportunity to voice their opinion and make suggestions. Sometimes they may even have one or more questions.

Consider again the cats and dogs example. To add a call to action, change the title from *9 pets out of 10 are cats* to *Increase the advertising campaign on dog-related websites!* Figure 1.12 shows the resulting chart, which adds the call to action in the title.

Figure 1.12 A stacked bar chart showing the current percentage of pets participating in the event, enriched with a call to action



The pets' scenario demonstrated how to turn data into wisdom. Firstly, we took data that contained the raw number of participants in the event. Then, we extracted information that told us that 1 participant out of 10 is a dog and the remaining 9 are cats. Next, we added a context that explained why dogs were not interested in the event. The motivation was that the advertising campaign for dogs was poor. Finally, we added a call to action that invited the audience to increase the dog advertising campaign.

In this chapter, you have learned the basic concepts behind data storytelling and how to transform data into stories using the DIKW pyramid. In the next chapter, you'll see how to use Python Altair and Generative AI to implement the DIKW pyramid approach.

1.5 Summary

- Data storytelling is a powerful tool that helps you communicate your data more effectively. Data storytelling makes complex data more relatable and easier to understand when used correctly.
- Python Altair and Generative AI are great tools for creating stunning data stories.
- Do complex exploratory data analysis: exploratory data analysis helps data analysts summarize a dataset's main features, identify relationships between variables, and detect outliers. This approach is often used when working with large datasets or datasets with many variables.
- Don't use Python Altair and Generative AI if you want to perform big data analytics or write complex reports that summarize big data.
- Use the DIKW pyramid to turn your data into wisdom.

1.6 References

Berengueres, J. and Sandell, M. (2019) Introduction to Data Visualization & Storytelling: A Guide For The Data Scientist. Independently published.

Cairo, A. (2012). The Functional Art. New Riders.

Compass UOL (2023). Generative AI Speeds Up Software Development: Compass UOL Study. https://blog.compass.uol/noticias/generative-ai-speeds-up-software-development-compass-uol-study/ (Last Access, 2023/05/16)

Duolingo and GitHub Enterprise (2022). Duolingo empowers its engineers to be force multipliers for expertise with GitHub Copilot, Codespaces.https://github.com/customer-stories/duolingo (Last Access, 2023/05/16)

Dykes, B. (2023). LinkedIn post: https://www.linkedin.com/feed/update/urn:li:activity:7061798908565336065/ (Last Access, 2023/05/10)

2 Running your first data story in Altair and GitHub Copilot

This chapter covers

- Introduction to Altair
- Prerequisites
- Use Case: Describing the Scenario
- First Approach: Using Altair
- Second Approach: Using Copilot

In this chapter, we will look at the basic concepts behind Altair, and then we will implement a practical use case, which will allow us to transform a raw dataset into a story. We will progressively apply the Data, Information, Knowledge, Wisdom (DIKW) pyramid principles in Altair and see the results achieved step by step. In the second part of the chapter, we will use Copilot to automate some steps of the story creation process. We will focus only on Copilot as a Generative-AI tool to keep the chapter simple and the flow understandable. In the next chapters, we will introduce ChatGPT and DALL-E to the DIKW pyramid.

2.1 Introduction to Altair

The Vega-Altair library (Altair, for short) is a declarative Python library for statistical visualization based on the Vega and Vega-Lite visualization grammars. Vega is a visualization grammar for creating expressive and interactive data visualizations. Vega-Lite is a high-level declarative language built on top of Vega, designed to simplify the creation of common data visualizations with concise and intuitive syntax. We will discuss Vega and Vega-lite in the next chapter.

Altair is a declarative library because you can describe the intended chart outcome rather than manually programming every step, defining a concise and intuitive syntax for creating interactive visualizations. We use declarative libraries to specify what we want to see in a chart. We can specify the data and the type of visualization we want, and the library creates the visualization for us automatically. Declarative libraries stand in contrast to imperative libraries, which instead focus on building a visualization manually, for example specifying the desired axis, size, legend, and labels. Matplotlib is an example of an imperative library.

Every Altair chart has at least three main elements: **Chart**, **mark**, and **encodings**. We will cover each of these briefly in the following pages. Refer to Appendix A to install Altair and the other Python libraries described in this chapter. Refer to Appendix B for more details on the Pandas DataFrame methods used in this chapter.

2.1.1 Chart

A Chart in Altair is an object acting as a starting point for constructing and customizing interactive visualizations in Altair.. Use it to input the dataset you would like to represent visually:

Listing 2.1 Chart in Altair

```
import altair as alt
import pandas as pd

# Create data for the DataFrame
data = {'Name': ['Alice', 'Bob', 'Charlie'],
'Age': [25, 30, 35],
'City': ['New York', 'Paris', 'London']
}

# Create the DataFrame from the data
df = pd.DataFrame(data)

# Print the dataframe
print(df)

# Create the chart
chart = alt.Chart(df)
```

Chart

First, import the required libraries, then build the DataFrame containing your data. Finally, pass the DataFrame to, `alt.Chart()`. The example does not create any visualization yet because we have not specified what we want to represent.

2.1.2 Mark

A mark defines how to represent data. Examples of marks include bar charts, line charts, area charts, and many more. To specify a mark, append it to the Altair Chart. For example, in the listing below, we will specify that we want to use a bar chart.

Listing 2.2 Mark in Altair

```
chart = alt.Chart(df).mark_bar()
```

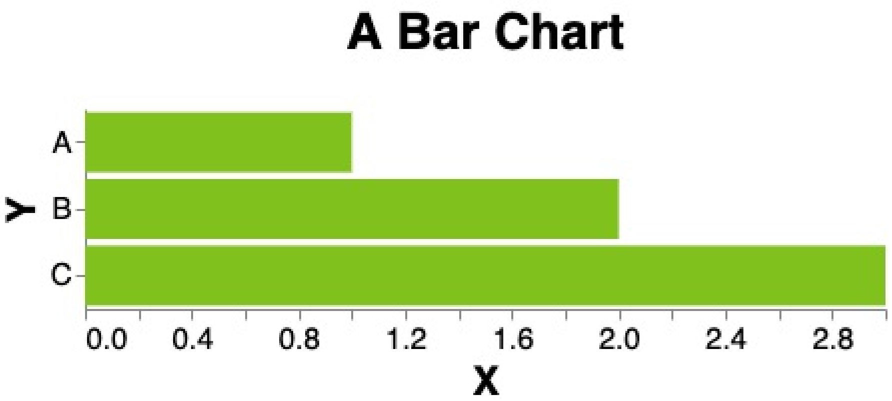
Mark

Use `mark_bar()` to create a bar chartin Altair. If we run the code as it is, it raises the following remarks: 1. from config import * 2. configure_layout(chart) is required to display the diagram 3. This is because we have not defined which dataset columns we want to represent.

Each mark is `mark_<type_of_graph>()`. Examples of marks are:

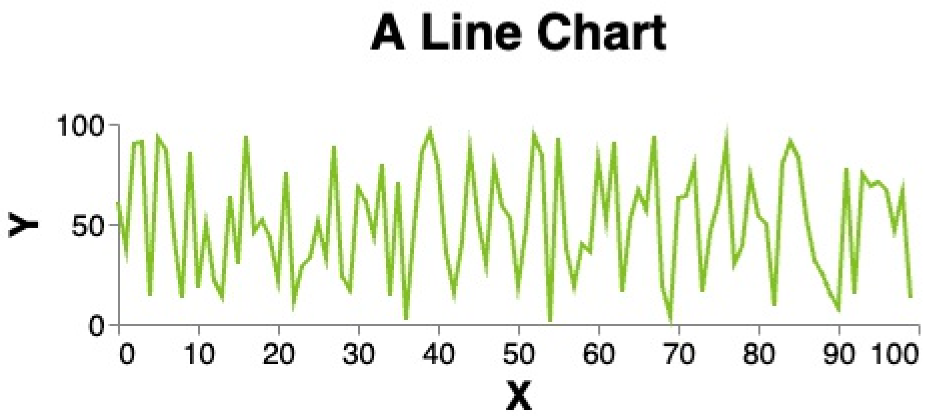
- `mark_bar()` to plot a bar chart (Figure 2.1)

Figure 2.1 An example of a bar chart



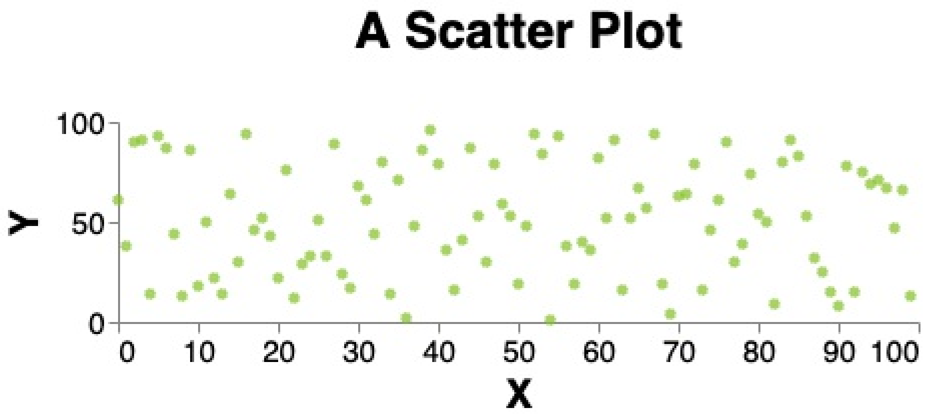
- `mark_line()` to plot a line chart (Figure 2.2)

Figure 2.2 An example of a line chart



- `mark_circle()` to plot a scatter plot (Figure 2.3).

Figure 2.3 An example of a scatter plot



2.1.3 Encodings

Encodings specify the mapping between the DataFrame columns and their visual representation in the Altair chart. In practice, encodings define where to represent data in the chart, such as their position, size, and color. We call *channel* every visual representation of data in the chart. Examples of channels are axes, color, size, and so on. For example, in the listing below, we use encodings to specify which columns of the DataFrame we must use in the x and y axes:

Listing 2.3 Encodings in Altair

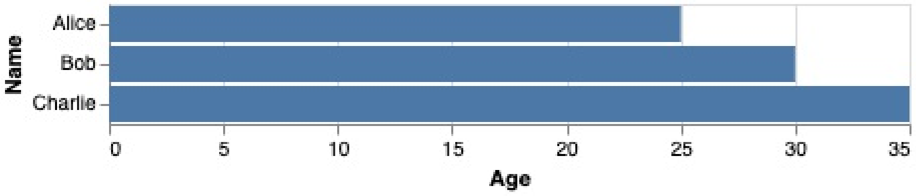
```
import pandas as pd
import altair as alt

# Create data for the DataFrame
data = {'Name': ['Alice', 'Bob', 'Charlie'],
        'Age': [25, 30, 35],
        'City': ['New York', 'Paris', 'London']
}

# Create the DataFrame from the data
df = pd.DataFrame(data)

# Create the chart
chart = alt.Chart(df).mark_bar(
).encode(
    x = 'Age:Q',
    y = 'Name:N'
)

chart.save('chart.html')
```

Encodings

Use `encode()` to define encodings in Altair. Place the `Age` variable of `df` on the x-axis and the `Name` variable of `df` on the y-axis. Interpret `Age` as a quantity (`Q`), and `Name` as a nominal value (`N`). Use `chart.save('chart.html')` to export the chart as an HTML file. If you use a Jupyter notebook, use `chart.display()` to plot the chart inline.

Figure 2.4 shows the chart produced by the code contained in Listing 2.3.

Figure 2.4 The chart produced by Listing 2.3



You can practice with basic charts in Altair through the examples provided in the GitHub repository of the book. Start by cloning the GitHub repository of the book, available at the following link: https://github.com/alod83/Data-Storytelling-with-Python-Altair-and-Generative-AI[1]. To get started with GitHub, follow the procedure described in Appendix A of the book or the GitHub official documentation. Find the examples under `02/Basic Chart.ipynb` of the GitHub repository.

Now that you have learned the basic concepts behind Altair, we will review the prerequisites for implementing a practical scenario in Altair and Copilot.

2.2 Use Case: Describing the Scenario

Let us suppose we are at the end of 2019, before the Covid-19 pandemic. You work as a data analyst at a Portuguese hotel. Your boss has experienced increased tourist arrivals in their hotel and wants to invest money to build a new swimming pool. Before investing money, they ask you to study the phenomenon to understand whether their investment will be successful. The objective of your task is to see whether Portugal is seeing an increased number of tourist arrivals in recent years relative to global trends.

You start your searches and find a dataset about arrivals at tourist accommodation establishments[2], released as open data by Eurostat. For simplicity, in this chapter, you consider only one dataset, but in a real use-case scenario, you should consider more data, including the popularity of swimming pools among tourists visiting Portugal, the cost of building a new swimming pool at the hotel, and the potential revenue that the new swimming pool could generate for the hotel.

The dataset contains the number of arrivals at tourist accommodation establishments since 2019 in all European countries. For simplicity, focus only on the following countries: Portugal, France, Italy, Germany, Spain, and the United Kingdom.

Find the code described in this chapter in folder `02` of the GitHub repository and the dataset in folder `02/source` of the same repository.

2.2.1 The Dataset

Table 2.1 shows a sample of the dataset we will analyze. The dataset contains the tourist arrivals at establishments from 1994 to 2019 for Italy (IT), France (FR), Germany (DE), Portugal (PT), Spain (ES), and the United Kingdom (UK).

Table 2.1 Arrivals at tourist arrivals establishments since 1994

Date	IT	FR	DE	PT	ES	UK
1990-01-01	2.543.920		3.185.877	325.138	1723786	1.776.000
1990-02-01	2.871.632		3.588.879	381.539	1885718	2.250.000
1990-03-01	3.774.702		4.272.437	493.957	2337847	2.662.000
…	…	…	…	…	…	…
2019-08-01	11.649.500	13.692.822	14.570.339	2.531.809	12893366	8.889.049
2019-09-01	9.888.817	11.684.845	14.373.815	2.263.748		5.858.984
2019-10-01	7.692.388	10.401.793	13.780.441	1.995.942		7.455.781

The table contains some missing values for some countries. Load the dataset as a Pandas DataFrame:

Listing 2.4 Loading the dataset as a Pandas DataFrame

```
import pandas as pd

df = pd.read_csv('source/tourist_arrivals_countries.csv', parse_d
```

Dataset loading

Use Pandas to load the dataset as a DataFrame. Use the `parse_dates` parameter to load the `Date` field as a date.

Now, you are ready to perform basic data exploration.

2.2.2 Data Exploration

Exploratory Data Analysis (EDA) aims at analyzing and visualizing data to extract insights and understanding of its underlying patterns, distributions, and relationships. You may even use Generative AI to perform EDA. Performing a complete data exploration is out of the scope of this book. However, for quick data exploration, you can use an existing library, such as ydata profiling[3] or Sweetviz[4]. In this chapter, we'll use ydata-profiling. Import the library, create a `ProfileReport()` object, and pass the Pandas DataFrame as an input argument. Finally, build the report by invoking the `to_file()` method:

Listing 2.5 Building a summary report

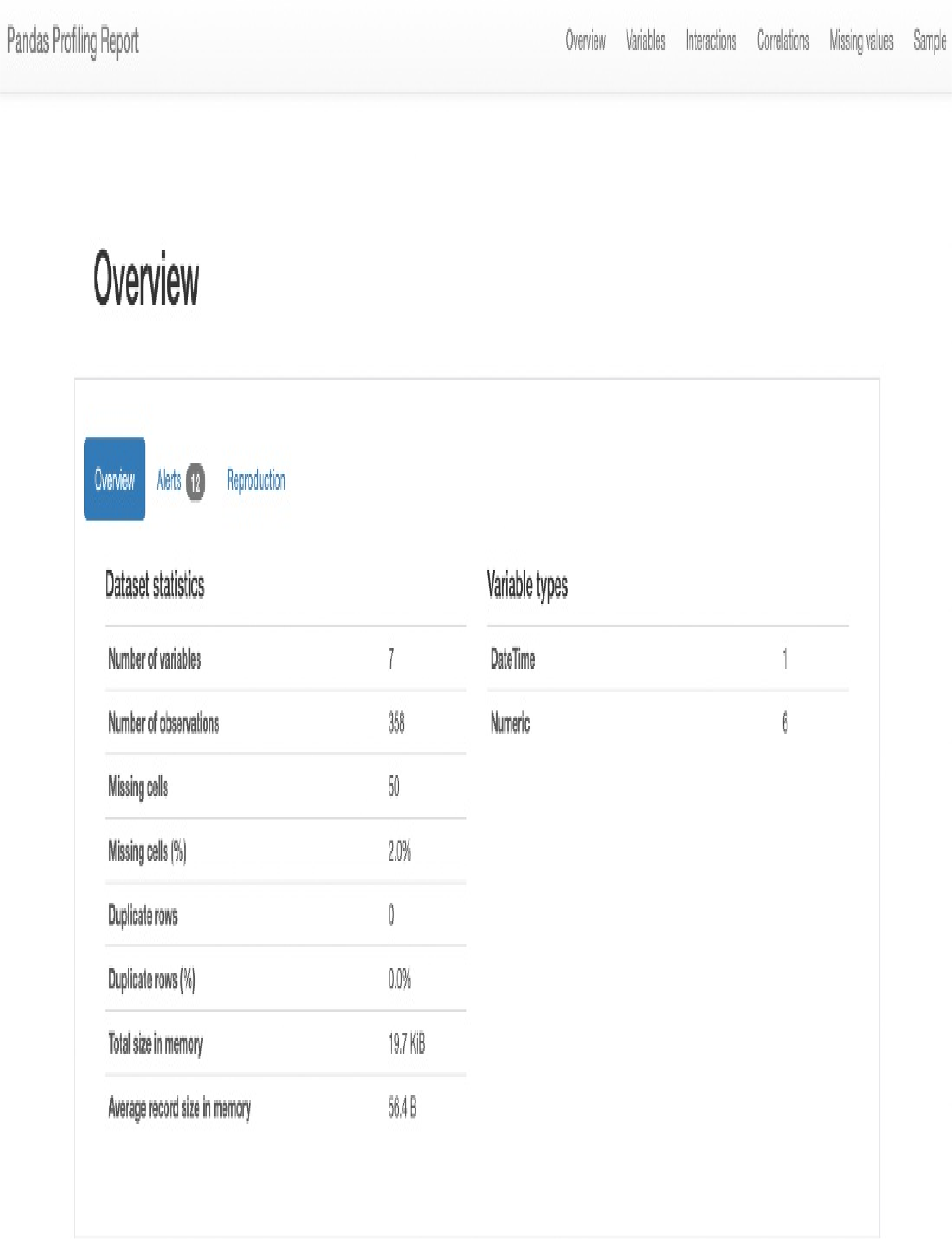
```
from ydata_profiling import ProfileReport

eda = ProfileReport(df)
eda.to_file(output_file='eda.html')
```

Dataset exploration

Use pandas-profiling to build a summary report of the dataset.

As a result, the `to_file()` method produces an HTML file that contains the report. The following figure shows a snapshot of the produced report.

Figure 2.5 A snapshot of the report produced by pandas-profiling



Overview

Dataset statistics

Number of variables	7
Number of observations	358
Missing cells	50
Missing cells (%)	2.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	19.7 KiB
Average record size in memory	56.4 B

Variable types

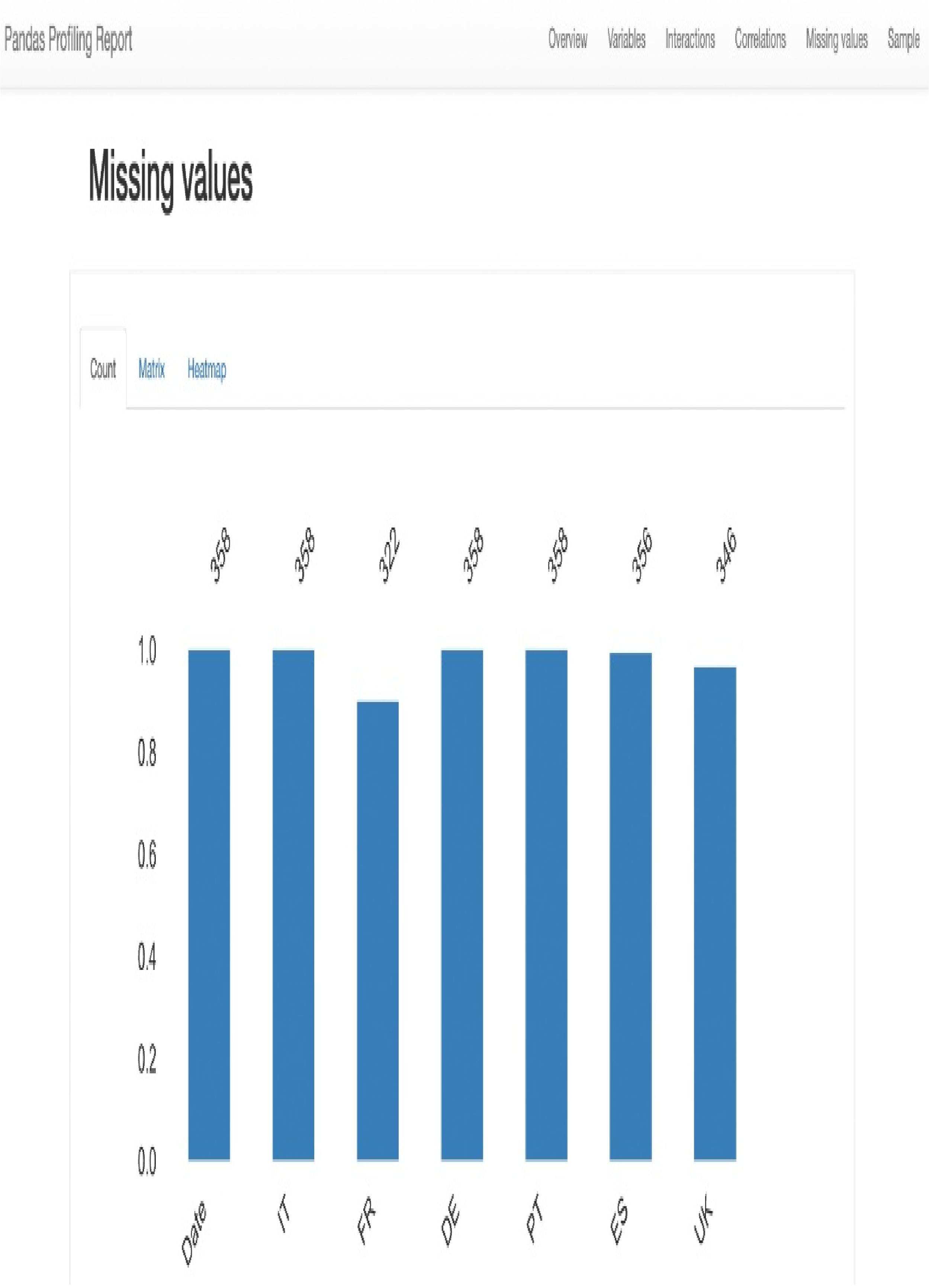
DateTime	1
Numeric	6

The report contains many sections, as shown in the top right menu of Figure 2.5. The report may help you to

- Understand the data: the report provides an overview of the dataset, including the number of observations, variables, and missing values. Use this information to understand the data quality.
- Identify data types: the report identifies the data types of each variable and provides a summary of their distribution, including measures such as mean, median, standard deviation, and range. Use this information to identify variables that may require further preprocessing.
- Identify correlations: the report provides a correlation matrix, heatmap, and scatter plot matrix, which can help identify variables that are highly correlated or related.
- Identify distributions: The report provides distribution plots, including histograms and density plots, which can help identify the shape and spread of the distribution of each variable. Use this information to understand the underlying patterns and trends in the data.

In our case, the missing value section describes the presence of missing values for some countries (France, Spain, and the United Kingdom), as shown in the following figure:

Figure 2.6 The report highlights missing values for France, Spain, and the United Kingdom


Pandas Profiling Report
Overview
Variables
Interactions
Correlations
Missing values
Sample
Missing values
Count
Matrix
Heatmap
358
358
322
358
358
356
346
1.0
0.8
0.6
0.4
0.2
0.0
Date
IT
FR
DE
PT
ES
UK

Now that we have quickly explored the dataset, you are ready to build a data story from that dataset. Let’s start with the first approach: using Altair.

2.3 First Approach: Altair

To transform data into a story, we will proceed step by step. First, we’ll build a basic chart and then gradually enrich it with the elements necessary to transform the chart into a story in its own right. We’ll use the principles of the DIKW pyramid to implement this progressive enrichment:

- From data to information
- From information to knowledge
- From knowledge to wisdom

Let’s start with the first step of the DIKW pyramid, turning data into information.

2.3.1 From Data to Information

Turning data into information means extracting some meaning from data. Start by drawing the raw chart, shown in the following code:

Listing 2.6 Building the raw chart in Altair

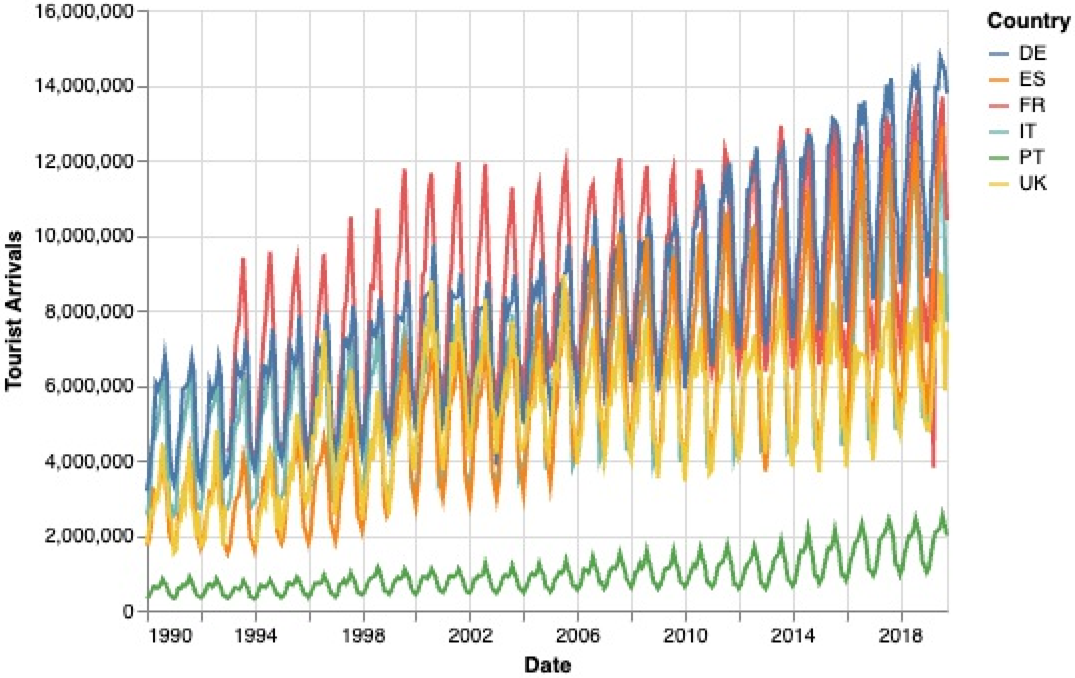
```
import altair as alt

df2 = pd.melt(df, id_vars='Date', value_name='Tourist Arrivals',

chart = alt.Chart(df2).mark_line().encode(
    x = 'Date:T', #A
    y = 'Tourist Arrivals:Q', #B
    color=alt.Color('Country:N') #C
)

chart.save(‘chart.html)
```

The raw chart in Altair

Use the `mark_line()` property to build the raw chart in Altair. Draw the number of tourist arrivals on the y-axis and the date on the x-axis.

Use the `melt()` function to unpivot the dataset, i.e., transform data rows into columns. Then, build the chart. The chart uses three channels: `x` and `y`, to describe the axes, and `color`, to group by countries. For each channel, specify the column in the dataset (for example, `Date` for the `x` channel) and the data type (`T` for temporal data, `Q` for quantitative data, and `N` for nominal data). The following figure shows the produced chart:

Figure 2.7 The raw chart produced in Altair without any manipulation



The chart is difficult to read because all countries overlap. However, we can extract the following highlight: all the trend lines have increased since 1990. This is a positive message to communicate to our boss. From 1990 until 2019, there has been a progressive increase in the arrival of tourists. Let's focus more on this information.

In all cases, we are not interested in knowing the intermediate data, as there are no outliers, only the starting point (1990) and the final point of the time series (2019). However, we note that for some countries, there is no data for

1990, and 2019 is incomplete (the data goes up to October). In this case, the question is: what to do in the presence of missing values? The answer is: it depends. In our case, we are dealing with a time series that grows progressively (seasonality excluded), so we can restrict the analysis range only to the dates for which we have data. Thus, we narrow our search to the range 1994-2018.

Let’s implement this strategy in Altair.

Start by removing missing values from the dataset:

Listing 2.7 Removing missing values from the dataset

```
mask_fr = df['FR'].isnull() == False
mask_uk = df['UK'].isnull() == False
mask_es = df['ES'].isnull() == False
df = df[(mask_fr) & (mask_uk) & (mask_es)]
```

Removing missing values

Use the `isnull()` method to build a mask that selects missing values. Then use the mask to filter missing values in the original dataset.

Then, group data by year and calculate the average value. Also, select only data for 1994 (the first in the dataset) and 2018 (the penultimate in the dataset).

Listing 2.8 Grouping by year and calculating the average value

```
df2 = df.copy()
df2.loc[:, 'Year'] = df['Date'].dt.year

df2 = df.groupby(by='Year').mean(numeric_only=True)
df2 = df2.iloc[[0, -2]]
df2.reset_index(inplace=True)
```

Extracting the first and the last year from the dataset

First, extract the year from the Date field, then group data by year and calculate the average value. Next, select only the first year (1994) and the penultimate year (2018). Finally, reset the index to retrieve the year.

Use `loc` to access a group of rows and columns by labels and `iloc` to access group rows and columns by numerical indices. Use the `reset_index()` function to create a new index column starting from 0 and move the current index values to a new column.

Finally, build the chart in Altair:

Listing 2.9 Building the chart in Altair

```
df3 = pd.melt(df2, id_vars='Year',value_name='Tourist Arrivals',

chart = alt.Chart(df3).mark_line(point=alt.OverlayMarkDef()).enco
    x = alt.X('Year:O', title='', axis=alt.Axis(labelAngle=0)), #
    y = 'Tourist Arrivals:Q',
    color=alt.Color('Country:N')
).properties(
    width=400,
    height=300,
    title='Countries Trend in Arrivals'
)

chart.save(‘chart.html’)
```

Building a line chart in Altair

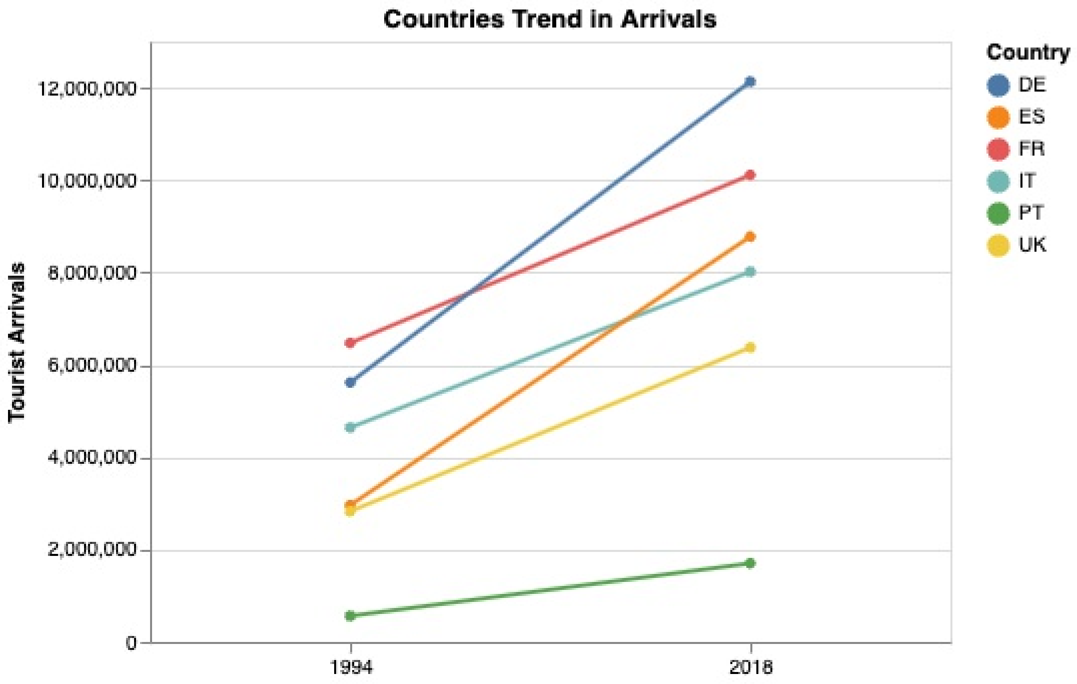
Use the `mark_line()` property to build a line a chart in Altair. Pass `point=alt.OverlayMarkDef()`to highlight points for each line. Configure the label angle for the `x` channelwith `axis=alt.Axis(labelAngle=0)`. Finally, set the chart width, height, and title through the `width`, `height` and `title` properties of the `properties()` function..

To encode the x channel, we have used the `alt.X()` object and not a simple string as we did for the other channels. When we want to specify the channel details, we can use the channel object instead of a simple string. For example, we can use `alt.Y()` to set the properties of the y channel, `alt.Color()` for the color channel, and so on. Within a channel object, you can set different properties that depend on the channel. For example, for the x and y channels, you can set the axis property. For each property within a channel, you can define a value or an additional object. In practice, the same strategy of channels applies to the channel properties. For example, `axis` is a property of the `x` and `y` channels. We can use `alt.Axis()` to set additional properties of the axis. In the example, we have set the `labelAngle` property of the axis.

To build the chart, we have also used the `properties()` function, which configures the chart's general properties, such as the width, the height, and the title.

The following figure shows the produced chart:

Figure 2.8 The produced chart after calculating the average value for each year and removing intermediate years



The chart is clearer than that in Figure 2.7: you can easily distinguish the trendline for each country. However, the presence of too many colors and countries does not transmit the message. Our objective is to focus on Portugal, which is the country about which our boss has asked for information. In addition, we calculate the average number of tourist arrivals to the other countriesfor comparison. The following listing describes how to prepare the DataFrame for the chart:

Listing 2.10 Preparing the DataFrame

```
mask = df3['Country'] == 'PT' #A
df4 = df3.loc[~mask].groupby('Year').mean().reset_index() #B
df4['Country'] = 'Others (mean)' #C
df5 = df3.loc[mask].append(df4, ignore_index=True) #D
```

Prepare the DataFrame

Build a new DataFrame with two countries: Portugal and the average value for the other countries.

Table 2.2 shows the resulting DataFrame.

Table 2.2 Arrivals at tourist arrivals establishments since 1994

Year	Country	Tourist Arrivals
1994	PT	563,354.83
2018	PT	1703,339.0
1994	Others (mean)	4,503,850.8
2018	Others (mean)	9,082,776.1

Now we can draw the chart, as shown in the following listing:

Listing 2.11 Drawing the chart with Portugal and Average

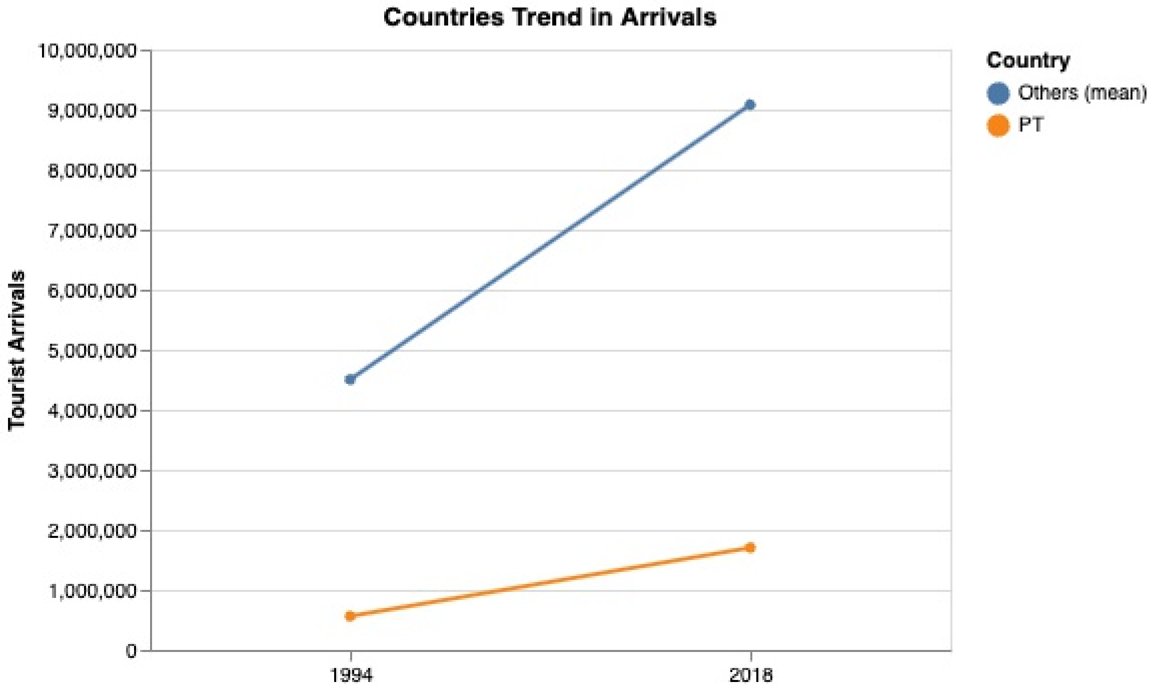
```
chart = alt.Chart(df5).mark_line(point=alt.OverlayMarkDef()).enco
    x = alt.X('Year:O', title='', axis=alt.Axis(labelAngle=0)),
    y = 'Tourist Arrivals:Q',
    color=alt.Color('Country:N')
).properties(
    width=400,
    height=300,
    title='Countries Trend in Arrivals'
)
chart.save('chart.html')
```

Draw the chart

Use `df5` to draw the new chart.

The following figure shows the resulting chart after selecting only Portugal and the average value for the other countries.

Figure 2.9 The produced chart after selecting only Portugal and Germany



The increase in the number of tourists over time in the other countriesis greater than that of Portugal. However, we are not interested in knowing the absolute values, but rather the percentage increase of each country over time. If we use absolute values, instead, we can’t answer the following questions immediately:

- For each country, what is the percentage increase since 1994?
- Which of the two nations experiences the greater percentage increase?

If we use the percentage increases, we can put the two countries on a more comparable level and we can answer the previous questions. Let’s calculate the percentage increase since 1994 for each country and then plot the chart of percentage increases instead of that of absolute values.

Calculate the percentage increase using the following formula:

Percentage Increase = (Final Value - Starting Value) / Starting Value * 100

In our case, calculate the percentage increase for each country as follows:

Listing 2.12 Calculating the percentage increase for each country

```
mask_first = df5['Year'] == 1994
mask_last = df5['Year'] == 2018
mask_pt = df5['Country'] == 'PT'
mask_ot = df5['Country'] == 'Others (mean)'

pi_pt = (df5[mask_last & mask_pt]['Tourist Arrivals'].values[0] -
        df5[mask_first & mask_pt]['Tourist Arrivals'].values[0])
        df5[mask_first & mask_pt]['Tourist Arrivals'].values[0]*1

pi_ot = (df5[mask_last & mask_ot]['Tourist Arrivals'].values[0] -
        df5[mask_first & mask_ot]['Tourist Arrivals'].values[0])
        df5[mask_first & mask_ot]['Tourist Arrivals'].values[0]*1
```

Percentage increase

First, build two masks, one for each year and for each country. Then calculate the percentage increase for each country. Use `df[field].values[0]` to get the value of the first row of a field.

Now, build again the chart, as shown below:

Listing 2.13 Drawing the chart of percentage increases

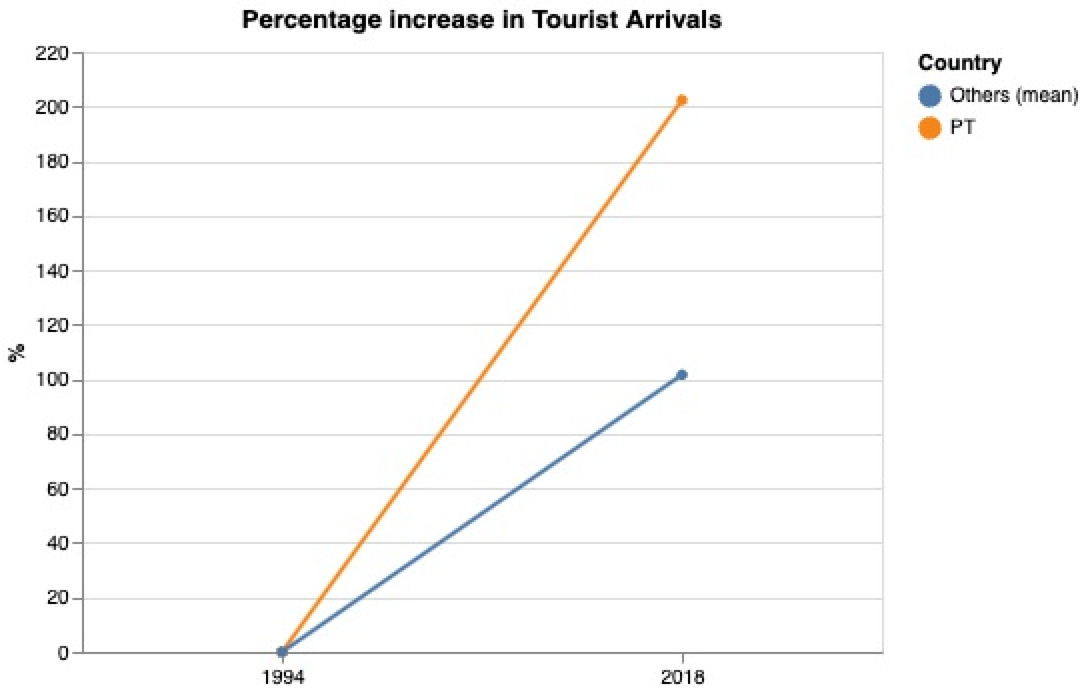
```
df_pi = pd.DataFrame({
    'Date'    : [1994,2018,1994,2018],
    'Country' : ['PT', 'PT', 'Others (mean)', 'Others (mean)'],
    'Tourist Arrivals' : [0, pi_pt, 0, pi_ot]
})

chart = alt.Chart(df_pi).mark_line(point=alt.OverlayMarkDef()).en
    x = alt.X('Date:O', title='', axis=alt.Axis(labelAngle=0)),
    y = alt.Y('Tourist Arrivals:Q',title='%'),
    color=alt.Color('Country:N')
).properties(
    width=400,
    height=300,
    title='Percentage increase in Tourist Arrivals'
)

chart.save('chart.html')
```

Drawing the chart of percentage increase

First, create a new DataFrame with the required values. Then, draw the chart. Use `title` to modify the title of the y axis..

The following figure shows the resulting chart:

Figure 2.10 The produced chart after calculating the percentage increase for each country



The chart shows that Portugal has experienced a higher percentage increase in tourist arrivals than the other countries. This is an extraordinary discovery, that answers our boss' question. We have extracted information from the data. To highlight our discovery, add an annotation to the chart, which describes exactly the percentage increase value in correspondence of 2018.

Listing 2.14 Adding an annotation to the chart

```
df_ann = pd.DataFrame({
    'Text' : ['Portugal: ' + '{:.2f}'.format(pi_pt) + '%',
              'Others (mean): ' + '{:.2f}'.format(pi_ot) + '%'],
    'Y' : [pi_pt,pi_ot],
    'X' : [2018,2018],
    'Country' : ['PT', 'Others (mean)']
})

pi = alt.Chart(df_ann).mark_text(
    dx=10, #A
    align='left', #B
    fontSize=20 #C
).encode(
    text='Text:N',
    y='Y:Q',
    x='X:O',
    color=alt.Color('Country:N',
                    scale=alt.Scale(domain=['Others (mean)', 'PT'
                    range=['lightgrey', 'orange']),
                    legend=None)
)
total = (chart + pi).configure_view(strokeWidth=0)
total.save('chart.html')
```

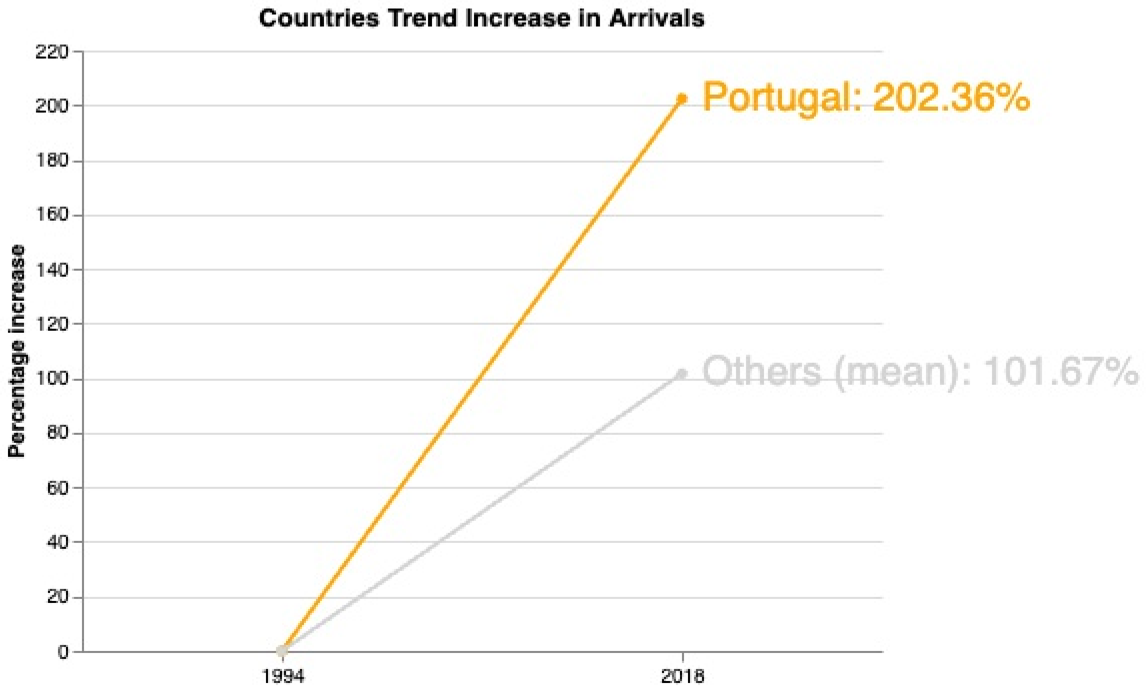
Annotating the chart

First, build a DataFrame with the text to add to the chart and the position where to place the text. Then, use the `mark_text()` property to draw the text. Finally, combine the previous chart and the text annotation through the + operator. Use `configure_view(strokeWidth=0)` to remove the borders of the annotation chart. Also, note that we have defined a scale in the color channel. The scale defines the range (colors to use) and the domain (the labels to which apply colors)..

The `mark_text()` function (and the other `mark_*()` functions) can receive as input some parameters that define some static properties of the chart. For example, if you want to use a static color for all the texts in the chart, set the color in the `mark_*()` function. Instead, if the color of the text depends on a column in the DataFrame, set it in the `encode()` function.

The following figure shows the resulting chart at the end of the first step, turning data into information:

Figure 2.11 The produced chart at the end of the first step: turning data into information



The chart is clear and easy to understand. Adding information means extracting only the relevant data from data and representing them in a way that is easy to understand by the audience without further calculation. Now that you have turned data into information, it’s time to move to the next step of the DIKW pyramid, turning information into knowledge.

2.3.2 From Information to Knowledge

Turning information into knowledge means adding context to the extracted information. Usually, context answers the following question: why does this situation happen? If possible, extend your search or look at the same dataset to answer this question. In our case, the question is: Why has Portugal experienced such an incredible increase in tourist arrivals over 25 years? To answer this question, you searched on the Web and discovered that Portugal (and other countries) introduced many low-cost flights in the early 1990s (Dobruszkes 2013). This has allowed for more massive development of tourism.

This finding is the context for our graph: *Thanks to the introduction of low-cost flights, Portugal has experienced an increase in tourist arrivals of over 200% in 25 years, even surpassing the increase in the other countries*. Add this discovery as an annotation in the chart:

Listing 2.15 Adding the context to the chart

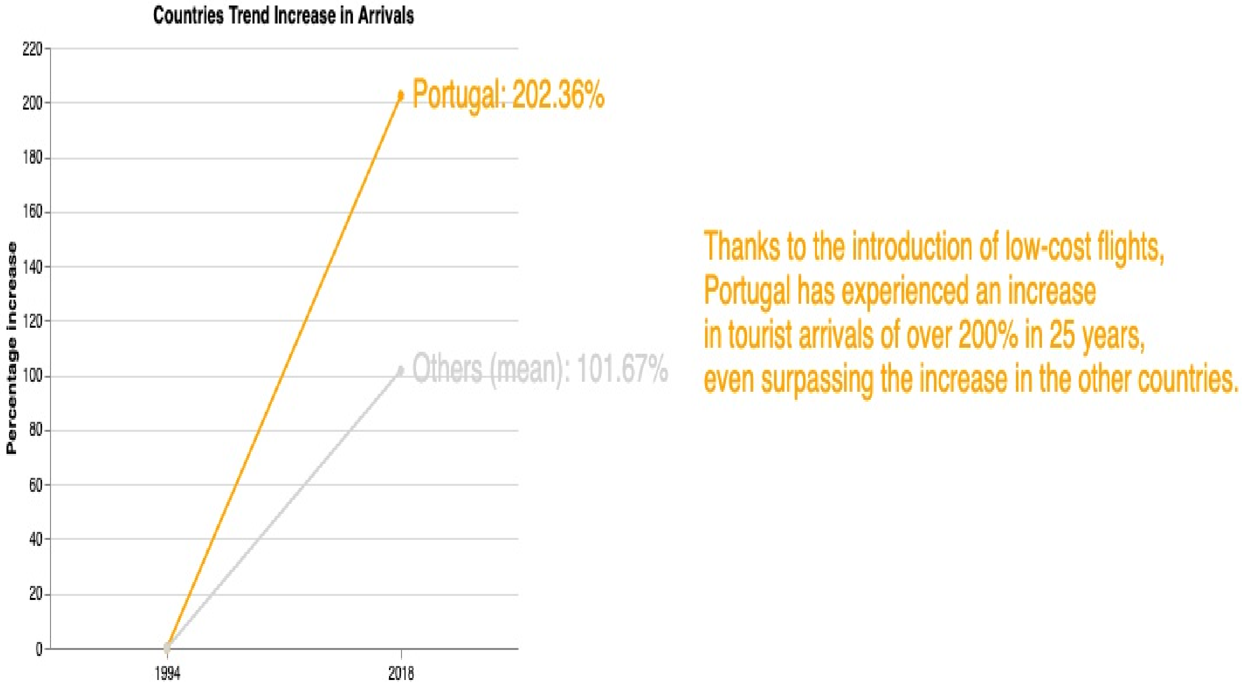
```
annotation = f"""Thanks to the introduction ..."""
df_text = pd.DataFrame([{'text' : annotation}])

text = alt.Chart(df_text
).mark_text(
    lineBreak='\n',
    align='left',
    fontSize=20,
    y=100,
    color='orange'
).encode(
    text='text:N'
)

# build the final chart
total = (chart + pi | text).configure_view(strokeWidth=0)
total.save('chart.html')
```

Adding context to the chart

First, build a DataFrame with the text to add to the chart. Then, use `mark_text()` to draw the text. Also, specify the line break symbol (`lineBreak='\n'`), how to align the text (`align='left'`), the font size (`fontSize=20`), the position (`y=100`), and the color (`color='orange'`). Finally, combine the text with the previous chart through the + operator.

The following figure shows the resulting chart at the end of the second step, turning information into knowledge:

Figure 2.12 The produced chart at the end of the second step: turning information into knowledge



Thanks to context, the chart answers why this happens. Now that you have turned information into knowledge, it’s time to move to the final step of the DIKW pyramid, turning knowledge into wisdom.

2.3.3 From Knowledge to Wisdom

Turning knowledge into wisdom means adding a call to action to the chart and proposing the next steps. In this chapter, we focus only on adding a call to action. We’ll see how to add the next steps in the following chapters. The call to action invites the audience to do something. In our case, the call to action should answer our boss’s original question: can I build a new swimming pool?

Given the results of the previous analysis, formulate the following call to action: *Yes, you can build a new swimming pool!* Add the call to action as the chart's title:

Listing 2.16 Adding the call to action to the chart

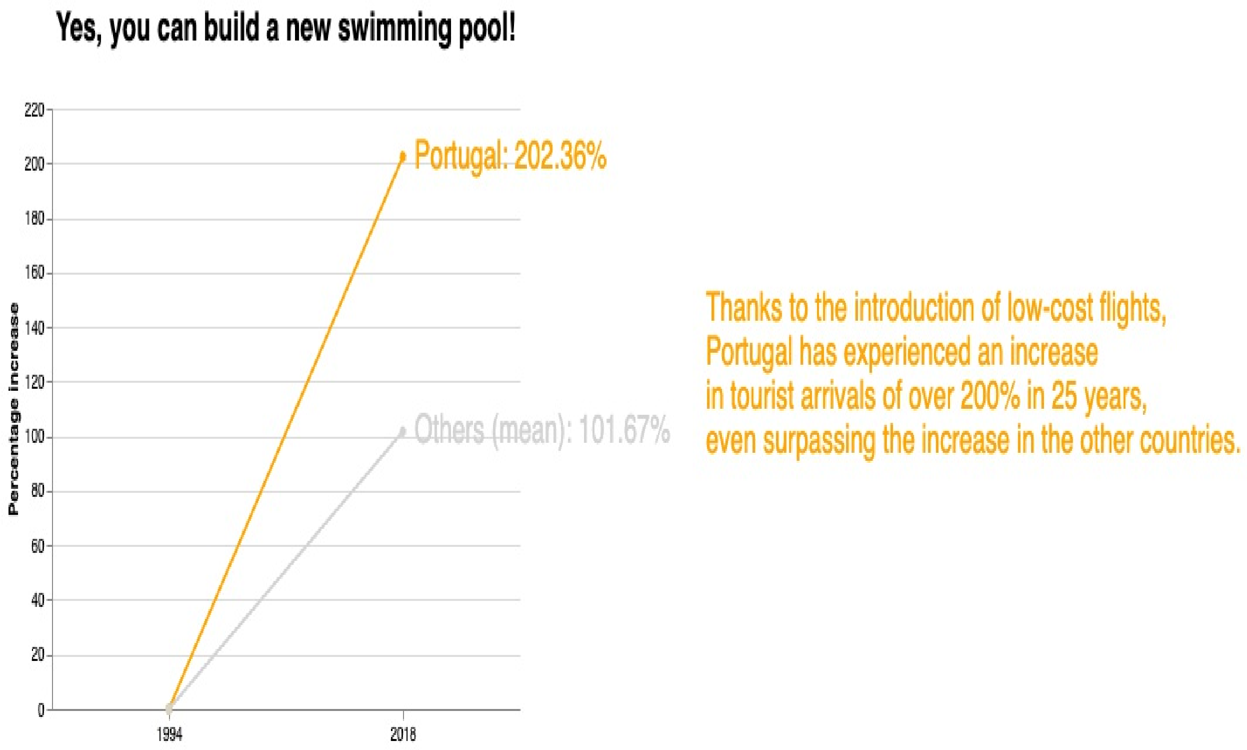
```
chart = alt.Chart(df_pi).mark_line(point=alt.OverlayMarkDef()).en
    x = alt.X('Date:O', title='', axis=alt.Axis(labelAngle=0)),
    y = alt.Y('Tourist Arrivals:Q',title='Percentage increase'),
    color=alt.Color('Country:N')
).properties(
    width=400,
    height=300,
    title='Yes, you can build a new swimming pool!'
)

total = (chart + pi | text
).configure_view(
    strokeWidth=0
).configure_title(
    fontSize=20,
    offset=25
)
total.save('chart.html')
```

Adding call to action to the chart

Set the font size of the title of the chart through `fontSize=24` and the distance between the title and the chart (`offset=25`).

The following figure shows the resulting chart at the end of the last step, turning knowledge into wisdom:

Figure 2.13 The produced chart at the end of the final step: turning knowledge into wisdom



The chart contains the answer to your boss’s question. Comparing this chart with Figure 2.5, you probably notice a big difference. The charts in Figures 2.9 and 2.10 show that the audience doesn't have to do any calculations or processing. Just read the result. If, on the other hand, they look at the chart in Figure 2.5, the audience has to extract the information by themselves.

Someone may argue that a lot of information was lost in the process, such as what happened to the other European countries and what happened in the years between 1994 and 2018. Thus the boss wouldn't want to make a big investment based only on the numbers shown in the chart. This observation is correct. However, we have simplified the example to show the workflow to turn data into a story. Obviously, in a more complex scenario, you must consider more factors and build an appropriate story, where each factor is a single scene of your data story.

To have an idea of the differences between Altair and Matplotlib, in the next section, we build the same chart using Matplotlib. If you are not familiar with Matplotlib, jumpy directly to the next section, Second approach: Copilot.

2.3.4 Comparing Altair and Matplotlib

Matplotlib is a popular Python library for data visualization. Probably, it is the first Python library you used to plot your data quickly. If you use Matplotlib as an alternative to Altair, consider that Matplotlib is an imperative library, so you must define all the steps to produce the chart. In Altair, instead, you focus only on the desired output. Anyway, all the concepts related to data storytelling described in this book are also valid for Matplotlib, so use it if you prefer the imperative approach.

The following table compares the Altair code used to generate the chart in Figure 2.13 with the equivalent code in Matplotlib. You can find the complete code in Matplotlib in the GitHub repository of the book, under `02/Matplotlib Example.ipynb`.

Table 2.3 A comparison between Altair and Matplotlib

Step	Altair Code
Import libraries	`import altair as alt`

Draw the main	`chart = alt.Chart(df_pi).mark_line(`
	`point=alt.OverlayMarkDef()`
	`).encode(`
	`x = alt.X(`
	`'Date:O',`
	`title='',`
	`axis=alt.Axis(labelAngle=0)`
	`),`
	`y = alt.Y(`

chart	'Tourist Arrivals:Q',
	title='Percentage increase'),
	color=alt.Color('Country:N')
	).properties(
	width=400,
	height=300,
	title='Yes, you can build a new swimming pool!'
	)

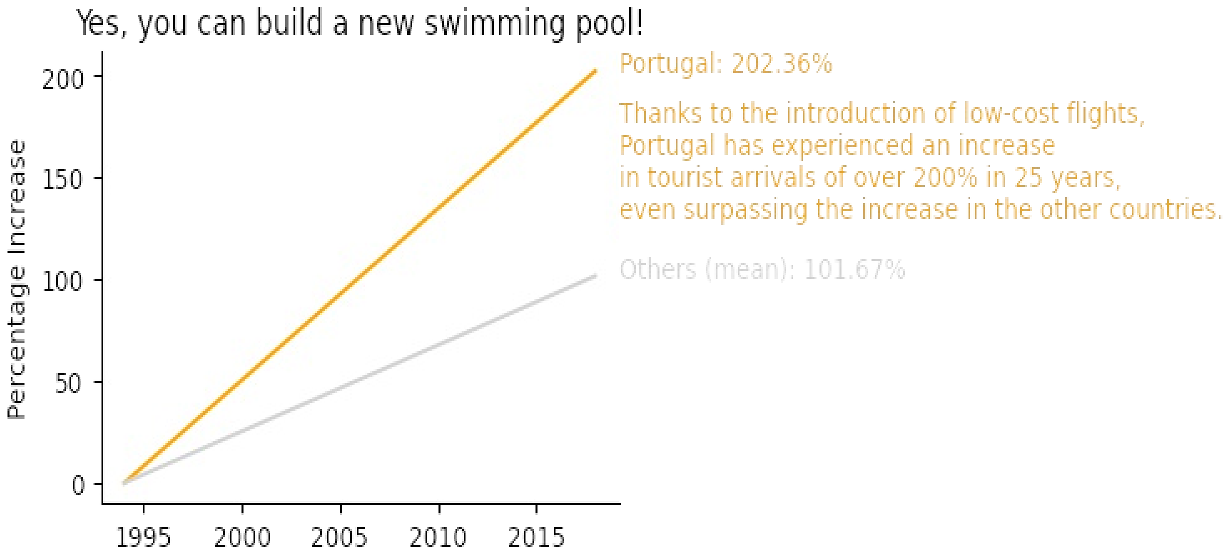
Draw the line annotations	pi = alt.Chart(df_ann).mark_text(
	dx=10,
	align='left',
	fontSize=20
	).encode(
	text='Text:N',
	y='Y:Q',
	x='X:O',
	color=
	alt.Color('Country:N',
	scale=alt.Scale(
	domain=['Others (mean)', 'PT'], range=['lightgrey',
	)

Draw the textual annotation	text = alt.Chart(df_text
	).mark_text(
	lineBreak='\n',
	align='left',
	fontSize=20,
	y=100,
	color='orange'
	).encode(
	text='text:N'
	)

Final Configurations	total = (chart + pi | text
	).configure_view(strokeWidth=0)
	total.save('chart.html')

Figure 2.14 shows the produced chart in Matplotlib.

Figure 2.14 The produced chart in Matplotlib



The Altair code is more verbose than that in Matplotlib. However, in Altair, you specify only the desired output, while in Matplotlib, you must also specify some configurations not strictly related to the output layout, such as `blended_transform_factory()`, to include annotations. In addition, in Matplotlib, you must draw each line or text separately, while in Altair, you pass the DataFrame, and the library draws a line or a mark in general for you.

Now that you have turned your data into wisdom by hand and seen the differences between Altair and Matplotlib, we will move to the alternative method: using GitHub Copilot to build the chart semi-automatically.

2.4 Second Approach: Copilot

Copilot enables you to write code to implement something automatically. We’ll enable Copilot as an extension of Visual Studio Code (VSC). To configure Copilot, please refer to Appendix A.

To use Copilot, it is sufficient to describe the sequence of operations to be implemented orderly, and Copilot will do the rest for us, even proposing different implementations options. In this section, we will use Copilot to build the framework of the graph, i.e., its main parts. Then, we will add the details by hand, using colors and font size. The main difficulty in using Copilot is describing what the code needs to implement in natural language.

The clearer the text, the more consistent Copilot will be in implementation.

In this section, we’ll describe how to translate the code written in the previous sections into a sequence of texts understandable by Copilot. We will focus only on translating the framework of the graph into the natural language without considering the details, such as the font size and the colors used. Find the code generated by GitHub Copilot in the book's repository under `02/GitHubCopilotExample.py`.

We will split the problem into steps:

- Loading and cleaning the dataset
- Calculating the percentage increase
- Building the chart
- Plotting the basic chart in Altair
- Enriching the chart

Let’s look at each point in turn, in the coming sections.

2.4.1 Loading and Cleaning the Dataset

Before using the dataset, you must load the required libraries, open the dataset as a Pandas DataFrame, and calculate the percentage increases. The idea is to prepare data to build the chart in Figure 2.13. For simplicity, we will compare Portugal with Germany and not with the mean of the other countries, but you can apply the same procedure also if you want to generalize to the other countries.

Open VSC and write the following comments:

Listing 2.17 Loading and filtering the dataset

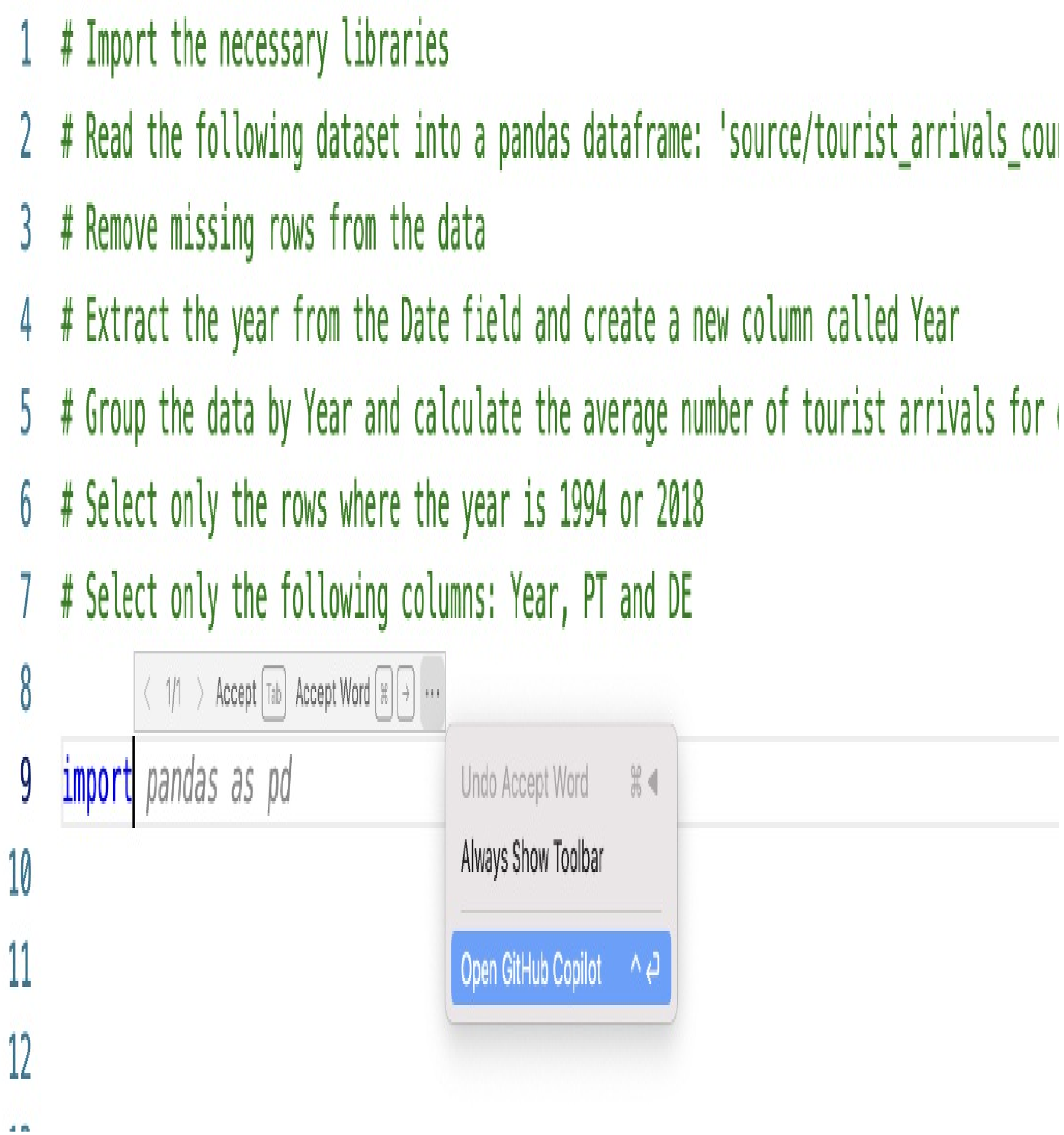
```
# Import the necessary libraries
# Read the following dataset into a pandas dataframe: 'source/tou
# Remove missing rows from the data
# Extract the year from the Date field and create a new column ca
# Group the data by Year and calculate the average number of tour
# Select only the rows where the year is 1994 or 2018
# Select only the following columns: Year, PT and DE
```

Write the text to load and filter the dataset

Describe the sequence of operations to load and filter the dataset.

If you press **Enter** at the end of the last comment, and start writing `import`, Copilot will propose to complete the statement. Just press **Enter** to accept the generated code. Alternatively, hover over the proposed text with the mouse, and open Copilot, as shown in the following figure:

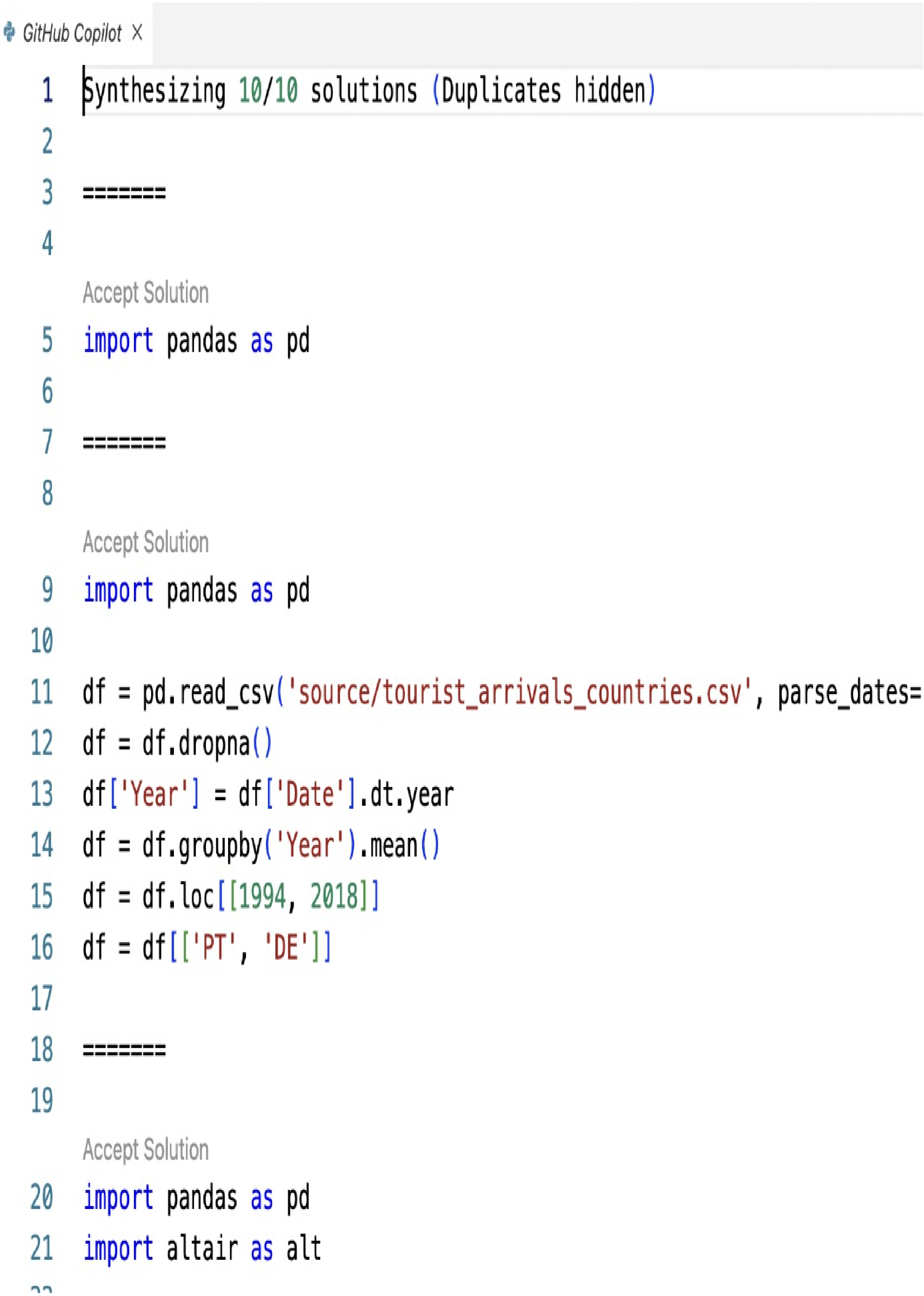
Figure 2.15 How to open Copilot in VSC



```
# Import the necessary libraries
# Read the following dataset into a pandas dataframe: 'source/tourist_arrivals_cou
# Remove missing rows from the data
# Extract the year from the Date field and create a new column called Year
# Group the data by Year and calculate the average number of tourist arrivals for
# Select only the rows where the year is 1994 or 2018
# Select only the following columns: Year, PT and DE

import pandas as pd
```

Depending on the generated code, Copilot may propose different solutions, as shown in the following figure:

Figure 2.16 The different solutions proposed by Copilot in VSC



```
Synthesizing 10/10 solutions (Duplicates hidden)

=======

Accept Solution
import pandas as pd

=======

Accept Solution
import pandas as pd

df = pd.read_csv('source/tourist_arrivals_countries.csv', parse_dates=
df = df.dropna()
df['Year'] = df['Date'].dt.year
df = df.groupby('Year').mean()
df = df.loc[[1994, 2018]]
df = df[['PT', 'DE']]

=======

Accept Solution
import pandas as pd
import altair as alt
```

Click on **Accept Solution** to accept a solution. In our case, the second solution implements the described steps. Please notice that every time you run Copilot, it may generate different solutions, even if you re-run the same text.

2.4.2 Calculating the Percentage Increase

The next step involves building a new dataset, starting from the previously cleaned dataset. The new dataset will contain the percentage increase values for Portugal and Germany. Write the following sequence of operations:

Listing 2.18 Building a new DataFrame with percentage increase

```
# Create a new dataframe df_pi with three columns Year, Country,
# Add a new row to df_pi with the following values: 1994, PT, 0.0
# Add a new row to df_pi with the following values: 1994, DE, 0.0
# For columns PT and DE in the original dataframe df,
   calculate the percentage increase of the second row respect to
   store the result as two new columns in df_pi. For the column y
```

Build the df_pi DataFrame

Describe the structure of the `df_pi` DataFrame.

Similarly to the previous case, Copilot could propose multiple solutions. Open Copilot to check the possible solutions and accept that implementing your operations. If you don’t find any solution that implements your text, rewrite the text by adding more details. If, even after rewriting the text, you can't find any solution that implements your operations, accept the solution closest to your requests and modify it according to your needs.

2.4.3 Plotting the Basic Chart in Altair

Now, we’ll instruct Copilot to build a chart similar to what we saw in Figure 2.13 regarding the percentage increase comparison of Portugal and Germany.

Listing 2.19 Building the chart in Altair

```
# Use the Altair library to plot two lines in the same chart show
```

Build the chart in Altair

Describe how the chart should look.

Copilot could propose multiple solutions. Open Copilot to check the possible solutions and accept that implementing your operations.

2.4.4 Enriching the Chart

Instruct Copilot to add the context and the call to action (as we saw in our approach using Altair), as a title. In addition, save the resulting chart into an HTML file, named `output.html`.

Listing 2.20 Enriching the chart

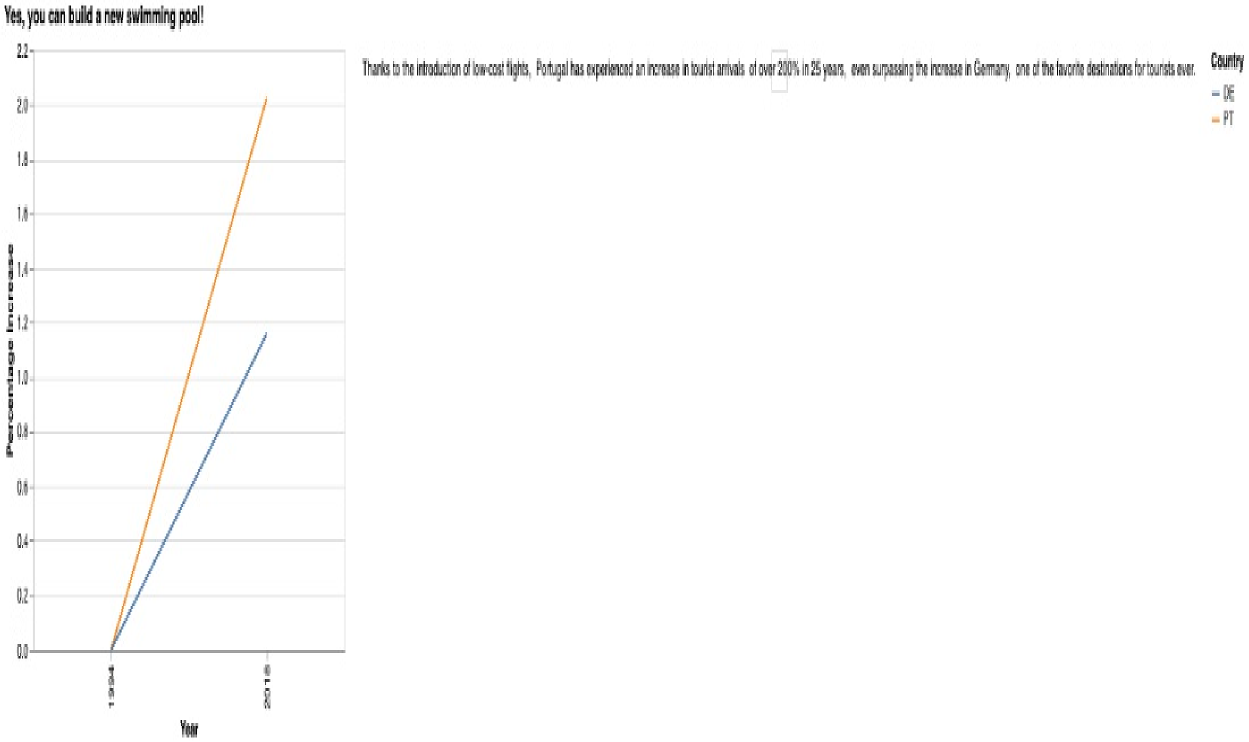
```
# Create a new chart with the following text: 'Thanks to the intr
# Place the two graphs side by side. Set the title to 'Yes, you c
# Save the chart as an HTML. Name the file output.html
```

Add the context and a title

Write the annotation and title to show in the chart.

Once the code is complete, save and run the script. The following figure shows the resulting chart:

Figure 2.17 The chart produced by Copilot



The chart is a good starting point, but it still needs improvement. To improve the chart, you can run the following steps manually:

- Increase the font size of the title and the annotation
- Rotate the labels of the x-axis
- Split the annotation text on the right across multiple lines

In this chapter, you have learned how to turn a raw dataset into a data story, using Altair and Copilot. In the next chapter, you'll review the basic concepts of Altair.

2.5 Summary

- Altair is a declarative library for data manipulation and visualization. It provides three main elements to build a chart: chart, mark, and encodings. Use the chart element to specify the dataset to plot. Use the mark to specify the type of chart to draw and encodings to set the channels, such as the x and y axes.
- To turn raw data into a data story, use the DIKW principles

progressively. Start by cleaning your dataset, and then extract only meaningful information. Draw the extracted information. Then add a context and a call to action as annotations.
- Copilot is a very powerful tool to speed up your code generation. To make Copilot generate your code, split the problem into small steps, and describe each step using natural language. Although Copilot is not perfect, it can assist you in building the framework of your code.
- Use Copilot to generate your code's framework automatically, then improve the code manually.

2.6 References

Dobruszkes, F. (2013). The geography of European low-cost airline networks: a contemporary analysis. Journal of Transport Geography, 28, 75-88.

[1] https://github.com/alod83/Data-Storytelling-with-Python-Altair-and-Generative-AI

[2] https://ec.europa.eu/eurostat/databrowser/view/tour_occ_arm/default/table?lang=en

[3] https://pypi.org/project/pandas-profiling/

[4] https://pypi.org/project/sweetviz/

3 Reviewing the basic concepts of Altair

This chapter covers

- Vega and Vega-Lite
- The basic components of an Altair chart
- Case study

In this chapter, we will review the basic concepts behind Vega and Vega-lite, the visualization grammars upon which Altair is built. Then, we’ll focus on the Altair main components: encodings, marks, conditions, compound charts, and interactivity. In the last part of the chapter, we’ll implement a practical example.

3.1 Vega and Vega-Lite

Vega and Vega-Lite are two visualization grammars used by Altair. A visualization grammar is a set of rules and principles defining how to represent data visually, much like how a spoken language has its grammar. A visualization grammar includes a vocabulary of visual elements, such as points, lines, and bars, and rules for combining and arranging these elements to create meaningful visualizations. Using a visualization grammar allows you to create clear and effective data visualizations that convey insights and tell stories.

Vega and Vega-lite provide declarative language for creating interactive visualizations. Learning Vega and Vega-lite before Altair is important because Altair is built on top of these two visualization libraries. By mastering these foundational tools first, you can gain a deeper understanding of how Altair works and leverage its full potential. Additionally, learning Vega and Vega-lite enable you to create custom visualizations that may not be possible with Altair alone, allowing for more flexibility and creativity in data exploration and communication. In the following section, we'll explore the main components of a Vega and Vega-Lite specification.

3.1.1 Vega

Vega is a visualization grammar to define the visual aspects and interactive features of a chart by writing code in JSON format. Using Vega, you can generate web-based views that utilize HTML5 Canvas or SVG to display the resulting chart. HTML5 Canvas is a bitmap-based drawing technology that enables you to render dynamic graphics and animations on the web. SVG is a vector-based graphics format for building scalable and resolution-independent graphics.

A Vega JSON file contains the specifications for the visual appearance and interactive behavior of a chart. Listing 3.1 shows the basic structure of a JSON specification. For testing, you can use the Vega Editor to render the chart produced by Vega.

Listing 3.1 The basic structure of a Vega JSON file

```
{
  "$schema": "https://vega.github.io/schema/vega/v5.json",
  "description": "An example of vega specification.",
  "width": 600,
  "height": 400,

  "data": [],
  "scales": [],
  "axes": [],
  "marks": [],
  "signals": []
}
```

The Vega JSON file

First, define the version of the schema (`$schema`), a `description`, the `width` and the `height` of the chart. Then, specify the main sections (`signals`, `data`, and so on).

The main sections of a Vega JSON file include data, scales, axes, marks, and signals. In the remainder of this section, we’ll describe an overview of the main sections of a Vega specification. For more details, please refer to the Vega official documentation.

Data

This section defines the data source to use. It specifies the data format, where the data is located, and how it should be loaded and transformed. Listing 3.2 shows an example of a `data` section.

Listing 3.2 An example of data section

```
"data": [
    {
      "name": "mydata",
      "url" : "/path/to/csv/myfile.csv",
      "format": {"type": "csv"}
    }
]
```

The data section

Specify the list of data to use in the visualization. For each data, specify the name and the content. The example retrieves the data from the myfile.csv file at /path/to/csv/. Alternatively, you can retrieve data from other formats or embed them in the JSON directly.

Scales

This section defines the scales that map the data to visual properties. Scales map a data domain (input range) to a visual range (output range). Listing 3.3 shows an example of a `scales` section.

Listing 3.3 An example of scales section

```
"scales": [
    {
      "name": "xscale",
      "domain": {"data": "mydata", "field": "X"},
      "range": "width"
    },
    {
      "name": "yscale",
      "domain": {"data": "mydata", "field": "Y"},
      "range": "height"
    }
]
```

The scales section

Specify the list of scales to use in the visualization. For each scale, specify at least the name, the range, and the domain. The example defines two scales, one for the x-axis and the other for the y-axis. Specify the data to use as the value of the domain attribute.

Axes

This section defines the axes of the chart. Use axes to define tick marks and labels along an axis. Listing 3.4 shows an example of an `axes` section.

Listing 3.4 An example of axes section

```
"axes": [
    { "orient": "bottom", "scale": "xscale" },
    { "orient": "left", "scale": "yscale" }
]
```

The axes section

Specify the list of axes to use in the visualization. For each axis, specify at least the scale and how to orient it. The example defines two axes, one for the x-axis and the other for the y-axis.

Marks

This section defines the visual marks that represent the data. Marks include points, lines, rectangles, areas, and other shapes. Use visual properties like size, color, opacity, and shape to style marks. Listing 3.5 shows an example of a `marks` section.

Listing 3.5 An example of marks section

```
"marks": [
    {
      "type": "line",
      "from": {"data":"mydata"},
      "encode": {
        "enter": {
          "x": {"scale": "xscale", "field": "X"},
          "y": {"scale": "yscale", "field": "Y"},
          "stroke": { "value": "#F67E00"}
        }
      }
    },
]
```

The marks section

Specify the list of marks to use in the visualization. For each mark, specify at least the `type` (line, symbol, rect, and so on), the source of data (`from`), and how to encode the data (`encode`). The example defines a line chart representing data contained in the `mydata` variable defined in the `data` section. It also defines two encoding channels, `x` and `y` and the color of the line stroke.

Signals

This section defines the interactive signals you can use to modify the visualization, such as sliders, dropdowns, and checkboxes. Every signal is composed of two parts: the *listener*, which is identified by the `signals` section, and the *handler*, which is defined by the keyword `signal` within the marks section. The signal listener responds to signal changes, and the signal handler determines how the visualization reacts. For example, a signal handler might change the color or size of data points based on a signal's value.

Listing 3.6 shows an example that changes the color of a selected bar in a bar chart when the user hovers over the bar with the mouse (Figure 3.1).

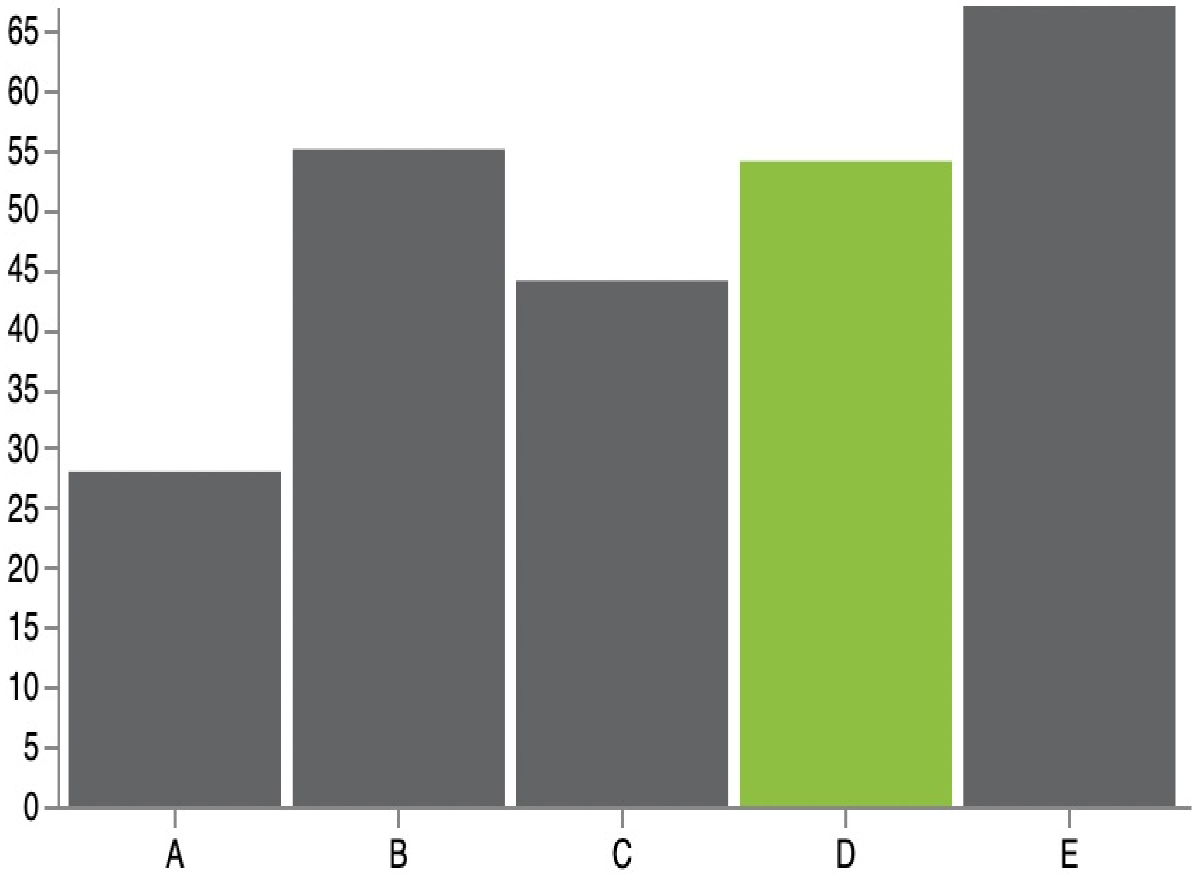
Listing 3.6 An example of signals section

```
"signals": [
    {
      "name": "changeColor",
      "value": {},
      "on": [
        {"events": "rect:mouseover", "update": "datum"},
        {"events": "rect:mouseout",  "update": "{}"}
      ]
    }
  ],

"marks": [
    {
      "type": "rect",
      ...
      "encode": {
         ...
         "update": {
          "fill": { "signal": "changeColor === datum ? '#80C11E'
        }
      }
    }
  ]
```

The signals section

Specify the list of signals to use in the visualization. For each signal, specify the property to modify in the mark specification (`value`, in the example) and the list of events to listen (`on`). Then, specify the handler, under the marks section, through the `signal` property. The example defines a signal that listens to mouse over and mouse out events on the bars of a bar chart. In case of mouse over, the signal passes the current `datum` to the handler. Within the `update` property of `marks` section, the example defines a handler that checks whether the signal corresponds to the mouse over event (`changeColor === datum`) and then sets the color accordingly. Refer to the Vega documentation for more details.

