O'REILLY®

# Deep Learning for Finance

## Creating Machine & Deep Learning Models for Trading in Python

Early Release

RAW & UNEDITED

Sofien Kaabar

# Deep Learning for Finance

## Creating Machine and Deep Learning Models for Trading in Python

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

**Sofien Kaabar**

## Deep Learning for Finance

by Sofien Kaabar



Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (http://oreilly.com). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Acquisitions Editor: Michelle Smith

Development Editor: Corbin Collins

Production Editor: Elizabeth Faerm

Copyeditor: TO COME

Proofreader: TO COME

Indexer: TO COME

Interior Designer: David Futato

Cover Designer: Karen Montgomery

Illustrator: Kate Dullea

April 2024: First Edition

## Revision History for the Early Release

- 2023-06-09: First Release

See http://oreilly.com/catalog/errata.csp?isbn=9781098148393 for release

details.



978-1-098-14833-1

[LSI]

# Chapter 1. Introducing Data Science and Trading

> **A NOTE FOR EARLY RELEASE READERS**
>
> With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.
>
> This will be the 1st chapter of the final book. Please note that the GitHub repo will be made active later on.
>
> If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at *ccollins@oreilly.com*.

The best way to begin learning about complex topics is to slowly build up momentum until you start completing the puzzle. Understanding deep learning for finance requires a certain knowledge in basic and intermediate data science topics as well as financial markets and their structure.

This chapter lays the building blocks needed to have a thorough understanding of data science and its uses, but also of financial markets and how trading and forecasting can benefit from data science.

By the end of the chapter, you should know what data science is, what its applications are, and how you can use it in finance to extract value.

## Understanding Data

It is impossible to understand the field of data science without understanding the types and structures of data first. After all, the first word for the name of this immense field is *data*. So what is data? And more importantly, what can you do with it?

*Data* in its simplest and purest form is a collection of raw information that can be of any type (numerical, text, boolean, etc.).

The final aim of collecting data is decision-making. This is done through a complex process which ranges from the act of gathering and processing data to interpreting it and using the results to make a decision.

Let's take an example of using data to make a decision. Suppose you have a portfolio composed of five different equal-weighted dividend-paying stocks as detailed in table 1-1.

*Table 1-1. Dividend table*

| Stock | Dividend yield |
|---|---|
| A | 5.20% |
| B | 3.99% |
| C | 4.12% |
| D | 6.94% |
| E | 5.55% |

> **NOTE**
>
> A *dividend* is the payment made to shareholders from a company's profits. The *dividend yield* is the amount distributed in monetary units over the current share price of the company.

Analyzing this data can help you understand the average dividend yield you are receiving from your portfolio. The average is basically the sum divided by the quantity, and it gives a quick snapshot of the overall dividend yield of the portfolio:

$$Average\ dividend\ yield = \frac{5.20\%+3.99\%+4.12\%+6.94\%+5.55\%}{5} = 5.16\%$$

Therefore, the average dividend yield of your portfolio is 5.16%. This

information can help you compare your average dividend yield to other portfolios so that you know if you have to make any adjustments.

Another metric you can calculate is the number of stocks held in the portfolio. This may provide a first information brick in constructing a wall of diversification. Even though these two pieces of information (average dividend yield and the number of stocks in the portfolio) are very simple, complex data analysis begins with simple metrics and may sometimes not require sophisticated models to properly interpret the on-going events.

The two metrics you have calculated in the previous example are called the *average* (or mean) and the *count* (or number of elements). They are also part of a field called *descriptive statistics* which is also itself part of data science.

Let’s take another example of data analysis for inferential purposes. Suppose you have calculated a yearly correlation measure between two currency pairs and you want to predict whether the next yearly correlation will be positive or negative. Table 1-2 has the details of the calculations.

*Table 1-2. Correlation table*

| Year | Correlation |
|---|---|
| 2015 | Positive |
| 2016 | Positive |
| 2017 | Positive |
| 2018 | Negative |
| 2019 | Positive |
| 2020 | Positive |
| 2021 | Positive |
| 2022 | Positive |
| | |

| | |
|---|---|
| 2023 | Positive |

**NOTE**

*Correlation* is a measure of the linear reliance between two time series. A *positive correlation* generally means that the two time series move on average in the same direction, while a *negative correlation* generally means that the two time series move on average in opposite directions.

From table 1-2, the historical correlations between the two currency pairs were mostly positive with around 88% of the time. Taking into account historical observations, you can say that there is an 88% probability that the next correlation measure will be positive. This also means that there is a 12% probability that the next correlation measure will be negative:

$$E\left(Positive\ correlation\right) = \frac{8}{9} = 88.88\%$$

This is another simplistic example of how to use data to infer observations and make decisions. Of course, the assumption here is that historical results reflect exactly the future results, which is unlikely in real life, but occasionally, to predict the future, all you have is the past.

Now, before discussing data science, let's review what types of data there can be and segment them into different groups.

*Numerical data*

This type of data is composed of numbers that reflect a certain type of information that is collected at regular or irregular intervals. Examples can include market data (OHLC[1], volume, spread, etc.) and financial statements data (assets, revenue, costs, etc.).

*Categorical data*

Data that can be organized into groups or categories using names or labels. It is qualitative rather than quantitative. For example, the blood type of patients

is a type of categorical data.

*Text data*

Text data is on the rise during the recent years with the development of *natural language processing* (NLP). Machine learning models use text data to translate, interpret, and analyze the sentiment of the text. Furthermore, you can even use the models to create an algorithm that outputs structured paragraphs.

*Visual data*

Images and videos are also considered data, and you can process and transform them into valuable information. For example, a *convolutional neural network* (CNN) is a type of algorithm that can recognize and categorize photos by labels (for example, labeling cat photos as cats).

*Audio data*

Audio data is very valuable and can help save time on transcriptions. For example, you can use algorithms on audio to create captions and automatic subtitles. You can also create models that interpret the sentiment of the speaker using the tone and the volume.

You are likely to encounter the following data when dealing with Python:

*Integers*

These are whole numbers, which can be either positive or negative. Examples are −8 and 745. They are, however, limited to between −2147483648 and 2147483647.

*Floats*

These are real numbers with decimal points such as 18.54 and 311.52.

*Strings*

These are words stored in a variable. More scientifically, they are a set of structured characters (text). In Python, you write strings between single or double quotes.

*Boolean*

These are true or false statements to evaluate a condition.

*Data science* is an transdisciplinary field that tries to extract intelligence and conclusions from data using different techniques and models, be they simple or complex. The process of data science is composed of many phases besides to just analyzing data. The following summarizes the different stages of data science:

1. *Data gathering*: This process involves the acquisition of data from reliable and accurate sources. A widely known quote in computer science generally credited to George Fueschel goes as follows "*Garbage in, garbage out*", and it sums up the need to have quality data that you can rely on for proper analysis. Basically, if you have inaccurate or faulty data, then all your process would be invalid and it would be waste of time.
2. *Data preprocessing*: Occasionally, when you acquire data, it can be in a raw form that needs preparation for the data science models in the data analysis step to come. For example, dropping some unnecessary data, handling missing data, or eliminating invalid and duplicate data are part of the preprocessing phase. Other more complex examples can include *normalization* and *denoising* of data. The aim of this step is to get the data ready for analysis.
3. *Data exploration*: This is the first step in data analysis, and it is a basic

statistical exploration in order to find trends and different properties so that you have a preliminary idea on the expected behavior. One example is to check for data stationarity, a concept discussed in detail throughout the book.

4. *Data visualization*: This is an important step that is an add-on to the previous step. It includes creating visualizations such as histograms and heatmaps to help identify patterns and trends and make the interpretation easier.
5. *Data analysis*: This is the long-awaited step which is basically the main focus of the data science process. This is where you *fit* the data using different learning models so that they interpret and predict the future outcome based on the given parameters.
6. *Data interpretation*: This phase deals with the feedback and conclusions after the models have performed their jobs. *Optimization* may also be a part of this phase which then loops back to phase 5 in order to run the models again with the updated parameters before interpreting them again and evaluating the performance.

**NOTE**

To sum up the previous points, data science comprises many steps that start with acquiring the data through interpreting and optimizing the models that predict the future values of the data.

Let’s take a simple example in Python that applies the data science process discussed in the previous six steps. Suppose you want to analyze and predict the VIX, a volatility time series indicator published by the CBOE on a daily basis.

**NOTE**

There is a hidden step that I refer to as *step zero* which is the idea and the intuition for the whole process. You wouldn’t be applying the process if you didn’t have a motive first. For example, believing that inflation numbers may drive the returns of certain commodities is an idea and a motive to start exploring the data in search for real numbers that prove this hypothesis.

The first step is data gathering, which in this case can be automated using Python. The next code block connects to the website of the Federal Reserve of Saint Louis and downloads the historical data of the VIX between January 1990 and January 2023.

> **NOTE**
>
> The *VIX* stands for the *volatility index* and it represents the implied volatility of the S&P 500 index. It has been available since 1993 and is issued by the Chicago Board Options Exchange (CBOE).
>
> Because it is meant to measure the level of fear or uncertainty in the stock market, the VIX is frequently referred to as the *fear index*. The index is a percentage and is computed using the pricing of options on the S&P 500 index. A higher VIX value correlates with a greater market turbulence and uncertainty, whereas a lower value correlates with greater stability on average.

Note that Chapter 3 is entirely dedicated into introducing Python and harnessing its power. For the moment, you do not have to understand the code as it is not yet the learning outcome:

```
# Importing the required library
import pandas_datareader as pdr

# Setting the beginning and end of the historical data
start_date = '1990-01-01'
end_date   = '2023-01-23'

# Creating a dataframe and downloading the VIX data using its code
name and its source
vix = pdr.DataReader('VIXCLS', 'fred', start_date, end_date)

# Printing the latest five observations of the dataframe
print(vix.tail())
```

The code uses the `pandas` library to import the `DataReader` function, which fetches historical data online from a variety of sources such as Yfinance and Fred. The `DataReader` function takes the name of the data as a first argument, followed by the source, and the dates. The output of `print(vix.tail())`is shown in table 1-3:

*Table 1-3. Output of the code*

| DATE | VIXCLS |
|---|---|
| 2023-01-17 | 19.36 |
| 2023-01-18 | 20.34 |
| 2023-01-19 | 20.52 |
| 2023-01-20 | 19.85 |
| 2023-01-23 | 19.81 |

Let’s move on to the second step: data preprocessing. I divide this part into checking for invalid data and transforming the data so that it is ready for use. When dealing with time series, especially downloaded time series, you may sometimes encounter NaN values which are not numbers as there has not been a proper input in their respective cells.

> **NOTE**
>
> *NaN* stands for *Not a Number* and it occurs due to missing, invalid, or corrupt data.

You can deal with NaN values in many ways. For the sake of this example, let’s see the simplest way of dealing with these invalid values, which is to eliminate them. But first, let’s write a simple code that outputs the number of NaN values in the dataframe so that you have an idea on how many values you will delete:

```
# Importing the required library
import pandas as pd

# Checking if there are NaN values in the VIX dataframe previously
imported
count_nan = vix['VIXCLS'].isnull().sum()

# Printing the result
print('Number of NaN values in the VIX dataframe: ' + str(count_nan))
```

The code uses the `isnull()` method and sums the number it gets which gives out the number of NaN values. The output of the previous code snippet is as follows:

```
Number of NaN values in the VIX dataframe: 292
```

Now that you have an idea of how many rows you will delete, you can use the following code to drop the NaN rows, thus cleaning up the dataframe from any invalid inputs:

```
# Dropping the NaN values from the rows
vix = vix.dropna()
```

The second part of the second step is to transform the data. Data science models typically like *stationary* data which is data with stable statistical properties such as the mean and the standard deviation.

> **NOTE**
>
> The concept of *stationarity* and the required statistics metrics are discussed in detail in Chapter 2. For now, all you need to know is that it is likely that you will have to transform your raw data into stationary data when using data science models.

To transform the VIX data into stationary data, you can simply take the differences from one value relative to the previous value. This is similar to taking price data and transforming it into returns data. The following code snippet takes the VIX dataframe previously defined and transforms it into a theoretically implied[2] stationary data:

```
# Taking the differences in an attempt to make the data stationary
vix = vix.diff(periods = 1, axis = 0)

# Dropping the first value of the data frame
vix = vix.iloc[1: , :]
```

The third step is data exploration, which is all about understanding the data you have in front you, statistically speaking. As you will see statistical metrics in

detail in the next chapter, I'll limit the discussion to just calculating the mean of the dataset.

The *mean* is simply the value that can represent the other values in the dataset if they were to elect a leader. It is the sum of the values divided by their quantity. The mean is the simplest stat in the descriptive statistics world and it is definitely the most used one. The following formula shows the mathematical representation of the mean of a set of values:

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{i} x_i$$

Using `pandas`, you can easily calculate the mean of the dataset as follows:

```
# Calculating the mean of the dataset
mean = vix["VIXCLS"].mean()

# Printing the result
print('The mean of the dataset = ' + str(mean))
```

The output of the previous code snippet is as follows:

```
The mean of the dataset = 0.0003
```

The next step is data visualization, which is mostly considered as the fun step. Let's chart the VIX's differenced values through time. The following code snippet plots the VIX data shown in Figure 1-1:

```
# Importing the required library
import matplotlib.pyplot as plt

# Plotting the latest 250 observations in black with a label
plt.plot(vix[-250:], color = 'black', linewidth = 1.5, label = 'Change
in VIX')

# Plotting a red dashed horizontal line that is equal to the
calculated mean
plt.axhline(y = mean, color = 'red', linestyle = 'dashed')

# Calling a grid to facilitate the visual component
plt.grid()

# Calling the legend function so it appears with the chart
plt.legend()
```

```
# If you are using a terminal or a script, you might want to add
plt.show()
```

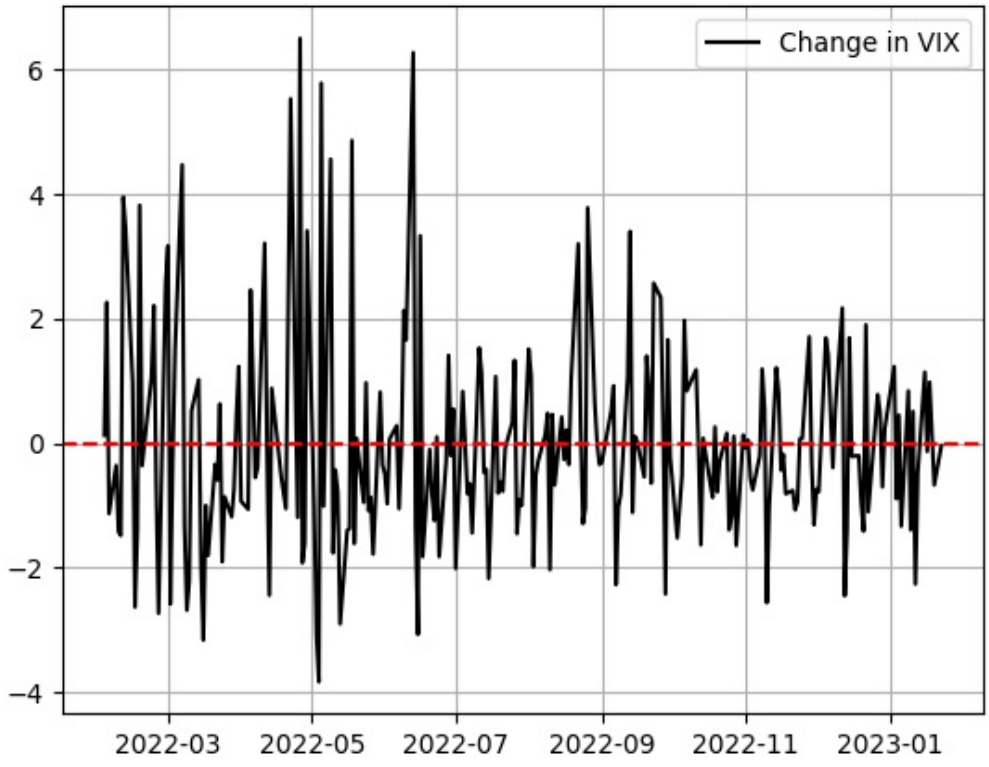


*Figure 1-1. Change in VIX since early 2022*

Steps 5 and 6, data analysis and data interpretation, are what you are going to study thoroughly in this book, so let's skip them for now and concentrate on the introductory part of data science.