Figure 3.1 A bar chart highlighting a bar when hovering over it with the mouse



You can find the described example in the GitHub repository of the book under 03/vega/json/spec.json. The repository also describes an example of line chart specification in Vega (03/vega/json/spec2.json).

Now that you have learned the basic concepts behind Vega, let’s move to the next step, Vega-lite.

3.1.2 Vega-lite

Vega-lite is a concise JSON of a Vega visualization. Vega-lite still maintains the specifications already defined in Vega, but it represents them through a more compact syntax. The basic sections of a Vega-lite JSON specification are data, encoding, mark, and layer.

Data

This section specifies the data source for the visualization. Listing 3.7 shows an example of a data section in Vega-lite.

Listing 3.7 An example of data section

```
"data": {
  "url": "/path/to/my/mydata.csv",
  "format": {"type": "csv"}

}
```

The data section

Specifies the data to use in the visualization. Refer to the [Vega-lite documentation]() for more details.

Encoding

This section maps the data fields to visual properties such as position, color, and size. Listing 3.8 shows an example of an encoding section in Vega-lite.

Listing 3.8 An example of encoding section

```
"encoding": {
      "x": {"field": "X", "type": "quantitative"},
      "y": {"field": "Y", "type": "quantitative"}
    },
```

The encoding section

Specify the encodings to use in the visualization. The example defines two channels, `x` and `y`, and specifies the data type as `quantitative`.

Mark

This section defines the type of visual mark such as bars, points, or lines. Listing 3.9 shows an example of a mark section in Vega-lite.

Listing 3.9 An example of mark section

```
"mark": {"type": "line", "color": "#F67E00"}
```

The mark section

Specify the mark to use in the visualization. The example defines a line chart and sets its color to `#F67E00`.

Layer

This section combines multiple marks in a single chart. Listing 3.10 shows an example of a mark section in Vega-lite.

Listing 3.10 An example of layer section

```
"layer": [
   {"mark": {"type": "line", "color": "#F67E00"}},
   {"mark": {"type": "point", "shape": "circle", "size": 100, "co
]
```

The layer section

Specify the list of marks to combine. This combines two charts, a line chart, and a point chart.

Now that you have learned the basic concepts behind Vega and Vega-lite, let’s see how to render a Vega/Vega-lite visualization.

3.1.3 How to render a Vega/Vega-lite visualization

Both Vega and Vega-lite are JSON objects that need a renderer to be shown, such as an external Javascript library, for the visualization into an HTML page. Vega-lite provides a Javascript API that automatically builds both Vega and Vega-lite visualizations.

To render a Vega or Vega-lite JSON, import the following Javascript libraries in your HTML file:

- https://cdn.jsdelivr.net/npm/vega@5.22.1

- https://cdn.jsdelivr.net/npm/vega-lite@5.6.0
- https://cdn.jsdelivr.net/npm/vega-embed@6.21.0

The library's versions may vary. In our case, we import version 5.22.1 for vega, 5.6.0 for vega-lite, and 6.21 for vega-embed. Then, you can write the code described in Listing 3.11 to render your Vega/Vega-lite JSON file.

Listing 3.11 Rendering a Vega/Vega-lite JSON specification

```
var request = new XMLHttpRequest();
request.open("GET", "/path/to/JSON/specification", false);
request.send(null)
var data = JSON.parse(request.responseText);
vegaEmbed('#vis', data);
```

The Javascript code to render a Vega/Vega-lite JSON specification

Open a new `XMLHttpRequest()` to load the JSON file and then use the `vegaEmbed()` function to render the JSON. Note that you must provide a valid external URL to the request object. You can’t use a local JSON file.

For example, to render the chart described in Listing 3.6, use the following HTML specification:

Listing 3.12 The complete HTML file

```
<!DOCTYPE html>
<html>
  <head> #A
    <script src="https://cdn.jsdelivr.net/npm/vega@5.22.1"></scri
    <script src="https://cdn.jsdelivr.net/npm/vega-lite@5.6.0"></
    <script src="https://cdn.jsdelivr.net/npm/vega-embed@6.21.0">
  </head>
  <body>
    <div id="vis"></div> #B
    <script type="text/javascript"> #C
      var request = new XMLHttpRequest();
      request.open(‘GET’, ‘/path/to/json/file’, false);
      request.send(null)
      var data = JSON.parse(request.responseText);
      vegaEmbed('#vis', data);
    </script>
  </body>
</html>
```

The HTML code

First, import the Vega and Vega-lite libraries in the header section. Then, create a div that will contain the chart. Finally, build the chart. Use an external JSON file to define the chart specifications.

You can find the complete example and other examples in the GitHub repository of the book, section 03/vega and 03/vega-lite.

Now that you have learned the basic concepts behind Vega and Vega-lite, let’s move to the next step, the basic components of an Altair chart.

3.2 The Basic Components of an Altair Chart

In this book, we will use the *just-in-time teaching* methodology, which combines theory and practice so that you’ll learn a concept at the moment you’ll need it. However, in this chapter, we will not use this methodology. The reason for this is that we want to establish a strong foundation of fundamental concepts about Altair before diving into the more advanced topics covered later in the book. By deviating from the just-in-time teaching methodology for this chapter, we can provide a comprehensive overview and in-depth understanding of the underlying principles that will serve as building blocks for the subsequent material.Altair is built on top of Vega-lite and offers a user-friendly API to build charts. You have already learned how to build a basic chart in Altair in Chapter 2, *Running Your First Data Story in Altair and GitHub Copilot*. In this chapter, we’ll dive deep into the basic components of an Altair chart.

The basic components of an Altair chart are:

- **Encodings** define how data is mapped to visual properties, such as color, shape, and size
- **Marks** refer to the visual elements representing the data, such as bars, points, and lines

- **Conditions** enable us to create more complex visualizations by specifying rules for when we should use certain encodings or marks
- **Compound charts** combine multiple visual elements to create more complex and informative visualizations
- **Interactivity** enables us to interact with the visualization and explore the data
- **Configurations** define general properties for compound charts.

Let’s analyze each component separately, starting from the first one, encodings. You can find some sample code in the 03/altair directory of the GitHub repository of the book.

3.2.1 Encodings

Altair utilizes encodings to indicate where to show data. To define an encoding, you mustdefine an encoding channel, which is the fundamental component of visualizations in Altair. Each encoding channel corresponds to a specific column of the DataFrame that can be mapped to a visual feature of the plot. The encoding process determines where to display the data in the chart according to the chosen channel type. Altair identifies several key types of channels, including the following ones:

- **Position**, which specifies the location of data in the chart
- **Mark Property**, which determines the chart's appearance such as color, size, and shape
- **Text and Tooltip**, which offer supplementary annotations for the graph.

Table 3.1 shows a short description of the main encodings. For more details, please refer to the Altair documentation.

Table 3.1 The main encodings in Altair

Type	Channel	Description
		Horizontal

Position	`x, y`	and vertical positions
	`longitude, latitude`	Geographica coordinates
	`xError, xError`	Error values for x and y
	`x2,y2, longitude2, latitude2`	Second positions fo ranges
	`theta, theta2`	Start and en arc angles
Mark Property	`angle, radius`	The angle and the radius of the mark
	`color, fill`	The color and the fill of the mark
	`opacity, fillOpacity`	The opacity and the fill opacity of the mark

	`shape, size`	The shape and the size of the mark
	`stroke, strokeDash, strokeOpacity, strokeWidth`	Stroke properties o the mark
Text and Tooltip	`text`	The text of the mark
	`tooltip`	The tooltip of the mark

Refer to Chapter 2, *Running Your First Data Story in Altair and GitHub Copilot*, to build a chart using encodings.

3.2.2 Marks

Marks enable you to represent data in visualizations. Examples of marks include bar charts, line charts, heat maps, and box plots, among others. To specify the type of mark, use the `mark_<type>()` function, where `<type>` represents the specific type of mark. For instance, to create a bar chart, use `mark_bar()`. Additionally, you can provide a list of attributes as input parameters for each mark property defined in Table 3.1. When using a mark property within the `mark_<type>()` function, the value of that property is fixed to a constant value. For example, `mark_circle(size=5)` will draw circles with a constant size of 5. However, if you want to vary the size of the circles depending on a specific column of your DataFrame, use the size channel within the encoding function.

Table 3.2 shows the main mark types provided by Altair.

Table 3.2 The main mark types provided by Altair

Method	Description
mark_arc()	A pie chart
mark_area()	A filled area chart
mark_bar()	A bar chart
mark_circle()	A scatter plot with filled points
mark_geoshape()	A geographic shape
mark_line()	A line chart
mark_point()	A scatter plot with configurable points
mark_rect()	A filled rectangle for heatmaps
mark_rule()	A vertical or horizontal line spanning the axis
mark_text()	A text

Refer to Chapter 2, *Running Your First Data Story in Altair and GitHub*

Copilot, to build a chart using marks.

3.2.3 Conditions

Altair provides several ways to create conditions in visualizations, which enable you to customize and control the appearance of your charts based on specific criteria. One common way to create conditions in Altair is to use the *if-else* statements in the encoding channels of a chart. To define a condition in Altair, use the `alt.condition()` method, which takes three arguments:

- A condition
- A value to use when the condition is true
- A value to use when the condition is false.

For instance, if we want to change the color of the bars in a bar chart based on a specific condition, write the code shown in Listing 3.13.

Listing 3.13 Using the alt.condition() method

```
df = pd.DataFrame(
    {"Country": ["Japan", "USA", "Germany", "Spain", "France", "I
     "Medals": [4, 6, 10, 3, 7, 8],
     "Region":["Asia","Americas","Europe","Europe","Europe","Euro

chart = alt.Chart(df).mark_bar().encode(
    x='Medals',
    y='Country',
    color=alt.condition(
        alt.datum.Region == 'Europe',
        alt.value('red'),  # color to use when condition is true
        alt.value('blue')  # color to use when condition is false
    )
)

chart.show()
```

How to write a conditional statement in Altair

The condition in the example checks if the Regionis the Europe. If the condition is true, the bars will be colored red, and if the condition is false, the bars will be colored blue.

In addition to using *if-else* statements with the `alt.condition()` method and the type-checking functions, Altair provides two other ways to create conditions in visualizations: *transformations* and *selections*. You will see transformations in Chapter 5, *From Data to Information: Extracting Meaning*, and selections in Chapter 6, *From Information to Knowledge: Building Context*.

3.2.4 Compound Charts

Compound charts enable you to combine visualizations together in a single chart. Altair supports the following types of compound charts: layering, horizontal concatenation, vertical concatenation, and repeated charts. To explain how each type of compound chart works, consider the two charts described in Listing 3.13.

Listing 3.14 Building two charts in Altair

```
import pandas as pd
import altair as alt

df = pd.DataFrame(
    {"Country": ["Japan", "USA", "Germany", "Spain", "France", "I
     "Medals": [4, 6, 10, 3, 7, 8],
     "Region":["Asia","Americas","Europe","Europe","Europe","Euro


chart1 = alt.Chart(df).mark_bar(color='#636466').encode(
    x='Country',
    y='Medals'
).properties(width=width, title='A bar chart')

chart2 = alt.Chart(df).mark_line(color='#80C11E').encode(
    x='Country',
    y='Medals'
).properties(width=width, title='A line chart')
```

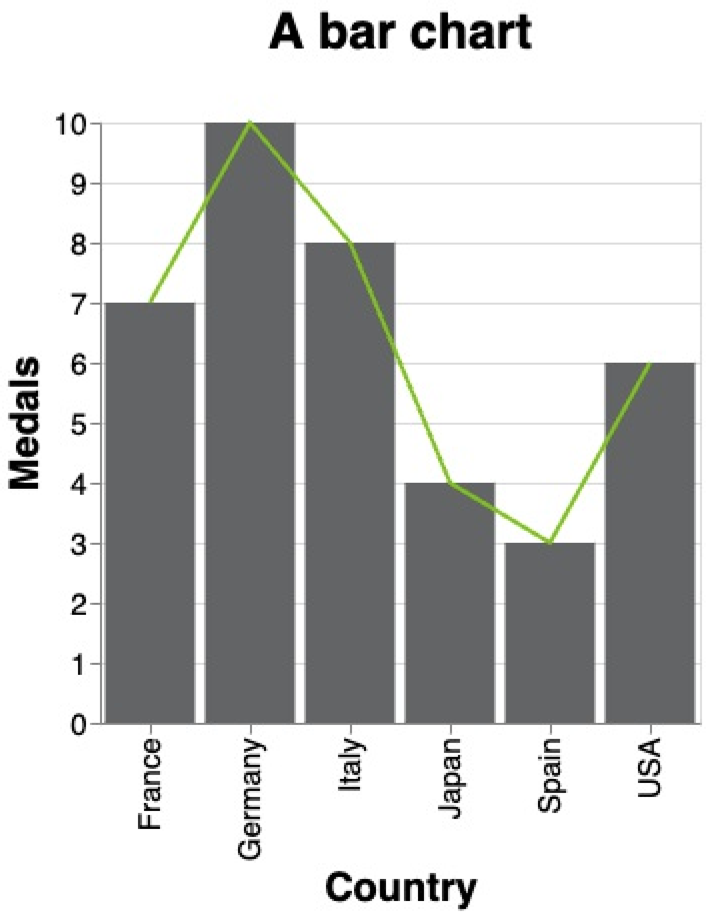
Two charts in Altair

Build a bar chart (`chart1`) and a line chart (`chart2`) from a Pandas DataFrame. Use `properties()` to set the chart properties, such as the width and the title.

Layering

Layering lets you combine charts on top of each other, as shown in Figure 3.2. In Altair, use the `alt.layer(chart1,chart2)` function to layer two charts, or the + operator. If each chart has a title, and you don't specify any general title for the compound chart, layering will set the title of the layered chart as that of the first chart.

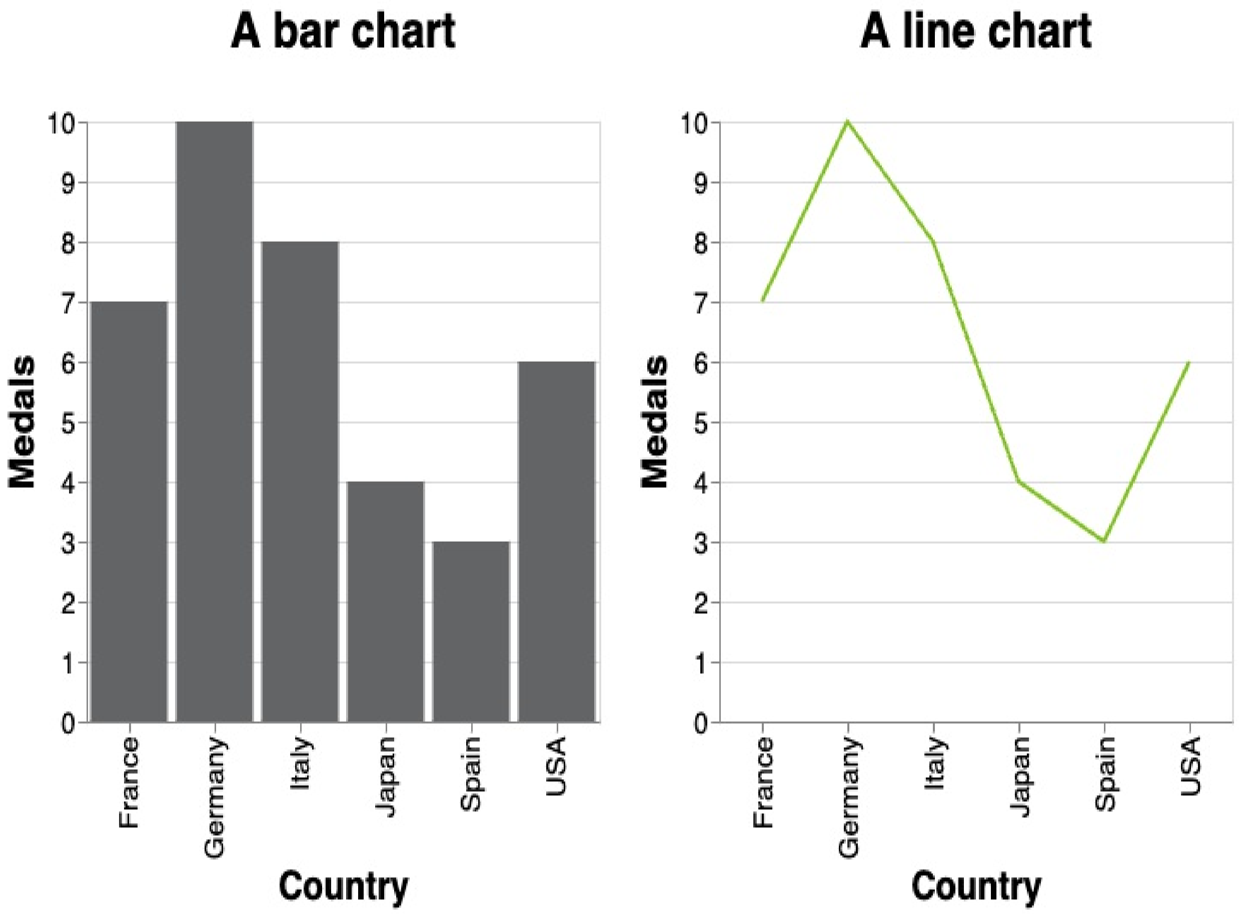
Figure 3.2 A bar chart layered with a line chart



Horizontal Concatenation

Horizontal Concatenation enables you to concatenate charts horizontally, as shown in Figure 3.3. In Altair, use the `alt.hconcat(chart1,chart2)` function to layer two charts, or alternatively, the `|` operator.

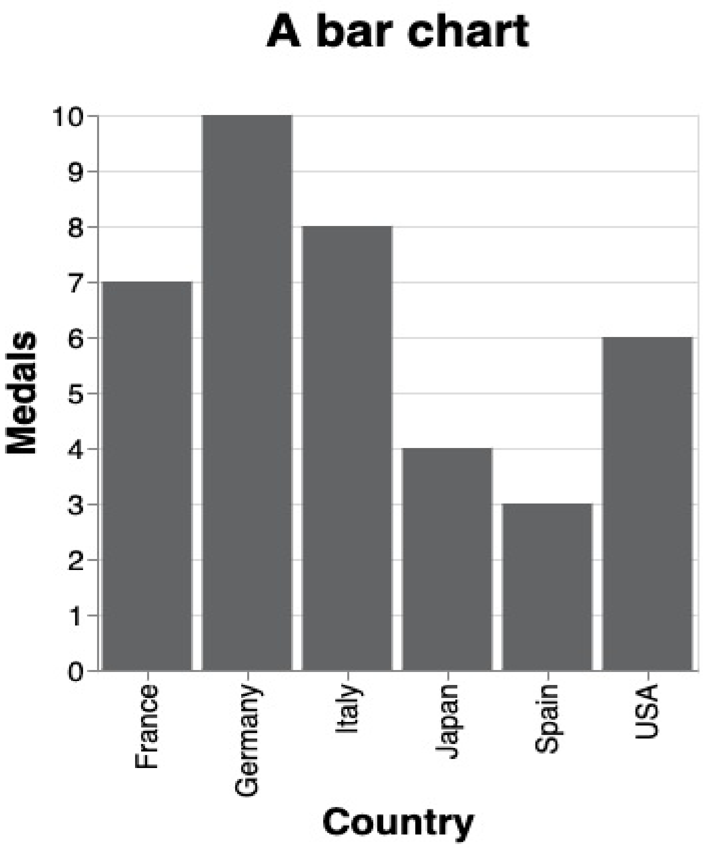
Figure 3.3 A bar chart concatenated horizontally with a line chart

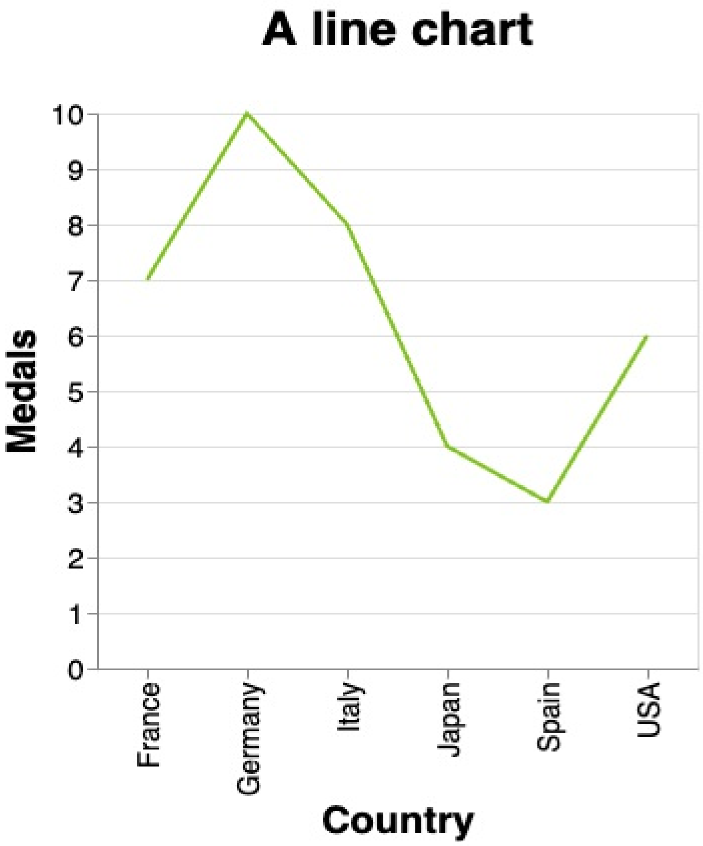


Vertical Concatenation

Vertical Concatenation enables you to concatenate charts vertically, as shown in Figure 3.4. In Altair, use the `alt.vconcat(chart1,chart2)` function to layer two charts or the `&` operator.

Figure 3.4 A bar chart concatenated vertically with a line chart


A bar chart
Medals
10
9
8
7
6
5
4
3
2
1
0
France
Germany
Italy
Japan
Spain
USA
Country


A line chart
Medals
10
9
8
7
6
5
4
3
2
1
0
France
Germany
Italy
Japan
Spain
USA
Country

Repeated Charts

A repeated chart displays similar data views in a single visualization. Through a repeated chart, you can create a set of charts, each displaying a subset of the data, by repeating a base chart using different data subsets or by changing a visual encoding.

Use the `repeat()` function to create a repeated chart in Altair. This function takes the list of fields for the repeat, as shown in Listing 3.14.

Listing 3.15 Building a repeated chart in Altair

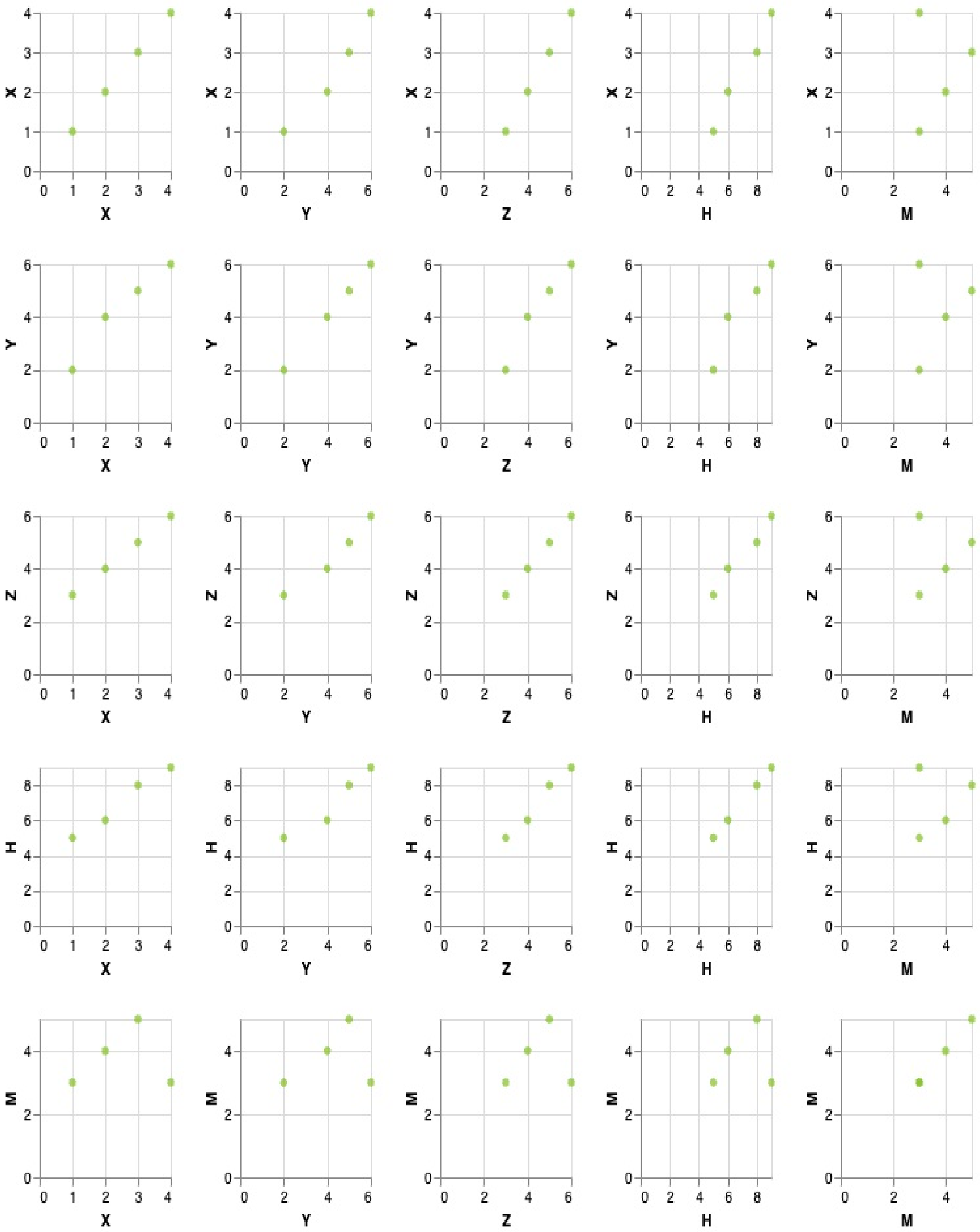
```
import pandas as pd
import altair as alt

df = pd.DataFrame({
'X' : [1,2,3,4],
'Y' : [2,4,5,6],
'Z' : [3,4,5,6],
'H' : [5,6,8,9],
'M' : [3,4,5,3],
'Country' : ['USA', 'EU', 'EU', 'USA']
})

fields = df.columns.tolist()
fields.remove('Country')

chart = alt.Chart(df).mark_circle(color='#80C11E').encode(
    alt.X(alt.repeat("column"), type='quantitative'),
    alt.Y(alt.repeat("row"), type='quantitative')
).properties(
    width=100,
    height=100
).repeat(
    row=fields,
    column=fields
)

chart.show()
```

A repeated chart

First create the Pandas DataFrame. Then, extract the list of fields to repeat and store them in the fields variable. Next, define the `repeat()` method with the rows and columns to repeat. Finally, use the `row` and `column` variables just defined as a variable of the chart encodings.

Figure 3.5 shows the output of Listing 3.15: Altair has built 25 charts, one for each combination of the columns of the DataFrame specified in the `fields` variable.

Figure 3.5 A bar chart repeated varying the encodings



3.2.5 Interactivity

Interactivity refers to the ability to manipulate visualizations through user input, such as hovering over data points, clicking on legends, or dragging sliders. In Altair, use the `interactive()` method to enable features like zooming, panning, and hovering over data points to display tooltips. Listing 3.16 shows an example of how to make a chart interactive, and Figure 3.6 the resulting chart.

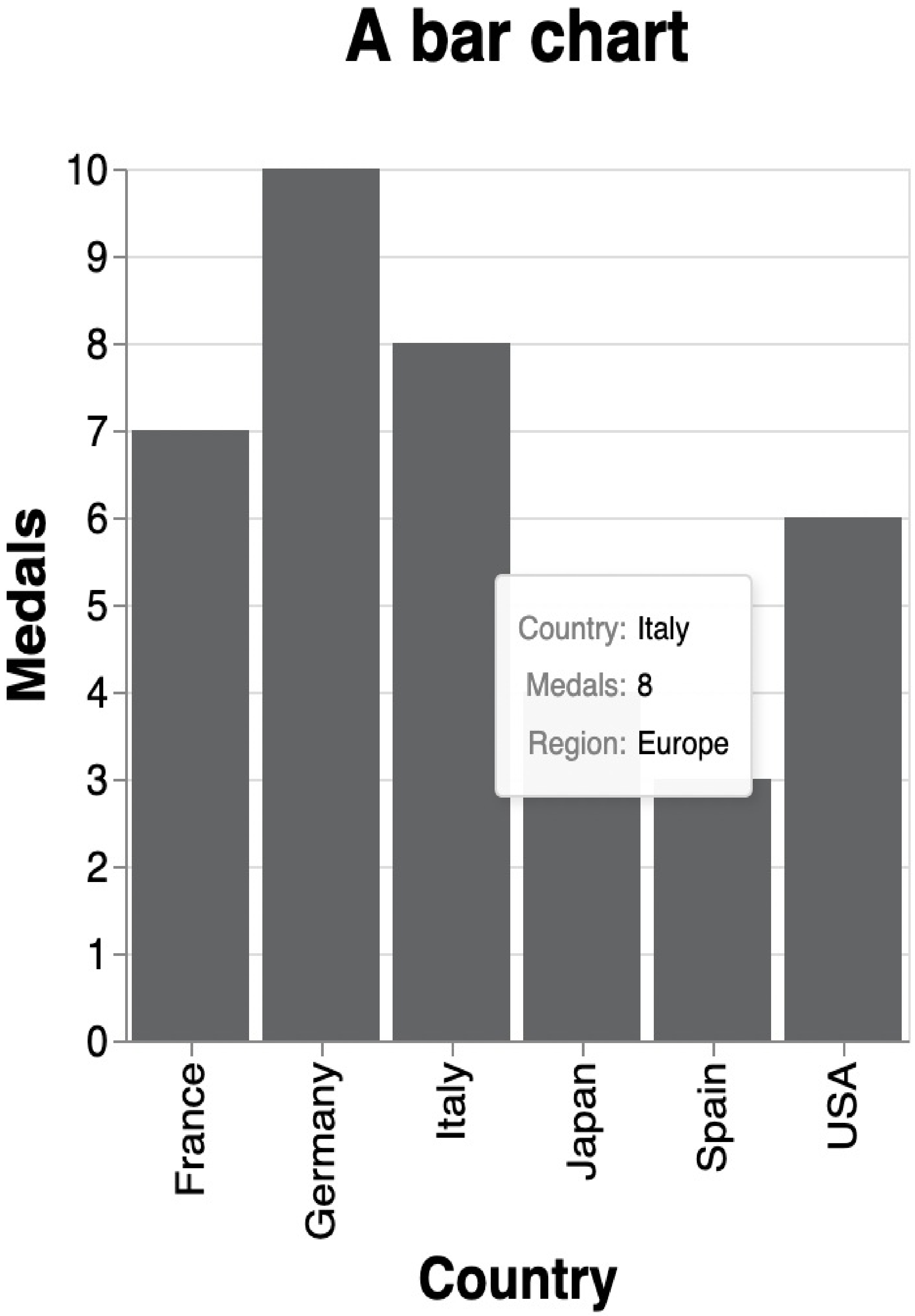
Listing 3.16 Building an interactive chart in Altair

```
import pandas as pd
import altair as alt

df = pd.DataFrame(
    {"Country": ["Japan", "USA", "Germany", "Spain", "France", "I
     "Medals": [4, 6, 10, 3, 7, 8],
     "Region":["Asia","Americas","Europe","Europe","Europe","Euro

chart = alt.Chart(df).mark_bar(color='#636466').encode(
    x='Country',
    y='Medals',
    tooltip=['Country', 'Medals', 'Region']
).properties(
    width=width,
    title='A bar chart'
).interactive()

chart.show()
```

An interactive chart

First, create the Pandas DataFrame. Then, build the Altair chart appending the `interactive()` method at the end. Also, add a tooltip to show. The tooltip receives the list of DataFrame columns to show.

Figure 3.6 A bar chart with interactivity enabled



Country: Italy
Medals: 8
Region: Europe

3.2.6 Configurations

To configure the basic properties of a single chart, use the `properties()` method. For more complex properties and to configure a compound chart, instead, you must use global configurations. Altair supports many global configurations. In this section, we will describe axes, title, and view. We will describe the other configurations later in the book, when we will require them.

To show how configurations work, we will start with the chart described in Figure 3.7 and generated through the code contained in the following listing:

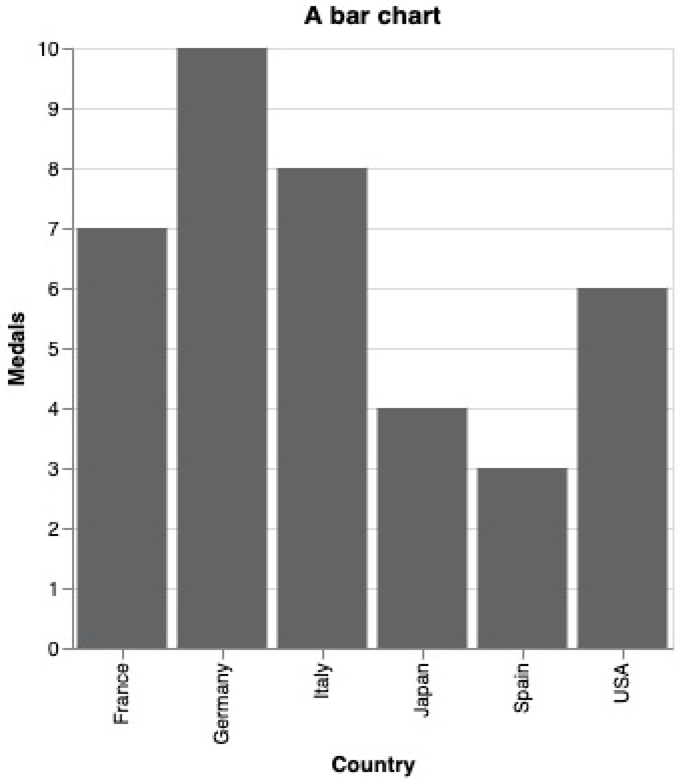
Listing 3.17 Defining the basic chart

```
import pandas as pd
import altair as alt

df = pd.DataFrame(
    {"Country": ["Japan", "USA", "Germany", "Spain", "France", "I
     "Medals": [4, 6, 10, 3, 7, 8],
     "Region":["Asia","Americas","Europe","Europe","Europe","Euro

chart = alt.Chart(df).mark_bar(color='#636466').encode(
    x='Country',
    y='Medals'
).properties(width=width, title='A bar chart')

chart.show()
```

The basic bar chart

Use `chart.show()` to render the chart.

Figure 3.7 The basic bar chart built by the default configuration



Now let’s start to modify the basic chart by configuring axes.

Configuring Axes

To configure axes at a global level, Altair supports three methods:

- `config_axis()` configures both axes
- `configure_axisX()` configures x-axis
- `configure_axisY()` configures y-axis.

Configuring axes enables you to customize properties such as labels, tick marks, scales, etc. The following listing shows an example:

Listing 3.18 Configuring axes

```
chart = chart.configure_axis(
    labelAngle=0,  #A
    titleFontSize=14,  #B
    labelFontSize=12  #C
)
```

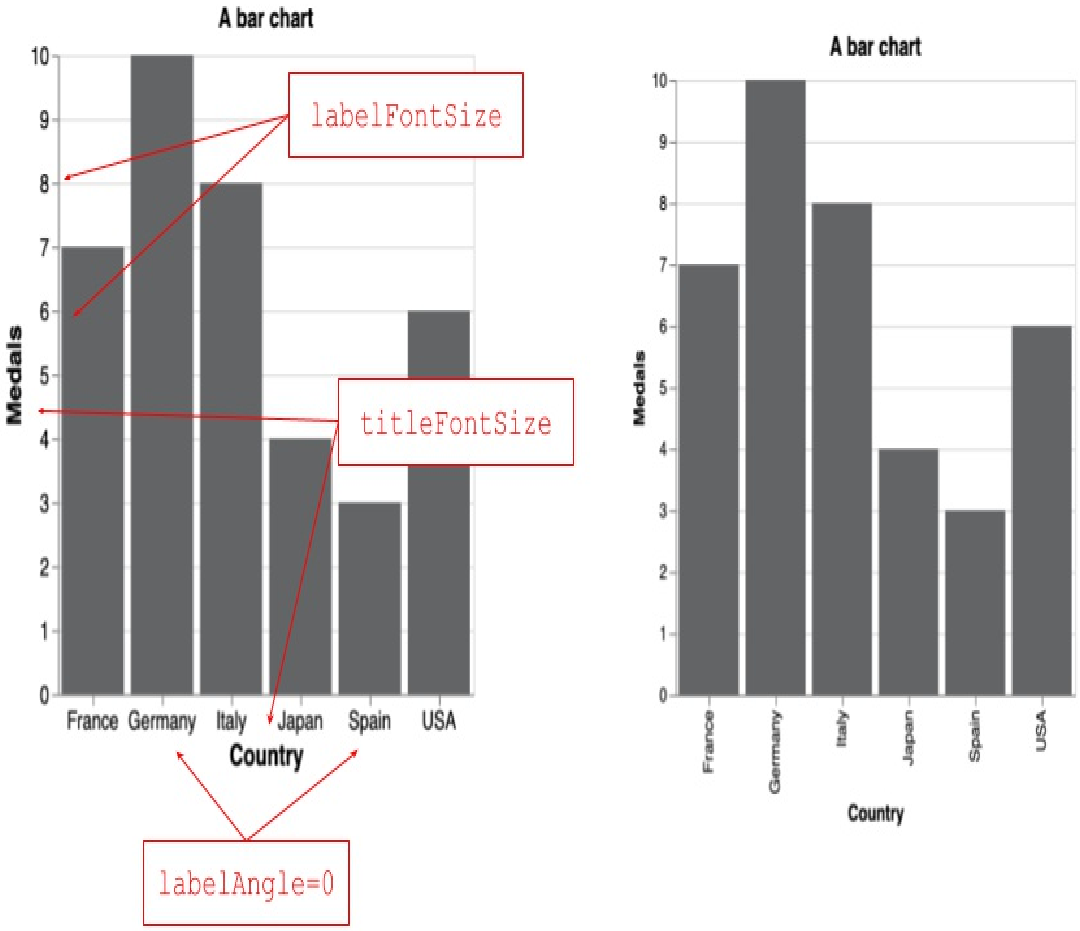
```
chart.show()
```

How to configure axes

Use `configure_axis()` to configure labels and titles in axes. As an alternative, use `configure_axisX()` to configure the x-axis and `configure_axisY()` to configure the y-axis.

Figure 3.8 shows the difference between the layouts when using `configure_axis()` (on the left) and not using `configure_axis()` (on the right).

Figure 3.8 The difference between the layouts with configure_axis() and without configure_axis()



with `configure_axis()`

without `configure_axis()`

Configuring Title

Use the `configure_title()` method to customize the title properties of your chart, such as color, font size, and font family. The following listing shows an example of the usage of the `configure_title()` method.

Listing 3.19 Configuring title

```
chart = chart.configure_title(
    fontSize=20, #A
    color='#80C11E', #B
    offset=30, #C
    anchor='start' #D
)
```

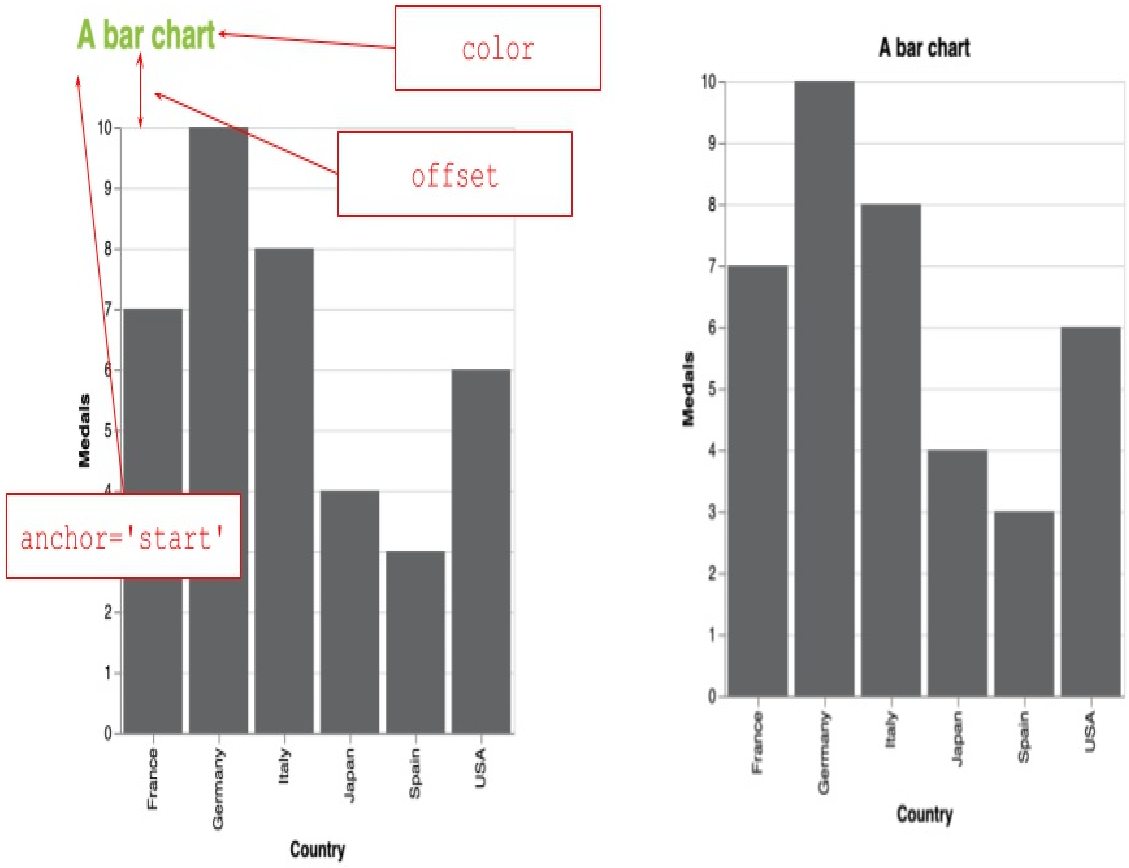
```
chart.show()
```

How to configure the title

Use `configure_title()` to configure the title properties. Anchor must be one of None, 'start', 'middle', 'end'.

Figure 3.9 shows the difference between the layouts when using `configure_title()` (on the left) and not using `configure_title()` (on the right).

Figure 3.9 The difference between the layouts with configure_title() and without configure_title()



with `configure_title()`

without `configure_title()`

Configuring View

Use the `configure_view()` method to customize the general view of the chart. The following listing shows an example of usage of the `configure_view()` method.

Listing 3.20 Configuring view

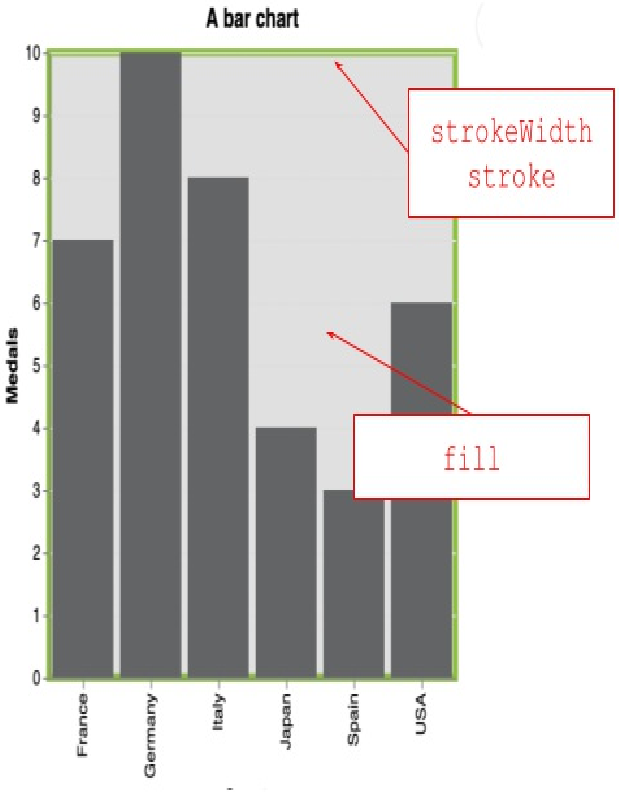
```
chart = chart.configure_view(
    strokeWidth=0, #A
    stroke='#80C11E', #B
    fill='#E0E0E0' #C
)

chart.show()
```

How to configure the view

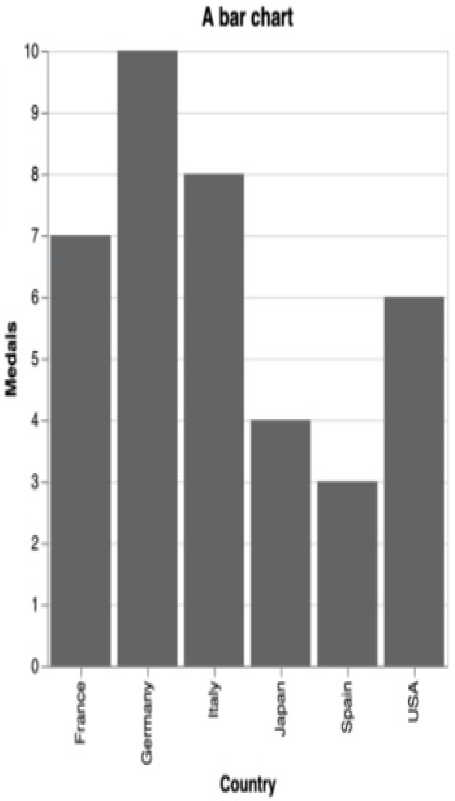
Use `configure_view()` to configure the general view of the chart.

Figure 3.10 shows the difference between the layouts when using `configure_view()` (on the left) and not using `configure_view()` (on the right).

Figure 3.10 The difference between the layouts with configure_view() and without configure_view()



with `configure_view()`



without `configure_view()`

Now that you have learned the basic components of an Altair chart, let’s move to the next step, implementing a case study.

3.3 Case Study

Let us imagine you work as a researcher at the Global Demographic Analysis Center (GDAC). You want to study the population growth in North America. You have found the Population dataset, released by World Bank under the CC BY-4.0 license[1]. You can find a cleaned version of the dataset in the GitHub repository of the book under 03/data/population.csv. The dataset contains the population from 1960 to 2021 for all the countries of the world, as shown in Table.

Table 3.3 The population dataset

Country Name	1960	…	2021
Aruba	54,608		106,537
…			
Zimbabwe	3,806,310		15,993,524

You start by drawing a raw chart (Listing 3.21), showing all the trendlines from 1960 to 2021 for all the countries. Figure 3.11 shows the resulting chart.

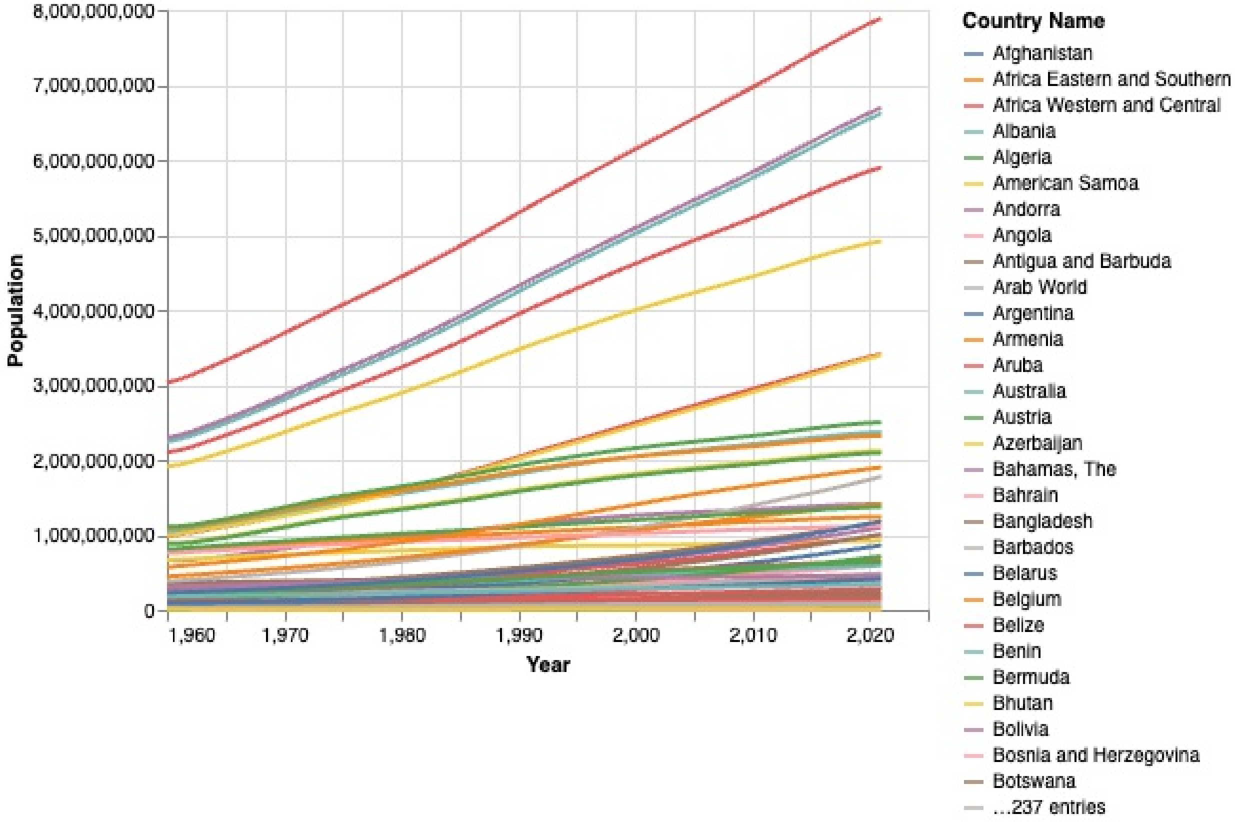
Listing 3.21 Drawing the raw chart

```
import pandas as pd
import altair as alt
alt.data_transformers.disable_max_rows() #A

df = pd.read_csv('data/population.csv')
df = df.melt(id_vars='Country Name',
             var_name='Year',
             value_name='Population')
df['Year'] = df['Year'].astype('int') #B

chart = alt.Chart(df).mark_line().encode(
    x = 'Year:Q',
    y = 'Population:Q',
    color = 'Country Name:N'
) #C

chart.show()
```

How to draw the raw chart

Use `mark_line()` to draw the chart. To plot the different countries, use the `color` channel.

Figure 3.11 The chart with raw data



The chart is very confusing because it presents the following problems: too many countries, too many colors, and no focus in North America. To solve these issues, let us use the DIKW pyramid, starting from the first step, turning data into information.

3.3.1 From Data to Information

Turning data into information involves extracting meaning from data. From the point of view of a chart, it means making the chart readable and understandable. Start by grouping countries by continent. The dataset already contains values for continents. List the countries using the `unique()` method provided by the Pandas DataFrame applied to the Country Name column. For more details on the `unique()` method, please refer to Appendix B. We note that the list of countries also contains the continents, so we can build a list of the continents and filter the DataFrame based on that list. Then, we plot the resulting chart, as shown in the following listing.

Listing 3.22 Decluttering the chart

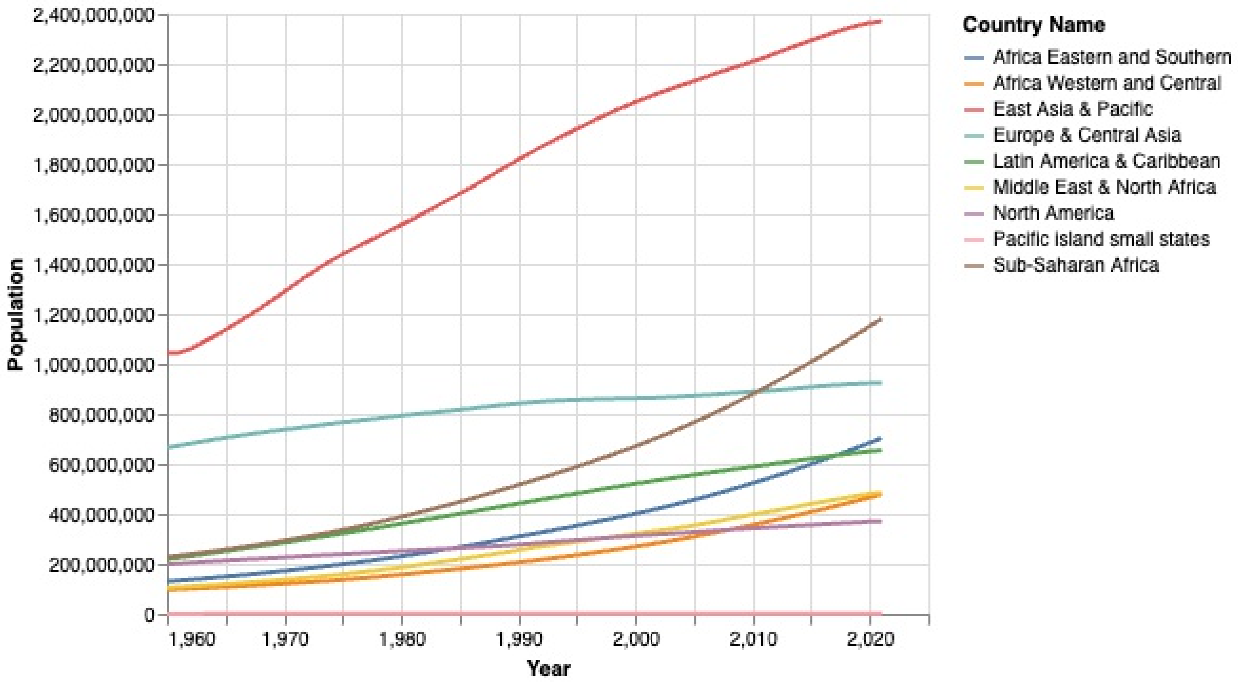
```
continents = ['Africa Eastern and Southern',
             'Africa Western and Central',
             'Middle East & North Africa',
              'Sub-Saharan Africa',
             'Europe & Central Asia',
             'Latin America & Caribbean',
             'North America',
             'Pacific island small states',
             'East Asia & Pacific'] #A

df = df[df['Country Name'].isin(continents)] #B

chart = alt.Chart(df).mark_line().encode(
    x = 'Year:Q',
    y = 'Population:Q',
    color = 'Country Name:N'
) #C

chart.show()
```

How to declutter the raw chart

Group data by continents to reduce the number of lines.

Figure 3.12 shows the resulting chart, which is more readable. However, it still presents some problems: too many colors and no focus on North America. In practice, it is difficult to compare North America with the other countries.

Figure 3.12 The chart with a focus on continents



Change the chart by focusing on North America and grouping the other continents by calculating their average value, as shown in the listing below:

Listing 3.23 Focussing on North America

```
mask = df['Country Name'].isin(['North America']) #A
df_mean = df[~mask].groupby(by='Year').mean().reset_index() #B

df_grouped = pd.DataFrame({
    'Year' : df[mask]['Year'].values,
    'North America' : df[mask]['Population'].values,
    'World': df_mean['Population'].values
}) #C

df_melt = df_grouped.melt(id_vars='Year', var_name='Continent', v
```

How to focus on a single continent

Focus on North America and calculate the average value of the other continents.

Then, draw the chart, as shown in the listing below:

Listing 3.24 Drawing the chart

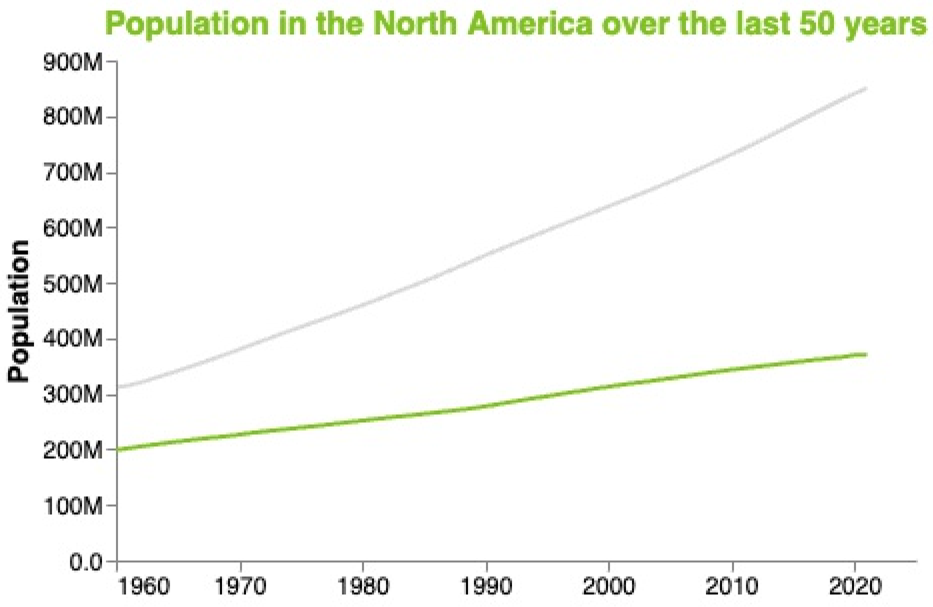
```
colors=['#80C11E', 'grey']

chart = alt.Chart(df_melt).mark_line().encode(
    x = alt.X('Year:Q',
              title=None,
              axis=alt.Axis(format='.0f',tickMinStep=10)), #A
    y = alt.Y('Population:Q',
              title='Difference of population from 1960',
              axis=alt.Axis(format='.2s')), #B
    color = alt.Color('Continent:N',
                      scale=alt.Scale(range=colors), #C
                      legend=None), #D
    opacity = alt.condition(alt.datum['Continent'] == 'North Amer
).properties(
    title='Population in the North America over the last 50 years
    width=400,
    height=250
).configure_axis(
    grid=False,#E
    titleFontSize=14,
    labelFontSize=12
).configure_title(
    fontSize=16,
    color='#80C11E'
).configure_view(
    strokeWidth=0
)

chart.show()
```

How to draw the chart

To format the x-axis labels, use the `format` parameter. Use `.0f` not to show commas. Also, use the `tickMinStep` parameter to set the distance between two labels. The example sets the value to `10`, thus showing the years 10 by 10. To format the y-axis labels, set the format parameter to .2s, using scientific notation with two significant figures.

Figure 3.13 shows the resulting chart. We have omitted the legend because we will add it later. The chart is clearer. However, the audience needs to do some calculations to understand the difference between the two lines

Figure 3.13 The chart focusing on North America



When drawing a chart, do the calculations instead of the audience to serve them the ready-made dish. Let us calculate the difference from the starting year (1960) for both lines, thus making them comparable. The listing below shows how to calculate the difference.

Listing 3.25 Calculating the difference

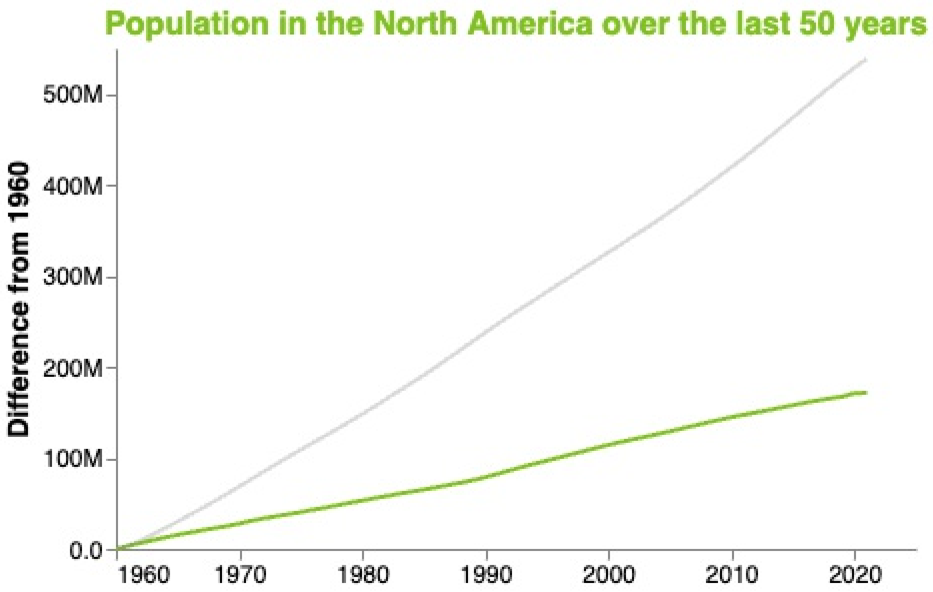
```
baseline = df_melt[df_melt['Year'] == 1960]

continents = ['North America', 'World']
for continent in continents:
    baseline_value = baseline[baseline['Continent'] == continent]
    m = df_melt['Continent'] == continent
    df_melt.loc[m, 'Diff'] = df_melt.loc[m,'Population'] - baseli
```

How to calculate the difference

Calculate the baseline value for 1960 and then, for each continent, calculate the difference between each year and the baseline.

Once calculated the difference, draw the chart. The code is similar to that in Listing 3.24. Change only the y-axis column to `Diff`. Figure 3.14 shows the resulting chart.

Figure 3.14 The chart with a focus on the difference in population from 1960



The chart clearly shows the difference between the two lines, although it is not explicit. To complete the chart, add a label for each line as an additional textual mark, as shown in the listing below:

Listing 3.26 Adding labels

```
mask = df_melt['Year'] == 2021
na = df_melt[mask]['Diff'].values[0] #A
oth = df_melt[mask]['Diff'].values[1]

df_text = pd.DataFrame({'text' : ['Rest of the world','North Amer
       'x' : [2023,2023], #B
       'y' : [oth,na]}) #C

text = alt.Chart(df_text).mark_text(fontSize=14, align='left').en
    x = 'x',
    y = 'y',
    text = 'text',
    color = alt.condition(alt.datum.text == 'North America', alt.
)
```

How to add line labels

Use `mark_text()` to set the line labels. To label the lines, we place a text annotation next to each line. We could have used the legend to label the lines, but as Cole Nussbaumer Knaflic says in her book, it's clearer to use text annotations for better clarity (Nussbaumer Knaflic 2015). To set the y text position, use the coordinates of the last point in each line. For the x, instead, slightly increase the coordinates of the last point in each line (e.g., 2023 in the example, while the last point is 2021).

Now, combine the textual mark and the original chart, as shown in the following listing:

Listing 3.27 Combining the chart and the text labels

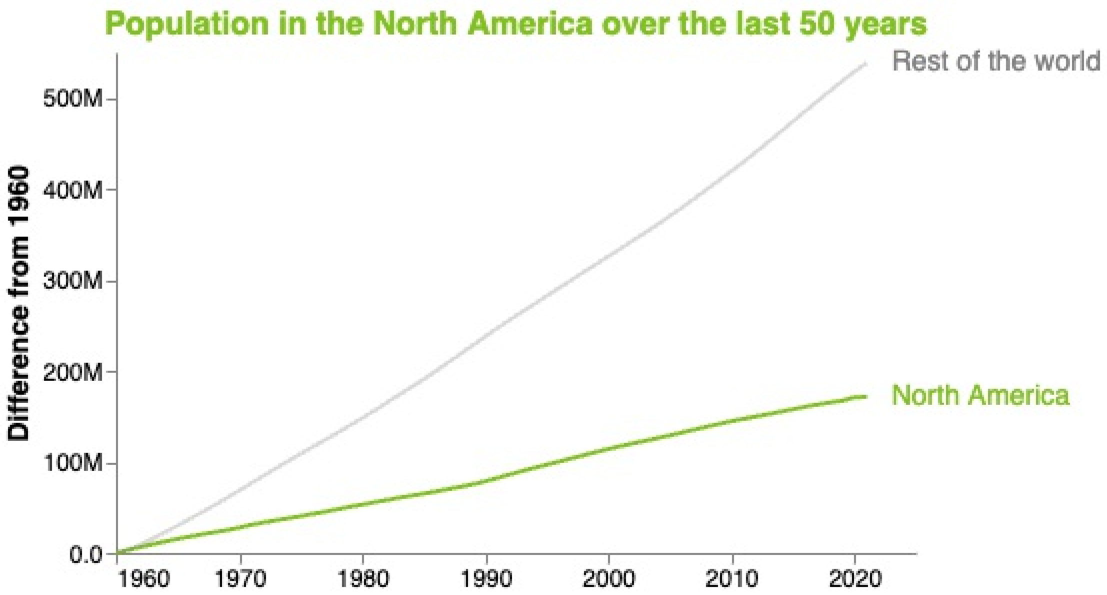
```
total = (chart + text).configure_axis(
    grid=False,
    titleFontSize=14,
    labelFontSize=12
).configure_title(
    fontSize=16,
    color='#80C11E'
).configure_view(
    strokeWidth=0
)

total.show()
```

How to combine the chart and the label text

Use + to combine the two layers. Then use `configure_axis()`, `configure_title()`, and `configure_view()` to set the global chart configurations.

Figure 3.15 illustrates the resulting chart at the end of the information step.

Figure 3.15 The chart at the end of the information step



The next step of the DIKW pyramid is turning information into knowledge. So let us proceed.

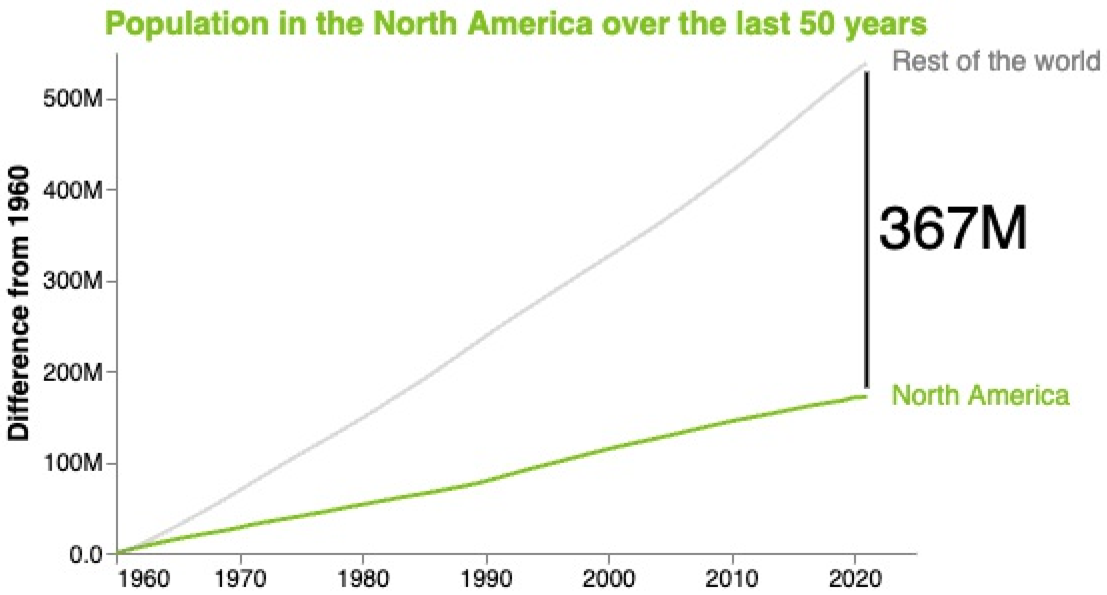
3.3.2 From Information to Knowledge

Turning information into knowledge involves adding context to the chart. We add two types of context: an annotation showing the gap between the two lines and a text explaining the causes of this gap.

Chart Annotation

The objective of the chart annotation is to ease the audience's life by making explicit the gap between the two lines. Figure 3.16 shows the annotation we are going to add to the chart. The annotation comprises two elements, a vertical line, showing the gap between the two lines, and a Big Ass Number (BAN), a big number containing the gap value.

Figure 3.16 The chart with an explicit annotation on the gap between North America and the rest of the world.



To draw the chart annotation, use `mark_line()` to draw the vertical line and `mark_text()` to draw the text. The following listing describes how to render the previous annotations.

Listing 3.28 Adding the annotation

```
offset = 10000000 #A
mask = df_melt['Year'] == 2021
na = df_melt[mask]['Diff'].values[0] #B
oth = df_melt[mask]['Diff'].values[1] #C

df_vline = pd.DataFrame({'y' : [oth - offset,na + offset],
                         'x' : [2021,2021]})

line = alt.Chart(df_vline).mark_line(color='black').encode(
    y = 'y',
    x = 'x'
) #D

df_ann = pd.DataFrame({'text' : ['367M'],
       'x' : [2022],
       'y' : [na + (oth-na)/2]}) #E

ann = alt.Chart(df_ann).mark_text(fontSize=30, align='left').enco
    x = 'x',
    y = 'y',
    text = 'text'
) #F
```

```
total = chart + text + line + ann #G
total = total.configure_axis(
    grid=False,
    titleFontSize=14,
    labelFontSize=12
).configure_title(
    fontSize=16,
    color='#80C11E'
).configure_view(
    strokeWidth=0
)

total.show()
```

How to add the annotation

Use the last data points of the lines of the continents as y coordinates for the vertical line and the BAN.

Text Annotation

Place the text annotation to the left of the graph to prepare your audience for the chart. The text explains the possible causes of the gap between North America and the other continents. The following listing describes how to add a text annotation to the chart.

Listing 3.29 Adding a text annotation

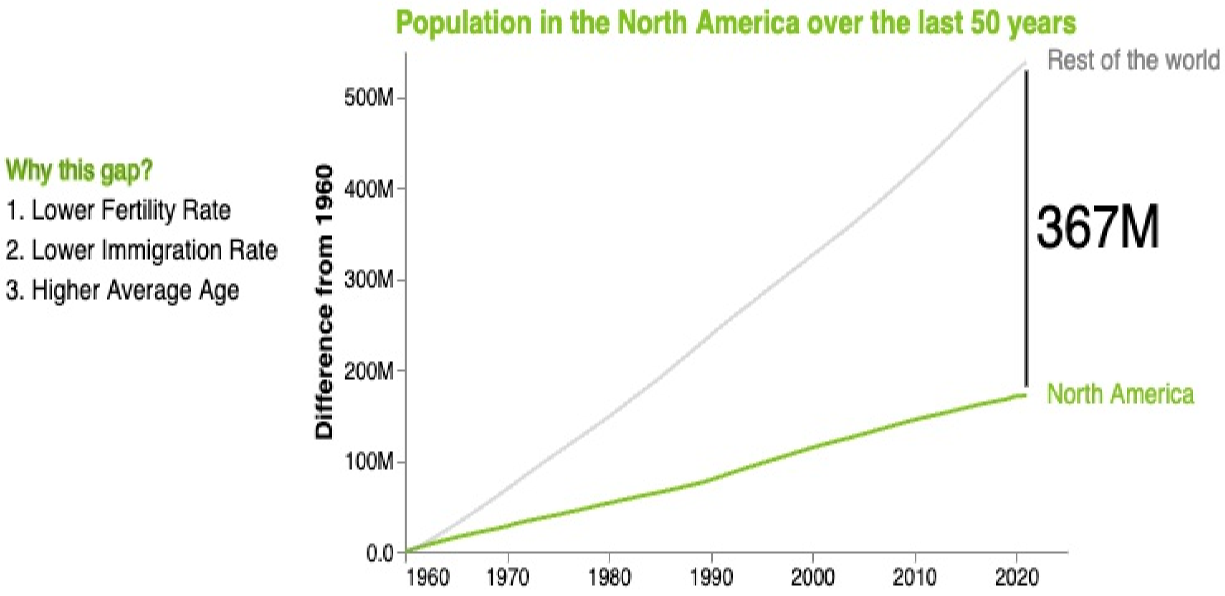
```
df_context = pd.DataFrame({'text' : ['Why this gap?',
                            '1. Lower Fertility Rate',
                            '2. Lower Immigration Rate',
                            '3. Higher Average Age'],
                           'y': [0,1,2,3]})

context = alt.Chart(df_context).mark_text(fontSize=14, align='lef
    y = alt.Y('y:O', axis=None),
    text = 'text',
    stroke = alt.condition(alt.datum.y == 0, alt.value('#80C11E')
    strokeWidth = alt.condition(alt.datum.y == 0, alt.value(1), a
)

total = (context | (chart + text + line + ann)).configure_axis(
    grid=False,
    titleFontSize=14,
    labelFontSize=12
).configure_title(
    fontSize=16,
    color='#80C11E'
).configure_view(
    strokeWidth=0
)

total.show()
```

How to add a text annotation to the chart

Use `mark_text()` to draw the text. Then, combine the text annotation with the remaining chart.

Figure 3.17 shows the resulting chart at the end of the knowledge step.

Figure 3.17 The chart at the end of the knowledge step



The chart is almost ready. Let us proceed with the last step, turning knowledge into wisdom.

3.3.3 From Knowledge to Wisdom

Turning information into wisdom involves adding a call to action to the chart. In the previous chapters, we added a simple call to action in the title, inviting the audience to do something. However, in a real scenario, the call to action must include a proposal of possible next steps. The next step invites the audience to do something concrete thus, the situation depicted in the chart changes. For example, you can add the call to action as an additional chart, which describes the percentage of success of each possible alternative. In our example, we can add a bar chart with possible strategies to improve population growth in North America. The percentages in the example are imaginary and do not reflect reality. We've added them just to show how a call to action works. The following listing shows how to build the call to action as a bar chart and add it to the total chart.

Listing 3.30 Adding a call to action

```
df_cta = pd.DataFrame({
    'Strategy': ['Immigration Development', 'Enhance Family-Frien
    'Population Increase': [20, 30, 15]  # Sample population incr
})

cta = alt.Chart(df_cta).mark_bar(color='#80C11E').encode(
    x='Population Increase:Q',
    y=alt.Y('Strategy:N', sort='-x', title=None),
    tooltip=['Strategy', 'Population Increase']
).properties(
    title='Strategies for population growth in North America',
)

total = alt.vconcat((context | (chart + text + line + ann)), cta,
    grid=False,
    titleFontSize=14,
    labelFontSize=12
).configure_title(
    fontSize=20,
    color='#80C11E',
    offset=20
).configure_view(
    strokeWidth=0
).configure_concat(
    spacing=50
)

total.show()
```

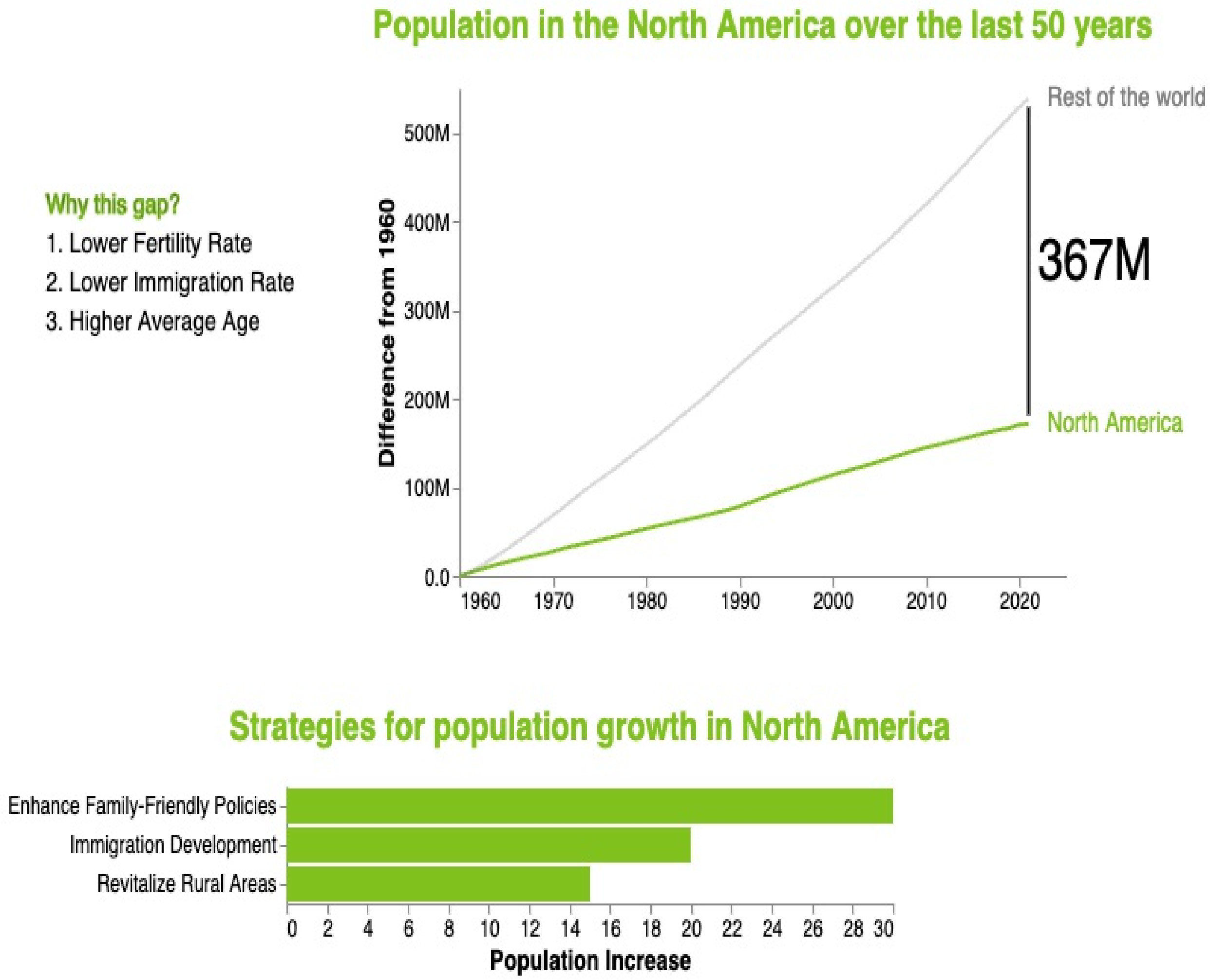
How to add a call to action to the chart

Draw a chart specifying the possible alternatives and their percentage of success.

Figure 3.18 shows the final chart. Following the steps specified by the DIKW, we have transformed the raw chart of Figure 3.11 into a data story.

In this chapter, you have learned the basic concepts of Altair. In the next chapter, you’ll review the basic concepts of Generative AI.

Figure 3.18 The final chart at the end of the wisdom step



3.4 Summary

- Vega and Vega-lite are visualization grammars to define the visual aspects and interactive features of a chart by writing code in JSON format.
- Altair is built at the top of Vega-lite and offers a user-friendly API to build charts. The basic components of an Altair chart are encodings, marks, conditions, compound charts, interactivity, and configurations.

3.5 References

Knaflic, C. N. (2015). Storytelling with data: A data visualization guide for business professionals. John Wiley & Sons.

[1] https://data.worldbank.org/indicator/SP.POP.TOTL

4 Generative AI Tools for Data Storytelling

This chapter covers

- Basic Concepts: Artificial Intelligence, Machine Learning, Deep Learning, and Generative AI
- The Basic Structure of a ChatGPT Prompt
- The Basic Structure of a DALL-E Prompt
- GitHub Copilot

In this chapter, we will review the basic concepts behind Generative AI tools and how to use them in the context of data storytelling. The chapter does not describe the fundamentals of Generative AI. Instead, the first part gives a quick overview of AI, Generative AI, and Generative AI tools. Next, the chapter focuses on three main tools: ChatGPT, DALL-E, and GitHub Copilot, and describes how to incorporate them in data storytelling. Finally, it shows a practical use case, which demonstrates how you can combine the three tools to build data stories quickly.

4.1 Generative AI Tools: on the Giants' Shoulders

Some time ago, I participated in a webinar entitled *Considerations on GPT technology*[1], where panelists discussed the (possible) benefits introduced by Generative AI. During the discussion, one particular intervention by Dr. Michele Monti caught my attention. He drew an interesting analogy between using ChatGPT to assist you while writing and using a satellite navigator on a smartphone to assist you while going to a destination.

In the past, when satellite navigators were not readily available, travelers relied on paper maps to reach a destination. However, with the advent of satellite navigators, the convenience and efficiency they offered made paper maps almost obsolete. Nowadays, very few people would consider using a paper map when they have the option of using a satellite navigator.

A similar analysis could be done for Generative AI tools, especially ChatGPT. Without the emergence of Generative AI tools, you have to write your texts manually to manually write your texts. This could be time-consuming and labor-intensive. As Generative AI continues to advance, it could increasingly become the primary method for writing and generating images.

In my original idea, the book was only supposed to describe how to do data storytelling with Altair and the DIKW Pyramid. It was the editor (whom I sincerely thank), in light of the rapid emergence of generative AI at the time of writing, who suggested also including Generative AI. At first, I only thought about Copilot, but then as I wrote the examples, I made more and more use of other newly available Generative AI tools to complete the data stories. And that's why I decided to also describe in the book how to do data storytelling using Generative AI tools.

You might wonder why you had to wait until Chapter 4 to read about ChatGPT and DALL-E. The real reason is that to apply the Generative AI tools correctly, it is necessary first to understand the context in which you work and then apply them. If we had immediately introduced ChatGPT and DALL-E, the risk would have been to delegate everything to these tools without building a solid foundation. Knowledge of how to do data storytelling in Python would inevitably be superficial. Imagine the concepts described in this book as a house, where the foundations are data storytelling concepts, the various floors are Altair, and the roof and walls are Generative AI. Just as you cannot build a house from the roof and the walls, you cannot start immediately with Generative AI. However, to not leave you hungry for Generative AI in the first chapters, we still wanted to mention GitHub Copilot because, as they say, appetite comes with eating.

This book does not describe how to automate data storytelling but how to use Generative AI tools as assistants in developing our ideas. Before seeing how to use these tools to do data storytelling, let's try to understand what lies behind the concept of Generative AI.

Generative Artificial Intelligence (AI) is a subfield of Artificial Intelligence;

more precisely, a subfield of Deep Learning. Describing the foundations of Generative AI is out of the scope of this book, but a quick overview of it is mandatory to understand the general context before we delve into its use. At the time of writing this book, there is a lot of debate surrounding the use of generative ai tools. On the one hand, some people argue that these tools are dangerous, as they could take over humans in the more or less distant future. On the other hand, other people highlight their benefits, recognizing the level of innovation introduced by these tools under the supervision of a responsible user. In this book, we do not take a specific position, leaving you the responsibility to use these tools consciously.

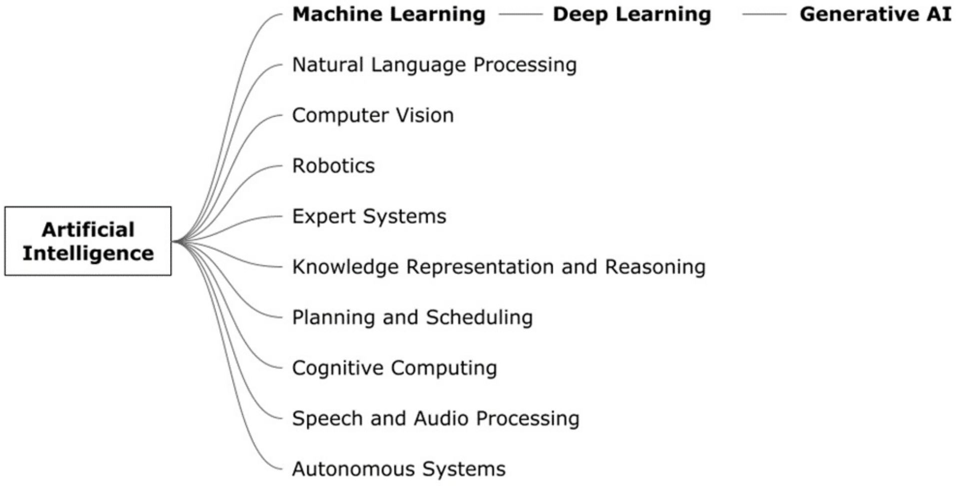
In this book, we'll try to have a conservative and responsible attitude toward Generative AI tools. We will use them as assistants in helping us better define our content, but they will never replace our supervision and ideas. By conservative and responsible attitude, we mean that we will not use the output produced by Generative AI tools as the absolute truth. Instead, we will always check the produced output.

This section gives a quick overview of what Artificial Intelligence, Machine Learning, Deep Learning, and Generative AI are. If you are impatient to get to the application of Generative AI in Data Storytelling, skip this section and jump to the next one directly.

4.1.1 What is Artificial Intelligence?

Artificial Intelligence (AI) is a discipline, like physics and mathematics, aiming to construct intelligent agents capable of independent reasoning, learning, and decision-making. AI focuses on developing theories and methodologies to create agents that can emulate human-like thinking and behavior. Figure 4.1 shows the main subfields of AI.

Figure 4.1 The main subfields of AI with a focus on Generative AI



The figure also highlights the subfield in which Generative AI falls. Generative AI is a subfield of Deep Learning, which is a subfield of Machine Learning, which is a subfield of Artificial Intelligence.

Now that you have learned the main subfields of AI, let’s focus on Machine Learning.

4.1.2 What is Machine Learning?

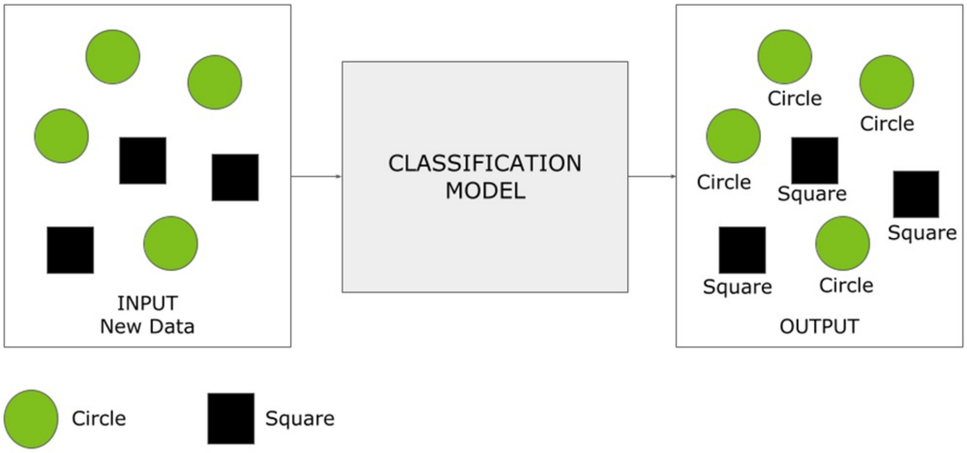
Machine Learning (ML) is a subfield of AI that enables computers to acquire knowledge and improve performance without explicit programming. Different from traditional programming, where you must write the code line by line to provide instructions, ML allows machines to learn from data and make predictions or take actions based on that learned information. This approach empowers machines to discover patterns, relationships, and insights that may not be immediately apparent to human programmers. ML relies on two types of learning: supervised learning and unsupervised learning.