Let's go back to the invalid or missing data problem before moving on. Sometimes, data is incomplete and has missing cells. Even though this has the potential to hinder the predictive ability of the algorithm, it should not stop you from continuing the analysis as there are quick fixes that help lessen the negative impact of the empty cells. For instance, consider table 1-4:

*Table 1-4. Quarterly GDP*

| Quarter | GDP |
|---|---|

| | |
|---|---|
| Q1 2020 | 0.9% |
| Q2 2020 | 1.2% |
| Q3 2020 | 0.5% |
| Q4 2020 | 0.4% |
| Q1 2021 | **#N/A** |
| Q2 2021 | 1.0% |
| Q3 2021 | 1.1% |
| Q4 2021 | 0.6% |

The table contains the quartely gross domestic product (GDP) of a hypothetical country. Notice how the table is missing the Q1 value of 2021. There are three basic ways to solve this issue:

- *Delete the cell that contains the missing value:* This is the technique used in the previous example of the data science process. It simply considers that the time stamp does not exist. It is the simplest fix.
- *Assume that the missing cell is equal to the previous cell:* This technique assumes that the current missing value equals the value previous to it. It is also a simple fix that has the aim of smoothing the data instead of completely ignoring the issue.
- *Calculate a mean or a median of the cells around the empty value:* This technique takes smoothing one step further and assumes that the missing value is equal to the mean between the previous and the next value.

Data science englobes a range of mathematical and statistical concepts. It entails a deep understanding of machine learning algorithms, such as decision trees, random forests, and neural networks. These concepts are discussed in detail but also in an easy-to-grasp manner so that technical and non-technical readers can

benefit from their intuition. Many models are assumed to be black boxes and there is a hint of truth in this, but the job of a data scientist is to first understand the models before interpreting their results. This helps in understanding the limitations of the said models.

This book uses Python as the go-to programming language to create the algorithms. As mentioned, Chapter 3 introduces Python and the required knowledge to know how to manipulate and analyze the data, but also provides the foundations to create the different models which, as you will see, are simpler than you might expect.

Before moving on to the next section, let's have a look at the concepts of data storage. After all, data is valuable, but you need to store it somewhere where it is easily fetched and analyzed.

*Data storage* refers to the techniques and areas used to store and organize data for future analysis. Data is stored in many formats, among them the common ones such as CSV and XLSX files. Other types of formats may include XML, JSON, and even JPG for images. The format is chosen according to the structure and organization of the data.

Data can also be stored in clouds or on-premise depending on the capacity of storage and the costs. For example, you may want to choose to keep your historical 1-minute Apple stock data in a cloud so that you save space on your local computer as opposed to keeping them in a CSV file.

When dealing with time series in Python, you are mostly going to deal with two types of data storages: arrays and data frames. Let's take a look at what they are:

*Arrays*

> An *array* is used to store elements of the same kind. Typically, a homogeneous data set (such as numbers) is best kept in an array. This occurs when you perform back-tests with no need of time stamps. The most used library to handle arrays in Python is `numpy`.

*Data frames*

A *data frame* is a 2-dimensional data structure that can hold data of various types (such as float, string, and so on). It can be compared to a table with columns and rows. This occurs when you perform back-tests with a need for time stamps in tandem with their respective values. The most used library to handle data frames in Python is `pandas`.

In general, arrays should be used whenever a homogeneous data collection needs to be efficiently stored. When dealing with heterogeneous data or needing to edit and analyze data in a tabular manner, you should use data frames.

> **NOTE**
>
> Data science is continuously evolving. New storage methods are being developed by time in an attempt to make them more efficient and increase their capacity and speed.

## Understanding Data Science

Data science has rapidly become an essential part in technology and progress. Algorithms rely on information provided from data science tools to perform their tasks. But what are algorithms?

An *algorithm* is a set of ordered procedures, that have the aim of completing a certain activity or address a particular issue. Algorithms can be as simple as flipping a coin or as sophisticated as the Risch algorithm [3].

Let's take a very simple algorithm that updates a charting platform with the necessary financial data:

1. Connect the server and the online data provider.
2. Copy the financial data with the most recent time stamp.
3. Paste the data into the charting platform.
4. Loop back to the first step and redo the whole process.

That is the nature of algorithms: performing a certain set of instructions with a finite or an infinite goal.

> **NOTE**
>
> The six data science stages that you saw in the previous section can also be considered an algorithm.

Trading strategies are also algorithms as they have clear rules for the initiation and liquidation of positions. An example of a trading strategy is market arbitrage.

*Arbitrage* is a type of trading strategy that aims to profit from price differences of the same asset quoted on different exchanges. These price differences are anomalies that are erased by arbitrageurs through their buying and selling activities. Consider a stock that is traded on exchange A and exchange B in different countries (for simplicity reasons, the two countries use the same currency). Naturally, the stock must trade at the same price on both exchanges. When this condition does not hold, arbitrageurs come out of their lairs to hunt.

They buy the stock on the cheaper exchange and immediately sell it on the more expensive exchange, thus ensuring a virtually risk-free profit. These operations are performed at lightning speed as differences do not last long due to the sheer power and speed of arbitrageurs. Here's a clear example:

- The stock's price at exchange A = $10.00
- The stock's price at exchange B = $10.50

The algorithm of the arbitrageur in this case will perform the following:

1. Buy the stock on exchange A for $10.00.
2. Sell the stock immediately on exchange B for $10.50.
3. Pocket the difference ($0.50) and repeat until the gap is closed.

> **NOTE**
>
> Trading and execution algorithms can be highly complex and require specialized knowledge and a

certain market edge.

Up until now, you should be aware of the main uses of data science: data interpretation and prediction:

*Data interpretation*

Also commonly referred to as *business intelligence* or simply *data intelligence*. The aim of deploying the algorithms is to understand the whats and hows of data.

*Data prediction*

Also commonly referred to as *predictive analytics* or simply *forecasting*. The aim of deploying the algorithms is to understand the whats next of data.

The main aim of using learning algorithms in financial markets is mainly to predict data so that you take an informed trading decision with the aim of capital appreciation at a success rate higher than random. This is done through many simple and complex algorithm that I discuss in this book. These learning algorithms or models can be categorized as follows:

*Supervised learning*

*Supervised learning algorithms* are models that require labeled data to function. This means that you must provide data so that the model trains itself on these past values and understands the hidden patterns with the aim of being able to deliver future outputs when encountering new data. Examples of supervised learning include *linear regression algorithms* and *autoregressive integrated moving average* (ARIMA) models. More complex models include *support vector regression algorithms* (SVR) and *neural networks*. Adding several layers to neural networks transforms them

into a deep learning model with a high aptitude of analyzing complex multi-layered data. All of these algorithms are discussed in greater depth later in the book.

*Unsupervised learning*

Unsupervised learning algorithms are models that do not require labeled data to function. This means that they can do the job with unlabelled data since they are built to find hidden patterns on their own. Examples include *clustering* algorithms and *principal component analysis* (PCA).

*Reinforcement learning*

Reinforcement learning algorithms are models that do not require data at all as they discover their environment and learn from it on their own. As opposed to supervised and unsupervised learning models, reinforcement learning models gain knowledge through feedback obtained from the environment via a reward system. Since this is generally applied to situations in which an agent interacts with the environment and learns to adopt behaviors that maximize the reward over time, it may not be the go-to algorithm for time series regression. On the other hand, it can be used to develop a policy that can apply to time series data to create predictions.

As you may have noticed, the book's title is *Deep Learning for Finance*. This means that in addition to other learning models, I will be spending a sizable portion of the book discussing deep learning models and coding trading strategies using them. This also means that the focus of the book is mainly discussing neural networks and their different variations.

Deep supervised learning models (such as deep neural networks) can learn hierarchical representations of the data because they include many layers, with

each layer extracting features at a different level of abstraction. As a result, hidden and complex patterns are learned by deep models that may be difficult for shallow (not deep) models to learn.

On the other hand, shallow supervised learning models (like linear regression) have a limited ability to learn complex non-linear relationships. But, they require less computational effort and are therefore faster.

Data science algorithms are deployed pretty much everywhere nowadays and not just in finance. Examples include the following:

- Business analytics: Optimizing pricing, predicting customer turnover, or improving marketing initiatives using data analysis.
- *Healthcare*: Improving patient outcomes, finding innovative therapies, or lowering healthcare costs through in-depth analysis of patient data.
- *Sports*: Sports data analysis to enhance team performance, player scouting, or bets.
- *Research*: Analyzing data to support scientific investigation, prove theories, or gain new knowledge.

Now, when someone talks about data science applications, it helps to know what a data scientist does.

A data scientist must evaluate and understand complex data in order to get insights and provide guidance for decision-making. Common tasks involved in this include developing statistical models, applying machine learning techniques, and visualizing data. They support the implementation of data-driven solutions and also inform stakeholders of their results.

A data engineer, on the other hand, is in charge of building and maintaining the infrastructure and tools needed to support data science projects. This entails tasks including constructing and executing data pipelines, optimizing data storage and retrieval, and building and maintaining big data-processing systems. They also work closely with data scientists to make sure they acquire the data they need for their study.

In other words, a data scientist is more concerned with the interpretation and analysis of data, whereas a data engineer is more concerned with the tools and

infrastructure needed to gather, store, and analyze data.

By now you should understand everything you need to get you started with data science. Let’s introduce the second main topic of the book: financial markets. After all, this book aims to show how to create data science models and algorithms and apply them on financial data in order to extract predictive value from them.

## Introduction to Financial Markets and Trading

The aim of this book is to present a hands-on approach onto applying different learning models to create different trading strategies. It is therefore imperative to gain a solid knowledge on how trading and financial markets work.

*Financial markets* are places where people can trade financial instruments, such as stocks, bonds, and currencies. The act of buying and selling is referred to as *trading*. The main, but not only, aim of buying a financial instrument is capital appreciation. The buyer believes that the value of the instrument is greater than its price, therefore the buyer buys the stock (*goes long*) and sells whenever they believe that the current price equals the current value. In contrast, traders can also make money if the price of the instrument goes down. This process is referred to as *short selling* and is common in certain markets such as futures and foreign exchange (FX).

The process of short selling entails borrowing the financial instrument from a third party, selling it on the market, and buying it back, before returning it to the third party. Ideally, as you expect the price of the instrument to drop, you would buy it back cheaper (after the price decrease) and give it back to the third party at the market price thus pocketing the difference.

*Long (buy) position example*

> A trader expects the share price of Microsoft to increase over the next couple of months due to improved technological regulations, which would increase the earnings. They therefore buy a number of shares at $250 and aim to sell them at $500. The trader is therefore long Microsoft stock (also referred to as being *bullish*).

*Short (sell) position example*

> A trader expects the share price of Lockheed Martin to decrease over the next couple of days due to signals from a technical strategy. They therefore sell short a number of shares at $450 and aim to buy them back at $410. The trader is therefore short Lockheed Martin stock (called being *bearish*).

**NOTE**

Markets that are trending upwards are referred to as bullish markets. Derived from the word *bull* and its aggressive nature, being bullish is related to optimism, euphoria, and greed. On the other hand, markets that are trending downwards are referred to as bearish markets. Derived from the word *bear* and its defensive nature, being bearish is related to pessimism, panic, and fear.

Financial instruments may come in their raw (physical) form or what is also known as *spot* and also in derivatives form. *Derivatives* are products that traders use to trade markets in certain ways. For example, a *forward* or a *futures* contract is a derivative contract where a buyer locks in a price for an asset to buy it at a later time.

Another type of derivatives is an option. An *option* is the right but not the obligation to buy a certain asset at a specific price in the future by paying a premium now (the option's price). When a buyer wants to buy the underlying stock, they exercise their option to do so; otherwise, they may let the option expire.

Trading activity may also occur for hedging purposes as it is not limited to just speculation. An example of this would be AirFrance (the main French airline company) hedging its business operations by buying oil futures. Buying oil futures protects AirFrance from rising oil prices which may hurt its main operations (aviation). The rising costs from using fuel to power the planes are offset by the gains from the futures. This allows the airline to focus on its main business. This whole process is called *hedging*.

Let's take an example to make things clearer, let's say an airline company expects to consume a certain amount of fuel in the next six months, but they are

worried about the potential increase in oil prices over that period. To protect against this price risk, the airline can enter into a futures contract to purchase oil at a fixed price on a future date.

If the price of oil increases during that time, the airline would still be able to purchase the oil at the lower, fixed price agreed upon in the futures contract. On the other hand, if the price of oil decreases, the airline would be obligated to pay the higher, fixed price, but the lower market price for the oil would offset that cost.

In this way, the airline can mitigate the risk of price fluctuations in the oil market and stabilize their fuel costs. This can help the airline to better manage its budget and forecast its future earnings.

As you can see, the aim is to make financial gains from the trading operations -- it aims to simply stabilize its costs by locking in a known price for oil.

Typically, financial instruments are grouped in asset classes based on their type:

*Stock markets*

> A stock market is an exchange place (electronic or physical) where companies issue shares of stock to raise money for business. When people buy shares of a company's stock, they become part owners of that company and may become entitled to dividends according the company's policy. Depending on the type of stocks, they can also gain the right to vote in board meetings.

*Fixed income*

> Governments and businesses can borrow money in the fixed income market. When a person purchases a bond, they are effectively lending money to the borrower, who has agreed to repay the loan along with interest. Depending on the borrower's creditworthiness and the prevailing interest rates, the bond's value may increase or decrease.

*Currencies*

The FX market, also referred to as the currencies market, is a place where people may purchase and sell various currencies. A currency's value can increase or decrease based on a variety of variables, including the economy, interest rates, and political stability of the nation.

*Commodities*

Agricultural products, gold, oil, and other physical assets with industrial or other uses are referred to as *commodities*. They typically offer a means to profit from global economic trends as well as being a form of hedge against inflation.

*Alternative investments*

In the world of finance, non-traditional investments such as real estate, private equity, and hedge funds are referred to as *alternative asset classes*. These alternative asset classes have the potential to offer better returns than traditional assets and offer the benefit of diversity, but they also tend to be less liquid and may be more difficult to evaluate. It's crucial to remember that each of these asset classes has unique qualities and various levels of risk, so investors should do their homework before investing in any of these assets.

Financial markets allow businesses and governments to raise money they need to operate. They also provide opportunities for investors to make money speculating and investing in interesting opportunities. Trading activities provide liquidity to the markets which makes the price more efficient and less costly. In other words, the more the market is liquid, the less are the costs to trade in it as

the number of orders makes it less likely to heavily impact the market when trading. But how do markets really work? What causes the price to go up and down?

*Market microstructure* is the research that deals with the trading of securities in financial markets. It looks at how trading works as well as how traders, investors, and market makers behave. Understanding price formation and the variables that affect trading costs is the aim of market microstructure research.

Order flow, liquidity, market effectiveness, and price discovery are just a few of the many subjects covered by market microstructure research. Additionally, it looks at how various trading techniques, including limit orders, market orders, and algorithmic trading, affect market dynamics. *Liquidity* is possibly the most important market microstructure concept. It describes how easily an asset may be bought or sold without materially changing its price. Liquidity can vary between financial instruments and over time. It can be impacted by a number of variables, including trading volume and volatility.

Furthermore, market efficiency is a crucial component of market microstructure research. Research in this area has demonstrated that some markets are more efficient than others and that elements like insider trading, price manipulation, and information asymmetry can have an impact on prices. Market efficiency is revisited in Chapter 4, which deals with technical analysis, a key analysis field in trading.

Finally, I want to discuss another important area of market microstructure: *price discovery*. This refers to the method used to set prices in a market. Prices can be affected by elements like order flow, market maker activity, and the presence of various trading methods.

Imagine you want to buy a sizable number of shares in two stocks: stock A and stock B. Stock A is very liquid while stock B is very illiquid. If you want to execute the buy order on stock A, you are likely to get filled at the desired market price with minimal impact if any. However, with stock B, you are likely to get a worse price as there is not enough sellers willing to sell at your desired buy price, and therefore, as you create more demand from your orders, the price rises to match the sellers' prices and thus, you will buy at a higher (worse) price. This is the impact liquidity can have on your trading.