Supervised Learning

In supervised learning there is a preliminary phase, called training, where algorithms learn from labeled data. During training, each sample in the training dataset is paired with its corresponding label or output. The algorithm learns how to map input features to their respective outputs by generalizing from the provided labeled data. During the training phase, the algorithm optimizes its parameters to minimize the discrepancy between the predicted outputs and the true labels. Once the training phase is completed, the algorithm can make predictions or classify new, unseen instances based on the patterns it has learned from the labeled data. Use this technique for tasks such as image classification, sentiment analysis, spam detection, and speech recognition.

Figure 4.2 shows an example of a classification model, which has already been trained with labeled data. As a result of the training process, the model has learned how to discriminate between two classes, circles and squares. When the model receives new data as an input, it predicts the class associated with each sample as an output.

Figure 4.2 An example of a classification model

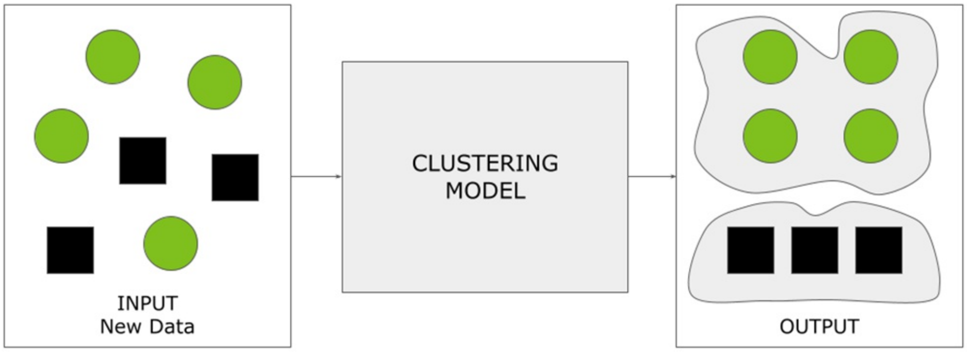


Unsupervised Learning

In unsupervised learning algorithms identify the hidden structure and patterns in unlabeled data. This technique operates on unlabeled data, where no specific outputs or labels are provided during training. Common unsupervised learning techniques include clustering, dimensionality reduction, and anomaly detection.

Figure 4.3 shows an example of a clustering model. The model has not been trained with previous data. Instead, it applies an internal formula to group data. The model receives a dataset of squares and circles as an input and groups them into two clusters, one for each shape.

Figure 4.3 An example of a clustering model



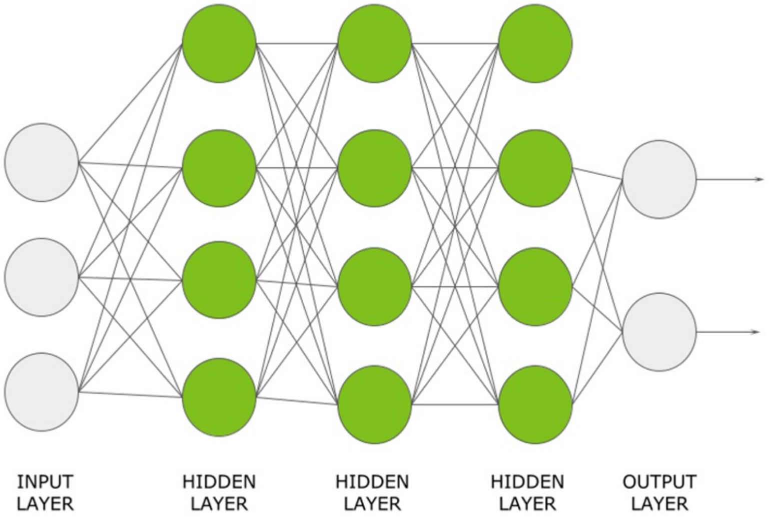
Now that you have learned the main types of ML, let’s move on a step further and focus on Deep Learning.

4.1.3 What is Deep Learning?

Deep Learning (DL) is a subfield of ML that focuses on training artificial neural networks to process and understand complex patterns in data. Inspired by the human brain, DL models consist of multiple hidden layers of interconnected nodes, known as artificial neurons or units. The greater the number of layers, the more complex the data structure the network can recognize. DL is quite resource-intensive because it requires high-performance GPUs and large amounts of storage to train models. You can use DL models to solve tasks such as image and speech recognition, natural language processing, and even game-playing.

Figure 4.3 shows an example of a DL network with one input layer with three nodes, three hidden layers, and two outputs.

Figure 4.4 An example of a DL network



Now that you have seen an overview of DL, let's move on to the next step, defining Generative AI.

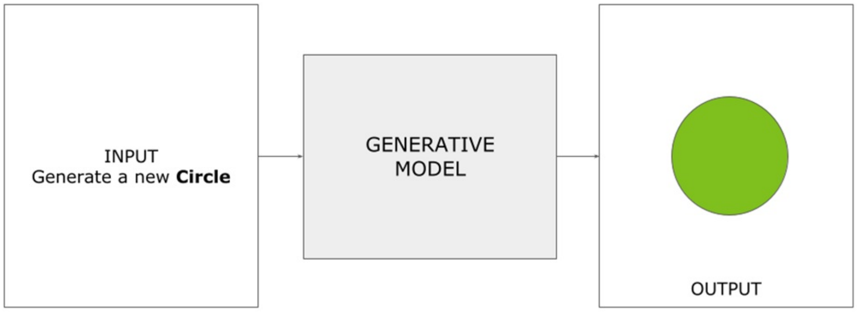
4.1.4 What is Generative AI?

Generative AI is a subfield of DL aimed at creating new content based on what it has learned from existing content. The result of the learning process from existing content is a statistical model (generative model), which is used to generate new content.

Generative models try to understand data distribution. Suppose we have a dataset of pictures of circles and squares. Generative models learn the overall patterns and characteristics of both circles and squares. Then, using this understanding, they generate new pictures that look similar to the ones they have seen.

Figure 4.5 shows an example of a generative model. The model takes a text asking to generate a new circle as an input and produces a new circle as an output.

Figure 4.5 An example of a generative model



A generative model takes a **prompt** as an input. A prompt is a piece of text used to control output quality. It serves as a guide for the generative model to generate responses or outputs that align with a specific desired outcome. The basic prompt is a text asking the tool to perform a specific action. In Chapter 2, *Running Your First Data Story in Altair and GitHub Copilot*, you have written some basic prompts to make Copilot generate the code to build an Altair chart.

Prompt Engineering is a discipline aiming at carefully crafting and refining the prompts to achieve the desired results. The goal of prompt engineering is to elicit responses that are coherent, relevant, and aligned with the desired outcome while minimizing any unintended biases or undesired behavior.

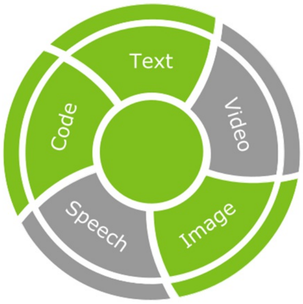
Now that you have learned what Generative AI is, let’s move on to describing the Generative AI tools landscape.

4.1.5 Generative AI Tools Landscape

You can use Generative AI for multiple purposes, including text, image, code, speech generation, and much more. For each specific task, you can use a different tool. In all cases, you start from a text, and the tool will convert it into the desired output (text, image, code, and so on) based on its specific nature. Please notice that in most cases, you must pay a fee to use Generative AI tools. However, some Generative AI tools have free-of-charge versions with downgraded capabilities.

Figure 4.6 shows the main application fields of Generative AI tools (Madhvani 2023). If not properly used, these tools may produce wrong or biased information. Thus, you must carefully monitor and adjust the generated output to ensure accuracy and fairness.

Figure 4.6 The main application fields of Generative AI tools. In this book we will focus on Code, Text, and Image.



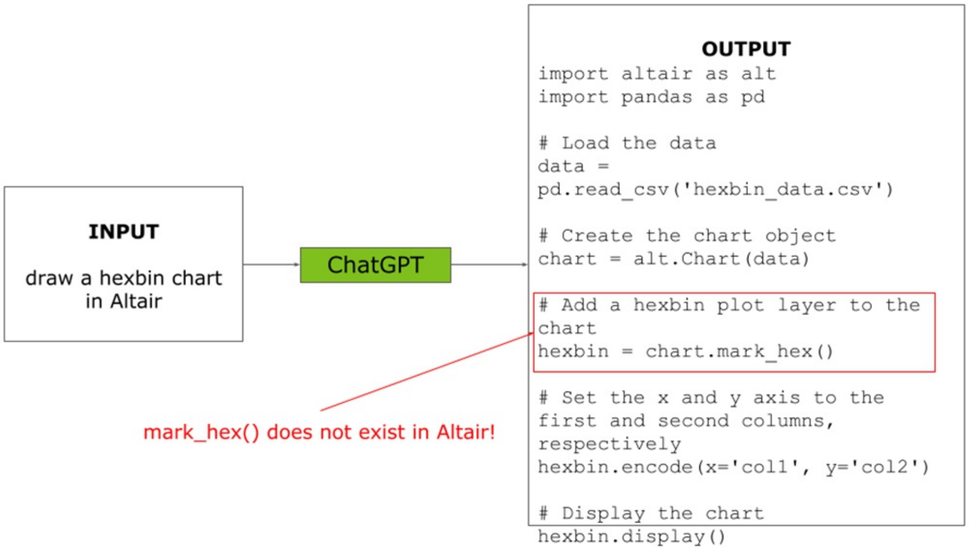
The figure also highlights the application fields used in this book:

- *Text-to-text tools*: use them to generate text, such as content creation, chatbots, virtual assistants, creative writing, and automating repetitive tasks like writing emails or generating reports. Examples of Text-to-Text tools are Open AI ChatGPT and Google Bard. In this book, we'll focus on ChatGPT.
- *Text-to-image tools*: use them to generate visual representations based on textual input. In this book we'll use Open AI DALL-E. If you signed up to DALL-E before April 6th, 2023, you will be granted free credits. New users must buy credits for a small fee to use the service.
- *Text-to-image tools*: use them to generate code. In this book, we'll focus on GitHub Copilot, which requires an active subscription.

You may ask why you should pay to use a tool. The main answer is that no quality Generative AI tool is available without paying; free tools are either not up to par, or are very limited until you pay or subscribe. Fortunately, the fees are fairly small.

We could have used ChatGPT to generate code, but from personal experience, I noticed that ChatGPT sometimes may invent a Python method or function, as shown in Figure 4.7.

Figure 4.7 An example of a bad output produced by ChatGPT to generate code



draw a hexbin chart in Altair

```
import altair as alt
import pandas as pd

# Load the data
data =
pd.read_csv('hexbin_data.csv')

# Create the chart object
chart = alt.Chart(data)

# Add a hexbin plot layer to the
chart
hexbin = chart.mark_hex()

# Set the x and y axis to the
first and second columns,
respectively
hexbin.encode(x='col1', y='col2')

# Display the chart
hexbin.display()
```

We asked ChatGPT to generate a hexbin chart, and ChatGPT used the `mark_hex()` method, which is not currently implemented in Altair.

We have chosen specific tools to generate text, images, and code for data storytelling. However, you can easily adapt the described concepts to other tools with minimal effort.

Now that you have quickly defined the context of Generative AI tools, we can move on to the next step, using Generative AI tools for data storytelling. We will start with ChatGPT, describe DALL-E, and finally, GitHub Copilot.

In all cases, it's crucial to give it accurate and concise instructions to ensure the tool generates the correct output. Write specific and unambiguous instructions, using simple language and avoiding complex terminology. Also, write consistent instructions regarding syntax, language, and format. Inconsistent instructions can confuse the tool, leading to incorrect outputs.

4.2 The Basic Structure of a ChatGPT Prompt

To interact with ChatGPT, you must write an input text (prompt) that defines the instructions to be performed. You can use either the ChatGPT web interface or the Open AI API, as described in Appendix A.

Different ways exist to structure a prompt for ChatGPT. In this book, we consider a prompt to be composed of three main consecutive texts:

- *Telling ChatGPT to act in a specific [role] - for example "You are an examiner looking at high school students' English papers."*
- *Telling ChapGPT to tailor its outputs to an intended [audience] - for example "Explain your gradings in a way that can be understood by high schoolers."*
- *Define the [task] - for example "Grade this text and explain your reasoning.*"

Although we have tested this prompt structure, sometimes they do not produce the desired output, so you can refine the prompt using different iterations.

In the remainder of this section, we'll describe all the parts of a prompt separately. Although the structure of a prompt should be [Role] [Audience] [Task], we'll start by describing first the task to perform, to show how the produced prompt changes while varying the roles and the audience.

We'll show some examples of produced texts. If you try your own tool, you'll obtain different results since ChatGPT generates new content every time you use it.

4.2.1 Defining the Task

A task is a specific objective ChatGPT must fulfill. A task includes clear instructions to guide ChatGPT in generating responses relevant to the given task. Examples of tasks are answering questions, generating creative content, translating language, and more. In the context of data storytelling, we'll focus on the following tasks:

- Generating a title

- Generating a subtitle
- Summarizing a text
- Translating a text
- Proposing ideas.

We’ll analyze each task in the next chapters in more detail.

Imagine a scenario where you must engage an audience striving for success in their personal and professional lives. You have prepared the following text that shows the contrast between appearing strong and perfect and showing one's fragility:

In a world that glorifies strength and perfection, remember that fragility is the essence of our humanity. Our struggles and imperfections make us human, yet we often conceal them out of fear and shame. However, those who embrace their weaknesses with courage and authenticity cultivate remarkable resilience. Those who fear their fragility risk becoming rigid, trapped in a facade of unattainable perfection. For it is in our vulnerabilities that we grow and truly live.[2]

Suppose that your task is to generate a title for the previous text. Use ChatGPT and formulate the following prompt: *Generate the title for the text in angular brackets <text>*. Replace the text word with the previous text about fragility and vulnerabilities. Examples of generating titles are:

- Embracing Fragility: The Path to Remarkable Resilience and Authentic Living
- Fragility as the Gateway to Authenticity and Resilience in a Perfect-Obsessed World
- *Fragility: Embracing Imperfections for Authentic Resilience and Growth.*

All the generated titles are general purpose. The next step is adding a role to the prompt to make ChatGPT generate a more specific output.

4.2.2 Acting as a Role

There are at least the following goals when conveying a message:

- *Entertainment*: The focus is on providing enjoyment, amusement, or relaxation. You may want to make people laugh or evoke emotions.
- *Education*: The focus is on providing knowledge, insights, or skills to the audience to broaden the audience's knowledge base and encourage learning.
- *Information*: The focus is on keeping the public informed to the audience about a specific topic, such as news updates or current events.
- *Inspiration*: The focus is to motivate the audience, such as sparking creativity or encouraging personal growth. Inspirational messages often share success stories or showcase role models.
- *Invitation to action*: The focus is to motivate the audience to engage in a specific behavior or event.
- *Relationship Building*: The focus is to foster connections with the audience through personal anecdotes and shared experiences. The message aims to build trust, strengthen existing relationships, or cultivate a sense of community.

Depending on the goal, you’ll cover a different role, which uses specific storytelling techniques. In some cases, in the same situation, you may cover different roles, but generally, the role depends on the context where you will present.

Consider again the scenario where you must engage an audience striving for success in their personal and professional lives. Use ChatGPT to adapt the text to your role. Use the following syntax to formulate the prompt for ChatGPT: *Act as a [role]. Generate the title for the text in angular brackets <text>*.

The table below shows the produced outputs for all the described roles.

Table 4.1 The result of the prompt for all roles

Role	Output
Entertainer	Unmasking Our Humanity: Embracing Fragility and Thriving in Imperfection
Educator	Nurturing Resilience: Embracing Fragility and

	Embracing Imperfection for Personal Growth and Well-being
Informer	The Power of Embracing Fragility: Celebrating Imperfections and Cultivating Resilience
Inspirer	Embrace Your Fragile Strength: Unleashing Resilience and Embracing Imperfections for a Fulfilling Life
Invitor to Action	Embrace Your Fragile Humanity: Rise Above Fear and Embrace Imperfection to Unleash Your Resilient Potential.
Relationship Builder	Embracing Fragility Together: Fostering Authentic Connections and Building Resilient Relationships

All the titles have the same structure ([text] [colons] [text]) but their goal changes based on the role.

Now that you have learned how to influence the output using a different role let’s move on to how to influence the output based on the different target audiences.

4.2.3 Tailored to an Audience

Depending on the context where you will tell your data story, your audience changes. For example, you could have an audience of stakeholders, businessmen, or even children or philosophers. More generally, there are the following types of audiences:

- *General Audience*: A broad group of people with varied interests, backgrounds, and demographics.
- *Target Audience*: A specific group with well-defined interests, backgrounds, and other factors.
- *Niche Audience*: A relatively small group of people with specific interests, preferences, or needs, such as enthusiasts of specific hobbies

or professionals in specific fields.
- *Mass Audience*: A large and diverse group of people that consume content on a wide scale.

Consider again the scenario where you must engage an audience striving for success in their personal and professional lives. Use ChatGPT to adapt the text to your audience. Use the following syntax to formulate the prompt for ChatGPT: *Act as a [role] tailored to [the audience]. Set the title for the text in angular brackets <text>.*

The table below shows the outputs for all the described audience types for the relationship builder role.

Table 4.2 The result of the prompt for all the audience types

Audience	Output
General Audience	Embracing Fragility: The Path to Resilience and Authentic Living
Target Audience of businessmen	Embracing Vulnerability: The Power of Authenticity and Resilience in Business Relationships
Niche Audience of football enthusiasts	The Beautiful Game: Celebrating Fragility as the Heart of Football Passion
Mass Audience of young people	Embracing Your Imperfections: Unleashing Your Inner Strength as a Young Person

All the titles have the same structure ([text] [colons] [text]) but their meaning is adapted to the specific audience. Consider again the title generated specifying only the role: *Embracing Fragility Together: Fostering Authentic Connections and Building Resilient Relationships*. This title is similar to that generated for a general audience. When you add a specific audience, the title is tailored to it.

Now that you have learned the basic concepts to define a prompt for ChatGPT let’s move on to the next step, defining a prompt for DALL-E.

4.3 The Basic Structure of a DALL-E Prompt

To interact with DALL-E, you must write a prompt specifying the instructions. You can use either the DALL-E web interface or the Open AI API, as described in Appendix A.

Different ways exist to structure a prompt for DALL-E. In this book, we consider a prompt composed of:

- *[Subject]*
- *[Style]*.

In the remainder of this section, we'll describe all the parts of a prompt separately by starting with the subject.

Remember that if the output is not satisfying, you can refine it or use the *Edit Image* tool, as described later in this section.

4.3.1 Subject

The subject of an image is the main focus that captures the viewer's attention. Examples of subjects include persons, objects, or scenes. When defining the subject of your image, be as specific as possible. A generic subject will generate a generic image. For each input, DALL-E generates four output images. In the remainder of the section, we'll show some examples of produced images. If you try your own the tool, you'll obtain different examples.

Figure 4.8 shows the output produced by DALL-E for the prompt *A person*. Since the input is very generic, also the output is very generic. The four generated images are different from each other.

Figure 4.8 The output for the prompt *A person*

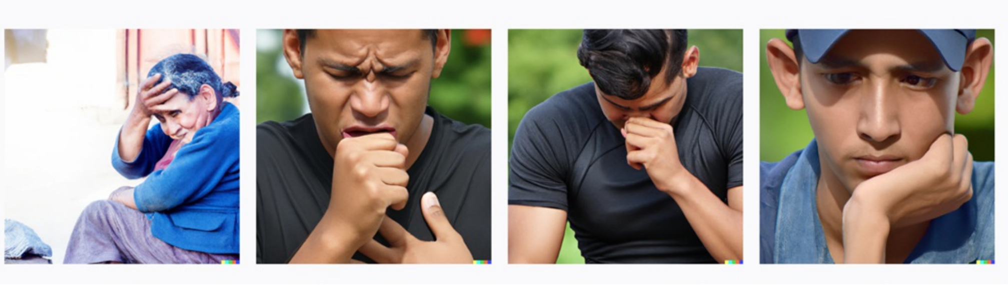
To improve the focus of the image, you can add the following elements:

- An adjective
- An emotion
- Context
- Color.

Let’s start by adding an adjective that qualifies the subject. We will give a specific example of generating images related to *poor people*. I chose this theme to show the potential of inserting images into a data story to arouse emotions in the audience. In fact, data storytelling also gives voice to the people behind data.

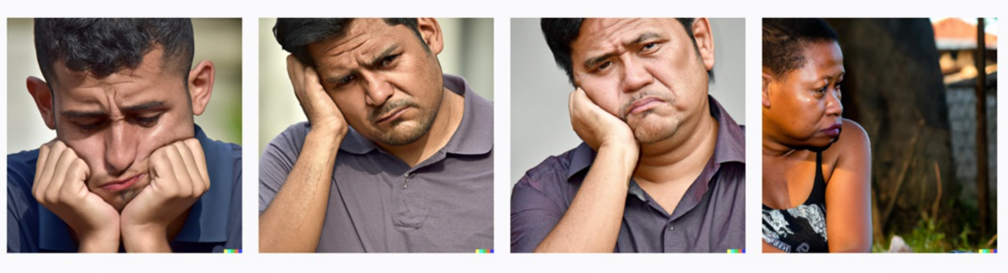
Figure 4.9 shows the outputs for the prompt *A poor person*, where *poor* is the qualifying adjective. All the produced images have a hand on the head and do not show a poor person. Only the first image describes a poor person. However, with respect to the generic output, we have obtained a better result.

Figure 4.9 The output for the prompt *A poor person*



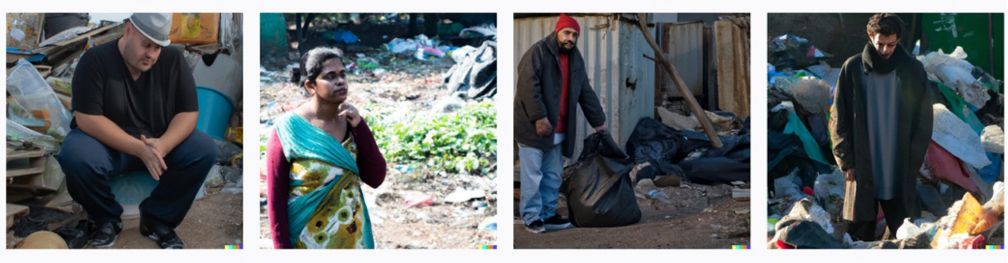
To be more specific, add emotion, which describes the person’s mood. Figure 4.10 shows the output produced by the prompt *a sad, poor person*.

Figure 4.10 The output for the prompt *A sad, poor person*



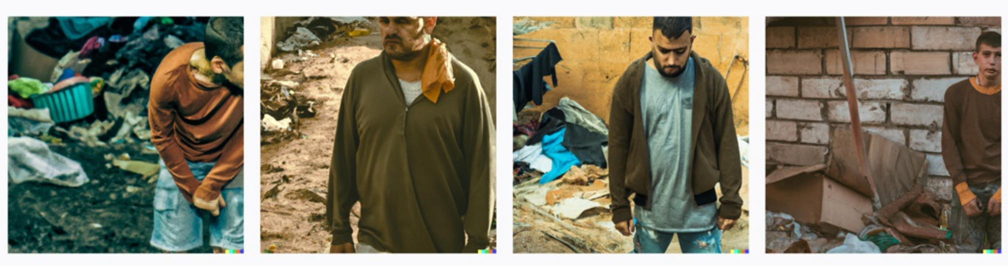
The figure still needs to be improved. Let's add a possible background, which sets the context for the subject. Figure 4.11 shows the output produced by the prompt *a sad, poor person, with rubbish in the background*.

Figure 4.11 The output for the prompt A sad, poor person, with rubbish in the background



Finally, let's add color. Figure 4.12 shows the output for the prompt *a sad, poor person, wearing a torn brown shirt with rubbish in the background*.

Figure 4.12 The output for the prompt A sad, poor person wearing a torn brown shirt with rubbish in the background



The output is now satisfying so that we can proceed with the next step, i.e., adding style.

4.3.2 Style

Style defines the type of art, such as photographs, digital art, pencil drawing, and so on. In addition, you can specify a particular artist's style, such as Van Gogh or Caravaggio. You just have to indulge your imagination!

Figure 4.13 shows the output for the prompt *a sad, poor person, with rubbish in the background, digital art,* and Figure 4.14 for the prompt *a sad, poor person, with rubbish in the background, black and white photograph*.

Figure 4.13 The output for the prompt A sad, poor person wearing a torn brown shirt with rubbish in the background, digital art

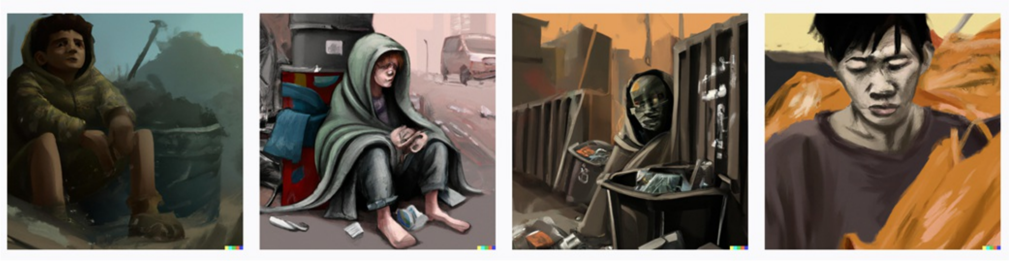
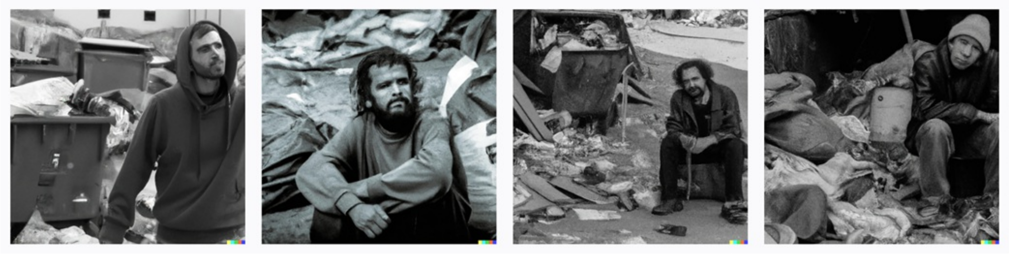


Figure 4.14 The output for the prompt A sad, poor person wearing a torn brown shirt with rubbish in the background, black and white photograph

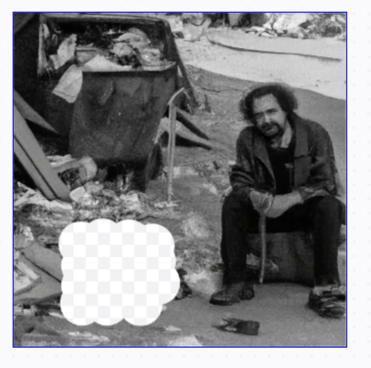


The prompt structure described in this chapter is only an entry point to draw images using DALL-E. You can also define the environment (indoor, outdoor), the day hour, the angle, and more. You can find more information in The DALLE 2 Prompt Book.

4.3.3 The Edit Image Tool

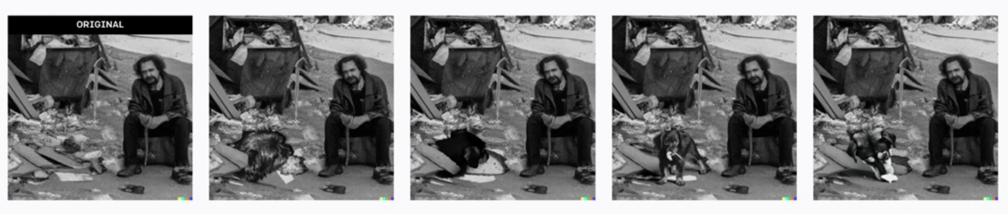
If you are not completely satisfied with a generated image, you can modify it using the Edit image tool provided by the DALL-E interface. To access the Editing image tool, select the image, and click *Edit*. You can erase a subpart of the image (as shown in Figure 4.15), add a new generation frame or upload an image.

Figure 4.15 An example of usage of the eraser



Then, you can write a new prompt for the selected part. Let’s add a dog to the image of Figure 4.15 using the following prompt: *a small black dog eating a piece of bread, a black and white photograph*. Figure 4.16 shows the produced output.

Figure 4.16 The output of the modified image using the eraser



Now that you have learned the basic concepts to generate an image with DALL-E, let’s move on to the next step, using Copilot to build the components of an Altair Chart.

4.4 Using Copilot to Build the Components of an Altair Chart

When writing instructions for Copilot, split them into single steps. Also, provide the context around what you want Copilot to do and provide specific details on the requirements of the code you are writing. Finally, remember that Copilot is a tool designed to assist developers, not replace them. Copilot is powerful, but it can’t replace you as a developer. Always use your expertise to evaluate the code suggestions provided by Copilot and ensure they are appropriate for your specific needs.

In this section, you’ll see how to write the instructions to make Copilot build an Altair chart. We’ll focus on prerequisites, marks, encodings, conditions, compound charts, and interactivity.

4.4.1 Prerequisites

Before drawing a chart, ask Copilot to import the required libraries and load the dataset as a Pandas DataFrame, as shown in Listing 4.1.

Listing 4.1 Importing the required libraries and loading the dataset

```
# Import the required libraries
# Load the file '/path/to/data.csv' into a pandas DataFrame
```

Prerequisites

The first instruction specifies importing the required libraries. Copilot will calculate which libraries automatically, based on the subsequent instructions. The second instruction specifies the exact path to the dataset to load.

4.4.2 Marks

To define a mark, simply instruct Copilot to draw it in Altair, as shown in Listing 4.2.

Listing 4.2 Defining marks

```
# Draw a <type of mark> named chart in Altair
```

Draw a mark type

Use the keywords Draw and Altair to specify that you want to build a chart in Altair. Replace `<type of mark>` with the type of mark you want to build. For example, write `bar chart` if you want to build a bar chart.

4.4.3 Encodings

To set an encoding, add the keyword `with:` to the instruction defined in Listing 3.17. In addition, add a bullet point for each encoding channel you want to define, as specified in Listing 4.3.

Listing 4.3 Setting encodings

```
# Draw a bar chart named bar in Altair with:
# * The category column as the x-axis
# * The value column as the y-axis
# * The country column as the color.
```

Encodings

After defining the mark, use the keyword `with:` followed by a list of encoding channels. For each encoding channel, specify the column of the dataset to use and the channel.

4.4.4 Conditions

To set a condition related to a specific encoding channel, use the instructions specified in Listing 4.4.

Listing 4.4 Defining conditions

```
# * The country column as the color. Set the color to:
#   - 'red' for 'IT'
#   - 'green' for 'FR'
```

Conditions

Append the condition to the selected encoding channel. Use a list to specify the content of the conditions. The example sets the color to red if the country is IT and green if the county is FR.

4.4.5 Compound Charts

Build each chart separately. Then, to build a compound chart, write a specific instruction, as shown in Listing 4.5. Ensure to give each chart a different name.

Listing 4.5 Building compound charts

```
# Build a compound chart named chart with the line and bar charts
```

Compound charts

Specify the type of combination you want to draw. The example aligns the charts vertically.

4.4.6 Interactivity

To make the chart interactive, write the simple instruction in Listing 4.6.

Listing 4.6 Making the chart interactive

```
# Make the chart interactive
```

Interactivity

Write simple and clear instructions to make the chart interactive.

For a complete example in Copilot, refer to the GitHub repository of the book, section 04/copilot.

Now that you have learned how to write the basic instructions to make Copilot build a chart in Altair let’s move to a practical case study that combines Altair and Copilot.

4.5 Case Study: Our Training Team

Let’s imagine you work in a sports company. You are training a team of young athletes in various disciplines. For each discipline, you have noted the world record and have recorded the best time achieved by your team for the sake of comparison. Unfortunately, your company has limited investment funds available. Your boss asks you to understand which disciplines are worth continuing to train in, to hope to achieve good results in the upcoming competitions. You can find the code of this example in the GitHub repository of the book, under CaseStudies/competitions. The directory contains different Python scripts, one for each step described in the remainder of this section.

Table 4.3 shows, for each discipline, the record and the best time achieved by your team.

Table 4.3 The dataset of the case study shows the record to beat, the record holder, the record time, and our best time for each training type

Training Type	Record to Beat	Record Holder	Record Time (Seconds)	Our Best Time
Sprinting	100m	Usain Bolt	9.58	15.07
Long-distance running	Marathon	Eliud Kipchoge	7299	13261.76
Swimming	100m freestyle	César Cielo	46.91	88.01
Cycling	Individual time trial (ITT)	Rohan Dennis	1963.49	2795.22
Rowing	2000m	Josh Dunkley-Smith	335.8	462.33

In our case, the objective is to display the difference between our time and the record for each training type and then choose in which training types we can continue investing.

We'll use the DIKW pyramid to transform our dataset into wisdom. Let's start with the first step of the pyramid, turning data into information.

4.5.1 Turning Data into Information

The simplest graph to utilize for this purpose is a stacked bar chart, which shows the two columns as stacked series. Let’s use GitHub Copilot to generate this simple chart.

Start by writing the list of comments to generate the chart, as shown in Listing 4.7.

Listing 4.7 Generating the stacked bar chart

```
# Import the required libraries
# Load '../data/competitions.csv' as pandas dataframe
# Select only the columns Training Type, Record Time (Seconds), O
# Use melt to transform the dataframe into a long format, with th
# Draw a bar chart named chart in Altair with:
# * The Training Type column as the x-axis
# * The value column as the y-axis
# * The variable column as the color
# Save the chart as 'competitions.html'
```

Write the text to generate the stacked bar chart

Describe the sequence of operations to load, manipulate and plot the dataset.

Copilot will generate different alternatives of codes. Select the code shown in Listing 4.8:

Listing 4.8 The stacked bar chart generated by Copilot

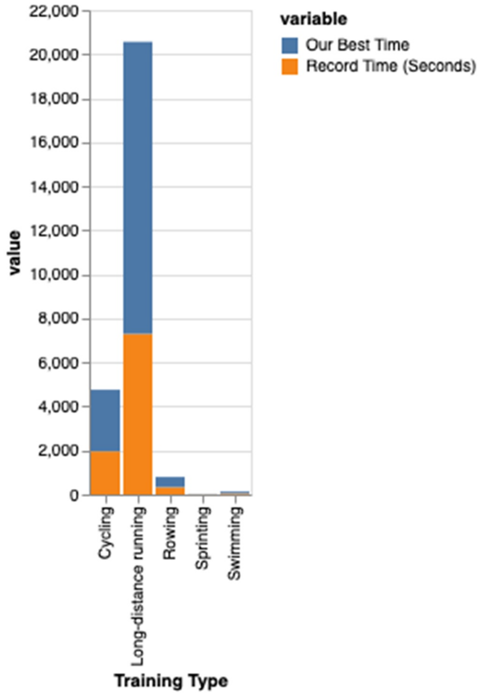
```
import pandas as pd
import altair as alt

df = pd.read_csv('data/competitions.csv')
df = df[['Training Type', 'Record Time (Seconds)', 'Our Best Time
df = pd.melt(df, id_vars='Training Type')

chart = alt.Chart(df).mark_bar().encode(
    x='Training Type',
    y='value',
    color='variable'
)

chart.save('competitions.html')
```

Copilot generates the code to plot the stacked bar chart in Altair

First, load, and manipulate the dataset. Then, use `mark_bar()` to draw the stacked bar chart.

Figure 4.17 shows the resulting bar chart. In addition to being very narrow, the generated chart presents a fundamental problem: we are unable to determine from the chart which disciplines we should focus on. This can be attributed to a lack of clarity in the chart, which makes it difficult for us to interpret the data effectively. As a result, we need to revise the chart design to convey information easily.

Figure 4.17 The stacked bar chart generated by Copilot



In general, when we want to compare two variables across multiple categories, it can be useful to calculate the percentage difference or the percentage increase of one variable relative to the other for each category and then show it. This approach allows us to compare the two variables and identify any patterns or trends within the data. To generate this type of chart, we can once again make use of Copilot, which can assist us in writing the necessary code. Write the list of instructions specified in Listing 4.9.

Listing 4.9 Generating the bar chart with percentage difference

```
# Import the required libraries
# Load '../data/competitions.csv' as pandas dataframe
# Select only the columns Training Type, Record Time (Seconds), O
# Calculate the percentage difference between columns Our Best Ti
# Draw a bar chart with the following encodings:
# * The Percentage Difference on the y-axis with the following pr
#   - The domain of the Y scale to [0,100]
# * Training Type on the x-axis with the following properties:
#   - The values sorted in descending order (-y)
# Set the following properties of the bar chart:
# * width to 300 pixels
# Save the chart as 'competitions.html'
```

Write the text to generate the bar chart

Describe the sequence of operations to load, manipulate and plot the dataset. Also, specify to sort data in descending order and set the domain of the Y scale to [0,100] to make the chart clearer. Since the chart is more complex than the previous one, add the instruction `with the following encodings` when you ask Copilot to draw the chart. Do the same thing for mark properties. Also, note the nested lists to add specific property values to encoding channels.

Copilot will generate different alternatives of codes. Select the code shown in Listing 4.10:

Listing 4.10 The bar chart generated by Copilot

```
import pandas as pd
import altair as alt

df = pd.read_csv('data/competitions.csv')
df = df[['Training Type', 'Record Time (Seconds)', 'Our Best Time
df['Percentage Difference'] = (df['Our Best Time'] - df['Record T

chart = alt.Chart(df).mark_bar().encode(
    x=alt.X('Training Type', sort='-y'),
    y=alt.Y('Percentage Difference', scale=alt.Scale(domain=[0,10
).properties(
    width=300
)
```

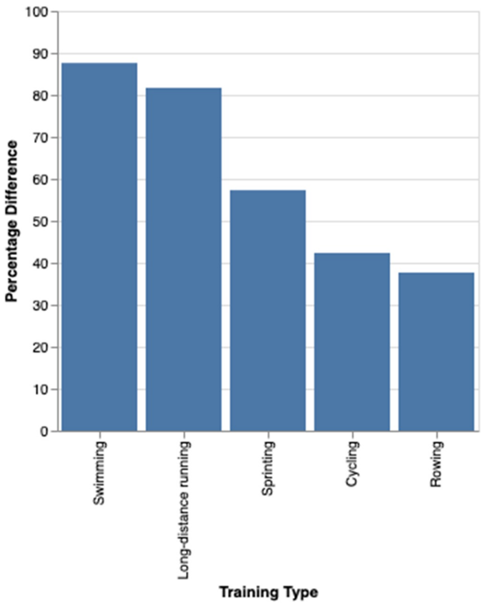
```
chart.save('competitions.html')
```

Copilot generates the code to plot the bar chart in Altair

First, load, and manipulate the dataset. Then, add a column to the dataset, named ‘Percentage Difference’, containing the percentage difference. Finally, use `mark_bar()` to draw the stacked bar chart.

Figure 4.18 shows the resulting chart.

Figure 4.18 The bar chart with percentage difference generated by Copilot



The chart highlights that the sports with the greatest percentage difference between the "Our Best Time" and the "Record Time" are Swimming and Long Distance Running. However, the chart may be misleading because it is not intuitive because the largest bars are in fact the worst performers. To improve the readability and intuitiveness of the chart and better convey the actual performance of athletes in each sport, we can calculate the percentage improvement, as shown in Listing 4.11.

Listing 4.11 Calculating the percentage improvement

```
df['Percentage Improvement'] = 100 -
```

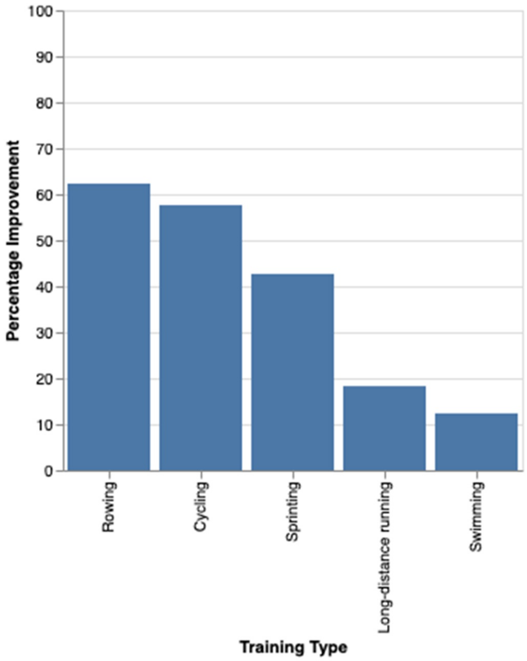
```
(df['Our Best Time'] - df['Record Time (Seconds)']) / 100
```

Percentage improvement

Calculate the percentage similarity 100 - percentage difference.

The percentage improvement represents the proportion of improvement in the performance of "Our Best Time" compared to the "Record Time". A higher percent improvement indicates that the athlete's performance is closer to or better than the record holder's, while a lower percent improvement indicates that the athlete's performance is further from the record holder's performance. This is more intuitive as the larger bars now indicate the best performance. Figure 4.19 shows the percentage improvement. Rowing and cycling are the sports with better performances thus we can choose those sports for further investments.

Figure 4.19 The bar chart with percentage improvement



Now that we have extracted information from the data, we can move to the next step of the DIKW pyramid, which is turning information into knowledge.

4.5.2 Turning Information into Knowledge

This step involves adding context to the information. In our case, the context may involve three aspects:

- Highlighting the two sports where we want to invest
- Adding an image that enforces the top two sports
- Adding a baseline, for example, at 50% of percentage improvement, showing why we choose those sports

Let's start from the first point, highlighting the two sports in which we want to invest.

Highlighting the top two sports

Within the previously generated code, add the comment shown in Listing 4.12, immediately after the y channel to generate the code using Copilot.

Listing 4.12 Highlighting the sports that perform better

```
# Add the color encoding. Set the color to:
    # - #003049 if the Percentage Improvement is greater than 50,
    # - lightgray otherwise
```

Highlight sports

Describe how to highlight the sports that perform better.

Listing 4.13 shows the generated code.

Listing 4.13 The generated code to highlight the sport performing better

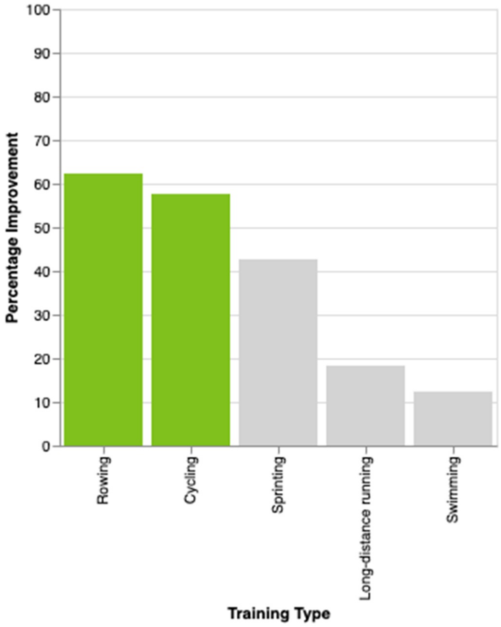
```
color=alt.condition(
        alt.datum['Percentage Improvement'] > 50,
        alt.value('#003049'),
        alt.value('lightgray')
    )
```

Highlight sports

The code generated by Copilot uses a conditional statement to select the color to apply to each bar.

Figure 4.20 shows the resulting chart.

Figure 4.20 The bar chart with the top two sports highlighted



Now that we have highlighted the top two sports, we can move on to add images to the chart.

Adding enforcement images

To generate an image for each of the top two sports, provide DALL-E with the following prompt: an enthusiastic athlete practicing [sport], in a white background, cartoon. Figures 4.21 and 4.22 show the produced outputs for rowing and cycling, respectively.

Figure 4.21 The bar chart with the top two sports highlighted

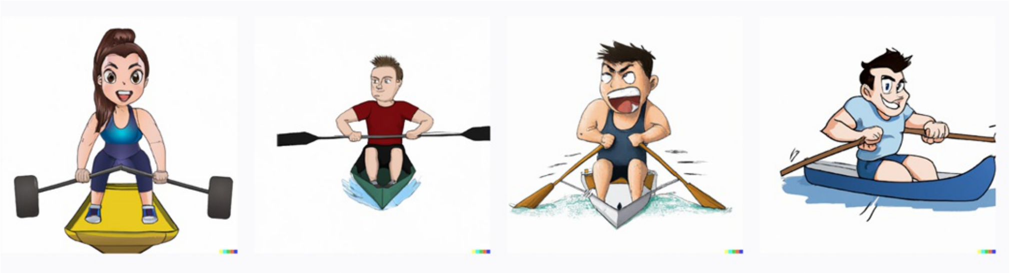
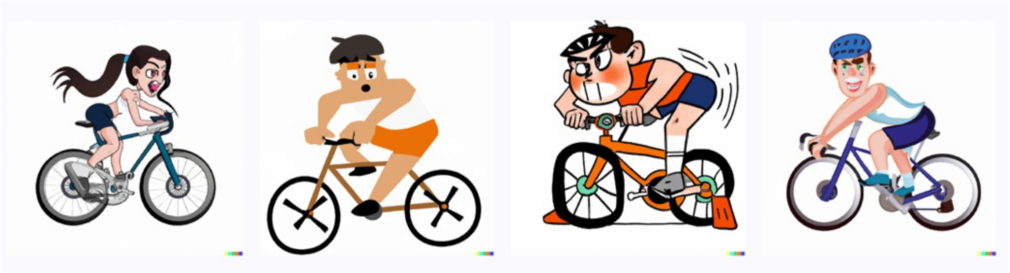


Figure 4.22 The bar chart with the top two sports highlighted



We choose image n.2 in Figure 4.21 and image n.1 in Figure 4.22. If you find some imperfections in an image, you can improve it by using the editing tool. To include an image in an Altair chart, you must save it into a remote repository, such as GitHub and provide Altair with the remote URL.

Now, let’s ask Copilot to write the code to add the images to the chart. Listing 4.14 describes the instructions for Copilot.

Listing 4.14 Adding images to the chart

```
# Add a new column to df called 'url' with the following value:
# * 'https://[..]/cycling.png' for Training Type = 'Cycling'
# * 'https://[..]/rowing.png' for Training Type = 'Rowing'
# * '' for all other Training Types
# Add the following image to the chart:
# * The image is a 35x35 pixel image
# * The image is located at x='Training Type', y='Percentage Impr
```

How to add an image

First add a column to the DataFrame with the URL to the image. Then specify how to add images to the chart.

Listing 4.15 shows the produced output.

Listing 4.15 The code for adding images

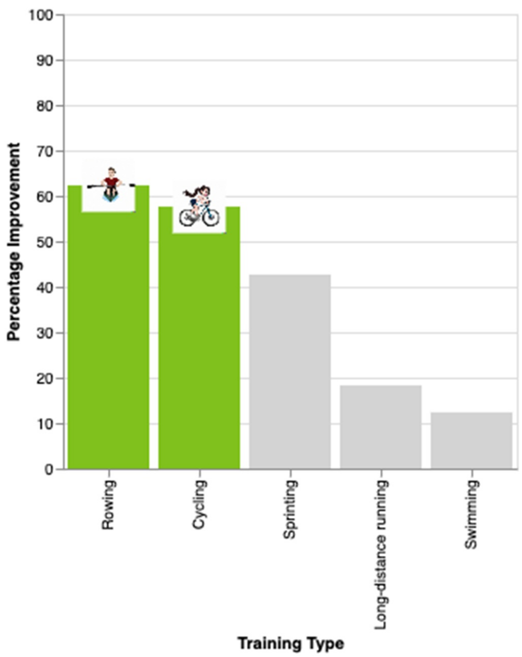
```
df['url'] = ''
df.loc[df['Training Type'] == 'Cycling', 'url'] = 'https://[..]/c
df.loc[df['Training Type'] == 'Rowing', 'url'] = 'https://[..]/ro
chart = chart + alt.Chart(df).mark_image(width=35, height=35).enc
    x=alt.X('Training Type', sort='-y'),
    y=alt.Y('Percentage Improvement'),
    url='url'
)
```

Add an image

First Copilot suggests to manipulate the DataFrame by adding a new column. Next, Copilot suggests to use the `mark_image()` mark.

The next step involves adding a baseline at 50% of percentage improvement. Listing 4.15 describes the instructions for Copilot. Figure 4.23 shows the produced output.

Figure 4.23 The bar chart after adding images



Now that you have added the images for the top two charts, we can move on to the next step, generating a baseline.

Adding a baseline

Adding a baseline means adding a horizontal red line to the chart for a comparison. For example, we can set the value of the horizontal line to 50, which corresponds to 50%. Listing 4.16 shows the instructions for Copilot.

Listing 4.16 Generating a baseline

```
# Add a horizontal red line to the chart at y=50
# Add the line to the chart
```

Generate the baseline

Describe how to generate the baseline at 50% of the percentage improvement.

Listing 4.17 shows the generated code.

Listing 4.17 The generated code to generate a baseline

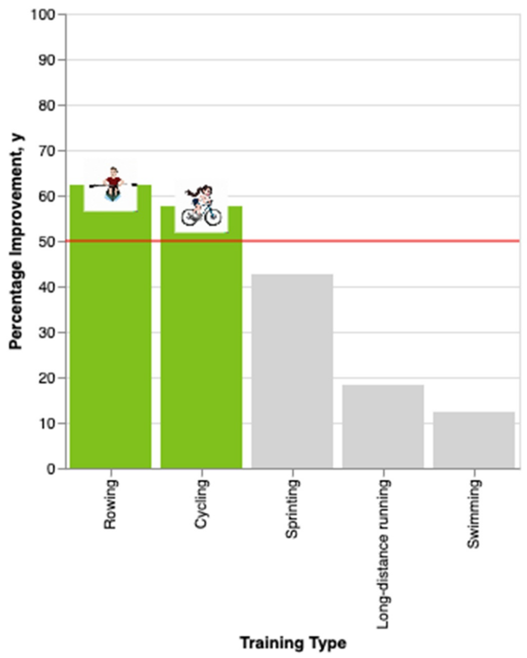
```
line = alt.Chart(pd.DataFrame({'y': [50]})).mark_rule(color='red'
chart = chart + line
```

Generate a baseline

The code generated by Copilot uses `mark_rule()` to generate a horizontal line and the + operator to layer the two charts.

Figure 4.24 shows the final chart. With respect to Figure 4.19, we have added context that helps the reader to understand immediately which sports we should invest in.

Figure 4.24 The bar chart with the two sports performing better



We could further improve the chart in Figure 4.24 by adjusting axes titles and applying decluttering. We'll see how to do it in the next chapters.

Now that we have added context to our chart, we can move to the next step of the DIKW pyramid, which is turning knowledge into wisdom.

4.5.3 Turning Knowledge into Wisdom

This step involves adding an action to the chart. Let's add a title that invites the audience to invest in rowing and cycling. We can use ChatGPT to generate the title. Write the following prompt:

- *Act as an inviter to action*
- *tailored to the boss of a sport company.*
- *Generate a compelling title for the following text in angle brackets <You should invest funds for rowing and cycling, which are the most practiced sports>.*

An example of a generated title is *Unlock the Potential: Invest in Rowing and Cycling for Maximum Returns!* If you are unsatisfied with the generated title, you can ask ChatGPT to generate another one, or a list of N titles.

Once we have defined the title, we can write the instructions for Copilot to add it to the chart, as shown in Listing 4.18. Add the comment within the `properties()` method of the chart.

Listing 4.18 Adding a title

```
# Add the following properties to the chart:
    # * title to 'Unlock the Potential: Invest in Rowing and Cycl
```

Add the title

Add this instruction within the properties function of the chart

Listing 4.19 shows the generated code.

Listing 4.19 The generated code to add a title

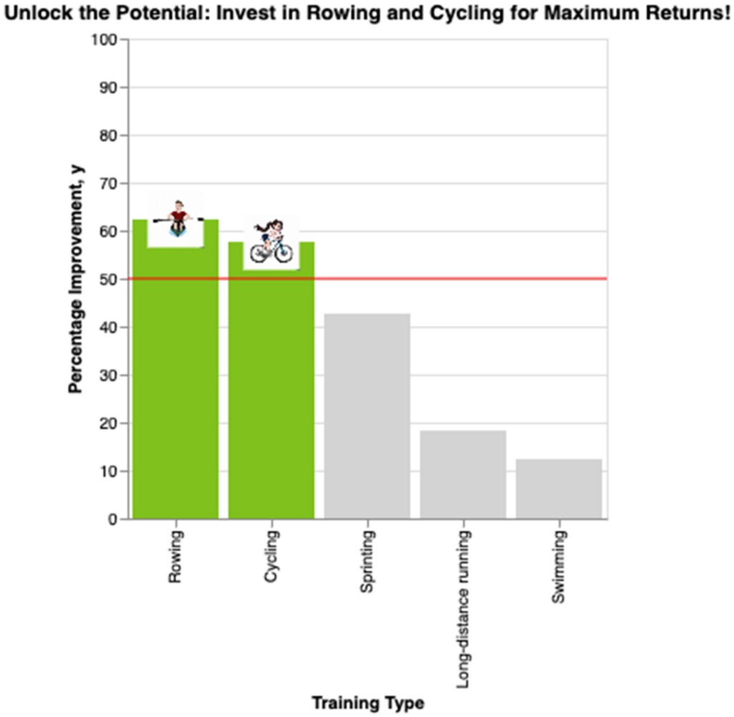
```
title='Unlock the Potential: Invest in Rowing and Cycling for Max
```

Add a title

The code generated by Copilot uses the `title` property.

Figure 4.25 shows the final chart.

Figure 4.25 The final bar chart with the call to action in the title



In this chapter, you have learned the basic concepts behind Generative AI tools, with a specific focus on ChatGPT, DALL-E, and Copilot. In the next chapter, you'll see how to combine Altair and Generative AI tools to build compelling data stories using the DIKW pyramid.

4.6 Summary

- Generative AI is a subfield of Artificial Intelligence, aiming at generating new content based on the trained data. You can use Generative AI for different purposes, including text, image, code, video, and speech generation.
- A prompt is an input text containing the instructions for a Generative AI tool.
- ChatGPT is a text-to-text Generative AI tool. To write effective prompts for ChatGPT structure the prompt in three parts: role, audience, and subject.
- DALL-E is a text-to-image Generative AI tool. To write effective prompts for DALL-E, define the subject and the style.
- Copilot is a text-to-code Generative AI tool. To make Copilot write code suggestions efficiently, write specific and unambiguous instructions,

using simple language and avoiding complex terminology.
- Don’t use Copilot to substitute you as a developer. Rather, use it to assist you while programming.
- Always keep in mind that Generative AI cannot substitute your creativity and reasoning capabilities.

4.7 Additional Resources

- Niyati Madhvani. *Generative AI Tools in the Creative Domains: The Power and Pressure Game Is On!* March 2023. https://www.rapidops.com/blog/generative-ai-tools/ (Last Access July 2nd, 2023)
- Jonathan Kemper. *ChatGPT Guide: Use these prompt strategies to maximize your results*. Jun 19, 2023 https://the-decoder.com/chatgpt-guide-prompt-strategies/ (Last Access June 29, 2023)
- Isa Fulford, Andrew Ng. *ChatGPT Prompt Engineering for Developers*. https://www.deeplearning.ai/short-courses/chatgpt-prompt-engineering-for-developers/ (Last Access June 29, 2023)
- *The DALL·E 2 Prompt Book* https://dallery.gallery/wp-content/uploads/2022/07/The-DALL%C2%B7E-2-prompt-book-v1.02.pdf (Last Access July 1st, 2023)
- Tory Barber. Dall-e Prompting Tips & Tricks. https://torybarber.com/dall-e-prompting-tips-tricks/ (Last Access July 1st, 2023)

[1] https://www.youtube.com/watch?v=So48YMYyl58

[2] freely inspired by Simone Oliani. *Ho paura di essere debole (I'm afraid of being weak).* Messaggero di sant’Antonio, 2023, n.6 p.53.

5 Crafting a data story through the DIKW Pyramid

This chapter covers

- The homelessness tale - our scenario for this chapter
- What a data story is and how it relates to the DIKW pyramid
- How to incorporate Generative AI in the DIKW pyramid

In this chapter, we will focus on the practice of using the DIKW pyramid by describing two examples. The first example will analyze the problem of homelessness in Italy, and the second the problem of fake news into a hypothetical website. We will also describe the concept of data story and how it relates to the DIKW pyramid already learned in the previous chapters. Finally, we will describe some strategies to incorporate Generative AI in the DIKW pyramid, based on how to write specific prompts for ChatGPT to generate the context of a chart and for DALL-E to generate contextual images to incorporate in the chart.

5.1 Breaking the Ice: The Homelessness Tale

Imagine that Angelica works for a humanitarian organization that wants to apply for funding from a Foundation to help reduce the homeless people problem in Italy. Humanitarian interventions can be applied to up to four Italian regions. The call for funds involves preparing a data visualization chart motivating the selected regions and why to fund the proposal.

Angelica’s boss asks her to do a study to figure out which regions to invest in and motivate her choice.Angelica starts a web search for possible datasets on homelessness in Italy. After several searches, she comes across the ISTAT dataset on homelessness in 2021[1]. She downloads the dataset and starts analyzing it. Table 5.1 shows a simplified version of an extract of the homelessness dataset. The original dataset column names are in Italian. We have translated them into English to improve readability.

Table 5.1 An extract of the ISTAT dataset on homelessness in 2021

ITTER107	Territory	Sex	Age	Citizenship	Value
ITC1	Piemonte	M	TOTAL	ITL	4218
ITC1	Piemonte	F	TOTAL	ITL	1496
ITC2	Valle d'Aosta	M	TOTAL	ITL	41
ITC2	Valle d'Aosta	F	TOTAL	ITL	17

The ISTAT dataset on homelessness

The dataset has the following columns:

- *ITTER107* - the region ID,
- *Territory* - the region name,
- *Sex* - one of Male (M), Female (F), Total (T),
- *Age* - one of Total (TOTAL), under 17 (Y_UN17), between 18 and 34 (Y18-34), between 35 and 54 (Y35-54), and greater than 55 ('Y_GE55)
- *Citizenship* - one of Italian (ITL), Foreign (FRGAPO), Total (TOTAL)
- *Value* - the actual number of homeless people

At the end of her analysis, Angelica produces the chart shown in Figure 5.1 by writing the code in Listing 5.1, using Altair. You can find the full code in the GitHub repository of the book under `05/homeless-people/raw-chart.py`.

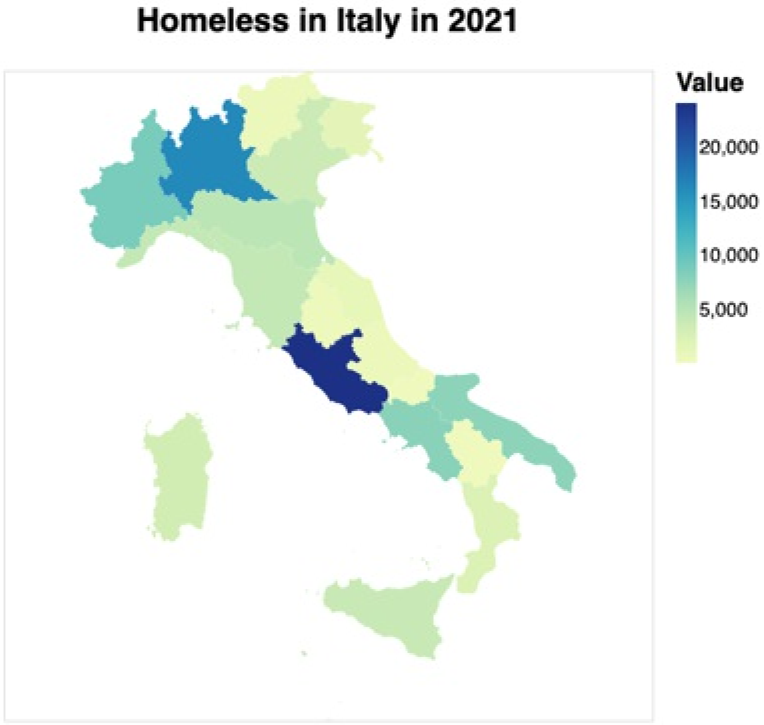
Listing 5.1 The map in Altair

```
import pandas as pd
import altair as alt

df = pd.read_csv('source/homeless.csv')

# Data Cleaning
df['Territory'] = df['Territory'].str.replace('Trentino Alto Adig

# Focus on total age, total sex and total citizenship
df = df[(df['Age'] == 'TOTAL') & (df['Sex'] == 'T') &  (df['Citiz

# Load the TopoJSON file by URL
url = "https://raw.githubusercontent.com/openpolis/geojson-italy/
map = alt.topo_feature(url, "regions")

chart = alt.Chart(map).mark_geoshape().encode(
    tooltip='properties.reg_name:N',
    color=alt.Color('Value:Q')
).project('mercator').properties(
    width=500,
    height=500
).transform_lookup(
    lookup='properties.reg_name',
    from_=alt.LookupData(df, 'Territory', ['Territory', 'Value'])
).properties(title='Homeless in Italy in 2021')

chart.save('raw-chart.html')
```

Chart

The chart builds a map using an underlying `topoJSON` object containing the map of Italy. To build the map, use the `mark_geoshape()` marker and specify the projection through the `project()` method. In addition, combine the homeless dataset (`df`) and the map through the `transform_lookup()` method. Map the `properties.reg_name` variable of the `topoJSON` file with the `Territory` variable of the `df` DataFrame. We’ll discuss how to build a geographical map more deeply in the next chapter.

Figure 5.1 A map showing the homeless people in Italy in 2021

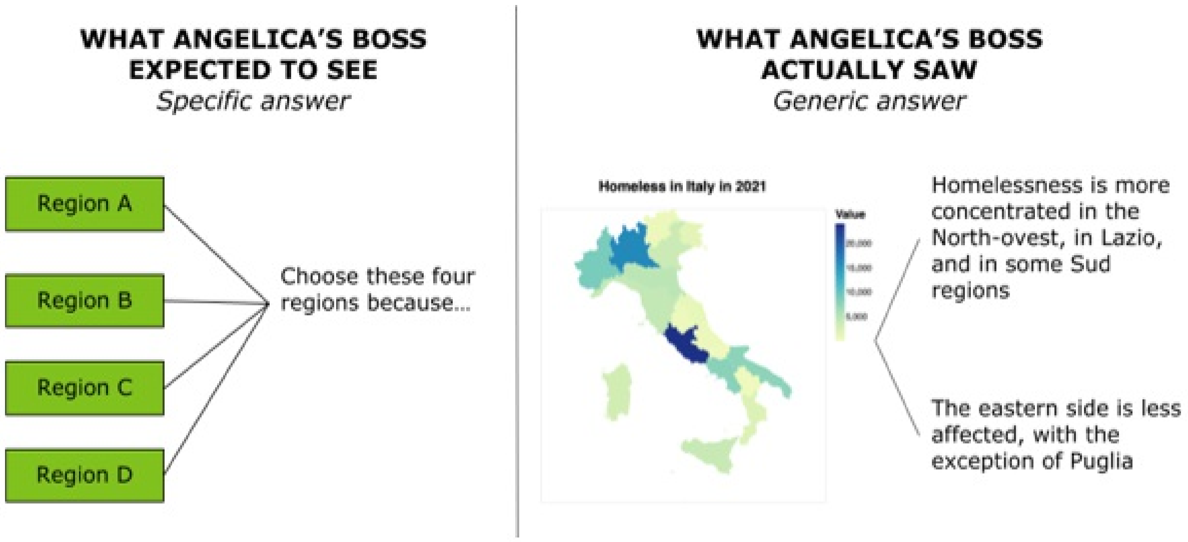


Angelica shows the chart to her boss, who asks her to answer their question: *which are the four regions for which we must apply for funding?* Angelica looks at the chart, and she answers: *those with the darker color*. Her answer is very ambiguous because, from the chart she produced, it is not immediately clear which exactly are the four regions with the darkest color. In addition, it is not clear why to select the four regions. Angelica has to admit that her chart has failed its mission.

5.1.1 What Was Wrong with the Chart?

The main problem with the chart is a large discrepancy between what Angelica's boss (audience) expected to see in the chart and what the chart actually shows. In practice, Angelica's chart does not answer the question for which it was thought (Figure 5.2).

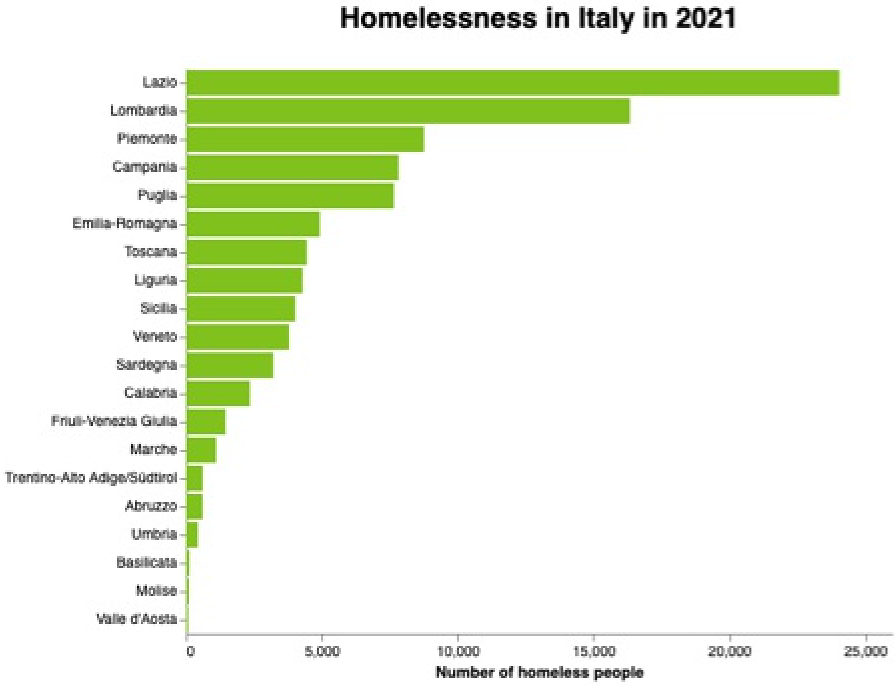
Figure 5.2 Angelica’s boss (audience) requested to answer a specific question (on the left). The chart actually answered generic questions (on the right).


WHAT ANGELICA'S BOSS EXPECTED TO SEE
Specific answer
Region A
Region B
Region C
Region D
Choose these four regions because...
WHAT ANGELICA'S BOSS ACTUALLY SAW
Generic answer
Homeless in Italy in 2021
Value
Homelessness is more concentrated in the North-ovest, in Lazio, and in some Sud regions
The eastern side is less affected, with the exception of Puglia

Here comes the first rule when building a chart: a chart must answer exactly the question the intended audience asks. Try not to leave it up to your audience to decipher the answer. Do the work for them!

Angelica's boss asks her to redo the chart. Then she returns to her office with her head down and analyzes her dataset again. Angelica understands that the problem is not so much data analysis but rather how to represent data effectively. She tries this reasoning: *at a glance, the audience must understand the four regions to apply for funding. What could these four regions be? Probably the ones that have the highest value of homeless people*. What is missing in her previous chart, is the *probably* part. She thinks about making a bar chart that shows the regions in descending order based on the number of homeless people present. After a small effort, she produces the chart in Figure 5.3, writing the code in Listing 5.2. You can also find the code in the GitHub repository of the book, under 05/homeless-people/bar-chart.py.

Figure 5.3 A bar chart showing the homeless people in Italy in 2021



Listing 5.2 The bar chart in Altair

```
chart = alt.Chart(df).mark_bar(
    color='#80C11E'
).encode(
    y = alt.Y('Territory',
              sort='-x',
              axis=alt.Axis(title='')),
    x = alt.X('Value',
              axis=alt.Axis(tickCount=4,title='Number of homeless
).properties(
    width=500,
    title='Homelessness in Italy in 2021'
)

).properties(title='Homeless in Italy in 2021')

chart = chart.configure_title(
    fontSize=20,
    offset=25
).configure_axis(
    grid=False
).configure_view(
    strokeWidth=0
)
```

```
chart.save('bar-chart.html')
```

Chart

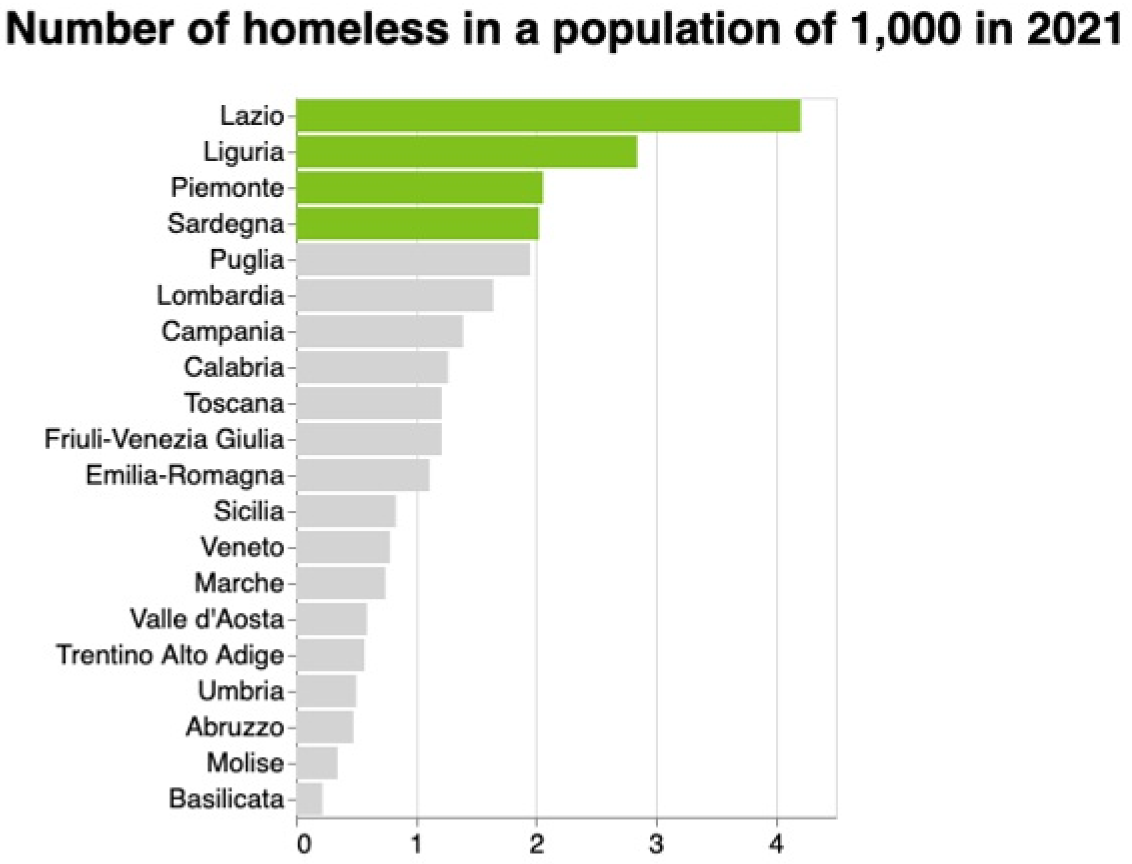
The chart builds a bar chart using the `mark_bar()` method. It also sets the bars' color through the `color` parameter. To order the bars in descending order, use the `sort` parameter within the `y` channel and specify to sort in descending order of the `x` encoding channel. Use configure_title() to configure the title parameters, such as the font size (fontSize) and the offset between the title and the chart.

The chart answers the question: the four regions are Lazio, Lombardia, Piemonte, and Campania. Angelica notes that three of the four regions also correspond to the most populous regions of Italy. As Angelica reflects on this, she notices that her chart contains an underlying error. The most populous regions also have the highest number of homeless people precisely because the population is greater.

Therefore, the chart does not show a real situation. She needs to normalize the data based on the population. Normalization means scaling the numbers by a population. Without any normalization, data could be biased. Consider, for example, the case of a population with 100 people of which 30 are homeless. Also, consider a population with 10 people of which 8 are homeless. If you represent absolute values, you conclude that in the first case, there are more homeless people than in the second one. But, if you pay more attention to the scenario, in the second case 80% of the population is homeless, while in the first case only 30%.

Figure 5.4 shows the resulting chart after the normalization process focusing on the top-4 regions. Listing 5.3 describes the code written to draw the chart (script 05/homeless-people/story-chart.py).

Figure 5.4 A bar chart showing the number of homeless in a population of 1,000 in 2021



Listing 5.3 The improved bar chart in Altair

```
import pandas as pd
import altair as alt

df = pd.read_csv('source/homeless.csv')

# Data Cleaning
df['Territory'] = df['Territory'].str.replace('Trentino Alto Adig

# Focus on total age, total sex and total citizenship
df = df[(df['Age'] == 'TOTAL') & (df['Sex'] == 'T') &  (df['Citiz
df = df[['Value', 'ITTER107']]

# Normalize the values by popultation
df_pop = pd.read_csv('source/population.csv')
df_pop = df_pop[(df_pop['Age'] == 'TOTAL') & (df_pop['Sex'] == 'T


df_pop = df_pop[['Value', 'ITTER107','Territory']]
```

```
df_tot = df_pop.set_index('ITTER107').join(df.set_index('ITTER107
df_tot['Ratio'] = df_tot['Value_hom']/df_tot['Value_pop']*1000


chart = alt.Chart(df_tot).mark_bar().encode(
    y = alt.Y('Territory',
              sort='-x',
              axis=alt.Axis(title='')),
    x = alt.X('Ratio',
              axis=alt.Axis(tickCount=4,title='')),
    color=alt.condition(alt.datum.Ratio > 2,
                        alt.value('#80C11E'),
                        alt.value('lightgray'))

).properties(
    width=500,
    title='Number of homeless people in a population of 1,000'
)
```

Chart

The code loads the population dataset, which contains the same fields as the homeless dataset, except for the *Value* column, which indicates the actual population. The code merges the two datasets, population (`df_pop`) and homeless (`df`), and calculates the ratio between the number of homeless and the population. Then, the chart builds a bar chart using the ratio as the `X` encoding channel.

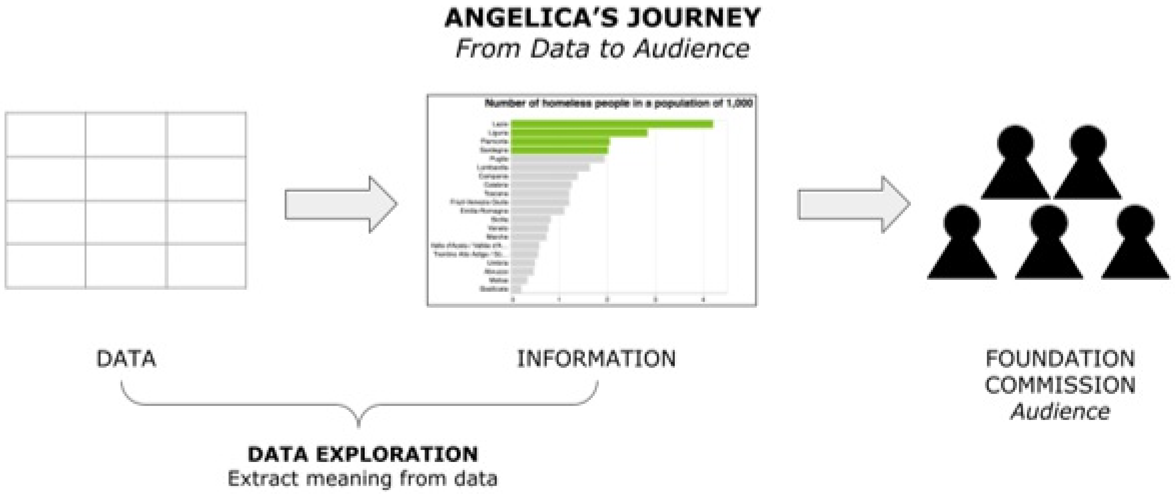
Angelica’s chart eventually answers her boss’s question: the four regions are Lazio, Liguria, Piemonte, and Sardegna. Finally, Angelica shows her chart to her boss, who is very satisfied with her result.

To apply for funds, Angelica must send her chart to a commission, which will decide whether or not to finance Angelica’s organization proposal. However, after a careful evaluation, the Foundation Commission decides not to finance her project because the chart does not contain any motivation.

5.1.2 What Was Wrong with the Presentation?

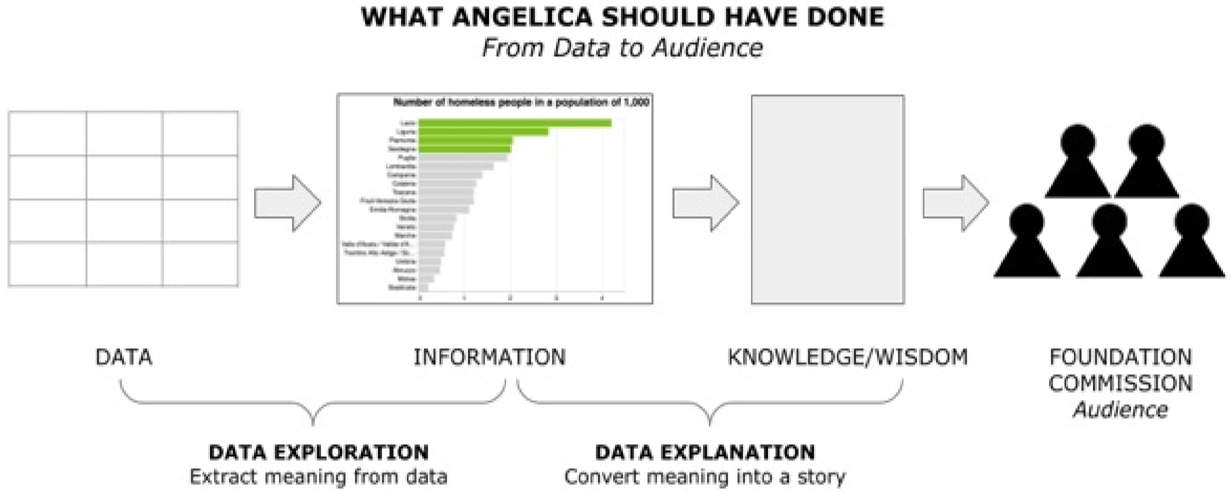
To understand why the Foundation Commission refused to fund Angelica’s proposal, let's trace back the path she followed. She started with raw data, analyzed it, extracted a meaning, and represented it through a chart, then she sent her chart to the Foundation Commission (Figure 5.5).

Figure 5.5 Angelica’s journey to chart generation. Angelica presented the results of data exploration directly to her audience



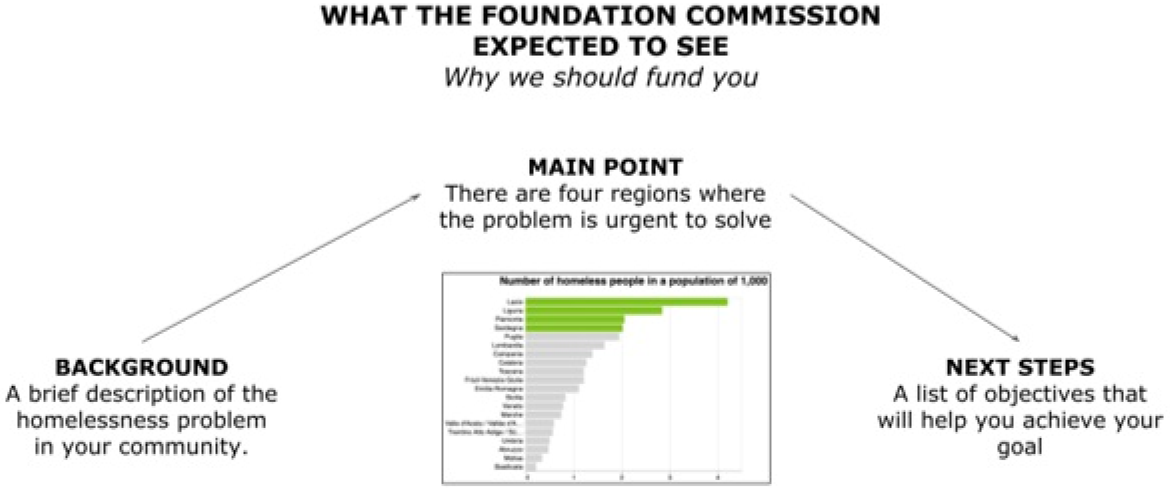
What is missing from her chart is that it contains no context, and no motivation to push the Foundation Commission to accept her proposal. Angelica should have added to the exploratory phase of the data an explanatory phase where she could have engaged the audience with her data and motivated them to accept her proposal. Angelica could have used the DIKW model to move from the exploratory phase of her data to the explanatory phase (Figure 5.6).

Figure 5.6 Angelica should have transformed her data exploration into a data explanation phase


WHAT ANGELICA SHOULD HAVE DONE
From Data to Audience
Number of homeless people in a population of 1,000
DATA
INFORMATION
KNOWLEDGE/WISDOM
FOUNDATION COMMISSION
Audience
DATA EXPLORATION
Extract meaning from data
DATA EXPLANATION
Convert meaning into a story

In other words, Angelica hasn't turned data into a story. In his book *Effective Data Storytelling*, Brent Dykes says: “The formation of a data story begins with using exploratory data visualizations to discover insights. Once a meaningful insight is uncovered, explanatory data visualizations are used to tell a story” (Dykes, 2019). The chart built by Angelica constitutes only the main point of the data story. However, to have a complete story, Angelica should have also included a background and next steps (Figure 5.7). The Foundation Commission expected Angelica’s chart to answer the question: why should we fund your project? Organizing the chart as a story would have increased the funding chances.

Figure 5.7 The Foundation Commission expected to see a story with a background, a main point, and next steps, inviting them to fund Angelica’s proposal


WHAT THE FOUNDATION COMMISSION
EXPECTED TO SEE
Why we should fund you
MAIN POINT
There are four regions where the problem is urgent to solve
Number of homeless people in a population of 1,000
BACKGROUND
A brief description of the homelessness problem in your community.
NEXT STEPS
A list of objectives that will help you achieve your goal

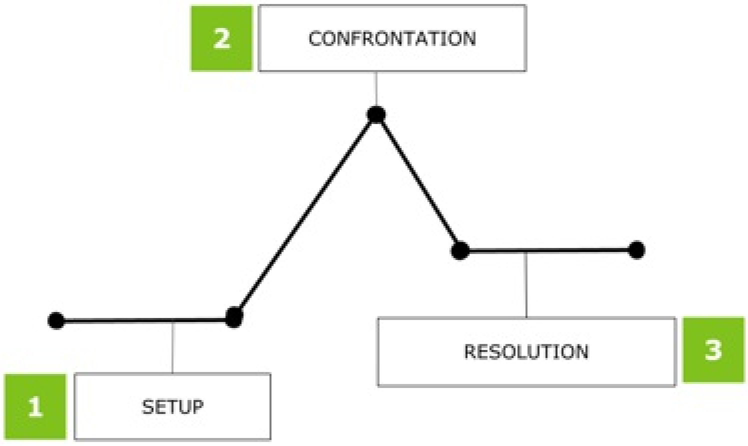
Now that we have understood the urgency to turn data into a story to communicate a message to an audience effectively, let’s move to the next step, what a data story is and its main components.

5.2 Uncovering the Narrative: What a Data Story is

According to the Cambridge Dictionnary, a story is “a description, either true or imagined, of a connected series of events.. A story is a way to share information or entertainment through a structured, cohesive narrative that engages the audience. The goal of a story is always to engage its audience to communicate a message.

In his book *Poetics*, the Greek philosopher Aristotle proposed a framework to structure a tragedy, also known as the Three-Act Structure. The structure consists of three main parts: the setup, the confrontation, and the resolution (Figure 5.8).

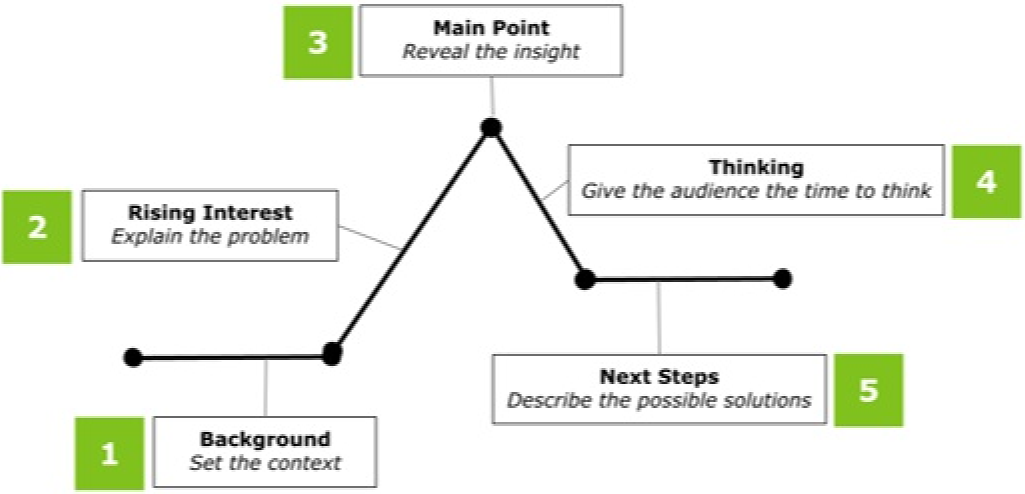
Figure 5.8 The Three-Act Structure proposed by Aristotle to structure a tragedy



The setup introduces the audience to the characters, setting, and basic conflict of the story. In the confrontation, the main character faces obstacles and struggles to overcome them. Finally, in the resolution, the story reaches the point at which conflicts are resolved. The Aristotle tragedy structure is still widely used today in literature, film, and other forms of storytelling. Its effectiveness lies in its ability to build tension and suspense, leading to a satisfying resolution for the audience.

In the aforementioned book, Dykes proposes an improved version of the Three-Act Tragedy, adapted to the data storytelling scenario (Figure 5.9). Dykes calls it the Data Storytelling Arc.

Figure 5.9 The Data Storytelling Arc proposed by Dykes to structure a data story



A data story starts by defining the background behind data (1) and raising the audience's interest (2). Dykes also proposes including a hook during this initial phase. A hook is “a notable observation that acts as a turning point in the story and begins to reveal a problem or opportunity” (Dykes, 2019). Then, the data story culminates with the main point behind the data by revealing the insight (3), leaving the audience the time to think (4), and terminates by proposing the next steps (5).

The structure proposed in Figure 4.9 might seem abstract and difficult to apply when constructing a chart. Using the DIKW pyramid learned in Chapter 1, you can turn a simple chart into a story.