# Applications of Data Science in Finance

Let’s begin peeking into the main areas of data science for finance. Every field has its challenges and problems that need simple and complex solutions. Finance is no different. Recent years have seen a gigantic leap in using data science to improve the world of finance, from the corporate world to the markets world. Let’s discuss some of these areas:

*Financial fraud detection*

Financial transactions can be examined for patterns and anomalies using data science models, which attempt to spot possible fraud. By examining transaction data from credit cards to spot odd or suspect spending patterns, one way to use data science to stop financial fraud is to find unusual or suspicious patterns of expenditures. This can involve making numerous minor purchases quickly after one another or making significant or frequent purchases from the same store. On the basis of this data, machine learning algorithms are trained to find anomalies that point to fraudulent conduct.

*Risk management*

To examine financial data and spot potential risks to portfolios, data science approaches might be applied. This can assist financial institutions in managing their risk exposure and making better-informed decisions. Analysis of market data to forecast changes in stock prices and other financial indicators is one example of applying data science for risk management in finance. This can involve analyzing vast amounts of historical data using methods like statistical modeling, machine learning, and artificial intelligence in order to spot patterns and trends that can be used to forecast the state of the market in the future.

*Credit scoring*

Data science can be used to examine financial data and credit history, forecast a person's or a company's creditworthiness, and make loan decisions. Utilizing financial data, such as income and credit history, to forecast a person's creditworthiness is one example of applying data science for credit score research. This can involve developing a prediction model that can use a number of indicators, such as prior credit performance, income, and job history, in order to evaluate a person's likelihood of repaying a loan using techniques like statistical modeling and machine learning.

*Natural Language Processing (NLP)*

In order to make better judgments, NLP analyzes and extracts insights from unstructured financial data, such as news articles, reports, and social media posts. A famous example of NLP is using the sentiment of the text to extract possible trading opportunities stemming from the intentions and feelings of the market participants and experts. NLP falls into the field of sentiment analysis (with help from machine learning).

## Summary

Data science keeps growing every day with new techniques and models appearing regularly that aim to improve the interpretation of data. This chapter has provided a simple introduction to what you need to know about data science and how you can use it in finance.

The next chapter presents the required knowledge in statistics, probability, and math that you may need when trying to understand data science models. Even though the aim of the book is to present a hands-on approach of creating and applying the different models using Python, it helps for you to understand what you're dealing with instead of blindly applying them onto data.

If you need a Python refresher, see Chapter 3, which is a basic introduction. It

sets the foundations to what's to come next in the book. You do not need to become a Python master to perform data science but you must understand code and what it refers to, and especially how to debug and detect errors in the code.

---

1 OHLC refers to the four essential pieces of market data: open price, high price, low price, and close price.

2 The reason I am saying implied is because stationarity must be verified through statistical checks that you will see in the next chapter. At the moment, the assumption is that differencing the data gives stationary time series.

3 The Rish algorithm is an indefinite integration technique used to find antiderivatives.

# Chapter 2. Essential Probabilistic Methods for Deep Learning

**A NOTE FOR EARLY RELEASE READERS**

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 2nd chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at *ccollins@oreilly.com*.

The rise and accessibility of technology have made it possible for everyone to deploy machine learning and deep learning algorithms for data analysis and optimization. But unfortunately, this means that a large number of users do not understand the basics and underlyings of the different learning models. This makes machine learning nothing short of a black box to them, which is a recipe for disaster.

Fundamental concepts in probability, statistics, and math are essential for understanding and mastering data as well as the creation of models that seek to interpret and forecast it if possible. This chapter presents the basics of the numerical concepts needed to understand the different learning algorithms, or at the very least, shows the starting points from where you can build up your knowledge towards mastering these mathematical topics.

For simplicity, the term *machine learning* used in this book refers to all types of learning models (such as machine, deep, and reinforcement learning).

## A Primer on Probability

*Probability* is all about describing random variables and random events. The world is filled with randomness, and the best way to find your way through chaos is to try to explain it using probabilistic methods. Granted, the phrase *explain chaos* may be an oxymoron, as chaos cannot really be explained, but we humans cannot relinquish control over uncertain events, and with progress we have developed tools to make sense out of the scary world.

You may wonder what is the use of understanding the basics of probability when trying to develop machine learning algorithms for financial trading. This is a reasonable question, and you must know that the foundations of a discipline do not necessarily resemble it.

For example, to become a pilot, you have to have to study aerodynamics first, which is filled with technical concepts that do not resemble the final skill acquired at graduation. This is similar to what is being done in this chapter; by studying probabilistic essentials, statistical concepts, and mathematical topics, you will start on the right track towards being a machine learning developer.

Knowing the utility of what you are learning should give you a motivation boost. Here are some key probability topics that are important for machine learning:

*Probability distribution functions*

The possibility of seeing various outcomes of a random variable is described by a *probability distribution*. For many machine learning techniques, it is essential to comprehend the features and attributes of typical probability distributions. Probability distribution functions also describe different types of time series data, which in turn helps in choosing the right algorithm.

*Bayes' theorem for updating probabilities*

Bayes' theorem is a cornerstone of probability theory and offers a method for updating an event's probability in light of new data. It is incorporated into a variety of machine learning techniques, including as Bayesian networks and classifiers.

*Hypothesis testing*

> *Hypothesis testing* is used to establish whether a population-based assertion is more likely to be true or incorrect based on a sample of data. Many machine learning models employ hypothesis testing in their process.

*Decision trees*

> *Decision trees* are a type of machine learning algorithm that borrows from probabilistic concepts such as conditional probability, a concept covered in this chapter. For more detail, decision trees are covered in Chapter 7.

*Information theory*

> *Information theory* is the complex study of how information is quantified, stored, and transmitted. It is incorporated into numerous machine learning techniques, including decision trees.

## Introduction to Probabilistic Concepts

The most basic piece of probabilistic information is a *random variable,* which is an uncertain number or outcome. Random variables are used to model events that are considered uncertain, such as the future return of a currency pair.

A random variable is either discrete or continuous. A *discrete random variable* has a finite set of values, while a *continuous random variable* has values within a certain interval. Consider the following two examples to clarify things:

- An example of a discrete random variable would be the result of a rolling a die. They are limited by the following set {1, 2, 3, 4, 5, 6}.
- An example of a continuous random variable would be the daily price returns of EURUSD (The exchange rate of 1 Euro per US Dollars).

Random variables are described by *probability distributions,* which are functions that give the probability of every possible value of these random variables. Generally, a histogram is used to show the probability. Histogram plotting is discussed later in the chapter.

At any moment, the probability that a certain event unfolds is between 0 and 1. This means that probability is assigned to random variables on a scale between 0 and 1 such that a probability of 0 represents zero chance of occurence and a probability of 1 represents a certainty of occurence.

You can also think of this in percentage terms, which range from 0% to 100%. Values within the two numbers are valid, which means that you can have a 0.5133 (51.33%) probability of a certain event occurring. Consider rolling a die that has six sides. What is the probability of getting 3 knowing that the die is not manipulated in any way?

As the die has six sides, there are six equal probabilities for every outcome, which means that for any outcome, the probability is found as follows:

$$P(x) = \frac{1}{6} = 0.167$$

With *P(x)* designating the probability of event *x*. This gives the answer to the question:

$$P(3) = \frac{1}{6} = 0.167$$

When a die is rolled, there can only be one result. It cannot give 3 and 4 simultaneously, since one side has to dominate the other. This is the concept of *mutual exclusivity*. Mutually exclusive events (such as getting a 3 or getting a 4 in a die roll) eventually sum up to 1. Take a look at the following example:

$$P(1) = \frac{1}{6} = 0.167$$

$$P(2) = \frac{1}{6} = 0.167$$

$$P(3) = \frac{1}{6} = 0.167$$

$$P(4) = \frac{1}{6} = 0.167$$

$$P(5) = \frac{1}{6} = 0.167$$

$$P(6) = \frac{1}{6} = 0.167$$

Summing all these mutually exclusive events gives 1, which means that the sum of the possible probabilities in a six-sided die is as follows:

$$P(1) + P(2) + P(3) + P(4) + P(5) + P(6) = 1$$

> **NOTE**
>
> Stating that a random variable has a 0.8 probability of occurring is the same as stating that the same variable has a 0.2 probability of not occurring.

Probability measures can be conditional or unconditional. A *conditional probability* is where the occurrence of an event impacts the probability that another events occurs. For example, the probability of a sovereign interest rate hike given positive employment data is an example of a conditional probability. The probability of event A given the occurrence of event B is denoted by the following mathematical notation: *P(A|B)*

In contrast, *unconditional probability* is not dependent on other events. Taking the example of the conditional probability, you can formulate an unconditional probability calculation which measures the probability of an interest rate hike regardless of other economic events.

Probabilities have specific addition and multiplication rules with their own interpretations. Let’s take a look at the formulas before seeing an example. The *joint probability* of the realization of two events is the probability that they will both occur. It is calculated using the following formula:

$$P(AB) = P(A|B) \times P(B)$$

What that formula says is that the probability of occurence for both A and B is the probability that A occurs given B occurs multiplied by the probability that B occurs. Therefore, the right side of the equation multiplies a conditional probability by an unconditional probability.

The *addition rule* is used to determine the probability that at least one of the two outcomes will occur. This works in two ways: the first one deals with mutually exclusive events, and the second one deals with events that are non mutually

exclusive:

If the events are not mutually exclusive, then to add avoid double counting, the formula is:

$$P(A\ or\ B) = P(A) + P(B) - P(AB)$$

If the events are mutually exclusive, then the formula is simplified to the following:

$$P(AB) = 0$$

$$P(A\ or\ B) = P(A) + P(B) - 0$$

$$P(A\ or\ B) = P(A) + P(B)$$

Notice how in mutually exclusive events, it’s either A or B that can be realized, and therefore the probability that both of them will occur is zero. To understand why you need to subtract the joint probability of A and B, take a look at Figure 2-1.

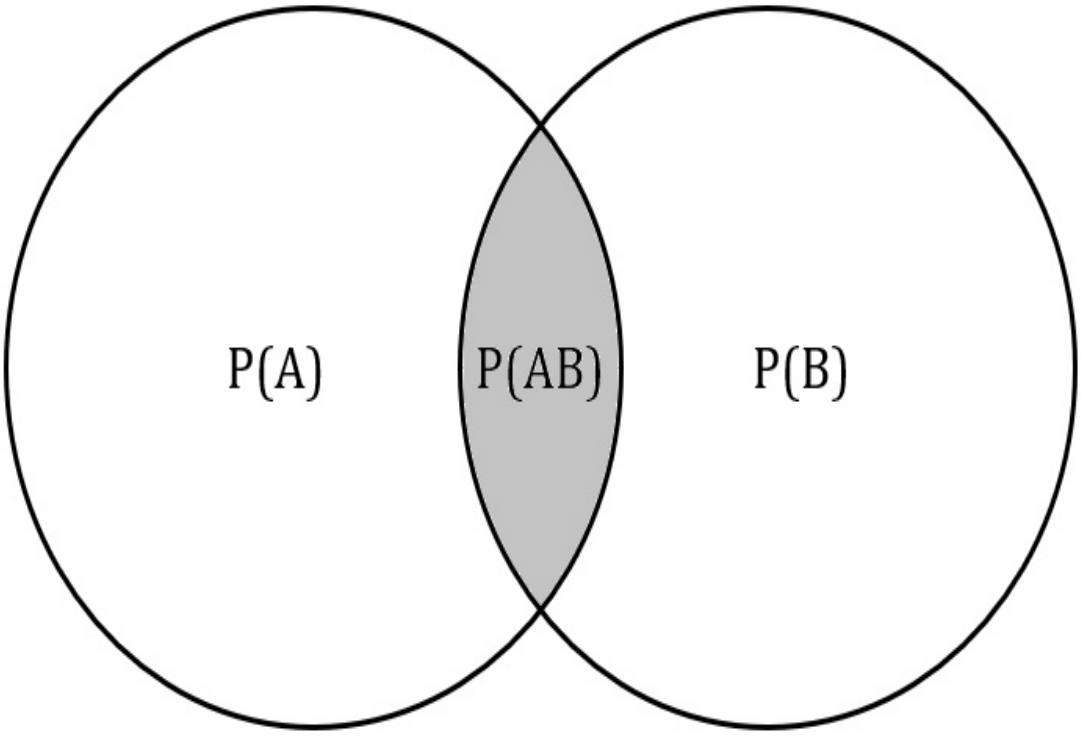


*Figure 2-1. The addition rule of probability*

Notice how the probability of either A or B occurring while they are mutually exclusive must not include their joint probability. Let's now look at the concept of independent events.

*Independent events* are not tied together (for example, rolling the die twice). The joint probability is calculated as follows:

$$P(AB) = P(A) \times P(B)$$

Independent events therefore refer to instances where the occurrence of one has absolutely zero impact on the occurrence of the others. Now, let's see an example to validate these concepts. Consider a simple coin toss. The probability of getting heads does not depend on what you have gotten in the previous coin toss. Therefore the probability of getting heads is always 0.50 (50%). To take things further, what is the probability of getting only heads after five coin tosses?

As the probability of each event is independent from the previous or the next one, the formula is as follows:

$$P(x) = 0.50 \times 0.50 \times 0.50 \times 0.50 \times 0.50 = 0.03125 = 3.125\%$$

The *expected value* of a random variable is the weighted average of the different outcomes. Therefore, the expected value is really another way of referring to the mean. Mathematically, the expected value is as follows:

$$E(X) = \sum_{i=1}^{n} \left( P(x_i)\, x_i \right)$$

Take a look at Table 2-1 and try to calculate the expected value of the next employment numbers in a certain month of the year.

*Table 2-1. Employment table*

| Non-farm payrolls | Probability |
|---|---|
| 300,000 | 0.1 |
| 400,000 | 0.3 |
| 500,000 | 0.5 |
| 600,000 | 0.1 |

*Non-farm payrolls* refer to a monthly report issued by the US Department of Labor that gives information on the total number of paid employees in the nation, excluding those employed in the agriculture sector, as well as those employed by the government and non-profit organizations.

From Table 2-1, economists assume there is a 50% probability that there will be a 500,000 increase in the total number of paid employees and a 30% probability that there will be a 400,000 increase in the total number of paid employees. The expected value is therefore:

$$E(X) = (300,000 \times 0.1) + (400,000 \times 0.3) + (500,000 \times 0.5) + (600,000 \times 0.1) = 460,000$$

Therefore, the number that represents the economists' consensus is 460,000, as it

is the closest weighted value to most forecasts. It is the value that represents the dataset.

Before closing the section on introductory probability, there exists a mathematical formula known as *Bayes' Theorem* that estimates an event's likelihood based on knowledge of previous, related events. The formula for Bayes' Theorem is as follows:

$$P(A|B) = \frac{P(B|A).P(A)}{P(B)}$$

where:

- *P(A|B)* is the probability of event A occurring given that event B has occurred.
- *P(B|A)* is the probability of event B occurring given that event A has occurred.
- *P(A)* is the probability of event A occurring.
- *P(B)* is the probability of event B occurring.

In other words, Bayes' Theorem allows you to update your beliefs about the probability of an event based on new information.

**NOTE**

The main takeaways from this section are as follows:

- Probability describes random variables and random events. It is a value between 0 and 1.
- Probabilities of events may be grouped together to form more complex scenarios.
- The expected outcome is the weighted average of every probability in the designated universe.

## Sampling and Hypothesis Testing

When populations are large, representative samples are taken so that they become the main describers of data. Take the United States. Its democratic system means that the people hold the right to decide their own fate, but it's not

possible to go to every person and ask them about their detailed opinions on every topic out there. This is why elections are held and representatives are elected so that they act in the people's name.

*Sampling* refers to the act of selecting samples of data within a larger population and and making conclusions about the statistical properties of the population. There are a few different methods of sampling. The most known ones are the following:

*Simple random sampling*

With simple random sampling, each element in the population has an equal chance of being selected for the sample. This can be a random number generated on a labeled population where each individual has the same probability of being selected.