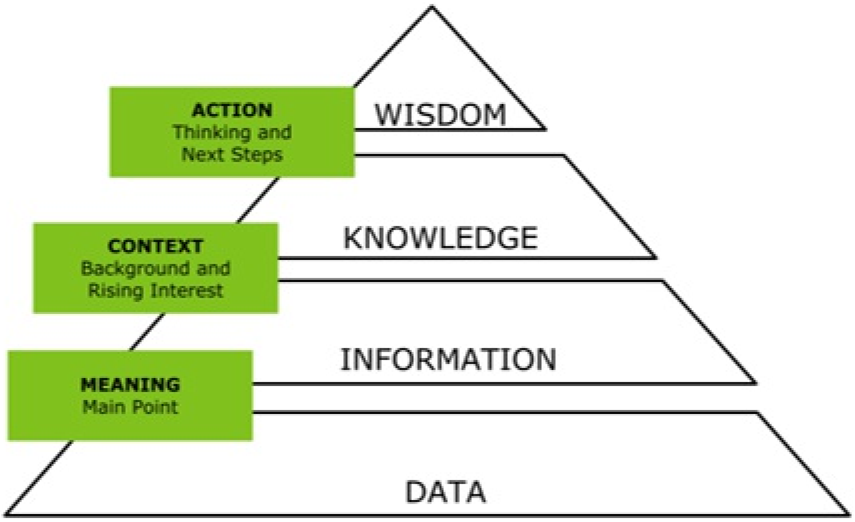
5.2.1 Using the DIKW Pyramid to Streamline a Data Story

In Chapter 1, we introduced the Data Information Knowledge Wisdom (DIKW) pyramid and used it to transform data into wisdom in the practical example of the event dedicated to pets. We can generalize the example described in Chapter 2 and use the DIKW to build any data story. Figure 5.10 shows how to map the Data Storytelling Arc to the DIKW pyramid:

- Extracting meaning from data corresponds to the main point of a data story
- Adding a context to the extracted information corresponds to defining the background and raise the audience interest

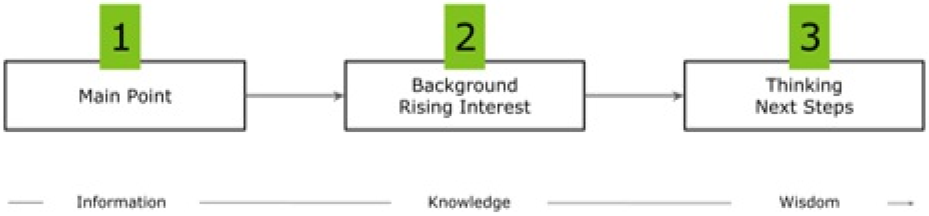
- Adding a call to action corresponds to inviting the audience to think and follow the next steps.

Figure 5.10 A mapping between the Data Storytelling Arc and the DIKW pyramid



The described mapping suggests the natural flow when you build a data story (Figure 5.11). First, identify the main point of your story, then set the background, and, finally, set the next steps.

Figure 5.11 The flow to build a data story



5.2.2 DIKW in Action: Completing the Homelessness Tale

The Homelessness example described at the beginning of the chapter stopped at the information level of the DIKW pyramid that, in terms of a data story, consisted only of the main point. Adding background to the chart involves grabbing your audience's attention and making them interested in the problem you're describing. In Chapter 2, you learned that in Altair you can add textual annotations to a chart to transform information into knowledge. In addition, there are other ways to add context to a chart, including labels, titles, sources, images, descriptions, and comparisons. We’ll describe them more in detail in Chapter 6, *From Information to Knowledge: Building Context*.

In the case of the homeless problem, you can add text describing the situation of these people and possibly a photo that gives a face to the people involved. We will see how to add a hero to the story in Chapter 6, *From Information to Knowledge: Building Context*. For now, it's enough to give a face to the people behind the data.

In Listing 4.4, we add a text describing the context as the chart subtitle. Use the `TitleParams()` function to specify the title properties, including the subtitle. Set the subtitle to *Homelessness is a heartbreaking reality that leaves individuals and families without a stable home, leading to devastating consequences such as poor health and social isolation*.

Listing 5.4 Adding the context as the subtitle

```
# Add context
chart = chart.properties(width=500,title=alt.TitleParams(
    text=["Together, Let's Make a Difference:","Support Our Proje
    subtitle=['Homelessness is a heartbreaking reality that leave
    subtitleFontSize=18,
))
```

How to add a subtitle to a chart

Within the `properties()` method, use the title parameter to set the chart title. Define the title using the `TitleParams()` function, which can receive many parameters as an input. The example passes the following parameters: title (`text`), subtitle (`subtitle`), subtitle font size (`subTitleFontSize`), title font size (`titleFontSize`).

Listing 5.5 shows how to add two photos to the chart and combine them with the original chart to build a dashboard. To make the code work, you must run it on a web server. If you don’t have a web server, you can run a local and temporary web server from command line, running the following command in the directory containing the produced HTML file: python -m http.server. The server should listen at port 8000 and should serve all the files contained in the directory from which it is started. Point to http://localhost:8000/chart.html in your browser to access the file chart.html.

Listing 5.5 Adding the context as images

```
# Build the first image
image1 = alt.Chart(pd.DataFrame({'image_url': ['source/homeless1.
    width=200,
    height=200,
).encode(
    url='image_url',
    x=alt.value(0),  # pixels from left
    y=alt.value(50)  # pixels from the top
)

# Build the second image
image2 = alt.Chart(pd.DataFrame({'image_url': ['source/homeless2.
    width=200,
    height=200,
).encode(
    url='image_url',
    x=alt.value(0),  # pixels from left
    y=alt.value(300)
)

image1 + image2 | chart
```

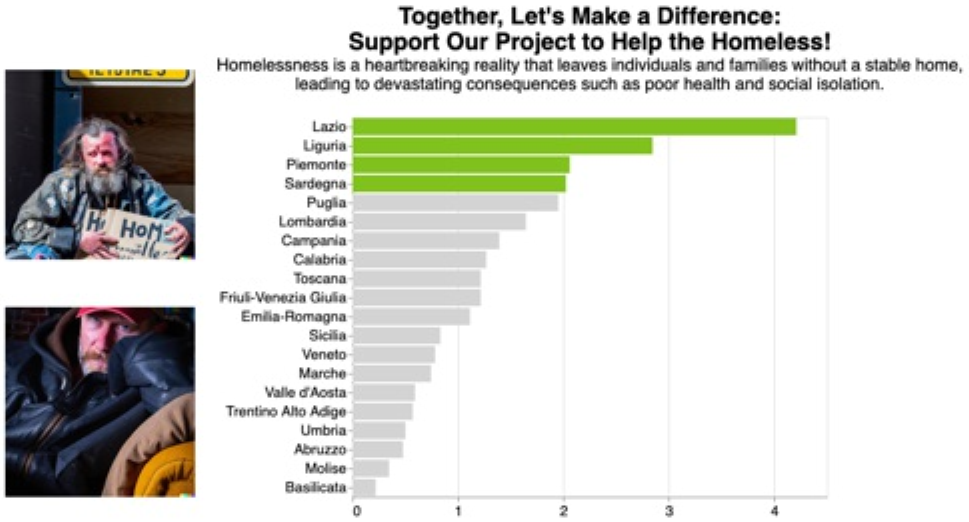
How to add images to a chart

Use `mark_image()` to add an image to a chart. Set the path to the image file in the DataFrame passed to the chart. Use the `x` and `y` channels to set the image position in the chart. Use the same chart to draw `image1` and `image2` (+ operator) and draw the main chart on the right with respect to `image1 + image2`.

To complete the homelessness story, add a call to action, which includes the next steps. In this example, we first modify the title to: *Together, Let's Make a Difference: Support Our Project to Help the Homeless!* We'll see different strategies to add a call to action to a chart in Chapter 7, *From Knowledge to*

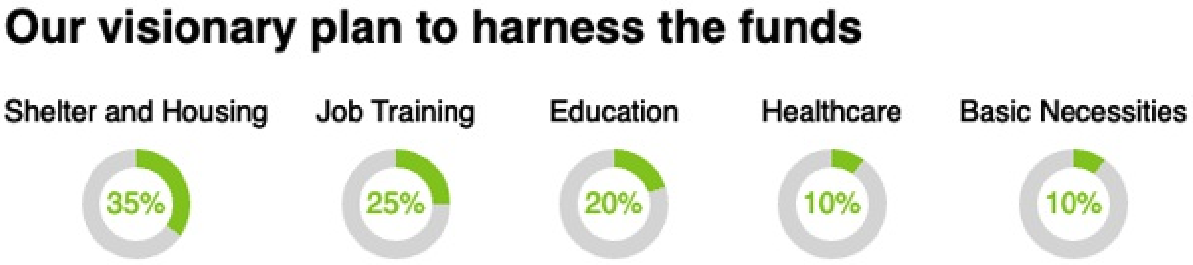
Wisdom: Adding Ethics. Figure 5.12 shows the resulting chart.

Figure 5.12 The homelessness chart is enriched with a context (images and subtitle)



Then, we add a part of the chart describing how we will use funds (Figure 5.13). For example, we will use 35% of the funds for shelter and housing, 25% for job training, and so on.

Figure 5.13 A possible next step explaining how we will use funds



Listing 5.6 shows how we implemented the charts in Figure 5.13. We assume we have a DataFrame storing the percentage of funds (Allocation) for each Category

Listing 5.6 Adding a next step

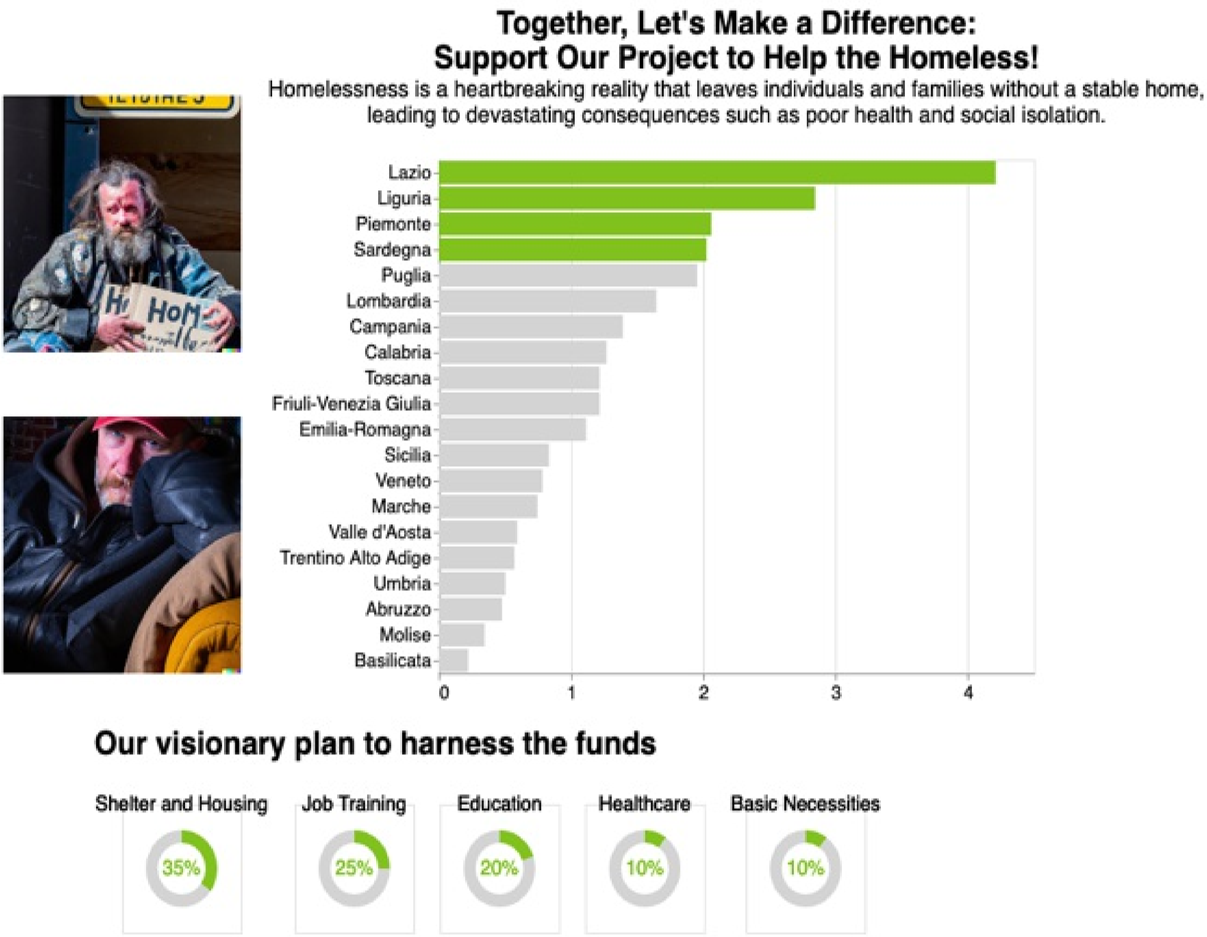
```
import pandas as pd
import altair as alt

donuts = None

for index, row in ns.iterrows():
    curr_ns = pd.DataFrame(
        {'Category': ['A', 'B'],
         'Value': [row['Allocation'], 100-row['Allocation']]
        } #A
    )

    donut = alt.Chart(curr_ns).mark_arc(outerRadius=30, innerRadi
        theta=alt.Theta("Value:Q", stack=True),
        color=alt.Color("Category:N", scale=alt.Scale(range=['gre
    )

    title = alt.Chart(curr_ns).mark_text(text=row['Category'], y=

    text = alt.Chart(curr_ns).mark_text(text=f"{row['Allocation']

    donut = donut.properties(
        height=100,
        width=100
    ) #E

    #F
    if index == 0:
        #G
        donuts = title + donut + text
    else:
        #H
        donuts = alt.hconcat(donuts, title + donut + text)
```

How to implement the charts in Figure 5.13

Implement each donut chart as a separate chart using `mark_arc()`. For each donut, build an ad-hoc DataFrame (`current_ns`) that contains the actual allocation value and its complementary value (100 - actual allocation value). Then, use different colors to plot the current allocation value and its complementary value.

Figure 5.14 shows the resulting chart. If Angelica had submitted the chart in Figure 5.14 to the Commission Foundation, she would probably have had more opportunities to get funded.

Figure 5.14 The final chart for the homeless tale



Now that you have completed the homelessness tale, let’s describe how you can incorporate the power of Generative AI in the DIKW pyramid.

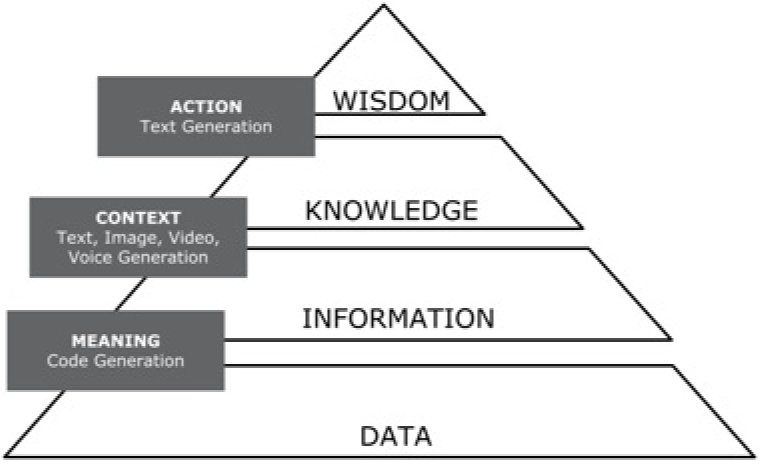
5.3 Incorporating Generative AI into the DIKW Pyramid

Generative AI, as we covered in Chapter 4, can help us take things one step further by enabling the creation of sophisticated and realistic computer-generated content that can revolutionize various industries and domains. Now, armed with this knowledge, we can leverage generative AI's capabilities to augment human creativity and automate content generation processes.

Figure 5.15 shows where we can incorporate Generative AI tools into each step of the DIKW pyramid:

- *Meaning extraction*: use generative algorithms to automate the creation code that builds the chart.
- *Context adding*: use text generation to transform basic texts into more engaging ones for the audience. Add images, voice, and videos generated by AI to give the audience additional context
- *Call to action*: use text generation to suggest new ideas and engaging text.

Figure 5.15 Where we can incorporate Generative AI tools into the DIKW pyramid



It's important to note that the examples listed are just a few of the ways that you can incorporate Generative AI into the DIKW pyramid. There are countless other ways in which you can leverage these tools, such as synthesizing large amounts of data, developing personalized, predictive models, and constructing personalized recommendations based on data. However, in this book, we will focus specifically on the techniques for integrating Generative AI into the DIKW pyramid, as it can assist in building a data story. Anyway, we encourage you to think outside the box and explore new and innovative ways to apply Generative AI in your work. With so much potential waiting to be unlocked, the possibilities are truly endless!

Now that you have learned how to incorporate Generative AI tools into the DIKW pyramid, let’s apply this strategy to our case study, the homelessness tale.

5.4 Behind the Scenes: The Homelessness Tale

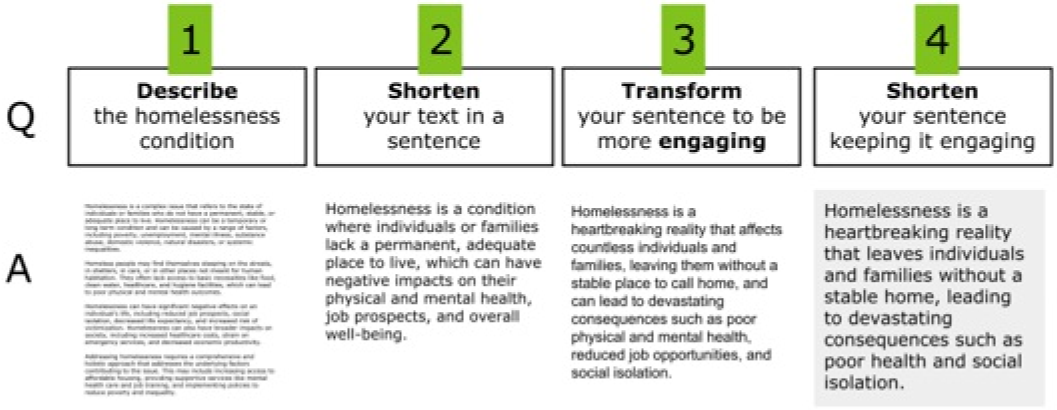
In the previous chapters, you saw how to use GitHub Copilot to generate the code to build the chart. Now, it’s time to go a step further.

Consider the homelessness tale again and specifically Figure 5.12. The figure contains a title, defining the call to action, a subtitle, setting the context, and two photos, setting the context too. While you may have thought that we came up with the title and created or downloaded the images, in reality, it was ChatGPT and DALL-E that worked together to generate the content! Let’s see how we used ChatGPT to generate the subtitle and DALL-E to generate the photos. In the next chapters, we will see how to use more deeply generative AI tools for data storytelling.

5.4.1 Creating a Compelling Subtitle

We started a conversation with ChatGPT to extract a possible context that describes the situation where homeless individuals live. The context should be a short and engaging sentence. Figure 5.16 shows the steps involved in the conversation. Q means the user questions (which will form the basis of our prompts), and A the ChatGPT answers. You can read the complete conversation in `05/genAI/SubtitleConversation.txt`.

Figure 5.16 The steps to generate the text used for context.


1
2
3
4
Q
Describe the homelessness condition
Shorten your text in a sentence
Transform your sentence to be more **engaging**
Shorten your sentence keeping it engaging
A

Homelessness is a condition where individuals or families lack a permanent, adequate place to live, which can have negative impacts on their physical and mental health, job prospects, and overall well-being.

Homelessness is a heartbreaking reality that affects countless individuals and families, leaving them without a stable place to call home, and can lead to devastating consequences such as poor physical and mental health, reduced job opportunities, and social isolation.

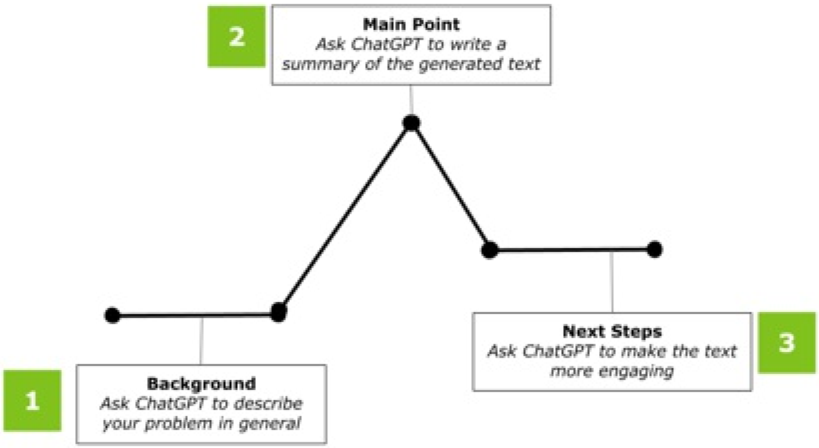
Homelessness is a heartbreaking reality that leaves individuals and families without a stable home, leading to devastating consequences such as poor health and social isolation.

ChatGPT generated the text after four steps. We used the following strategy to make ChatGPT generate the desired text:

1. *Describe*: ask ChatGPT to describe your problem in general. In this case, ask ChatGPT to describe the homelessness condition in general. As an answer, ChatGPT generates a long text
2. *Shorten*: ask ChatGPT to write a summary of the generated text
3. *Transform*: ask ChatGPT to make the summary more engaging for the audience
4. *Shorten*: if the text is still long, ask ChatGPT to reduce it.

Without realizing it, we applied the DIKW model to the use of ChatGPT. Starting from a long text (data), we extracted the information (summary) and then converted it into knowledge and wisdom (engaging text). In other words, when you talk to ChatGPT to generate context, organize the conversation as a story (Figure 5.17).

Figure 5.17 The mapping between the data story structure and the ChatGPT conversation to generate context.


2
Main Point
Ask ChatGPT to write a summary of the generated text
1
Background
Ask ChatGPT to describe your problem in general
3
Next Steps
Ask ChatGPT to make the text more engaging

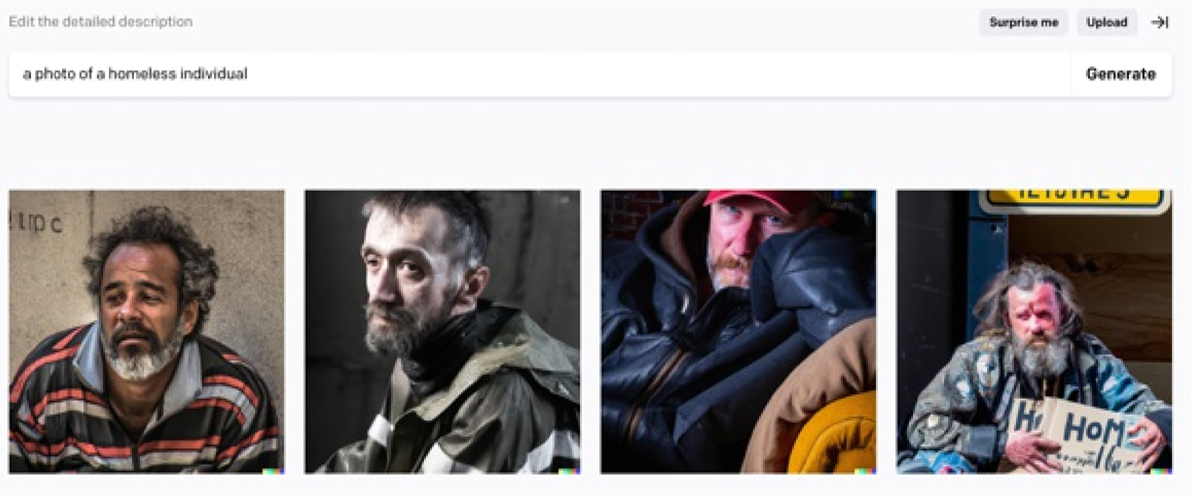
You can generalize the described procedure to generate a variety of possible subtitles and then choose the one that fits your needs. Additionally, you can add specific keywords that can help appeal to your target audience and increase the visibility of your text. We’ll see how to add keywords to the ChatGPT conversation in Chapter 6, *From Information to Knowledge: Building Context*.

5.4.2 Generating Images

Adding one or more images or photos to your chart helps give a face to the topic of your chart. However, images are often subjected to copyright, which means that using them without permission from the owner can lead to legal consequences. We can use DALL-E and other AI tools for images to create unique and original images that are free of copyright restrictions.

In the chart about homelessness, we have added two photos representing homeless individuals. We have generated them using the following simple prompt to DALL-E: *a photo of a homeless individual*. DALL-E has generated four images, and we have chosen two (Figure 5.18). In Chapter 6, *From Information to Knowledge: Building Context*, we will see more advanced techniques to generate images.

Figure 5.18 The photos generated by DALL-E when prompting a photo of a homeless individual.



a photo of a homeless individual

Now that you have learned how to use Generative AI tools to transform your raw data visualization chart into a data story, let’s look at another example, to consolidate the concepts.

5.5 Another Example: Fake News

Imagine that XX is an important website that publishes news from different contributors. At a given point, the editor-in-chief receives some complaints from different readers because they read a high percentage of fake news. The editor-in-chief contacts you to analyze the number of fake news contained in the XX website and advises the website editors to pay attention to the categories of news having the highest probability of being fake. You already have collected data, and you have the dataset shown in Table 5.2.

Table 5.2 An extract of the dataset of the example

Category	Number of Fake Articles	Number of Articles
Politics	1235	1300
Economy	1456	1678

Justice	300	570
Religion	30	100

The dataset shows the number of fake news and the total number of articles for each news category. You start by drawing a preliminary chart, paying attention to the percentage of fake articles for each category. Figure 5.19 shows the preliminary chart and Listing 5.7 the associated code. You can also find the code in the GitHub repository of the book, under 05/fake-news/raw-chart.py.

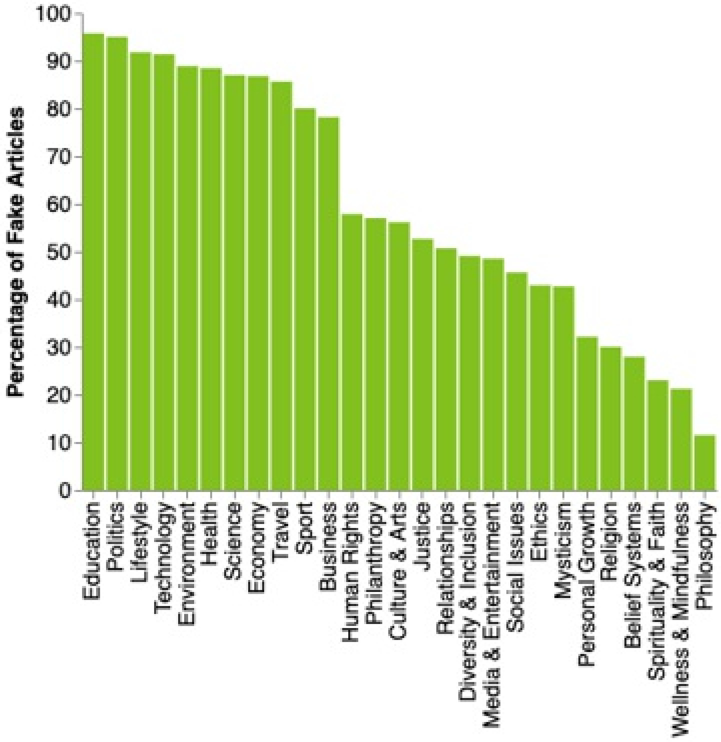
Listing 5.7 Building the raw chart

```
import pandas as pd
import altair as alt

# Load the dataset data/fakenews.csv
df = pd.read_csv('data/fakenews.csv')

# Create a column chart of the number of articles per category #A
# - Use the `Category` column for x channel
# - Use the `Number of Articles` for y channel

chart = alt.Chart(df).mark_bar(
    color='#81c01e'
).encode(
    x=alt.X('Category:N',
            sort='-y',
            title=None,
            axis=alt.Axis(labelFontSize=14)
            ),
    y=alt.Y('Percentage of Fake Articles:Q',
            axis=alt.Axis(labelFontSize=14, titleFontSize=14)
            )
).properties(
    width=400,
    height=300
).configure_axis( #B
    grid=False #C
).configure_view(
    strokeWidth=0
).transform_calculate(
    'Percentage of Fake Articles', alt.datum['Number of Fake Arti
)

chart.save('raw-chart.html')
```

How to draw the preliminary chart

Use transform_calculate() to dynamically add a new column to the DataFrame. This method receives the new column name as the first parameter (Percentage of Fake Articles, in the example), and the expression to calculate the new column, as the second parameter.

Figure 5.19 The raw chart showing the number of fake articles by category

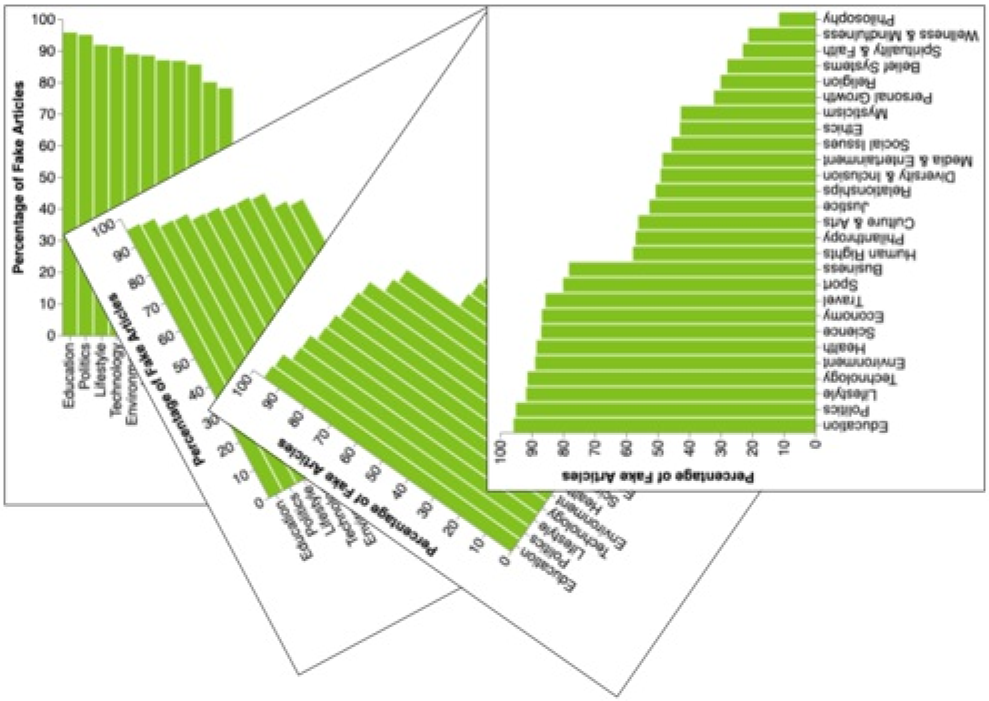


In the remainder of the section, we will apply the DIKW pyramid to transform the chart into a data story. Let’s start with the first step, turning data into information.

5.5.1 From Data into Information

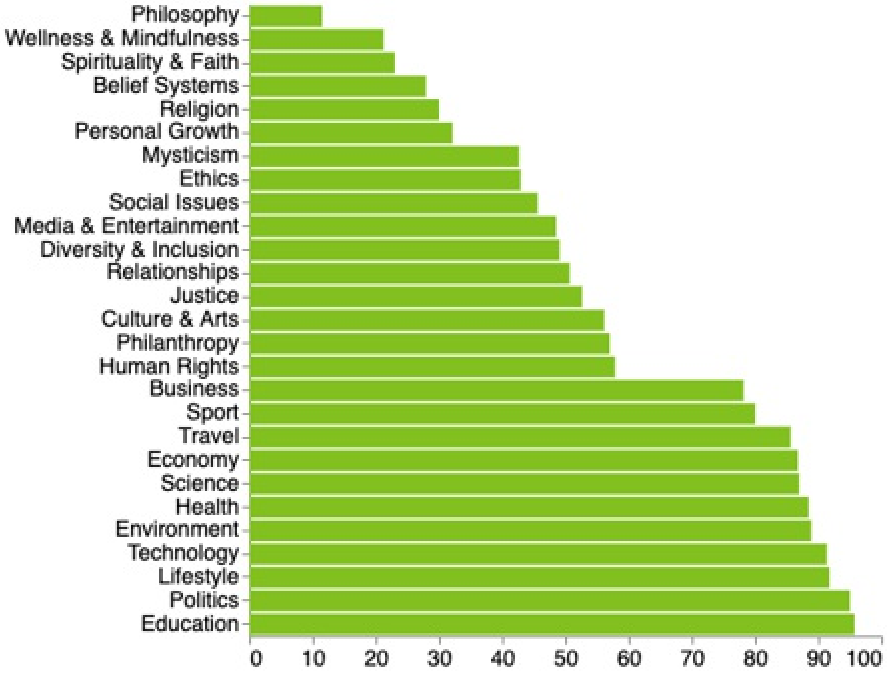
Turning data into information means extracting some meaning from data, an insight that is significant and helps your audience of editors to understand which kind of news has the highest probability of being fake. Let’s try the rotation strategy. This strategy, originally described by Berengueres in his already quoted book, involves rotating the chart to obtain a pyramid and searching for some pyramid framework, such as moral, economic, and other similar framework. Figure 5.20 shows the rotation process.

Figure 5.20 The rotation process of the chart in Figure 5.19



Next, we can flip the chart horizontally and obtain the chart in Figure 5.21, with corrected labels. You can also rewrite directly the Altair code to draw a column chart instead of a bar chart (the code is in the GitHub repository, under 05/fake-news/bar-chart.py. We have preferred to show the rotation process instead of drawing the chart directly to show how the rotation strategy works.

Figure 5.21 The rotation process of the chart in Figure 5.19



If you look at labels carefully, you can notice that at the bottom of the pyramid, there are categories related to material life (from Education to Business). In the middle of the pyramid, there are categories related to moral life (from Human Rights to Ethics). At the top of the pyramid, there are categories related to spiritual life (from Mysticism to Philosophy). This means that most fake news is related to material life (more than 70%) and moral life (more than 30% of fake news, but less than 60%). You can highlight the model material-moral-spiritual life using different colors in the chart, based on the different macro categories the news belongs to.

First, use Copilot to generate the list of macro categories, as shown in Listing 5.8 and in the GitHub repository of the book, under 05/fake-news/story-chart.py. The listing shows only how to generate the material life macro category, but you can apply the same strategy also for the other macro categories.

Listing 5.8 Using Copilot to generate a list

```
# Build a Python list with the following categories and name it m
# Technology
# Environment
# Health
# Science
# Education
# Business
# Lifestyle
# Travel
# Politics
# Economy
# Sport

material_life = ['Technology', 'Environment', 'Health', 'Science'
```

How to use Copilot to generate a list

Describe the elements to include in the list.

Now, ask Copilot to generate the code to add a new column to the DataFrame, containing the macro category. Listing 5.9 shows the instructions for Copilot.

Listing 5.9 Using Copilot to add a new column to a DataFrame

```
# Add a new column to the dataframe df called Macro Category that
# - if the Category is in material_life, then the value is Materi
# - if the Category is in moral_life, then the value is Moral Lif
# - if the Category is in spiritual_life, then the value is Spiri

df['Macro Category'] = df['Category'].apply(lambda x: 'Material L
```

How to use Copilot to add a new column to a DataFrame

Describe how to build the new category of the DataFrame.

Next, use the new column Macro Category to set the color of the bars in the chart, as shown in Listing 5.10.

Listing 5.10 Using the new column as the bars colors

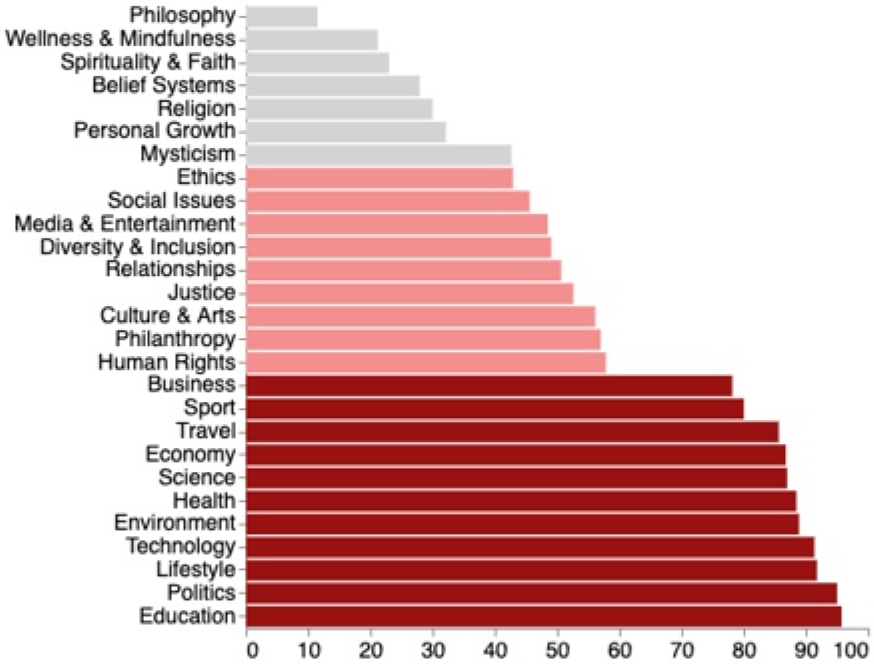
```
chart = alt.Chart(df).mark_bar(
).encode(
    y=alt.Y('Category:N',
            sort='x',
            title=None,
            axis=alt.Axis(labelFontSize=14)
            ),
    x=alt.X('Percentage of Fake Articles:Q',
            title=None,
            axis=alt.Axis(labelFontSize=14,
                          titleFontSize=14),
    ),
    color=alt.Color('Macro Category:N',
                    scale=alt.Scale(
                        range=['#991111', '#f38f8f','lightgray'],
                        domain=['Material Life', 'Moral Life', 'S
                    ),
                    legend=None
            )
).properties(
    width=400,
    height=400
).transform_calculate(
    'Percentage of Fake Articles', alt.datum['Number of Fake Arti
)
```

How to add color to the chart

Add the color channel to set the bars color.

Figure 5.22 shows the resulting chart. We have used two tonalities of red to highlight the urgency of paying attention to material and moral life.

Figure 5.22 The chart of Figure 5.21 with macro categories highlighted

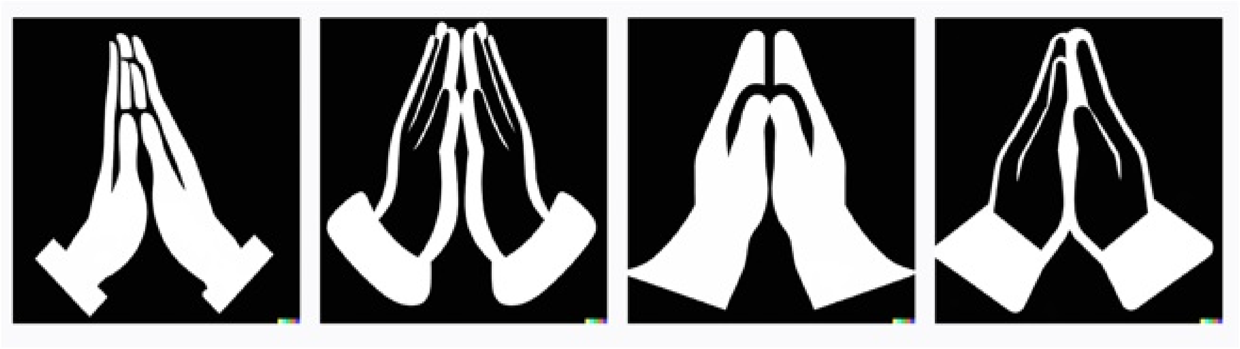


We could improve the chart by simplifying the chart, for example by grouping categories. However, our audience comprises experts (editors), who need very detailed information because they must know precisely which categories they must analyze more deeply. For this reason, we leave the chart with many details. We have removed the legend from the chart because we want to replace them with some images, which also act as the context for our story. Let’s see how to add these images, in the next step, turning information into knowledge.

5.5.2 From Information into Knowledge

The idea is to add an image for each macro category, more specifically an icon. We can use DALL-E to generate the three images. Use the following prompt to generate the spiritual life icon: a black-and-white icon with praying hands. DALL-E will generate four images, such as those shown in Figure 5.23.

Figure 5.23 The images generated by DALL-E related to the spiritual life



Similarly, you can generate icons related to the other categories. We have used the following prompts:

- A black and white icon with a balance symbol of moral life
- A black and white icon with a circle representing the world.

If you don’t have any ideas on how to generate an image, you can use ChatGPT to get some ideas. For example, you can write the following prompt in ChatGPT: how would represent material life through an icon? Among the other suggestions, ChatGPT suggests the following: A circle to represent the world or universe, with various smaller icons or symbols placed within it to represent different facets of material life.

Once you have generated the images, you can incorporate them into the chart. Listing 5.11 describes how to add the spiritual life image. You can adopt the same strategy also for the other images. Also, remember to load the generated HTML chart in a web server.

Listing 5.11 Adding an icon to the chart

```
# Add an image to the chart
spiritual_image = alt.Chart(
    pd.DataFrame({'image_url': ['media/spiritual-life.png']})
).mark_image(
    width=80,
    height=80,
).encode(
    url='image_url',
    x=alt.value(270),  # pixels from left
    y=alt.value(50)
)
```

How to add an icon to the chart

Calibrate the x and y positions manually based on how the chart appears.

Near the icon, add a text describing the macro category, as shown in Listing 5.12.

Listing 5.12 Adding a text to the chart

```
spiritual_text = alt.Chart(
    pd.DataFrame({'text': ['Spiritual Life']})
).mark_text(
    fontSize=30,
    color='black',
    align='center',
    baseline='middle',
    font='Monotype',
    fontStyle='italic'
).encode(
    x=alt.value(420),  # pixels from left
    y=alt.value(50),
    text='text'
)
```

How to add a text to the chart

Calibrate the x and y positions manually based on how the chart appears.

Finally, combine all the charts as shown in Listing 5.13.

Listing 5.13 Combining the images, the texts and the chart

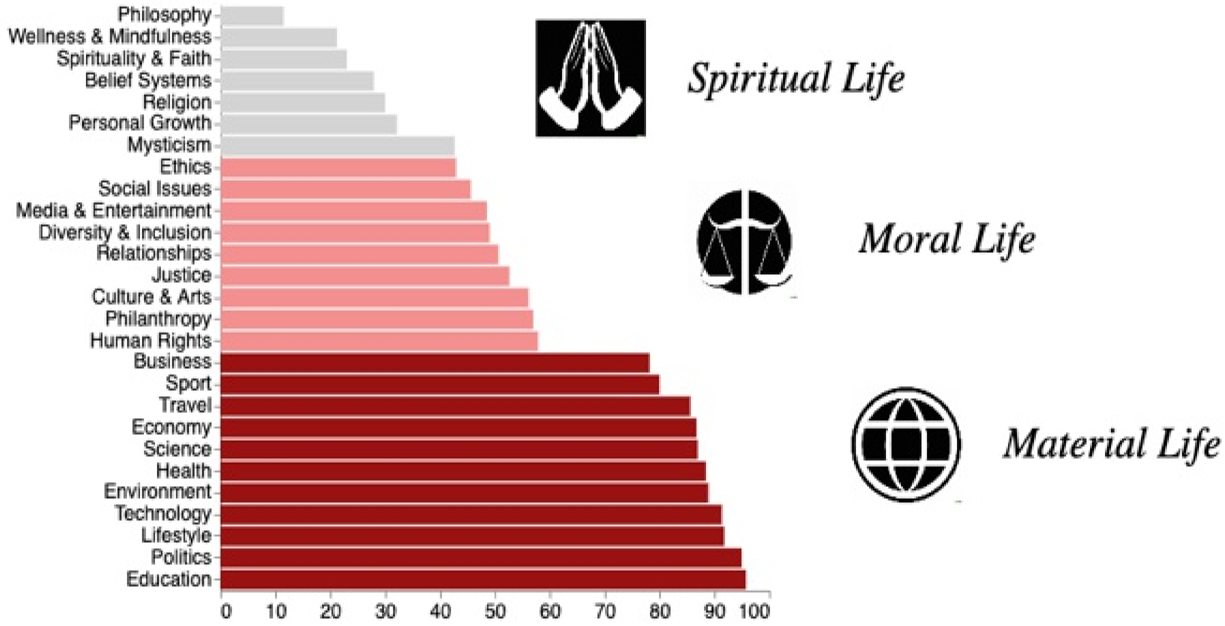
```
chart = chart + spiritual_image  + spiritual_text + moral_image +

chart = chart.configure_axis(
    grid=False
).configure_view(
    strokeWidth=0
)
chart.save('story-chart.html')
```

How to combine all the elements of the chart

Use the + operator to combine all the elements of the chart

Figure 5.24 shows the resulting chart.

Figure 5.24 The images generated by DALL-E related to the spiritual life



The icons and the text act as the legend. In addition, they are the characters of our story.

The next step involves adding a textual background to our chart and setting the context. We can add it as the subtitle of our chart. Let's use ChatGPT to generate some ideas. Write the following prompt:

Act as a storyteller. Describe the context of a visual chart showing that the highest number of fake news on the xx website is related to material life, followed by moral life and finally by spiritual life. Use max 150 characters.

We have specified three main elements:

- The ChatGPT role (we'll see this in the next chapter more in detail)
- The scenario
- The maximum number of characters to use.

A possible output produced by ChatGPT is the following: In the intriguing chart, depicting fake news trends on the XX website, material life claims the most prominent slice, trailed by moral life, and lastly, spiritual life. We can use it as it is, we can modify it, or we can even ask for a new subtitle. In our case, we'll use this prompt as the context of our chart.

Let's add this to our chart, by simply setting the subtitle in the chart properties, as shown in Listing 5.14.

Listing 5.14 Adding a subtitle to the chart

```
chart = chart.properties(width=500,title=alt.TitleParams(
    subtitle=['The XX website is a popular source of news and inf
    subtitleFontSize=18,
    fontSize=30,
    offset=40
))
```

How to add a subtitle to the chart

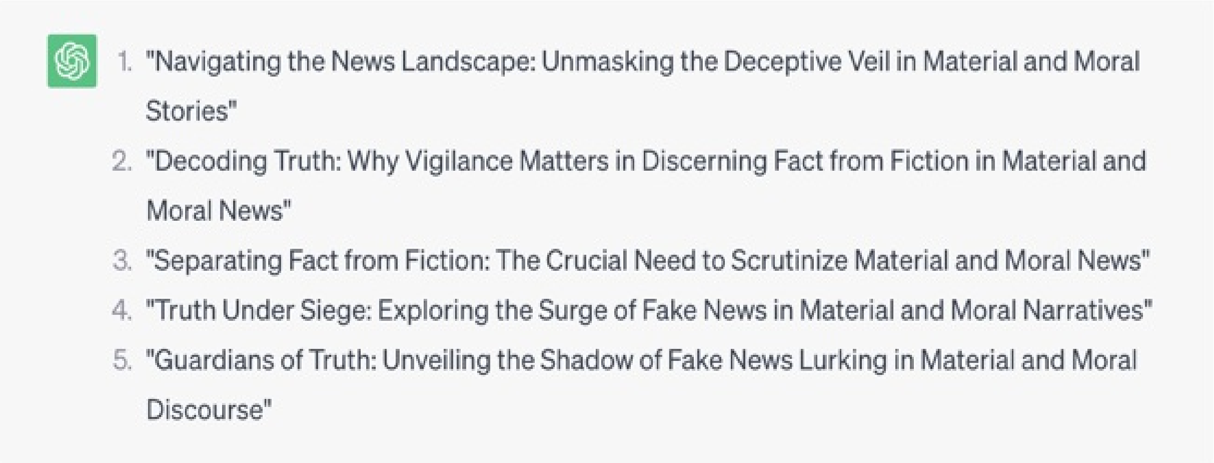
Use TitleParams() to specify the title and subtitle parameters

Now that we have turned information into knowledge, let's proceed with the last step, turning knowledge into wisdom.

5.5.3 From Knowledge into Wisdom

Wisdom involves adding a call to action to the story. In our case, we can simply say to the audience to pay attention to material and moral news because they have a high percentage of fake news. Let's use ChatGPT to generate an engaging title. Use the following prompt: Transform the following text into an engaging title: pay attention to material and moral news because they have a high percentage of fake news. Generate 5 titles. Figure 5.25 shows a possible output generated by ChatGPT.

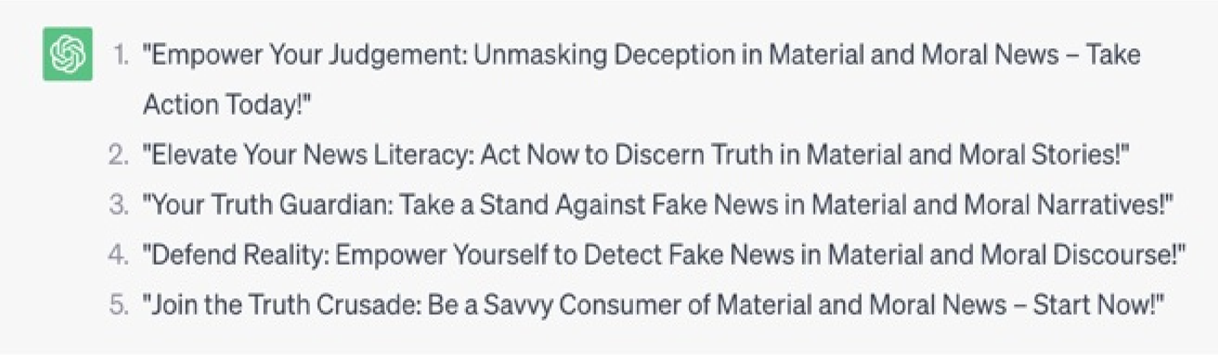
Figure 5.25 A possible output generated by ChatGPT



1. "Navigating the News Landscape: Unmasking the Deceptive Veil in Material and Moral Stories"
2. "Decoding Truth: Why Vigilance Matters in Discerning Fact from Fiction in Material and Moral News"
3. "Separating Fact from Fiction: The Crucial Need to Scrutinize Material and Moral News"
4. "Truth Under Siege: Exploring the Surge of Fake News in Material and Moral Narratives"
5. "Guardians of Truth: Unveiling the Shadow of Fake News Lurking in Material and Moral Discourse"

The generated titles do not contain a call to action. To add it, let’s improve the title, by writing the following new prompt for ChatGPT, in the same chat: *Transform the titles into a call to action that invites the audience to pay attention to news related to material and moral life*. Figure 5.26 shows a possible output produced by ChatGPT.

Figure 5.26 A possible output generated by ChatGPT when asking to add a call to action



1. "Empower Your Judgement: Unmasking Deception in Material and Moral News – Take Action Today!"
2. "Elevate Your News Literacy: Act Now to Discern Truth in Material and Moral Stories!"
3. "Your Truth Guardian: Take a Stand Against Fake News in Material and Moral Narratives!"
4. "Defend Reality: Empower Yourself to Detect Fake News in Material and Moral Discourse!"
5. "Join the Truth Crusade: Be a Savvy Consumer of Material and Moral News – Start Now!"

For example, select the third title *Your Truth Guardian: Take a Stand Against Fake News in Material and Moral Narratives* and add it to the chart, simply setting the text parameter, as shown in Listing 5.15.

Listing 5.15 Adding a title to the chart

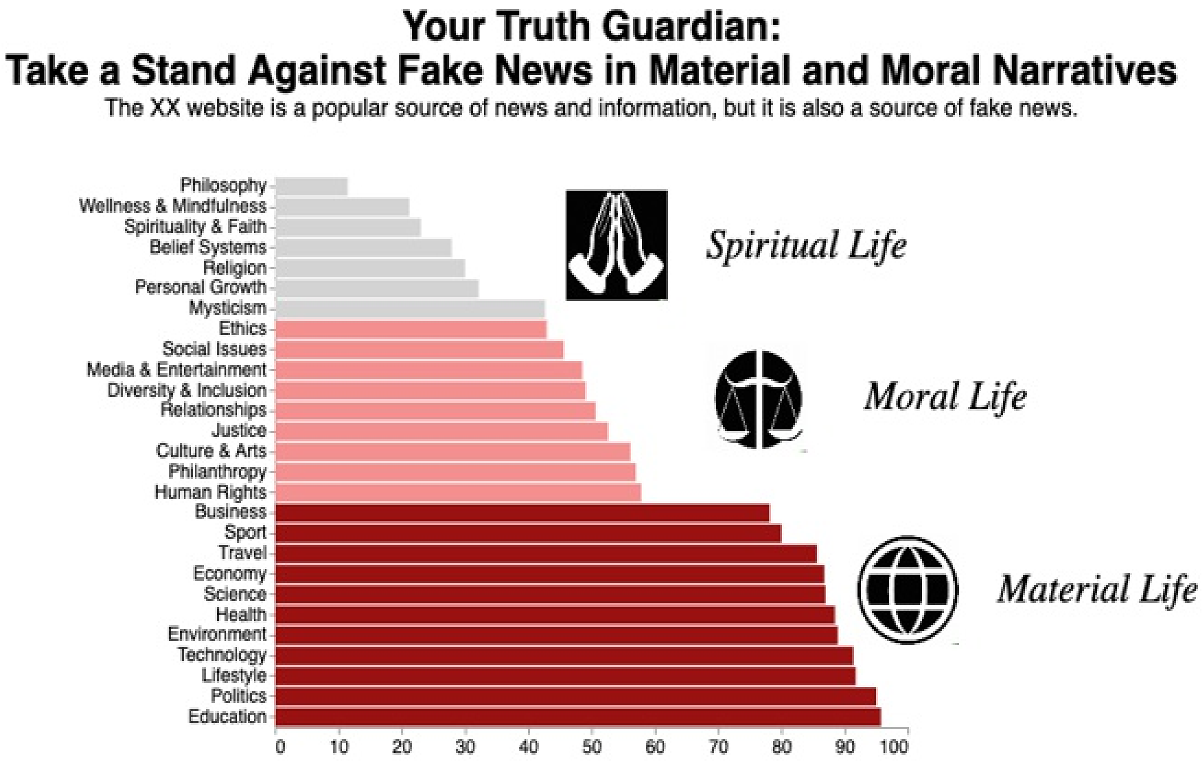
```
chart = chart.properties(width=500,title=alt.TitleParams(
    text=['Your Truth Guardian:', 'Take a Stand Against Fake News
    subtitle=['The XX website is a popular source of news and inf
    subtitleFontSize=18,
    fontSize=30,
    offset=40)
)
```

How to add a title to the chart

Use the text parameter to add a title to the chart

Figure 5.27 shows the final chart.

Figure 5.27 The final chart



In this chapter, you have learned the main concepts of building a data story using the DIKW pyramid and how to incorporate Generative AI tools in the flow. We've had a high-level overview of implementing the DIWK pyramid as part of our data story, as well as incorporating generative AI tools to enhance things further. In the next few chapters, we will look at traversing the DIWK pyramid in greater detail, starting with moving from "data" to "information".

5.6 Summary

- A story is a narrative that engages its audience to communicate a message and move them to action.
- The Data Storytelling Arc identifies five moments in a story: background, rising interest, main point, audience thinking, and next steps.
- Use the DIKW pyramid to implement practically the Data Storytelling Arc.
- Use Generative AI to assist you when scaling the DIKW pyramid, ensuring to drive it to produce what you want (text or image).

5.7 References

Dykes, B. (2019). Effective data storytelling: how to drive change with data, narrative and visuals. John Wiley & Sons.

Wikipedia. Narrative. https://en.wikipedia.org/wiki/Narrative (Last Access 2023/05/08)

[1] http://dati-censimentipermanenti.istat.it/Index.aspx?DataSetCode=DCSS_SENZA_TETTO

6 From Data to Information: Extracting insights

This chapter covers

- Introducing insights
- Choosing the characters of your story
- Choosing the right chart

In this chapter, first, we will focus on how to extract insights from data. An insight is something that is significant in data and needs to be communicated through a story. Next, we'll describe how to choose the characters of a data story and how to choose the right chart. The story characters and the right chart help to communicate your insight to an audience. Finally, you’ll learn how to implement some of the most popular data visualization charts in Altair. We'll exploit the power of Copilot to speed up the chart construction process. We'll provide the code to implement decluttered charts, ready to be added to a data story.

6.1 Introducing Insights

In the book *Journey to the Center of the Earth* by Jules Verne, Professor Otto Lidenbrock meticulously extracts information from a medieval Icelandic manuscript by employing his keen intellect and passion for exploration. With a deep understanding of languages, history, and geology, he deciphers cryptic clues, cross-references ancient texts, and applies scientific knowledge to unravel the secrets concealed within the pages. The manuscript provides details about the journey's route and the geological landmarks they would have encountered along the way. In other words, Professor Otto Lidenbrock turns the data contained in the old manuscript into information.

Similarly to the story told by Jules Verne, to tell any data story you can start by turning data into information. In the previous chapters, you have learned that to turn data into information you must extract insights from data. You have also learned that an insight is something relevant you have found in your data. In this book, we will not focus on how to extract insights from data. Tons of books exist on the topic, so you can refer to them for a detailed description (Guja et al. 2024, De Mauro et al. 2021). Anyway, in the remainder of this section, we will describe some strategies that can help you transform your data into information.

In his book *Seeing what others don't* (Klein 2017), Gary Klein identifies four main strategies to help you to identify an insight:

- Connections
- Coincidences
- Curiosity
- Contradictions.

Let's analyze each of the four proposed strategies separately, starting with the first: Connections. Keep in mind that you can apply multiple strategies at the same time.

6.1.1 Connection Strategy

This strategy involves identifying the main points of your data and then connecting them to identify a story. In his book, Klein says that sometimes you extract an insight when you see a new way to combine different pieces of data. Other times, you can extract an insight simply by looking at the pieces of data you already have in a different way.To explain how the connection strategy works, consider the following scenario. Angelica is a data analyst working for an e-commerce website. One day, Angelica receives a report on customer reviews and ratings for each product on the website. Each product is rated on a scale of 1 to 5, and customers can leave textual reviews. Table 6.1 shows the sales dataset Angelica must analyze.

Table 6.1 The sales dataset analyzed by Angelica

product_id	number_of_orders	product_rating	product_category	returns
P1001	30	4.5	Electronics	0

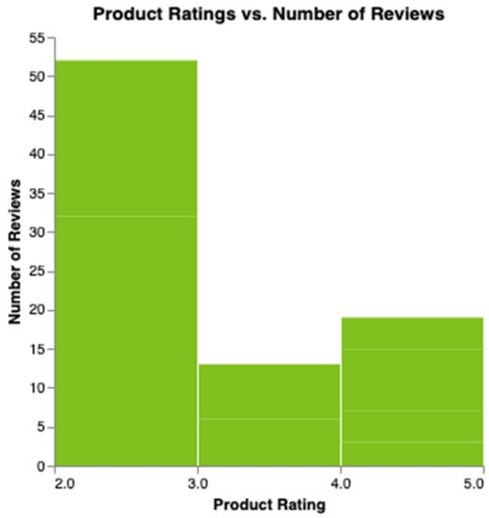
P1002	12	3.2	Home & Kitchen	6
…	…	…	…	
P1006	24	4.2	Electronics	0
P1006	22	2.1	Electronics	18

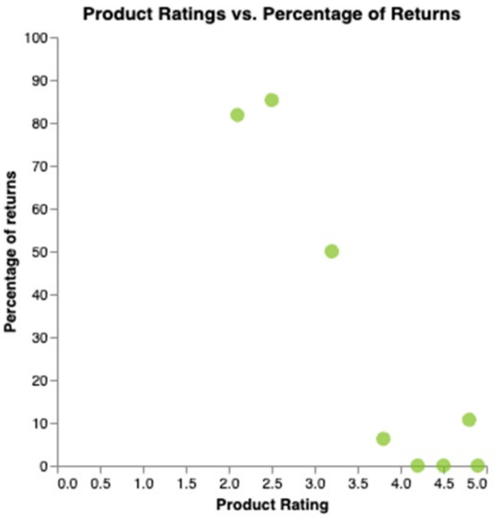
Here is an explanation of the dataset columns:

- **Product_id**: the unique identifiers for each product in the dataset.
- **Number_of_orders**: the quantity of times a particular product has been ordered.
- **Product_rating**: the rating assigned to each product by customers or users. Ratings could be on a scale (e.g., 1 to 5 stars) and reflect the satisfaction or perceived quality of the product.
- **Product_category**: This column categorizes products into different groups or types.
- **Returns**: the number of times a product was returned by customers. It provides insight into the rate of customer dissatisfaction or issues with products that lead to their return.
- **Number_of_reviews**: the count of reviews received for each product.

Angelica starts by plotting the product ratings vs. the number of reviews, and the product ratings vs. percentage of returns (Figure 6.1). You can find this example in the GitHub repository of the book under 06/connections.

Figure 6.1 The charts drawn by Angelica





Angelica discovers the following main points:

- Some products have high ratings but relatively fewer reviews
- Others have lower ratings but a substantial number of reviews

Angelica uses her intuition to connect the points and discovers that the negative reviews are connected to the high return rates. This may hurt the overall customer experience and brand reputation.

The previous example demonstrates that once we have identified the dots, extracting insights is just connecting them. However, the main problem with this strategy is identifying the *non-dots*, i.e., the irrelevant messages. An example of non-dot in the previous scenario is the product category vs. the product rating..

6.1.2 Coincidence Strategy

A coincidence is an occurrence of events that happen at the same time, by accident but seem to have some connection. It is the unexpected convergence of two or more appearing significant situations, even though they may be disconnected. Coincidences can range from everyday events, like meeting someone you know in an unexpected place, to extraordinary events that seem incredibly unlikely. However, while coincidences are intriguing, they do not necessarily indicate anything beyond chance. They could be rare or unusually timed events capturing our attention due to their curious alignment.

In his book, Gary Klein says: *“Observing a coincidence means that we’ve spotted some events that seem related to each other even though they don’t seem to have any obvious causal link.”* You should take a skeptical approach and consider something a coincidence (and thus ignore it) when it is an isolated incident with few data points to evidence a relationship between two variables. On the other hand, repeated events or trends should be considered less likely to be coincidental, and thus worth investigating further.

To identify whether a repeated event is a coincidence or not, you could approach data with a deliberately skeptical mindset, ignoring isolated events while looking for longer-term patterns that would suggest a relationship. The presence of correlated factors could indicate a potential causal relationship. For instance, by analyzing data over time, you might discover that certain variables precede the repeated event, hinting at a cause-and-effect dynamic. Through causality you could distinguish between mere coincidences and meaningful connections, explaining the mechanisms governing the observed phenomenon.

Let’s consider the following example to explain how we could apply the coincidence strategy. Imagine that Angelica works for a store selling accessories for electronic devices, such as headphones, USB cables, smartphone covers, and so on. Also, imagine that Angelicawants to monitor your orders. You have the dataset described in Table 6.2.

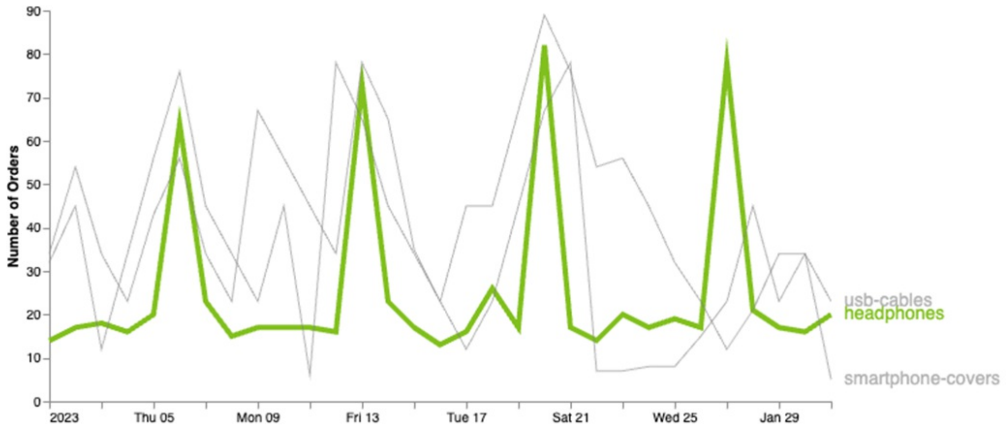
Table 6.2 The sales dataset of the electronic store

Date	Headphones	Usb-cables	Smartphone-covers
2023-01-01	14	34	32
2023-01-02	17	54	45
…	…	…	…
2023-01-30	16	34	34
2023-01-31	20	23	5

For simplicity, the dataset contains the number of orders for three products:

headphones, USB cables, and smartphone covers related to the period 1-31 January 2023. Figure 6.2 shows a visual representation of the dataset. You can find the complete code in the GitHub repository of the book, under 06/coincidences.

Figure 6.2 The electronic store orders, with a focus on headphones



The figure highlights the headphones orders. It’s worth noting that headphone orders peak every Friday. This could be a coincidence, but the repetitiveness of the event should be considered. For example, Angelicamight find that every Friday, there is an exciting radio show that causes people to rush to buy new headphones. Or she might find out that there's a law in your town on Fridays that doesn't allow the speakers to be turned on. In any case, this repetitiveness of events requires further analysis.

Looking at the x-axis of Figure 6.2, you can notice a strange labeling strategy: first the year, 2023, then the week days, and finally the month (Jan 29). This is the default Altair labeling for dates. You can read more details on how to format dates in the Altair official documentation.

Coincidence insights differ from connection insights in the way we discover them. In connection insights, we focus on important details, while in coincidence insights, we focus on repeated events. In addition, connections are helpful when you have aggregated metrics, while coincidence is best applied to things like raw sales numbers over time.

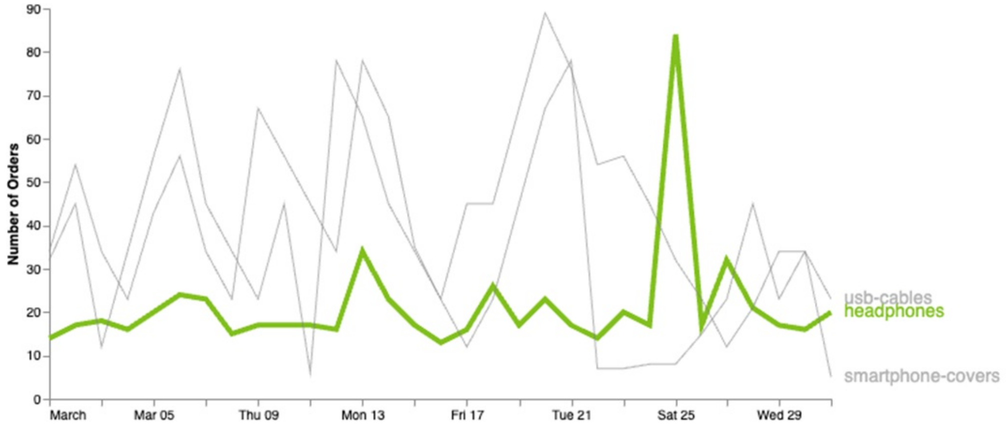
6.1.3 Curiosity

In 1928, Sir Alexander Fleming, a Scottish bacteriologist, discovered penicillin while studying staphylococci bacteria. One day, while he was meticulously examining his petri dishes filled with these microorganisms, he noticed something peculiar. A petri dish left open by mistake had become contaminated with mold. Intrigued by this unexpected occurrence, Fleming examined the petri dish closely. To his astonishment, he observed that the bacteria surrounding the mold were gradually deteriorating. Something unleashed by the mold was defeating them. Driven by curiosity, Fleming embarked on a series of experiments to explore this phenomenon. He carefully isolated the mold and extracted the substance it produced, which he named penicillin. Curiosity prompted Fleming to investigate the problem further, leading him to the discovery of penicillin,, the world's first antibiotic, saving countless lives and transforming medicine forever.

Curiosity is a driving force that sparks wonder, propels discovery, and fuels innovation, pushing us to question, seek answers, and expand the boundaries of knowledge. Curiosity is one strategy we can apply to extract insights. Curiosity in data can be triggered by a wide range of phenomena, including outliers, missing data, data gaps across time or space, sudden shifts or trends, unexpected patterns, and much more. When we note something curious in our data, we dig deeper, ask questions, and explore the underlying factors contributing to the observed patterns. This curiosity drives us to analyze the data from different angles, uncover hidden insights, and ultimately gain a deeper understanding of the phenomena.

Let’s consider the following example to explain how we could apply the curiosity strategy. Consider again the example of the electronic devices of the previous section. Imagine now, that you have a different trendline for March, as shown in Figure 6.3.

Figure 6.3 The electronic store orders in March, with a focus on headphones



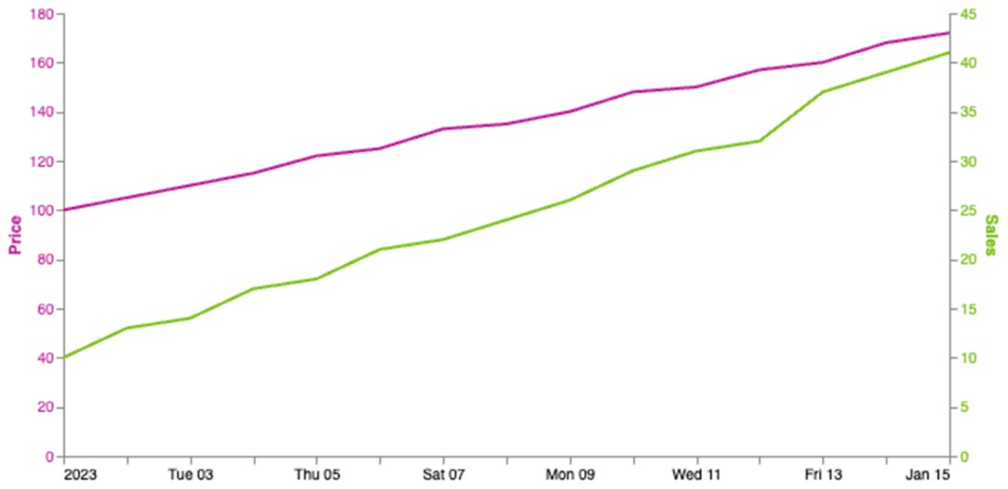
The figure shows a peak in sales on March 25th. This event may raise some curiosity, which moves us to further investigation. Curiosity could help us during the brainstorming process. For example, it could lead us to search for various factors that could have contributed to this spike, such as marketing efforts, special promotions, or external events. As we proceed in the research, curiosity may spur us to add other factors, such as customer demographics during this period. This investigation will not only satisfy our curiosity but also help us to extract insights.

6.1.4 Contradictions

Contradiction is a situation when two or more beliefs or pieces of evidence conflict, creating an inconsistency. According to Gary Klein, to extract insight from contradictions, you must approach the data with a skeptical mindset. Unlike a coincidence-based approach, where we look for repeating events, in the case of a contradiction-based strategy we look for seemingly conflicting events. An insight extracted from contradictions can disrupt conventional thinking, challenge established notions, and provide alternative perspectives that foster creative problem-solving. By embracing the inherent tension within contradictions, you can gain valuable insights, leading to breakthroughs, innovation, and growth.

Consider the chart in Figure 6.4, showing product sales and prices over time. You can find the code of the produced chart in the GitHub repository of the book under 06/contradictions.

Figure 6.4 A chart showing product sales and price



The chart shows that a price increase corresponds to an increase in sales. This event is quite contradictory because the general assumption is that a price increase should correspond to a decrease in sales. However, to understand this phenomenon, you could wear the skepticism lens and analyze this contradictory phenomenon more deeply. One possible motivation you could discover is the perceived value. When prices increase, customers may perceive your product as more exclusive, high-quality, or desirable. This perception of increased value can make customers willing to pay more, leading to increased sales. An alternative explanation could be the sense of urgency among consumers who may fear missing out on the product or believe that the higher price signifies scarcity or limited availability. Another reason may be inflation. In the context of inflation, when prices rise people may buy more for fear that the prices may rise even more, making the product unattainable.

All the proposed strategies, connections, coincidences, curiosity, and contradictions define some possible techniques to extract meaning. Regardless of which approach you take, always rely on your experience and a deep exploratory analysis phase to extract meaning from data.

Now that you have learned how to extract insight from data let’s move on to how to choose the characters of your story.

6.2 Choosing the Characters of Your Story

In the movie *Indiana Jones and the Raiders of the Lost Ark*, Indiana Jones, a brilliant archaeologist embarks on a thrilling adventure pursuing an ancient artifact, the Ark of the Covenant. Set in the 1930s, Indiana races against time to find the sacred relic before the Nazis seize it for their wicked plans. This movie and the others of the Indiana Jones saga are examples of engaging stories where the hero wants something.

Every story starts with a hero wanting something. Without any hero and something to achieve, there is no story. The same should be in your data story. It might seem that the term hero is too much to define the subject of our story. However, in our case we really want to emphasize the similarity between a data story and a real story. In any story there is always a hero who corresponds to the main character. Our idea is to see the subject to be represented just like the hero of our story, even if it is a simple product. Using this point of view, it will also be easier for us to construct a story.

- Every data story should have a *hero*: a product, a customer, or whatever.
- The hero must be *wanting something*: a sales increase, an increased satisfaction, or anything else.

Keep in mind that the hero of the story is something other than you. The hero is the main subject of your data story. It depends on the insight you have extracted. Examples of heroes are people, things, and places.

Once you have identified the hero of your story, the next step is asking the audience the following question: Will the hero get what they want? In his book *Building a StoryBrand: Clarify Your Message So Customers Will Listen*, Donald Milligan says that *before knowing what the hero wants, the audience has little interest in her fate. This is why screenwriters have to define the character’s ambition within the first nine or so minutes of a film getting started.*

In addition to having a hero, your story should have at least two other characters: the guide and the adversary. The guide, often a wise and experienced mentor, helps the hero achieve his goal. The adversary can be:

- A competitor who seeks to hinder or oppose the hero's progress
- A comparison character, which allows you to measure the hero's

progress

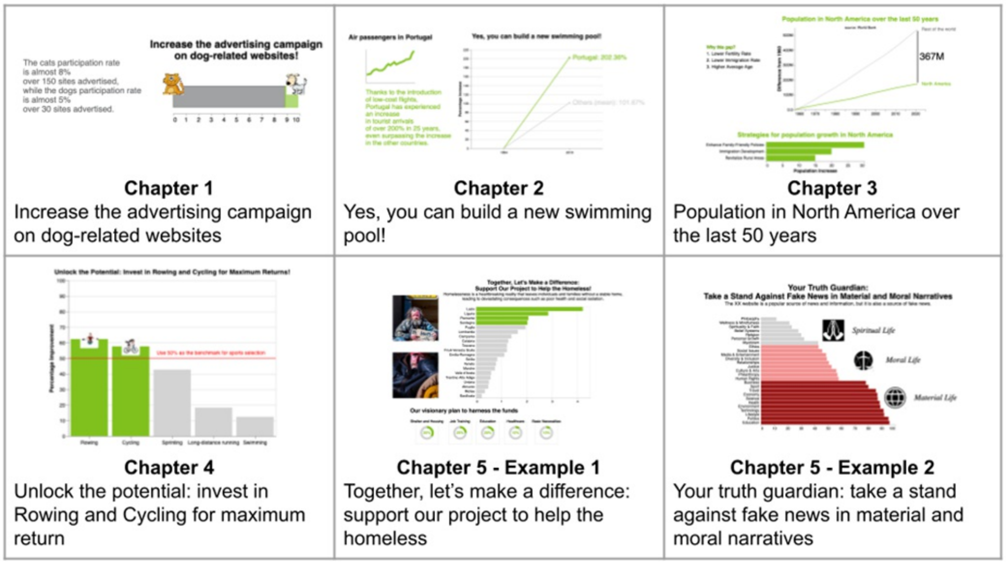
- The same hero at a different time (e.g. comparing the hero's condition now to a year ago) or in a different space (e.g. comparing the hero here to another location).

Let's review all the examples we have seen in the previous chapters and identify the hero of each story. For convenience, Figure 6.4 summarizes the examples analyzed.

Challenge

For each example described in Figure 6.4, search for the hero, the type of hero, what they want, and the adversary.

Figure 6.5 A summary of all the examples described in the previous chapters



Chapter 1
Increase the advertising campaign on dog-related websites

Chapter 2
Yes, you can build a new swimming pool!

Chapter 3
Population in North America over the last 50 years

Chapter 4
Unlock the potential: invest in Rowing and Cycling for maximum return

Chapter 5 - Example 1
Together, let's make a difference: support our project to help the homeless

Chapter 5 - Example 2
Your truth guardian: take a stand against fake news in material and moral narratives

Table 6.3 shows the hero and the adversary for each case study shown in Figure 6.5. For each case study, the table also shows the type of hero and what the hero wants. In the example of Chapter 2, *Yes, you can build a new swimming pool!* There is the implicit assumption that an increase in tourist arrivals corresponds to the possibility of building a new swimming pool. This is, obviously, a simplistic assumption, but we simplified the example just to get started and show how the DIKW pyramid works.

Table 6.3 The hero and adversary of the case studies analyzed in the previous chapters

Case Study	Hero	Type of hero	What the hero wants	Adversary
Chapter 1 Increase the advertising campaign on dog-related websites	Dog-related websites	Thing	Increase the number of views	Cat-related websites
Chapter 2 Yes, you can build a new swimming pool!	The flow of tourist arrivals to Portugal	Process	An increase in tourist arrivals	The flow of tourist arrivals to Germany
Chapter 3 Population in North America over the last 50 years	Population in North America	People	Explain the trend	The population in the rest of the world
Chapter 5 - Example 1 Together, let’s make a difference: support our project to help the homeless	Number of homeless people in Lazio, Piemonte, Liguria, and Sardegna	People	Reduce this number	Number of homeless people in the other Italian regions
Chapter 5 - Example 2 Your truth guardian:	News about the material and moral life	Thing	Establish whether they are genuine or fake	News about spiritual life

take a stand against fake news in material and moral narratives				

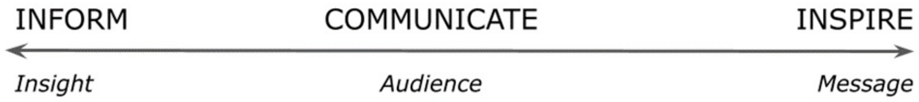
In all the examples shown in Table 6.3, the adversary is always a character different from the hero, although it is of the same type as the hero. In some cases, the adversary can be the same hero in a previous condition, such as a product sales in the past or into another geographical position.

Now that you have learned how to choose the characters of your story, let’s move on to how to choose the right chart for your data story.

6.3 Choosing the Right Chart

The best chart depends on the insights you want to represent, the message you want to convey, and the audience you are targeting. Figure 6.6 shows how insight, message and audience relate each other in a chart. If you only consider the insight, the effect of your chart is to inform.

Figure 6.6 The relationship among insight, audience and message.



When telling a story, you have three levels:

- **Inform** - in this case, you only describe the insight you have found in your data. No contact is made with the audience.
- **Communicate** - you involve the audience, by providing them details about your data, but the audience is not motivated to do something after

listening to your story.
- **Inspire** - based on your data, you propose a message, which inspires the audience to do something. The audience is plenty involved and will retell your story in other contexts.

In other words, if you only consider the insight, the effect of your chart is to inform. If you calibrate your insight to your audience, you can communicate it effectively. Finally, if you add a message to your chart, you can inspire your audience. We will see how to represent insights in the remainder of this chapter, how to calibrate the chart to your targeted audience in Chapter 7, *From Information to Knowledge: Building Context*, and how to add a message to a chart in Chapter 8, *From Knowledge to Wisdom: Adding Next Steps*.

Let’s focus on how to represent insights in a chart. The choice of a chart is driven by your specific goals. Different types of charts serve different purposes based on the nature of the data and the message you want to convey. Table 6.4 describes which chart to use based on the information you want to convey. Consider that the list of described charts is not exhaustive, and you can use your imagination and expertise to build new charts. In addition, you can follow the Dr. Abela’s chart chooser[1] to select the best chart for you.

Table 6.4 Suggested charts based on the information to convey

Information to convey	Description	Suggested Charts
A single piece of information	A single number representing critical information	- Big Number (BAN) - Donut Chart - 100% Stacked Bar Chart - Waffle Chart
Parts of a whole	The components contributing to the entirety of a system, object, or concept. The sum of all components must be 100%	- Pie Chart - 100% Stacked Bar Chart - Multiple Waffle Chart - Donut Chart

Comparison among entities	The similarities and differences between multiple entities to establish relationships and distinctions	- Bar Chart - Column Chart - Slope Graph - Dumbbell Chart - Table
Trend	The behavior of an entity over the time	- Line Chart - Small Multiple Line Chart - Stacked Area Chart - Stacked Column Chart
Outcomes of a survey or a questionnaire	Answers to questions contained in a survey or a questionnaire	- Stacked Bar Chart - Column Chart - Multiple Bar Charts
Distribution	Spread of values across a dataset, indicating how frequently different values occur	- Histogram - Pyramid - Box Plot
Relationship	Association, connection, or correlation between different entities to identify patterns, trends, and dependencies	- Scatterplot - Bubble Chart - Heatmap
Spatial information	The behavior of an entity over the space	- Choropleth Map - Dot Density Map - Proportional Symbol Map - Heatmap
Flow	Represent a process	- Sankey* - Chord* * not supported by Altair 5.0.1

As you can see from the table, you can use the same chart for different purposes. In addition, you can use many charts for the same purpose. The choice of a specific chart depends on the audience, as we will see in Chapter 7, *From Information to Knowledge: Building Context*. In the remainder of this section, we’ll describe how to implement some of the most important charts in Altair and Copilot, grouped by chart family. We’ll focus on the following chart families:

- Cooking charts
- Bar charts
- Line charts
- Geographical maps
- Dot charts

Let’s start with the first family, the cooking chart family. You can find the code associated with each chart in the GitHub repository of the book, under section 06.

6.3.1 Cooking Charts Family

The cooking charts family includes the pie chart and the donut chart. Use this category of charts only to represent numerical values. Lately, I have seen on social media and the web in general an unfounded opposition to these types of charts. The main argument given is that these graphs are too generic and do not convey the results correctly. I was a victim of it, and for a while, I avoided this type of chart as well. Then, one fine day, on LinkedIn, I read a post[2] by Brent Dykes re-evaluating cooking charts and especially pie charts. In his post, Dykes pointed out that pie charts aren't great at representing the precise dimensions of the slices, but they do represent the parts of the whole. Dykes explicitly says: *“I know everyone likes rules. "Never use this chart type..." "Always use this chart type for...", etc. However, it's not always so straightforward in data storytelling. It isn't just about how to display a certain type of data but how to best communicate a specific point to your audience.”*

Following Dykes’ suggestions, you can use cooking charts in three main cases:

- You must represent the part of a whole. The total sum of the slices must be 100%
- There are at most 2-3 slices to represent (provided that they are not very similar in size, otherwise it is very difficult to differentiate which one is bigger than the others)
- You don’t want to transmit details about your data but only a general overview. This could be the case for a general audience, not including technical experts.

Let's investigate each type of chart separately, starting with pie charts.

Pie Chart

A pie chart is a type of circular data visualization that displays data by dividing a circle into slices, each representing a proportion or percentage of a whole.

Suppose we want to represent the number 70% through a pie chart. First, create a Pandas DataFrame containing your number and its complementary value (30%), as shown in Listing 6.1.

Listing 6.1 The DataFrame with the number

```
import pandas as pd
import altair as alt

data = {
    'percentage': [0.7,0.3],
    'label'     : ['70%','30%'], #A
    'color'     : ['#81c01e','lightgray'] #B
}

df = pd.DataFrame(data)
```

Creating a DataFrame with the number

Create a Pandas DataFrame with the number to represent and its complementary value.

Next, ask GitHub Copilot to draw the chart for you. A pie chart does not have the classical `X` and `Y` channels to encode data. Instead, the pie chart uses the `Theta` and `Color` channels. The `Theta` channel specifies the angular position of data points in a polar plot. The `Color` channel refers to encoding data using different colors.

Listing 6.2 shows the starting prompt for GitHub Copilot.

Listing 6.2 How to generate a basic pie chart in Copilot

```
# Draw a pie chart in Altair with the following options:
# - Use the `percentage` column for theta channel
# - Use the `label` column for tooltip
# - Use the `color` column for color
# Save chart to `chart` variable
# Save chart as 'pie-chart.html'
```

Creating a basic pie chart in Copilot

The basic instructions to draw a pie chart in Copilot.

As a result, Copilot will generate the code, shown in Listing 6.3.

Listing 6.3 The code to generate a basic pie chart

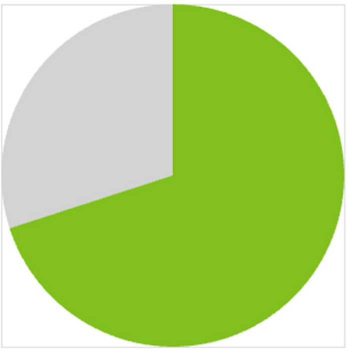
```
chart = alt.Chart(df).mark_arc(
).encode(
    theta=alt.Theta('percentage', stack=True), #A
    color=alt.Color('color', scale=None),
    tooltip='label'
).properties(
    width=300,
    height=300
)
chart.save('pie-chart.html')
```

Creating a basic pie chart

The code to generate a basic pie chart in Altair. Use the `mark_arc()` method to draw a pie chart. We have added the `stack=True` property manually for the next steps. This property means that the individual slices within the chart will be stacked on each other instead of being displayed side by side.

Figure 6.7 shows the resulting chart.

Figure 6.7 A basic pie chart in Altair



We can improve the chart by adding the label value next to each slice. Ask Copilot to draw a text containing the labels, as shown in Listing 6.4.

Listing 6.4 How to labels to the chart in Copilot

```
# Add text near to each slice of the pie chart
# - Use the `label` column for text channel
# - Use the `color` column for color

# Combine the pie chart and the text chart
# - Use `+` operator to combine the charts
# - Save the combined chart to `chart` variable
```

Creating labels using Copilot

The prompt for Copilot to generate labels next to the pie chart.

As a result, Copilot will generate a partial code that does not show the desired output. Modify it as described in Listing 6.5.

Listing 6.5 The code to generate labels

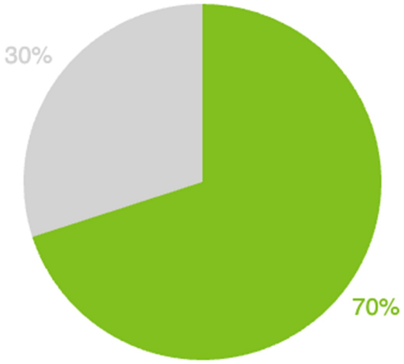
```
text = chart.mark_text( #A
    size = 20, #B
    radius=180 #C
).encode(
    text='label',
    color=alt.Color('color', scale=None)
).properties(
    width=300,
    height=300
)

chart = (chart + text
).configure_view(
    strokeWidth=0
) #D
```

Creating labels for the pie chart

The code to generate the labels for the basic pie chart in Altair. Use the `mark_text()` method to write the text.

Figure 6.8 shows the resulting chart.

Figure 6.8 A pie chart in Altair with labels



Now that you have learned how to draw a pie chart in Altair e Copilot, let’s move to the next chart of the pie chart family, the donut chart.

Donut Chart

A donut chart is a type of circular data visualization that displays data in a ring shape. It is similar to the pie chart but with a hole in the center, creating a visual representation of proportions or percentages of different categories.

To convert a pie chart into a donut chart, simply add the `innerRadius` and `outerRadius` properties to the `mark_arc()` method, as described in Listing 6.6.

Listing 6.6 The code to generate a donut chart

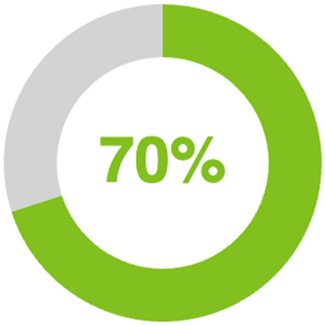
```
chart = alt.Chart(df).mark_arc(
    innerRadius=100,
    outerRadius=150
).encode(
    theta='percentage',
    color=alt.Color('color', scale=None),
    tooltip='label'
).properties(
    width=300,
    height=300
)
```

Creating a donut chart

The code to generate a donut chart in Altair. Use the `innerRadius` and `outerRadius` to transform a pie chart into a donut chart.

If your donut chart compares two values and you want to focus on just one value, add this value at the center of the donut, as shown in Figure 6.9.

Figure 6.9 A donut chart in Altair with a label in the center



Ask Copilot to generate the label within the donut, as shown in Listing 6.7.

Listing 6.7 The prompt to generate the label within the donut chart

```
# Add text to the center of the donut chart
# - Use df.head(1) to get the first row of the dataframe
# - Use the `label` column for text channel
# - Use the `color` column for color
```

Creating the label using Copilot

The sequence of instructions for Copilot to generate the label within the donut chart.

Listing 6.8 shows the generated code.

Listing 6.8 The code generated by Copilot

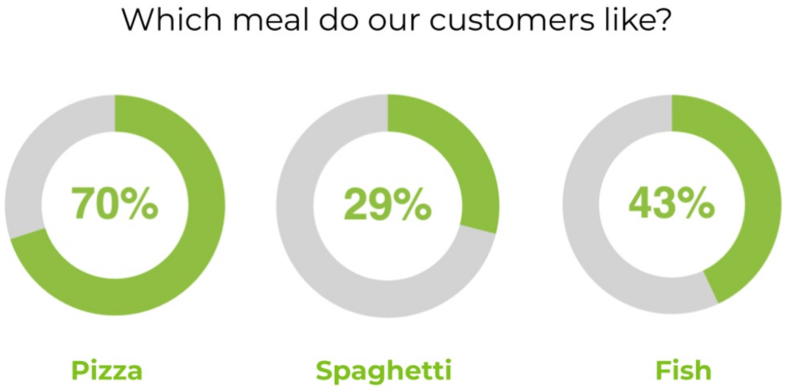
```
text = alt.Chart(df.head(1)).mark_text(
    align='center',
    baseline='middle',
    fontSize=60,
    fontWeight='bold'
).encode(
    text='label',
    color=alt.Color('color', scale=None)
).properties(
    width=300,
    height=300
)
```

Creating the label for the donut chart

Change the `fontSize` property to increase or decrease the label size.

From the data storytelling perspective, you can concatenate multiple donut charts, as illustrated in Figure 6.10, to show many percentage values, for example, related to the questionnaire output related to which meals the customers of a restaurant like. The total sum is greater than 100% because a customer may like more than one meal.

Figure 6.10 A possible usage of multiple donut charts to show the output of a questionnaire



In this particular scenario, a cumulative total of 100% might not be essential. However, it's worth noting that other business perspectives could potentially warrant such a requirement. Now that you have learned how to draw the cooking charts in Altair and Copilot, let’s move on to the next family of charts, the bar charts.

6.3.2 Bar Charts Family

The family of bar charts includes various types of charts where each data series is visualized as rectangular bars. These bars can be oriented either vertically, known as column charts, or horizontally, known as bar charts. In addition to the classical bar and column charts, this family also includes stacked column/bar charts, pyramid charts, and histograms. Use these charts to represent categorical data or compare different categories by displaying their corresponding values along a horizontal or vertical axis.

In a similar way to the backlash against pie charts, lately, on the web and in social media, I have noticed an excessive abuse of bar charts. While these charts have the advantages of simplicity and effectiveness in conveying a message, using them repeatedly with the same audience could bore the audience, and is likely a reason for the consternation directed at these charts.

You have already learned how to draw a bar chart and a column chart in the previous chapters. In this section, you’ll learn how to improve their readability thanks to Copilot. Let’s start with the bar chart.

Bar Chart

A bar chart is a type of chart that uses rectangular bars to represent data values, where the length of each bar corresponds to the quantity it represents. Use this chart to represent data across different categories.

Consider the dataset shown in Table 6.5, describing the number of likes for each type of meal.

Table 6.5 The dataset of the example

Meal Type	Number of Likes
Pizza	120
Burger	95
Pasta	80
Sushi	60
Salad	50
Steak	70
Tacos	90
Ice Cream	110
Curry	40
Sandwich	75

Listing 6.9 shows the code to generate a simple bar chart representing the previous dataset in Altair, and Figure 6.11 shows the produced chart.

Listing 6.9 The code to create a bar chart

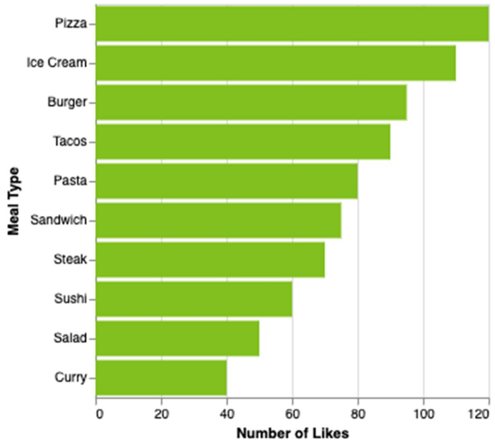
```
import pandas as pd
import altair as alt

df = pd.read_csv('data/meals.csv')
```

```
chart = alt.Chart(df).mark_bar(
    color='#81c01e'
).encode(
    y=alt.Y('Meal Type', sort='-x'),
    x='Number of Likes'
).properties(
    width=300,
    height=300
)

chart.save('bar-chart.html')
```

Creating a bar chart in Altair

Use the `color` property to set the bar color and the `sort` property of the `Y` channel to sort categories based on descending values of `X`.

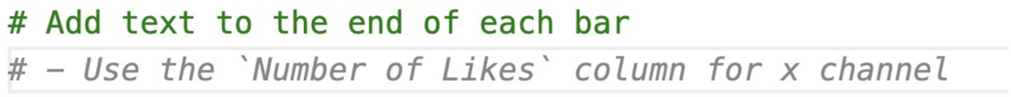
Figure 6.11 A bar chart



The chart needs to be decluttered so we should remove unnecessary elements, such as the y-axis title (Meal Type), the grids, and so on. For example, we can add the value of each bar at its end and remove the x-axis labels entirely. Use Copilot to help you build the chart. As you use Copilot, you will find that it will suggest more and more things that fit your coding style. If you start writing something, Copilot will suggest the next steps based on what you have written previously. This means that you can use Copilot not only with pre-defined prompts but also to suggest the next steps in your prompts.

To add the value of each bar at its end, start writing a prompt for Copilot, press enter, and you’ll see Copilot suggesting how to implement it, as shown in Figure 6.12.

Figure 6.12 If you start writing, Copilot will suggest the next steps



```
# Add text to the end of each bar
# - Use the `Number of Likes` column for x channel
```

Simply press enter, and Copilot will continue proposing how to add the text. Listing 6.10 shows the complete prompt proposed by Copilot.

Listing 6.10 The prompt generated by Copilot

```
# Add text to the end of each bar
# - Use the `Number of Likes` column for text channel
# - Use the `Meal Type` column for y channel and sort by the numb
# - Set the color of the text to '#81c01'
# - Set the text alignment to 'left'
# - Set the text baseline to 'middle'
# - Set font size to 14
```

Using Copilot to generate the next prompts

Use Copilot to generate the prompt to build your chart, and then, press enter to let Copilot generate the code

After the final prompt, press enter and Copilot will generate the code, as shown in Listing 6.11.

Listing 6.11 The code to generate labels

```
text = chart.mark_text(
    align='left',
    baseline='middle',
    dx=3,
    color='#81c01',
    fontSize=14
).encode(
    text='Number of Likes',
    y=alt.Y('Meal Type', sort='-x', title=None)
)
```

Adding labels at the end of each bar

Apply the `mark_text()` method to the chart to generate the labels. You can use the columns already set for the `chart` variable.

Now, add five vertical lines to the chart to ease the audience's comparison process. Ask Copilot to generate the code for you. Start writing the following prompt: *# Add five vertical lines to the chart*. Copilot will generate the prompt shown in Listing 6.12.

Listing 6.12 The prompt to generate vertical lines

```
# Add five vertical lines to the chart
# - Use alt.Chart(pd.DataFrame({'x': [20, 40, 60, 80, 100]})) to
# - Use alt.Chart().mark_rule() to draw vertical lines
# - Set the color of the lines to 'white'
# - Set the line width to 1
# - Set opacity to 0.5
```

How to generate the prompt to add vertical lines to a bar chart

Copilot generates the prompt to generate the code.

Press Enter, and Copilot will generate the code shown in Listing 6.13.

Listing 6.13 The code to generate vertical lines

```
lines = alt.Chart(pd.DataFrame({'x': [20, 40, 60, 80, 100]})).mar
    color='white',
    strokeWidth=1,
    opacity=0.5
).encode(
    x='x:Q'
)
```

Adding vertical lines to a bar chart

Use the `mark_rule()` mark to add vertical lines to the chart.

Finally, remove the x-axis from the original chart in Listing 6.9 (`x=alt.X('Number of Likes',axis=None)`), remove the title from the y-axis (`y=alt.Y('Meal Type', sort='-x', title=None)`), and combine the three charts, as shown in Listing 6.14.

Listing 6.14 Combining the charts to generate the final chart

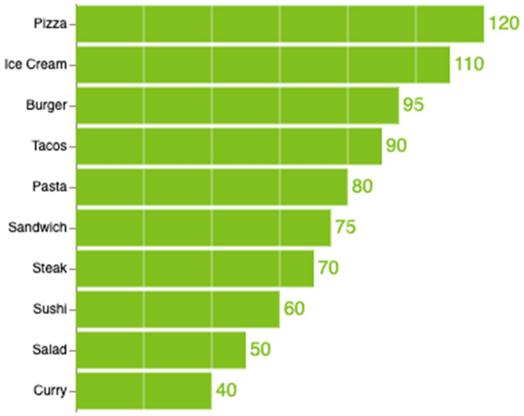
```
chart = (chart + text + lines
).configure_view(
    strokeWidth=0
).configure_axis(
    grid=False
)

chart.save('bar-chart.html')
```

How to combine the charts to generate the final chart

Use the + operator

Figure 6.13 shows the resulting chart.

Figure 6.13 A decluttered bar chart



Now that you have learned how to draw a decluttered bar chart in Atair e Copilot, let’s move on to the following chart, the column chart.

Column Chart

A column chart is similar to a bar chart, but it inverts axes: it shows categories on the x-axis and values on the y-axis. Since the x-axis is also used to represent temporal data, you can use a column chart to describe periods as categories. To draw a column chart in Altair, simply invert the X and Y channels in the code in Listing 6.9, as shown in Listing 6.15. Also, increase the chart width to 600 to leave more space for each column.

Listing 6.15 The code to create a column chart

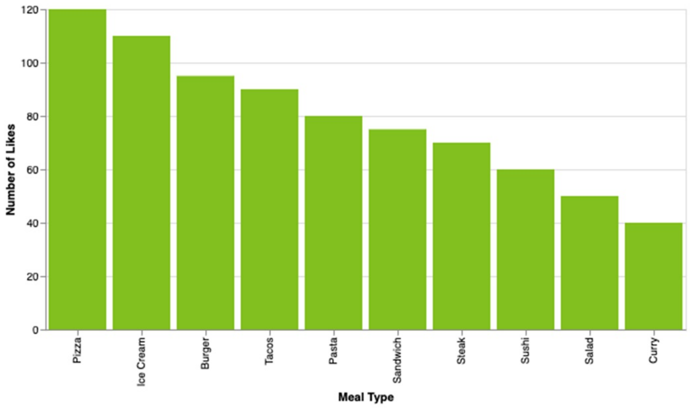
```
import pandas as pd
import altair as alt

chart = alt.Chart(df).mark_bar(
    color='#81c01e'
).encode(
    x=alt.X('Meal Type', sort='-y'),
    y='Number of Likes'
).properties(
    width=600,
    height=300
)

chart.save('column-chart.html')
```

Creating a column chart in Altair

Invert the `X` and `Y` channels. Also, change the `sort` property value.

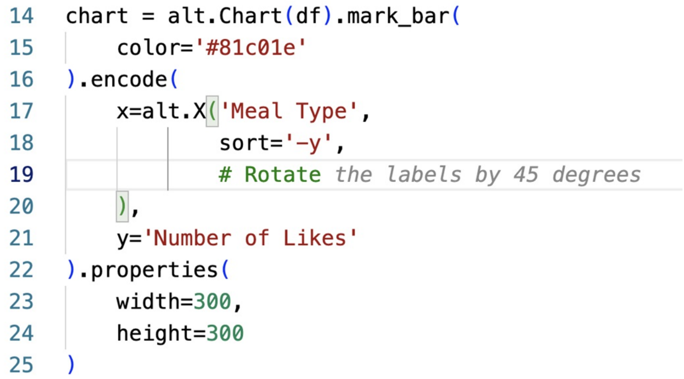
Figure 6.14 represents the generated column chart.

Figure 6.14 A column chart



The x-axis of the column chart shows rotated labels. Also, the x-axis title is unnecessary. Let’s proceed with decluttering. Start by formatting the code in the Visual Studio Code editor, then write the comment `# Rotate`, as shown in Figure 6.15. Copilot will suggest to you how to complete the sentence. Press the tab, change 45 degrees to 0 degrees, and press Enter, and Copilot will add the desired code in the row below your comment (`axis=alt.Axis(labelAngle=0),`).

Figure 6.15 How to add a comment within the code and Copilot will suggest the following steps



```
chart = alt.Chart(df).mark_bar(
    color='#81c01e'
).encode(
    x=alt.X('Meal Type',
            sort='-y',
            # Rotate the labels by 45 degrees
    ),
    y='Number of Likes'
).properties(
    width=300,
    height=300
)
```

Next, add a comma after `labelAngle=0` and start writing the word `title`. Copilot will suggest you `=None`. Press the Tab to confirm. This Copilot property is fantastic! You can add a comment at any point of your code and ask Copilot to generate new code for you! Since Copilot is a generative AI tool, it may happen that, in your case, the suggested prompts change. Anyway, you can easily adapt the code suggested by Copilot to your needs.

Now, let’s declutter the y-axis. Remove the y-axis by changing the `y='Number of Likes'` channel to `y=alt.Y('Number of Likes',axis=alt.Axis(grid=False))`. Then, add the number of likes at the top of each bar. Let’s use Copilot to do the job for us. Start writing the following text: `# Add a text mark to the chart with the following options:.` Copilot will suggest the next steps. Confirm them by pressing Tab and then Enter. At some point, Copilot stops suggesting. Press Enter, and Copilot will generate the code for you. Test the generated code. If it does not satisfy you, change it manually, or use Copilot again. In our case, we had to slightly change the Copilot comments to make it generate the desired code. Listing 6.16 shows the Copilot instructions.

Listing 6.16 The prompts generated by Copilot and slightly modified

```
# Add a text mark to the chart with the following options:
# - Use the `Number of Likes` column for y channel
# - Use the `Meal Type` column for x channel and sort by the numb
# - Set the color of the text to '#81c01e'
# - Set the text to the `Number of Likes` column
# - Set the font size to 14
# - Set the font weight to 600
# - Set the text baseline to 'center'
# - Set the text align to 'middle'
# - Set the y offset to 10
```

The prompt to add the label value at the top of each column

Ask Copilot to generate the code to draw labels at the top of each column

Listing 6.17 describes the produced code.

Listing 6.17 The code generated by Copilot to add labels

```
text = alt.Chart(df).mark_text(
    color='#81c01e',
    fontSize=14,
    fontWeight=600,
    baseline='middle',
    align='center',
    dy=-10
).encode(
    x=alt.X('Meal Type',
            sort='-y',
            axis=alt.Axis(labelAngle=0,title=None),
    ),
    y='Number of Likes',
    text='Number of Likes'
)
```

How to add label value to a column chart

Use the `mark_text()` method to add a label at the top of each column. Set the text properties, including color, font size, weight, etc.

Finally, combine the two charts, as shown in Listing 6.18.

Listing 6.18 Combining the charts to build a final chart

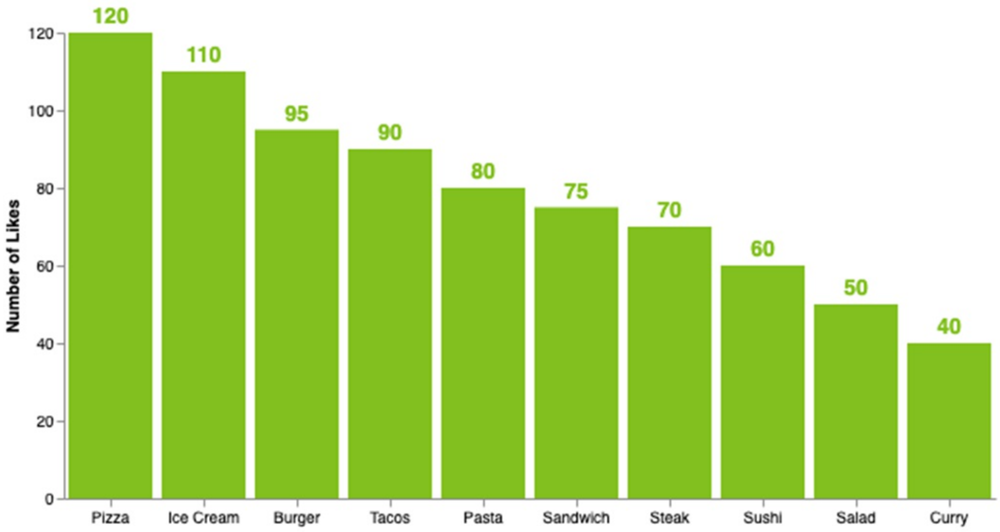
```
# Combine the bar chart and text mark into a single chart
chart = chart + text

chart = chart.configure_view(
    strokeWidth=0
)

chart.save('column-chart.html')
```

Creating the final chart in Altair

Write the prompt asking Copilot to generate the code to merge charts.

Figure 6.16 shows the final chart.

Figure 6.16 A decluttered column chart



Now that you have learned how to declutter a column chart in Altair and Copilot, let’s move on to the following chart, the column chart with multiple series.

A Column Chart with Multiple Series

So far, we have implemented a column chart (and also a bar chart) with just one series of data. However, in many cases, you want to compare two or more series of data. In this case, you must add the `column` channel to your chart.

Consider the dataset shown in Table 6.6, describing the number of likes for each type of meal for 2022 and 2023.

Table 6.6 A sample dataset with two series of data, one for 2022 and the other for 2023

Meal Type	Number or likes in 2022	Number of likes in 2023
Pizza	120	145
Burger	95	88
Pasta	80	97
Sushi	60	67

Salad	50	52
Steak	70	66
Tacos	90	78
Ice Cream	110	134
Curry	40	43
Sandwich	75	59

Listing 6.19 describes the code to build a multiple-series column chart.

Listing 6.19 How to create a multiple-series column chart

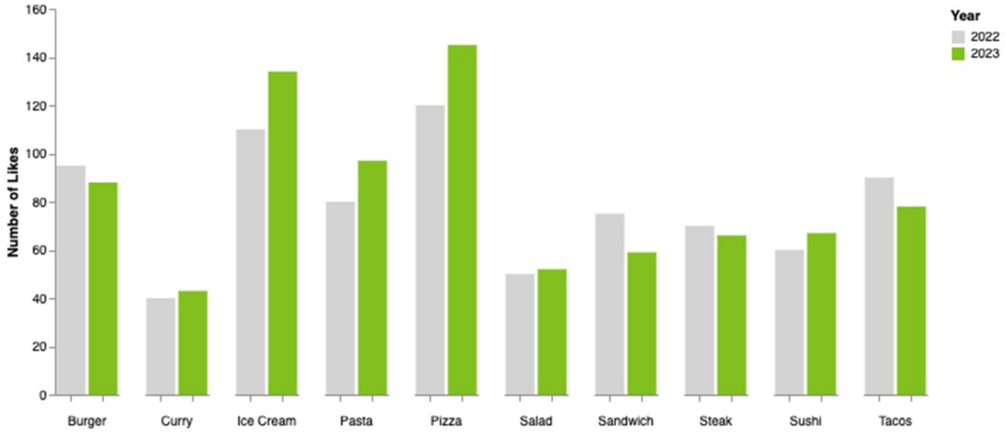
```
import pandas as pd
import altair as alt

# Load data data/meals.csv as a Pandas dataframe
df = pd.read_csv('data/meals-by-year.csv')

df = df.melt(id_vars=['Meal Type'],var_name='Year',value_name='Nu

chart = alt.Chart(df).mark_bar(
).encode(
    x=alt.X('Year',
            # Rotate the labels by 0 degrees
            axis=alt.Axis(title=None, labels=False)
    ),
    y=alt.Y('Number of Likes',axis=alt.Axis(grid=False)),
    column=alt.Column('Meal Type',
                      header=alt.Header(
                          labelOrient='bottom',
                          title=None
                          )),
    color=alt.Color('Year',scale=alt.Scale(range=['lightgray','#8
).properties(
    width=50,
    height=300
).configure_view(
    strokeWidth=0
)

chart.save('multiple-column-chart.html')
```

Creating a multiple-series column chart in Altair

Use the `column` channel to add multiple series to your chart. By default, the column chart adds the column labels to the top of the chart. To move them to the bottom, set the `header` attribute in the `alt.Column()` channel.

Figure 6.17 shows the resulting chart.

Figure 6.17 A column chart with multiple series



Now that you have learned how to implement a column chart with multiple series, let’s move on to the following chart, the pyramid chart.

Pyramid Chart

A pyramid chart consists of two back-to-back bar charts, with the two sides representing contrasting categories, such as males and females. The horizontal axis represents quantities, and the vertical axis shows categories to compare, typically periods.

Consider the dataset shown in Table 6.7, describing the number of orders of Pizza and Spaghetti from January to December 2023.

Table 6.7 A sample dataset with the number of orders of Pizza and Spaghetti from January to December

Month	Pizza	Spaghetti

January	200	18
February	220	19
March	240	10
April	230	20
May	250	20
June	280	40
July	300	60
August	320	80
September	310	70
October	290	50
November	270	30
December	240	20

A pyramid chart is composed of three main elements: the left part, the middle part and the right part. The left and right parts are bar charts, with the left part flipped. The middle part, instead, contains the labels of both the bar charts. We will not write prompts for Copilot to generate code. Instead, we will use the suggestions it will propose. We will show directly the produced code, without providing the screenshots of the Copilot suggestions. We suggest you try writing the code to experiment with the Copilot power.

Listing 6.20 describes the code to build the left part of the pyramid chart, focussing on pizza.

Listing 6.20 How to create the left part of the pyramid chart

```
import pandas as pd
import altair as alt

# Load data data/orders.csv as a Pandas dataframe
df = pd.read_csv('data/orders.csv')
```

```
months = [
    "January", "February", "March", "April",
    "May", "June", "July", "August",
    "September", "October", "November", "December"
]

left_base = alt.Chart(df).encode(
    y=alt.Y('Month:N', axis=None, sort=months),
    x=alt.X('Pizza:Q', title='',sort=alt.SortOrder('descending'),
)

left =  left_base.mark_bar(
    size=20,
    color='#c01e95'
).properties(
    title='Pizza',
    width=300,
    height=300
)

left_text = left_base.encode(
    text=alt.Text('Pizza:N'),
).mark_text(
    color='#c01e95',
    baseline='middle',
    align='right',
    dx=-10,
)
```

Creating the left part of the pyramid chart

Build a base chart (`left_chart`) and then, use it to build the bar chart and the labels. Set the `sort` attribute in the `x` channel to `descending` to anchor bars to the right. Also, set the `align` attribute to `right` in the `mark_text()` method.

Now, let’s build the middle part of the pyramid chart. This part contains the labels. Listing 6.21 shows the code.

Listing 6.21 How to create the middle part of the pyramid chart

```
middle = alt.Chart(df
).encode(
    y=alt.Y('Month:N', axis=None, sort=months),
    text=alt.Text('Month:N'),
).mark_text(
    size=20,
).properties(
    width=100,
    height=300,
    title='Number of orders in 2023'
)
```

Creating the middle part of the pyramid chart in Altair

Use `mark_text()` to set the labels. Also, set the `y` and `text` channels.

Finally, let’s draw the right part of the pyramid chart. Use the same strategy as the left chart, without sorting the `x` channel. Also, combine all the charts to build the final chart. Listing 6.22 shows the code.

Listing 6.22 How to create the right part of the pyramid chart

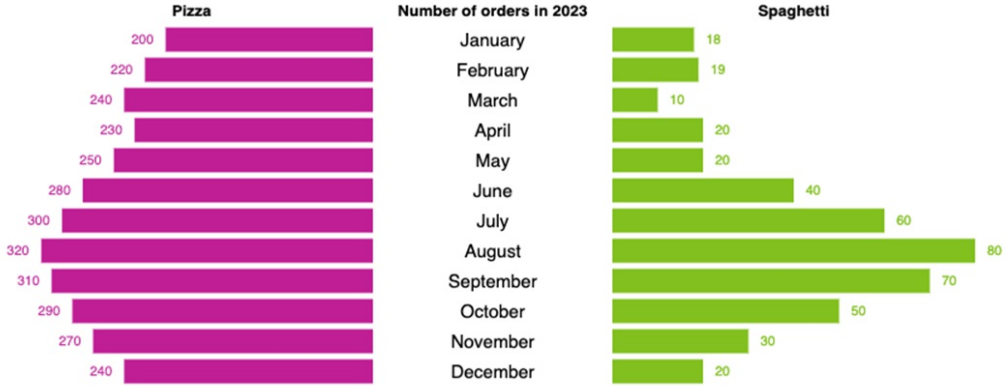
```
right_base =  alt.Chart(df
).encode(
    y=alt.Y('Month:N', axis=None,sort=months),
    x=alt.X('Spaghetti:Q', title='',axis=None),
)

right = right_base.mark_bar(
    size=20,
    color='#81c01e'
).properties(
    title='Spaghetti',
    width=300,
    height=300
)

right_text = right_base.encode(
    text=alt.Text('Spaghetti:Q')
).mark_text(
    baseline='middle',
    align='left',
    dx=10,
    color='#81c01e'
)

chart = left + left_text | middle | right + right_text

chart = chart.configure_view(
    strokeWidth=0
)
# save chart as 'pyramid-chart.html'
chart.save('pyramid-chart.html')
```

Creating the right part of the pyramid chart in Altair

Use the `+` operator to combine each single part and the `|` operator to combine the parts together.

Figure 6.18 shows the resulting chart.

Figure 6.18 A pyramid chart



Now that you have learned how to build a pyramid chart, let’s move on to the following chart, the stacked column chart.

Stacked Column Chart

Stacked column charts are similar to traditional column charts, but they show the contribution of each data series to the total value. Use stacked column charts to show how different factors contribute to a total over time or to compare the relative importance of different data series.

Let’s use Copilot to draw a stacked column chart. Surely you've noticed that as you write code, Copilot suggests more and more code that is close to your programming style. Therefore, we can try a new strategy of using Copilot, based on writing general instructions. This strategy assumes that we've already written a lot of code using Copilot. Surely if we had used generic instructions at the beginning of using Copilot, we would surely have gotten disappointing results. Listing 6.23 shows the general instructions for Copilot.

Listing 6.23 The generic instructions fo Copilot

```
# Consider the dataset in the file data/orders.csv.
# The dataset contains information about orders placed by custome
# Each row in the dataset represents the number of orders by mont
# The dataset contains the following columns:
# - `Month`: The month of the year
# - `Pizza`: The number of pizza orders
# - `Spaghetti`: The number of spaghetti orders
# Build a complete stacked column chart in Altair using the datas
```

Writing generic instructions to build a stacked column chart

First specify the dataset structure and, then, use Copilot to build a stacked column chart using the described dataset.

Listing 6.24 shows the code produced by Copilot. It is quite similar to the codes generated in the previous examples.

Listing 6.24 How to create a stacked column chart

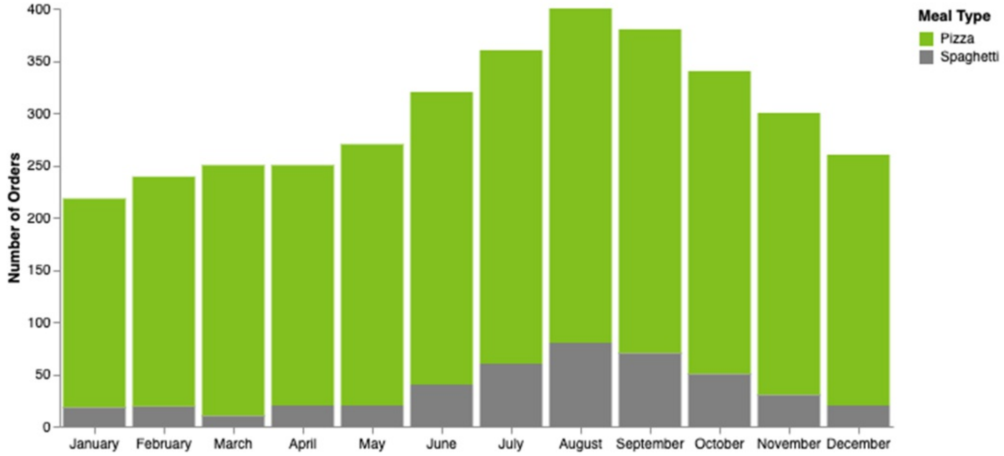
```
import pandas as pd
import altair as alt

# Load data data/orders.csv as a Pandas dataframe
df = pd.read_csv('data/orders.csv')

df = df.melt(id_vars=['Month'],var_name='Meal Type',value_name='N

# Build a list of months #A
months = ['January','February','March','April','May','June','July

chart = alt.Chart(df).mark_bar(
).encode(
    x=alt.X('Month',
            axis=alt.Axis(title=None,
                          labelAngle=0,
            ),
            sort=months
    ),
    y=alt.Y('Number of Orders'),
    color=alt.Color('Meal Type',scale=alt.Scale(range=['#81c01e',
).properties(
    width=600,
    height=300
).configure_view(
    strokeWidth=0
).configure_axis(
    grid=False
)
chart.save('stacked-column-chart.html')
```

Creating a stacked column chart in Altair

Before drawing the chart, transform the DataFrame using the `melt()` method.

Figure 6.19 shows the resulting chart.

Figure 6.19 A stacked column chart

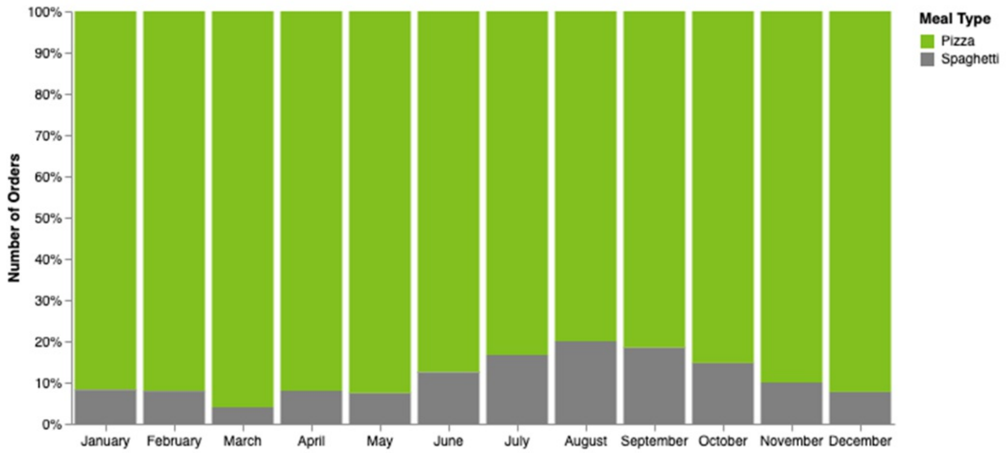


Now that you have learned how to build a stacked column chart in Altair and Copilot, let’s see how to transform a stacked column chart into a 100% stacked column chart.

100% Stacked Column Chart

A 100% stacked column chart is a stacked column chart with each column stacked to a height of 100% to show the proportional composition of each category. It is used to compare the contribution of different categories within each column across multiple data series. To transform a stacked column chart into a 100% stacked column chart, set `normalize=True` in the `y` channel of Listing 6.4: `y=alt.Y('Number of Orders',stack='normalize')`. Figure 6.20 shows the resulting chart.

Figure 6.20 A 100% stacked column chart



Now that we have built a 100% stacked column chart, let’s move to the last chart of the bar chart family, histograms.

Histograms

A histogram represents the distribution of numerical data, with continuous data divided into intervals called bins and the height of each bar representing the frequency of data points falling within that bin. Use histograms to visualize the spread of a dataset and identify patterns or outliers.

Consider the sample dataset described in Table 6.8, showing the average rating for different products. Rating ranges from 1 to 10. The table shows the values only for some products. You can read the full dataset in the GitHub repository of the book, under 06/bar-charts/data/product-ratings.csv

Table 6.8 A sample dataset with average rating from some products

ProductID	Rating
Product_1	4.8
Product_2	5.7
Product_3	5.3
Product_4	4.8
Product_5	5.9
Product_6	4.0
Product_7	4.7
Product_8	5.9
Product_9	4.0

Let’s use Copilot to draw a histogram. The idea is to specify generic prompts to build the chart framework, and then, refine the chart manually. Listing 6.25 shows the used prompts.

Listing 6.25 The generic prompts to build a histogram

```
# Import the required libraries
# Load the 'data/product-ratings.csv' into a dataframe
# Create a histogram of the Rating column using Altair
# Save the chart as a HTML file
```

Defining the basic prompts to build a histogram

Use these prompts to speed up the chart framework creation.

As an alternative, you could have specified very detailed prompts, as we did for the other charts. Here we want to illustrate the different Copilot usage strategies you can use to generate prompts for Copilot. In our case, the resulting code is not totally correct. We need to improve it, to produce the code shown in Listing 6.26.

Listing 6.26 The code to build a histogram

```
chart = alt.Chart(df).mark_bar(
    color='#81c01e'
).encode(
    x=alt.X('Rating:Q',
            bin=alt.Bin(maxbins=10, extent=[1, 10]), #A
            title='Rating',
            axis=alt.Axis(
                format='d',
            )
    ),
    y=alt.Y('count()', title='Number of Products')
)
```

Building a histogram

Use the `bin` attribute to specify the number of bins (`maxbins`) and their extent.

For comparison, we can add a one-dimensional kernel density estimation over our data to the chart, by using the `transform_density()` method provided by Altair, as shown in Listing 6.27.

Listing 6.27 One-dimensional kernel density estimation

```
line = alt.Chart(df).transform_density(
    'Rating',
    as_=['rating', 'density'],
).mark_line(
    color='red',
).encode(
    x='rating:Q',
    y=alt.Y('density:Q', axis=None)
)

# combine the bar chart and the density estimator
chart = chart + line

chart = chart.resolve_scale(y='independent'
).configure_view(
    stroke=None
).configure_axis(
    grid=False
)
```

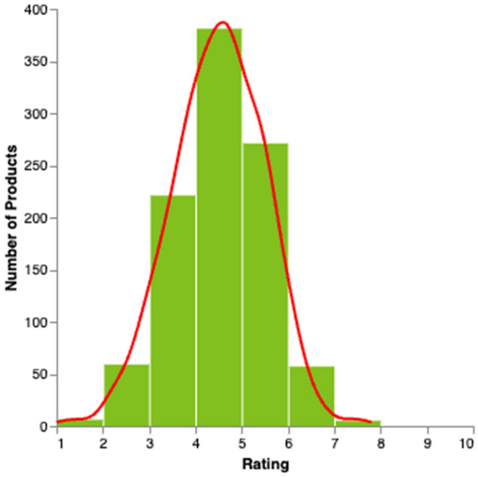
```
chart.save('histogram.html')
```

How to build a one-dimensional kernel density estimation

Use the `transform_density()` method to create a kernel density estimation. This method takes the `column` of the DataFrame to use for calculation and the names of the generated columns (`as_` attribute). Also, combine the produced line with the previous chart.

Figure 6.21 shows the resulting chart.

Figure 6.21 A histogram



We have completed the bar charts family. Let’s move on to analyze the line charts family.

6.3.3 Line Charts Family

Line charts family include all the charts that depict data using continuous lines to show the relationship or trends between different data points over a given time period. Some commonly used line chart types include basic line charts, area charts, slope charts, and Dumbbell charts. Let’s start analyzing the line chart.

Line Chart

Line charts, also known as line plots or line graphs, visualize data over time. They connect data points by straight lines, which makes it easy to see how the data has changed over time. Line charts are particularly useful for showing trends and patterns in data and comparing multiple data series.

Consider again the dataset in Table 6.7, showing the number of orders for Pizza and Spaghetti from January to December. In her book Data Storytelling with Data (Knaflic 2015), Cole Nussbaumer Knaflic proposes to replace the legend from a line chart with labels near each line. To follow Knaflic’s suggestion, we can build a line chart in Altair in three steps. First, we build a base chart, with the basic structure. It includes the general encodings and properties (Listing 6.28).

Listing 6.28 How to build the base chart of a line chart

```
base = alt.Chart(df).encode(
    x=alt.X('Month',
            axis=alt.Axis(title=None,
                            labelAngle=0,
            ),
            sort=months
    ),
    y=alt.Y('Number of Orders'),
    color=alt.Color('Meal Type',scale=alt.Scale(range=['#81c01e',
).properties(
    width=600,
    height=300
)
```

Drawing the base chart of a line chart

To build a base chart don’t specify the mark.

Next, we draw the line chart by applying the `mark_line()` method to the base chart. Finally, we draw the labels by applying the `mark_text()` method to the base chart (Listing 6.29).

Listing 6.29 How to build the line chart and the labels

```
chart = base.mark_line()
```

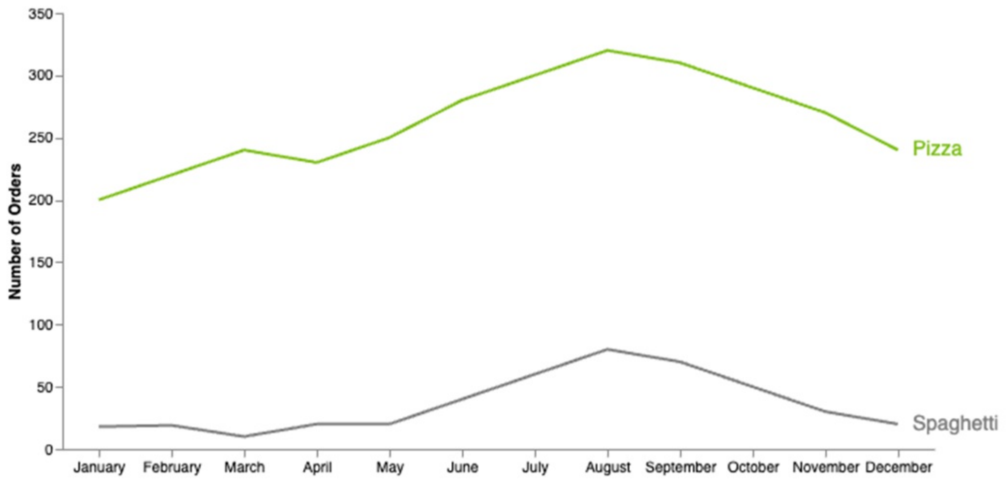
```
text = base.mark_text(
    fontSize=14,
    baseline='middle',
    align='left',
    dx=10
).encode(
    text=alt.Text('Meal Type:N'),
).transform_filter(
    alt.datum['Month'] == 'December'
)

# Combine the line chart and text mark into a single chart
chart = chart + text

chart = chart.configure_view(
    strokeWidth=0
).configure_axis(
    grid=False
)
chart.save('line-chart.html')
```

Creating a line chart in Altair

Once defined the base chart, build the line chart by only specifying the `mark_line()` method. Use the `transform_filter()` method to select only some data. In our case, select only the last values (month equal to December) of the dataset, to position the labels at the end of the lines.

Figure 6.22 shows the resulting chart.

Figure 6.22 A line chart



Now that you have learned how to draw a line chart, let’s move on to the following chart, the area chart.

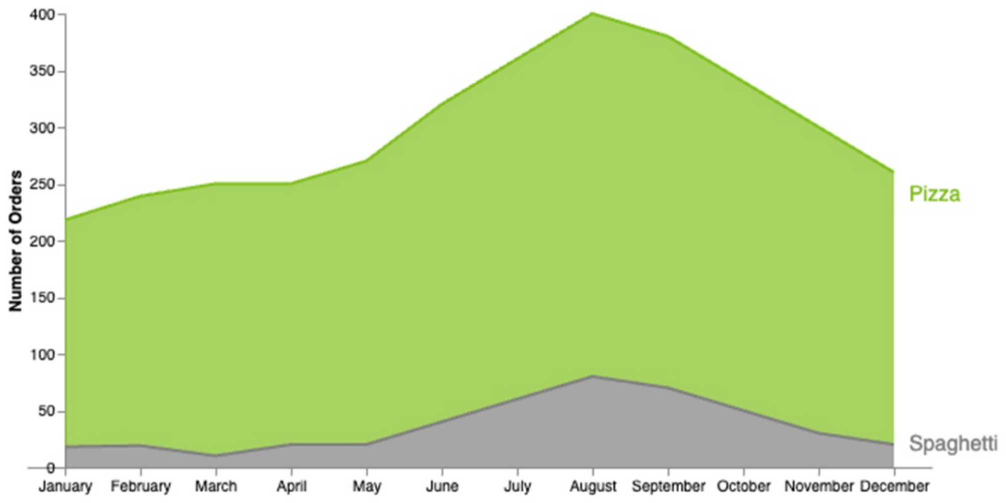
Area Chart

An area chart shows the cumulative trend over time of a variable. It is similar to a line chart, but the area between the x-axis and the line is filled with a color or pattern to distinguish it from a line chart. You can draw an area chart, by modifying the code in Listing 6.29 as follows:

- Change `base.mark_line()` to `base.mark_area(line=True)`
- Translate the y axis to align to the x axis by using the following code: `y=alt.Y('Number of Orders',axis=alt.Axis(offset=-25))`.

Figure 6.23 shows the resulting chart.

Figure 6.23 An area chart

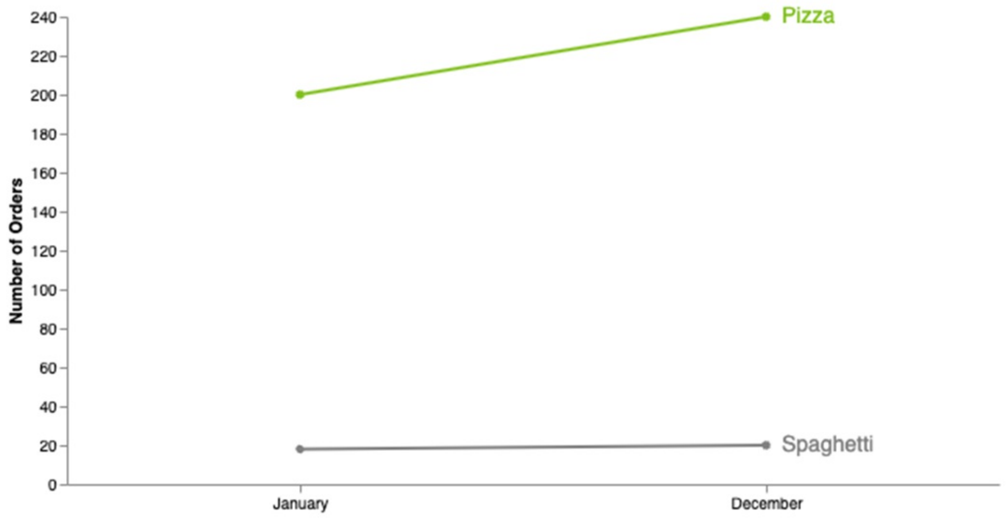


After seeing the area chart, let’s investigate the slope chart.

Slope Chart

Slope charts consist of two sets of data points connected by a line, with the x-axis representing the different categories or periods and the y-axis representing the values of the data. A slope chart is a line chart with just two points. To build a slope chart, add the `transform_filter()` method to Listing 6.28 to select only the first and the last value of your series. Figure 6.24 shows an example of a slope chart.

Figure 6.24 An slope chart



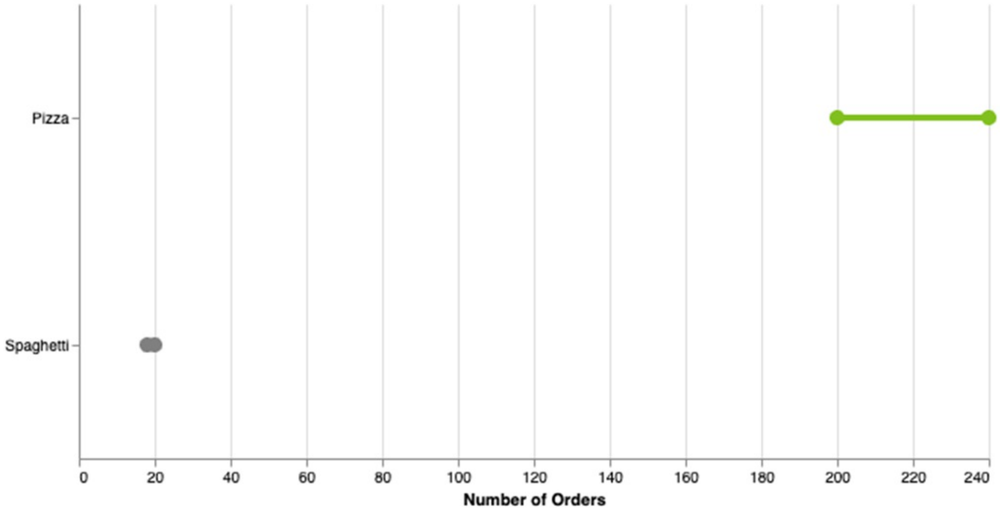
The next chart is the Dumbbell chart. Let's move on to analyze it.

Dumbbell Chart

Dumbbell charts or floating bar charts consist of two data points connected by a line, one at the beginning and one at the end. Dumbbell charts are similar to slope charts, but usually, we use Dumbbell charts to compare the difference between two data points. In contrast, we use slope charts to compare changes in data over time or between different groups.

Figure 6.25 shows an example of a Dumbbell chart. You can find the associated codein the GitHub repository of the book.

Figure 6.25 An area chart



We have completed the review of line charts. For some of them, we used Copilot, for others we didn't directly specify prompts. However, in all the cases, Copilot assisted us while writing the code, by suggesting new code while writing it.

Did you know

that line charts were first used by William Playfair, a Scottish engineer, in the late 18th century? In addition to inventing the line chart, Playfair also introduced other popular data visualizations such as bar charts and pie charts.

His innovative use of graphical representations revolutionized the way we present data today, making him a pioneer in the field of data visualization

Now that you have learned how to build the different types of line charts, let’s move on to the next family, the geographical map family.

6.3.4 Geographical Map Family

A geographical map shows the location of different data points on the earth's surface, such as countries, continents, oceans, and rivers. Also, a geographical map shows the distribution of resources and data on a given area of interest.

The most popular geographical maps include:

- **Choropleth maps** use color to show differences in values between different areas (Figure 6.26)
- **Dot density maps** use dots to show the concentration of a certain value within an area (Figure 6.27)
- **Proportional symbol maps** use symbols that are sized according to the value they represent (Figure 6.28).

You can find the code associated with each type of geographical map in the GitHub repository of the book.

Figure 6.26 A choropleth map

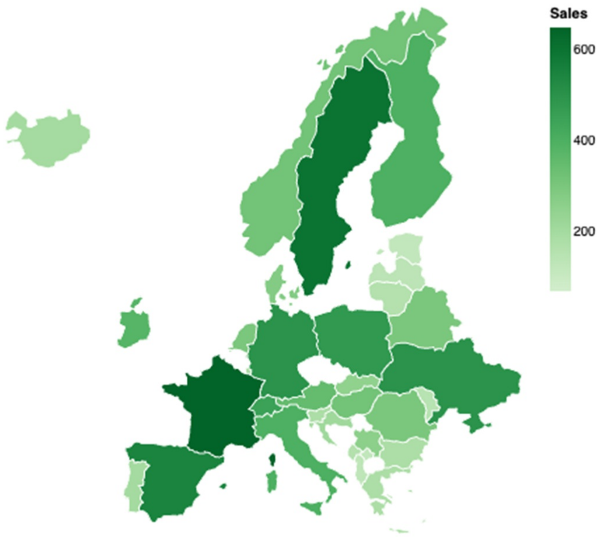


Figure 6.27 A dot density map

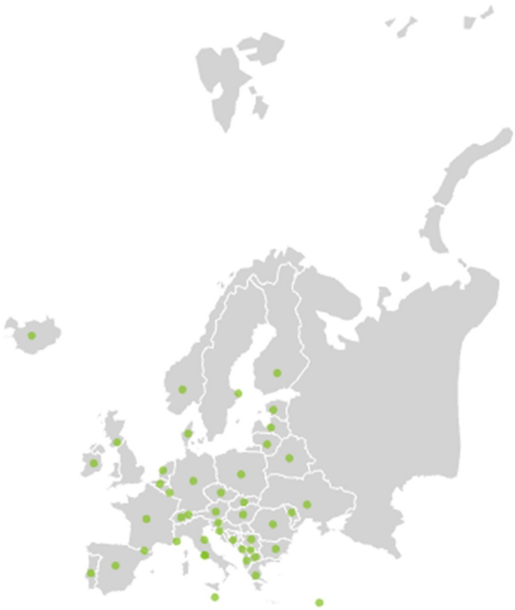
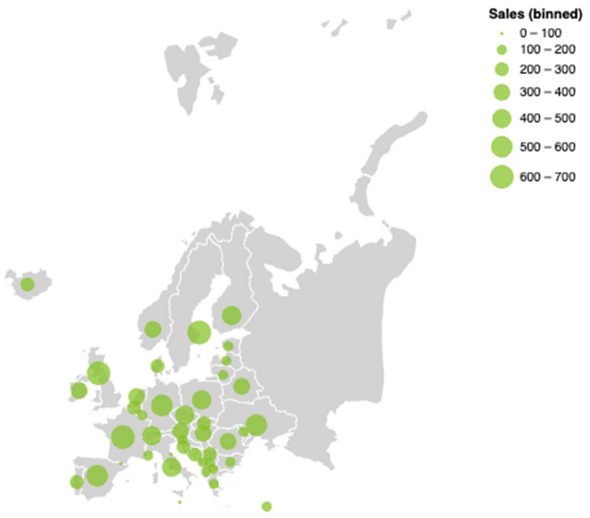


Figure 6.28 A proportional symbol map



When you build a geographical map, you must set the projection to use. One of the most popular projection maps is the Mercator projection map. Although this map is very popular, it does not represent the world correctly. For example, Greenland appears to be nearly the same size as Africa, when in reality Africa is about 14 times larger! This distortion occurs because it is difficult to accurately represent the curved surface of the Earth on a flat map.

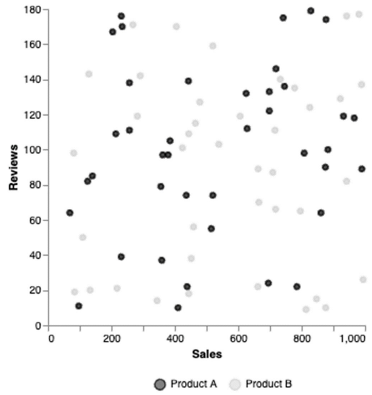
Now that you have learned some of the most popular geographical maps, let’s move on to the dot charts family.

6.3.5 Dot Charts Family

Dot charts family include charts that represent data points with dots along the x and y scales to show the relationship between two variables. Some of the most popular dot chart families include scatter plots and bubble charts. Use the `mark_point()` mark to draw a dot chart.

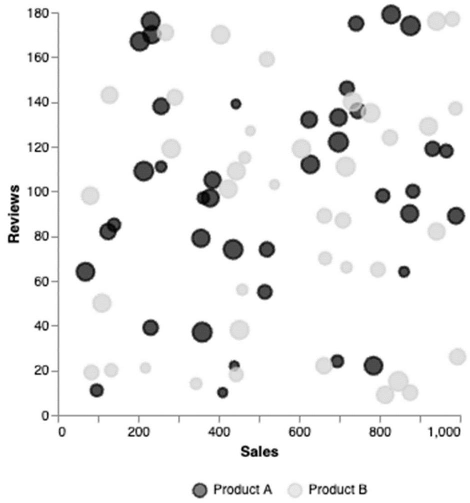
Scatter plots visualize the relationship between two numerical variables. Use them to identify patterns and trends in data and make predictions based on those trends. Figure 6.29 shows an example of a scatter plot. You can find the associated code in the GitHub repository of the book.

Figure 6.29 A scatter plot



Bubble charts are scatter plots that use the size of the data points to encode an additional variable. In Altair, you can use the size channel to set the bubble size: `size=alt.Size('Ranking:Q',scale=alt.Scale(range=[1,200]),legend=None)`. Figure 6.30 shows an example of a bubble chart.

Figure 6.30 A bubble chart



Now that you have learned the basic charts families, let’s move on to implement a practical case study.

6.4 Case Study: Salmon Aquaculture

Imagine that you want to study the problem of safety in the salmon aquaculture in the U.S.. You have the aquaculture dataset, provided by the US Department of Agriculture, Economic Research Service under the CC-0 license. This dataset describes for each fish category the exported KG and the dollars earned over the time, as shown in Table 6.9.

Table 6.9 A snapshot of the aquaculture dataset

SOURCE_ID	HS_CODE	COMMODITY_DESC	GEOGRAPHY_COD
63	302110000	TROUT (SALMO TRUTTA, S. CLARKI ETC) FRESH, CHILLED	1
63	1604112000	SALMON, WHOLE/PIECES, IN OIL IN AIRTIGHT CONTAINER	2740

The dataset contains the following columns:

- SOURCE_ID: The source of the data
- HS_CODE: The Harmonized System code for the commodity
- COMMODITY_DESC: The description of the commodity
- GEOGRAPHY_CODE: The code for the country or region of origin or destination
- GEOGRAPHY_DESC: The name of the country or region of origin or destination
- ATTRIBUTE_DESC: The type of data, either US Export, QTY or US Export VLU
- UNIT_DESC: The unit of measurement, either KG, or U.S. $
- YEAR_ID: The year of the data
- TIMEPERIOD_ID: The month of the data
- AMOUNT: The value of the data

The objective of your study is to establish if the safety measures adopted so far in salmon aquaculture are sufficient or should be improved. For simplicity, you base your study on this dataset. However, in a real situation, you should also consider other aspects, such as the costs to maintain these measures and other similar analyses.

You decide to plot the salmon aquaculture sales trendline versus the other types of aquaculture. You can find the code associated with this case study in the GitHub repository of the book, under CaseStudies/aquaculture/. Ask Copilot to generate the chart for you. Use the list of instructions described in Listing 6.30 to build the base chart.

Listing 6.30 The instructions for Copilot

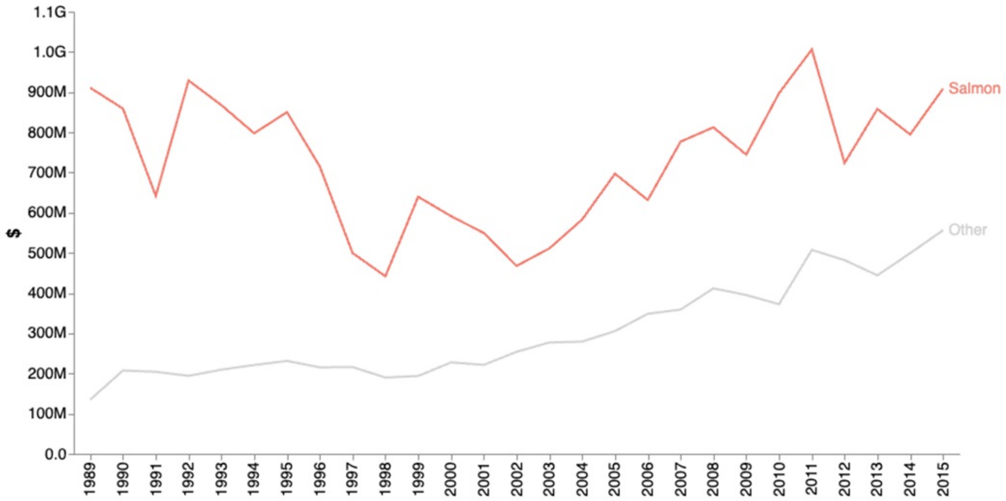
```
# import required libraries
# load the dataset '../source/Aquaculture_Exports.csv' as a panda
# apply the following filters to the dataframe:
#   - select only the rows where the 'GEOGRAPHY_DESC' column is '
#   - select only the rows where the 'UNIT_DESC' column is 'U.S.$
# add a new column to the dataframe called 'DATE' which is a date
#   - the year is the 'YEAR_ID' column
#   - the month is the 'TIMEPERIOD_ID' column
#   - the day is 1
# plot the dateframe using altair as follows:
#   - the x axis is the 'DATE' column
#   - the y axis is the 'AMOUNT' column
#   - the color is the 'COMMODITY_DESC' column
# save the plot as 'chart.html'
```

How to instruct Copilot

After importing the required libraries, ask Copilot to import the dataset and apply some filters to select only the world exportations, the exportations in terms of dollars. Next calculate the date from the YEAR_ID and the TIMEPEROID_ID column. Finally, plot the trendline in Altair.

The previous instructions help you to build the basic chart. Work on it to produce the chart shown in Figure 6.31.

Figure 6.31 The aquaculture case study



You can find the resulting code in the GitHub repository of the book, under CaseStudies/aquaculture/from-data-to-information/chart.py. As an insight, you discover that since 1998 there has been an increase in sales, following a period of decrease in sales from 1992 to 1998. Moved by curiosity on the decreasing period, you do some search and discover that the decreasing period was partially due to some health problems in the salmon aquaculture.

At the end of this first step, you have turned data into information. In the next chart, we will continue implementing this case study, by turning the extracted information into knowledge.

This chapter has described how to turn data into information. First, we have described some techniques to extract insights. Next, we have seen how to select the characters of a story. Finally, we have learned how to use Copilot to implement some of the most popular charts in Altair. You can find other charts in the GitHub repository of the book, under section 06/other-charts.

In the next chapter, you will learn how to turn information into knowledge.

6.5 Summary

- Turning data into information means extracting meaning from data, an insight that is significant for you.
- Some of the techniques to extract insights include connections, coincidences, curiosity, and contradictions.

- Connections involve searching for details in your data and connecting them.
- Coincidences involve searching for repeated or related events that apparently are not related each other
- Curiosity is searching for strange events, such as missing values and anomalies
- Contradiction is searching for events that apparently conflict with each other.
- Every story should have at least three characters: the hero, the guide, and the adversary.
- The right chart to tell your story depends on the information to convey.
- There are different families of charts, such as bar charts, line charts, geographical charts, cooking charts.

6.6 References

- Barr Moses, Lior Gavish, Molly Vorwerck *Data Quality Fundamentals: A Practitioner's Guide to Building Trustworthy Data Pipelines* O’Reilly Media, 2022
- Andrea De Mauro, Francesco Marzoni, Andrew J. Walter Data Analytics Made Easy, 2021, Packt Ltd.
- Thomas A Dorfer. *The Case Against the Pie Chart* https://towardsdatascience.com/the-case-against-the-pie-chart-43f4c3fccc6 (Last Access 2023/07/23)
- Artur Guja, Marlena Siwiak, and Marian Siwiak *Generative AI for Data Analytics* Manning, 2024
- Gary Klein *Seeing What Others Don't: The Remarkable Ways We Gain Insights* Nicholas Brealey Publishing, 2017
- Donald Miller *Building a StoryBrand: Clarify Your Message So Customers Will Listen* Thomas Nelson, 2017
- Cole Nussbaumer Knaflic *Storytelling With Data: A Data Visualization Guide for Business Professionals* Wiley, 2015

[1] https://extremepresentation.typepad.com/files/chart-chooser-2020.pdf

[2] https://www.linkedin.com/posts/brentdykes_datavisualization-datastorytelling-storytellingwithdata-activity-7041489036980494336-ieui

7 From Information to Knowledge: Building textual context

This chapter covers

- Introducing context
- Calibrating the story to the audience
- Using ChatGPT for commentaries and annotations
- Using Large Language Models for textual context
- Case study: from Information to Knowledge (part 1)

In this and the next chapter, we will review the basic concepts behind context in a data story and how to adapt it based on the audience. In this chapter we will focus on textual context and in the next one on images. We will introduce Large Language Models (LLMs) and use ChatGPT as an example of LLM implementation for data storytelling.. Finally, we will describe a practical example

7.1 Introducing Context

When I was a child, I often heard my parents discussing some topic and did not understand anything. Their words rang in my ears as meaningless until, eager to understand what they were talking about, I entered the conversation and asked for explanations. Then my father or mother, very patiently, explained to me what they were talking about, adapting their adult reasoning to my child's mind so that I, too, could understand. Years later, I was in the same situation now that I'm a mother. My children often ask me to explain more complex speech *in words they can understand*. And the satisfaction is enormous when I see their faces clear up and understand what I’m saying.

The examples described show us the need to adapt the words we use according to the audience we are facing. If we ignore who will read our story, we risk talking in a way that may make perfect sense to ourselves, but our audience will be excluded from the message we want to communicate.

In the previous chapter, we looked at how to extract and represent an insight through a chart. The next step is to enrich the chart with context (text and images), making reading easier for the reference audience. Context refers to the surrounding elements making the audience understand the displayed information, such as texts, images, and symbols. Data context should prepare the scene of your data story and raise interest in your audience. In this chapter, we'll primarily be dealing with textual context, while in the next chapter we'll look more at visual context.

Context depends on the type of audience you are faced with. For example, if you are talking with an adult about how much you paid for a product, you don't need to explain how money works. Instead, if you are talking to your kids about the same topic, you probably need to explain the denominations of the different banknotes and how the monetary system works.

You can use Generative AI tools, such as ChatGPT for text and DALL-E for images to ease context-building. You have already learned the basic techniques for building context using Generative AI tools. This chapter will focus on more advanced techniques to write an impactful context tailored to your audience.

We will consider the following types of context:

- *Commentary* - the text that precedes your insight. It includes the background that helps the audience to set the scene and understand the insight. In the example of the product cost explained to your kids, the commentary includes banknotes denominations, and how the monetary system works.
- *Annotation* - a short text that explains a detail of your chart, for example, an anomalous point or a trendline. Consider adding annotations only if they are necessary. Don't overload your chart with unnecessary annotations.
- *Image* - a picture enforcing the commentary or the annotation. In the example of the product cost, you could add banknote images to help your kids understand the different denominations.
- *Symbols* - arrows, circles, lines, and so on, combined with annotations.

They help the audience focus on particular points of your chart.

In the remainder of this chapter, we will use ChatGPT for commentaries and annotations. In the next chapter, we will focus on DALL-E for images and symbols. In addition, we will introduce Large Language Models and how to use them for commentaries and annotations. But, before this, let’s describe how to calibrate the story to your audience.

7.2 Calibrating the Story to The Audience

A few years ago, I was invited to give a seminar to master's students. The seminar topic concerned the implementation of web applications for the construction of data journalism projects. Unfortunately, I found myself faced with a somewhat embarrassing situation. My seminar topic was very technical, even commenting on some pieces of code. As I began to speak, I realized that the audience couldn't follow me because they didn't have the technical skills required to listen to me. My presentation was technically correct, but having spoken too technically to a non-technical audience, the result of my talk was that the audience learned very little. From the experience gained from that episode, I learned to find out about the audience I will have to deal with before communicating any message.

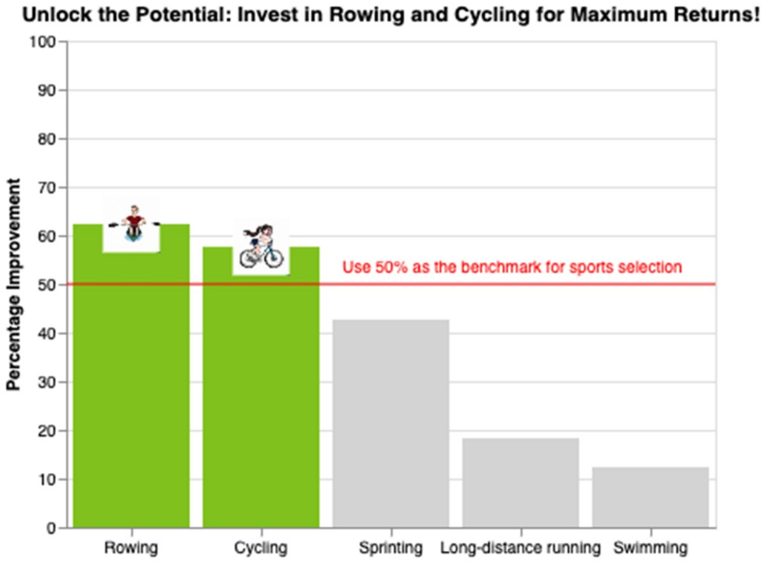
The audience is the person or the group of persons reading your data story. Understanding the target audience is crucial to building data stories that convey information effectively. In the previous chapter, we saw that you can use multiple types of charts to convey information (Table 6.4). Once you've chosen the set of charts that answer your question, you can refine your choice, tailoring the chart to your audience.

In Chapter 4, you have learned that there are different types of audiences. For simplicity, in this chapter, we group them into three common types of audiences:

- General Public
- Executives
- Professionals

Let’s investigate each type of audience separately. To explain how you can calibrate the chart to the target audience, we will use the case study described in Chapter 4: *Unlock the potential: invest in Rowing and Cycling for maximum return*. For convenience, Figure 7.1 shows the complete data story we implemented, moving from data to wisdom.

Figure 7.1 The use case described in Chapter 4

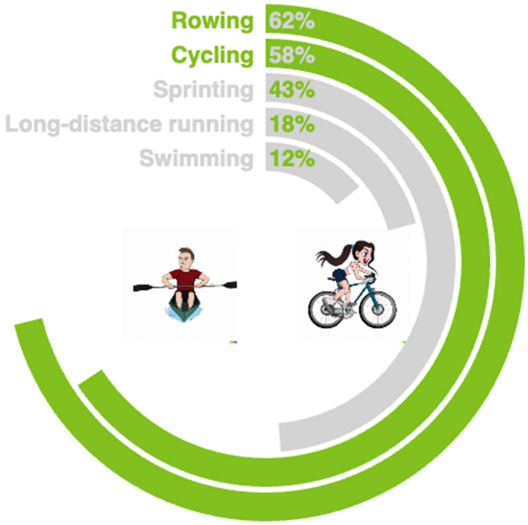


The objective of this case study was to understand which disciplines are worth continuing to train in to achieve good results in the upcoming competitions.

7.2.1 General Public

This audience includes individuals from various backgrounds and levels of knowledge. They may have little to no previous knowledge of your topic. When crafting data stories for the general public, use precise language, avoid overwhelming them with too much information, and focus on presenting the most relevant insights visually and engagingly. The general public could find the chart shown in Figure 7.1 complex, with an unnecessary baseline. As an alternative to the chart of Figure 7.1, you could draw the chart shown in Figure 7.2, which the audience could find more appealing.

Figure 7.2 The use case adapted to the general public



This chart is called a multi-layer donut chart. We could have placed the images close to the relevant bars, but in this specific case there wasn't enough space, so we placed them in the center of the chart. In other scenarios, you might consider placing images next to the bars. You can find the complete code to generate this chart in the GitHub repository of the book, under 07/general-public.

7.2.2 Executives

Executives are typically high-level decision-makers in organizations who rely on data-driven insights to make essential business choices. They often have limited time and need concise and actionable information. When creating data stories for executives, it is essential to present key findings, trends, and recommendations upfront.

Use visualizations highlighting the most critical data points and providing a straightforward narrative linking the data to strategic goals. It can also be helpful to provide additional context or industry benchmarks to support your analysis. The chart shown in Figure 7.1 could be great for executives because it does not contain many details and describes why we chose some sports, thanks to the baseline of 50%.

7.2.3 Professionals

This audience consists of individuals with a specific domain expertise or professional background. They have a deeper understanding of data and require more analytical information. When creating data stories for professionals, explain the data analysis's methodology, assumptions, and limitations. Consider including additional supporting data and references, allowing professionals to explore the data further.

As an alternative to the chart of Figure 7.1, you could draw the chart shown in Figure 7.3, which the audience could understand easily. The figure shows only the chart, without any annotation or context. You can find the complete code to generate this chart in the GitHub repository of the book, under 07/professionals.

Figure 7.3 The use case adapted to professionals

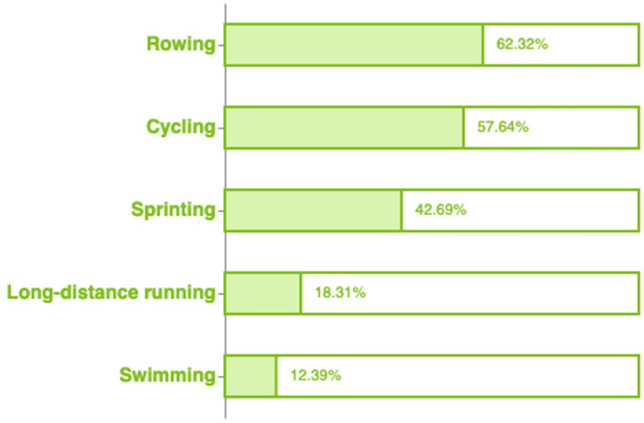


Table 7.1 summarizes what to represent in a chart based on the audience type.

Table 7.1 What to represent in a chart based on the audience type

Audience Type	Requirements	What to represent
General Public	Understand data	An appealing overview of insights
Executives	High-level overview of data trends to aid strategic decision-making	Highlight critical metrics and trends influencing business outcomes

Professionals	Detailed insights to understand the phenomenon behind data	Add numbers, statistics, and useful information to understand insights deeply

Now that you have learned how to adapt your chart based on the audience type, let's move on to the next step, using ChatGPT for commentaries and annotations.

7.3 Using ChatGPT for Commentaries and Annotations

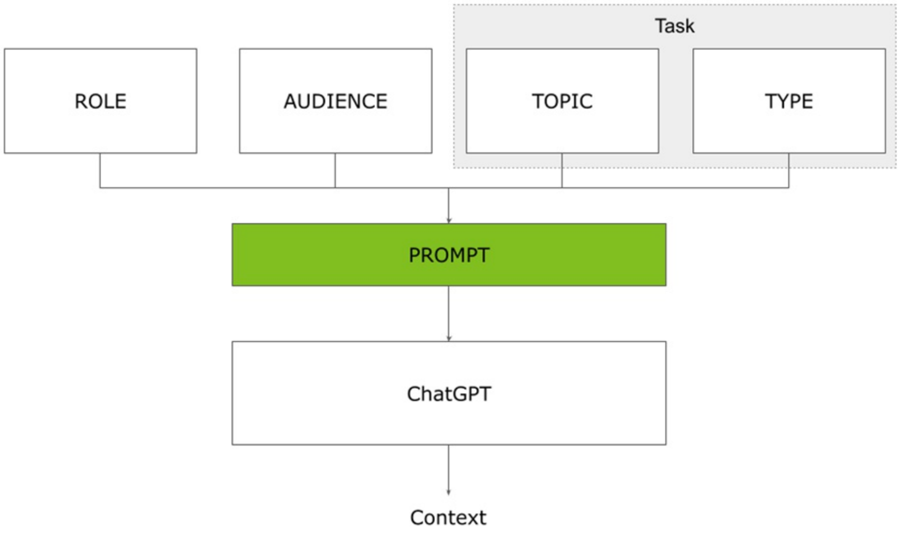
In his novella Metamorphosis, Franz Kafka tells the story of Gregor Samsa, a traveling salesman who wakes up one morning to be transformed into a giant insect. Encased in this insect's guise, Samsa cannot interact with his family or communicate his thoughts. Gregor's family struggles to accept his transformation, so their relationship with Gregor deteriorates, and Gregor becomes increasingly isolated. The novella unearths the fundamental isolation that emerges when one's inner world remains inaccessible to others. The data analysts could find themselves in a situation quite similar to that experienced by Gregor Samsa in Kafka's novella when they have to add text to a data visualization chart. The data analyst, by nature, is a technician and could encounter some difficulties in writing engaging text.

ChatGPT can assist you in adding textual context to your data visualization chart. You have already learned that a prompt's basic structure for ChatGPT comprises three main elements: role, audience, and task.

For example, you can write: *Act as an entertainer* (role), *writing for decision-makers* (audience). *Write 5 titles about <topic>* (task). The topic could be whatever you want. The main problem is **structuring the topic** so that ChatGPT **produces the correct context**.

To also include the topic in the ChatGPT prompt, we will generate context following the schema described in Figure 7.4.

Figure 7.4 The schema used to generate context



In a prompt, we specify the following four main elements:

- *Role* - the role you want ChatGPT to take. You have already learned many role types in Chapter 4, including entertainer, educator, informer, inspirer, invitor to action, relationship builder
- *Audience* - the audience of your chart. There are different types of audiences, such as the general public, executives, and professionals
- *Topic* - the subject of your chart
- *Type* - the text type to generate, including annotations and commentaries.

The process of generating context is iterative, in the sense that you can generate the context multiple times if you are not satisfied with the produced result. For example, you can adjust one or more elements to make ChatGPT converge on the desired output.

In the remainder of this section, we will focus on how to write the topic and type elements of the schema while keeping the role and the audience simple. However, you can adapt the strategies described for the topic and the audience to the other elements.

As an example of how to build the context, we will focus on the case study described in Chapter 4, *Unlock the potential: invest in Rowing and Cycling for maximum return* and shown in Figure 7.1. The following text summarizes the scenario for convenience:

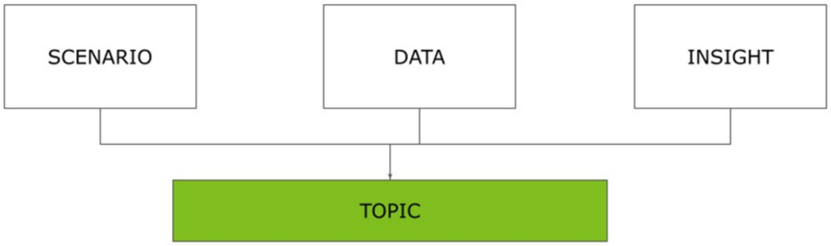
> *Imagine you work in a sports company. You are training a team of young athletes in various disciplines. For each discipline, you have noted the world record and recorded the best time achieved by your team for comparison. Unfortunately, your company has limited investment funds available. Your boss asks you to understand which disciplines are worth training in, hoping to achieve good results in the upcoming competitions.*

7.3.1 Describing the Topic

Describing the topic means writing for ChatGPT simple words that precisely depict what you have discovered and shown in your chart. The more precise you are, the better the produced output.

To describe the topic, focus on three aspects: scenario, data, and insight, as shown in Figure 7.5.

Figure 7.5 The elements used to describe the topic



Let’s go through each of those three aspects in a bit more detail, next.

Scenario

Describe an overview of your scenario, including the analysis's background and objective. For the scenario in Figure 7.1, we could write the following text to prompt ChatGPT:

> *We are training a team of young athletes in various disciplines. For each discipline, we have calculated the percentage improvement of each discipline compared to the world record in that discipline. The objective is to search for the best two disciplines to fund.*

Data

Describe an overview of your data. This may include explaining the type of data, its source, and any manipulations you performed.

Describe the data using your own words. Provide a more personalized description of the data. By manually describing the data, you can highlight important patterns, trends, or correlations that may not be apparent through automated methods alone. Additionally, through manual descriptions, you can incorporate domain expertise observations.

For the scenario in Figure 7.1, write the following prompt text:

> *There are five sports disciplines: Rowing (percentage improvement = 62.32%), Cycling (57.64%), Sprinting (42.69%), Long-distance running (18.31%), and Swimming (12.38%).*

Now that you have learned how to describe the data, let’s move on to the last step, describing insights.

Insights

Describe the central insight you have derived from the analysis, such as key patterns, trends, correlations, or relationships you have discovered. For the scenario in Figure 7.2, include the following text:

> *Rowing and Cycling percentages are more significant than the baseline of 50%.*

Before illustrating how we can build the audience description, let’s test the prompt built so far on ChatGPT.

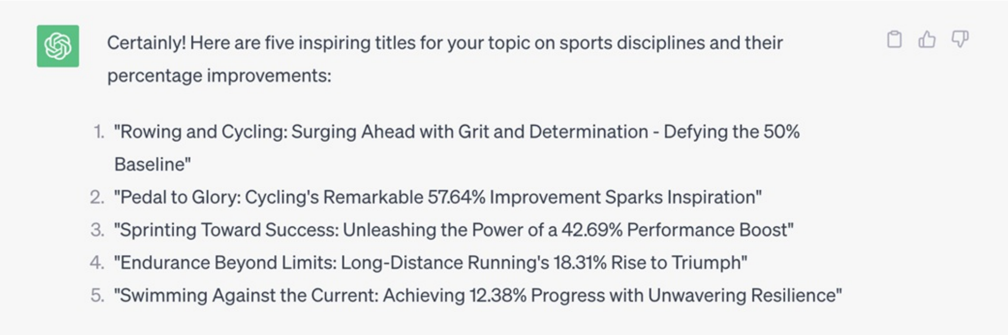
Test

We write the following prompt:

> *Act as an inspirer. Write **5 titles** for the following topic. There are five sports disciplines: Rowing (percentage improvement = 6.2.32%), Cycling (57.64%), Sprinting (42.69%), Long-distance running (18.31%), and Swimming (12.38%). Rowing and Cycling percentages are more significant than the baseline of 50%.*

Figure 7.6 shows a possible output produced by ChatGPT.

Figure 7.6 The five titles generated by ChatGPT



Certainly! Here are five inspiring titles for your topic on sports disciplines and their percentage improvements:

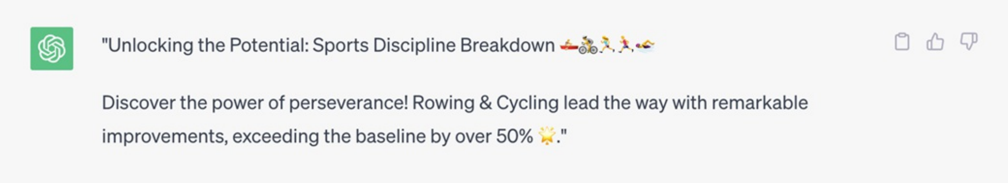
1. "Rowing and Cycling: Surging Ahead with Grit and Determination - Defying the 50% Baseline"
2. "Pedal to Glory: Cycling's Remarkable 57.64% Improvement Sparks Inspiration"
3. "Sprinting Toward Success: Unleashing the Power of a 42.69% Performance Boost"
4. "Endurance Beyond Limits: Long-Distance Running's 18.31% Rise to Triumph"
5. "Swimming Against the Current: Achieving 12.38% Progress with Unwavering Resilience"

As an alternative, you can ask ChatGPT to produce the context as follows:

> *Act as an inspirer. Write **the context of a chart using 30 words** for the following topic. There are five sports disciplines: Rowing (percentage improvement = 6.2.32%), Cycling (57.64%), Sprinting (42.69%), Long-distance running (18.31%), and Swimming (12.38%). Rowing and Cycling percentages are greater than the baseline of 50%.*

Figure 7.7 shows a possible output produced by ChatGPT.

Figure 7.7 The context generated by ChatGPT



"Unlocking the Potential: Sports Discipline Breakdown 🚣🚴🏃‍♀️🏃🏊

Discover the power of perseverance! Rowing & Cycling lead the way with remarkable improvements, exceeding the baseline by over 50% 🌟."

Now that you have learned how to describe the topic, try to generate the context for the case studies described in Chapter 1, the pets scenario. For example, you can act as an informer. For convenience, we summarize the scenario:

> *The organizers of an event dedicated to pets are collecting the type of pets that will participate. For each pet category, the organizers advertise the event on specific websites dedicated to that category. The organizers ask you to build a quick report about the current situation.*

For more details, please refer to Chapter 1, *Introducing Data Storytelling*. You can find a prompt with the generated context at this link. For further practice, write the topic for the other scenarios described in the previous chapters.

Now that you have learned how to describe the topic let’s move on to the next element, describing the type.

7.3.2 Describing the Type

We consider the following types: commentary and annotations. In the previous sections, you have seen different ways to instruct ChatGPT to generate context types, such as *writing the context of a chart using 30 words*.

When describing the type, be as precise as possible, specifying the following aspects:

- The type (commentary, annotation, or in general, a text)
- The maximum number of words to generate.

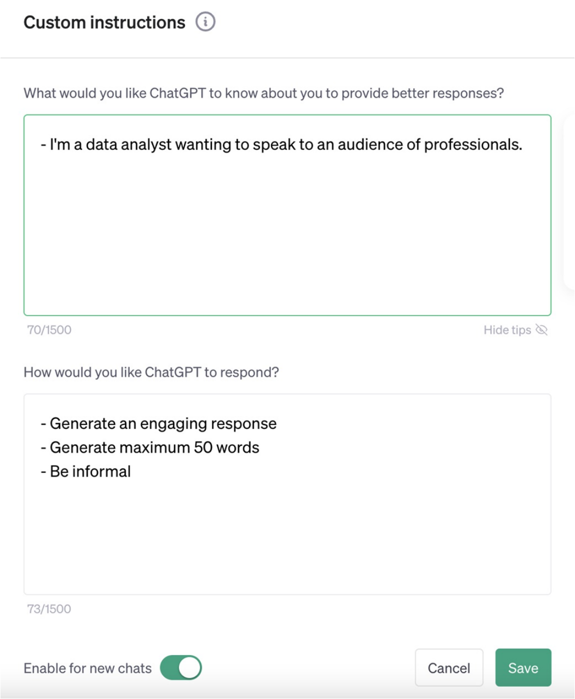
Our suggestion is to try different types and evaluate the results based on your needs.

7.3.3 Setting Custom Instructions

ChatGPT enables you to configure custom instructions for all your new chats. For example, if we build our charts for the same audience type and act with the same role, we can use this property as a default configuration for ChatGPT.

To enable custom instructions, access the ChatGPT web interface, click on the three dots near your profile, and then on Custom Instructions. In the new window, write the custom instructions. For example, you can use the first box to configure your role and the target audience and the second box for more details, such as the number of words to generate, the tone, and the style, as shown in Figure 7.8.

Figure 7.8 The custom instructions dialog box



What would you like ChatGPT to know about you to provide better responses?

- I'm a data analyst wanting to speak to an audience of professionals.

How would you like ChatGPT to respond?

- Generate an engaging response
- Generate maximum 50 words
- Be informal

Enable for new chats ●

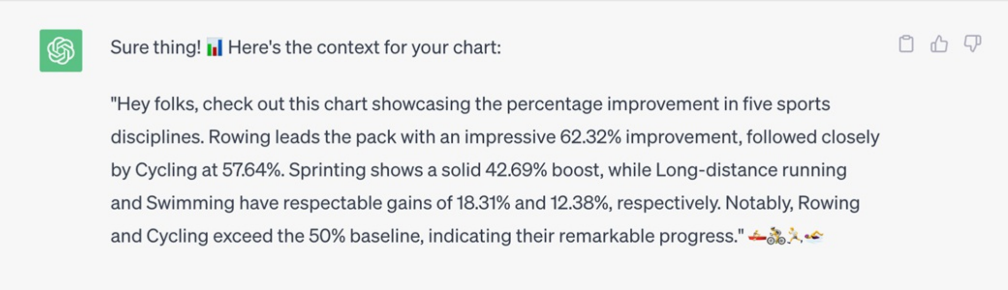
Click on the save button to enable the custom instructions for new chats. If you want to disable this property, access the dialog box and unselect the property Enable for new chats. Then, click on the Save button.

Let’s try the custom instructions with the previous text (we have removed the number of words to generate since we have configured them in the custom instructions):

> *Write the context of a chart for the following topic. There are five sports disciplines: Rowing (percentage improvement = 6.2.32%), Cycling (57.64%), Sprinting (42.69%), Long-distance running (18.31%), and Swimming (12.38%). Rowing and Cycling percentages are greater than the baseline of 50%.*

Figure 7.9 shows a possible output. For example, you can notice the informal tone we have set in the custom instructions.

Figure 7.9 A possible output produced when configuring custom instructions



Sure thing! 📊 Here's the context for your chart:

"Hey folks, check out this chart showcasing the percentage improvement in five sports disciplines. Rowing leads the pack with an impressive 62.32% improvement, followed closely by Cycling at 57.64%. Sprinting shows a solid 42.69% boost, while Long-distance running and Swimming have respectable gains of 18.31% and 12.38%, respectively. Notably, Rowing and Cycling exceed the 50% baseline, indicating their remarkable progress." 🚣🚴🏃🏊

For optimal usage of custom instructions in data storytelling, we suggest using them to configure the role and the audience. In addition, you can configure other specific details correlated to your job or data, as specified in the ChatGPT documentation.

Now that you have learned how to use ChatGPT for commentaries and annotations, let’s move on to the next step, using large language models in a different way. So far, you have used the web interface provided by OpenAI to write your prompt for ChatGPT. Now, we move to an advanced use of large language models, based on APIs calls.

7.4 Using Large Language Models for Context

So far, you have used three Generative AI tools: Copilot for code generation, ChatGPT for text generation, and DALL-E for image generation. All these tools are examples of application usage of a Large Language Model (LLM).

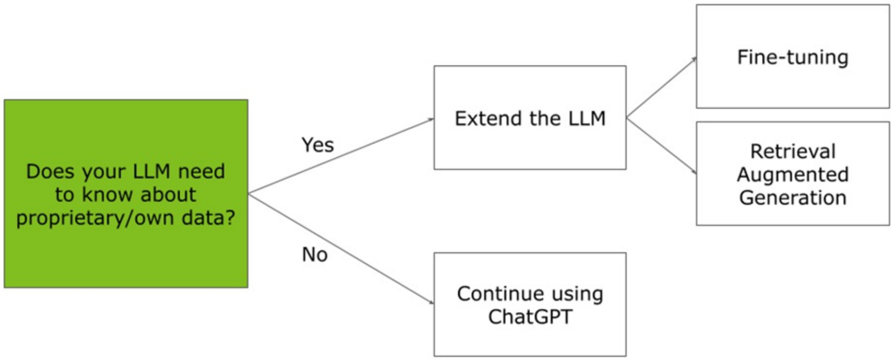
An LLM is a Machine Learning model aiming at predicting plausible language. The diffusion of LLMs has exploded since 2017, when Google researchers introduced the concept of transformers, a revolutionary architecture that allowed the training of large language models, such as Generative Pre-trained Transformers (GPTs), on which ChatGPT is based, and Bidirectional Encoder Representations from Transformers (BERT). Transformers allowed for the training of LLMs on massive datasets, resulting in models with incredible language generation capabilities.

In this book, we will not focus on how LLMs work. Instead, we aim to demonstrate you can use them effectively for Data Storytelling. However, if you're interested in delving deeper into the technical aspects, a vast bibliography is available on the topic (Vaswani 2017, Koenigstein 2024).

Before you embark on using LLM to build your data-driven story, it's essential to ask yourself whether the model needs to know specific information related to your domain of work, as shown in Figure 7.10. If the answer is no, then you can safely continue using ChatGPT. If, however, your answer is yes, then you can apply one of the following techniques:

- **Fine-tuning** adapts a pre-trained LLM to a specific domain by updating its parameters on task-specific data, optimizing its performance for that particular domain.
- **Retrieval Augmented Generation (RAG)** combines information retrieval and language generation, enabling LLMs to incorporate external knowledge sources during the generation process.

Figure 7.10 A criterion to establish whether to extend an LLM or not


Does your LLM need to know about proprietary/own data?
Yes
No
Extend the LLM
Continue using ChatGPT
Fine-tuning
Retrieval Augmented Generation

In the remainder of this section, we assume that your answer is yes and that you must tailor your LLM to your specific domain. A practical case where fine-tuning is useful could be when you must generate different data stories for the same type of audience or even the same audience. In this case you could build your database with the same structure of annotations, so that all new annotations have the same structure as the previous ones. This may generate some familiarity for your audience when they read your data stories. In other cases, you may need to use RAG, for example, when you have a long document and you want to build a short annotation for your data story based on it. Using RAG could help you to build textual summaries.

Now that you have learned the potential benefits of extending the LLM, let's start by analyzing the first strategy, fine-tuning.

7.4.1 Fine-tuning

GPT-3 was trained on 17 gigabytes of data, and GPT-4, the most recent model of OpenAI, has 45 gigabytes of training data. This means they contain a variety of information you can use in almost all cases. However, in some cases, fine-tuning on your model could provide better results.

Fine-tuning is the process of further training a pre-trained language model on a specific dataset that is more relevant to your specific domain. During fine-tuning, you use a smaller dataset, which typically contains examples and specific input-output pairs relevant to your task. In practice, the dataset is a collection of samples, each containing the prompt and the suggested completion.