*Stratified sampling*

With stratified sampling, the population is divided into groups based on some characteristic, and then a simple random sample is taken from each group in proportion to its size.

*Cluster sampling*

With cluster sampling, the population is divided into clusters, and a random sample of clusters is selected. Then, all elements within the selected clusters are included in the sample.

*Systematic sampling*

With systematic sampling, an element is selected by choosing every *nth* individual from the population, where *n* is a fixed number. This means that it is not random but pre-specified in advance.

A rule of thumb is that the more data you acquire, the better the metrics reflect the population. Sampling is extremely important in the world of machine learning as quite often, you are taking samples of data to represent the true population. For example, when performing a back-test on a strategy, you will be required to split the whole data set into a *training sample* and a *testing sample* where the first is the sample of data on which the algorithm understands its structure, and the second is the sample of data on which the algorithm tests its predictive power.

Similarly, another example of using sampling is *cross validation*, a technique that divides a dataset into two or more subgroups. The model is trained using one subset, and its results are tested using the other subsets. For various subsets of the data, this procedure is repeated numerous times, and then the model's average performance is determined.

These terms are discussed in more depth in the coming chapters. For now you should understand that the concept of sampling is very important in machine learning (and even more in deep learning with optimization techniques).

Sampling is not perfect and errors may be possible just as any other estimation method. *Sampling error* refers to the difference between the statistic of the sample and the statistic of the population (if it's known). A *statistic* is a metric that describes the analyzed dataset (an example of this would be the mean, a statistic you will see in greater detail in Chapter 3 dealing with statistics). Now, what is the minimum sample size you should have to be able to make inferences about the population? The rule of thumb is to have a minimum of 30 observations and the more the merrier. This brings the discussion to the *central limit theorem* which states that random samples drawn from a population will approach a normal distribution (a probability distribution that is symmetric and bell-shaped) as the sample gets larger.

The central limit theorem makes it simple to apply inferences and conclusions as hypothesis testing goes well with a normal distribution. Before proceeding to hypothesis testing, let's look at *confidence intervals*, ranges of values where the population parameter is expected to be. Confidence intervals are generally constructed by adding or subtracting a factor from the point estimate. For example, given a sample mean $\bar{x}$, a confidence interval can be constructed as follows:

$$\overline{x} \pm (reliability\ factor \times standard\ error)$$

Let’s try to understand the calculation step by step. The sample mean is an estimate of the population and is calculated because it is not possible to calculate the population means, therefore, by performing a random sample, the assumption is that the sample mean should be equal to the population mean. However, in real life, things may differ, and this why you should construct a confidence interval using probabilistic methods.

> **NOTE**
>
> The *significance level* is the threshold of the confidence interval. For example, a confidence interval of 95% means that with 95% confidence, the estimate should lie within a certain range. The remaining 5% probability that it does not, is called a significance level (generally marked with the alpha symbol α).

A *reliability factor* is a statistical measure that depends on the distribution of the estimate and the probability that it falls within the confidence interval. For the sake of simplicity, let’s assume that the variance of the population is normal and the population is normally distributed. For a significance level of 5% (thus, a confidence interval of 95%), the reliability factor is 1.96 in this case (the way you get this number is less relevant to the discussion).

The *standard error* is the standard deviation of the sample. *Standard deviation* is discussed in greater depth in Chapter 3; for now, just know that it represents the fluctuations of the different values around the mean. Standard error is found using the following formula:

$$s = \frac{\sigma}{\sqrt{n}}$$

$$\sigma\ is\ the\ population\ standard\ deviation$$

$$\sqrt{n}\ is\ the\ square\ root\ of\ the\ population\ number$$

It is also worth knowing that for a 1% significance level, the reliability factor is 2.575, and for a 10% significance level, the reliability factor is 1.645. Let’s take a practical example to make sense out all of this math.

Consider a population of 100 financial instruments (bonds, currency pairs, stocks, structured products, etc.). The mean annual return of these instruments is

1.4%. Assuming a population standard deviation of 4.34%, what is the confidence interval at 1% significance level (99% confidence interval) of the mean?

The answer is just plugging the values in the formula as follows:

$$1.4\% \pm 2.575 \times \frac{4.34\%}{\sqrt{100}} = 1.4\% \pm 1.11\%$$

This means that the confidence interval is between (0.29%, 2.51%).

Let's see another example. Consider that the annual returns on precious and industrial metals (such as gold and copper) are normally distributed with a mean of 3.5% and a known population standard deviation of 5.1%. What is the confidence interval with 10% significance level of the annual returns on 5 different commodities? The answer is as follows:

$$3.5\% \pm 1.645 \times \frac{3.5\%}{\sqrt{5}} = 3.5\% \pm 2.23\%$$

This means that the confidence interval is between (1.27%, 5.8%).

**NOTE**

If the sample size is small and / or the population standard deviation is unknown, a t-distribution may be a better choice than a normal distribution.

The *t-distribution* is a type of probability distribution used to model the distribution of a sample mean when the sample size is small and/or when the population standard deviation is unknown. It resembles the normal distribution in shape, but with heavier tails, which represents the uncertainty associated with smaller sample sizes.

Before closing the discussion on sampling and estimation, the following list shows the appropriate distributions given the characteristics of the population:

- A small normal distribution with known variance should use the reliability factor of the normal distribution.
- A large normal distribution with known variance should use the reliability factor of the normal distribution.
- A small normal distribution with unkown variance should use the reliability factor of the t-distribution.

- A large normal distribution with unkown variance should use the reliability factor of the t-distribution.
- A large non-normal distribution with known variance should use the reliability factor of the normal distribution.
- A large non-normal distribution with known variance should use the reliability factor of the t-distribution.

Remember that *large* means that the number of observations are greater than 30. The non covered combinations in the previous list are complex and out of scope of this discussion.

The next stop is hypothesis testing, a key probabilistic technique of getting conclusions on samples of data. This part is extremely important as it's used in almost all types of statistical analyses and models.

In statistics, *hypothesis testing* is a technique for drawing conclusions about a population from a small sample of data. It entails developing two competing hypotheses, the *null hypothesis* and the *alternative hypothesis*, about a population parameter, and then figuring out which is more likely to be accurate using sample data.

For example, a financial analyst is evaluating two portfolios from a risk perspective. They formulate two hypotheses:

- The null hypothesis states that there is no significant difference in the volatility of the two portfolios.
- The alternative hypothesis states that there is a significant difference in the volatility of the two portfolios.

The hypothesis is then tested using statistical analysis to determine if the difference in volatility is statistically significant or due to pure chance.

Following the definition of the null and alternative hypotheses, a test statistic is computed using the sample data. To assess the result's significance, the test statistic is then compared to a critical value drawn from a standard distribution. The null hypothesis is rejected and the alternative hypothesis is accepted if the test statistic is inside the crucial zone. The null hypothesis is not rejected and the conclusion that there is insufficient evidence to support the alternative

hypothesis is reached if the test statistic does not fall inside the crucial zone.

This is all fancy talk to say that hypothesis testing is basically creating two opposing scenarios, running a probability check, and then deciding which scenario is more likely true. Hypothesis testing can take two forms:

- *One-tailed test*: An example of this would be to test if the return on certain financial instruments is greater than zero.
- *Two-tailed test*: An example of this would be to test if the the return on certain financial instruments is different from than zero (meaning that it can be either greater or smaller than zero).

**NOTE**

Hypothesis tests are generally two-tailed.

The null hypothesis is the one that you want to reject and therefore is tested in the hopes of getting rejected and accepting the alternative scenario. A two-tailed test takes the following general form:

$$H_0 : x = x_0$$
$$H_a : x \neq x_0$$

As the alternative scenario allows for values above and below zero (which is the stated level in the null hypothesis), there should be two critical values. Therefore, the rule of a two-tailed test is to reject the null hypothesis if the test statistic is greater than the upper critical value or if the test statistic is lower than the lower critical value. For instance, for a normally distributed data, the test statistic is compared with the critical values (at 5% significance level) at +1.96 and -1.96. The null hypothesis is rejected if the test statistic falls outside the range between +1.96 and -1.96.

The process of hypothesis testing entails the calculation of the test statistic. It is calculated by comparing the point estimate of the population parameter with the hypothesized value of the null hypothesis. Both are then scaled by the standard error of the sample. The mathematical representation is as follows:

$$test\ statistic = \frac{sample\ statistic - hypothesized\ value}{standard\ error}$$

An important consideration in hypothesis testing is that the sample may not be representative, which leads to errors in describing the population. This gives rise to two types of errors:

- *Type I error*: This error occurs when rejecting the null hypothesis even though it is true.
- *Type II error*: This error occurs when failing to reject the null hypothesis even though it is false.

Intuitively, the significance level is the probability of making a type I error. Remember that if α = 5%, then there is a 5% chance of rejecting a true null hypothesis by mistake. An example would make things clearer.

Consider an analyst doing research on the annual returns of a long-short portfolio over a period of 20 years. The mean annual return was 1% with a standard deviation of 2%. The analyst's opinion is that the annual mean return is not equal to zero and they want to constuct a 95% confidence interval for this and then construct a hypothesis test:

1. State the variables. The size of the sample is 20, the standard deviation is 2% and the mean is 1%.
2. Calculate the standard error, which in this case is 0.44% as per the formula.
3. Define the critical values for the 95% confidence interval, which are +1.96 and -1.96.
4. The confidence interval is therefore (0.13%, 1.86%).
5. Specify the null hypothesis, which is, according to the analyst's opinion, a two-tailed test. The null hypothesis is that the annuel return equals zero. You should reject it if the test statistic is less than -1.96 or greater than +1.96.
6. Using the formula to find the test statistic gives 2.27. Therefore, the null hypothesis is rejected.

One more important metric to discuss: the p-value. The *p-value* is the probability

of seeing a test statistic more extreme than the one seen in the statistical test given that the null hypothesis is true. Comparing a p-value to a significance level —typically 0.05—allows you to understand it. The result is deemed statistically significant, and the null hypothesis is rejected in favor of the alternative hypothesis if the p-value is less than or equal to the significance level.

If the p-value is less than the significance level of 5%, it means that there is a 5% chance to see a test statistic as extreme as the current one if the null hypothesis is true. Another way of defining the p-value is to consider it as the smallest significance level for which the null hypothesis can be rejected.

**NOTE**

The main takeaways from this section are as follows:

- Sampling refers to the collection of data within a population in the aim of making conclusions about the statistical properties of the aforementioned population.
- Sampling is used extensively in machine learning. One example is cross validation.
- Hypothesis testing is a technique for drawing conclusions about a population from a small sample of data.

## A Primer on Information Theory

*Information theory* is a complex abstract mathematical field that is closely related to probability. It is the study of how information is quantified, stored, and transmitted. There are three conditions of occurrence when it comes to an event:

- *Uncertainty*: If the event has not occurred yet.
- *Surprise*: If the event has just occurred.
- *Information*: If the event has occurred in the past.

One of the key concepts in information theory is *entropy*: the level of uncertainty or randomness in a message or information source. It describes the degree to which an event or message is unexpected. In contrast, *information gain* measures the reduction in entropy (surprise) when receiving new information.

Basically, information theory describes the surprise of events. When an event has a low probability of occurrence, it has more surprise and hence, more information to provide. Similarly, when an event has a high probability of occurrence, it has less surprise and therefore, less information. What you should retain from this is that the amount of information learned from an unlikely event is greater than the amount of information learned from a likely event.

Before starting to dig a little deeper in information theory, it is important to understand what a *logarithm* is and for that matter what an *exponent* is. A general exponential function takes a certain constant or a variable to a certain power:

$$f(x) = a^x$$

In other words, the *exponent of a number* is the number of times you will multiply it by itself:

$$4^3 = 4 \times 4 \times 4 = 64$$

In contrast, a logarithm is the opposite of an exponent, and its aim is to find the exponent (knowing 4 and 64 from the previous example and finding 3):

$$\log_4(64) = 3$$

A logarithm, therefore, is the answer to how many of one number to multiply to get another number. Since they are literally inverse functions, you can use them together to simplify or even solve for x. Take the following example:

$$\log_4(x) = 3$$

The objective here is to find x given the logarithmic function. The first step is simply to use the exponential function on one side as you want it to cancel out the logarithm on the right (remember, inverse functions cancel each other out). This gives us the following result:

$$4^{log_4(x)} = 4^3$$

$$x = 4^3$$

$$x = 64$$

Logarithms can have different bases. However, the most used logarithm has a base of 10. In computer science, base 2 logarithms represent bits (binary digits). Therefore, information is represented as bits. The formula of information gain is

as follows:

$$H\left(x_i\right) = -log_2\left(P\left(x_i\right)\right)$$

Let's assume two variables *x* and *y* where *x* has a probability of 1 (100% and therefore, certain) and *y* has a probability of 0.5 (50% and therefore, mostly random), what would be the information in these two cases? The answer is as follows:

$$H\left(x\right) = -log_2\left(P\left(1\right)\right) = 0$$

$$H\left(y\right) = -log_2\left(P\left(0.5\right)\right) = 1$$

So the certain event gives zero information and the one that has a fifty-fifty chance of realizing has an information of 1. What about the very unlikely event *z* that has a probability of 0.05 (5%)?

$$H\left(z\right) = -log_2\left(P\left(0.05\right)\right) = 4.32$$

A negative relationship between probability and information is therefore one of the principles of information theory. Entropy and information are related concepts, but they have different meanings and applications.

*Entropy* is a metric used to assess how chaotic or random a system is. Entropy describes how uncertain or unpredictable a signal is. The degree of disorder or unpredictability in the system or communication increases as entropy increases.

*Information* is the decrease in entropy or uncertainty that happens as a result of receiving a signal. A signal's ability to lessen the receiver's uncertainty or entropy increases with its informational content.

**NOTE**

Entropy is maximized whenever all the events are equally likely.

Entropy is calculated using the following formula:

$$S\left(x_n\right) = \sum_{i=1}^{n}\left(-log_2\left(P\left(x_i\right)\right).\left(P\left(x_i\right)\right)\right)$$

Therefore, it is the average of the sum of logarithms times their respective probabilities.

Now, let's discuss the final concept of the section, *information gain*. The reduction in entropy caused by changing a dataset is calculated via information gain.

Information gain is one of the key concepts you will see in Chapter 7 with decision trees, and therefore you may want to refer to this section after understanding what decision trees are.

You mainly calculate information gain by comparing the entropy of a dataset before and after a transformation. Recall that entropy is maximized when all the outcomes of a random event have the same probability. This can also be presented as a distribution where a symmetrical distribution (such as the normal distribution) has high entropy and a skewed distribution has low entropy.

**NOTE**

Minimizing entropy is related to maximizing information gain.

Before closing this introductory section on information theory (you will see it in greater depth when discussing decision trees), let's look at the concept of *mutual information*. This measure is calculated between two variables, hence the name *mutual*, and it measures the reduction in uncertainty of a variable given another variable. The formula for mutual information is as follows:

$$MI(x, y) = S(x) - S(x|y)$$

$$MI(x, y) \; is \; the \; mutual \; information \; of \; x \; and \; y$$

$$S(x) \; is \; the \; entropy \; of \; x$$

$$S(x|y) \; is \; the \; conditional \; entropy \; of \; x \; given \; y$$

Mutual information therefore measures the dependence between the variables. The greater the mutual information, the bigger the relationship between the variables (a value of zero represents independent variables). Keep this concept in mind as you will see it in Chapter 3 in the section that deals with correlations. This is because mutual information can also be a measure of non-linear correlation between the variables.

**NOTE**

Let’s do a summary of what you need to retain in information theory to have a basic knowledge of what’s to come:

- Information theory uses concepts from probability to calculate information and entropy that are used in machine learning models and other calculations (such as correlation).
- Information is the decrease in entropy or uncertainty that happens as a result of receiving a signal. Entropy is a metric used to assess how chaotic or random a system is.
- Mutual information is a measure of dependence between two random variables. It can also be used to calculate the correlation between the two.
- Tools from information theory are used in some machine learning models such as decision trees.

## Summary

Probability presents a basic framework before continuing towards more advanced topics. This chapter skimmed over the concepts that you may encounter when dealing with machine and deep learning models. It is important to understand how probability is calculated and how hypothesis testing is performed (even though, in reality algorithms will do this for you).