When you apply fine-tuning to data storytelling, you can build a different dataset for each audience type, thus having better results. For example, you can build a dataset for the general public, another for professionals, and another for professionals. You can even create a different dataset for each scenario you work with (products, topic, and so on) and for each type of text you want to generate (title, annotation, commentary). The more specific your dataset is, the better your results will be.

Preparing the dataset is the most significant effort during the process of fine-tuning. In the remainder of this section, we will describe two strategies to prepare the dataset: manual building and building from sources. In both cases, we will use the OpenAI API. For more details on the installation, refer to Appendix A.

Manual building

Manual building involves defining each pair (prompt, completion) manually. This solution enables you to obtain the best results since you specify the exact behavior of your model, given a specific input. Consider, for example, the following pair:

- Prompt - generate a title for the general public about topic X
- Completion - X revealed to you!

Now, imagine that you have fine-tuned your model with that pair and want to use your model to generate titles for the general public. If you give *the theory of relativity* topic as input to your fine-tuned model, it will probably generate a title similar to the following one: *The theory of relativity revealed to you!*

The drawback of this strategy is that it is time-consuming because you must write each pair manually.

To start, you can define a minimum number of curated samples covering all your possible cases. The OpenAI model requires you to represent at least ten samples. Next, train the model. After that, proceed with the model evaluation by considering the original model (i.e., without fine-tuning) as a reference. Provide the same prompt to the two models, original and fine-tuned, and compare the produced outputs. Use your new model if your fine-tuned model performs better than the original one. Instead, if it performs worse than or has the same behavior as the original model, try to add new samples or improve the existing ones. Repeat this procedure until you reach a good result.

To show how manual building works, we will build a dataset tailored for the general public and generate a commentary as an output. In the previous chapter, you saw that you should use a different chart based on the information you want to convey. Here, we will build a different output based on the information to convey. We will build one or more (prompt, completion) pairs for each type of information to convey. Table 7.2 shows a possible dataset representing the described scenario. You can find the code described in this section in the GitHub repository of the book, under 07/manual-chatgpt-fine-tuning.

Table 7.2 The samples based on the information to convey

Information to convey	Prompt	Completion*
Parts of a whole	Percentage of participants in the conference by type (researchers 5%, students 30%, professors 65%)	More professors participated in the conference (65%). Researchers were not interested in the event (5%).
Comparison among entities	Comparison between Red (80%), Yellow and Green performance	Compared to Yellow and Green, Red experienced an improvement in performance of 80%.
Trend	Sales changed in the last 12 months (-12%) due to fewer subscribers and video views	Over the last 12 months, sales decreased by 12% based on various metrics, including subscribers and video views.
Outcomes of a survey or a questionnaire	Questionnaire: 3 positive answers 7 negative answers	3 out of 10 people answered the questionnaire with a positive answer.
Distribution	Sales of Product A (+30%) and Product B over the last 12 months	Compared to Product B, sales of Product A have increased by 30% over the last 12 months.
Spatial information	Sales in North Europe (+23%) compared to	Compared to South Europe, sales in North Europe

	South Europe	increased by 23%.
Relationship	The sales trendline from 1990 to 2020 increased by 120%	Between 1990 and 2020, sales increased by 120%.
Comparison among entities	Top ingredients for our recipe: sugar and salt	The chosen ingredients for our recipe are sugar and salt.
Comparison among entities	Comparison between Gold (30), Silver (20), and Bronze (40)	Bronze beats Silver and Gold with 40.
Distribution	Distribution of household chores (Cooking 35%, Cleaning 30%, Laundry 20%, Yard work 15%)	Cooking takes up the most significant portion at 35%. Cleaning follows at 30%, while laundry and yard work account for 20% and 15%, respectively.

* The word *completion* could be confusing but it is used by the OpenAI API. Completion refers to the output produced by the model.

Once you have built the dataset, you must format it as a JSONL file. This file contains a list of messages. Consider each message as a separate chat where you can specify a general configuration, the user prompt, and the assistant (model) output, as shown below:

Listing 7.1 The structure of a JSONL file

```
{
  "messages": [
     {
       "role": "system",
      "content": "You are a data analyst showing data to the gene
     },
     {
         "role": "user",
         "content": “Distribution of household chores
                                  (Cooking 35% Cleaning 30% Laund
     },
     {
          "role": "assistant",
           "content": "Cooking takes up the largest portion at 35
                      Cleaning follows at 30% while laundry
                      and yard work accounts for 20% and 15% resp

     }
  ]
}
```

Structuring a JSONL file

Use the keyword `messages` to define the list of samples. Imagine each sample as a separate chat, where you can specify the model role: `system`, for general configuration, `user`, for user prompt, `assistant`, for model output.

If your dataset is saved as a CSV file, use the code shown in Listing 7.2 and available in `prepare-data.py` to convert it into JSONL.

Listing 7.2 How to convert the CSV file into jSONL

```
import pandas as pd
import json

df = pd.read_csv('general-public.csv')

json_list = []

for index, row in df.iterrows():
    json_object = {
        "messages": [
            {
                "role": "system",
                "content": "You are a data analyst showing data t
            },
            {
                "role": "user",
                "content": row['prompt']
            },
            {
                "role": "assistant",
                "content": row['completion']
            }
        ]
    }
    json_list.append(json_object)

with open('general-public.jsonl', 'w') as outfile:
    for json_object in json_list:
        json.dump(json_object, outfile)
        outfile.write('\n')
```

Converting the dataset into JSONL

First, load the dataset as a Pandas DataFrame, next, format it in the JSONL format, as described in Listing 7.1. Finally, save the generated JSONL file.

Now, we are ready to fine-tune our model. We need an `OPEN_API_KEY`, as specified in Appendix A. If you are transitioning from a free to a paid plan, you might need to generate a new API key because the initial key does not work after the switch to a paid plan. Open a terminal and export your `OPEN_API_KEY` as an environment variable (`export OPENAY_API_KEY='my key'`). Next, upload the produced file to the OpenAI server, and when the uploading process is complete, create a job for fine-tuning. Listing 7.3 shows the code to perform these operations. Alternatively, read the `tune-model.py` script in the GitHub repository of the book. Remember that this option is exclusively available with the paid version.

Listing 7.3 How to fine-tune the model

```
import os
import openai
import time

openai.api_key = os.getenv('OPENAI_API_KEY') #A

dataset = openai.File.create(file=open('general-public.jsonl', 'r
print('Uploaded file id', dataset.id)

while True: #C
    print('Waiting while file is processed...')
    file_handle = openai.File.retrieve(id=dataset.id)
    if len(file_handle) and file_handle.status == 'processed':
        print('File processed')
        break
    time.sleep(3)
```

```
# Create a new fine-tuning job
job = openai.FineTuningJob.create(training_file=dataset.id, model

while True:
    print('Waiting while fine-tuning is completed...')
    job_handle = openai.FineTuningJob.retrieve(id=job.id)
    if job_handle.status == 'succeeded':
        print('Fine-tuning complete')
        print('Fine-tuned model info', job_handle)
        print('Model id', job_handle.fine_tuned_model) #D
        break
    time.sleep(3)
```

Fine-tuning the model

First, use the `openai.File.create()` method to create a new dataset and upload it to the OpenAI server. Next, use the `openai.FineTuningJob.create()` method to create a fine-tuning job using GPT-3.5-turbo. Wait until the job is completed. This could take a long time, depending on the dataset size. Once the model is trained, use the `fine_tuned_model` variable to print the information associated with the fine-tuned model.

Listing 7.4 shows an example of information printed after the execution of the `fine-tune-model.py` script. This fine tuning would cost around 0.05$.

Listing 7.4 An example of information associated with a fine-tuned model

```
Fine-tuned model info {
  "object": "fine_tuning.job",
  "id": "your model id",
  "model": "gpt-3.5-turbo-0613",
  "created_at": 1693347869,
  "finished_at": 1693348340,
  "fine_tuned_model": "ft:gpt-3.5-turbo-0613:personal::7t1Xuct5",
  "organization_id": "org-jWkYw8hPpaNwkesXezsWOwK8",
  "result_files": [
    "file-ro0BoeariIjOl7NSGRC80v8r"
  ],
  "status": "succeeded",
  "validation_file": null,
  "training_file": "file-InGnigMTto3YLrsiLuIUr7ty",
  "hyperparameters": {
    "n_epochs": 10
  },
  "trained_tokens": 5930
}
```

The fine-tuned model

You can see helpful information, including the model type, the model ID, the hyperparameters used, and so on.

Now, we can use the fine-tuned model to generate new commentaries tailored to the general public. Use the value corresponding to the fine_tuned_model key of the previous listing to refer to your model (`"ft:gpt-3.5-turbo-0613:personal::7t1Xuct5"` in the example).

To generate a new commentary, start a new chat session by using the `openai.ChatCompletion.create()` method, as shown in Listing 7.5 and in the generate-description.py script of the GitHub repository of the book. As a use case, consider again the example of Figure 7.1.

Listing 7.5 How to generate a new commentary

```
import os
import openai

openai.api_key = os.getenv("OPENAI_API_KEY")

model_id = "ft:gpt-3.5-turbo-0613:personal::7t1Xuct5"

completion = openai.ChatCompletion.create(
    model=model_id,
    messages=[
        {
            'role': 'system',
            'content': 'You are a data analyst showing data to th
        },
        {
            'role': 'user',
            'content': 'Top sports: rowing (62%) and cycling (58%
        },
    ],
)
```

```
print(completion.choices[0].message)
```

Generating a new commentary

Create a new `ChatCompletion` instance by specifying the model ID and the list of messages. The example defines only one message with the same system role as the fine-tuning dataset and the user role with a short description of our scenario.

Listing 7.6 shows an example of a possible produced output.

Listing 7.6 An example of a possible produced output

```
{
  "role": "assistant",
  "content": " \"The most popular sports are rowing and cycling w
}
```

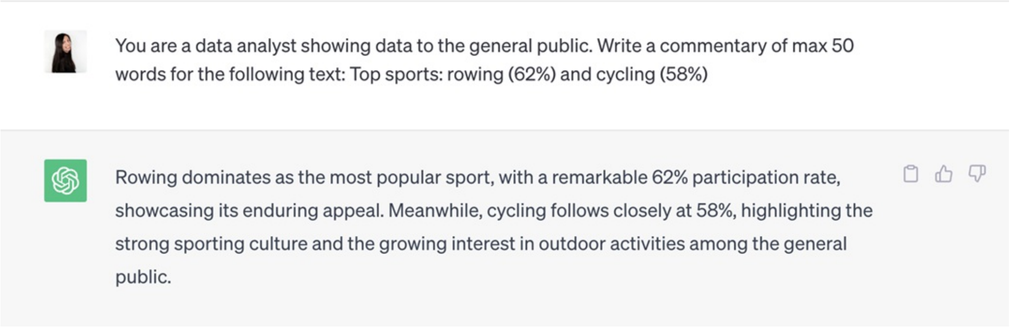
A possible output of the fine-tuned model

The output contains the role (assistant) and the content.

Incorporate the produced content into your chart as a commentary. This example has demonstrated how you can perform model fine-tuning using a manual dataset. In the example, the produced output is straightforward. If you want your model to produce more complex outputs, you must complicate your fine-tuning dataset, for example by adding new pairs (prompt, completion) specifically designed for your audience or your topic.

Just for comparison with the fine-tuned model, Figure 7.11 shows the output produced by ChatGPT (without fine-tuning) with a similar input.

Figure 7.11 The output produced by ChatGPT without fine tuning



You are a data analyst showing data to the general public. Write a commentary of max 50 words for the following text: Top sports: rowing (62%) and cycling (58%)

Rowing dominates as the most popular sport, with a remarkable 62% participation rate, showcasing its enduring appeal. Meanwhile, cycling follows closely at 58%, highlighting the strong sporting culture and the growing interest in outdoor activities among the general public.

Now that you have learned how to build your dataset manually, let’s move on to the next strategy, building from sources.

Building from sources

This strategy involves building your dataset from external sources, such as your company website, a domain-specific blog, etc. For example, if you work in the health field, you could download the title and abstracts of scientific papers about health. This enables you to build a dataset with a very domain-specific language. Or, if you work in the ICT field, you can download the titles and subtitles of blog articles from feeds to build your technical dataset. Anyway, you must pay attention to the data license in all cases. If the right explicitly prohibits their usage, you cannot use them, and you must search for other data sources. In some cases, contacting the data author directly could be sufficient, for example, if you want to download their data.

In the remainder of this section, we will build a dataset tailored to a technical audience by extracting feeds from my Medium blog. The objective is to build a chart title corresponding to the blog title, provided the blog subheading as an input. You can find the example in the GitHub repository of the book under 07/from-source-chatgpt-fine-tuning.

First, download the data. Ask Copilot to generate the code for you. Listing 7.7 shows the instructions for Copilot.

Listing 7.7 The instructions for Copilot

```
# import required libraries
# extract the title and link from the following rss/feed url: htt
# for each extracted link, extract the subheading from the articl
# create a dataframe with the following columns: 'prompt', ‘compl
# save the dataframe to a csv file called 'medium-articles.csv'
```

Writing the instructions for Copilot

Specify the feed URL and the information to extract for each item. Also, ask Copilot to generate the code to save the extracted items into a CSV file.

Copilot will generate an output similar to that shown in Listing 7.8. Set the prompt to the subheading and the completion to the title. Save the script and run it. You can find the code generated by Copilot in the GitHub repository of the book, in the script `download-raw-data.py`. You should see the `medium-articles.csv` file in your working directory.

Listing 7.8 How to extract data from feeds

```
import feedparser
import pandas as pd
import requests
from bs4 import BeautifulSoup

url = 'https://alod83.medium.com/feed'
feed = feedparser.parse(url)

titles = []
links = []
subheadings = []

for entry in feed.entries:
    titles.append(entry.title)
    links.append(entry.link)
    print(entry.link)
    response = requests.get(entry.link)
    soup = BeautifulSoup(response.content, 'html.parser')
    subheading = soup.find('h2', attrs={'class': 'pw-subtitle-par
    subheadings.append(subheading)

df = pd.DataFrame({'prompt': subheadings,'completion': titles})
df.to_csv('medium-articles.csv', index=False)
```

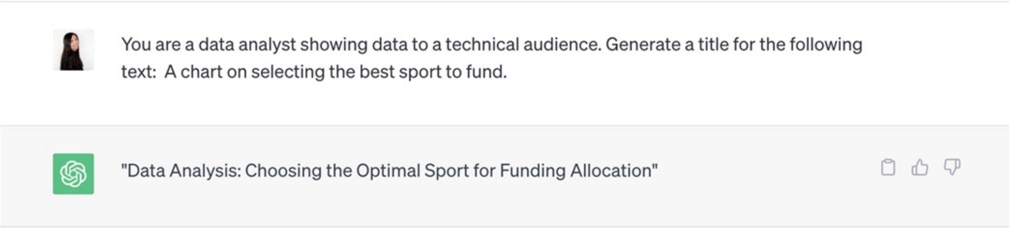
Extracting data from feeds

Use the `feedparser`, `requests`, and `bs4` libraries. If you don’t have them in your environment, install them using the pip package manager.

Once you have built the dataset, follow the procedure described in the manual building section (Section 7.4.1) to fine-tune the dataset (Listings 7.2-7.6). You can find the complete example in the GitHub repository of the book.

To test the fine-tuned model, provide the following prompt as input: *A chart on selecting the best sport to fund*. The model generates an output similar to the following: *How to Choose the Best Sport to Fund: A Data-Driven Approach*. Try a similar prompt with ChatGPT. Figure 7.12 shows a possible output. Since ChatGPT is not fine-tuned, you must specify more details in your prompt, as previously seen. Instead, for your fine-tuned model, describing the content in your prompt is sufficient.

Figure 7.12 The output produced by ChatGPT



You are a data analyst showing data to a technical audience. Generate a title for the following text: A chart on selecting the best sport to fund.

"Data Analysis: Choosing the Optimal Sport for Funding Allocation"

Now that you have learned how to perform fine-tuning for data storytelling, let’s move on to the next strategy to adapt your model to your specific context: Retrieval Augmented Generation (RAG).

7.4.2 Retrieval Augmented Generation

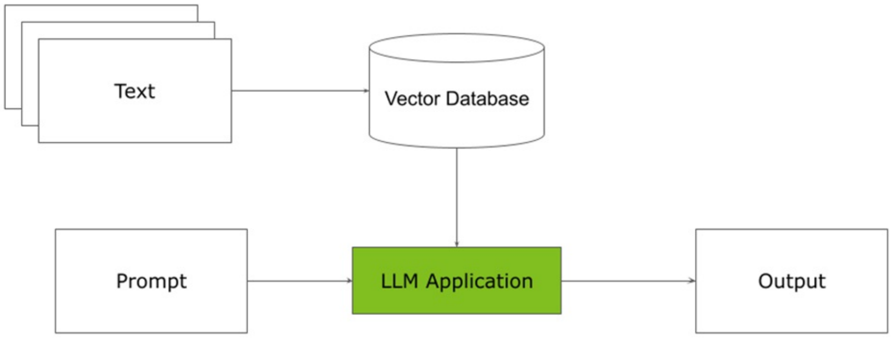
So far, you have seen how to adapt an LLM to a context by building an ad-hoc dataset. The effort, in this case, consists in preparing the dataset. Imagine how nice it would be to pass a text directly to the LLM without converting it to a specific format. Well, the good news is that this is possible thanks to Retrieval Augmented Generation (RAG).

RAG is an advanced Natural Language Processing (NLP) technique that combines elements of information retrieval and text generation. First, RAG performs a retrieval step, which queries an external knowledge source, such as a vast text corpus or a structured database. Next, RAG uses this knowledge source to enhance its response generation. RAG integrates the retrieved facts into its generated text.

In the data storytelling domain, you can use RAG to adapt your LLM to your specific topic, such as a product, real-time data, customer reviews, and other relevant information. For instance, by querying the knowledge base for specific product details, you can generate ad-hoc commentaries and annotations.

Imagine you want to build a RAG-based system that retrieves information about a product from your website company. Figure 7.13 shows the architecture of the RAG system we will implement.

Figure 7.13 A RAG-based system



First, we will download the text from a specified URL, split it, and represent it as vectors we store in a vector database. We will provide the vector database as an input to an LLM application, which can answer queries by querying the vector database. We will implement an example that generates commentaries for a specific smartphone, based on its description contained in an HTML page. In practice, we will load the HTML page into the vector database and then we will implement an LLM application to query it. We will use LangChain to implement the LLM application, Chroma for the vector database, and OpenAI for the LLM to make everything work. For more details on how to install these tools, refer to Appendix A.

In the remainder of this section, you will learn how to implement the described system. We will start by introducing LangChain. Next, we will see how to store data in Chroma, and finally, you will learn how to query the built system.

Introducing LangChain

LangChain is a framework that enables you to create applications that connect an LLM to other sources. LangChain supports different providers like OpenAI, Google, Microsoft, Hugging Face, and many more. In this book, we will focus on the models provided by OpenAI.

The core idea behind LangChain is the concept of a chain, which consists of several components from different modules. There are three main components:

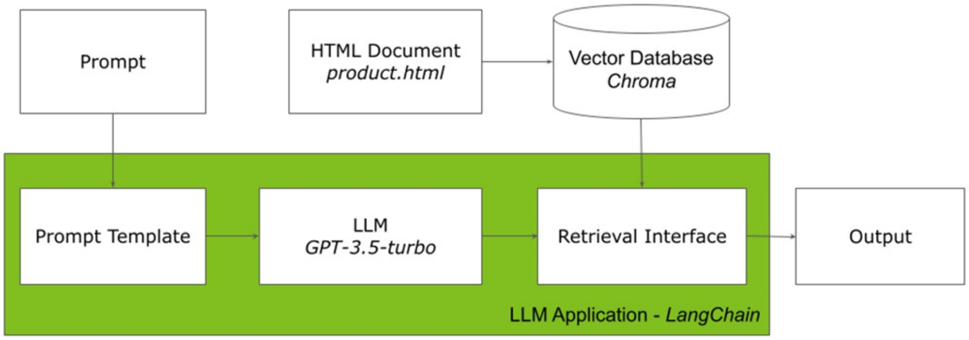
- *LLM wrappers* - wrappers for LLMs provided by external providers, such as OpenAI and Hugging Face
- *Prompt templates* - templates for different prompts, such as chatbot, question-answering, etc.
- *Indexes* - external structures you can use to provide additional context to an LLM.

In addition, the LangChain-based applications are *context-aware* because they connect LLM to external sources. In addition, such applications are useful because they can answer questions based on the provided context, what actions to take, etc.

The most straightforward chain consists of just one LLM chained with a prompt that enables you to query the model.

In the remainder of this section, we will implement a LangChain composed of the components shown in Figure 7.14: the vector database (Chroma), the prompt template, the LLM model (GPT-3.5-turbo, provided by OpenAI) and the retrieval interface.

Figure 7.14 The implemented architecture



You can find the full code described in this example in the GitHub repository of this book, under 07/rag.

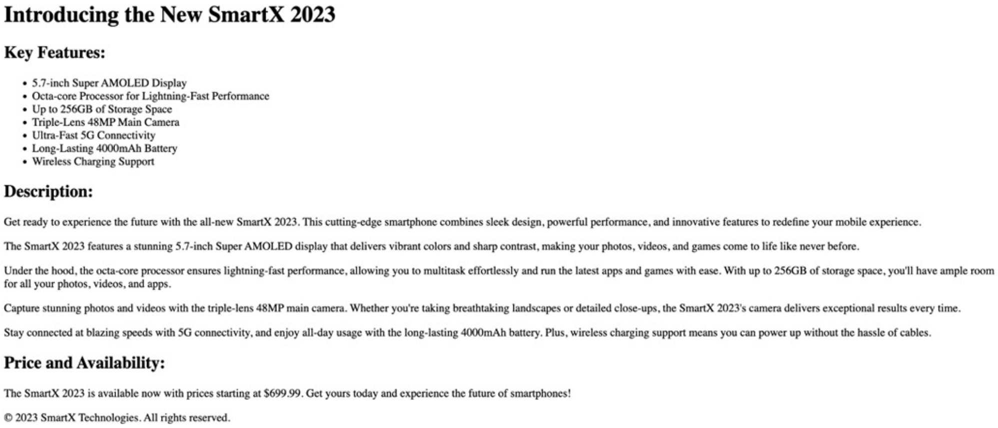
Let's start with the first component, the indexer.

The Indexer: Using Chroma

Chroma[1] is an embedding database you can use as an indexer for your LangChain. To install and configure Chroma, refer to Appendix A. An embedding is a numerical representation of data that is easy to index and retrieve, often for real-time tasks (Lane and Dyshel 2024). Before storing a text in Chroma, we must convert it into vector embeddings. For more details about embeddings, refer to the bibliography of this chapter.

Consider the product description available on a hypothetical website as shown in Figure 7.15. The objective of our task is to store the product description shown in Figure 7.15 in Chroma.

Figure 7.15 The HTML page with the product description



Introducing the New SmartX 2023

Key Features:

- 5.7-inch Super AMOLED Display
- Octa-core Processor for Lightning-Fast Performance
- Up to 256GB of Storage Space
- Triple-Lens 48MP Main Camera
- Ultra-Fast 5G Connectivity
- Long-Lasting 4000mAh Battery
- Wireless Charging Support

Description:

Get ready to experience the future with the all-new SmartX 2023. This cutting-edge smartphone combines sleek design, powerful performance, and innovative features to redefine your mobile experience.

The SmartX 2023 features a stunning 5.7-inch Super AMOLED display that delivers vibrant colors and sharp contrast, making your photos, videos, and games come to life like never before.

Under the hood, the octa-core processor ensures lightning-fast performance, allowing you to multitask effortlessly and run the latest apps and games with ease. With up to 256GB of storage space, you'll have ample room for all your photos, videos, and apps.

Capture stunning photos and videos with the triple-lens 48MP main camera. Whether you're taking breathtaking landscapes or detailed close-ups, the SmartX 2023's camera delivers exceptional results every time.

Stay connected at blazing speeds with 5G connectivity, and enjoy all-day usage with the long-lasting 4000mAh battery. Plus, wireless charging support means you can power up without the hassle of cables.

Price and Availability:

The SmartX 2023 is available now with prices starting at $699.99. Get yours today and experience the future of smartphones!

© 2023 SmartX Technologies. All rights reserved.

The first step involves loading the data from the URL, as shown in Listing 7.9. Since Chroma is fully integrated with Langchain, we will use Langchain to accomplish our task. LangChain supports multiple formats, including PDFs, URLs, etc.

Listing 7.9 How to load the HTML document in LangChain

```
from langchain.document_loaders import UnstructuredHTMLLoader
from langchain.text_splitter import RecursiveCharacterTextSplitte
from langchain.vectorstores import Chroma
from langchain.embeddings import OpenAIEmbeddings
from langchain.chains import RetrievalQA
from langchain.prompts import PromptTemplate
from langchain.chat_models import ChatOpenAI

# load data
loader = UnstructuredHTMLLoader('product.html')
data = loader.load()
```

Loading a text in LangChain

To load an HTML document in Langchain, build an `UnstructuredLoader()` object.

Next, split data in chunks of 20, as shown in Listing 7.10. We could have chosen any number smaller than the total text size for the chunk size.

Listing 7.10 How to split the text in chunks

```
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size = 100,
    chunk_overlap  = 20,
    length_function = len,
    is_separator_regex = False,
)

splitted_data = text_splitter.split_documents(data)
```

Splitting the text

Create a `RecursiveCharacterTextSplitter()` object to split the text into chunks.

After that, convert the split text into embeddings and store them in Chroma, as shown in Listing 7.11.

Listing 7.11 How to generate embeddings in Chroma

```
embeddings = OpenAIEmbeddings()

store = Chroma.from_documents(
    splitted_data,
    embeddings,
    ids = [f"{item.metadata['source']}-{index}" for index, item i
    collection_name='Product-Info',
persist_directory='db',
)
store.persist()
```

Generating embeddings from split text

First, create a new `OpenAIEmbeddings()` object. Next create a Chroma store with the split data and the embeddings and associate it with the collection `Product-Info`. Finally, store the Chroma store on the filesystem using the `persist()` method.

Now our vector store is ready, so we can move on to the next step, defining a prompt template.

Defining a Prompt Template

A prompt template is a pre-defined text used for generating prompts for LLMs. A prompt template may include instructions, examples, context and questions appropriate for your task. Listing 7.12 shows an example of a prompt we can provide as an input to our system.

Listing 7.12 How structure a prompt template

```
template = """You are a bot that answers questions about the prod
If you don't know the answer, simply state that you don't know.

{context}

Question: {question}"""

prompt = PromptTemplate(
    template=template, input_variables=['context', 'question']
)
```

Creating a prompt template

First, define the structure of your template. Use brackets to define input variables. In the example, there are two variables: `context` and `question`. Next, create a new `PromptTemplate()` object, and pass it the template and the input variables are parameters.

Once we have built the prompt template, we are ready to proceed with the last step, retrieval and query.

Retrieval Interface

A retrieval interface is an interface that enables us to combine the data stored in the Chroma database and the OpenAI LLM. We can use a retrieval to query our system and generate commentaries and annotations to incorporate in our charts. Listing 7.13 shows an example of the usage of a retrieval.

Listing 7.13 How to build a retrieval

```
llm = ChatOpenAI(temperature=0, model='gpt-3.5-turbo')

qa = RetrievalQA.from_chain_type(
    llm=llm,
    chain_type='stuff',
    retriever=store.as_retriever(),
    chain_type_kwargs={'prompt': prompt, },
    return_source_documents=True,
)

print(
    qa('Describe the product New SmartX 2023 using 30 words')
)
```

Creating a retrieval object

First, create an LLM instance using `ChatOpenAI()`. Set the temperature to 0 for conservative output. The temperature spans from 0 (low creativity) to 1 (high creativity). Set the model to `GPT-3.5-turbo`. Next, create a retrieval interface using `RetrievalQA()`, by specifying the LLM, the vector store (`retriever`), the prompt and other parameters. Set the `chain_type` to stuff, a pre-packaged document chain, that takes a list of documents, inserts them into the prompt which is then passed to the LLM. Finally, ask the question.

Listing 7.14 shows the produced output. You can insert the produced text (highlighted) in your chart.

Listing 7.14 The produced output

```
{'query': 'Describe the product New SmartX 2023 using 30 words',
    'result': 'The New SmartX 2023 is a cutting-edge smartphone w
    'source_documents':
    [Document(page_content='© 2023 SmartX Technologies. All right
        metadata={'source': 'product.html'}),
        Document(page_content='Get ready to experience the future
        metadata={'source': 'product.html'}),
        Document(page_content='Introducing the New SmartX 2023\n\
            metadata={'source': 'product.html'}),
        Document(page_content="you're taking breathtaking landsca
            metadata={'source': 'product.html'})
    ]
}
```

The produced output

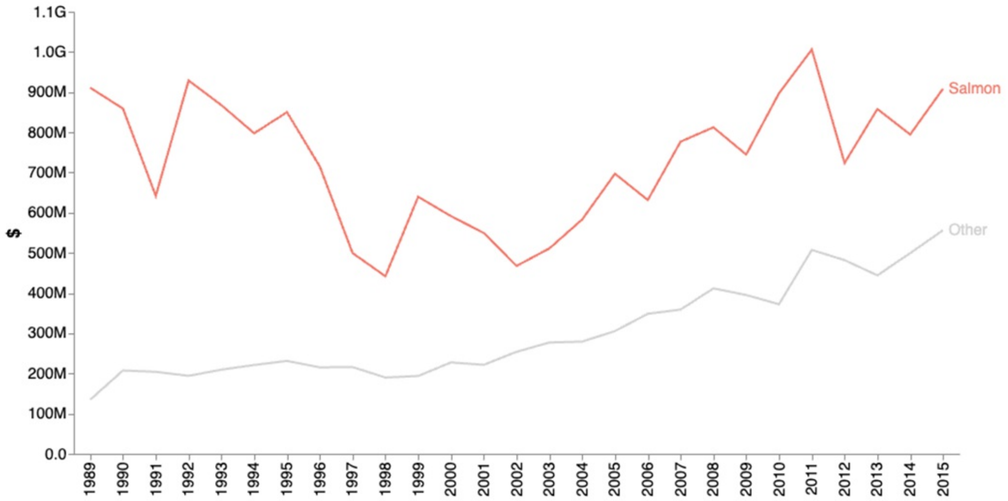
The output contains the text to insert in the chart (highlighted) and other useful information, such as the original query and the source documents.

Now that you have learned how to apply LLMs to build context in data storytelling, let’s move on to a practical example.

7.5 Case Study: From Information to Knowledge (part 1)

In the previous chapter, we have analyzed how to turn data to information in the aquaculture case study. Just for a quick reminder, the case study involved building a story around the problem of safety in the salmon aquaculture in the U.S. We decided to plot the salmon aquaculture sales trendline versus the other types of aquaculture. As an insight, we discovered that since 1998 there has been an increase in sales, following a period of decrease in sales from 1992 to 1998. We discovered that the decreasing period was partially due to some health problems in the salmon aquaculture. Figure 7.16 shows the chart produced at the end of the first step of the DIKW pyramid, from data to information.

Figure 7.16 The chart produced at the end of the data-to-information phase



To transform the chart into a data story, the next step involves turning information into knowledge. We will do it by:

- Some design considerations to tailor the chart to the audience
- Adding a commentary describing the general situation regarding safety in aquaculture
- Adding an annotation and a symbol to highlight the period of decrease in sales.

Let’s start with the first point, tailoring the chart to the audience.

7.5.1 Tailoring the Chart to the Audience

The scenario required to present the data story to an audience of executives, who require a chart easy to understand to make decisions quickly. In general, executives are familiar with trend lines, so we do not need to modify the chart. In addition, the chart is neither too detailed nor too simple. It contains the right level of detail that allows the audience not to be overwhelmed by information. Additionally, the chart doesn't give the impression of being sparse. Therefore, we can conclude that the chart type is perfect for our audience.

We also suppose that our audience is familiar with the $ symbol on the y-axis and the years on the x-axis, so we do not need to add any further specifications. We can leave the comparison between the salmon trendline and the others because it is useful for our audience to understand how the salmon sales behave compared to the other categories. It is not necessary to add further details.

Challenge: How could you tailor the chart to the general public or to an audience of professionals?

For the general public, you could consider simplifying the chart, for example by reducing the number of years. You may also need to better explain the y-axis meaning. For professionals, you could add more details, such as points with values for each year or you could even show the other aquaculture categories.

Now that we have discussed some design considerations to tailor the chart to the audience, let’s move on to the next step, adding a commentary. We will use RAG to generate the commentary.

7.5.2 Using RAG to Add a Commentary

We will add a commentary to the chart immediately under the title. Our commentary should explain how safety works in the U.S. aquaculture. We will build the commentary based on the Aquacultured Seafood document, which is an official website of the United States government. This document describes, among other aspects, the safety of aquaculture seafood.

You can implement a RAG-based system that builds required commentary using the code implemented in Listing 7.4.2. You only need to ask this question: *Describe the Safety of Aquaculture Seafood in the U.S.*. The code of the implemented RAG system is also available in the GitHub repository of the book, under CaseStudies/aquaculture/from-information-to-knowledge/rag.py. Listing 7.15 shows the produced output, containing the required commentary.

Listing 7.15 The produced output

```
{'query': 'Describe Safety of Aquaculture Seafood in the U.S.', '
```

The produced output

Use the produced output as a commentary for the chart.

Now we can add this text as a commentary for our chart. Listing 7.16 shows only the modifications to our original chart including the commentary. You can find the complete code in the GitHub repository of the book, under CaseStudies/aquaculture/from-information-to-knowledge/chart.py.

Listing 7.16 The commentary

```
commentary = ['Aquaculture seafood in the U.S. is regulated by th

'Regular inspections and testing are conducted to minimize risks
]

base = alt.Chart(df).encode(
    x=alt.X('YEAR_ID:O', title=''),
    y=alt.Y('AMOUNT', title='$',axis=alt.Axis(format='.2s')),
    color=alt.Color('CATEGORY',
        legend=None,
        scale=alt.Scale(range=range, domain=domain)
    )
).properties(
    width=800,
    height=400,
    title=alt.TitleParams(
        text='Aquaculture Exports of Salmon in the U.S.',
        subtitle=commentary,
        fontSize=20,
        subtitleFontSize=14,
        align='left',
        anchor='start',
        offset=20,
        color=color,
        subtitleColor='black'
    )
)
```

Adding a commentary to the chart

Use the title property to add a commentary to the chart, immediately before the title. Also, add a provisory title to the chart.

The next step involves highlighting the period of decrease in sales. So, let’s proceed with it.

7.5.3 Highlighting the Period of Decrease in Sales

The period of decrease in sales ranges from 1992 to 1998. We want to highlight it to let the audience know that during this period there were health problems in the salmon aquaculture. This will prepare the audience to consider respecting the safety rules to avoid having the same problems in the future. We will add two elements to highlight this decreasing period:

- A light-gray rectangle covering the decreasing period
- A textual annotation describing the health problems.

Listing 7.17 shows the code to build the rectangle.

Listing 7.17 The rectangle

```
N = 100000000 #A
y = df['AMOUNT'].max() + N

rect_df = pd.DataFrame({'x': [1992],
            'x2': [1998],
            'y' : [0],
            'y2': [y]
        })

rect = alt.Chart(rect_df).mark_rect(
    color='lightgrey',
    opacity=0.5
).encode(
    x='x:O',
    x2='x2:O',
    y= 'y:Q',
    y2= 'y2:Q'
)
```

Adding a rectangle to the chart

First, build a DataFrame with the rectangle coordinates. Next draw the rectangle using `mark_rect()`.

Challenge: Which instructions could you write for Copilot to speed-up the coding process?

You could try adding the following instruction to generate the rectangle: *# Add a rectangle starting from 1993 to 2000*. Which output do you obtain?

Listing 7.18 shows the code to add the annotation *The decline in sales was partially due to fish health issues*.

Listing 7.18 The annotation

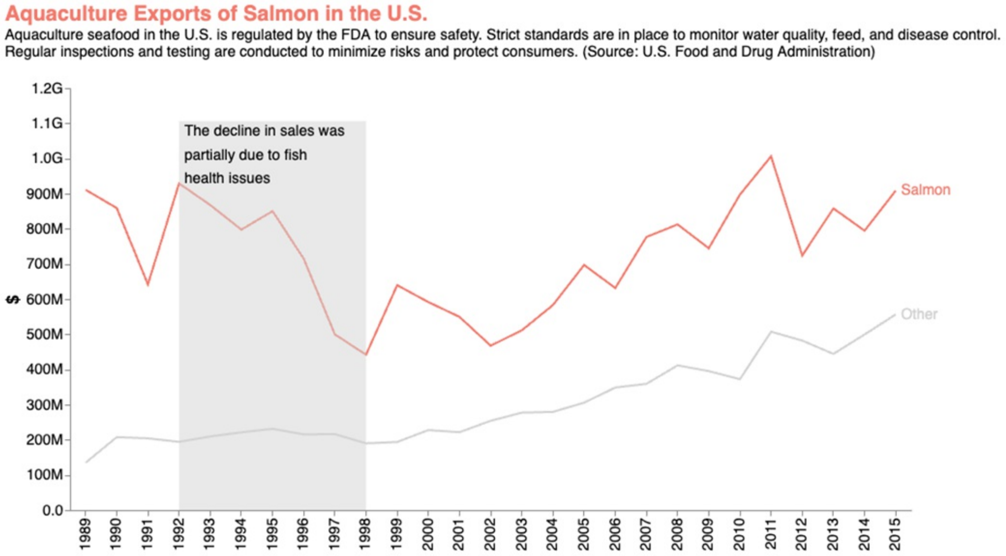
```
ann_df = pd.DataFrame({'x': [1992, 1992, 1992],
            'y': [y, y-N/3*2, y-N/3*4],
            'text': ['The decline in sales was',
            'partially due to fish',
            'health issues']
            })

annotation = alt.Chart(ann_df
).mark_text(
    align='left',
    baseline='middle',
    fontSize=14,
    dx=5,
    dy=10
).encode(
    x='x:O',
    y='y:Q',
    text='text:N'
)

chart = (chart + text + rect + annotation
).configure_axis(
    labelFontSize=14,
    titleFontSize=16,
    grid=False
).configure_view(
    strokeWidth=0
)
chart.save('chart.html')
```

Adding an annotation

First, build a DataFrame with the annotation text and its position information. Next, draw the annotation using `mark_text()`. Finally, plot and save the chart.

Figure 7.17 shows the final chart, after adding context. We have turned information into knowledge. In the next chapter, we will further enrich context by adding some images, and in Chapter 9, we will complete the story, by adding the next step, wisdom.

Figure 7.17 The chart produced at the end of the information-to-knowledge phase



Now that you have implemented a practical example to turn information into knowledge, let's further practice through a practical exercise.

7.5.4 Exercise

Modify the previous chart as follows:

1. Tailor the chart to an audience of professionals
    a. Add points to the salmon line chart. Suggestion: use `point=True` as a parameter of `mark_line()`
    b. Add values for each point. Suggestion: use `mark_text()` to add values for each point
2. Implement a RAG-based system to extract an annotation for the decreasing period from the Governor's Task Force on the Planning and Development of Marine Aquaculture in Maine Report and Recommendations (pp. 28-32). Suggestion: use `PDFMinerLoader()` to extract data from PDF. You may need to install some additional Python packages, including pdf2image, pdfminer, and pdfminer.six. You can find the solution in the GitHub repository of the book, under CaseStudies/aquaculture/from-information-to-knowledge/rag-annotation.py

In the first part of this chapter, you have learned how to turn information into knowledge by adding context to your data visualization chart. You have seen that context depends on the audience reading your chart. For example, if your chart will be read by a general public, avoid technical details, and use an appealing visualization. Instead, if your chart will be read by technical experts, add the many details as you can, while keeping the chart easy to read. In the second part, you have seen how to use generative AI tools as assistants to build your context. Finally, you have learned where to put the textual context in your chart.

In the next chapter, you will see how to add images to your chart to enrich context.

7.6 Summary

- Adding context to your data visualization is crucial for turning information into knowledge. Textual context includes all the relevant facts and events useful for the audience to understand data
- When you build a chart, tailor it to the audience. In general, there are three types of audiences: the general public, professionals, and executives.
- Use generative AI tools as assistants to help create context for your data. In particular, use ChatGPT to generate commentaries and annotations.
- If ChatGPT needs to know custom data or topics, extend your LLM with fine-tuning or RAG.
- Fine-tuning enables you to optimize a pre-trained LLM based on a dataset of prompt, completion pairs.
- Retrieval Augmented Generation (RAG) uses an external database, called vector database, to extend the LLM knowledge with domain-specific topics.

7.7 References

Embeddings

- Lane, H. Dyshel, M. (2024) *Natural Language Processing in Action, Second Edition*. Manning Publications
- OpenAI *Embeddings*

https://platform.openai.com/docs/guides/embeddings (Last Access 2023/10/10)

Fine-tuning

- Bantilan, N. (2023) *Fine Tuning vs. Prompt Engineering Large Language Models* https://mlops.community/fine-tuning-vs-prompt-engineering-llms/ (Last Access 2023/09/21)
- Jolley, E. (2023) *Introduction To Retrieval Augmented Generation* https://arize.com/blog-course/introduction-to-retrieval-augmented-generation/ (Last Access 2023/09/21)
- Marcelo X (2023) *How to fine-tune OpenAI GPT* https://medium.com/@marceloax.br/how-to-fine-tune-openai-gpt-3-d06741f915f4 (Last Access 2023/09/21)
- OpenAI API *Fine-tuning* https://platform.openai.com/docs/guides/fine-tuning (Last Access 2023/09/21)

LangChain

- Biswas, A. (2023) *How to work with LangChain Python modules* https://www.packtpub.com/article-hub/how-to-work-with-langchain-python-modules (Last Access 2023/09/17)
- Namaldesign *Introduction to Langchain* https://www.geeksforgeeks.org/introduction-to-langchain/ (Last Access 2023/09/21)
- Pinecone *LangChain: Introduction and Getting Started* https://www.pinecone.io/learn/series/langchain/langchain-intro/ (Last Access 2023/09/21)

LLM

- De Angelis, L., Baglivo, F., Arzilli, G., Privitera, G. P., Ferragina, P., Tozzi, A. E., & Rizzo, C. (2023). *ChatGPT and the rise of large language models: The new AI-driven infodemic threat in public health*. Frontiers in Public Health, 11, 1166120. https://doi.org/10.3389/fpubh.2023.1166120

- Google Developers. *Introduction to Large Language Models* https://developers.google.com/machine-learning/resources/intro-llms?hl=en (Last Access 2023/09/17)

RAG

- Jolley, E. (2023) *Introduction To Retrieval Augmented Generation* https://arize.com/blog-course/introduction-to-retrieval-augmented-generation (Last Access 2023/09/17)
- Needham, M. (2023) *Retrieval Augmented Generation with OpenAI/GPT and Chroma* https://www.youtube.com/watch?v=Cim1lNXvCzY (Last Access 2023/09/17)
- Needham, M. (2023) *Vector search LLM* https://github.com/mneedham/LearnDataWithMark/tree/main/vector-search-llm (Last Access 2023/09/17)
- Routu, V. (2023) *Answering with OpenAI and LangChain: Harnessing the Potential of Retrieval Augmented Generation (RAG)* https://www.linkedin.com/pulse/transforming-question-answering-openai-langchain-harnessing-routu/ (Last Access 2023/09/17)
- Schwaber-Cohen, R. *What is a Vector Database?* https://www.pinecone.io/learn/vector-database (Last Access 2023/09/21)

Thinking for the Audience

- Bettes, S. (2019) *Technical and professional writing genres* https://open.library.okstate.edu/technicalandprofessionalwriting/chapter/
  2/ (Last Access 2023/09/21)
- Chip *How to Identify Your Audience for Impactful Data Storytelling* https://www.quanthub.com/how-to-identify-your-audience-for-impactful-data-storytelling/ (Last Access 2023/09/21)
- Emery, A.K. (2021) *Why "Know Your Audience" is Terrible Dataviz Advice—and What to Do Instead* https://depictdatastudio.com/why-know-your-audience-is-terrible-dataviz-advice-what-to-do-instead/ (Last Access 2023/09/21)
- Technical Marketing *How to you engage and nurture your technical audience* https://www.linkedin.com/advice/0/how-do-you-engage-

nurture-your-technical-audience (Last Access 2023/09/21)

- WirelessLAN Professionals *How to present to a technical audience* https://wlanprofessionals.com/how-to-present-to-a-technical-audience/ (Last Access 2023/09/21)

Transformers

- Koenigstein, N. (2024) *Transformers in action*. Manning Publications
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). *Attention is all you need*. Advances in neural information processing systems, 30.

[1] https://www.trychroma.com/

8 From Information to Knowledge: Building the visual context

This chapter covers

- Emotions: The Foundations of Visual Context
- Colors, Size, and Interaction
- Using DALL-E for Images
- Strategic placement of textual and visual context
- Implementing an example

In this chapter, we will review the basic concepts behind visual context in a data story with a focus on emotions. We will describe how to set color, size and interactivity in a data story. Next, we will deepen the usage of DALL-E for building images. Afterward, we will focus on how to place the context in your chart. Finally, we will focus on a practical case study implementing the described concepts.

8.1 Emotions: The Foundations of Visual Context

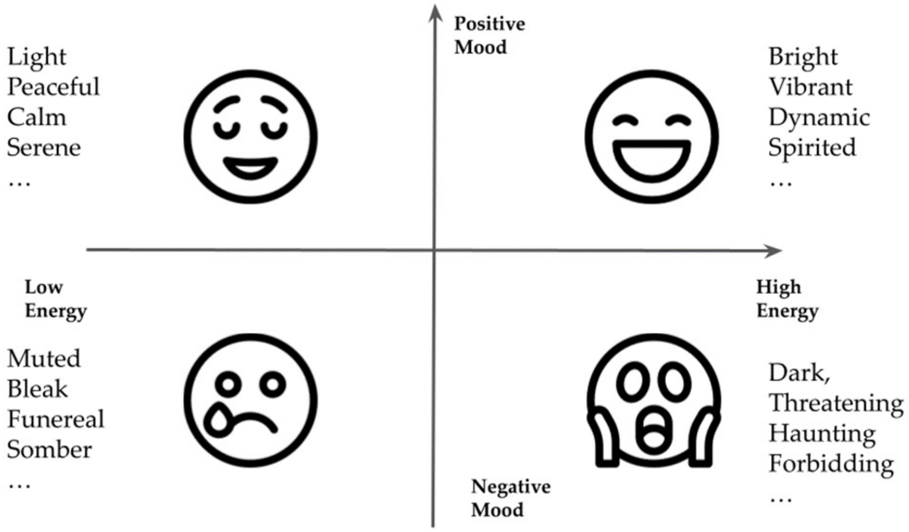
I went to the Natural History Museum with my family some time ago. There were many animals, reconstructions of archaeological excavations, some primitive men, and much more. At the end of the visit, the children were so enthusiastic that they talked about nothing else. Again, after a few days, the youngest told me he wanted to build an archaeological excavation in his grandmother's garden! The same should happen with our stories: our audience should still talk about it after some time. To make our story memorable, there is a way that directly involves our audience: **emotions**. Emotions are central to perceiving, interpreting, and remembering visual information. Emotions bridge the gap between raw data and human comprehension. Visual stimuli trigger emotional responses, influencing our perception and engagement with what we see.

The Collins dictionary defines an emotion as a feeling such as happiness, love, fear, anger, or hatred, which can be caused by the situation that you are in or the people you are with.

Emotions play a crucial role in the effectiveness of visual communication. The visual context, including colors, shapes, and layout, can evoke specific emotional responses in the audience. For instance, warm colors like red and orange might evoke feelings of excitement or urgency, while cooler tones like blue and green can convey calmness or trust. Intentionally using these emotional cues in visual elements can significantly impact how information is perceived and retained by the audience.

In their book, Communicate to Influence: How to Inspire Your Audience to Action, Ben and Kelly Decker, explain that we can understand emotions through the dual dimensions of energy and mood, which help categorize various emotional states based on the level of positivity/negativity of the emotion, as shown in Figure 8.1 (Decker 2015).

Figure 8.1 Macro categories for emotion classification



Emotions can vary widely along these dimensions, and individuals may experience and express emotions in unique ways based on their personal and cultural backgrounds.

Considering the energy axis (x-axis), we have two types of emotions:

- **High-energy emotions** are often associated with heightened physiological arousal. They typically involve activating the body's "fight or flight" response. Examples of high-energy emotions include excitement, anger, and fear. These emotions are characterized by increased heart rate, heightened alertness, and a sense of urgency. High-energy emotions are often expressed through animated body language, enthusiastic gestures, and rapid speech.
- **Low-energy emotions** involve reduced physiological arousal. A sense of calm and relaxation characterizes them. Examples of low-energy emotions include contentment, tranquility, and sadness. People experiencing low-energy emotions may exhibit slower movements, a quieter demeanor, and reduced physical tension. Low-energy emotions often lead to contemplation and introspection.

Considering the mood axis (y-axis), we have two types of emotions:

- **High-mood Emotions** are associated with positive valence, which evokes happiness, joy, and positivity. These emotions create an optimistic mood. High mood emotions often lead to smiles, laughter, and general well-being. Examples include happiness, excitement, and love.
- **Low-mood Emotions** are characterized by negative valence, leading to feelings of sadness, anger, or anxiety. These emotions create a more somber mood. Low mood emotions may result in expressions of sadness, frowns, and a general sense of distress. Examples include sadness, fear, and anger.

We can combine mood and energy to generate different emotions, as shown by the emoticons in Figure 8.1.

Incorporating emotions in data stories is not a mere aesthetic choice but **a strategic approach to conveying knowledge**. By aligning emotions with the intended message, we can transform the audience's passive reception of information into an active quest for knowledge and understanding. For example, high-energy emotions can capture immediate attention and incite curiosity, driving the viewer to explore the visual content more deeply.

Conversely, low-energy emotions can encourage reflection, allowing the audience to contemplate the content more profoundly.

The quickest and most effective way to arouse emotions is through **sight**. Just as my children at the Natural History Museum were impressed by what they saw, our audience should be impressed by what they see in our visual representations. There are different ways to capture the audience’s attention through sight, such as chart colors, size, interactivity, and images. There are tons of books and resources dealing with all these types of sights thus, this book will give only a general overview of them with a focus on how to configure them in Altair. You can refer to the bibliography for more details. In addition, this chapter will focus on how to use Generative AI, specifically DALL-E, to generate images with specific emotions.

Let’s start with a general overview of colors and how to implement them in Altair.

8.2 Colors

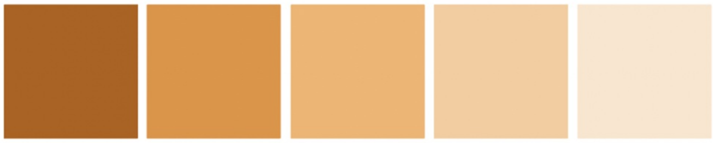
Color helps communicate a message by setting the scene and the tone. It creates an emotional connection between the data and the audience. Color comprises three main components:

- **Hue** is the dominant color wavelength. It identifies color family or color name (such as red, green, purple)
- **Saturation** describes the intensity of that color. It defines how pure a color is
- **Brightness** measures how light or dark a color appears.

In the previous chapters, we saw that every data story has a main character and an adversary. When drawing a chart, choose a primary color to highlight your main character, and always use it when referring to them. Use an alternative color for your adversary, but if you want to move the adversary to the background, use a gray tonality for them. You can choose colors using one of the following palettes:

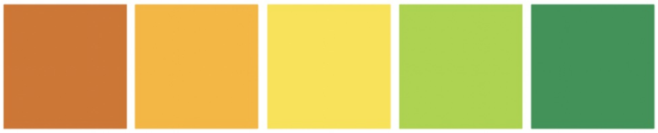
- **Sequential color palettes** organize quantitative data from high to low using a single color in various gradients or saturations. Use this palette to show a progression rather than a contrast (Figure 8.2).

Figure 8.2 An example of a sequential color palette



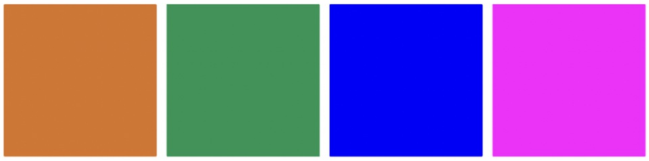
- Diverging color palettes highlight the ranges of quantitative data by using two contrasting hues on the extremes and a lighter-tinted mixture to highlight the middle range. A diverging palette shows where data are on a spectrum, such as cold to hot (Figure 8.3). Cold could represent lower quantitative data values, typically depicted with a cooler hue, while hot could refer to higher values, often represented with a warmer hue.

Figure 8.3 An example of a diverging color palette



- **Qualitative color palettes** highlight qualitative categories. This palette type uses a different hue for each concept to represent (Figure 8.4).

Figure 8.4 An example of a qualitative color palette



You can use colors to focus attention and highlight important information the chart wants to show. In addition, use color to create associations, such as brown and green, to recall nature and shades of yellow if you talk about bees. You can even use colors **to evoke emotions**, such as red for energy and blue for peace. Do not use overlapping colors to represent different concepts or too many colors.

Challenge

Consider the case study implemented in Chapter 5 about homeless people. We used a tonality of green to color bars. This should evoke hope in the audience. The objective of the color was to demonstrate that our project would help solve the homeless problem. What happens if we use another color, such as #6F4E37 (coffee)? Probably, the message we transmit to the audience is that our project will not solve the homeless problem.

8.2.1 Setting Colors in Altair

In the previous chapters, we have seen how to set colors in Altair using the color channel. In this section, we will give more details about it. The following listing shows a quick reminder of how to use colors in Altair:

Listing 8.1 Setting color in Altair

```
import pandas as pd
import altair as alt

df = pd.DataFrame({
    'Category' : ['A', 'A', 'B', 'B', 'C', 'C', 'C'],
    'Cost' : [12,45,64,23,45,78,72],
    'Sales' : [34,56,45,34,44,23,23],
    'Product' : ['P1','P2','P3','P4','P5','P6','P7']
})

chart = alt.Chart(df).mark_bar().encode(
        x=alt.X('Product:N'),
        y=alt.Y('Cost:Q'),
        color=alt.Color('Category:N')
    )

chart.save('chart.html')
```

How to set color in Altair

Use the `alt.Color()` channel to set the color.

The color channel may receive a specific color palette using the `scheme` parameter as input. Refer to the Altair documentation for the supported palettes. In addition, you can use the scale parameter to specify the range and the domain of your colors.

- The **range** parameter specifies the output range of the scale. For example, when mapping a quantitative variable to color, use a color scale ranging from blue to red. Specify the range as a list of colors or as one of Altair's built-in color schemes.
- The **domain** parameter specifies the input domain of the scale. For example, if we have a quantitative variable with values between 0 and 100, set the domain of a color scale to [0, 100].

8.2.2 Exercise: Setting Colors

Consider the case study implemented in Chapter 3, *Population in North America over the last 50 years*. You can find the complete code of this case study in the GitHub repository of the book, under CaseStudies/population/population.ipynb. The final data story compared the population in North America with that in the rest of the world and showed a gap of 367 million people. We used a tonality of green to represent the North American trend line.

1. Change the color of the main chart (i.e., the chart showing the difference from 1960) to #963232 (a tonality of red), while keeping the green tonality to the next step chart (i.e., the chart entitled Strategies for Population Growth in North America).
2. Compare the obtained results with the chart implemented in Chapter 3. Did the use of different colors change the message in any way? You will probably notice a negative emphasis on the fact that population growth in North America is slower than in the rest of the world. However, the next steps still remain optimistic.

A Possible Solution

There are different ways to obtain the same result. A possible solution is to set the colors locally for each chart. For example, you can set the colors for the next step charts as shown in the following listing.

Listing 8.2 Setting color in the next step chart

```
color='#80C11E'

df_cta = pd.DataFrame({
    'Strategy': ['Immigration Development', 'Enhance Family-Frien
    'Population Increase': [20, 30, 15]  # Sample population incr
})

# Creating the stacked column chart
cta = alt.Chart(df_cta).mark_bar(color=color).encode(
    x='Population Increase:Q',
    y=alt.Y('Strategy:N', sort='-x', title=None),
    tooltip=['Strategy', 'Population Increase'],
).properties(
   title=alt.TitleParams(
       text='Strategies for population growth in North America',
       color=color)
)
```

How to set color locally to a chart

Add a local color to a chart thus, when you configure all the layered charts, this chart is not affected by the global configuration.

This example shows a basic example of conveying emotions using colors. For more details, refer to specific books about the topic, such as those in the bibliography. Now that you have learned the basic concepts of conveying emotions using colors let’s move on to the next step, using size.

8.3 Size

When creating a chart, consider the size, as it can significantly affect how the audience perceives your data. Size impacts your visual representation's readability and overall impact and effectiveness. Choosing an appropriate size for your chart depends on multiple factors, such as where you will display it, what kind of content you want to show, and who will view it. The best way to determine the correct size is through experimentation; try different sizes until you land on one that looks good enough.

Moreover, choosing a smaller or larger-sized chart affects how much information can fit into one frame without cluttering or squishing text together. It also influences how easily readers can read all data points without straining their eyes or getting lost in too many elements.

You can configure two types of sizes in a chart: font size and chart size.

8.3.1 Font Size

Font size directly affects the readability and comprehensibility of the data. The font size should be set to make it easy for the reader to understand and interpret the information presented in the chart.

When setting up a chart, you can specify different font sizes for different chart components like titles, axis labels, or tick labels. A good practice is to keep them consistent throughout your plot so that they look visually appealing and organized.

By default, Altair sets the font size to 12px. However, you can adjust the font size to make your charts more legible or to match the style.

To change the font size in Altair, identify which part of your visualization you want to modify. Once identified, use Altair's `configure_*()` method along with the axis, title, text, or legend property based on what component you want to modify. You can then pass parameters like `fontSize` or `labelFontSize` along with their desired value.

It is essential not to go overboard with increasing font sizes, as larger fonts might lead to cluttered plots that are hard on eyesight. On the other hand, smaller fonts may result in difficult-to-read charts. So always pick a reasonable font size that complements your charts well without negatively affecting readability.

8.3.2 Chart size

The final aspect to consider when setting the size of your Altair chart is the actual chart size. You can adjust this using the width and height arguments in your property. Both arguments are expressed in pixels.

There is no one-size-fits-all approach when determining the appropriate chart size. It will depend on factors such as where you plan on displaying the chart, what type of data it represents, and how much detail you want to include.

However, you can follow the following general guidelines:

- If you want to create a simple chart to include in an email or presentation slide, then use a smaller size
- If you want to create a complex chart, use a smaller size.

To set the chart size in Altair, use:

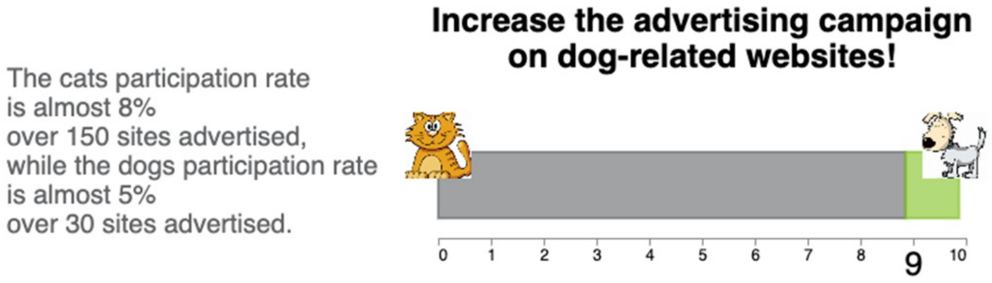
- properties(width=my_width, height=my_height) for single charts
- configure_view(width=my_width, height=my_height) for single or compound charts.

8.3.3 Exercise: Setting Size

Consider the case study implemented in Chapter 1, Increase the advertising campaign on dog-related websites!

1. Change the size of the x-axis label only for the value 9, as shown in Figure 8.5.
2. Compare the obtained result with the original chart. Is the chart clearer?

Figure 8.5 The case study of Chapter 1 with increased font size in correspondence of value 9.



Solution

The following listing shows how to encode the x-channel to increase the font size for just one axis label.

Listing 8.3 Setting label size

```
x=alt.X('x:Q',
        scale=alt.Scale(domain=[0, 10]),
        axis=alt.Axis(tickMinStep = 1,
            grid=False,
            title=None,
            orient='bottom',
            labelFontSize=alt.condition(alt.datum.value == 9,
                alt.value(25),
                alt.value(12)))
        ),
```

How to set a label size based on a condition

Use the `alt.condition()` method to set the label size based on a condition.

Now that you have learned how to configure size in Altair, let’s move on to the next step, interactivity.

8.4 Interaction

So far, we have implemented static charts that present a static story to the audience. The benefit of this chart type is that you decide the story and guide the audience through it. In other words, you don’t allow the audience to explore data autonomously.

Adding interactive elements to a chart, such as buttons, menus, and zoom options, transforms a static chart into a dynamic one. With a dynamic chart, you build dynamic data stories, where the audience builds their own story. With a dynamic chart, you lose control over the audience and the possibility of having a focused and precise message. The audience is free to explore data and extract their insights, potentially leading to other calls to action. For this reason, it’s important to design in advance what parts of the charts you want to make browsable by the audience based on the message to communicate.

Use interactive charts when you want the audience to explore data dynamically or when dealing with large datasets. For example, for financial data or in an educational context, an interactive chart allows the audience to zoom in on specific timeframes or hover over data points for details.

Conversely, you may prefer static charts for simple presentations or reports where the data doesn't require exploration. For instance, a static chart could be ideal for a printed annual report that presents trends or comparisons without needing user interaction. Additionally, in situations where the message needs to be straightforward and consistent, static charts may be more effective.

Consider, for example, the variant of the scenario described in Chapter 3, *Population in North America over the last 50 years*. The chart contains a dropdown menu enabling you to select the reference country to show in the chart. This chart type is okay for a data exploratory phase, but it is not to build a data story simply because it has no focus. Instead, if you focus on North America and leave the audience the possibility to compare it with another country, you are still telling the story about the population in North America while giving the audience the freedom to compare it with other countries.

There are two main types of interactivity:

- **Passive interactivity** refers to features that allow the audience to interact with the chart without changing its state, such as tooltips or hover effects.
- **Active interactivity** involves changing the chart's state in response to user input.

In the remainder of this section, we will see how to add a tooltip, a slider, and a dropdown menu to an Altair chart.

8.4.1 Tooltip

A tooltip is a tiny pop-up box that appears when you hover over certain chart elements, such as data points or bars. To add a tooltip to your Altair chart, specify which element(s) you want it to appear for. In Altair, you can do it through the tooltip encoding channel using square braces `[]`, as shown in Listing 8.4. The example uses the population dataset of Chapter 3, focusing on a completely different insight: the population growth in 2018 by continents.

Listing 8.4 Setting a tooltip

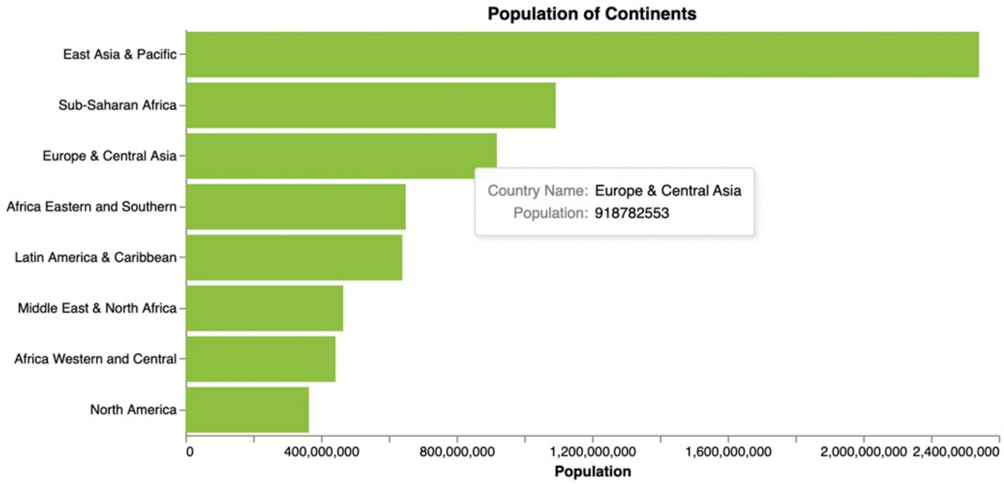
```
import pandas as pd
import altair as alt

df = pd.read_csv('data/population.csv')

# [...]

df = df[df['Country Name'].isin(continents)]

color = '#80C11E'

# Create visualization
chart = alt.Chart(df).mark_bar(
    color=color
).encode(
        y=alt.Y('Country Name:O', sort='-x', title=''),
        x=alt.X('Population:Q'),
        tooltip=['Country Name', 'Population']
).transform_filter(
    alt.datum.Year == 2018
)

# [...]

chart.save('tooltip.html')
```

How to set a tooltip

Use the `tooltip` channel to set a tooltip in Altair.

You can find the complete code of the example in the GitHub repository of the book under 08/tooltip.py. Figure 8.6 shows a snapshot of a tooltip produced when hovering over a bar.

Figure 8.6 An example of a tooltip



Now that you have learned how to generate a tooltip in Altair, let’s move on to the next interactive element, the slider.

8.4.2 Slider

A slider is a graphical element that allows users to input or adjust a value within a specified range by moving a visual indicator along a track or bar. To build a slider in Altair, follow the steps described below:

1. Create a `binding_range()`, an object that defines the range of values to which the slider can be set
2. Create a selection using the `selection_point()` method. The selection is an object that defines how the slider will interact with chart
3. Add the selection to the chart using the `add_params()` method
4. Optionally, apply the `transform_filter()` method to the chart to show only the selection.

Listing 8.5 shows an example of the usage of sliders. The example still uses the population dataset of Chapter 3, focusing on the population growth over the years by continents. You can find the complete code in the GitHub repository of the book under 08/slider.py. The listing shows only the relevant code to implement a slider. For the other details, read the code in the GitHub repository.

Listing 8.5 Setting a slider

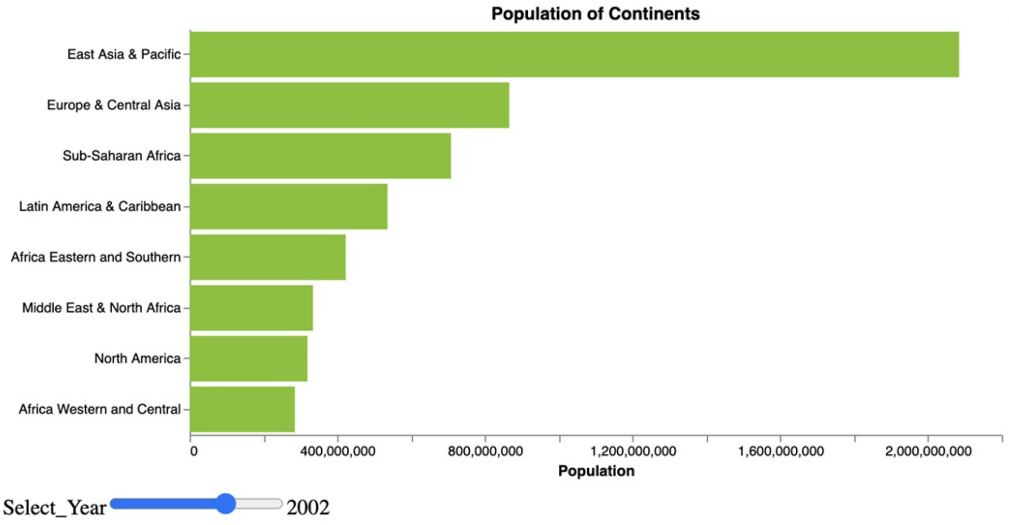
```
import pandas as pd
import altair as alt

df = pd.read_csv('data/population.csv')

# [...]

color = '#80C11E'
slider = alt.binding_range(min=1960, max=2021,step=1)

select_year = alt.selection_point(name="Select", fields=['Year'],
                                   bind=slider)

# Create visualization
chart = alt.Chart(df).mark_bar(
    color=color
).encode(
        y=alt.Y('Country Name:O', sort='-x', title=''),
        x=alt.X('Population:Q'),
).add_params(
    select_year
).transform_filter(
    select_year
)

# [...]

chart.save('slider.html')
```

How to set a slider

After importing the required libraries, use the `binding_range()` function to define a binding range for the slider with a minimum of 1960, maximum of 2021, and step size of 1. Next, use the `selection_point()` function to create a selection named "Select" and specify the field it should bind to (in this case, 'Year') using the slider. After that, use the `add_params()` method to add the select_year selection to the chart and the `transform_filter()` method to filter the data based on the selected year from the slider.

Figure 8.7 shows a static representation of the resulting chart.

Figure 8.7 An example of a slider



Now that you have learned how to generate a tooltip in Altair, let’s move on to the next interactive element, the dropdown menu.

8.4.3 Dropdown Menu

A dropdown menu is a graphical element that typically appears as a small rectangular box with an arrow or triangle icon, indicating that additional options or choices are available. When a user clicks or hovers over the dropdown menu, a list of selectable items or options "drops down" or expands below or above the menu. The user can then choose from the available options by clicking on them. To create a dropdown menu in Altair, follow these sequential steps:

1. Begin by generating a `binding_select()` object. This object defines the list of values assigned to the dropdown menu.
2. Create a selection using the `selection_point()` method. This selection object governs the interaction between the dropdown menu and the

chart.

3. Integrate the selection into the chart using the `add_params()` method.
4. Apply the `transform_filter()` method to display only the selected data.

Listing 8.6 shows an example of the usage of a dropdown menu. The example uses the population dataset of Chapter 3, focusing on the population growth over the years by continent. The dropdown menu enables you to select a specific continent. You can find the complete code in the GitHub repository of the book under 08/dropdown.py. The listing shows only the relevant code to implement a slide. For the other details, read the code in the GitHub repository.

Listing 8.6 Setting a dropdown menu

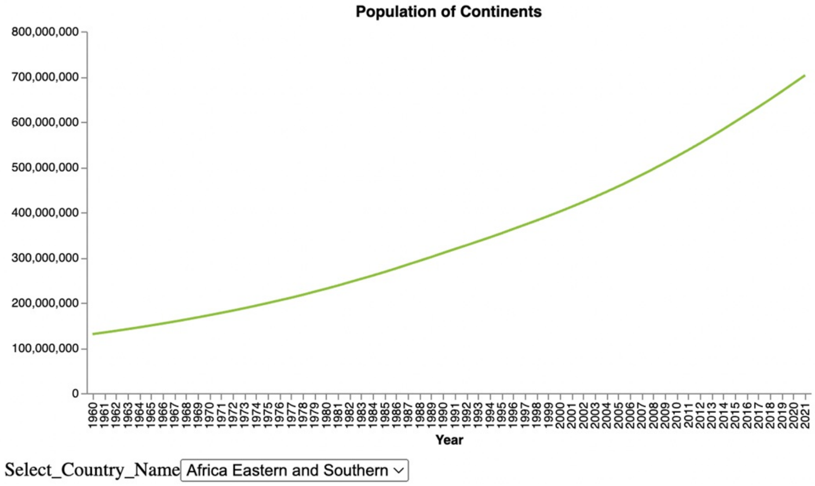
```
import pandas as pd
import altair as alt

df = pd.read_csv('data/population.csv')

# [...]

color = '#80C11E'
input_dropdown = alt.binding_select(options=df['Country Name'].un
select_country = alt.selection_point(name='Select',fields=['Count


# Create visualization
chart = alt.Chart(df).mark_line(
    color=color
).encode(
        y=alt.Y('Population:Q', title=''),
        x=alt.X('Year:O'),
).add_params(
    select_country
).transform_filter(
    select_country
)

# [...]
chart.save('dropdown.html')
```

How to set a dropdown menu

After importing the required libraries, use the `binding_select()` function to define a binding selection for the dropdown menu. Next, proceed as you do for sliders

Figure 8.8 shows a static representation of the resulting chart.

Figure 8.8 An example of a dropdown menu



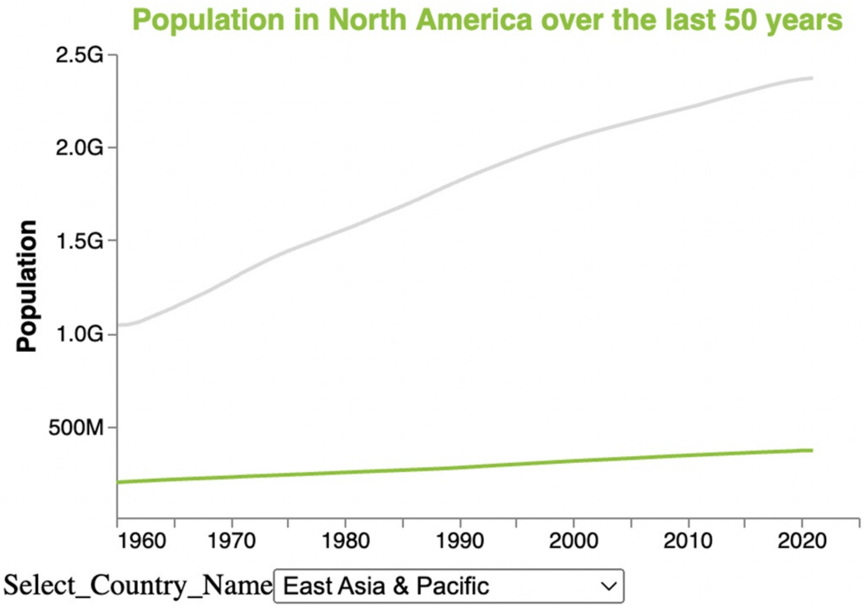
Now it’s time to practice. So let’s proceed with an exercise.

8.4.4 Exercise: Setting Interactivity

Consider the case study implemented in Chapter 3, *Population in North America over the last 50 years*. You can find the complete code of this case study in the GitHub repository of the book, under CaseStudies/population/population.ipynb.

1. Remove the *Others* line and add a dynamic line showing a selectable continent from a dropdown menu, as shown in Figure 8.9. Leave the North America line static.
2. Compare the obtained result with the original chart. For which type of audience is this dynamic graph most suitable?
3. Optionally, adapt all the graph elements to the animations.

Figure 8.9 The case study of Chapter 3 with a dynamic dropdown menu for the continent to compare with North America.



You can find the solution to the exercise in the GitHub repository of the book, under [08/population-exercise.py]()

Now that you have learned how to introduce interactivity in Altair through tooltips, sliders, and dropdown menus, let’s move to the next element, images, focusing on DALL-E.

8.5 Using DALL-E for Images

Imagine reading a text that describes the following scene: a beautiful little girl with black hair, wearing a yellow dress, and dark eyes, running in a green meadow full of yellow flowers. You can use your creativity to set up the scene in your mind, but this could require a little effort from your brain. Now look at Figure 8.10 (realized with DALL-E), showing the same scene visually. You see exactly what we described before, but surely you depicted the scene differently in your mind.

Figure 8.10 An image of a little girl wearing a yellow dress

The same situation occurs when you describe the subject of your data story. If you don't give a face to your subject and give only a general description, your audience is free to imagine the subject in various ways depending on their background. Instead, if you add one or more images to your data story, you help the audience have a clear representation of the story's subject. You will bring the audience up to the intended emotional level through images, especially photos, and they will be more open to listening to your data story. Adding images to a data story reinforces the context and prepares the audience to accept the data story message.

In Chapter 4, you have already learned how to draw images in DALL-E and insert them into a chart. In this section, you'll see how to use DALL-E-generated images to evoke emotions. We'll also look at how to maintain consistency in the images we're generating in DALL-E. Let's start with emotions.

8.5.1 Adding Emotions

The DALL-E book of prompts suggests some keywords to add emotions to images. Figure 8.1 shows some possible adjectives associated with each emotion category:

- High Energy, Positive Mood - Bright, Vibrant, Dynamic, Spirited
- High Energy, Negative Mood - Dark, Threatening, Haunting, Forbidding

- Low Energy, Positive Mood - Light, Peaceful, Calm, Serene
- Low Energy, Negative Mood - Muted, Bleak, Funereal, Somber.

You can read the complete list of keywords in the DALL-E book of prompts[1].

Using emotional keywords enables you to obtain different results. Figure 8.11 shows a possible output produced by DALL-E, for the following input: *a calm woman, playing in a garden, photograph.*

Figure 8.11 An image of a calm woman in a garden.

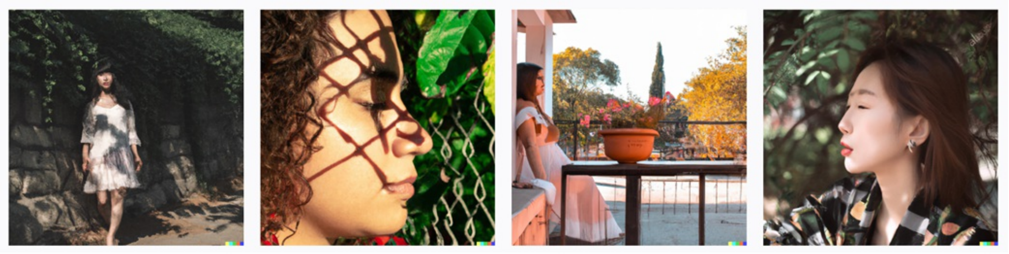
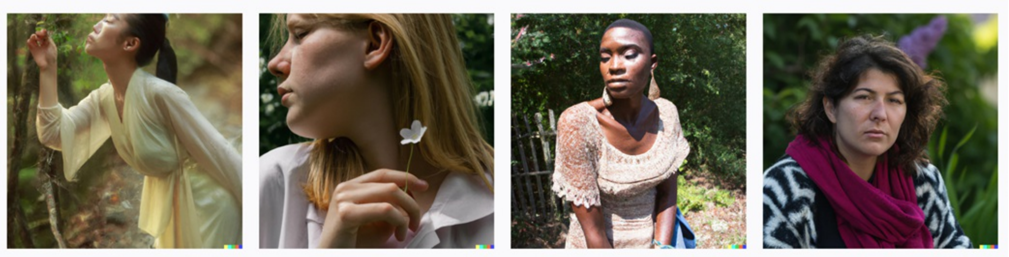


Figure 8.12 shows a total divergent result, if you simply change the keyword calm with *muted*.

Figure 8.12 An image of a muted woman in a garden.



Incorporating this type of image into a data story helps the audience give your main character a face. The main character of your story can communicate directly with the audience through his emotions and expressiveness.

Now that you have learned how to add emotion to images generated by DALL-E, let’s move on to the next step: generating consistent images.

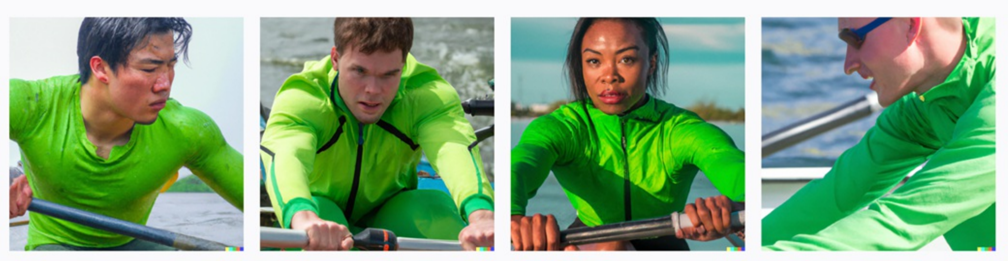
8.5.2 Generating Consistent Images

Consistent images are a set of images sharing common characteristics, such as colors, style, and so on. It is important to build consistent images when you want to add more than one image to your data story. At the time of writing this book, DALL-E does not provide any explicit way to build consistent strategies. However, in this section, you will see a trick proposed by Tyler Taggart in his YouTube video (Taggart 2023) to build consistent images in DALL-E. To do it, we will use the Editor Tool, provided by DALL-E and already introduced in Chapter 4.

To show how the proposed strategy works, we will re-draw the images related to rowing and cycling used in the case study of *Chapter 4, Unlock the Potential: Invest in Rowing and Cycling for Maximum Returns!* The idea is to insert real photos in the data story tailored to executives to capture their attention better.

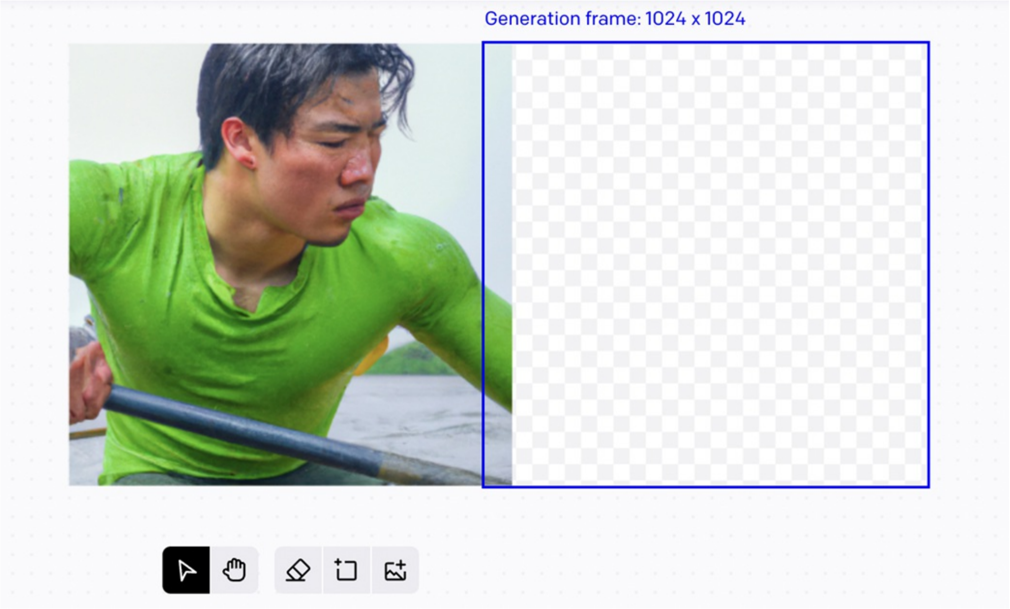
Start by writing the following prompt in DALL-E: A photo showing the effort of a rowing athlete. Show off the sweat and muscles. The athlete wears a green jacket. Figure 8.13 shows a possible output produced by DALL-E.

Figure 8.13 A possible output produced by DALL-E.



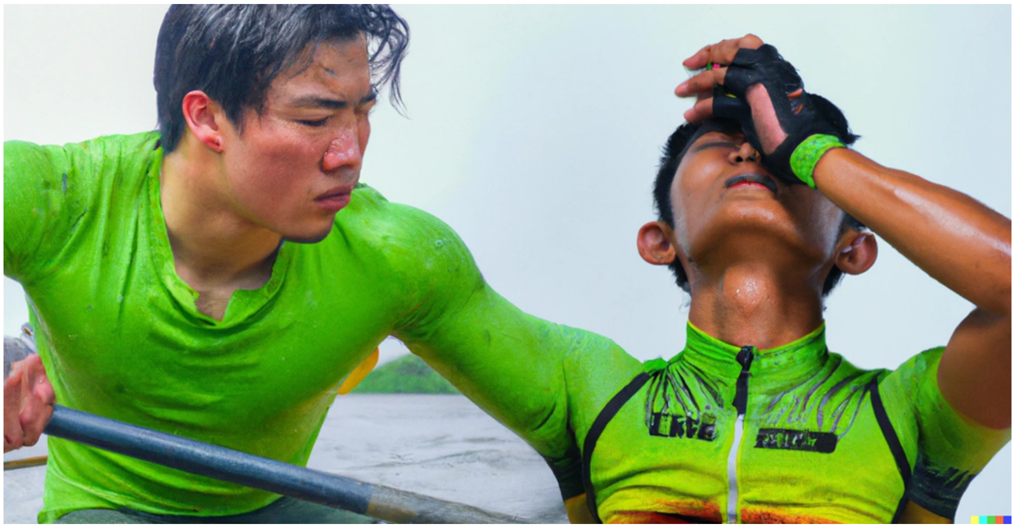
Select a photo among the four available, and click on Edit. Now click on Add a generation frame and insert it near the image, as shown in Figure 8.14.

Figure 8.14 Where to put the generation frame



Now, edit your prompt as follows: *A photo showing the effort of a cycling athlete. Show off the sweat and muscles. The athlete wears a green jacket.* You have simply changed rowing with cycling. DALL-E will generate four images. Select the one that best fits your requirements, or generate other images. Figure 8.15 shows a possible output. The two parts of the photos (left and right) are very similar. For example, they use the same green tonality.

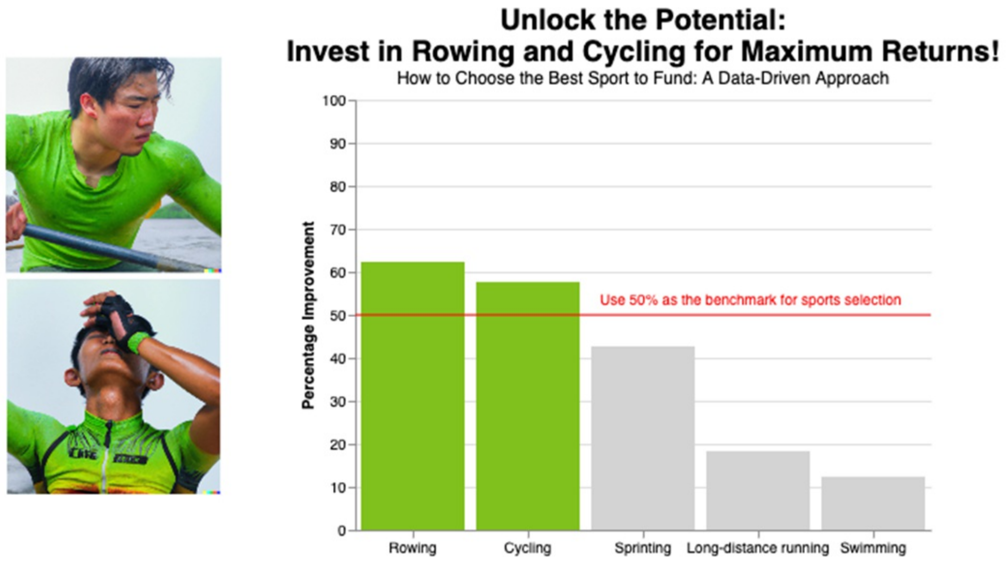
Figure 8.15 The output produced by DALL-E



As you can see, the two photos are linked to each other. The result looks a bit strange, but you can split them (producing two normal-looking photos) by downloading them and using any image tool.

Now that you have obtained two photos of the most performing sports, you can replace them to the chart of the case study in Chapter 4, as shown in Figure 8.16. We have also added to the chart the subtitle produced with fine-tuning. The photos in the story highlight the effort made by the athletes to achieve their records, thus inviting the executives to accept to continue financing them.

Figure 8.16 The final chart tailored to an audience of executives



Now it’s time to practice, so let’s move on to an exercise.

8.5.3 Exercise: Generating Images

Consider the case study in Chapter 1, Increase the advertising campaign on dog-related websites!