The next chapter is extremely important and presents the required statistical knowledge you need, not just for machine learning but also for financial trading and even complex data analysis.

# Chapter 3. Descriptive Statistics and Data Analysis

> **A NOTE FOR EARLY RELEASE READERS**
>
> With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.
>
> This will be the 3rd chapter of the final book. Please note that the GitHub repo will be made active later on.
>
> If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at *ccollins@oreilly.com*.

*Descriptive statistics* is a field that describes data and extracts as much information as possible from it. Basically, descriptive statistics can act like the representative of the data since it briefs up its tendencies, behavior, and trends.

Trading and analysis borrows a lot from the metrics of this field. You will see in this chapter the main concepts that you need in order to have a solid grasp on data analysis. I always found that the best educational tools are practical examples, therefore, I will present this chapter using one example of an economic dataset.

Let's take inflation numbers coming from the USA. The consumer price index (CPI) measures the prices paid by urban consumers for a selection of products and services on a monthly basis (meaning that every month, a new observation is released to the public, thus forming a continuous time series). The inflation rate between any two time periods is measured by percentage changes in the price index. For example, if the price of bread last year was $1.00 and the price today is $1.01, then the inflation is 1.00%.

The code that you can use to get the CPI data resembles the one you have used

to get the VIX data in Chapter 1.

```
# Importing the required library
import pandas_datareader as pdr

# Setting the beginning and end of the historical data
start_date = '1950-01-01'
end_date   = '2023-01-23'

# Creating a dataframe and downloading the CPI data using its code
name and its source
cpi = pdr.DataReader('CPIAUCSL', 'fred', start_date, end_date)

# Printing the latest five observations of the dataframe
print(cpi.tail())

# Importing the required library
import pandas as pd

# Checking if there are NaN values in the CPI dataframe previously
defined
count_nan = cpi['CPIAUCSL'].isnull().sum()

# Printing the result
print('Number of NaN values in the CPI dataframe: ' + str(count_nan))

# Dropping the NaN values from the rows
cpi = cpi.dropna()

# Transforming the CPI into a year-on-year measure
cpi = cpi.pct_change(periods = 12, axis = 0) * 100
cpi = cpi.dropna()
```

By now, you should have a data frame that contains the yearly changes on the CPI. The year-on-year change is the most observed transformation on the CPI as it gives a clear and simple measurement of the change in the overall price level over a sufficiently enough period of time to account for short-term swings and seasonal impacts (recall the bread example).

Hence, the yearly change of the CPI serves as a gauge of the general trend in inflation. It is also simple to comprehend and compare across other nations and historical times, making it a popular measure among policymakers and economists (albeit the flaw of element weightings in the baskets between different countries). Let’s see how to analyze the dataset from a statistical point of view.

## Measures of Central Tendency

*Central tendency* refers to the calculations that summarize the dataset into a value that can represent them. The first and most known central tendency measure is the mean (average). The *mean* is simply the sum of the values divided by their quantity. It is the center point of the dataset and most likely the value that represents it the best. The mathematical formula of the mean is as follows:

$$\overline{x} = \frac{1}{n}\sum_{n}^{i=1} x_i = \frac{1}{n}(x_1 + \ldots + x_n)$$

Let's take a simple example of two datasets. Suppose you want to calculate the mean on dataset A and dataset B. How would you do it?

- Dataset A = [1, 2, 3, 4, 5]
- Dataset B = [1, 1, 1, 1]

Dataset A contains 5 values (quantity) with a total sum of 15. This means that the mean is equal to 3. Dataset B contains 4 values with a total sum of 4. This means that the mean is equal to 1.

> **NOTE**
>
> When all the values in a dataset are the same, the mean is the same as the values.

Figure 3-1 shows the US CPI year-on-year values for the last twenty years. The higher dashed line is the monthly mean calculated over the past twenty years. The lower dashed line symbolizes zero where below it are deflationary periods.

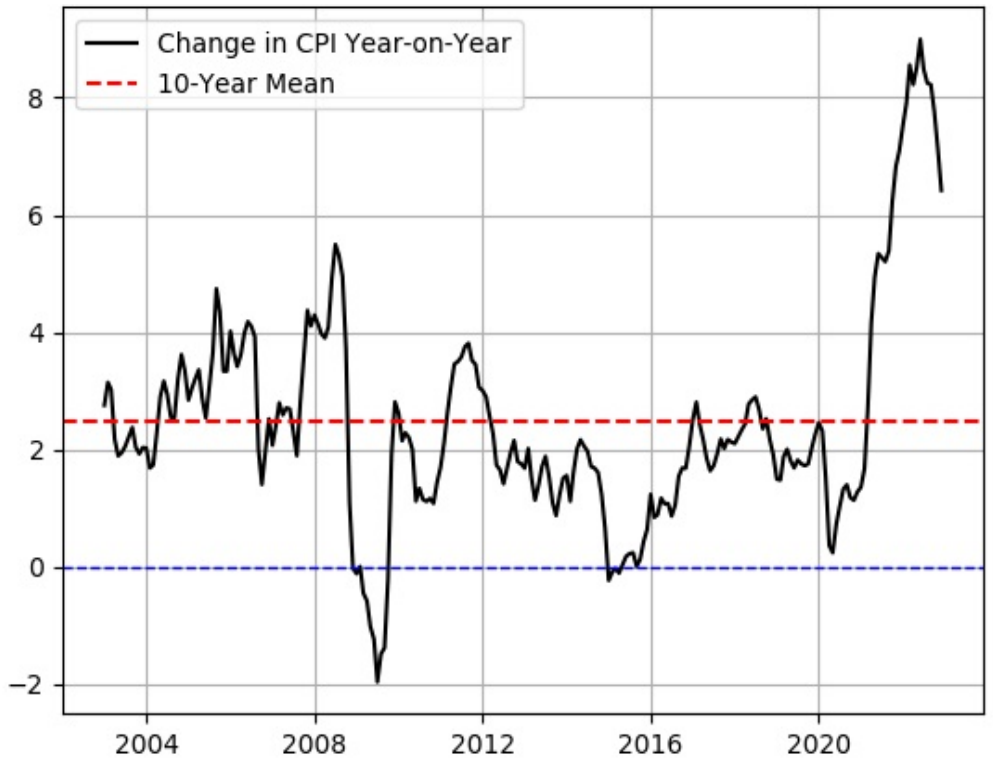


*Figure 3-1. US CPI year-on-year changes for the last twenty years*

You can create Figure 3-1 by using the following code:

```
# Calculating the mean of the CPI over the last 10 years
cpi_last_ten_years = cpi.iloc[-240:]
mean = cpi_last_ten_years["CPIAUCSL"].mean()

# Printing the result
print('The mean of the dataset: ' + str(mean), '%')

# Importing the required library
import matplotlib.pyplot as plt

# Plotting the latest observations in black with a label
plt.plot(cpi_last_ten_years[:], color = 'black', linewidth = 1.5,
label = 'Change in CPI Year-on-Year')

# Plotting horizontal lines that represent the mean and the zero
threshold
plt.axhline(y = mean, color = 'red', linestyle = 'dashed', label =
'10-Year Mean')
```

```
plt.axhline(y = 0, color = 'blue', linestyle = 'dashed', linewidth =
1)

# Calling a grid to facilitate the visual component
plt.grid()

# Calling the legend function so it appears with the chart
plt.legend()
```

The output of the mean should be as follows.

```
The mean of the dataset: 2.4794 %
```

This means that the average observation of the CPI's change on a yearly basis is around 2.50%. Even though the Federal Reserve does not have an explicit inflation target, it is generally believed that there is a consensus to maintain the annual change in inflation around 2.00%, hence not far from the historical observations. With the recent high inflation numbers since 2021 as a result of political and economic turmoil, it becomes necessary to revert back to the mean to stabilize the current situation. This examples gives a numerical value to what is referred to as normality (~2.50%) in the past 10 years.

Clearly, with the high inflation numbers (~6.00%) around the beginning of 2023, the situation is a bit far from normality, but how far? This question is answered in the next section that discusses measures of variability. For now, let's continue the discussion on central tendency.

The next measure is the *median* which in simple terms is the value that splits the data set into two equal sides. In other words, if you arrange the dataset in an ascending order, the middle value is the median. The median is used whenever there are many outliers or skew in the distribution (which may bias the mean and make it less representative).

There are generally two topics associated with calculating the median, the first one relates to a dataset that contains an even number of values (for example, 24 rows) and the second one relates to a dataset that contains an uneven number of values (for example, 47 rows):

*Calculating the median of an even data set*

If the arranged dataset has an even number of values, the median is the

average of the two middle values.

*Calculating the median of an uneven data set*

If the arranged dataset has an uneven (odd) number of values, the median is simply the middle value.

Let’s take a simple example of two datasets. Suppose you want to calculate the median on dataset A and dataset B. How would you do it?

- Dataset A = [1, 2, 3, 4, 5]
- Dataset B = [1, 2, 3, 4]

Dataset A contains five values which is an uneven number. This means that the middle value is the median. In this case, it is 3 (notice how it is also the mean of the dataset). Dataset B contains four values which is an even number. This means that the average between the two middle values is the median. In this case, it is 2.5 which is the average between 2 and 3.

Figure 3-2 shows the US CPI year-on-year values for the last twenty years. The higher dashed line is the monthly median calculated over the past twenty years. The lower dashed line symbolizes zero. Basically, this is like Figure 3-1 but instead of the mean, the median is charted.

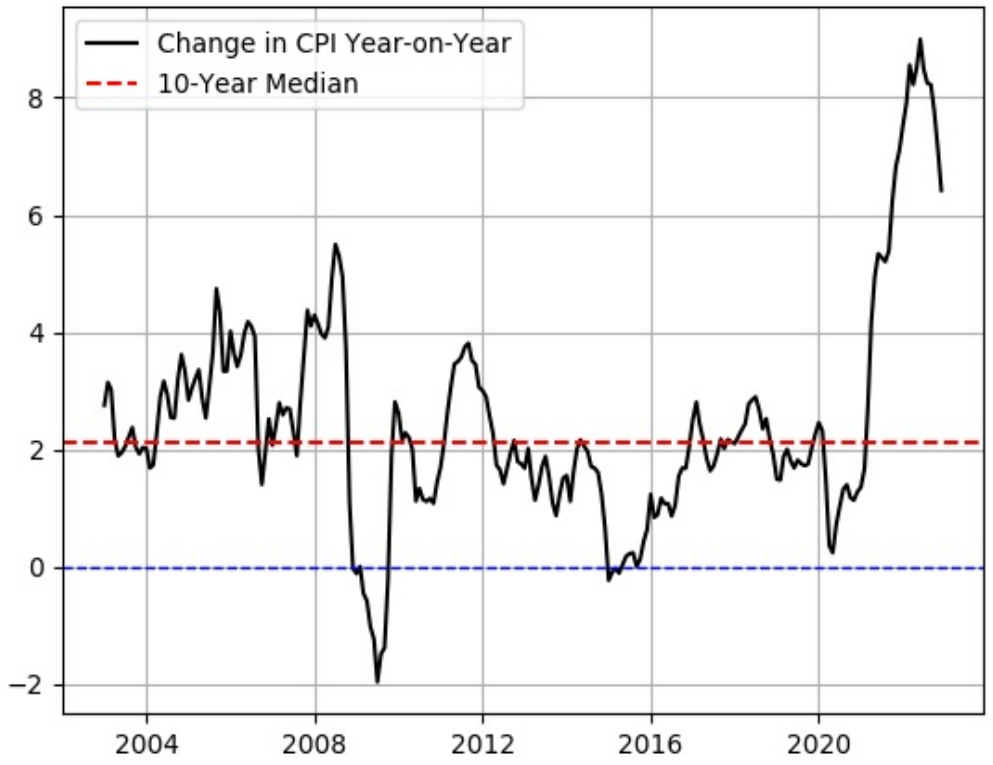


*Figure 3-2. US CPI year-on-year changes for the last twenty years with the median*

You can create Figure 3-2 by using the following code:

```
# Calculating the median of the dataset
median = cpi_last_ten_years["CPIAUCSL"].median()

# Printing the result
print('The median of the dataset: ' + str(median), '%')

# Plotting the latest observations in black with a label
plt.plot(cpi_last_ten_years[:], color = 'black', linewidth = 1.5,
label = 'Change in CPI Year-on-Year')

plt.axhline(y = median, color = 'red', linestyle = 'dashed', label =
'10-Year Median')
plt.axhline(y = 0, color = 'blue', linestyle = 'dashed', linewidth =
1)

# Calling a grid to facilitate the visual component
plt.grid()
```

```
# Calling the legend function so it appears with the chart
plt.legend()
```

The output of the median should be as follows:

```
The median of the dataset: 2.1143 %
```

Clearly, the median is less impacted by the recent outliers that are coming from unusual environments. The median is around 2.10% which more in line with the implied target of 2.00%.

> **NOTE**
>
> Remember that Chapter 6 will give you all you need to know about the Python snippets you are seeing in this chapter, so you don't need to worry if you are missing out on the coding concepts.

The last central tendency measure in this section is the mode. The *mode* is the value that is the most frequently observed (but also the least used in data analysis).

Let's take a simple example of two datasets. Suppose you want to calculate the mode on the following datasets. How would you do it?

- Dataset A = [1, 2, 2, 4, 5]
- Dataset B = [1, 2, 3, 4]
- Dataset C = [1, 1, 2, 2, 3]

Dataset A contains two times the value 2 which makes it the mode. Dataset B doesn't have a mode as every value is observed once. Dataset C is multimodal since it contains more than one mode (which are 1 and 2).

> **NOTE**
>
> The mode is useful with categorical variables (like credit rankings) as opposed to continuous variables (like price and returns time series)

You are unlikely to use the mode in analyzing time series as the mean and the median are more useful. To list a few examples that use the mean and the median in financial analysis:

- Calculating a moving mean (average) on the price data to detect the underlying trend.
- Calculating a rolling median on a price-derived indicator to know its neutral zone.
- Calculating the expected return of a security using the historical mean.
- Checking for the normality of the returns distribution by comparing the mean to the median.

The discussion on central tendency metrics is very important especially that the mean and the median are heavily used not only as standalone indicators but also as ingredients in more complex measures.

**NOTE**

The key takeaways from this section are as follows:

- There are mainly three central tendency measures: the mean, the median, and the mode.
- The mean is the sum divided by the quantity while the median is the value that splits the data in half. The mode is the most frequent value in the dataset.

## Measures of Variability

*Measures of variability* describe how spread out the values in a dataset are relative to the central tendency measures. The first and most known measure of variability is the variance. The *variance* describes a set of numbers' variability from their mean. The idea behind the variance's formula is to determine how far away from the mean each data point is, then square those deviations to make sure that all numbers are positive (this is because distance cannot be negative).

The formula to find the variance is as follows:

$$\sigma^2 = \frac{1}{n} \sum_{n}^{i=1} \left(x_i - \overline{x}\right)^2$$

The intuition behind this formula is to calculate the sum of the squared deviations of each data point from the mean thus giving different distance observations and then calculating the mean of these distance observations.

Let’s take a simple example of two datasets. Suppose you want to calculate the variance on dataset A and dataset B. How would you do it?

- Dataset A = [1, 2, 3, 4, 5]
- Dataset B = [5, 5, 5, 5]

The first step is to calculate the mean of the dataset as that is the benchmark from where you will calculate the dispersion of the data. Dataset A has a mean of 3. The next step is to use the variance formula step by step as follows:

$$(x_1 - \overline{x})^2 = (1 - 3)^2 = 4$$

$$(x_2 - \overline{x})^2 = (2 - 3)^2 = 1$$

$$(x_3 - \overline{x})^2 = (3 - 3)^2 = 0$$

$$(x_4 - \overline{x})^2 = (4 - 3)^2 = 1$$

$$(x_5 - \overline{x})^2 = (5 - 3)^2 = 4$$

The previous results are summed up as follows:

$$4 + 1 + 0 + 1 + 4 = 10$$

And finally, the result is divided by the quantity of the observations to find the variance:

$$\sigma^2 = \frac{10}{5} = 2$$

As for dataset B, you should think about it intuitively. If the observations are all equal, they all represent the dataset which also means that they are their own mean. What would you say about the variance of the data in this case, considering that all the values are equal to the mean?