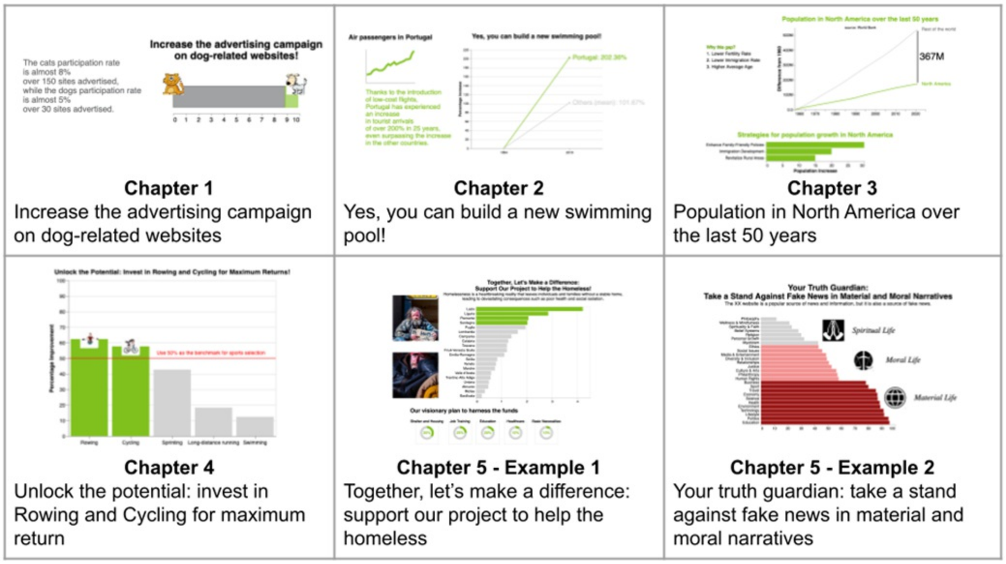
- Generate two consistent images for a dog and a cat and replace the original ones in the chart with the generated images
- Generate a new image for a dog (or a cat) with a specific emotion (e.g. angry, calm, …). Is DALL-E still able to generate emotional images for animals?

We have just completed the journey towards using DALL-E to generate images for data stories. As a last step in this chapter, we will give tips on positioning your chart in your data story.

8.6 Strategic Placement of Context

Context should precede the main point of your data story. For this reason, we suggest you place your context in one of the following three main positions in your data visualization chart: on the top, on the left, and within the chart. In the remainder of this section, we will investigate each context position separately. As an example of using each context position, we will focus on the case studies described in Chapters 1-5. For convenience, Figure 8.17 summarizes the case studies analyzed.

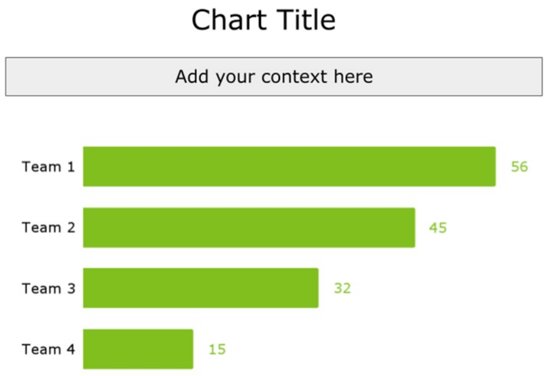
Figure 8.17 A summary of all the examples described in the previous chapters



8.6.1 Top

Adding context at the top of a chart means adding a textual description immediately under the title, as shown in Figure 8.18.

Figure 8.18 The context position at the top of the chart



Use the top part of the chart to describe an overview of the topic or data sources. Don’t use this position to insert images. While images undoubtedly engage audiences, using them in the top part of a chart can potentially distract viewers from focusing on the essential details. Instead, rely on well-crafted words to capture their attention and guide their interpretation.

You can also use the top part of the chart to add the data source. Adding data sources means acknowledging the source of information used in a chart. This gives credibility to the information presented. When the audience sees that we have correctly credited the data source, they are more likely to trust the information presented.

Consider the case studies in Figure 8.17. Table 8.1 shows the case studies using the top context and the type of context used.

Table 8.1 Case studies using the top-context position

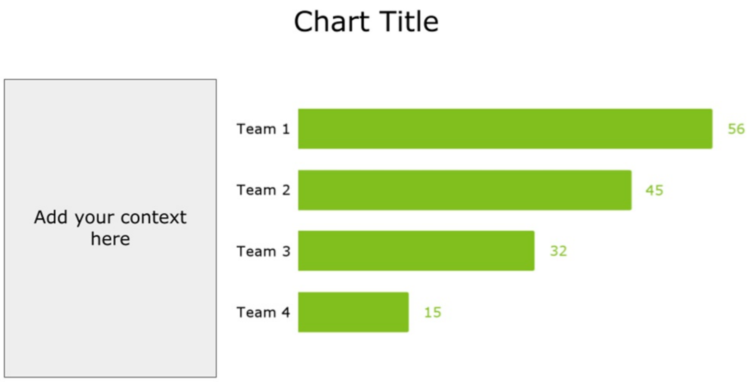
Name	Type of Top Context	Text
Chapter 3 Population in North America over the last 50 years	Data Source	Source: World Bank
Chapter 5 - Example 1 Together, let’s make a	Textual Description	Homelessness is a heartbreaking reality that

difference: support our project to help the homeless		leaves individuals and families without a stable home, leading to devastating consequences such as poor health and social isolation.
Chapter 5 - Example 2 Your truth guardian: take a stand against fake news in material and moral narratives	Textual Description	The XX website is a popular source of news and information, but it is also a source of fake news.

8.6.2 Left

Adding context on the left of a chart involves adding one or more annotations immediately before the title, as shown in Figure 8.19.

Figure 8.19 The context position on the left of the chart



Use the left part of the chart to add deep details such as textual descriptions, images, or photos to catch audiences' emotions or preliminary charts.

Consider again the case studies in Figure 8.17. Table 8.2 shows the case studies using the left context and the type of context used.

Table 8.2 Case studies using the left-context position

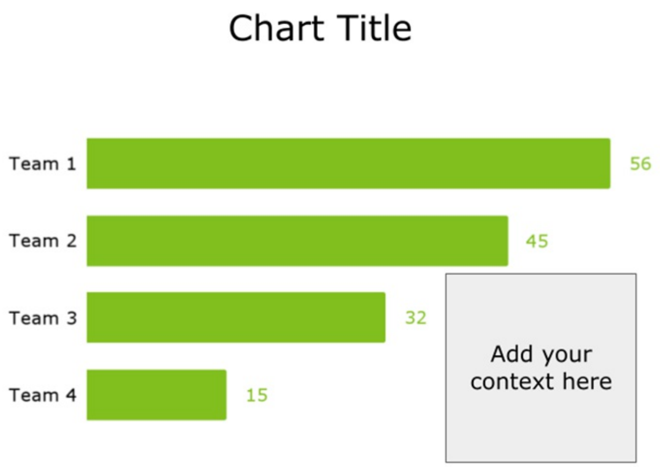
Name	Type of Top Context	Text
Chapter 1 Increase the advertising campaign on dog-related websites	Textual Description	The cats participation rate is almost 8% over 150 sites advertised, while the dogs participation rate is almost 5% over 30 sites advertised.
Chapter 2 Yes, you can build a new swimming pool!	Secondary Chart Textual Description	Thanks to the introduction of low-cost flights, Portugal has experienced an increase in tourist arrivals of over 200% in 25 years, even surpassing the increase in the other countries.
Chapter 3 Population in North America over the last 50 years	Textual Description	Why this gap 1. Lower Fertility Rate 2. Lower Immigration Rate 3. Higher Average Age
Chapter 5 - Example 1 Together, let’s make a difference: support our project to help the homeless	Images	-

Compared to that written in the top position, the text in the left position contains additional details, sometimes extracted from other sources.

8.6.3 Within

Context within a chart highlights some interesting points or parts through chart annotations, as shown in Figure 8.20.

Figure 8.20 The context position within the chart



Annotations help the audience to focus on some specific parts of a chart. Examples of annotations include text, circles, baselines, images, and arrows. When using annotations, make sure they are not excessive and do not overlap the chart itself, preventing its reading.

Consider again the case studies in Figure 8.17. Table 8.3 shows the case studies using the left context and the type of context used.

Table 8.3 Case studies using the within-context position

Name	Type of Top Context	Text
Chapter 1 Increase the advertising campaign on dog-related websites	Image	-
Chapter 3 Population in North America over the last 50 years	Text Vertical Line	367M
Chapter 4 Unlock the potential:	Text Baseline	Use 50% as the benchmark for sports selection

invest in Rowing and Cycling for maximum return	Images	
Chapter 5 - Example 2 Your truth guardian: take a stand against fake news in material and moral narratives	Images Text	Material Life Moral Life Spiritual Life

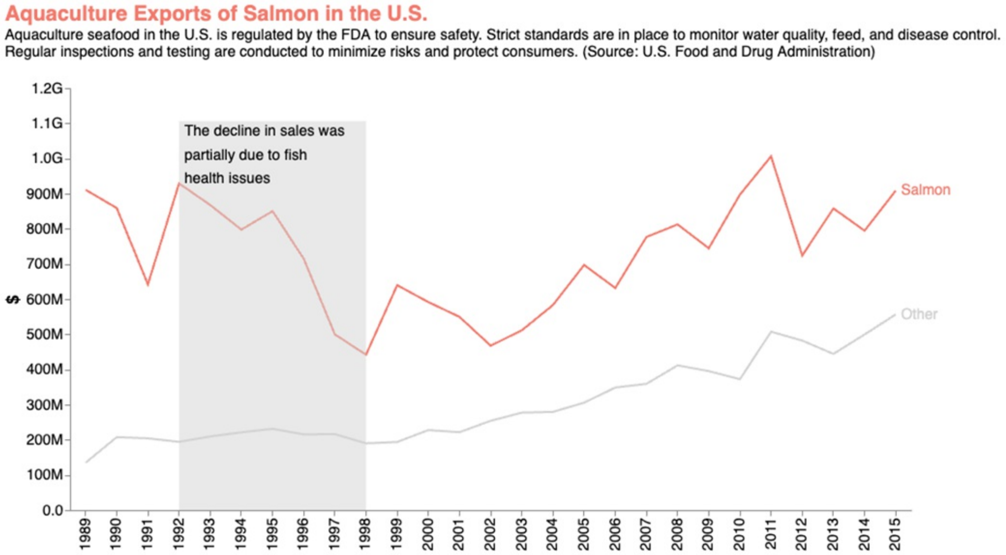
Compared to that written in the top and left positions, the text within the annotation contains concise sentences, highlighting data details.

Now that you have learned how to add a visual context to your data story let’s implement a practical case study.

8.7 Case Study: From Information to Knowledge (part 2)

In the previous chapter, we analyzed how to turn information into knowledge through textual context in the aquaculture case study. Just for a quick reminder, the case study involved building a story around the safety problem in salmon aquaculture in the U.S. We decided to plot the salmon aquaculture sales trendline versus the other types of aquaculture. As an insight, we discovered that since 1998 there has been an increase in sales, following a period of decrease in sales from 1992 to 1998. We discovered that the decreasing period was partially due to some health problems in the salmon aquaculture. Figure 8.21 shows the chart produced when adding textual context to turn information into knowledge.

Figure 8.21 The chart produced when adding textual context



The next step involves adding visual context to the story. We will follow two different scenarios:

- Communicating a negative mood, with a focus on the negative years (1992-1998)
- Communicating a positive mood, with a focus on the years where the

trend is increasing (from 1998).

In both cases, we will generate images, set colors and size related to the specific mood. Let’s start with the first point, communicating a negative mood.

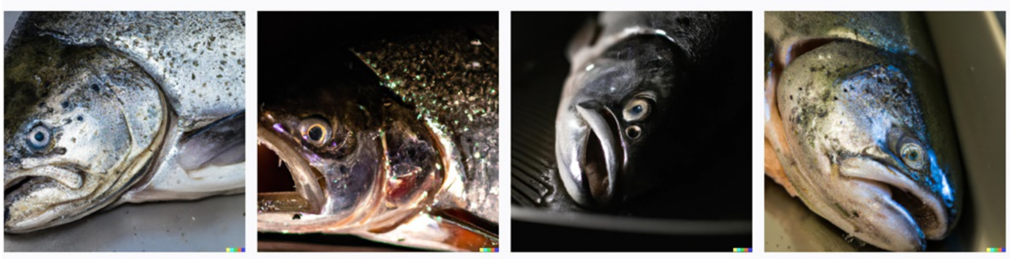
8.7.1 Setting a Negative Mood

In this scenario, the focus is on the negative years. The aim is to show the horror of this period to raise awareness among the audience of the urgency of ensuring that it never happens again. We will generate shocking images and use dark colors.

Images

Generate shocking images (high energy, negative mood) about salmon. Let’s use DALL-E with the following prompt: *a* shocking *photo of a dead salmon for a disease*. Figure 8.22 shows a possible output of images generated by DALL-E.

Figure 8.22 The output produced by DALL-E with the following input: a shocking photo of a dead salmon for a disease



Let’s choose photo n.2 and add it to the chart. Listing 8.7 shows the instructions for Copilot to generate the image in Altair.

Listing 8.7 Instructions for Copilot

```
# build a DataFrame named img_df with the following columns:
#   - 'url' with the values '../images/deadsalmon.png'
#   - 'x' with the value 1993
#   - 'x2' with the value 1997
#   - 'y' with the value 0  #A

# draw a image with the img_df DataFrame:
# - set x to 'x'
# - set y to 'y'
# - set x2 to 'x2'
```

How to instruct Copilot to generate images

First, instruct Copilot to generate a DataFrame containing all the relevant channels. Next, specify how to draw the image using Altair. We have asked Copilot to set y = 0. This is an initial value. You can adapt it manually by looking at the image.

You can find the complete code in the GitHub repository of the book, under CaseStudies/from-information-to-knowledge-visual/chart-negative.py.

Color and size

In addition to the photo, we can use dark colors to emphasize the negative scenario. For example, we can use a dark red tonality (#460805), which recalls the blood. Next, we can increase the stroke width in correspondence of years 1992-1998. We generate another line for those years, overlapping with the original chart. Listing 8.8 shows the code to generate the line.

Listing 8.8 Adding a focused line

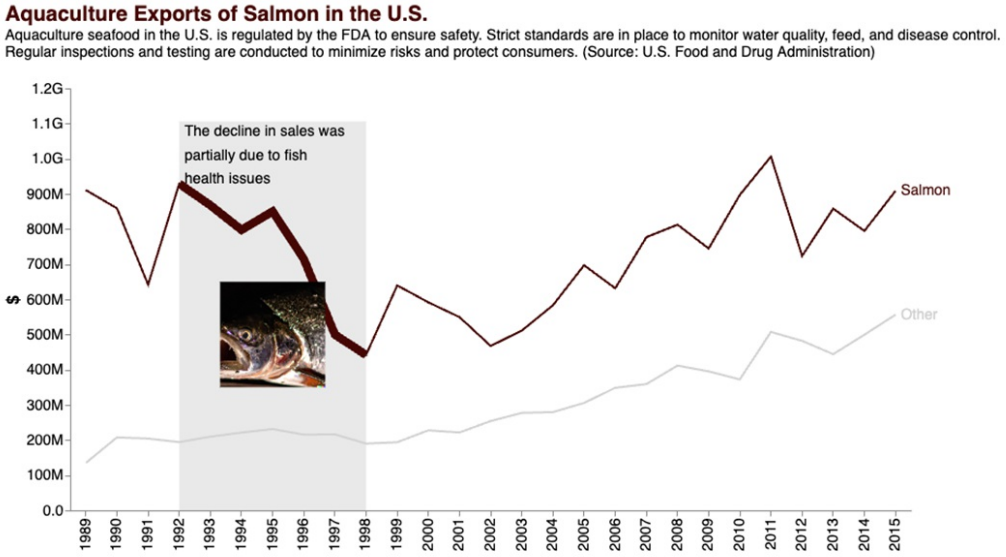
```
base = alt.Chart(df
).encode(...)

chart = base.mark_line()

chart_line = base.mark_line(
    strokeWidth=8
).transform_filter(
    (alt.datum.CATEGORY == 'Salmon') &
    (alt.datum.YEAR_ID >= 1992) &
    (alt.datum.YEAR_ID <= 1998)
)
```

How to add a focused line to a chart

First, build a base chart with all the encoding options. After drawing the main chart, draw another chart with focusing on a selection. Use the `transform_filter()` method to select only some data.

Again, you can find the complete code in the GitHub repository of the book under CaseStudies/from-information-to-knowledge-visual/chart-negative.py. Figure 8.23 shows the resulting chart.

Figure 8.23 The chart after adding a negative visual context



Now that you have implemented the negative-mood scenario, let's move to the opposite one, the positive-mood.

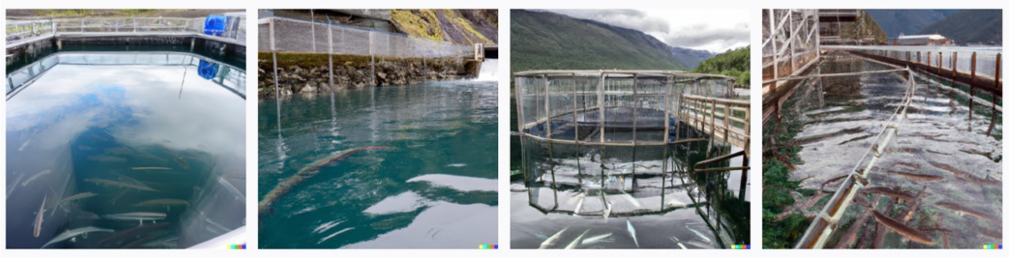
8.7.2 Setting a Positive Mood

In this scenario the focus is on the positive trendline, started since 1998. The objective is to encourage the audience to continue to adopt the current strategy for the sake of the salmon population. Instead of focusing on the negative effects, here we can focus on the positive scenario, mentioning the negative aspects only as a possibility. We will use relaxing photos and colors.

Images

The procedure to generate the relaxing images is similar to the negative scenario, with the difference of using words falling in the top-left clock face of Figure 8.1 (low energy, positive mood). Use the following prompt for DALL-E to generate the required images: *a* calm *aquaculture of safe salmon*. Figure 8.24 shows a possible output. Let's choose photo n.1.

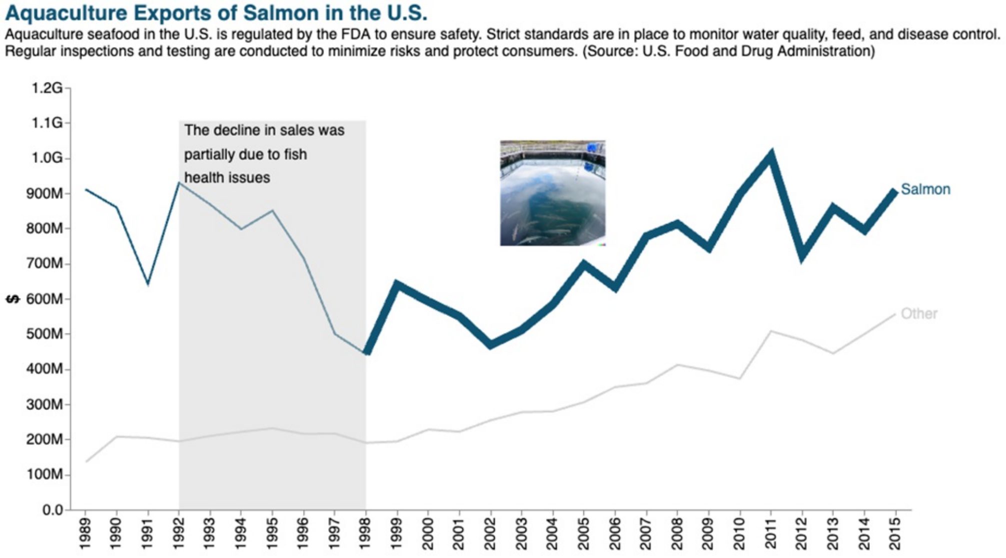
Figure 8.24 The output produced by DALL-E with the following input: a calm aquaculture of safe salmons



Color and size

Use a peaceful color (e.g., #105473) to represent the salmon line. In addition, highlight the trendline since 1998. Figure 8.25 shows the resulting chart. You can find the complete code in the GitHub repository of the book, under [CaseStudies/from-information-to-knowledge-visual/chart-positive.py]().

Figure 8.25 The chart after adding a positive visual context



Aquaculture Exports of Salmon in the U.S.

Aquaculture seafood in the U.S. is regulated by the FDA to ensure safety. Strict standards are in place to monitor water quality, feed, and disease control. Regular inspections and testing are conducted to minimize risks and protect consumers. (Source: U.S. Food and Drug Administration)

The decline in sales was partially due to fish health issues

We have generated two types of chart, the negative or the positive, but not both together (say, with both images in the same graph). This is because we don't want conflicting emotional cues, potentially confusing the audience or diluting the intended message we aim to convey.

Now that you have learned how to add a positive or a negative mood to a data story, let’s move on to an additional exercise, adding interactivity.

8.7.3 Exercise

Modify the previous chart to make it interactive as follows: define a dropdown menu with two values: positive and negative. When the user selects negative, the chart highlights the salmon trendline between 1992 and 1998. When the user selects positive, the charts highlights the trendline since 1998. You can find the solution to the exercise in the GitHub repository of the book, under CaseStudies/from-information-to-knowledge-visual/chart-dropdown.py.

In the first part of this chapter, you have learned how to turn information into knowledge by adding visual context to your data visualization chart. You have seen how to add emotions to your charts in terms of colors, size, and interactivity. You have also learned how to generate mood-based and consistent images in DALL-E. Next, you have learned where to put the context in your chart. Finally, you have implemented a practical case study.

In the next chapter, you will see how to turn knowledge into wisdom, by adding a call to action and possible next steps to your chart.

8.8 Summary

- Adding visual context to your data visualization is crucial for rising audience emotions.
- Incorporating colors, size and interactivity helps you to better focus your story and tailor it to your audience.
- When you draw your images in DALL-E, keep them consistent using the editing tool.
- Consider position the context strategically within your chart to enhance

its effectiveness.

8.9 References

Using Emotions for Communication

- Decker Ben and Decker Kelly (2015). Communicate to Influence: How to Inspire Your Audience to Action. McGraw Hill. ISBN: 978-0071839839 Colors and Size
- Abbott, Desireé (2023). *Everyday Data Visualization*. Manning Publications ISBN: 978-1633438408
- Christiansen, Jen (2022) Building Science Graphics: An Illustrated Guide to Communicating Science through Diagrams and Visualizations. A K Peters/CRC Press. ISBN: 978-1032109404
- Strachnyi, Kate (2022) ColorWise: A Data Storyteller's Guide to the Intentional Use of Color. O’Reilly Media. ISBN: 978-1492097846

8.9.1 Images

- Chip *Photos and illustrations in a visual narrative* https://www.quanthub.com/photos-and-illustrations-in-a-visual-narrative/ (Last Access 2023/09/21)
- The DALLE 2 Prompt Book https://dallery.gallery/wp-content/uploads/2022/07/The-DALL%C2%B7E-2-prompt-book-v1.02.pdf (Last Access 2023/10/20)
- Gemignani, Z. (2021) *Tips for Using Photos in Data Storytelling* https://www.juiceanalytics.com/writing/tips-for-using-photos-in-data-stories (Last Access 2023/09/21)
- Hang, I. (2023) *How to Create Consistent Characters in Midjourney - EASY Step by Step AI Tutorial for Amazon KDP* https://www.youtube.com/watch?v=zdSIPkbvsek (Last Access 2023/09/21)
- Nussbaumer Knaflic, C. *using images* https://www.storytellingwithdata.com/blog/2018/3/26/using-images (Last Access 2023/09/21)
- Taggart, T. (2023) *Dalle 2 Tutorial: How To Get Image Consistency* https://www.youtube.com/watch?v=MU_yXYsfBR0 (Last Access

2023/09/21)

[1] https://dallery.gallery/wp-content/uploads/2022/07/The-DALL%C2%B7E-2-prompt-book-v1.02.pdf

9 From Knowledge to Wisdom: Adding Next Steps

This chapter covers

- Introducing wisdom, next steps and calls to action
- Types of calls to action
- Types of next steps
- Anchoring next steps to an ethical framework
- Strategic placement of next steps

In this chapter, we will review the basic concepts behind wisdom (in the context of data storytelling, at the top of the DIKW pyramid), next steps, and calls to action in a data story and how to adapt it based on the story purpose. We will also see how to adapt next steps to an ethical framework and place them strategically in the story.

9.1 The Chart is Ready. What Now?

If you are reading this chapter, you have finally climbed almost to the top of the DIKW pyramid and have built a fantastic chart. You are delighted with your work and have probably already shown it to colleagues and perhaps even friends and relatives. It pains me, however, to tell you that your work is incomplete. The last piece is missing, the one that will transform your graph into a real story.

To better understand the problem, let's take an example. Imagine you have started reading a beautiful novel - my favorite is *Quo Vadis?* by Henryk Sienkiewicz - in which the protagonist faces many challenges and encounters many adversities or problems. While you are reading the protagonist's adventures, suddenly, the novel ends without knowing how or whether or not the protagonist has overcome the challenges or solved the problems. Well, if you limit your chart only to the knowledge phase, it's like a novel without an end. You shouldn’t just present your data findings; you should tell the viewer how you think they should *act* upon those findings. There's a well-known saying that 'knowledge is not power; it is potential power when used for action. Here then, in your chart, you must add what we’d call *next steps.* These are implemented as a call to action that invites your audience to do something.

The next steps are not something dropped from the sky or, even worse, something theoretical but is the results of your studies and analysis. It proposes to your audience what to do after seeing your chart, for example, regarding possible alternatives or concrete steps to move forward.

The next steps can never be just the fruit of the knowledge acquired through data. It must be anchored to something broader, commonly called wisdom. Wisdom is a cultural treasure trove derived from experience (from the data analyzed) and knowledge shared with others.

Based on the context in which you live, the type of audience, and your values, the next steps must be anchored to an ethical framework. You cannot think of proposing a generic next step based only on your own beliefs or experience. It is necessary to anchor the next steps to the values of the society in which you live, if they are to have broad application. These values vary from community to community, but fundamental principles must be respected, and the next steps must be anchored to them.

The next step is often used as a synonym for a call to action. However, in this book we consider the next steps and calls to action as different concepts. The next steps indicate **what** to do after reading a data story. Calls to action instead define **how** to do it practically. A call to action is a possible implementation of the next steps. Imagine you want your audience to collaborate with your story after reading it (next steps). You can implement this collaborative request using different calls to action, such as sharing the story on social media or contacting someone for more details.

In the remainder of this section, we will introduce the concepts of wisdom, next steps, and call to action. Let’s start with the concept of wisdom.

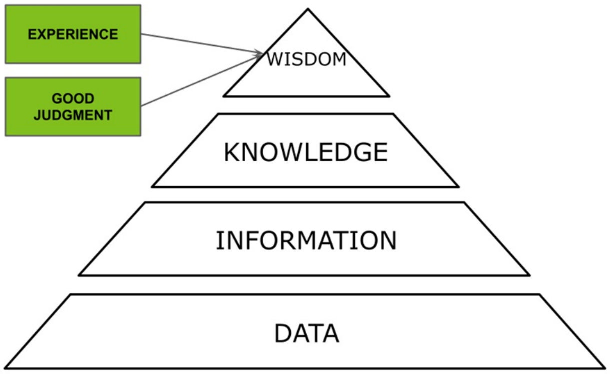
9.1.1 What Wisdom Is

The Collins dictionary defines wisdom as *the ability to use your experience and knowledge in order to make sensible decisions or judgments*[1]. Applied to data storytelling, wisdom enables us to make better decisions based on our data. Wisdom is not simply the result of transforming knowledge. It also requires other aspects that depend on our baggage in life. Wisdom is the result of of:

- **Knowledge** provides the foundation upon which wisdom is built. We cannot make informed decisions without a broad understanding of our data.
- **Experience** provides the context through which our knowledge is applied. It deepens our understanding of the world and helps us to see the nuances and complexities that can't be learned from books or lectures. Experience encompasses all the moments we've lived through, the challenges we've faced, and the lessons we've learned.
- **Good judgment** is the ability to weigh the pros and cons of a situation, analyze the risks and benefits, and come to a well-informed and wise decision. It requires the ability to evaluate a situation from multiple perspectives and understand the potential outcomes of each choice.

Figure 9.1 shows how the DIKW pyramid should be modified to include wisdom properly.

Figure 9.1 The DIKW pyramid enriched with the components of wisdom



9.1.2 Transforming Knowledge into Wisdom: Next Steps

Every self-respecting story carries a message, something that remains in the hearts of the audience after reading the story. And this is the addition of wisdom to the story. Wisdom allows the audience to be engaged by the story, to be touched, and to be ready to do something after reading the story. However, the message alone is not enough. The audience must do something after reading the story.

Transforming knowledge into wisdom involves adding the next steps to the data story. Next steps are the resolution of our data story and defining what the audience should do after reading the story.

Every story ends with an ending, which can be good or bad, but there must be one. Even in the case of data storytelling, the addition of an ending to the story is envisaged. Unlike a novel, where the ending concludes the story, data storytelling should be an open story, in the sense that it should not end with the story itself but should expect the audience to do something after reading the story.

Next steps are implemented through practical actions or strategies suggested to address a problem, leverage an opportunity, or make improvements based on the insights. It outlines the actionable path forward, providing a clear plan or actions to follow.

9.1.3 The Call to Action: A Way to Implement Next Steps

A Call to Action (CTA) is a practical implementation of a next-step strategy. The same next step can be implemented through different calls to action. The Cambridge Dictionary defines the CTA as *something such as a speech, piece of writing, or act that asks or encourages people to take action about a problem*[2]. The CTA term is used especially in the marketing sector, for example, to invite the audience to buy a product. On websites, the CTA is implemented through buttons, links, forms, and much more.

While the next steps specify what we want our audience to do after reading our data story, the CTA specifies how to implement the next steps practically.

Suppose we have a data story about climate change, and we want our audience to collaborate with environmental protection organizations after reading the story (next steps). We can implement this strategy in different ways in our story, using different calls to action, for example, through an in-depth link, an email to write to, or a series of steps to follow to get involved with the organization. All three calls to action implement the same next step in different ways.

Now that you have learned the basic concepts of wisdom, next steps, and the call to action, let’s move on to how to use them in data storytelling. We will start with the classification of calls to action and then move on to the description of the next steps because in the next steps section, we will implement different scenarios using different types of calls to action.

9.2 Typesof Calls to Action

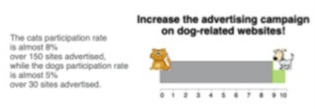
We classify the calls to action into the following popular categories:

- Learning
- Sharing
- Acting
- Visualizing roadmaps
- Engaging and interacting
- Supporting and assisting

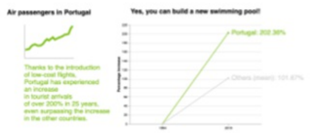
This classification includes the most popular categories. However, you could find other less popular categories of calls to action, for example by searching the key word *call to action types* on your preferred web search engine

Let’s investigate each of them separately. For each of them, we will give a short description, the purpose, and an example. In the next section, we will apply the different calls to action to case studies we have implemented throughout the book. Just for a quick reminder, Figure 9.2 summarizes the case studies implemented in the previous chapters.

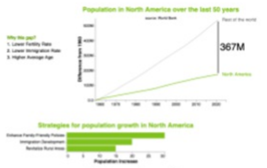
Figure 9.2 A summary of all the examples described in the previous chapters



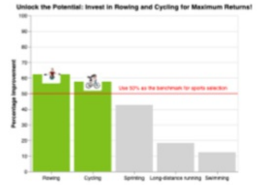
Chapter 1
Increase the advertising campaign on dog-related websites



Chapter 2
Yes, you can build a new swimming pool!



Chapter 3
Population in North America over the last 50 years



Chapter 4
Unlock the potential: invest in Rowing and Cycling for maximum return

Together, Let's Make a Difference:
Support Our Project to Help the Homeless!
Our visionary plan to harness the funds

Chapter 5 - Example 1
Together, let's make a difference: support our project to help the homeless

Your Truth Guardian:
Take a Stand Against Fake News in Material and Moral Narratives
Spiritual Life
Moral Life
Material Life

Chapter 5 - Example 2
Your truth guardian: take a stand against fake news in material and moral narratives

Let's look at each of the categories briefly in turn in the following subsections.

9.2.1 Learning

Description: This call to action encourages the audience to delve deeper into the topic or insights presented in the data story.

Purpose: Direct the audience to additional resources, articles, studies, or references for a more comprehensive understanding of the topic.

Example: A link to an in-depth analysis report.

9.2.2 Sharing

Description: Encourage the audience to share the data story with their social networks.

Purpose: Amplify the reach of the data story by leveraging the audience's networks, fostering discussions, and increasing visibility.

Example: Use social media buttons to share the story.

9.2.3 Acting

Description: Ask the audience to take specific, practical actions based on the insights revealed in the data.

Purpose: Guide users toward implementing strategies or changes to address identified issues or capitalize on opportunities.

Example: Implement the XX strategy to improve customer satisfaction.

9.2.4 Visualizing Roadmaps:

Description: Show a flowchart, timeline, or roadmap outlining the sequence of actions to be taken.

Purpose: Provide a clear, graphical layout of the steps, offering a visual guide for implementing the recommended actions.

Example: An interactive roadmap to see a step-by-step plan for optimizing your marketing strategy.

9.2.5 Engaging and Interacting

Description: Encourage the audience to actively engage with interactive elements or simulations related to the data story.

Purpose: Foster audience participation, allowing them to experiment with different scenarios or elements discussed in the data story.

Example: Try our interactive tool to explore the impact of various strategies on sales growth.

9.2.6 Supporting and Assisting

Description: Offer avenues for the audience to seek help or guidance in implementing the suggested actions.

Purpose: Provide support channels, expert consultation, FAQs, or troubleshooting guides to aid audiences in executing the recommended steps.

Example: The email of the supporting team or a link to the FAQ page.

Now that you have learned the calls to action types let’s move on to the classification of next steps. We will use the learned concepts about calls to action to implement different examples.

Challenge

Can you think about other categories of calls to action? If yes, please share them in the book forum.

You have just learned the basic calls to action. Now it’s time to see how we can use them as a practical implementation of next steps.

9.3 Types of Next Steps

We can classify the next steps into different categories based on the specific objectives they aim to achieve:

- Awareness and campaigns
- Collaborative efforts
- Advocacy and policy
- Training and development
- Strategic actions
- Engagement and communication.

In the remainder of this section, we will describe each category separately. Let’s start with the first one, awareness and campaigns.

9.3.1 Awareness and Campaigns

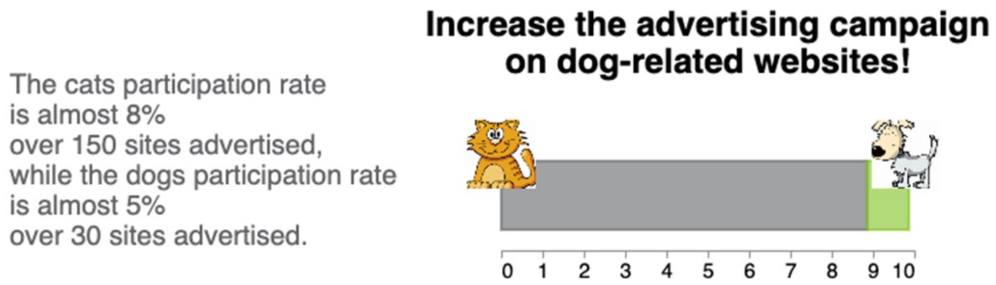
This category of next steps includes:

- **Public Awareness Campaigns**: Launching campaigns to raise awareness among the audiences regarding insights highlighted by the data.
- **Behavioral Change Initiatives**: Encouraging changes in behavior or

habits based on the insights derived from data analysis.
- **Social Impact Initiatives**: Creating initiatives that drive positive social change in response to data findings.

Consider the case study we analyzed in Chapter 1, dogs/cat campaign. Just for a quick reminder, the case study focused on an event dedicated to pets (dogs and cats). For each pet category, the organizers strategically advertised the event on specialized websites dedicated to that particular pet category. As a result, 9 out of 10 pets participating in the event were cats. The next steps included improving the campaign related to the dog websites to increase the dog participants. Figure 9.3 shows the final data story related to this case study.

Figure 9.3 The final data story related to the dog-cat case study

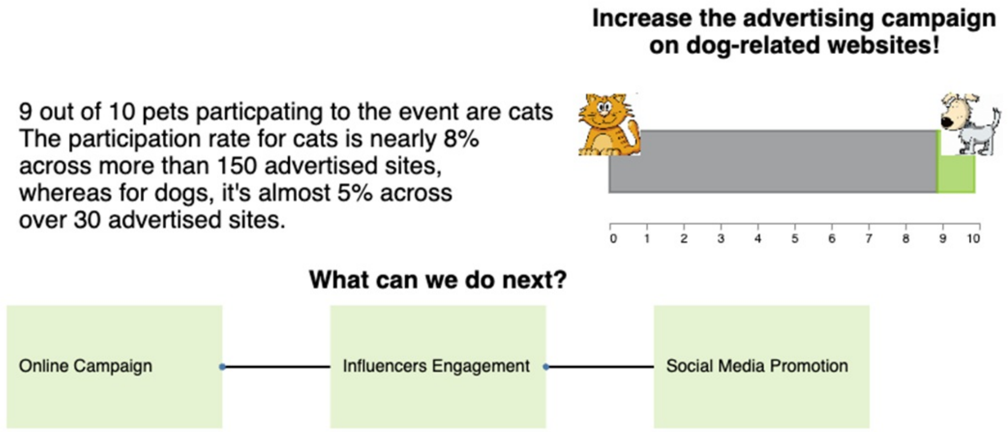


The case study described in Chapter 1 contained a generic next step, which invited the audience to increase the dog-related website campaign. Some possible improvements to the next steps may include more specific next steps, such as:

- Defining tailored online campaigns
- Engaging influencers and communities
- Promoting the event on social media.

Let’s implement the previous next steps using a visualized roadmap as a call to action. We will implement the roadmap as three rectangles connected through a line, as shown in Figure 9.4.

Figure 9.4 The dog-cat case study enriched with a visualized roadmap



The implemented code is available in the GitHub repository of the book, under 09/awareness-and-campaigns/pets-visualized-roadmap.py. We will focus only on the next step part (in the bottom part of the chart). Start by defining the rectangle size. We use parametric values to make the code as general as possible. Listing 9.1 shows the code to generate the three rectangles at positions 0, 15, and 30.

Listing 9.1 Setting the rectangles positions

```
import pandas as pd

width = 10
space = 5
N = 3

x = [i*(width+space) for i in range(N)]
y = [0 for i in range(N)]
x2 = [(i+1)*width+i*space for i in range(N)]
y2 = [10 for i in range(N)]
text = ['Online Campaign', 'Influencers Engagement', 'Social Medi

df_rect = pd.DataFrame(
    {   'x': x,
        'y': y,
        'x2': x2,
        'y2': y2,
        'text' : text
    }
)
```

How to set the rectangles positions

We suppose that we want to generate `N` (3 in the example) rectangles and each rectangle is large `width` (10) and the space between two rectangles is `space` (5). Then, we generate the rectangle coordinates using `x`, `x2`, `y`, and `y2` as vertices. We also include the label text for each rectangle. Finally, we create a DataFrame with the defined variables.

Next, draw the rectangles in Altair, as shown in Listing 9.2.

Listing 9.2 Drawing the rectangles

```
import altair as alt

rect = alt.Chart(df_rect).mark_rect(
    color='#80C11E',
    opacity=0.2
).encode(
    x=alt.X('x:Q', axis=None),
    y=alt.Y('y:Q', axis=None),
    x2='x2:Q',
    y2='y2:Q'
).properties(
    width=700,
    height=100,
    title=alt.TitleParams(
        text=['What can we do next?'],
        fontSize=20,
        offset=10
    )
)
```

Draw the rectangles

Use the `mark_rect()` mark to draw rectangles. Use the `x`, `y`, `x2`, and `y2` channels and remove the axes (`axis=None`). Finally, set the chart size and the title. Use `alt.TitleParams()` to specify the title font size (`fontSize`) and the title offset from the chart.

Now we are ready to draw the labels in the rectangles, as shown in Listing 9.3.

Listing 9.3 Drawing the labels

```
text = alt.Chart(df_rect).mark_text(
    fontSize=14,
    align='left',
    dx=10,
).encode(
    text='text:N',
    x=alt.X('x:Q', axis=None),
    y=alt.Y('y_half:Q', axis=None),
).transform_calculate(
    y_half='datum.y2/2'
)
```

Draw the labels

Use the `mark_text()` mark to draw the labels. Use the `transform_calculate()` method to set the y coordinate for each label.

The following step involves defining the line coordinates. We must draw N-1 lines, the first starting at position 10 and ending at position 15, and the second starting at position 25 and ending at position 30. Use a formula to calculate these positions, as shown in Listing 9.4.

Listing 9.4 Defining the lines coordinates

```
x = [width*i+space*(i-1) for i in range(1,N)]
y = [5 for i in range(N-1)]
y2 = [5 for i in range(N-1)]
x2 = [(width+space)*i for i in range(1,N)]

df_line = pd.DataFrame(
    {   'x': x,
        'y': y,
        'x2': x2,
        'y2': y2
    }
)
```

Define the line coordinates

Use a formula to calculate the x coordinates and set the y coordinates to a constant value.

Draw the lines, as shown in Listing 9.5.

Listing 9.5 Drawing the lines

```
line = alt.Chart(df_line).mark_line(
    point=True,
    strokeWidth=2
).encode(
    x=alt.X('x:Q', axis=None),
    y=alt.Y('y:Q', axis=None),
    x2='x2:Q',
    y2='y2:Q'
)
```

Draw the lines coordinates

Use the `mark_line()` mark to draw the line. Set `point=True` to show points.

If we have drawn the previous elements of the dog-cat chart, we can build the final chart, as shown in Listing 9.6.

Listing 9.6 Building the final chart

```
chart = ((context | (chart + annotation + img)) & (rect + line +
).configure_view(
    strokeWidth=0
).resolve_scale(
    color='independent',
    x='independent',
    y='independent'
).configure_axis(
    grid=False
)

chart.save('pets-visualized-roadmap.html')
```

Combine all the charts

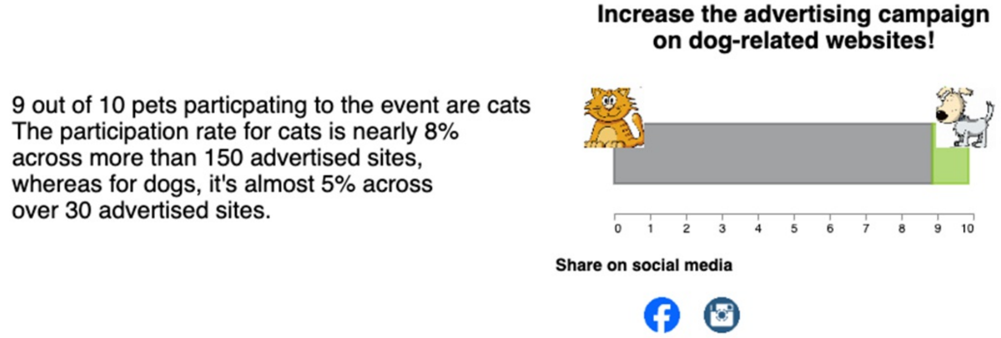
The final chart includes the `context` (on the left), the main bar chart (`chart`), the `annotation` (texts dog and cat) and the images (`img`), on the right, and the next steps (`rect`, `line` and `text`) below. Set color, x, and y to independent in the `resolve_scale()` method to render the chart scales independently.

Now that we have completed this example, try out an exercise on the same topic.

Exercise 1

Consider again the dog-cats case study and the *awareness and campaigns* next step. Implement this strategy using the *social sharing* call to action. Use the `href` channel to include the link to an external website. Figure 9.5 shows a possible result.

Figure 9.5 The dog-cat case study enriched with a social sharing call to action



You can find the solution to this exercise in the GitHub repository of the book, under 09/awareness-and-campaigns/pets-social-sharing.py.

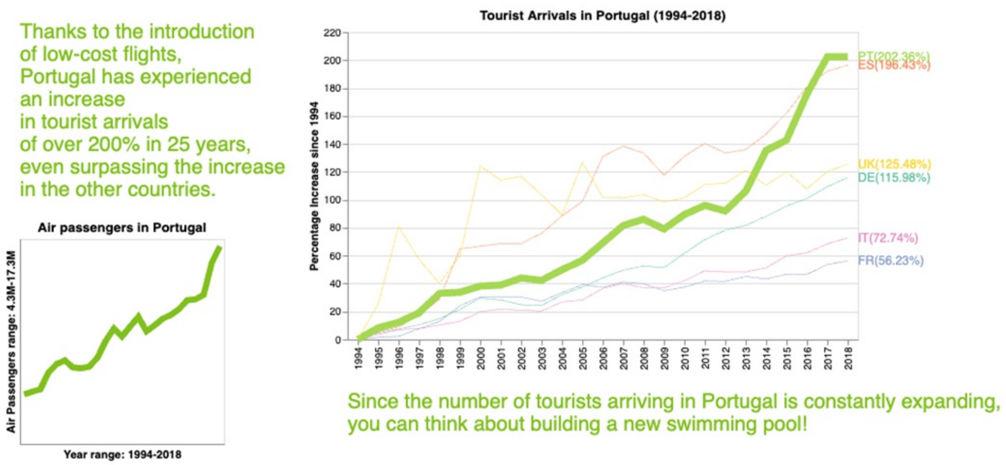
Now that you have learned how to implement the awareness and campaigns next steps, let’s move on to the next category, collaborative efforts.

9.3.2 Collaborative Efforts

This category of next steps includes inviting the audience to collaborate with other organizations and stakeholders or building a team to solve a problem highlighted by the data story.

Consider the case study we analyzed in Chapter 2, Swimming Pool. As a quick reminder, the case study focused on the possibility of building a new swimming pool in a Portuguese hotel. The data story showed an increasing number of tourists in Portugal in recent years, as shown in Figure 9.6.

Figure 9.6 The final data story related to the swimming pool case study



Thanks to the introduction of low-cost flights, Portugal has experienced an increase in tourist arrivals of over 200% in 25 years, even surpassing the increase in the other countries.

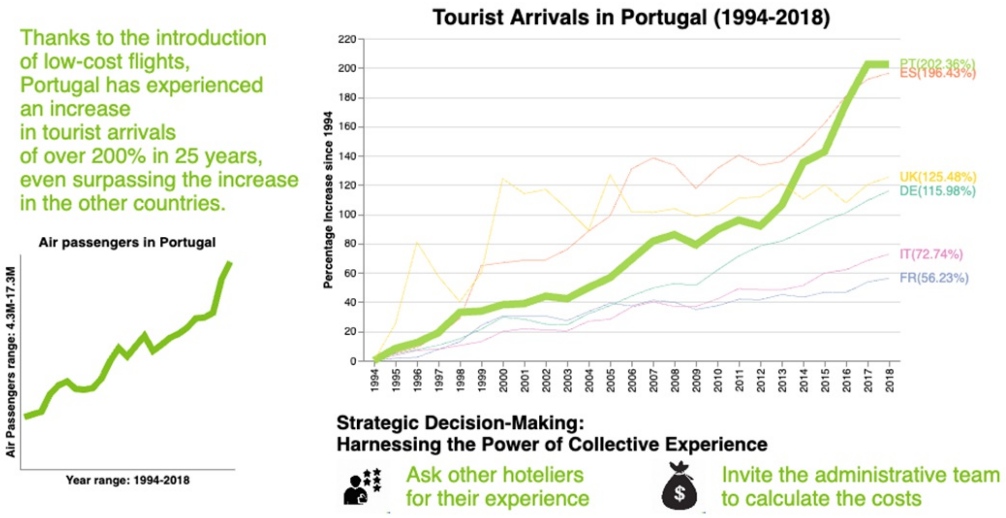
Since the number of tourists arriving in Portugal is constantly expanding, you can think about building a new swimming pool!

The case study described in Chapter 2 contained a generic next step, which invited the audience to build a swimming pool. Some possible improvements to the next steps may include more specific next steps, such as:

- Ask other hoteliers for their experience
- Invite the administrative team to calculate the costs

Let’s implement the previous next steps using actionable recommendations as a call to action, as shown in Figure 9.7.

Figure 9.7 The dog-cat case study enriched with a social sharing call to action



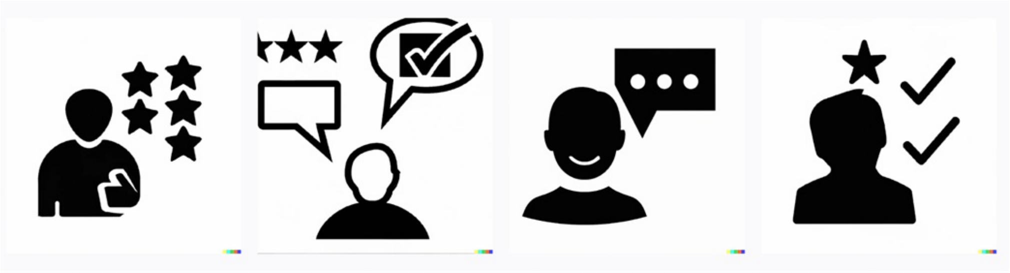
Thanks to the introduction of low-cost flights, Portugal has experienced an increase in tourist arrivals of over 200% in 25 years, even surpassing the increase in the other countries.

Strategic Decision-Making: Harnessing the Power of Collective Experience

Ask other hoteliers for their experience

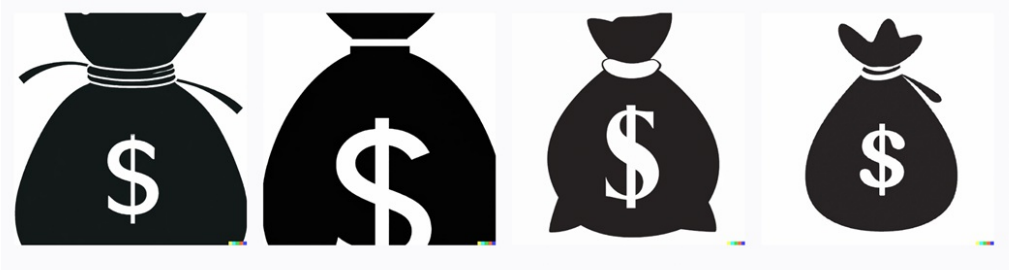
Invite the administrative team to calculate the costs

Figure 9.8 A possible output produced by DALL-E using the following prompt: customer experience, black and white icon



For the second, use the following prompt: *dollar bag, black and white icon*. Figure 9.9 shows a possible result produced by DALL-E. Choose one image and download it.

Figure 9.9 A possible output produced by DALL-E using the following prompt: dollar bag, black and white icon



Now we are ready to write the code to generate the next step section. You can find the code in the GitHub repository of the book under 09/collaborative-efforts/actionable-recommendations.py. Let’s focus on the call to action part of the code. Start by creating the DataFrame containing the texts and the images to include in the CTA, as shown in Listing 9.7.

Listing 9.7 The CTA DataFrame

```
df_cta = pd.DataFrame(
    {'text' : [['Ask other hoteliers', 'for their experience'], [
    'x' : [0,9.8],
    'img' : ['img/experience.png', 'img/costs.png']})
```

Creating the CTA DataFrame

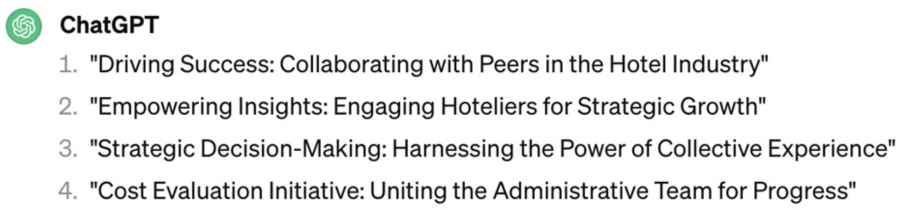
Create a new DataFrame containing the CTA texts, their position on the x-axis, and the path to the images.

Next, let’s use ChatGPT to generate the title for the CTA. Ask ChatGPT for the following prompt:

Act as an invitator to action for executives. Generate four titles from the following text: Next steps: Ask other hoteliers for their experience. Invite the administrative team to calculate the costs.

Figure 9.10 shows a possible output produced by ChatGPT.

Figure 9.10 A possible output produced by ChatGPT to generate four possible titles for the CTA



ChatGPT

1. "Driving Success: Collaborating with Peers in the Hotel Industry"
2. "Empowering Insights: Engaging Hoteliers for Strategic Growth"
3. "Strategic Decision-Making: Harnessing the Power of Collective Experience"
4. "Cost Evaluation Initiative: Uniting the Administrative Team for Progress"

Let’s choose the third title, *Strategic Decision-Making: Harnessing the Power of Collective Experience*, as the title for our next step part of the story. Visually speaking, we implement the next steps as the combination of two charts, a text and an image chart. Since the two charts share some common parts, let’s draw a base chart first, as shown in Listing 9.8.

Listing 9.8 Building base chart for the next steps

```
base_cta = alt.Chart(df_cta
).encode(
    x=alt.X('x:Q', axis=None)
).properties(
    title=alt.TitleParams(
        text=['Strategic Decision-Making:', 'Harnessing the Power
        fontSize=18,
        anchor='start'
    )
)
```

How to build a base chart

Build a basic structure of the chart, without specifying any mark. Only specify the input DataFrame (`df_cta`), the shared encode channels (`X`), and properties (`title`). Use `TitleParams()` to specify the title details.

Once we have defined the basic chart, we can define each chart detail separately. Listing 9.9 describes how to implement the text.

Listing 9.9 Drawing the text chart

```
text_cta = base_cta.mark_text(
    lineBreak='\n',
    align='left',
    fontSize=20,
    y=0,
    dx = 40,
    color='#81c01e'
).encode(
    text='text:N'
)
```

How to draw the text

The text chart details the `base_cta chart`, by adding the text mark property (`mark_text`) and the specific encode for the text (the `text` channel). Within the `mark_text()` property, specify some text properties, such as the lineBreak character (`lineBreak`), how to align the text (`align`), the font size (`fontSize`), the y position (`y`), the x-axis shift from the x position (`dx`), and the text color (`color`).

Next, draw the images, as shown in Listing 9.10.

Listing 9.10 Drawing the image chart

```
img_cta = base_cta.mark_image(
    width=50,
    height=50,
    y=10
).encode(
    url='img'
)
```

How to draw the images

The image chart details the `base_cta chart`, by adding the image mark property (`mark_image`) and the specific encode for the images (the `url` channel). Within the `mark_image()` property, specify some image properties, such as the image width (`width`) and height (`height`), and the y position (`y`).

Now we can build the final chart. We assume that we have already implemented the previous parts of the data story (chart, annotation, commentary and airport chart). Listing 9.11 shows how to layer the charts to render the final data story.

Listing 9.11 Building the final data story

```
chart = ((commentary & airports) | ((chart + annotation) & img_ct
).resolve_scale(
    x='independent',
).configure_view(
    strokeWidth=0
)

chart.save('actionable-recommendations.html')
```

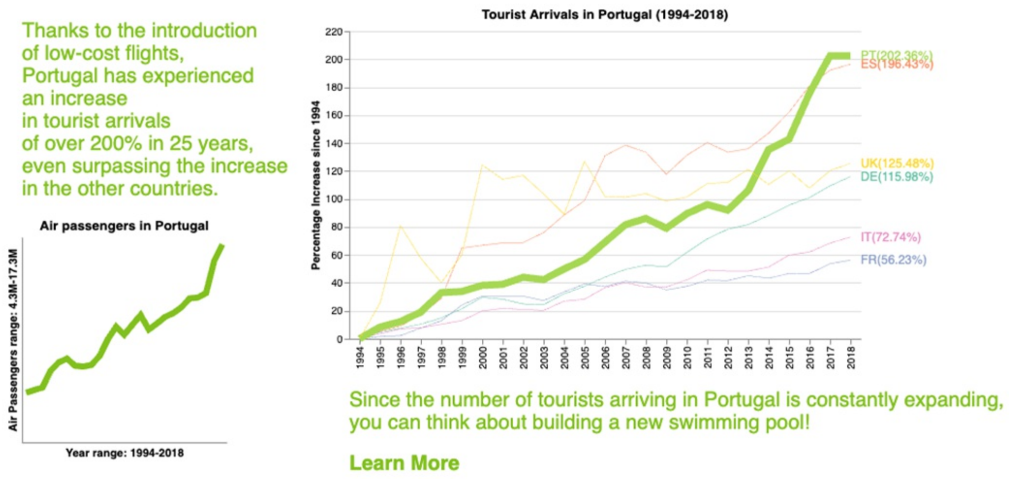
Combine all the charts

The final chart includes the context (`commentary` and `airport`), the main chart (`chart` and `annotation`), and the next steps (`image_cta` and `text_cta`).

Exercise 2

Consider again the swimming pool case study and the *collaborative efforts* next step. Implement this strategy using the *learn more* call to action. Use the `href` channel to include the link to an external website. Figure 9.11 shows a possible result.

Figure 9.11 The final data story related to the swimming pool case study


Thanks to the introduction of low-cost flights, Portugal has experienced an increase in tourist arrivals of over 200% in 25 years, even surpassing the increase in the other countries.

Since the number of tourists arriving in Portugal is constantly expanding, you can think about building a new swimming pool!

You can find the solution to this exercise in the GitHub repository of the book, under 09/collaborative-efforts/learn-more.py.

Now that you have learned how to implement the collaborative efforts next steps let’s move on to the next category: advocacy and policy.

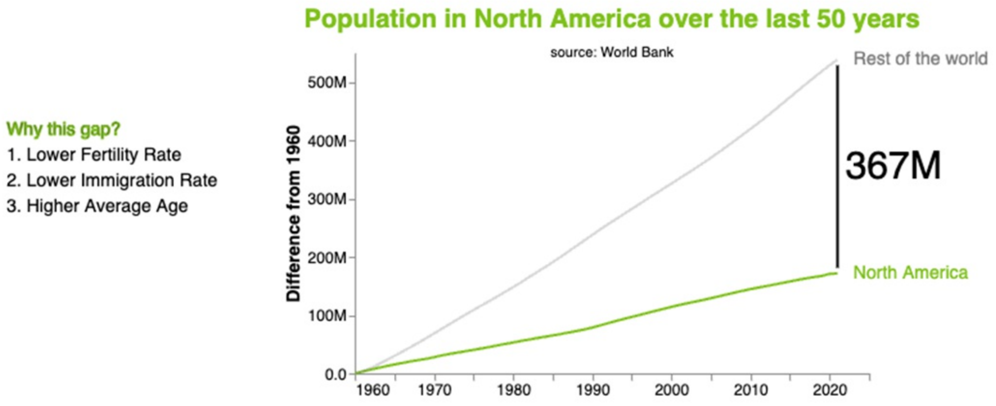
9.3.3 Advocacy and Policy

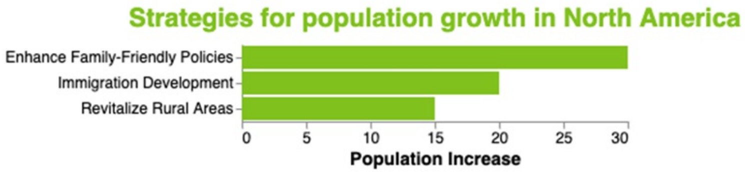
This category of next steps provides recommendations for policy changes based on data insights. There are two ways to implement this category of next steps:

- **Policy Implementation**: Working towards the execution and enforcement of policies
- **Policy Impact Assessment**: Continuously evaluating the impact of existing policies.

Consider the case study we analyzed in Chapter 3, Population in North America, which aims to study the population growth in North America. The case study discovered that there was a gap between the population growth in North America and in the rest of the world. The story, as a next step, proposed some strategies to increase the population growth in the US, as shown in Figure 9.12.

Figure 9.12 The final data story related to the North America population case study





This case study contained an advocacy and policy next step implemented as actionable recommendations.

Challenge

What can we do to improve the proposed next steps? Think about explaining better how the population increase could benefit from each suggested policy.

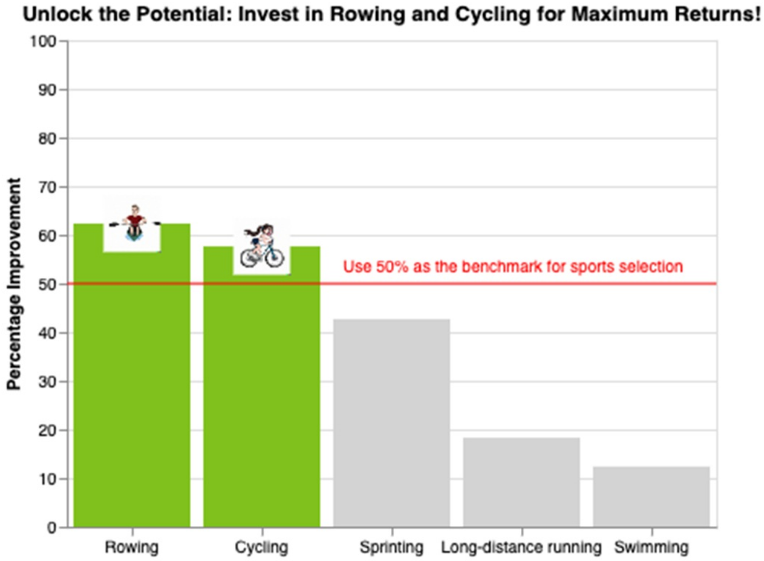
Now that you have learned how to implement advocacy and policy next steps let’s move on to the next category: training and development.

9.3.4 Training and Development

This category of next steps aims to improve skills or knowledge in areas identified by the insights, by proposing training sessions, learning initiatives, and so on.

Consider the case study we analyzed in Chapter 4, related to sport disciplines to train in, to hope to achieve good results in the upcoming competitions (Figure 9.13). As next steps for your story you proposed to invest in cycling and rowing. This next step was too generic so we can improve it.

Figure 9.13 The case study in Chapter 4



In this example, we have embedded the next steps in the title. Let’s improve them by detailing the investment we could make to improve rowing and cycling. Let’s use the support and assistance as a call to action.

Let’s implement the previous next steps using actionable recommendations as a call to action, as shown in Figure 9.14. We suppose that the performance in each sport is given by four factors: physical, psychological, tactical, and technical. Each factor contributes to the performance in different ways, based on the sport discipline. Figure 9.14 shows some hypothetical values. In addition, the data story imagines that we want to focus on technical factors to improve the athletes' performance.

Figure 9.14 The athletes’ case study with an actionable recommendation as a next-step

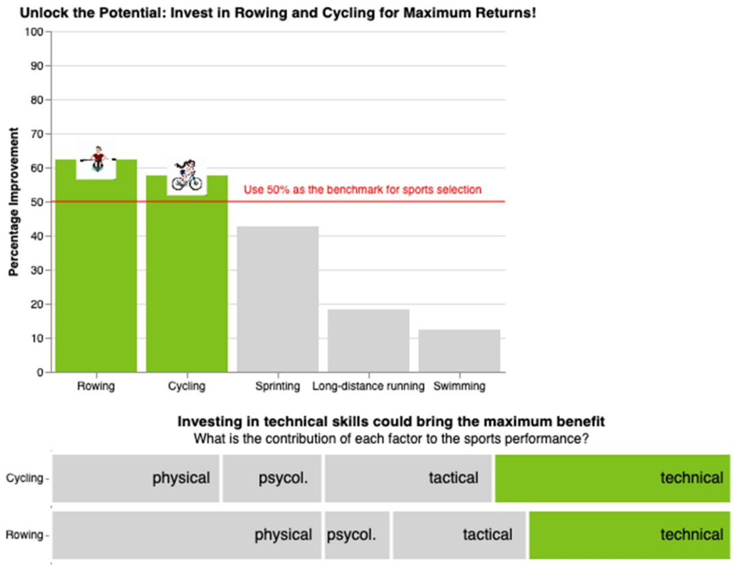


Table 9.1 shows the structure of the dataset containing the contribution of each factor to the sports performance.

Table 9.1 The factor dataset

Rowing	Cycling	Factor
40	25	physical
30	35	technical
20	25	tactical
10	15	psycol.

We will implement the next step as a stacked bar chart with a layered text. You can find the code of this example in the GitHub repository of the book, under 09/training-and-development/support-and-assistance.py. Let's start by defining the basic chart, with the encodings and properties shared by the bar chart and the text, as shown in Listing 9.12.

Listing 9.12 Building the basic chart

```
df_cta = pd.read_csv('source/factors.csv')

df_cta = df_cta.melt('Factor', var_name='Sport', value_name='Valu
base_cta = alt.Chart(df_cta).encode(
    y=alt.Y('Sport', title=''),
    x=alt.X('Value', axis=None, title='',  stack=True),
).properties(
    title=alt.TitleParams(
        text='Investing in technical skills could bring the maxim
        subtitle='What is the contribution of each factor to the
    ),
    width=600,
    height=100
)
```

How to build the basic chart

First read the factors dataset and use `melt()` to transform the `DataFrame` from wide to long format. Next, create the base chart (`base_cta`) by defining the input dataset (`alt.Chart(df_cta)`), the encodings (`x` and `y`), and the properties (`title`, `width`, and `height`). For the x channel, specify `stack=True` for the stacked chart.

Now we can build the stacked bar chart, as described in Listing 9.13.

Listing 9.13 Building the stacked bar chart

```
cta = base_cta.mark_bar(
    strokeWidth=3,
    stroke='white'
).encode(
    color=alt.Color('Factor',
                    scale=alt.Scale(range=['lightgrey', '#80C11E'
                    ),
                    legend=None),
)
```

How to build the stacked bar chart

Use the base chart to build the stacked bar chart. Set the mark property (`mark_bar`) and the color channel.

Finally, draw the text and combine it with the other parts of the data story, as shown in Listing 9.14.

Listing 9.14 Drawing the text

```
text_cta = base_cta.mark_text(
    xOffset=-35,
    fontSize=14,
    color='black'
).encode(
    text = 'Factor:N',
)
chart = ((chart + annotation) & (cta + text_cta)
).configure_view(
    strokeWidth=0
)
```

Drawing the text

Use the base chart to build the text chart. Add the mark property (mark_text) and the text channel. Next, draw the final chart, which includes the the main chart (`chart`), the `annotation`, and the next steps (`cta` and `text_cta`) below.

Challenge

Improve the next steps in Figure 9.14 by replacing the y labels (Cycling and Rowing) with the same icons shown in the main chart.

Now that you have learned how to implement training and development next steps let’s move on to the next category: strategic actions.

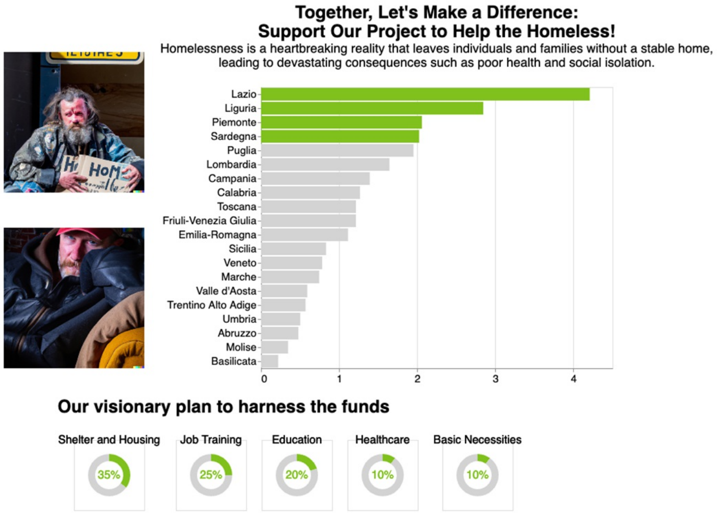
9.3.5 Strategic Actions

This category of next steps proposes to implement changes by altering existing strategies, processes, or systems based on the insights gathered. This may involve:

- Developing new strategies: Creating entirely new approaches
- Allocating resources: Assigning specific resources (financial, human, technological).

Consider the case study we analyzed in Chapter 5, the homeless people, shown in Figure 9.15. The case study focused on searching for funds to finance a project about homeless people. In the next steps section, the case study proposed to implement some strategic actions to mitigate the homeless problem.

Figure 9.15 The case study described in Chapter 5, the homeless people



This case study implemented the next steps as actionable recommendations.

Challenge

Among the other calls to action (learn more, social sharing, visualized roadmaps, engagement and interaction, and support and assistance) which ones could you use for this case study and why?

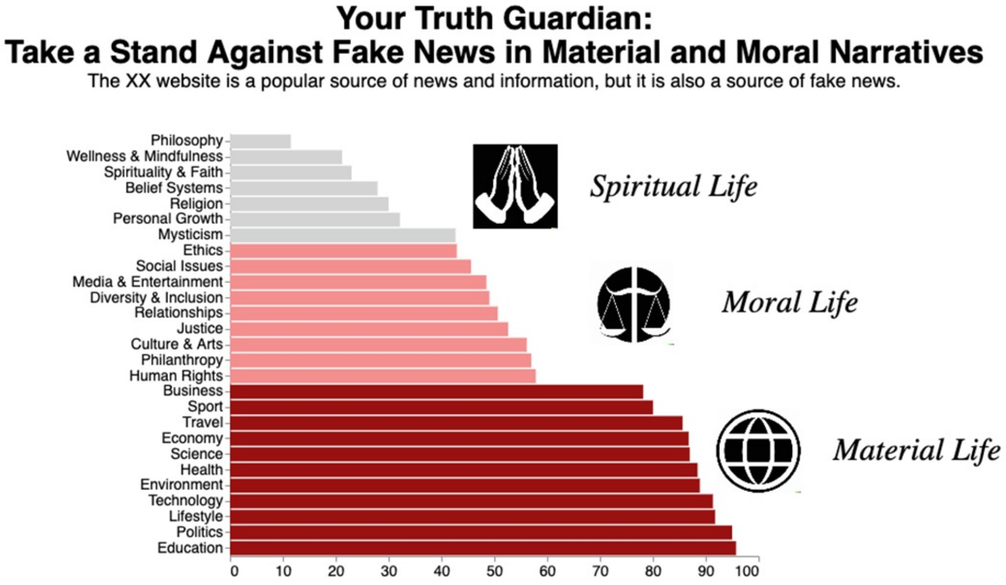
Now that you have learned how to implement strategic action next steps let's move on to the next category: engagement and communication.

9.3.6 Engagement and Communication

This category of next steps aims to educate and inform the audience, to influence their behavior, or involve them in activities related to the data findings.

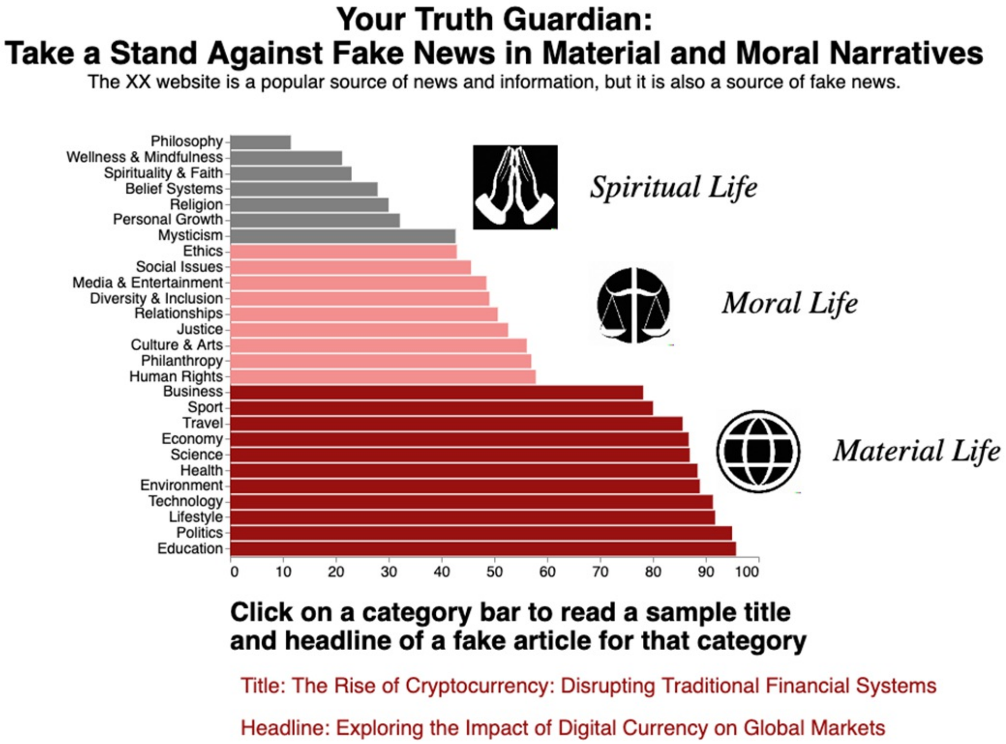
Consider the case study we analyzed in Chapter 5, fake news, as shown in Figure 9.16. This case study focused on identifying the news categories where the probability of being fake was higher. We included the next steps in the title.

Figure 9.16 A summary of all the examples described in the previous chapters



We can improve next steps by adding an interactive box showing a sample article title and headline, whenever the audience clicks on a specific bar, as shown in Figure 9.17. Specifically, we use the engagement and interaction call to action to implement the next step of this example.

Figure 9.17 A summary of all the examples described in the previous chapters



We implement the text containing the title and the headline as a text chart changing based on the selected bar. Initially, no text is shown. You can find the code related to this example in the GitHub repository of the book, under 09/engagement-and-communication/engagement-and-interaction.py.

We will build a selection point and we will connect it to the Category field of the bar chart. Next, we will use the selection point to filter the text to show in the next step part. We will assume to have a dataset containing an article title and a headline for each category, as shown in Table 9.2.

Table 9.2 The articles dataset

Category	Title	Headline
Politics	The Future of Global Diplomacy: Navigating 21st Century Challenges	How International Relations Are Shaping Our World Today
Economy	The Rise of Cryptocurrency: Disrupting Traditional Financial Systems	Exploring the Impact of Digital Currency on Global Markets

Justice	Reforming Criminal Justice: Toward a Fair and Equitable System	Addressing Inequality within Legal Systems Across Nations
Religion	Interfaith Dialogue in Modern Society: Building Bridges Amidst Differences	Understanding Diversity in Religious Beliefs and Practices

Start by creating a selection point for your chart. Add the Category field of your dataset as a working field for your interactive element, as shown in Listing 9.15.

Listing 9.15 Defining the selection point

```
click = alt.selection_point(name='Select',
                            fields=['Category'], empty=False)
```

How to define the selection point

Define a selection point named `Select` that allows users to select data points based on their `Category` values. The `empty` parameter set to `False` ensures that at least one data point must be selected.

Now, use the defined selection point to set the color of a selected bar, as shown in Listing 9.16. The code bolds the modified code with respect to the original version implemented in Chapter 5.

Listing 9.16 Building the interactive bar chart

```
color=alt.Color('Macro Category:N',
        scale=alt.Scale(
            range=['#991111', '#f38f8f','gray'],
            domain=['Material Life', 'Moral Life', 'Spiritual Lif
        ),
        legend=None
    )

chart = alt.Chart(df).mark_bar(
).encode(
    y=alt.Y('Category:N',
            sort='x',
            title=None,
            axis=alt.Axis(labelFontSize=14)
            ),
    x=alt.X('Percentage of Fake Articles:Q',
            title=None,
            axis=alt.Axis(labelFontSize=14,
                          titleFontSize=14),
    ),
    color=alt.condition(click | ~click, color, alt.value('lightgr
    )
).properties(
    width=400,
    height=400
).transform_calculate(
    'Percentage of Fake Articles', alt.datum['Number of Fake Arti
).add_params(
    click
)
```

How to build the interactive bar chart

First define the color channel as a variable (we will use it also for the text chart, so we define it as a variable to avoid to write it twice). Next, build the bar chart, by defining the `Y` and `X` channels. Also define the color channel as dependent on the `click` selection point. Use the expression `click | ~click` to set the color when click is `True` (this happens with a selected bar) and when it is `False` (this happens when no bar is selected because we have set `empty` to `False`). Also, set the properties and add the click interactivity through the `add_params` method.

Now we can draw the text chart, as the combination of two charts, the title and the headline. First, build the base chart, with the details shared between the two charts, as shown in Listing 9.17.

Listing 9.17 Building the basic chart

```
df_cta = pd.read_csv('source/articles.csv')
df_cta['Macro Category'] = df_cta['Category'].apply(lambda x: 'Ma

base_cta = alt.Chart(df_cta).mark_text(
    fontSize=20,
    align='left',
).encode(
    color=color
).transform_filter(
    click
)
```

How to build the basic chart

Start by reading the article dataset. Then, add the macro category, used to color the text. Finally, define the basic chart, by specifying the mark property (`mark_text`), the color channel and the filter used to select the text dynamically.

Now, let’s proceed with the article title, as shown in Listing 9.18.

Listing 9.18 Building the article title

```
title_cta = base_cta.encode(
    text='Label:N',
).properties(
    title=alt.TitleParams(
        text=['Click on a category bar to read a sample title', '
        fontSize=25,
        offset=20,
        anchor='start'
    )
).transform_calculate(
    Label= 'Title: ' + alt.datum.Title
)
```

How to build the article title

Use the base chart to draw the title article text. Specify the `text` channel, using the `Label` column, generated through the `transform_calculate` method. Also, set the chart properties.

Finally, build the article headline, using a similar procedure as the article title, as shown in Listing 9.19.

Listing 9.19 Building the article headline

```
headline_cta = base_cta.encode(
    text='Label:N'
).transform_calculate(
    Label= 'Headline: ' + alt.datum.Headline
)


chart = (chart & (title_cta & headline_cta)).configure_axis(
    grid=False
).configure_view(
    strokeWidth=0
)
chart.save('engagement-and-interaction.html')
```

How to build the article headline

Use the base chart to draw the headline article text. Next combine the charts to obtain the final data story (suppose that chart contains the main chart). Use `configure_axis(grid=False)` to remove the grid from the charts.

So far you have learned how to implement next steps using different calls to action. In the following section, you’ll learn how to anchor your story to an ethical framework.

9.4 Anchoring the Action to an Ethical Framework

An ethical framework serves as a guideline for decision-making based on principles that guide moral behavior. It provides a framework for evaluating the potential consequences of our actions and helps us identify the best course of action based on what is right and just.

Actions not grounded in ethical considerations can lead to unintended consequences and may even cause harm. Some of the most popular ethical frameworks include (Berengueres 2019):

- **Utilitarianism**: This framework bases ethical decisions on maximizing utility or happiness. According to this view, an action is right if it leads to the greatest happiness for the greatest number of people.
- **Deontology**: This framework focuses on following moral rules or duties.

It argues that certain actions are inherently good or bad, regardless of their consequences.
- **Virtue ethics**: This framework emphasizes the importance of developing virtues such as courage, honesty, and compassion. It argues that good people are more important than following rules or maximizing happiness.
- **Care ethics**: This framework emphasizes the importance of relationships and empathy. It bases ethical decisions on caring for others and responding to their needs.

Regardless of the ethical framework chosen, the next steps must be anchored within one if you wish to safeguard your call to action against misinterpretation or misuse. This means considering the potential impact of our actions on others and ensuring that they align with our values and principles. By doing so, we can move beyond simply making decisions based on what is expedient and towards creating positive change rooted in ethical considerations.

To adapt next steps to a specific ethical framework, we can implement the following two steps:

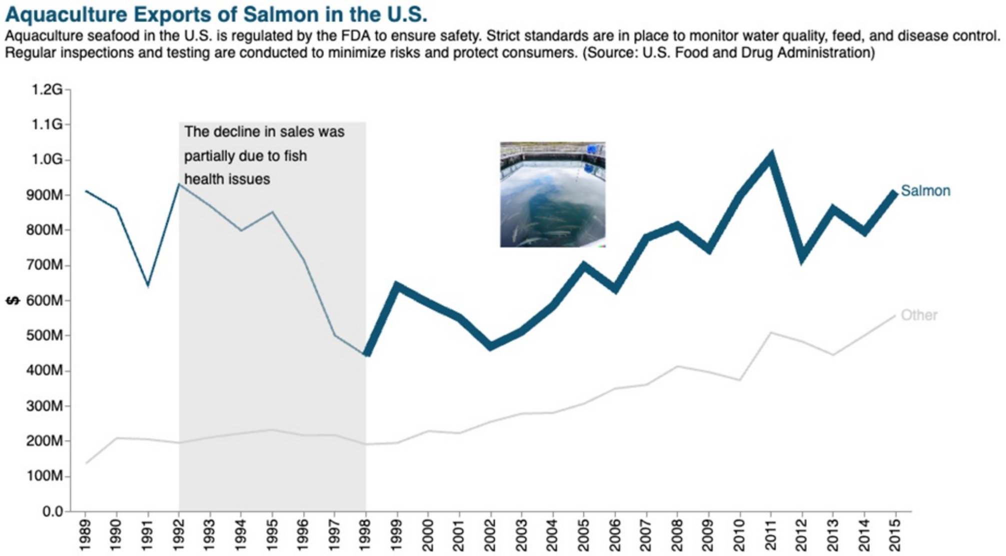
- Using ChatGPT to generate a preliminary text
- Revising the produced content.

Let’s apply this strategy to the salmon aquaculture case study implemented in the previous chapters.

9.4.1 The Salmon Aquaculture Case Study

Consider the salmon aquaculture case study. Just for a quick recap, the salmon aquaculture case study aims to study the problem of safety in the salmon aquaculture in the U.S.. So far, we have represented the amount of dollars earned in the U.S. from the salmon exports. We considered two types of scenarios, a positive one, where we focused on the increase of earnings, and the negative one, where we focused on the disease period. In this chapter, we only focus on the positive scenario, but you can easily adapt the described concepts also to the negative one. Figure 9.18 shows the data story produced at the end of the previous chapter, turning information into knowledge.

Figure 9.18 The story produced after turning information into knowledge



Let’s suppose that as the next step for our data story we want to propose some strategic actions to continue the positive increase in the amount of dollars trendline. We propose the following next steps:

- Emphasize salmon safety measures
- Enhance salmon aquaculture practices
- Promote sustainable salmon consumption

Let’s use ChatGPT to adapt the proposed next steps to the different ethical framework. Use the following prompt for ChatGPT to generate the utilitarianism-based texts:

Consider the following ethical framework: Utilitarianism

This framework bases ethical decisions on maximizing utility or happiness. According to this view, an action is right if it leads to the greatest happiness for the greatest number of people. Reformulate the following titles to adapt to the Utilitarianism framework:

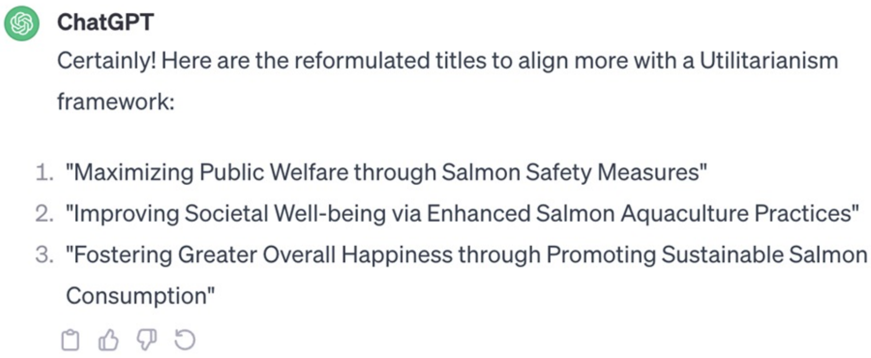
Emphasize salmon safety measures

Enhance salmon aquaculture practices

Promote sustainable salmon consumption

Figure 9.19 shows a possible output produced by ChatGPT.

Figure 9.19 A possible output generated by ChatGPT



ChatGPT

Certainly! Here are the reformulated titles to align more with a Utilitarianism framework:

1. "Maximizing Public Welfare through Salmon Safety Measures"
2. "Improving Societal Well-being via Enhanced Salmon Aquaculture Practices"
3. "Fostering Greater Overall Happiness through Promoting Sustainable Salmon Consumption"

Now, let’s review it by asking the following question: is this output really anchored to an utilitarian framework? If the answer is yes, we can incorporate the output in our story, otherwise, we must correct it before adding it to our story. In the next chapter, we will see more of the details regarding the ethical problems connected to ChatGPT and Generative AI in general. For now, it’s sufficient to control the produced output.

In our case, the produced output contains words such as Welfare, Societal well-being, and overall-happiness that are in line with the utilitarianism framework.

Apply the same procedure to the other ethical frameworks. Table 9.3 shows some possible outputs by ChatGPT.

Table 9.3 The outputs generated by ChatGPT for each ethical framework

Original Text Ethical Framework	Emphasize salmon safety measures	Enhance salmon aquaculture practices	Promote sustainable salmon consumption

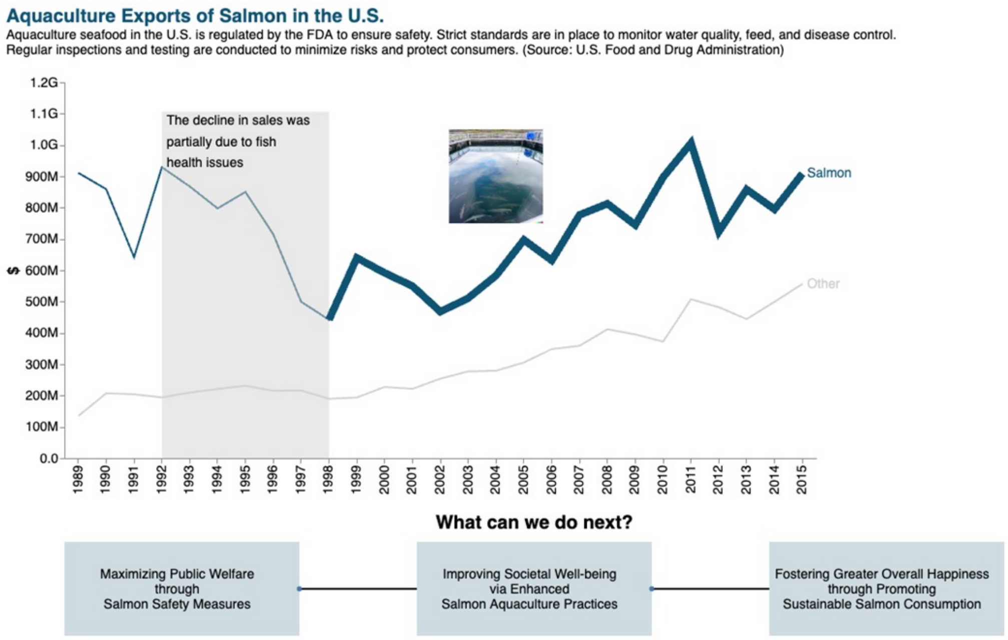
Utilitarianism	Maximizing Public Welfare through Salmon Safety Measures	Improving Societal Well-being via Enhanced Salmon Aquaculture Practices	Fostering Greater Overall Happiness through Promoting Sustainable Salmon Consumption
Deontology	Uphold Salmon Safety Measures	Adhere to Enhanced Salmon Aquaculture Practices	Advocate for Ethical and Sustainable Salmon Consumption
Virtue Ethics	Cultivating Ethical Stewardship for Salmon Well-being	Fostering Virtuous Aquaculture Practices for Salmon Health	Advancing Virtues through Sustainable Salmon Consumption
Care Ethics	Prioritizing Salmon Well-being: Implementing Safety Measures	Nurturing Salmon: Improving Aquaculture Practices with Empathy	Caring Consumption: Fostering Sustainable Salmon Choices

Once you have chosen your specific ethical framework, you can use it to implement the next steps for your case study.

Exercise 3

Consider the output produced by ChatGPT for a utilitarian ethical framework and implement it as actionable recommendations. For example, you can generate the final data story described in Figure 9.20.

Figure 9.20 The final data story for the aquaculture case study



You can find the solution to this exercise in the GitHub repository of the book, under CaseStudies/aquaculture/from-knowledge-to-wisdom/chart.py.

Now that you have learned how to anchor your next steps to an ethical framework, let’s move on to the last part of this chapter, how to place your next steps strategically in your data story.

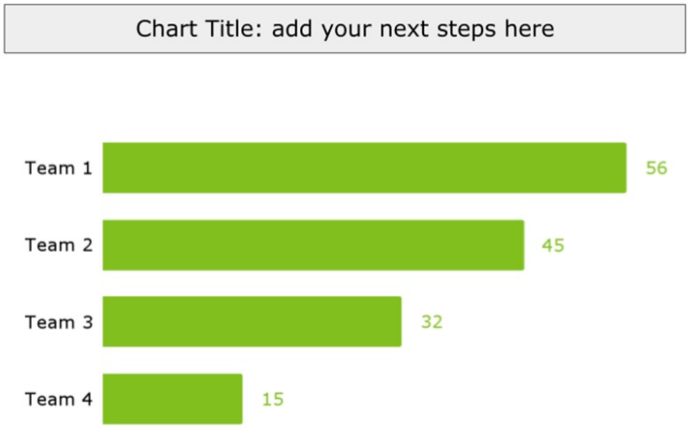
9.5 Strategic Placement of Next Steps

Next steps must follow the main point of your data story. For this reason, we suggest you place your next steps in one of the following three main positions in your data visualization chart: in the title, on the right, or below the chart. In the remainder of this section, we will investigate each next-step position separately.

9.5.1 Title

Adding next steps in the title means adding it to the top part of the chart, as shown in Figure 9.22.

Figure 9.21 The next-step position at the top of the chart



Use the title to state what the audience should do with the data story.

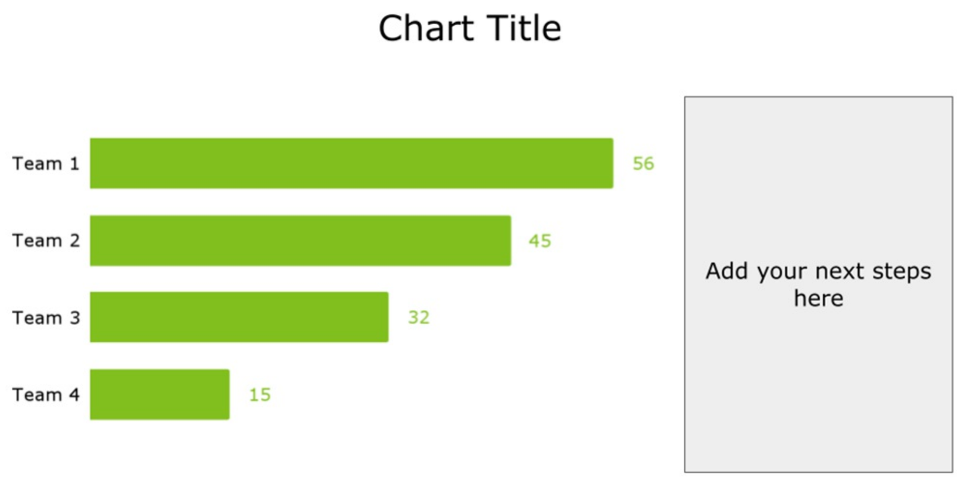
Challenge

Consider the case studies described in Chapters 1-5 (Figure 9.2). Which ones added the next steps in the title?

9.5.2 Right

Adding next steps on the right of a chart involves adding it immediately after the chart, as shown in Figure 9.22.

Figure 9.22 The next-step position on the right of the chart



Use the right part of the chart to add deep details such as textual descriptions, and other charts to explain to the audience what to do after reading the previous parts.

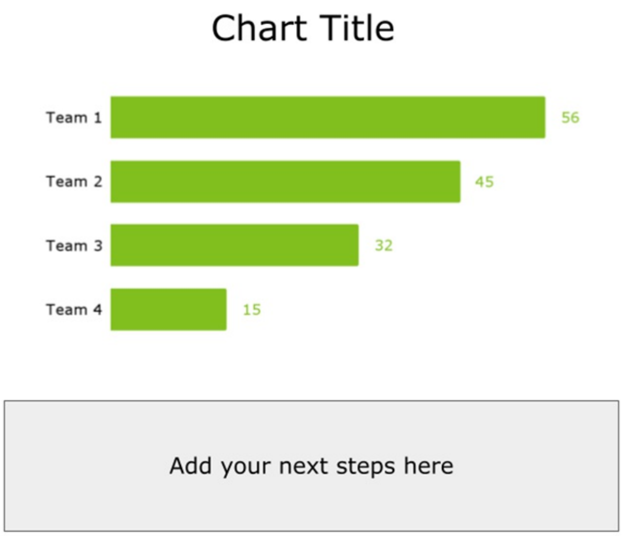
Challenge

Consider the case studies described in Chapters 1-5 (Figure 9.2). Which ones added the next steps on the right?

9.5.3 Below

Next steps below a chart differ from the previous case only in their position, as shown in Figure 9.23.

Figure 9.23 The next step position below the chart



Challenge

Consider the case studies described in Chapters 1-5 (Figure 9.2). Which ones added the next steps below the chart?

In the first part of this chapter, you have learned the concept of wisdom, call to action, and next steps. Next you have learned how to anchor your next steps to an ethical framework. Finally, you have seen how to place the next steps strategically in your story.

In the next chapter, you will see how to publish your data story and what ethical implications could have the use of Generative AI in data storytelling.

9.6 Summary

- Wisdom is the ability to use your experience and knowledge to make sensible decisions or judgments. It depends on knowledge, experience, and good judgment.
- Transforming knowledge into wisdom involves adding the next steps to the data story. Next steps are the resolution of our data story and define what the audience should do after reading the story.
- We can classify the next steps into different categories based on the specific objectives they aim to achieve: awareness and campaigns, collaborative efforts, advocacy and policy, training and development,

strategic actions, and engagement and communication.
- A Call to Action (CTA) is a practical implementation of a next-step strategy. The same next step can be implemented through different calls to action.
- We classify the calls to actions into the following popular categories: learning, sharing, acting, visualizing roadmaps, engaging and interacting, supporting and assisting.An ethical framework serves as a guideline for decision-making based on principles that guide moral behavior. Next steps must be anchored to an ethical framework.

9.7 References

Berengueres, Jose Berengueres and Sandell, Marybeth (2019) Introduction to Data Visualization & Storytelling: A Guide For The Data Scientist (Visual Thinking). Self-published.

[1] https://www.collinsdictionary.com/dictionary/english/wisdom

[2] https://dictionary.cambridge.org/dictionary/english/call-to-action

10 Publishing the Data Story

This chapter covers

- Ethical considerations before disseminating the story
- Using Generative AI ethically
- Crediting the sources
- Exporting the story
- Publishing the story using streamlit
- Alternative ways to publish the story: Tableau, Power BI, and Comet

This is the last chapter of this book. Throughout this book we have built our data story. Now it is time to show it to our audience. This chapter is all about taking your data story and getting it to your audience, and doing so in an ethical manner. In the first part of this chapter, we’ll see some ethical points to consider before publishing your story and how to use Generative AI ethically. Next, we’ll see how to credit the sources correctly, including both data sources and Generative AI. Then, we’ll focus on Streamlit, a Python library fully integrated with Altair. Streamlit helps you build a complete standalone website hosting your Python code. Finally, we’ll see some alternative ways to publish your story, including techniques to integrate it into some popular tools for data analytics and visualization: Tableau and Power BI. We’ll also describe how to integrate your data story in Comet, an experimentation platform for Machine Learning.

Let’s start with the first point, ethical considerations, before disseminating the story.

10.1 Ethical Considerations Before Disseminating the Story

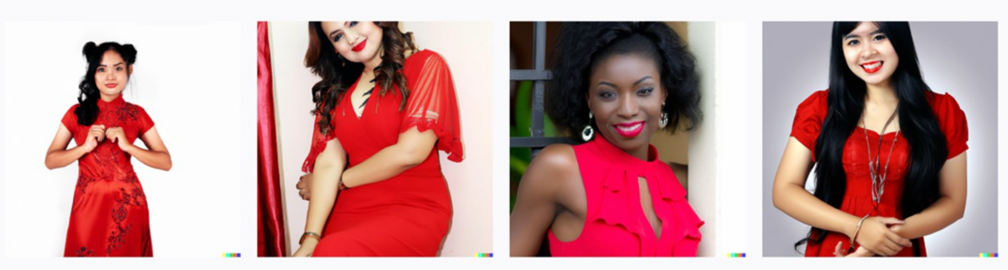
We have almost reached the end of this fantastic journey, which has seen us gradually build our data-based stories. We used the DIKW pyramid to assemble our story, and now we are ready to disseminate it to our audience.

However, before seeing the techniques for publishing the story, I would like to focus on the Generative AI used throughout the book. The first consideration to keep in mind is that Generative AI is not an omniscient oracle but is the result of human effort, and as such, reflects the characteristics of humanity, made up of intelligence, scientific progress, and so on, but also of discrimination, inequalities, and injustices. Generative AI, while a product of human ingenuity and scientific advancement, inherently carries the biases and limitations of the data it is trained on, reflecting not only our knowledge but also our societal and historical prejudices. It's crucial to remember that these systems are limited by their training data and algorithms and do not possess any innate understanding or consciousness.

All this, everything that humanity is, is therefore reflected in Artificial Intelligence, both what is good and what is less good in humanity. It is for this reason that when Generative AI is used as a work tool, particularly in Data Storytelling, it must always be handled with care, paying attention to both the benefits and the possible damage that it could cause among the most vulnerable people. Generative AI products could, in some way, offend the sensitivity of minorities. Generative AI could even be used to build manipulated stories, yet with a plausible appearance.

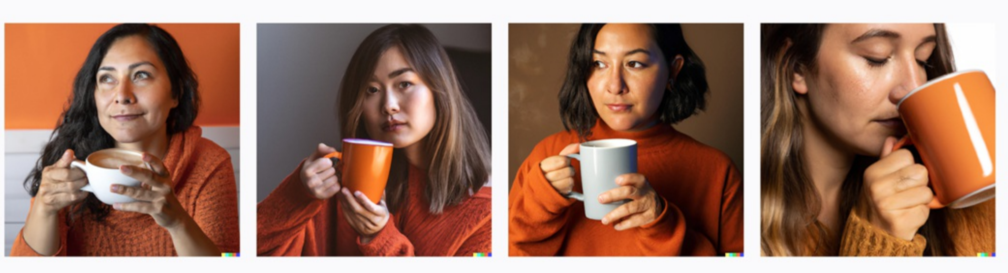
To get a practical idea of discrimination that may occur inadvertently while using Generative AI, consider the following prompt for DALL-E: *a photo of a woman wearing a red dress*. Figure 10.1 shows a possible output generated by DALL-E.

Figure 10.1 A photo of a woman wearing a red dress



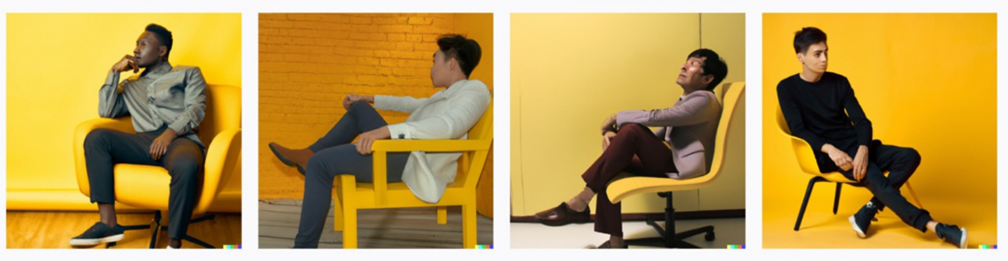
Now, consider this prompt for DALL-E: *a woman wearing an orange sweater drinking a coffee*. Figure 10.2 shows a possible output generated by DALL-E.

Figure 10.2 A woman wearing an orange sweater drinking a coffee



Finally, consider the following prompt: *a man sitting on a yellow chair*. Figure 10.3 shows a possible output generated by DALL-E.

Figure 10.3 A man sitting on a yellow chair



We may continue generating images more and more, but we will probably always obtain the same results. What do Figures 10.1, 10.2, and 10.3 have in common? The problem is that almost all the represented persons have dark hair (except for the woman on the right in Figure 10.2). This may be a minor problem if we generate a small number of images. However, if we use DALL-E to generate images at a large scale, using only dark-haired people may generate a sort of discrimination towards blonde-haired people.

The previous example shows a simple case where Generative AI is partial. The problem is probably in the data used to train the generative model. This data may contain more dark-haired people than blonde-haired people. This is a rather benign example, but it clearly illustrates the point that you may need to be mindful about bias and partiality within your outputs.

Challenge 1

Can you find other kinds of discrimination while generating images with people in DALL-E? For example, try to generate a woman with a red dress. What kind of stereotype would you obtain?

Challenge 2

DALL-E associates color and mood. Can you figure out this? For example, try the following prompts: 1) a woman wearing a yellow sweater, 2) a woman wearing a blue dress. Do you obtain different moods, although you didn’t specify any?

Ethical issues caused by Generative AI depend on different causes, such as AI hallucinations, bias, and copyright. We will not delve into the technical details related to these issues; for more information, you can read the resources described in the references of this chapter. In the remainder of this section, we will shortly describe the potential ethical issues Generative AI may introduce. Let’s start with hallucinations.

10.1.1 AI Hallucinations

Hallucination happens when Generative AI generates content that does not correspond to reality. Hallucinations within AI can lead to the creation of misleading or entirely fabricated data, potentially causing significant ethical concerns. Hallucination is a problem brought about by Generative AI’s very design, in the sense that the LLM behind Gen AI is generated without any communication of intent. It simply generates a text that is **statistically** more probable given the training dataset. Additionally, for instance, if a generative AI is primarily trained on datasets containing English language texts, it might struggle with accurately understanding or generating content in less represented languages, reflecting a statistical bias towards English. Hallucinations are not directly an ethical issue but can lead to ethical problems due to their potential misuse for generating content for user manipulation.

Hallucination in Generative AI can manifest in various ways. For example, a data story completely generated by AI could describe a non-existent political scandal, potentially misleading the public and impacting decision-making based on false information.

To mitigate the hallucination problem, we always recommend having a **human-in-the-loop validation**. Before incorporating any AI-generated content in your story, please judge and review it.

Challenge 3

Just for fun, consider the case study in Chapter 2, Swimming Pool. As a quick reminder, the case study focused on the possibility of building a new swimming pool in a Portuguese hotel. Use the following prompt to generate the next steps: *Consider the following scenario: the case study focused on the possibility of building a new swimming pool in a Portuguese hotel. The data story showed an increasing number of tourists in Portugal in recent years.* Write some hallucinated next steps. What do you obtain?

To mitigate hallucinations, you can try to set the following parameters in your prompt:

- **Temperature**: The temperature controls the degree of randomness applied during the output generation process. It allows users to tailor the level of creativity and unpredictability in the generated content. It ranges from 0 for more structured and predictable outputs to 2 for more creative and unexpected results. The default value rests at 1. To introduce temperature in your prompt, simply add the text **set temperature = N** (for example, use temperature = 2). We can use a lower temperature value to reduce the probability that the model hallucinates.
- **Top P**: Top P is also known as nucleus sampling or penalty-free sampling. It helps to control the diversity of the generated text. Use this technique to generate responses that don't completely deviate from the topic. The range is between 0 and 1. A higher Top P makes the output more diverse, while a lower value makes the model more deterministic. The default value is 1. To introduce Top P in your prompt, add the text **set Top P = N** (for example, use Top P = 1).

Usually, you set one parameter per prompt.

Challenge 4

Set the temperature to 0 or the Top P value to 0 in the prompt of Challenge 3 and compare the result with the output of Challenge 3. Do you note some differences?

You can find a detailed description on how to set the temperature and the Top P parameters in my blog post, How to Improve Your ChatGPT Outputs Using Configuration Parameters[1].

Now that we’ve covered hallucinations let’s move to the next problem: Bias.

10.1.2 Bias

Bias is a systematic and often unconscious inclination, prejudice, or tendency that influences decision-making, actions, perceptions, or judgments in favor of or against a particular person, group, object, or idea. Bias relies on human beliefs, stereotypes, or discrimination. Since LLMs are trained on datasets mostly created by humans, inevitably, these datasets contain bias. Bias is intrinsic to humans, so we can’t remove it from our datasets.

Even in a hypothetical scenario where people are equal and free, and there is no discrimination and war, bias may occur. In fact, **bias is multifaceted**. Bias in AI is not limited to negative aspects like discrimination or war. Bias can also be in the form of cultural preferences, idiomatic expressions, or even what is considered "normal." An LLM trained on data from even an ideal world might still develop biases based on what is prevalent or dominant in that data.

In addition, even with ideal data, LLMs might still develop biases due to their design, the algorithms they use, or the inherent limitations in understanding and processing human language.

In other words, our LLM could still exhibit bias even in this hypothetical scenario.

- There are different perspectives to classify bias, such as those proposed by Baer in his book (see the references at the end of this chapter for further details).One possible approach classifies bias in the following types:**Data Bias** is the presence of bias in the training set. It derives from different causes, such as the underrepresentation of some groups in

the training set. LLMs are training with data extracted from the Web. However, the text on the Web is written by a minority. This means that although big data is used to train LLMs, their size does not guarantee diversity.
- **Algorithm Bias** derives from the assumptions and decisions made during the design, coding, or implementation of machine learning algorithms. This bias can arise due to feature selection, model complexity, and other technical problems related to the algorithms.
- **Measurement Bias occurs** when the methods or tools used to collect data systematically misrepresent or distort the information being gathered. This bias can originate from different factors, such as instruments, human observers, etc.

At the time of writing this book, there was no definitive solution to remove bias from generative AI tools. However, some possible techniques to mitigate bias could include data cleaning and balancing, inserting a human in the middle, model evaluation, and so on. Removing bias from Generative AI would also mean removing it from humanity first, which could be significant progress for the world but is unlikely to happen any time soon. Anyway, we could mitigate bias by always paying attention to the generated output and anchoring our data story to an ethical framework, as explained in the previous chapter.

Having considered the problem of bias, let’s move to the following problem: Copyright.

10.1.3 Copyright

Generative AI models have been trained on huge quantities of data, derived especially from content shared freely in the public space. However, the creators of the original datasets used to train the models do not allow access to them, so we don’t know whether proprietary data have also been used to train the models. In this case, AI-generated content might raise questions about intellectual property rights and ownership. Copyright questions may be connected to the fact that Generative AI models are black boxes, and the data used to train them is unavailable.

For example, consider an AI system trained on an extensive database of music that generates a piece similar to an existing copyrighted song. Determining the original creator becomes complex, raising questions about the ownership of AI-generated content and the rightful attribution of intellectual property, potentially leading to legal disputes between creators and AI systems.

Before using Generative AI, you should alway on understand copyright law, use clear licensing agreements for data sources, create original datasets, emphasize attribution and acknowledgment, implement copyright filters, and seek regular legal consultation.

All the described ethical issues are unsolved at the time of writing this book. For this reason, we recommend consistently controlling the output produced by Generative AI when using it for data storytelling and, in general, for all the application fields.

Now that we have learned the basic causes of ethical problems related to Generative AI, let’s deepen our understanding of how to use Generative AI ethically.

10.1.4 Using Generative AI Ethically

Incorporating Generative AI into Data Storytelling may bring further ethical issues. For this reason, it is essential to follow the guidelines defined by experts in using AI. Many initiatives exist to regularize the use of AI and Generative AI in all domains, such as the European EU AI Act (European Commission, 2021) and the White House’s Executive Order on the Safe, Secure, and Trustworthy Development and Use of Artificial Intelligence (White House, 2023). You can find the links to these documents at the end of the chapter, in the references section.

Referring to the specific field of Data Storytelling, using Generative AI ethically means respecting the guidelines for human values. For this reason, in this book, we follow the guidelines proposed by UNESCO.

The UNESCO AI ethics guidelines emphasize four core values:

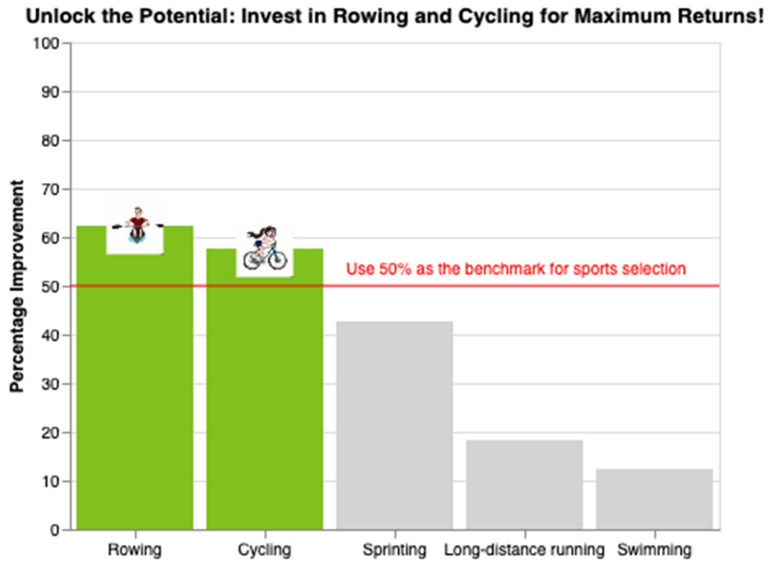
- **Human rights and dignity**: Every data story, including the pieces generated through Generative AI, must respect human rights and dignity. The UNESCO guidelines say explicitly that *no human being or human community should be harmed or subordinated, whether physically, economically, socially, politically, culturally or mentally, during any phase of the life cycle of AI systems*.
- **Peaceful and just societies**: AI should be used to foster harmony and equity within communities, promoting fairness, transparency, and accountability in decision-making processes.
- **Diversity and inclusiveness**: AI should respect the richness of human diversity in all its forms, including but not limited to race, gender, ethnicity, culture, and more.
- **Environmental flourishing**: AI should prioritize sustainability and contribute positively to environmental preservation.

We can apply the previous guidelines to check if AI-generated content is ethically correct. For each AI-generated content, we should answer the following questions:

1. Does the AI-generated content respect human rights?
2. Does the AI-generated content respect society?
3. Does the AI-generated content respect diversity and inclusiveness?
4. Does the AI-generated content respect the environment?

To understand how to apply these guidelines to a practical data story, consider the case study we analyzed in Chapter 4, related to sports disciplines to train in, to hope to achieve good results in the upcoming competitions. As the next steps for your story, we proposed to invest in cycling and rowing. Figure 10.4 shows the final data story.

Figure 10.4 The sports disciplines data story described in Chapter 4



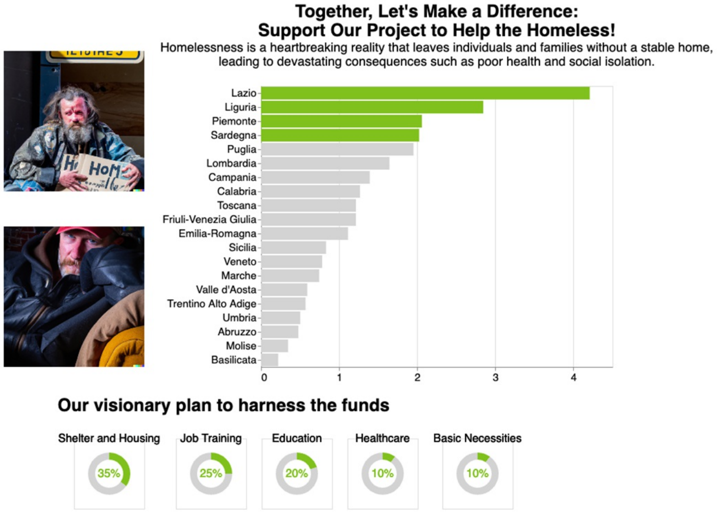
Let’s look at each part generated by AI:

- General Title: Unlock the Potential: Invest in Rowing and Cycling for Maximum Returns!
- Image 1: The white man practicing rowing
- Image 2: The white woman is practicing cycling.

The only problem in the AI-generated content for this scenario regards diversity and inclusiveness. The two images, in fact, describe two white-skinned individuals. To make the data story fit the ethical guidelines, we could replace one of the two images with an individual from another ethnicity.

10.1.5 Exercise

Figure 10.5 The case study described in Chapter 5, the homeless people



1. Identify the AI-generated content
2. For each AI-generated content, answer the following questions:
    a. Does the AI-generated content respect human rights?
    b. Does the AI-generated content respect society?
    c. Does the AI-generated content respect diversity and inclusiveness?
    d. Does the AI-generated content respect the environment?

Now that you have learned how to use Generative AI ethically let’s move on to the next step, crediting the sources.

10.2 Crediting the Sources

Crediting the sources means referencing the sources used in the data story. This is particularly important because it allows you to recognize the work done by others. It also adds credibility to the story, as the audience can personally check the sources used in the story. What types of sources should be credited? In general, any source used to make the story, but in particular the following sources:

- The data source
- The fact that Generative AI was used
- Any documents used for fine-tuning or RAG (Retrieval Augmented Generation).

Traditionally, we can add credits to a data story in four alternative places, although you can use your creativity to place it wherever you want:

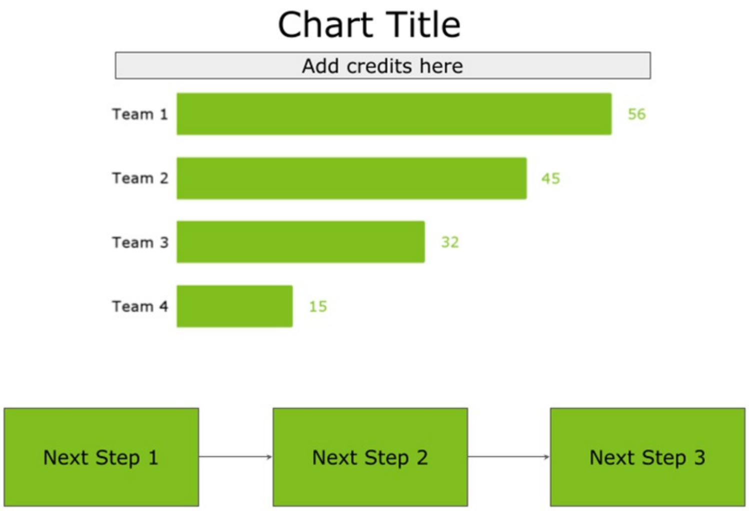
- Under the title/subtitle
- Under the main chart
- Under the next steps
- Sideways.

Let’s investigate each of them separately.

10.2.1 Under the Title/Subtitle

Placing the credits under the title/subtitle generates a sense of trust in the audience from the story's beginning. Figure 10.6 shows an example of credits placed under the title/subtitle.

Figure 10.6 An example of a data story with credits under the title/subtitle

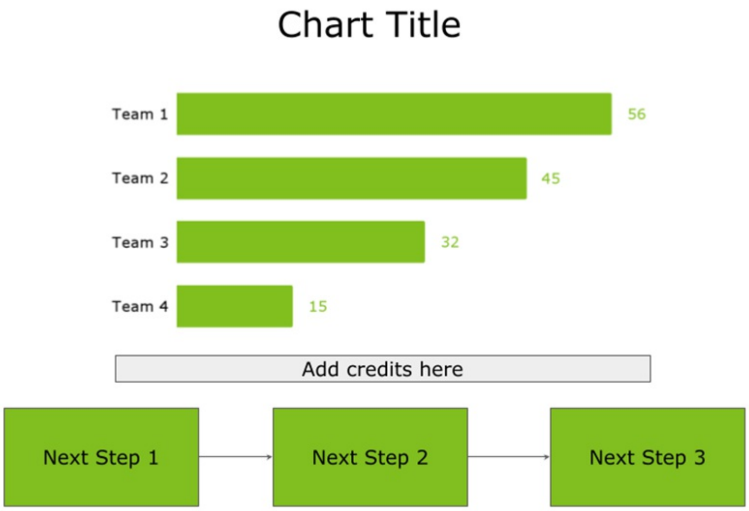


Use this placement if you want your audience to know the sources from the story's beginning. Although this placement may generate trust, it could also be distracting since the audience may leave your story to search for the sources.

10.2.2 Under the Main Chart

Placing the credits under the main chart involves adding a detail to the main point of the story. This helps reinforce the essential points of the story. Figure 10.7 shows an example of credits placed under the main chart.

Figure 10.7 An example of a data story with credits under the main chart

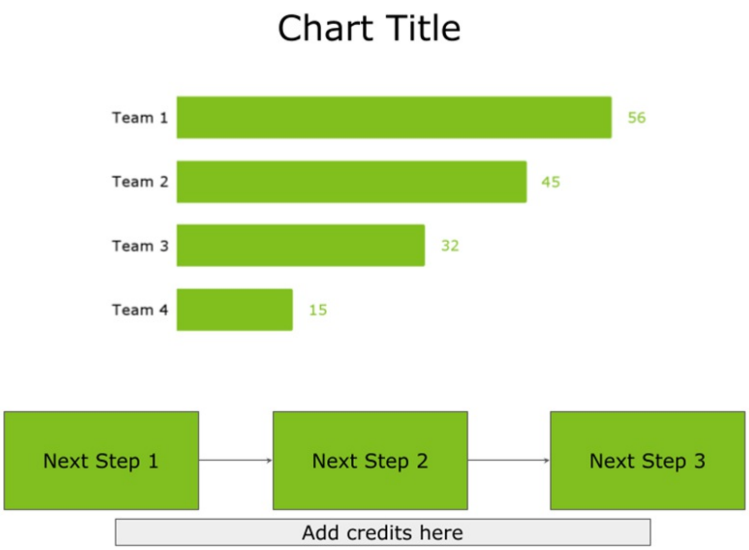


Use this placement if you want to reinforce the main message of your chart.

10.2.3 Under the Next Steps

In this case, credit the sources at the end of your story, as an appendix to the next steps, as shown in Figure 10.8.

Figure 10.8 An example of a data story with credits under the next steps

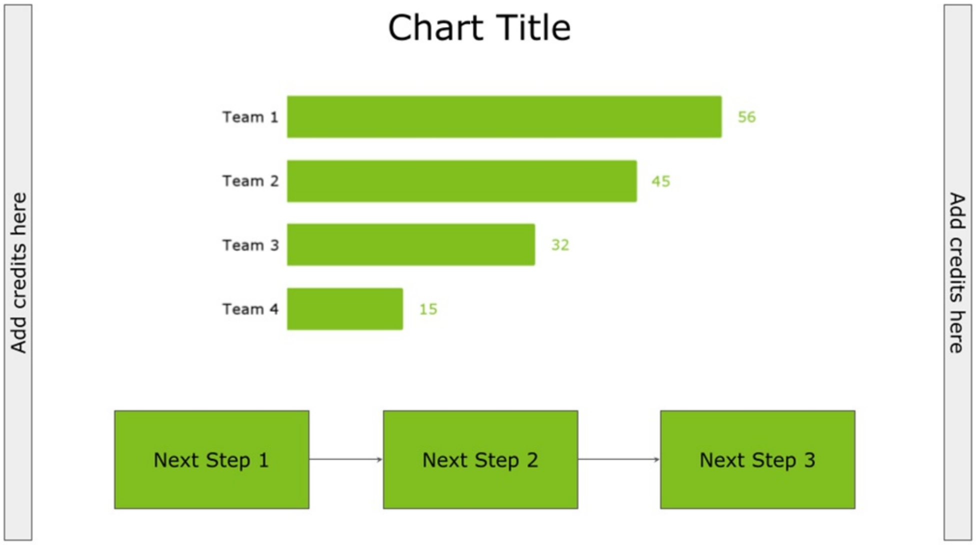


Use this placement if you want to reinforce the next steps of your story.

10.2.4 Sideways

Placing credits sideways means considering them as external to the main data story workflow. You can place credits either on the left or on the right, as shown in Figure 10.9.

Figure 10.9 An example of a data story with credits on the left and the right



Use this placement to keep credits external to your main data story workflow and keep the audience concentrated on the story.

Now that we’ve considered the various places you can place credits, let’s move on to how to implement credits practically.

10.2.5 Implementing the Credits in Altair

To implement credits in Altair, use `mark_text()` with a smaller font than the one you used for the main story. Optionally, you can include a hyperlink to the original source.

Consider the case study described in Chapter 4, related to sports disciplines to train in, and already show in Figure 10.4. Let’s credit DALL-E for images and ChatGPT for the title. Add the following text as credits: *Images: source DALL-E, Title: source ChatGPT.*

We will place credits on the left side. You can find the implemented code in the GitHub repository of the book, under 10/crediting/left-sideways.

Listing 10.1 shows the code to implement the credits section on the left.

Listing 10.1 Adding the credits on the left

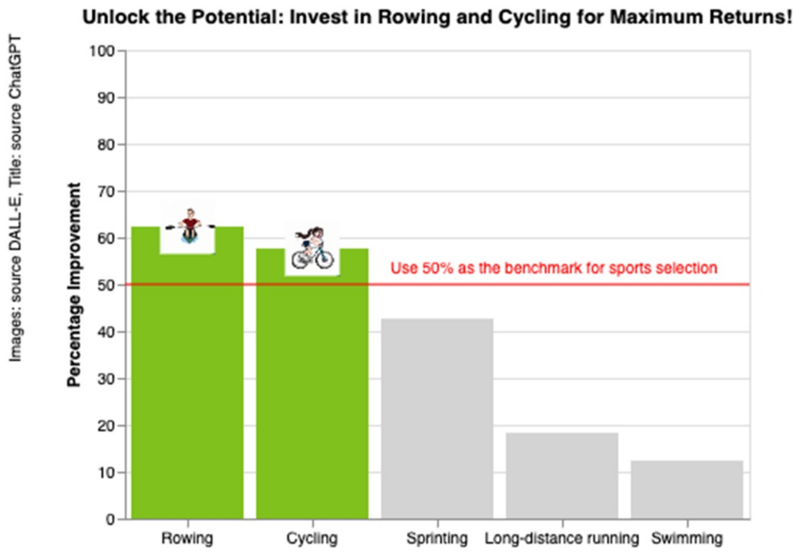
```
credits_df = pd.DataFrame({'text': ['Images: source DALL-E, Title
credits = alt.Chart(credits_df
        ).mark_text(
            size=10,
            align='left',
            color='black',
            angle=270,
            x=-70,
            y=200

        ).encode(
            text='text'
        )

chart = (credits + chart + annotation)
```

How to add the credits on the left

Use `mark_text()` to add credits. Use the `angle` attribute to rotate the text of 270 degrees. Adapt x and y to the chart. Try different values to obtain the best result.

As usual, you can ask Copilot to generate the code for you. Figure 10.10 shows the resulting chart.

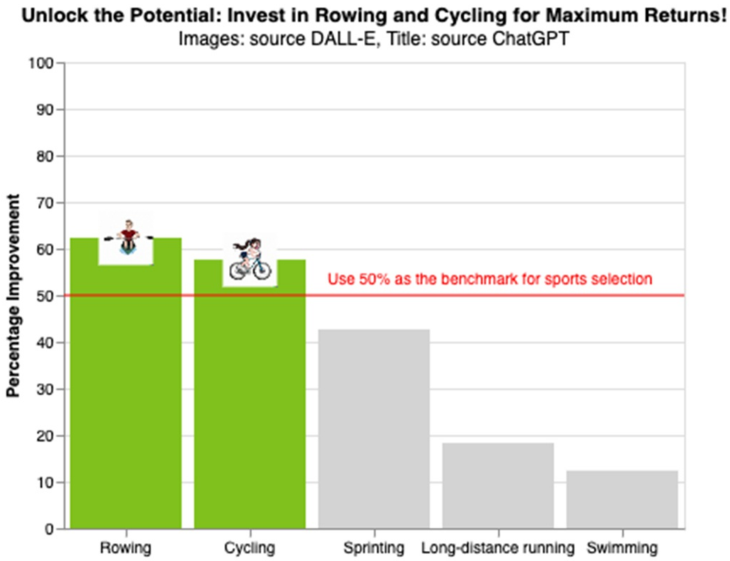
Figure 10.10 The case study described in Chapter 4, the sports disciplines with credits on the left



Exercise

Modify the previous example by implementing credits under the title, as shown in Figure 10.11.

Figure 10.11 The case study described in Chapter 4, the sports disciplines with credits under the title



You can find the solution to this exercise in the GitHub repository of the book, under 10/crediting/under-the-title.

Challenge 5

Compare Figures 10.10 and 10.11. Can you distinguish any difference in terms of the reading flow?

So far, you have learned how to credit sources in your data story. Now it’s time to see how to export the final story.

10.3 Exporting the Story

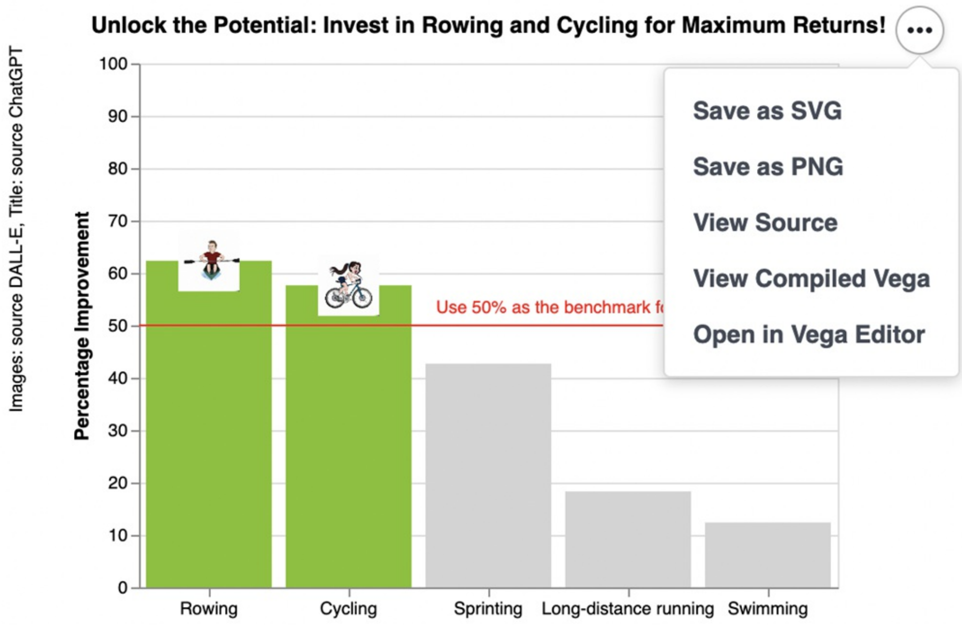
Altair provides different formats to export your data story. Throughout this book, we have used `chart.save('chart.html')`. Anyway, Altair also supports other formats, including:

- JPEG
- PNG
- SVG
- JSON
- PDF

To save the chart into a specific format, simply add the format extension to your file, such as `chart.save('chart.png')` for PNG files. For PNG files, you can also specify the pixels per inch (ppi) through an additional parameter: `chart.save('chart.png', ppi=300)`

You may wonder why throughout this book, we have used the HTML format and not PNG or SVG. The main reason is that HTML is the only format among the supported ones that shows interactivity. In addition, once you have created the HTML file, Altair also provides a default action menu on the top right part of the chart, enabling you to download the story into another format, as shown in Figure 10.12.

Figure 10.12 The top right menu in the HTML file enables you to export the story into another format



If you prefer not to show the action menu, you can disable it, as shown in Listing 10.2.

Listing 10.2 Disabling the action menu

```
chart.save('chart.html', embed_options={'actions': False})
```

Your journey from transforming raw data into a data story through the DIKW pyramid, Generative AI, and Altair is completed. Your story is finally ready to be disseminated to your audience!

In the remaining two sections, we will see some alternative techniques to publish your story. Feel free to skip them if you are not interested. First, we will see how to publish the story using Streamlit, and next, we will see how to embed our story into some popular tools.

Let’s start with Streamlit.

10.4 Publishing the Story over the Web: Streamlit

Streamlit is an open-source framework enabling you to build web applications quickly using Python. To start with Streamlit, refer to Appendix A. The main advantage of using Streamlit compared to other options is that it is fully integrated with Altair.

In this section, we'll see how to publish a data story using Streamlit. Streamlit uses specific functions to show content on a web page, such as `st.Title()` to set the page title.

Consider the example shown in Listing 10.3. Start by importing the Streamlit library. Then, build your chart as you usually do in Altair. To add the Altair chart wherever on the web page, use `st.altair_chart()`.

Listing 10.3 Building the app in Streamlit

```
import streamlit as st
import pandas as pd
import altair as alt

# Read the data
df = pd.read_csv('../data/population.csv')
df = df.melt(id_vars='Country Name', var_name='Year', value_name=
df['Year'] = df['Year'].astype('int')

continents = ['Africa Eastern and Southern', 'Africa Western and
              'Middle East & North Africa', 'Sub-Saharan Africa',
              'Europe & Central Asia', 'Latin America & Caribbean
              'North America', 'East Asia & Pacific']
```

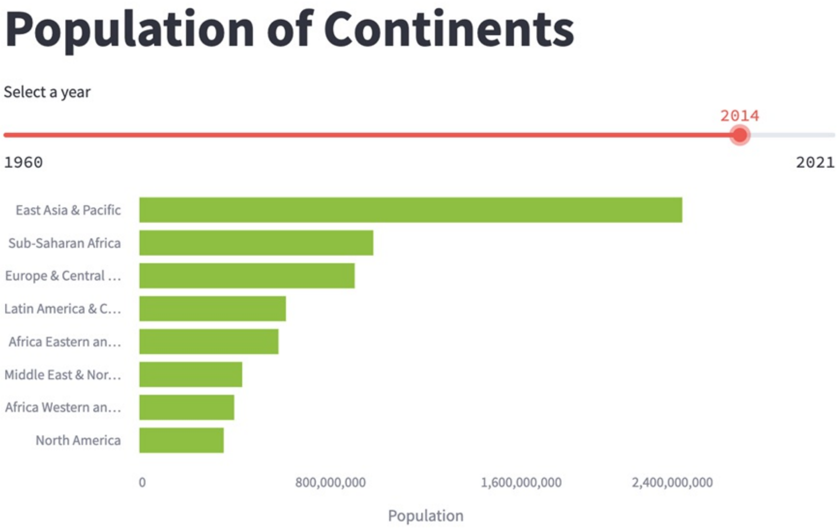
```
df = df[df['Country Name'].isin(continents)]

# Create Streamlit app
st.title('Population of Continents')

# Add a slider for year selection
selected_year = st.slider('Select a year', min_value=1960, max_va

# Filter data based on selected year
filtered_df = df[df['Year'] == selected_year]

# Create Altair chart
chart = alt.Chart(filtered_df).mark_bar(color='#80C11E').encode(
    y=alt.Y('Country Name:O', sort='-x', title=''),
    x=alt.X('Population:Q', title='Population')
).properties(
    width=600,
    height=300
).configure_view(
    strokeWidth=0
).configure_axis(
    grid=False
)

# Display chart in Streamlit app
st.altair_chart(chart)
```

How to build a web app in Streamlit

Start by importing the required libraries. Then, load the data as a Pandas DataFrame. Next, build the chart in Altair as you usually do. Use `st.slider()` to build a slider in Streamlit. Finally, display the chart in the Streamlit app using `st.altair_chart()`

To launch the app, run the following command in a terminal: streamlit app.py, with app.py the name of the script containing the code shown in Listing 10.3. Figure 10.13 shows the resulting app.

Figure 10.13 The resulting app in Streamlit



You can download the code described in this section from the GitHub repository of the book, under 10/streamlit/app.py.

Now that you have learned how to publish your data story in Streamlit, let’s move on to some alternative ways to publish the story. If you are not interested in them, feel free to skip directly to the conclusions.

10.5 Alternative Ways to Publish the Story: Tableau, Power BI, and Comet

There are different ways to publish your data story, including reports, presentations, HTML pages and so on. In this section, we will see three tools:

- Tableau
- Power BI
- Comet.

Let’s start with the first one, Tableau.

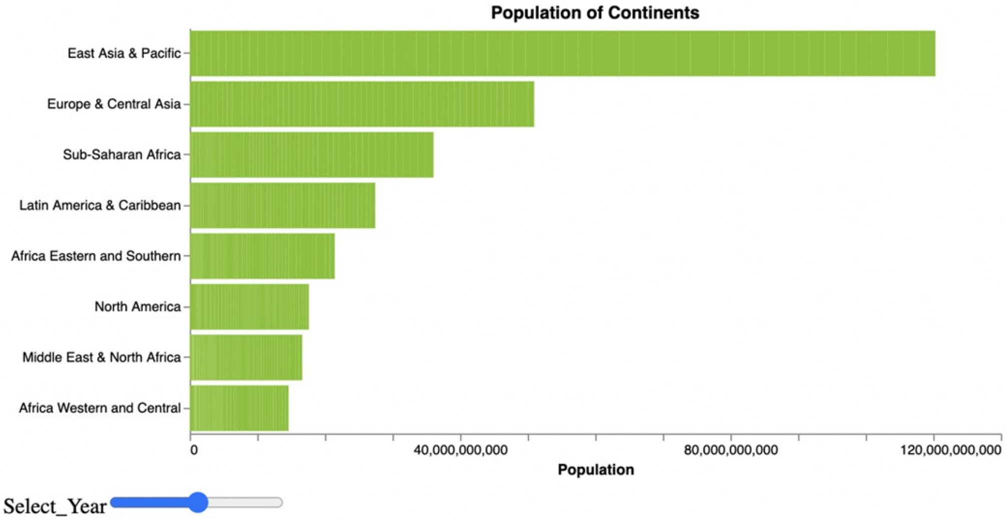
10.5.1 Tableau

Tableau is a business intelligence tool that helps users to explore and analyze data. Different from Streamlit, Tableau does not require any programming skills, since its interface is completely visual. Combining Tableau and Altair may be useful if you already know Tableau and use it to implement your dashboards. You can integrate your data story produced in Altair into Tableau as an additional item of your dashboard.

Before using Tableau, you must download it on your local computer and pay for a license. Tableau also offers a trial version and a completely free license for students and teachers. In the bibliography of this chapter, you can find some interesting resources to get started with Tableau. In this section, we describe how to import your data story in Tableau. If you are not interested, feel free to skip this section and move to the next one.

Consider the example in Chapter 08/slider.py and shown in Figure 10.14.

Figure 10.14 A chart with a slider

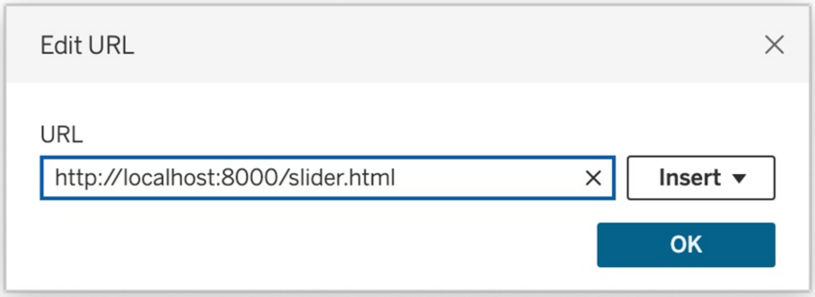


To import your chart in Tableau, execute the following operations:

1. Open a terminal and point it to the directory containing the 08/slider.py. Run the command python3 slider.py to generate the HTML file.
2. In the terminal, launch a Python server as follows: python3 -m http.server
3. Point to localhost:8000 in your browser and access slider.html. You should see the chart shown in Figure 10.14. Move the slider to check if

   the code is working correctly.
4. Start your Tableau Desktop application, import the 08/data/population.csv file and create a new dashboard.
5. In Tableau, double-click on Web Page from the Object menu and enter the following URL in the dialog box: http://localhost:8000/slider.html, as shown in Figure 10.15

Figure 10.15 The Edit URL dialog box in Tableau



URL: http://localhost:8000/slider.html

Now you should see your Altair chart in your Tableau Dashboard.

Note

In some cases, the Web Page importer fails because it does not support the Javascript wrapper generated by Altair to include the chart. In this case, export the chart as a .png file and use the Image object to import it into Tableau.

Now that you have learned how to include your data story in a Tableau dashboard, let’s move on to the next alternative, including your data story in a Power BI report.

10.5.2 Power BI

[Power BI](#) is another business intelligence tool released by Microsoft. Power BI offers both a desktop and online version. Like Tableau, you must buy a license to use it. However, you can test using a trial version. In the bibliography of this chapter, you can use some resources to start with Power BI.

Similar to Tableau, you may need to integrate your data story produced in Altair into a Power BI dashboard if you already know and use Power BI. Otherwise, it is better to use other solutions like Streamlit.

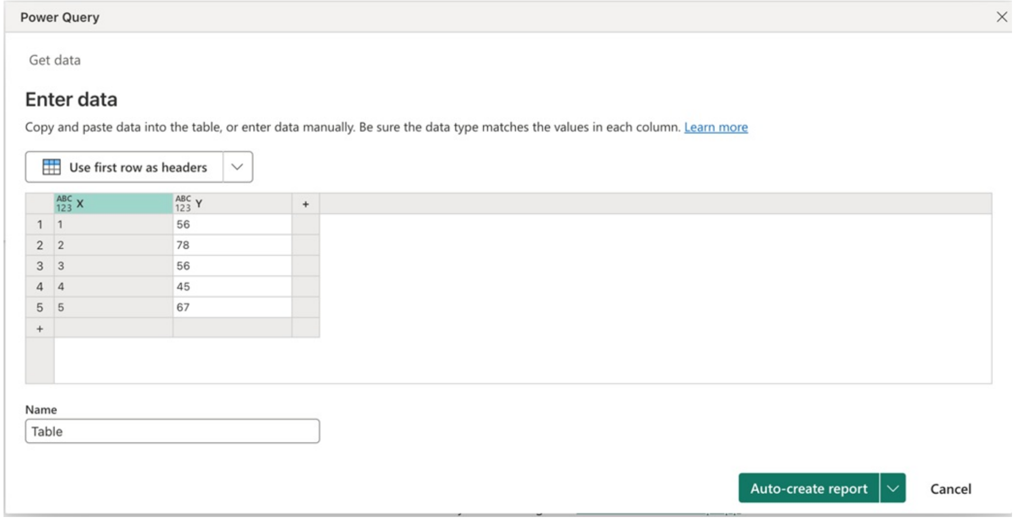
In this section, we only describe how to import a chart generated in Altair to Power BI. We will use the online version. Specifically, we will use the Deneb plugin, which enables you to import data written in Vega. From Altair, you can always export the Vega or Vega-lite JSON file directly from the HTML version, as shown in Figure 10.12.

For simplicity, in the following example, we will directly use the Vega code, available in the GitHub repository of the book, under 03/vega/json/spec.json.

Execute the following steps:

1. Login to your Power BI account and click on New Report → Paste or manually enter your data → create a blank report (Figure 10.16)

Figure 10.16 A snapshot of the Power BI interface to create a new report



Enter data

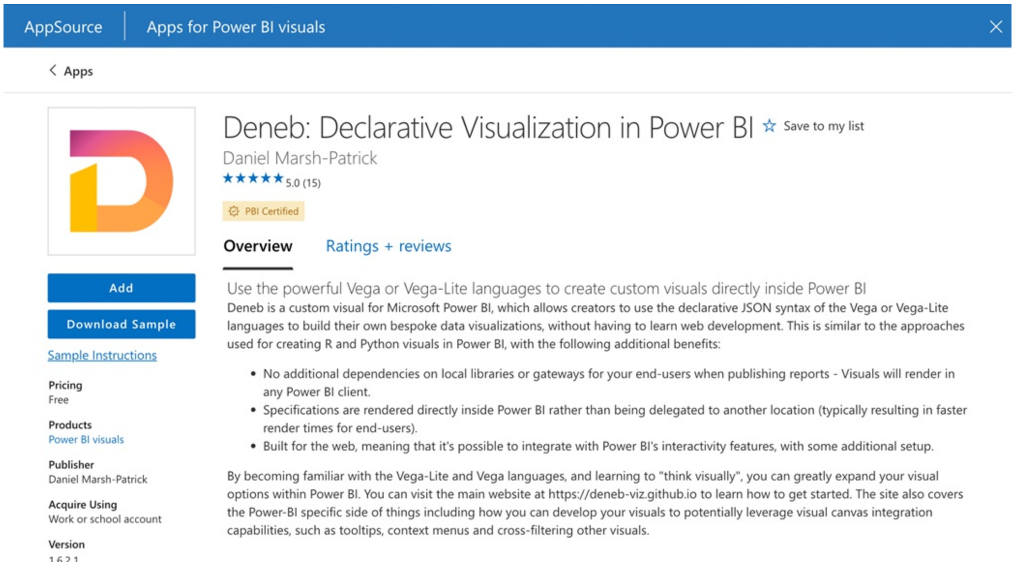
Copy and paste data into the table, or enter data manually. Be sure the data type matches the values in each column. Learn more

	X	Y
1	1	56
2	2	78
3	3	56
4	4	45
5	5	67

Name: Table

2. Click on Visualizations → (three dots) Get more visuals → Vega → Deneb: Declarative Visualization in Power BI → Add (Figure 10.17)

Figure 10.17 The Deneb plugin



AppSource | Apps for Power BI visuals

< Apps

Deneb: Declarative Visualization in Power BI ☆ Save to my list

Daniel Marsh-Patrick

★★★★★ 5.0 (15)

PBI Certified

Overview | Ratings + reviews

Add

Download Sample

Sample Instructions

Pricing
Free

Products
Power BI visuals

Publisher
Daniel Marsh-Patrick

Acquire Using
Work or school account

Version

Use the powerful Vega or Vega-Lite languages to create custom visuals directly inside Power BI

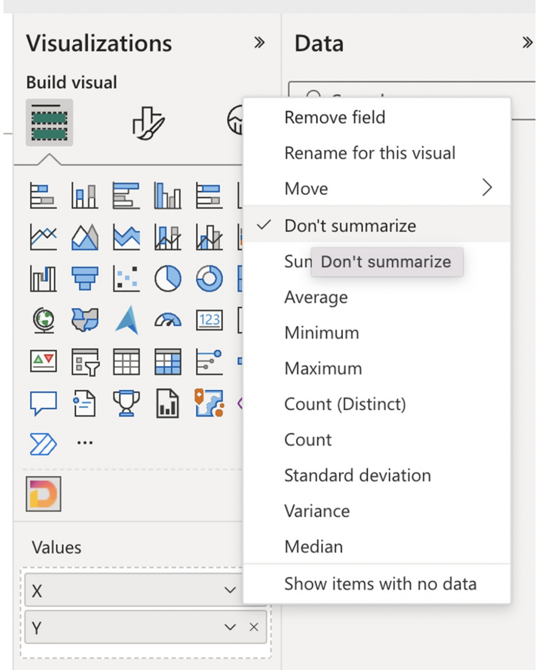
Deneb is a custom visual for Microsoft Power BI, which allows creators to use the declarative JSON syntax of the Vega or Vega-Lite languages to build their own bespoke data visualizations, without having to learn web development. This is similar to the approaches used for creating R and Python visuals in Power BI, with the following additional benefits:

- No additional dependencies on local libraries or gateways for your end-users when publishing reports - Visuals will render in any Power BI client.
- Specifications are rendered directly inside Power BI rather than being delegated to another location (typically resulting in faster render times for end-users).
- Built for the web, meaning that it's possible to integrate with Power BI's interactivity features, with some additional setup.

By becoming familiar with the Vega-Lite and Vega languages, and learning to "think visually", you can greatly expand your visual options within Power BI. You can visit the main website at https://deneb-viz.github.io to learn how to get started. The site also covers the Power-BI specific side of things including how you can develop your visuals to potentially leverage visual canvas integration capabilities, such as tooltips, context menus and cross-filtering other visuals.

3. In the visualization menu, select Deneb,
4. In the Data menu, add X and Y to Deneb. Click on the arrow near X and Y and click on Don't summarize (Figure 10.18)

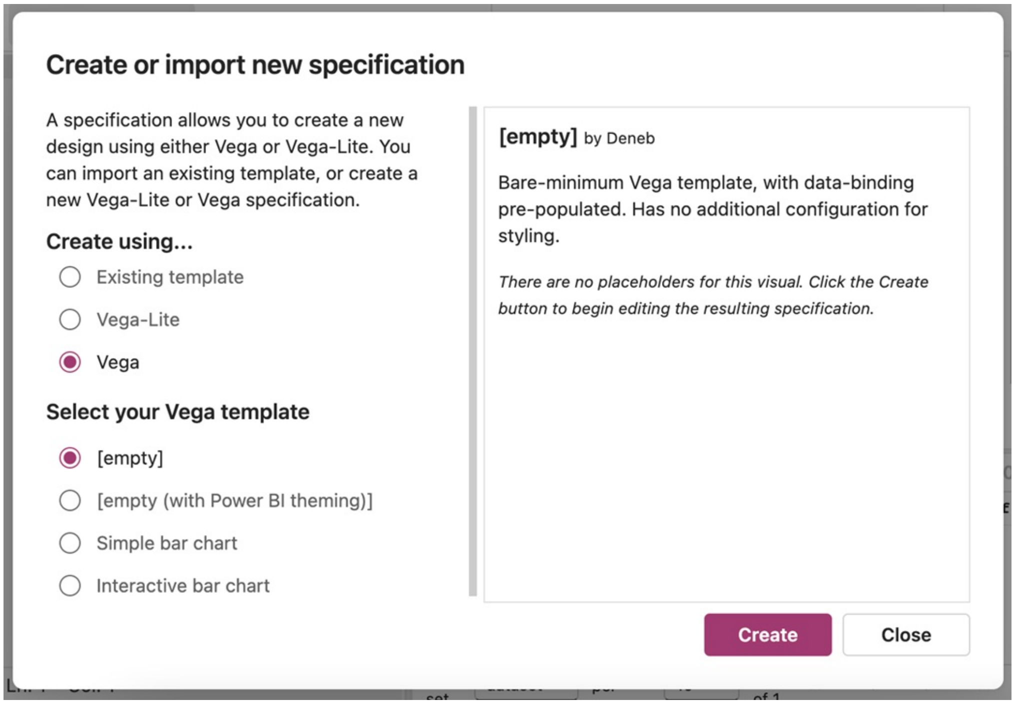
Figure 10.18 The selection menu for each variable



5. On the left of the app, you should see the Deneb box. Click on the three

dots in the Deneb box in the report, and edit → Vega - empty (Figure 10.19)

Figure 10.19 The selection menu for each variable



Create or import new specification

A specification allows you to create a new design using either Vega or Vega-Lite. You can import an existing template, or create a new Vega-Lite or Vega specification.

Create using...

○ Existing template
○ Vega-Lite
● Vega

Select your Vega template

● [empty]
○ [empty (with Power BI theming)]
○ Simple bar chart
○ Interactive bar chart

[empty] by Deneb

Bare-minimum Vega template, with data-binding pre-populated. Has no additional configuration for styling.

There are no placeholders for this visual. Click the Create button to begin editing the resulting specification.

6. Copy the code in 03/vega/json/spec.json and paste it into the Deneb editor. You need to remove data to make it work. Listing 10.4 shows the modified version. Next click on Create.

Listing 10.4 The Vega representation of the chart

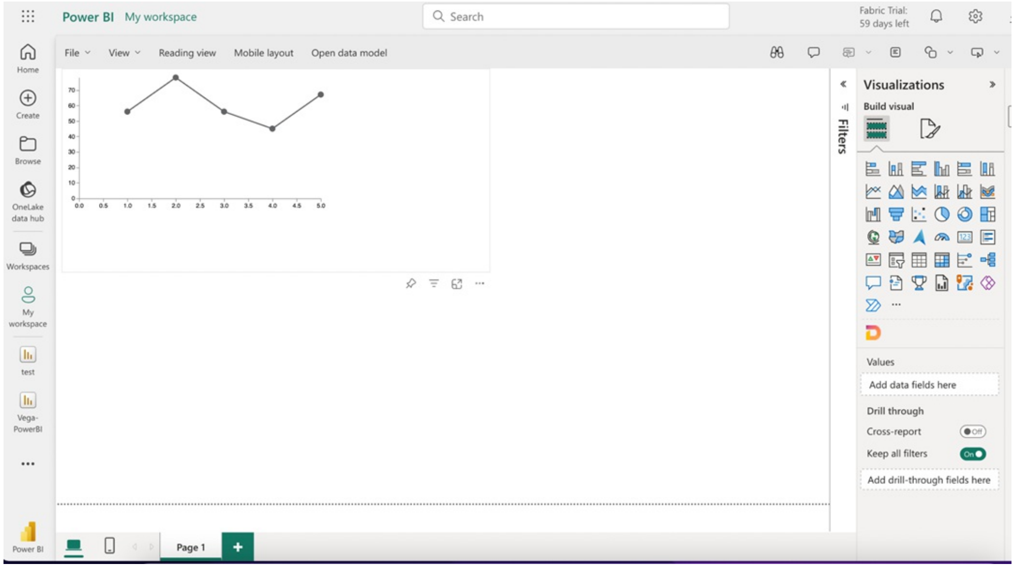
```
{
  "description": "A basic line chart",
  "width": 400,
  "height": 200,
  "padding": 5,
  "data": [
    {
      "name": "dataset"
    }
  ],

  "scales": [
    {
      "name": "xscale",
      "domain": {"data": "dataset", "field": "X"},
      "range": "width"
    },
    {
      "name": "yscale",
      "domain": {"data": "dataset", "field": "Y"},
      "range": "height"
    }
  ],

  "axes": [
    { "orient": "bottom", "scale": "xscale" },
    { "orient": "left", "scale": "yscale" }
  ],

  "marks": [
    {
      "type": "line",
      "from": {"data":"dataset"},
      "encode": {
        "enter": {
          "x": {"scale": "xscale", "field": "X"},
          "y": {"scale": "yscale", "field": "Y"},
          "stroke": { "value": "#636466"}

        }
      }
    },
    {
      "type": "symbol",
      "from": {"data":"dataset"},
      "encode": {
        "enter": {
          "x": {"scale": "xscale", "field": "X"},
          "y": {"scale": "yscale", "field": "Y"},
          "shape": {"value": "circle"},
          "size" : {"value": 100},
          "fill": { "value": "#636466"}
        }
      }
    }
  ]
  }
```

The Vega JSON file

Replace the data section with a link to the dataset imported in Power BI.

Figure 10.20 shows the resulting chart.

Figure 10.20 The resulting chart in Power BI



Now that you have learned how to import a Vega chart in Power BI, let’s move on to the final tool, Comet.

10.5.3 Comet

Comet is an experimentation platform for machine learning model tracking and testing. In my previous book, Comet for Data Science, I thoroughly described how to start with Comet, and how to track machine learning, deep learning, time series, and natural language processing tasks. So, for more details about Comet, please refer to this book.

You may decide to integrate your data story produced in Altair with Comet if your data storytelling story is the result of a complete data science project, which includes model experimentations and evaluations. Since Comet supports Python, using it to export your data story does not require any particular effort.

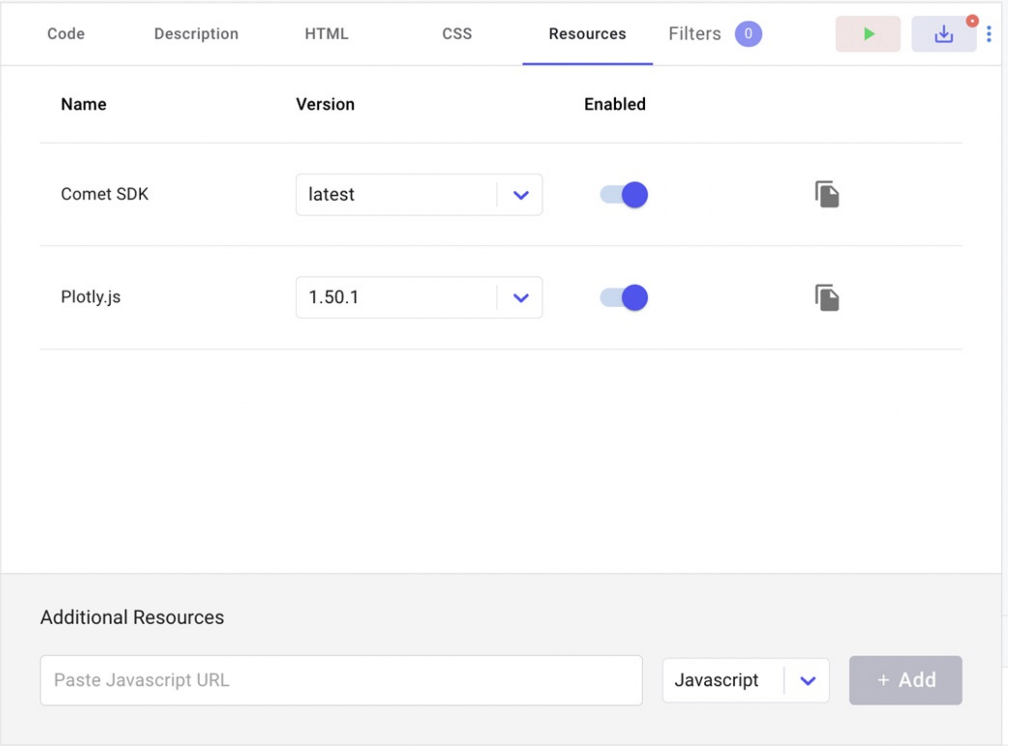
In Comet, you can also create reports. A report is a collection of text and panels showing the results of your experiments. This is a very helpful feature because you can use just one tool for experiments and reporting. Unlike Tableau and Power BI, Comet is free for personal use, so you don't need to pay to access the service. In addition, Comet supports many programming languages, including Python, so you can continue using Python for your reports.

In this section, we will describe how to include a simple interactive chart in Comet. We will use the HTML code generated by 08/slider.py. We will write the code in HTML/Javascript, but Comet also supports Python. If you don't have the slider.html file, open a terminal, move to the 08 directory, and run the command `python slider.py`.

Execute the following commands:

1. log in to Comet and start a new project by following these steps: New Project → Project Name, Description, Project Type: Experiment Tracking
2. Create a new report by clicking Report → New Report → Add Panel → New
3. The Comet SDK should open. By default, it is configured to work in Javascript. If you want to work with Python, on the top right part, click on Javascript and select Python.
4. Under the **Resources** tab, add the following three Javascript scripts (Figure 10.21):
    a. https://cdn.jsdelivr.net/npm/vega@5
    b. https://cdn.jsdelivr.net/npm/vega-lite@5.8.0
    c. https://cdn.jsdelivr.net/npm/vega-embed@6

Figure 10.21 The Resources tab in the Comet SDK



Name	Version	Enabled
Comet SDK	latest	●
Plotly.js	1.50.1	●

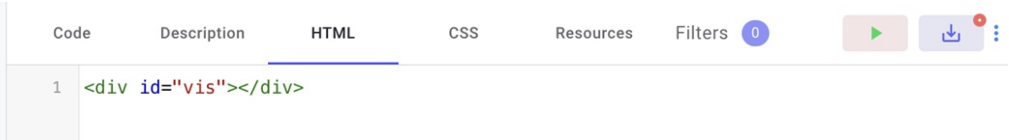
Additional Resources

Paste Javascript URL

Javascript

5. Switch to the **HTML** tab and change the id name to vis, as shown in Figure 10.22

Figure 10.22 The HTML tab in the Comet SDK



```
<div id="vis"></div>
```

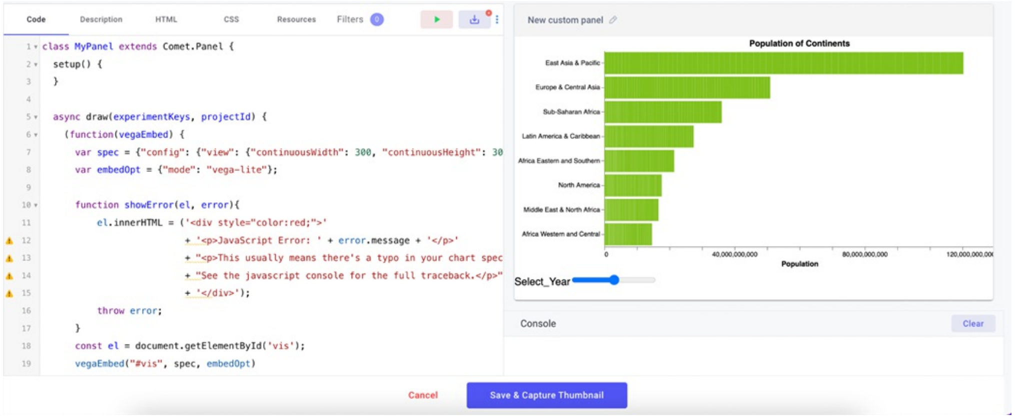
6. Open the slider.html file in edit mode and copy the code between the <script> and </scripts> tags (Listing 10.5). Don't worry if this code is very long, it contains all the Vega specs including the original dataset. Paste the copied code in the **Code** tab of the Comet SDK.

Listing 10.5 The Javascript code generating the chart

```
(function(vegaEmbed) {
      var spec = {"config": {"view": ...}};
      var embedOpt = {"mode": "vega-lite"};

      function showError(el, error){
          el.innerHTML = ('<div style="color:red;">'
                          + '<p>JavaScript Error: ' + error.messa
                          + "<p>This usually means there's a typo
                          + "See the javascript console for the f
                          + '</div>');
          throw error;
      }
      const el = document.getElementById('vis');
      vegaEmbed("#vis", spec, embedOpt)
        .catch(error => showError(el, error));
    })(vegaEmbed);
```

7. Click the run button (a green triangle). You should see the rendered chart on the right (Figure 10.23)

Figure 10.23 The rendered chart in the Comet SDK



```
class MyPanel extends Comet.Panel {
  setup() {
  }

  async draw(experimentKeys, projectId) {
    (function(vegaEmbed) {
      var spec = {"config": {"view": {"continuousWidth": 300, "continuousHeight": 30
      var embedOpt = {"mode": "vega-lite"};

      function showError(el, error){
          el.innerHTML = ('<div style="color:red;">'
                          + '<p>JavaScript Error: ' + error.message + '</p>'
                          + "<p>This usually means there's a typo in your chart spec
                          + "See the javascript console for the full traceback.</p>"
                          + '</div>');
          throw error;
      }
      const el = document.getElementById('vis');
      vegaEmbed("#vis", spec, embedOpt)
```

8. Click on Save & Capture Thumbnail. You should see the chart embedded in the Comet report.

Table 10.1 summarizes the pros and the cons for each analyzed tool.

Table 10.1 Pros and Cons of the analyzed tools

Tool Name	Description	Pros	Cons
Streamlit	A Python library to implement a web server	· Simple to use · Fully integrated with Altair · You use	· You must write your dashboard from scratch · You must

		Python to configure the interface · Free	have a machine to host the web server
Tableau	A software for data exploration and data analysis	· It does not require any programming skills · Fast dashboard building	· Not free · You must learn the basics to use it
Power BI	A software for data exploration and data analysis	· It does not require any programming skills · Fast dashboard building	· Not free · You must learn the basics to use it · It does not support Altair directly. You must use Vega instead
Comet	An experimentation platform for data science	· Free for personal usage · It enables you to track all your data science workflow, up to data storytelling · Fully integrated with Altair	· You must learn the basics to connect the data to a data visualization chart

In the first part of this chapter, you have learned the main issues behind generative AI and how to use Generative AI ethically. Next, we have seen how to credit sources properly, and how to publish the final data story using different tools: Streamlit, Tableau, Power BI, and Comet.

10.6 Final Thoughts

Our journey together has come to an end. Dearest reader, I thank you for sharing a stretch of the road with me, for trusting me, and for getting to the end of the reading. I hope that through this book you have learned to love data storytelling as an effective tool for communicating your data. I also hope that although the book is finished, your journey into the world of data storytelling can continue, reading other books, perhaps more interesting than this one, updating yourself with articles, courses, and anything else good you can find in the world. In the book, here and there I have indicated interesting resources that you can draw on, but do not limit your personal research.

Throughout the book, I have tried to equip you with the basic tools to undertake the difficult role of the data storyteller. The task of using them responsibly is now up to you. Always have the courage and strength to look for the truth behind the data, and after finding it, still have the desire to delve deeper into it and look for it again. I recommend only one thing: never forget to handle the data that you gradually tell with care, because always remember that behind the data, there are people waiting for someone to tell their story.

Dear reader, I wish you all the best and hope to meet you again in the future!

10.7 Summary

- Using Generative AI in any application may generate different issues, such as bias and discrimination. Thus, it is very important to review the content generated by Generative AI.
- Using Generative AI ethically means respecting people, society, and the environment, according to the UNESCO principles.
- Before publishing your data story, make sure to credit sources. It’s a way to recognize the work done by others and to generate a greater sense of trust in your audience
- You can embed your data story in different tools, such as Tableau, Power BI and Comet. Alternatively, you can publish your data story as a standalone web application using Streamlit.
- Use Tableau or Power BI to integrate your data story produced in Altair, if you already have some knowledge about these tools. This will enable you to enrich your dashboards quickly.
- Use Comet to integrate your data story produced in Altair, if the story is the result of a complete data science project, which included some

experimentations, such as model tracking and evaluation.Use Streamlit to export your data story produced in Altair if you need a web application to share it with your audience.

- In all cases, after completing your data story, remember to share it with your audience. And after having told the first story, begin to work on the second. And then on the third, and so on until you become an effective data storytelling. Happy work! Happy data storytelling!

10.8 References

10.8.1 Generative AI Issues

- Tobias Baer (2019) Understand, Manage, and Prevent Algorithmic Bias. Apress
- Jakub M. Tomczak (2022) Deep Generative Modeling. Springer

10.8.2 Ethics and AI

- EU AI Act https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:52021PC0206
- White House Executive Order on the Safe, Secure, and Trustworthy Development and Use of Artificial Intelligence https://www.whitehouse.gov/briefing-room/presidential-actions/2023/10/30/executive-order-on-the-safe-secure-and-trustworthy-development-and-use-of-artificial-intelligence/
- UNESCO Recommendation on the Ethics of Artificial Intelligence https://unesdoc.unesco.org/ark:/48223/pf0000381137

10.8.3 Power BI, Tableau, Comet

- Greg Deckler, Brett Powell (2022) Mastering Microsoft Power BI: Expert techniques to create interactive insights for effective data analytics and business intelligence, 2nd Edition, Packt Publishing
- Joshua N. Milligan (2022) Learning Tableau 2022: Create effective data visualizations, build interactive visual analytics, and improve your data storytelling capabilities, 5th Edition, Packt Publishing

- Angelica Lo Duca (2022) Comet for Data Science, Packt Publishing

[1] https://medium.com/towards-data-science/how-to-improve-your-chatgpt-outputs-using-configuration-parameters-0eebd575646e

Appendix A. Technical Requirements

This chapter covers

- Cloning the GitHub Repository of the book
- Installing the Python Packages
- Installing GitHub Copilot
- Configuring ChatGPT

This appendix describes how to install the software used in this book.

A.1 Cloning the GitHub Repository

To clone the GitHub repository of the book on your local machine, you can adopt one of the following two strategies:

- Using the terminal
- Using GitHub Desktop

A.1.1 Using the terminal

To clone the GitHub repository of the book from the command line, follow the steps described below:

1. Install the Git suite, if you do not have it yet
2. Open the terminal on your computer
3. Navigate to the directory where you want to clone the repository
4. Run the command `git clone https://github.com/alod83/Data-Storytelling-with-Python-Altair-and-Generative-AI`
5. Wait for the repository to be cloned to your local machine.

A.1.2 Using GitHub Desktop

To clone the GitHub repository of the book from GitHub Desktop, follow the steps described below:

1. Download and install GitHub Desktop from the official website
2. Launch GitHub Desktop
3. Sign in to your GitHub account or create a new account if needed
4. Click on the "File" menu and select "Clone Repository."
5. In the "Clone a repository" window, choose the "URL" tab.
6. Enter the repository URL https://github.com/alod83/Data-Storytelling-with-Python-Altair-and-Generative-AI in the "Repository URL" field
7. Choose the local path where you want to clone the repository
8. Click on the "Clone" button
9. Wait for GitHub Desktop to clone the repository to your local machine.

A.2 Installing the Python Packages

The examples illustrated in this book use Python 3.8. You can download it from the official website: https://www.python.org/downloads/ and choose the 3.8 version.

The examples described in this book use the following Python packages:

- altair
- jupyterlab 3.5.1
- ydata-profiling 4.0.0
- matplotlib 3.4.3
- numpy 1.19.5
- pandas 1.3.4

To install the above packages, run the following command in a terminal:

```
pip install <package_name> version==<version>.
```

A.3 Installing GitHub Copilot

To use GitHub Copilot, you must set up a free trial or subscription for your personal GitHub account. If you are a teacher or a student, you can set up a free subscription plan at the following link: https://education.github.com/discount_requests/pack_application.

Once your account can use GitHub Copilot, configure it as an extension of Visual Studio Code (VSC), a free, open-source code editor designed for developers to write and debug code. Download VSC from its official website: https://visualstudio.microsoft.com/it/downloads/.

To start with GitHub Copilot, open Visual Studio and navigate to the **Extensions** tab. Download and install the GitHub Copilot extension, and select **Connect to your account** from the dashboard. Enter your GitHub credentials. Once logged in, the extension will detect existing repositories and provide options for configuring new projects.

A.4 Configuring ChatGPT

To use ChatGPT, you must set up an account on the Open AI website. Then, you can use two ways to interact with ChatGPT:

- Web interface
- Open AI API.

In this book, we prefer using the Web interface since we don’t have any specific need to automatize the process. In addition, at the time of writing this book, the Web interface is free, while the Open AI API requires a paid account.

Anyway, there is a substantial difference between how the web interface and the Open AI API manage data. OpenAI uses the inputs provided to the ChatGPT web interface to train and improve the model while ensuring user confidentiality and protecting sensitive information. This means that the ChatGPT web interface uses your input data to improve the model. Thus, don’t use it if you have sensitive information. However, as of March 1st, 2023, OpenAI no longer uses these interactions to improve its models.

On the other hand, when using the OpenAI API, data sent via the API is subject to OpenAI's data usage policy. OpenAI retains the right to collect and store the data sent through the API, but they do not use the data to improve their models.

A.4.1 Web Interface

Point to https://chat.openai.com/, log in to your account, and start writing your prompts in the input text box like a live chat.

Whenever you want to start a new topic, create a new chat session by clicking on the top left button New Chat.

ChatGPT retains the entire history of prompts within a single chat session. This means you can write step-by-step instructions, building on the previous prompts and maintaining a coherent conversation.

The Web interface also provides a paid account that gives some additional features, such as the possibility to use advanced models. In this book, we use the free version of the web interface.

A.4.2 Open AI API

To use the Open AI API, you must upgrade your existing account to a paid one. Next, you can configure an API Key by performing the following steps:

1. Click on the Personal button on the top right of your dashboard
2. Select View API Keys in your dropdown menu
3. Click on Create new secret key
4. Insert the key name and create secret key
5. Click on the Copy symbol to copy the key value and paste it into a secure place
6. Click on Done.

Once created the API key, you must install the openai Python library by running the command pip install openai. Next, you can use the Open AI API in your Python scripts.

The following listing shows an example of usage of the Open AI API, asking ChatGPT to produce some output given an input prompt.

Listing A.1 Calling the ChatGPT API

```
import openai
openai.api_key = 'MY API KEY'
prompt = 'MY PROMPT'

messages = [{"role": "user", "content": prompt}]
response = openai.ChatCompletion.create(
    model="gpt-3.5-turbo",
    messages=messages,
    temperature=0, #A
)
output = response.choices[0].message["content"]
print(output)
```

Define a prompt Use the ChatCompletion.create() method to call the ChatGPT API.

A.5 Configuring DALL-E

To use DALL-E, you must set up an account on the Open AI website. Then, you can use two ways to interact with DALL-E:

- Web interface
- Open AI API.

As for ChatGPT, also for DALL-E, in this book, we prefer using the Web interface. The Open AI API requires a paid account, while the Web interface requires buying credits to generate images. In addition, if you created your DALL-E account before April 6, 2023, every month, you have a free pool of credits.

To use the Web interface, point to https://labs.openai.com/, log in to your account, and write your prompts in the input text box.

To use the Open AI API, you must upgrade your existing account to a paid one. Next, you can configure an API as described in the section related to ChatGPT:

Once created the API key, you must install the openai Python library by running the command pip install openai. Next, you can use the Open AI API in your Python scripts.

The following listing shows an example of usage of the Open AI API, asking DALL-E to produce an image given an input prompt.

Listing A.2 Calling the DALL-E API

```
import openai
openai.api_key = 'MY API KEY'
prompt = 'MY PROMPT'

response = openai.Image.create(
  prompt=prompt,
  n=1, #A
  size=’1024x1024’
)
output = response['data'][0]['url']
print(output)
```

Define a prompt Use the Image.create() method to call the DALL-E API. The output contains the URL to the produced image.

In addition, you can modify an existing image using the Image.create_edit() method. Refer to the [DALL-E official documentation]() for more details.

Appendix B. Python Pandas DataFrame

This chapter covers

- An overview of the Pandas DataFrame
- The DataFrame methods used in this book

This appendix describes an overview of the Pandas DataFrame and the methods used in this book.

B.1 An Overview of Pandas DataFrame

Python Pandas is a data manipulation, analysis, and visualization library. It provides easy tools to load, manipulate, analyze, and visualize data. In this book, we use the Pandas DataFrame, a two-dimensional structure composed of rows and columns. The DataFrame stores data in a tabular form, enabling you to manipulate, analyze, filter, and aggregate data quickly and easily.

There are different ways to create a Pandas DataFrame. In this book, we consider two ways: from a Python dictionary and from a CSV file. You can download the code described in this appendix from the GitHub repository of the book, under `AppendixB/Pandas DataFrame.ipynb`.

B.1.1 Building from a Dictionary

Consider the following example, which creates a Pandas DataFrame from a Python dictionary:

Listing B.1 Creating a DataFrame from a dictionary

```
import pandas as pd

data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'BirthDate': ['2000-01-30', '2001-02-03', '2001-04-05'],
    'MathsScore': [90, 85, None],
    'PhysicsScore': [87, 92, 89],
    'ChemistryScore': [92, None, 90],
    'Grade' : ['A', 'B', 'A']
} #A

df = pd.DataFrame(data) #B

df['BirthDate'] = pd.to_datetime(df['BirthDate'], format='%Y-%m-%
```

Pandas DataFrame

Use `DataFrame()` to create a new DataFrame from a dictionary.

B.1.2 Building from a CSV file

Use the `read_csv()` file to load a CSV file into a Pandas DataFrame, as shown in Listing B.2.

Listing B.2 Creating a DataFrame from a CSV file

```
import pandas as pd

df = pd.read_csv('data.csv')
```

Pandas read_csv()

Use `read_csv()` to create a new DataFrame from a CSV file.

Now that you have seen how to create a Pandas DataFrame, let us describe the main DataFrame functions used in this book.

B.2 dt

The `dt` variable within a Pandas Dataframe enables you to access the Python’s built-in DateTime library. Use it to store and manipulate DateTime values, such as the year, month, day, hour, minute, and second. Consider the following example that extracts the year from a DateTime column:

Listing B.3 How to use Pandas dt

```
import pandas as pd

data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'BirthDate': ['2000-01-30', '2001-02-03', '2001-04-05'],
    'MathsScore': [90, 85, None],
    'PhysicsScore': [87, 92, 89],
    'ChemistryScore': [92, None, 90],
    'Grade' : ['A', 'B', 'A']
}

df = pd.DataFrame(data)
df['BirthDate'] = pd.to_datetime(df['BirthDate'], format='%Y-%m-%

year = df['BirthDate'].dt.year #A
month = df['BirthDate'].dt.month #B
day = df['BirthDate'].dt.day #C
weekOfYear = df['BirthDate'].dt.isocalendar().week #D
```

Pandas dt

Use Pandas `dt` to access the DateTime functions of Python’s DateTime library.

B.3 groupby()

The Pandas `groupby()` method splits data into groups based on the values of certain columns. This process often involves creating an aggregate statistic for each group, such as a sum or mean. Consider the following example:

Listing B.4 How to use Pandas groupby

```
import pandas as pd

data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'BirthDate': ['2000-01-30', '2001-02-03', '2001-04-05'],
    'MathsScore': [90, 85, None],
    'PhysicsScore': [87, 92, 89],
    'ChemistryScore': [92, None, 90],
    'Grade' : ['A', 'B', 'A']
}

df = pd.DataFrame(data)

df_grouped = df.groupby(by='Grade').mean().reset_index()
```

Pandas dt

Use Pandas `groupby` to group by instrument and calculate the average score by grad. Use the `reset_index()` method, to restore the indexer column (`Grade` in the example).

The following table shows the result:

Table B.1 The result of groupby() in Listing B.4

Grade	MathsScore	PhysicsScore	ChemistryScore
A	90.0	88.0	91.0
B	85.0	92.0	

B.4 isnull()

The Pandas DataFrame `isnull()` method returns a new boolean DataFrame indicating which values in the DataFrame are null (`NaN`). Use this method to detect missing values in a DataFrame.

Consider the following example:

Listing B.5 How to use Pandas isnull()

```
import pandas as pd
```

```
data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'BirthDate': ['2000-01-30', '2001-02-03', '2001-04-05'],
    'MathsScore': [90, 85, None],
    'PhysicsScore': [87, 92, 89],
    'ChemistryScore': [92, None, 90],
    'Grade' : ['A', 'B', 'A']
}

df = pd.DataFrame(data)

df_isnull = df.isnull()
```

Pandas isnull()

Use Pandas `isnull()` to check if a DataFrame contains missing values. You can apply the isnull() method also to a single column (e.g., `df['ChemistryScore'].isnull()`)

The following table shows the result, a boolean DataFrame indicating which values in the DataFrame are null (`NaN`):

Table B.2 The result of isnull() in Listing B.5

Name	BirthDate	MathsScore	PhysicsScore	ChemistryScore	Grade
False	False	False	False	False	False
False	False	False	False	True	False
False	False	True	False	False	False

B.5 melt()

We use the Pandas `melt()` function to reshape data by turning columns into rows. This function unpivots a DataFrame from wide to long format, optionally leaving identifiers set. Consider the following example:

Listing B.6 How to use Pandas melt()

```
import pandas as pd

data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'BirthDate': ['2000-01-30', '2001-02-03', '2001-04-05'],
    'MathsScore': [90, 85, None],
    'PhysicsScore': [87, 92, 89],
    'ChemistryScore': [92, None, 90],
    'Grade' : ['A', 'B', 'A']
}

df = pd.DataFrame(data)

df_melted = df.melt(id_vars='Name',
        var_name='Subject',
        value_name='Score',
        value_vars=['MathsScore', 'PhysicsScore', 'ChemistryScore
)
```

Pandas melt()

Use Pandas `melt()` to transform the DataFrame from wide to long format. Set the `id_vars` argument to specify which variables to keep as an identifier and the `var_name` and `value_name` arguments to set the column names for the new variables in the resulting melted DataFrame. Use `value_vars` to select the columns to group.

The following table shows the result of melting the data used in Listing B.6:

Table B.3 The result of the melting operation described in Listing B.6

Name	Subject	Score

Alice	MathsScore	90.0
Bob	MathsScore	85.0
Charlie	MathsScore	Nan
Alice	PhysicsScore	87.0
Bob	PhysicsScore	92.0
Charlie	PhysicsScore	89.0
Alice	ChemistryScore	92.0
Bob	ChemistryScore	Nan

B.6 unique()

We use the Pandas `unique()` method to obtain distinct values from a specific column in a DataFrame. This method returns an array-like object containing the unique values found in the specified column.

Consider the following example

Listing B.7 How to use Pandas unique()

```
import pandas as pd

data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'BirthDate': ['2000-01-30', '2001-02-03', '2001-04-05'],
    'MathsScore': [90, 85, None],
    'PhysicsScore': [87, 92, 89],
    'ChemistryScore': [92, None, 90],
    'Grade' : ['A', 'B', 'A']
}

df = pd.DataFrame(data)
unique_grades = df['Grade'].unique()
```

Pandas unique()

Use Pandas `unique()` to get the unique values of a column.

The following listing shows the result of calculating the unique values for the column Grade (Listing B.7):

Listing B.8 The results of Pandas unique()

```
array(['A', 'B'], dtype=object)
```

The output of unique()

.The method returns an array containing the unique values for a column.