If your response is that the variance is zero, then you are correct. Mathematically, you can calculate it as follows:

$$(x_1 - \overline{x})^2 = (5 - 5)^2 = 0$$

$$(x_2 - \overline{x})^2 = (5 - 5)^2 = 0$$
$$(x_3 - \overline{x})^2 = (5 - 5)^2 = 0$$
$$(x_4 - \overline{x})^2 = (5 - 5)^2 = 0$$

The previous results sum up to zero and if you divide zero by 4 (the quantity of the dataset), you will get zero. Intuitively, there is no variance because the all the values are constant and they do not deviate from their mean.

$$\sigma^2 = \frac{0}{5} = 0$$

You can calculate the variance in Python using the following code:

```
# Calculating the variance of the dataset
variance = cpi_last_ten_years["CPIAUCSL"].var()

# Printing the result
print('The variance of the dataset: ' + str(variance), '%')
```

The output of the variance should be as follows:

```
The variance of the dataset: 3.6248 %
```

There is a flaw nonetheless, and it is that the variance represent squared values and are not comparable to the mean since they use different units. This is easily fixable by taking the square root of the variance. Doing so brings the next measure of variability, the *standard deviation.* It is the square root of the variance and is the average deviation of the values from the mean.

A low standard deviation indicates that the values tend to be close to the mean (low volatility), while a high standard deviation indicates that the values are spread out over a wider range relative to their mean (high volatility).

**NOTE**

The words standard deviation and volatility are used interchangeably. They refer to the same thing.

The formula to find the standard deviation is as follows:

$$\sigma = \sqrt{\frac{1}{n}\sum_{n}^{i=1}(x_i - \overline{x})^2}$$

If you consider the previous examples with the variance, then the standard deviation can be found as follows:

$$\sigma_{Dataset\ A} = \sqrt{2} = 1.41$$

$$\sigma_{Dataset\ B} = \sqrt{0} = 0$$

Standard deviation is commonly used with the mean since they use the same units. You will soon understand the importance of this stat when I discuss the normal distribution function, a key concept in descriptive statistics.

You can calculate the standard deviation in Python using the following code:

```
# Calculating the standard deviation of the dataset
standard_deviation = cpi_last_ten_years["CPIAUCSL"].std()

# Printing the result
print('The standard deviation of the dataset: ' +
str(standard_deviation), '%')
```

The output of the standard deviation should be as follows:

```
The standard deviation of the dataset: 1.9039 %
```

How are you supposed to interpret the standard deviation? On average, the CPI year-on-year values tend to be ±1.90% from the mean of the same period which is at 2.48%.

In the coming section, you will see how to make better use of standard deviation numbers. The last measure of variability in this section is the range. The *range* is a very simple stat that shows the distance between the greatest value and the lowest value in the dataset. This gives you quick glance about the two historical extreme values. The range is used in the normalization formula that you will see later chapters. The formula to find the range is as follows:

$$Range = max(x) - min(x)$$

Let’s take the same example and calculate the range. In Python, you can easily do this as there are built-in functions that show the maximum and the minimum

value given a set of data:

```
# Calculating the range of the dataset
range_metric = max(cpi["CPIAUCSL"]) - min(cpi["CPIAUCSL"])

# Printing the result
print('The range of the dataset: ' + str(range_metric), '%')
```

The output of the following code should be as follows:

```
The range of the dataset: 16.5510 %
```

Figure 3-3 shows the CPI values since 1950. The diagonal dashed line represents the range.

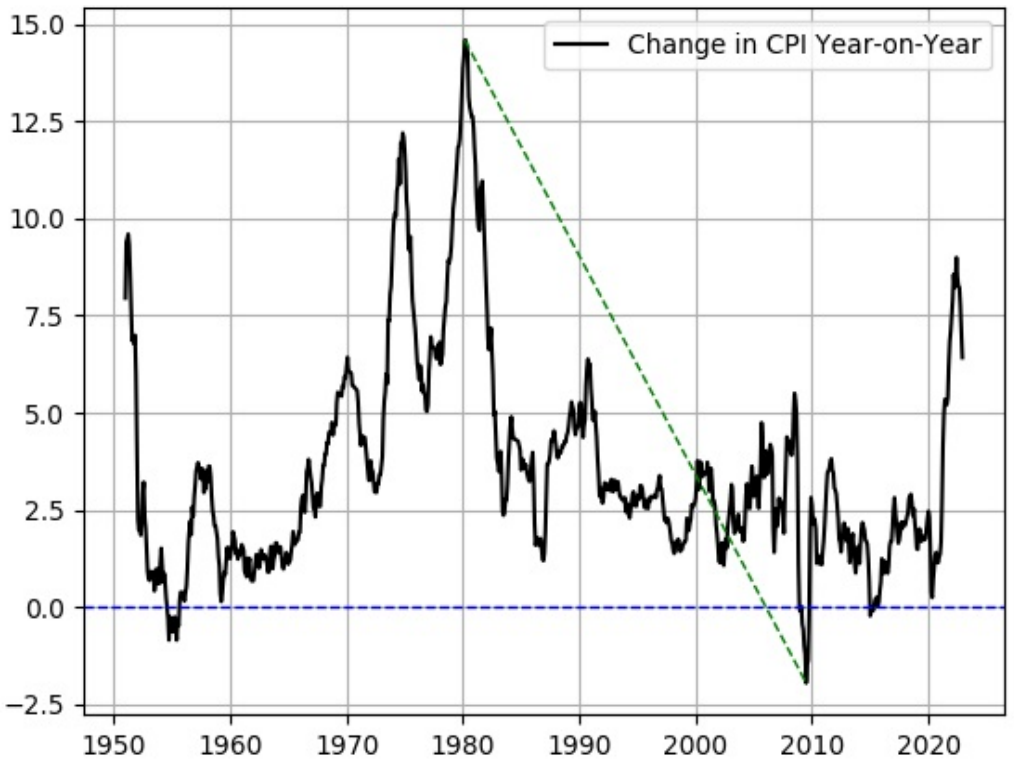


*Figure 3-3. US CPI year-on-year change since 1950 with a diagonal dashed line that represents the range*

The range of the CPI shows the size of the variations in inflation measures from one period to another considering that the range occurred in 30 years. Yearly

changes in inflation numbers vary from one country to another. Generally, developed world countries such as France and the United States have stable variations (in times of stability) while emerging and frontier world countries such as Argentina and Turkey have more volatile and more extreme inflation numbers.

**NOTE**

Make sure to retain the following points as you continue to the next section:

- Three key variability metrics that you should know are the variance, the standard deviation, and the range.
- The standard deviation is the square root of the variance. This is done so that it becomes comparable to the mean.
- The range is the difference between the highest and the lowest value in a dataset. It is a quick snapshot of the overall volatility of the observations.

## Measures of Shape

Measures of shape describe the distribution of the values around the central tendency measures in a dataset.

The mean and the standard deviation are the two factors that describe the normal distribution. The standard deviation depicts the spread or dispersion of the data and the mean reflects the distribution's center.

A *probability distribution* is a mathematical function that describes the likelihood of different outcomes or events in a random experiment. In other words, it gives the probabilities of all possible values of a random variable.

There are many types of probability distributions, including discrete and continuous distributions. *Discrete distributions* take on a finite number of values. The most known discrete distributions are the Bernoulli distribution, Binomial distribution, and Poisson distribution.

*Continuous distributions* are used for random variables that can take on any value within a given range (such as stock prices). The most known continuous distribution is the normal distribution.

The normal distribution (also known as the Gaussian distribution) is a type of continuous probability distribution that is symmetrical around the mean and has a bell shape. It is one of the most widely used distributions in statistical analysis and is often used to describe natural phenomena such as age, weight, and test scores. Figure 3-4 shows the shape of a normal distribution.

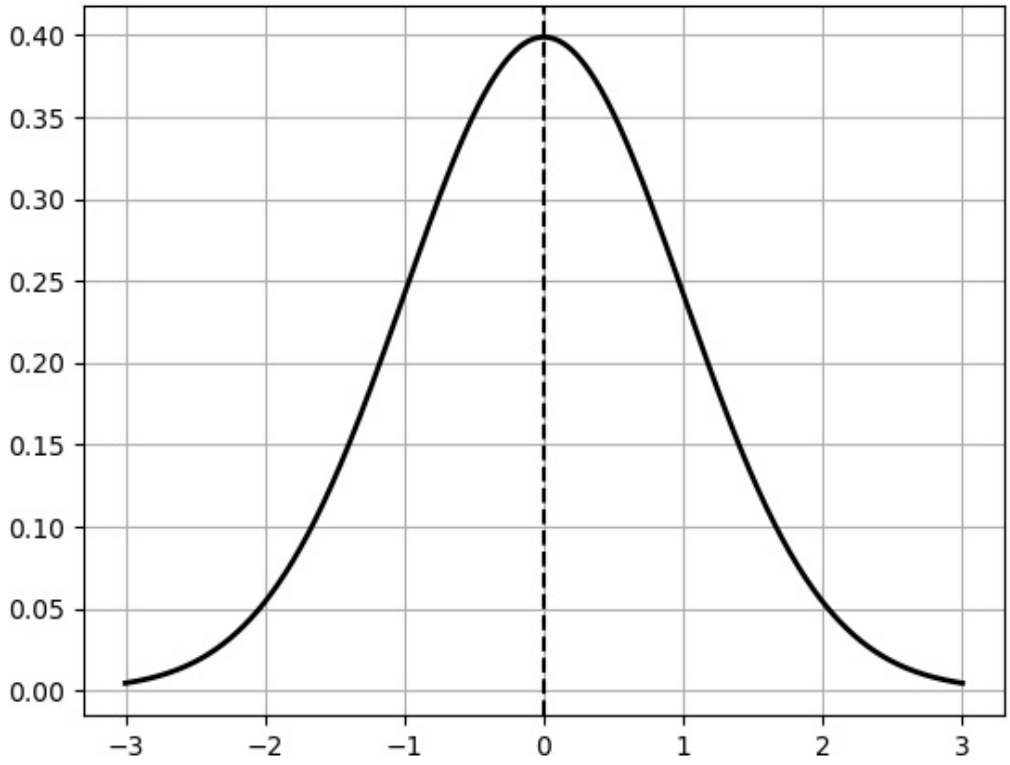


*Figure 3-4. A normal distribution plot with mean = 0 and standard deviation = 1*

You can generate Figure 3-4 using the following code block:

```
# Importing libraries
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as stats

# Generate data for the plot
data = np.linspace(-3, 3, num = 1000)

# Define the mean and standard deviation of the normal distribution
mean = 0
```

```
std = 1

# Generate the function of the normal distribution
pdf = stats.norm.pdf(data, mean, std)

# Plot the normal distribution plot
plt.plot(data, pdf, '-', color = 'black', lw = 2)
plt.axvline(mean, color = 'black', linestyle = '--')

# Calling a grid to facilitate the visual component
plt.grid()

# Show the plot
plt.show()
```

**NOTE**

Since normally distributed variables are common, most statistical tests and models assume that the analyzed data is normal. With financial returns, they are assumed normal even though they experience a form of skew and kurtosis, two measures of shape discussed in this section.

In a normal distribution, the data is distributed symmetrically around the mean which also means that the mean is equal to the median and to the mode. Furthermore, around 68% of the data fall within one standard deviation of the mean, around 95% fall within two standard deviations, and around 99.7% fall within three standard deviations. This property makes the normal distribution a useful tool for making inferences.

To sum up, what you should retain from the normal distribution is the following:

- The mean and the standard deviation describe the distribution.
- The mean splits the distribution halfway making it equal to the median. Due to the symmetrical property, the mode is also equal to the mean and the median.

Now, let’s discuss the measures of shape. The first measure of shape is skewness. *Skewness* describes a distribution’s asymmetry. It analyzes how far from being symmetrical the distribution deviates.

As you may have already understood, the skewness of a normal distribution is equal to zero. This means that the distribution is perfectly symmetrical around its

mean, with an equal number of data points on either side of the mean.

A *positive skew* indicates that the distribution has a long tail to the right which means the the mean is greater than the median due to the fact that the mean is sensible to outliers which will push it upwards (therefore, to the right of the x-axis). Similarly, the mode which represents the most frequent observations will be the lowest value between the three central tendency measures. Figure 3-5 shows a positive skew.

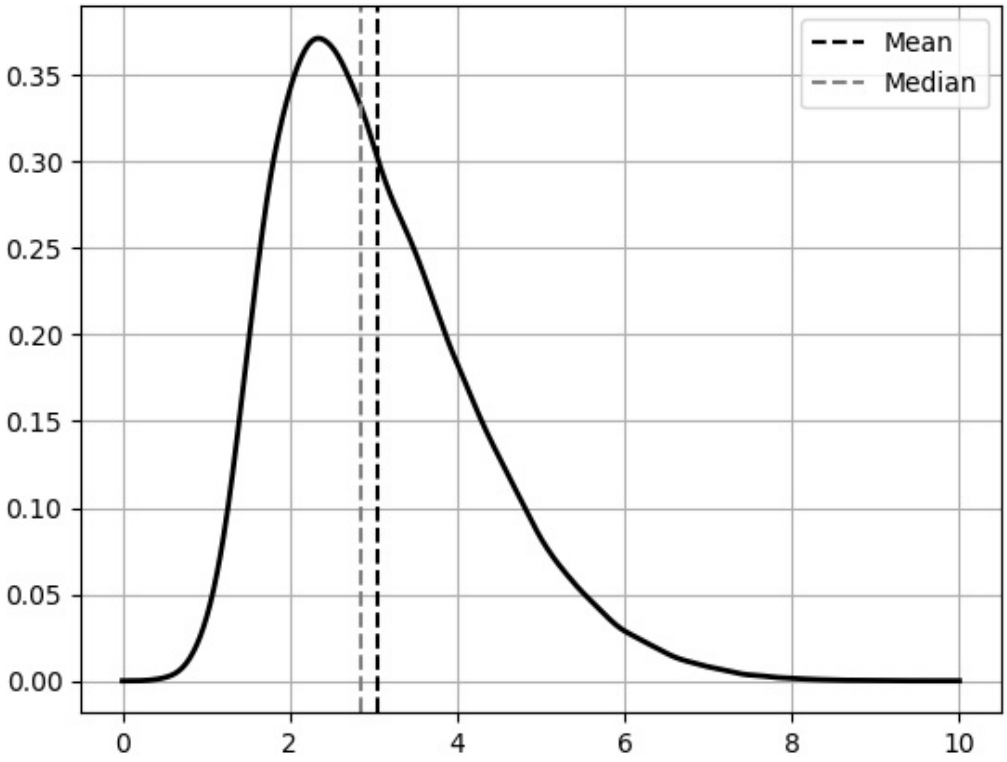


*Figure 3-5. An example of a positively skewed distribution*

A *negative skew* indicates that the distribution has a long tail to the left which means the the mean is lower than the median for the reasons mentionned when discussing the positive skew. Similarly, the mode will be the greatest value between the three central tendency measures. Figure 3-6 shows a negative skew.

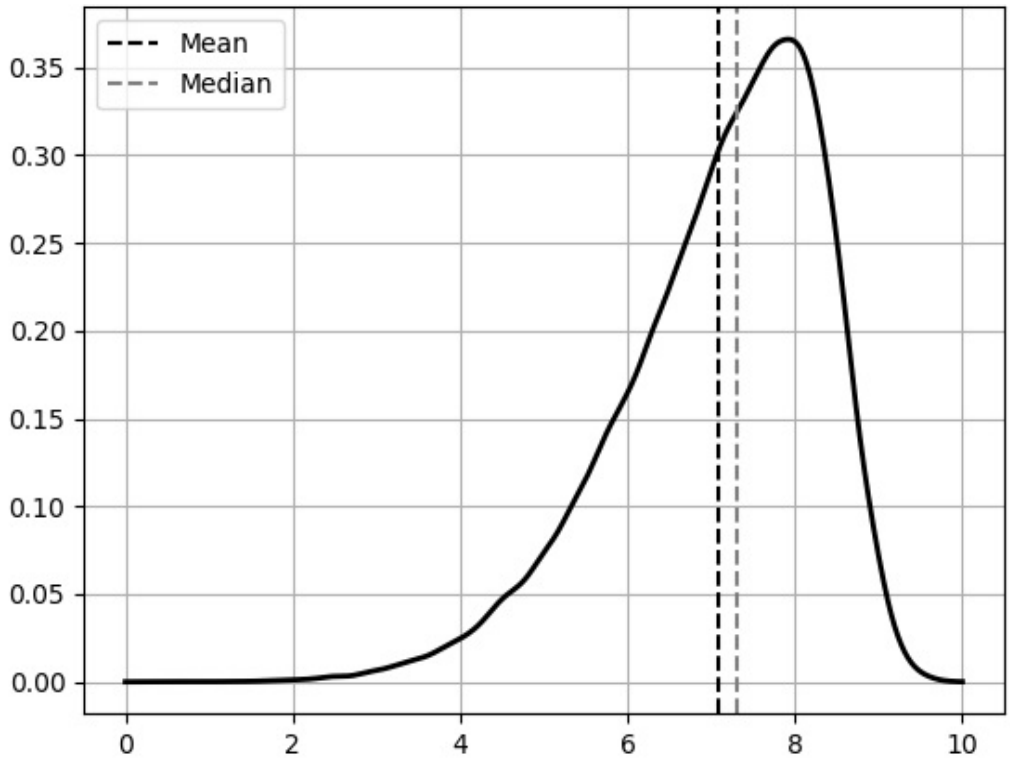


*Figure 3-6. An example of a negatively skewed distribution*

**NOTE**

How can skewness be interpreted in the world of financial markets? If the distribution is positively skewed, it means that there are more returns above the mean than below it (the tail of the distribution is longer on the positive side).

On the other hand, if the distribution is negatively skewed, it means that there are more returns below the mean than above it (the tail of the distribution is longer on the negative side).

The skew of a returns series can provide information about the risk and return of an investment. For example, a positively skewed returns series may indicate that the investment has a higher potential for high returns, but also a higher risk of large losses. In contrast, a negatively skewed returns series may indicate that the investment has a lower potential for high returns, but also a lower risk of large losses.

The formula to find skewness is as follows:

$$\tilde{\mu}_3 = \frac{\sum_{n=1}^{i} (x_i - \overline{x})^3}{N\sigma^3}$$

The formula divides the third central moment by the standard deviation to the power of three. Let's check the skewness of the US CPI year-on-year data:

```
# Calculating the skew of the dataset
skew = cpi["CPIAUCSL"].skew()

# Printing the result
print('The skew of the dataset: ' + str(skew))
```

The output of the following code should be as follows:

```
The skew of the dataset: 1.4639
```

The skew of the data is 1.46 but what does that mean? Let's chart the distribution of the data so that the interpretation becomes easier. You can do this using the following code snippet:

```
# Plotting the histogram of the data
fig, ax = plt.subplots()
ax.hist(cpi['CPIAUCSL'], bins = 30, edgecolor = 'black', color =
'white')

# Add vertical lines for better interpretation
ax.axvline(mean, color='black', linestyle='--', label='Mean',
linewidth = 2)
ax.axvline(median, color='grey', linestyle='-.', label='Median',
linewidth = 2)

# Calling the grid function for better interpretability
plt.grid()

# Calling the legend function to show the labels
plt.legend()

# Showing the plot
plt.show()
```

Figure 3-7 shows the result of the previous code snippet. The data is clearly positively skewed since the mean is greater than the median and the skewness is positive (above zero).

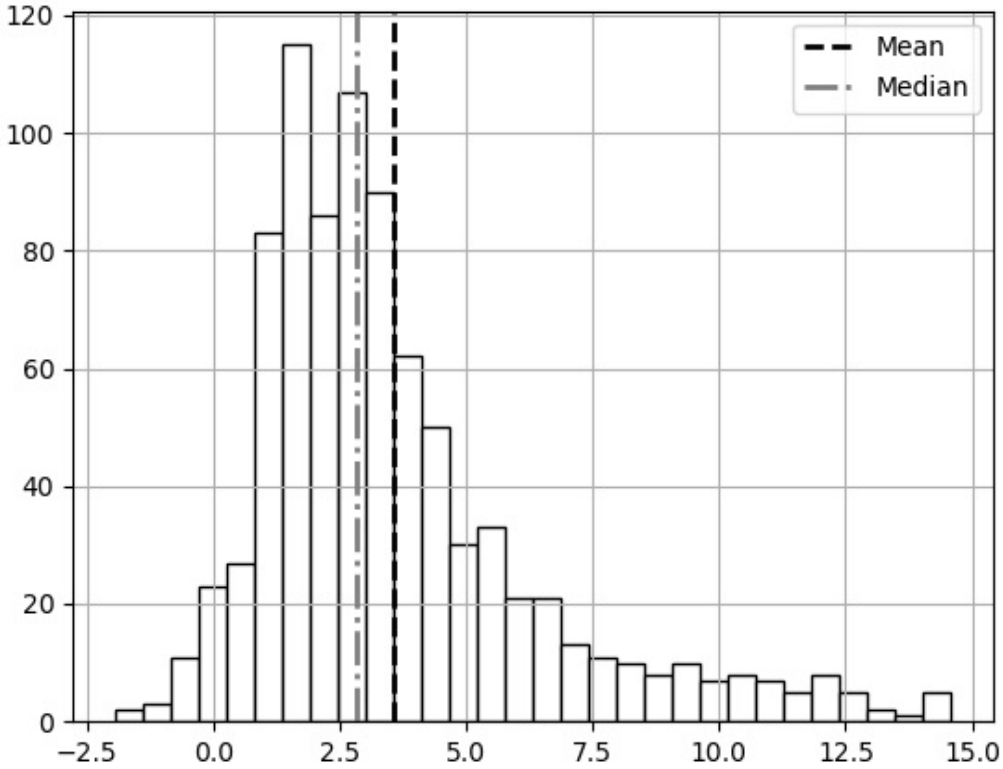


*Figure 3-7. Data distribution of the US CPI year-on-year*

Remember, skewness is a measure of the asymmetry of a probability distribution. It therefore, measures the degree to which the distribution deviates from normality. The rules of thumb to interpret skewness are as follows:

- If skewness is between -0.5 and 0.5, the data is considered symmetrical.
- If skewness is between -1.0 and – 0.5 or between 0.5 and 1.0, the data is considered mildly skewed.
- If skewness is less than -1.0 or greater than 1.0, the data is considered highly skewed

What does a positive skew mean? 1.17 is a highly skewed data (in the positive side) which is in line with a monetary policy that favors inflation as the economy grows (with a few inflationary spikes that cause the skew).

> **NOTE**
>
> It may be interesting to know that with a skewed distribution, the median is the preferred metric since the mean tends to be pulled by outliers, thus distorting its value.

The next measure of shape is *kurtosis* which a measure of the peakedness or flatness of a distribution relative to a normal distribution. Kurtosis describes the tails of a distribution, in particular, whether the tails are thicker or thinner than those of a normal distribution. Mathematically, kurtosis is the fourth central moment divided by the fourth power of the standard deviation.

A normal distribution has a kurtosis of 3, which means it is a mesokurtic distribution. If a distribution has a kurtosis greater than 3, it is referred to as leptokurtic, meaning it has a higher peak and fatter tails than a normal distribution. If a distribution has a kurtosis less than 3, it is referred to as platykurtic, meaning it has a flatter peak and thinner tails than a normal distribution.

The formula to find kurtosis is as follows:

$$k = \frac{\sum_{n=1}^{i} (x_i - \overline{x})^4}{N\sigma^4}$$

Sometimes, kurtosis is measured as excess kurtosis to give it a starting value of zero (for a normal distribution). This means that the kurtosis measure is subtracted from 3 so as to calculate the excess kurtosis. Let's calculate excess kurtosis for the US CPI year-on-year data:

```
# Calculating the excess kurtosis of the dataset
excess_kurtosis = cpi["CPIAUCSL"].kurtosis()

# Printing the result
print('The excess kurtosis of the dataset: ' + str(excess_kurtosis))
```

The output of the following code should be as follows:

```
The excess kurtosis of the dataset: 2.2338
```

The excess kurtosis obtained from `pandas` should be zero in the case of a normal distribution. In the case of the US CPI year-on-year values, it is 2.23

which is more in line with a leptokurtic (peakier with fatter tails) distribution. A positive value indicates a distribution more peaked than normal and a negative kurtosis indicates a shape flatter than normal.

**NOTE**

Independent from statistics, it is interesting to know the terminology of what you are analyzing. *Inflation* is the decrease in the purchasing power of the economic agents (such as households). The decrease in the purchasing power means that agents can buy less over time with the same amount of money, otherwise referred to as a general price increase. Inflation in the economic sense has the following forms:

- *Inflation*: Controlled inflation is associated with a steady economic growth and expasion. It is a desired attribute for a growing economy. Regulators monitor inflation and try to stabilize it in order to prevent social and economic issues.
- *Deflation*: Whenever inflation is in the negative territory, it is referred to as deflation. Deflation is very dangerous for the economy and as tempting as it may be for consumers who see a price decrease, deflation is a growth killer and may cause extended economic gluts which lead to unemployment and bearish stock markets.
- *Stagflation*: This occurs where inflation is either high or rising while the economic growth is slowing down. Simultaneously, unemployment remains high. It is one of the worst possible case scenarios.
- *Disinflation*: This is a decrease in inflation but in the positive territory. For example, if this year's inflation is 2% while last year's inflation is 3%, you can say that there was a disinflation situation on a yearly basis.
- *Hyper inflation*: This is the nightmarish scenario that occurs when inflation goes out of control and experiences astronomical percent changes such as millions of percentage change from year to year (famous cases include Zimbabwe, Yugoslavia, and Greece).

Finally, one last metric to see in the descriptive statistics department, the quantiles. *Quantiles* are measures of both shape and variability since they provide information about the distribution of values (shape) and provide information about the dispersion of such values (variability). The most used type of quantiles are called quartiles.

*Quartiles* divide the dataset into four equal parts. This is done by arranging the data in order and then performing the split. Consider table 3-1 as an example:

| Value |
| --- |
| |

| |
|---|
| 1 |
| 2 |
| 4 |
| 5 |
| 7 |
| 8 |
| 9 |

The quartiles are as follows:

- The lower quartile (Q1) is the first quarter which in this case is 2.
- The midde quartile (Q2) which is also the median which in this case is 5.
- The upper quartile (Q3) in this case is 8.

Mathematically, you can calculate Q1 and Q3 using the following formulas:

$$Q_1 = \left(\frac{n+1}{4}\right)$$

$$Q_3 = 3\left(\frac{n+1}{4}\right)$$

Keep in mind that the result of the formulae gives you the ranking of the values but not the values themselves:

$$Q_1 = \left(\frac{7+1}{4}\right) = 2^{nd}\ term = 2$$

$$Q_3 = 3\left(\frac{7+1}{4}\right) = 6^{th}\ term = 8$$

The *interquartile range* (IQR), is the difference between Q3 and Q1 and provides a measure of the spread of the middle 50% of the values in a data set. The IQR is robust to outliers (since it relies on middle values) and provides a brief summary of the spread of the bulk of the values. The IQR of the data in

table 3-1 is 6 as per the following formula:

$$IQR = Q_3 - Q_1$$

$$IQR = 8 - 2 = 6$$

The IQR is a valuable indicator and can be used as an input or a risk metric in many different models. It can also be used to detect outliers in the data since it is immune to them. Also, the IQR can help evaluate the current volatility of the analyzed asset which in turn can be used with other methods to create more powerful models. As understood, the IQR outperforms the range metric in terms of usefulness and interpretation as the former is prone to outliers.

Be careful with calculating quartiles as there are many methods that use different calculations for the same dataset. The most important thing is to use a consistent way all throughout your analyses. The method used to calculate the quartiles in table 3-1 is called the *Tukey's hinges* method. By default, when you want to calculate the quartiles using `pandas`, the default method is the *linear interpolation* method which will give different results.

The key diffences between the methods is that some may fit better with smaller datasets or normal-sized datasets with different distribution characteristics.

**NOTE**

The key takeways from this section are the following:

- The normal distribution is a continuous probability distribution that has a bell-shaped curve. The majority of the data cluster around the mean. The mean, median, and the mode of a normal distribution curve are all equal.
- Skewness measures the asymmetry of a probability distribution.
- Kurtosis measures the peakedness of a probability distribution. Excess kurtosis is commonly used to describe the current probability distribution.
- Quantiles divide the arranged dataset into equal parts. The most famous quantiles are quartiles which divide the data into four equal parts.
- The IQR is the difference between the third quartile and the first quartile. It is immune to outliers and thus, very helpful in data analysis.

## Visualizing Data

If you remember from the previous chapter, I have presented a six-phase process in data science. Phase four dealt with data visualization. This section will show you a few ways to present the data in a clear visual manner that allows you to interpret it.

There are many types of statistical plots that are commonly used to visualize data such as scatter plots and line plots. Let's discuss these plots and create them using the same inflation data.

The first data visualization method is *scatter plots*, which are used to graph the relationship between two variables through points that correspond to the intersection between the variables. Let's create and visualize a scatter plot using the following code:

```
# Importing the required library
import pandas_datareader as pdr
import pandas as pd
import matplotlib.pyplot as plt

# Setting the beginning and end of the historical data
start_date = '1950-01-01'
end_date   = '2022-12-01'

# Creating a dataframe and downloading the CPI data using its code
name and its source
cpi = pdr.DataReader('CPIAUCSL', 'fred', start_date, end_date)

# Transforming the CPI into a year-on-year measure
cpi = cpi.pct_change(periods = 12, axis = 0) * 100

# Dropping the NaN values
cpi = cpi.dropna()

# Resetting the index
cpi = cpi.reset_index()

# Creating the chart
fig, ax = plt.subplots()
ax.scatter(cpi['DATE'], cpi['CPIAUCSL'], color = 'black', s = 8,
label = 'Change in CPI Year-on-Year')

# Calling the grid function for better interpretability
plt.grid()

# Calling the legend function to show the labels
plt.legend()
```

```
# Showing the plot
plt.show()
```

Figure 3-8 shows the result of a scatter plot in time. This means that you have the CPI data as the first variable (y-axis) and time as the second variable (x-axis). However, scatter plots are more commonly used to compare variables, thus removing the time variable can give more insights.

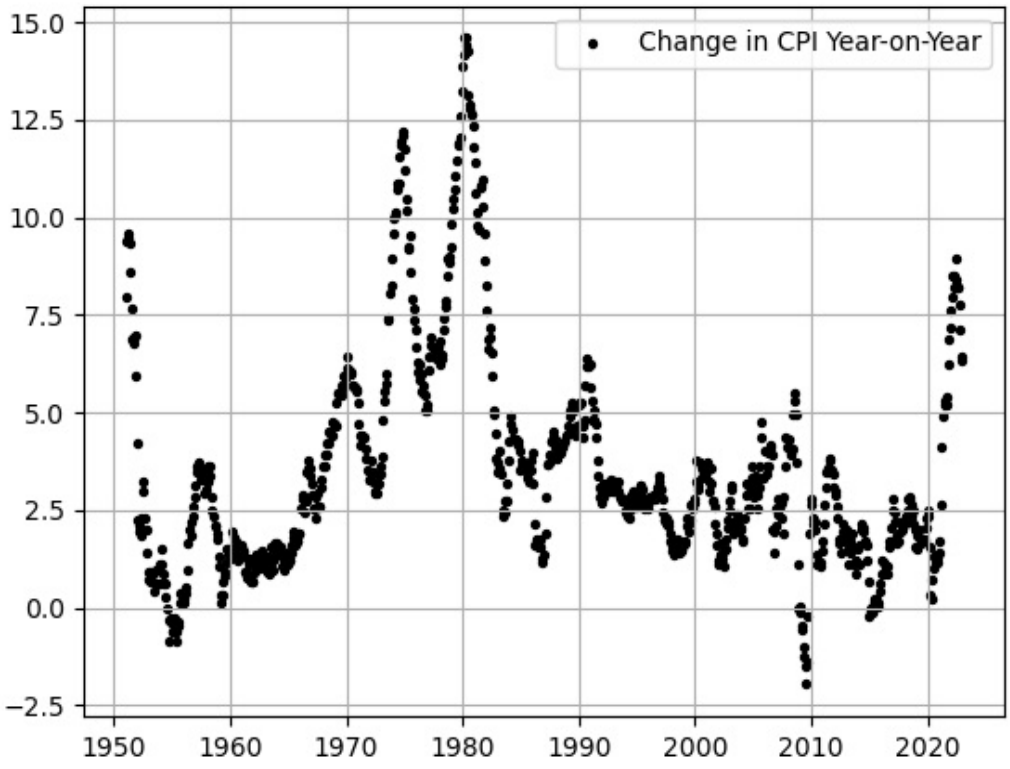


*Figure 3-8. Scatter plot of US CPI versus the time axis*

If you take the UK CPI year-on-year change and want to compare it with the US CPI year-on-year change, you will should get Figure 3-9. Notice the positive association between the two, as higher values of one are correlated with higher values of another. Correlation is a key measure that you will see in detail in the next section. The code to plot Figure 3-9 is as follows:

```
# Setting the beginning and end of the historical data
start_date = '1995-01-01'
```

```
end_date   = '2022-12-01'

# Creating a dataframe and downloading the CPI data using its code
name and its source
cpi_us = pdr.DataReader('CPIAUCSL', 'fred', start_date, end_date)
cpi_uk = pdr.DataReader('GBRCPIALLMINMEI', 'fred', start_date,
end_date)

# Dropping the NaN values from the rows
cpi_us = cpi_us.dropna()
cpi_uk = cpi_uk.dropna()

# Transforming the CPI into a year-on-year measure
cpi_us = cpi_us.pct_change(periods = 12, axis = 0) * 100
cpi_us = cpi_us.dropna()

cpi_uk = cpi_uk.pct_change(periods = 12, axis = 0) * 100
cpi_uk = cpi_uk.dropna()

# Creating the chart
fig, ax = plt.subplots()
ax.scatter(cpi_us['CPIAUCSL'], cpi_uk['GBRCPIALLMINMEI'], color =
'black', s = 8, label = 'Change in CPI Year-on-Year')

# Adding a few aesthetic elements to the chart
ax.set_xlabel('US CPI')
ax.set_ylabel('UK CPI')
ax.axvline(x = 0, color='black', linestyle = 'dashed', linewidth = 1)
# vertical line
ax.axhline(y = 0, color='black', linestyle = 'dashed', linewidth = 1)
# horizontal line
ax.set_ylim(-2,)

# Calling the grid function for better interpretability
plt.grid()

# Calling the legend function to show the labels
plt.legend()

# Showing the plot
plt.show()
```

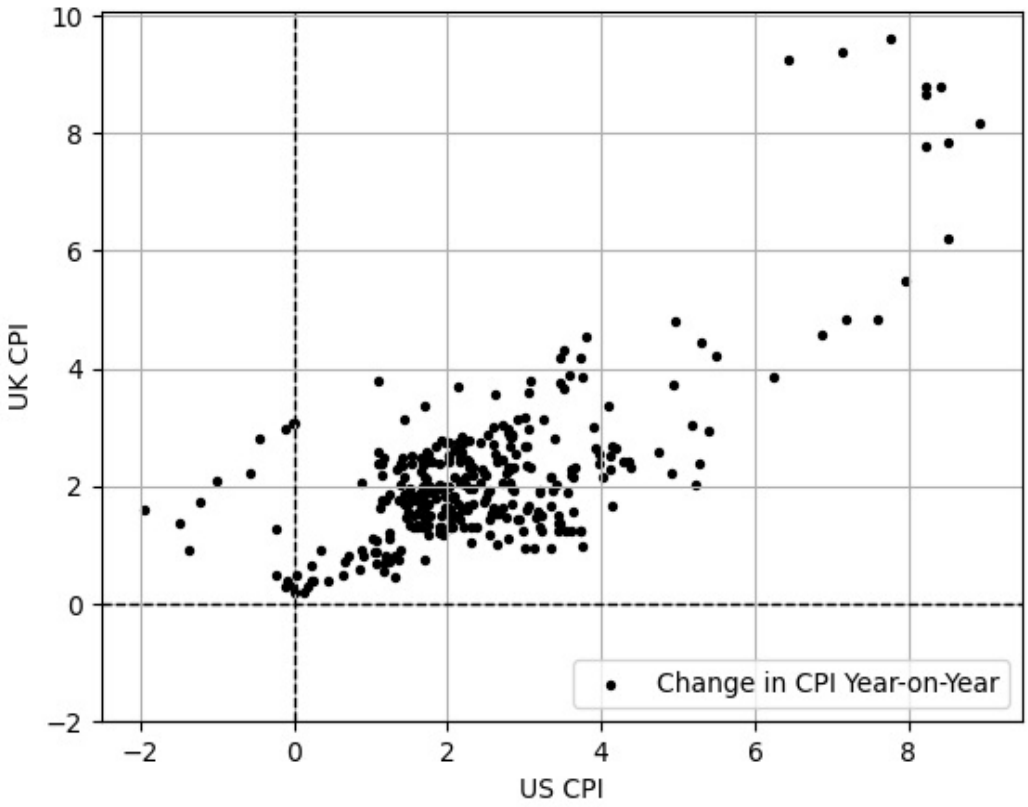


*Figure 3-9. Scatter plot of UK CPI versus US CPI*

Scatter plots are good when visualizing the correlation between data. They are also easy to draw and interpret. Generally, when the points are scattered in such a way that when a diagonal upwards sloping line can be drawn to represent them, the correlation is assumed to be positive since whenever variables on the x-axis increase, variables on the y-axis also increase.

On the other hand, when a diagonal downwards sloping line can be drawn to represent the different variables, a negative correlation may exist. A negative correlation implies that whenever variables on the x-axis move, it is likely that variables on the y-axis move the other way. Figure 3-10 draw a best fit line (generated through code) between the two inflation data. Notice how it is upwards sloping:

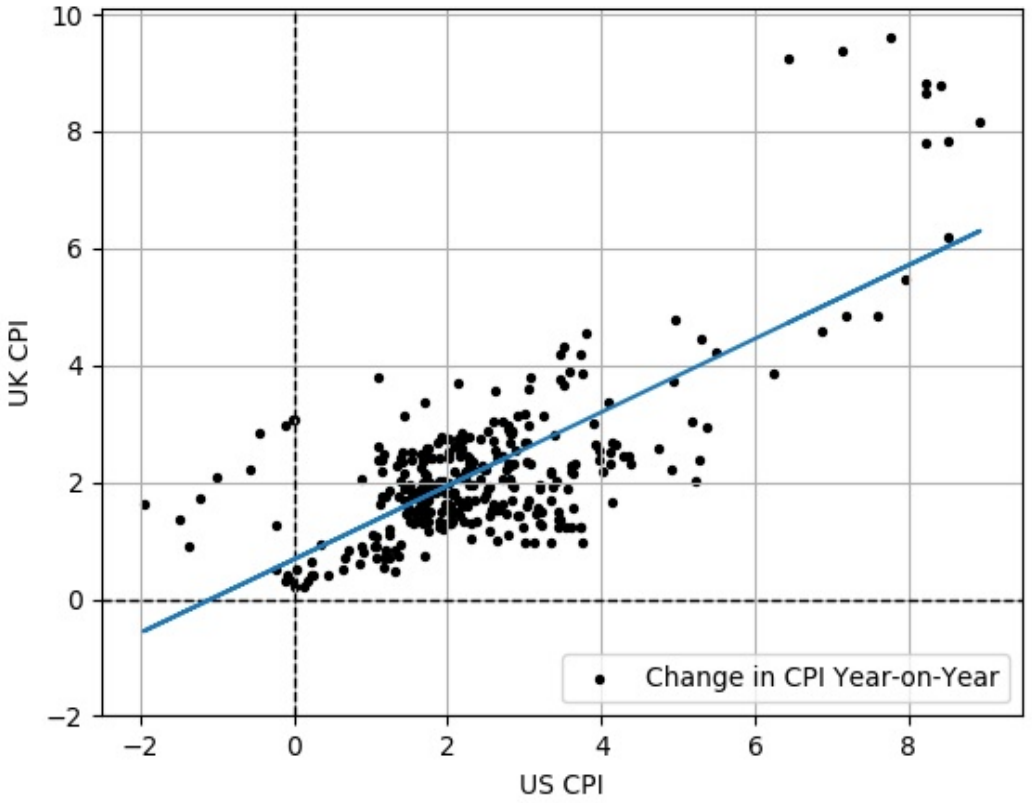


*Figure 3-10. Scatter plot of UK CPI versus US CPI with a best fit line*

Let's now move to another charting method. *Line plots* are basically scatter plots that are joined and are mostly charted against the time axis (x-axis). You have already seen line plots in previous charts such as Figure 3-1 and Figure 3-2 as it is the most basic form of plotting.

The advantage of line plots is their simplicity and ease of implementation. They also show the evolution of the series through time which helps detecting trends and patterns. In Chapter 5, you will see a more elaborate version of plotting financial time series called *candlestick plots*. Figure 3-11 shows a basic line plot on the US CPI since 1950:

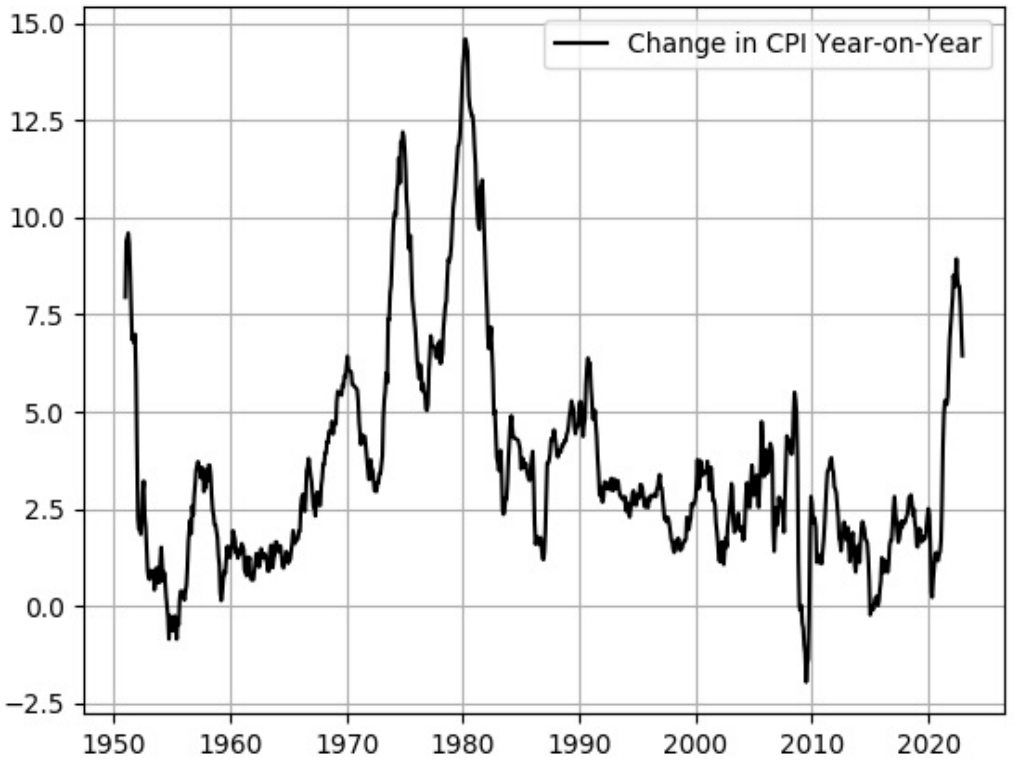


*Figure 3-11. Line plot of US CPI versus the time axis*

To create Figure 3-11, you can use the following code snippet:

```
# Setting the beginning and end of the historical data
start_date = '1950-01-01'
end_date   = '2022-12-01'

# Creating a dataframe and downloading the CPI data using its code
name and its source
cpi = pdr.DataReader('CPIAUCSL', 'fred', start_date, end_date)

# Transforming the CPI into a year-on-year measure
cpi = cpi.pct_change(periods = 12, axis = 0) * 100

# Dropping the NaN values
cpi = cpi.dropna()

# Resetting the index
cpi = cpi.reset_index()

# Creating the chart
```

```
plt.plot(cpi['DATE'], cpi['CPIAUCSL'], color = 'black', label =
'Change in CPI Year-on-Year')

# Calling the grid function for better interpretability
plt.grid()

# Calling the legend function to show the labels
plt.legend()

# Showing the plot
plt.show()
```

Next up are *bar plots* which display the distribution of variables (generally, categorical). Figure 3-12 shows a bar plot on the US CPI since the beginning of 2022:

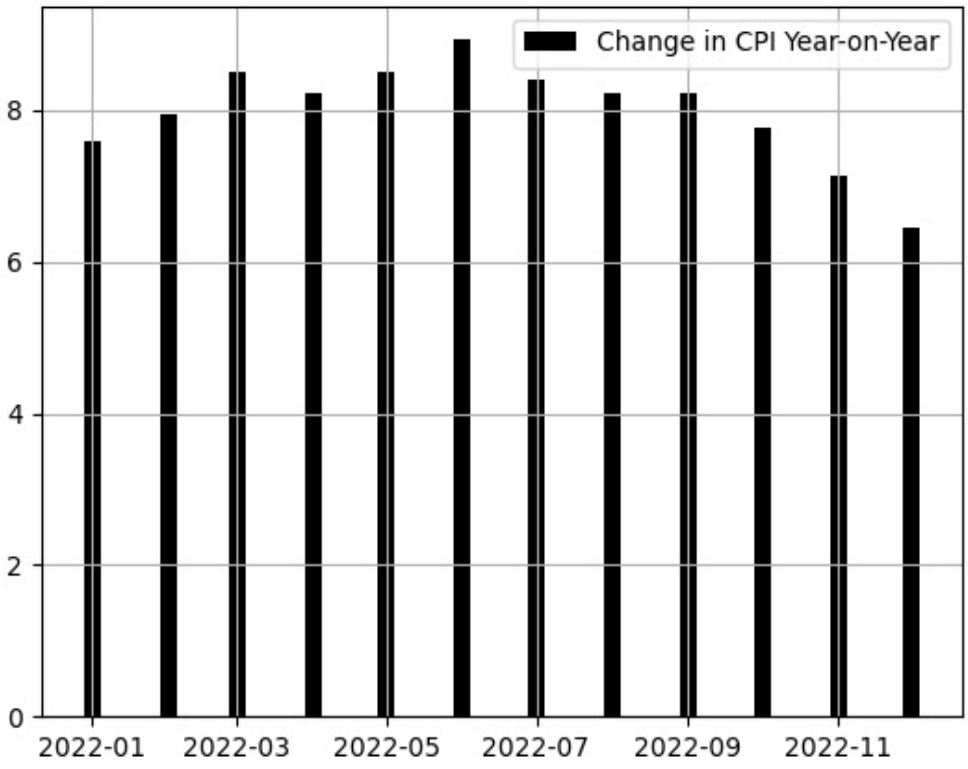


*Figure 3-12. Bar plot of US CPI versus the time axis*

To create Figure 3-12, you can use the following code snippet:

```
# Taking the values of the previous twelve months
```

```
cpi_one_year = cpi.iloc[-12:]

# Creating the chart
plt.bar(cpi_one_year['DATE'], cpi_one_year['CPIAUCSL'], color =
'black', label = 'Change in CPI Year-on-Year', width = 7)

# Calling the grid function for better interpretability
plt.grid()

# Calling the legend function to show the labels
plt.legend()

# Showing the plot
plt.show()
```

Bar plots are easy to implement and are versatile. However, they can be limited for plotting continuous data such as the US CPI or stock prices. They can also be misleading when the scale is off. Bar plots are also not recommended for large datasets since they clutter up the space. For the latter reason, histograms are a better fit.

A *histogram* is a specific sort of bar chart that is used to display the frequency distribution of continuous data by using bars to represent statistical information. It indicates the number of observations that fall into the class or bin of values. An example of a histogram is Figure 3-13 (and Figure 3-7 from the last section):

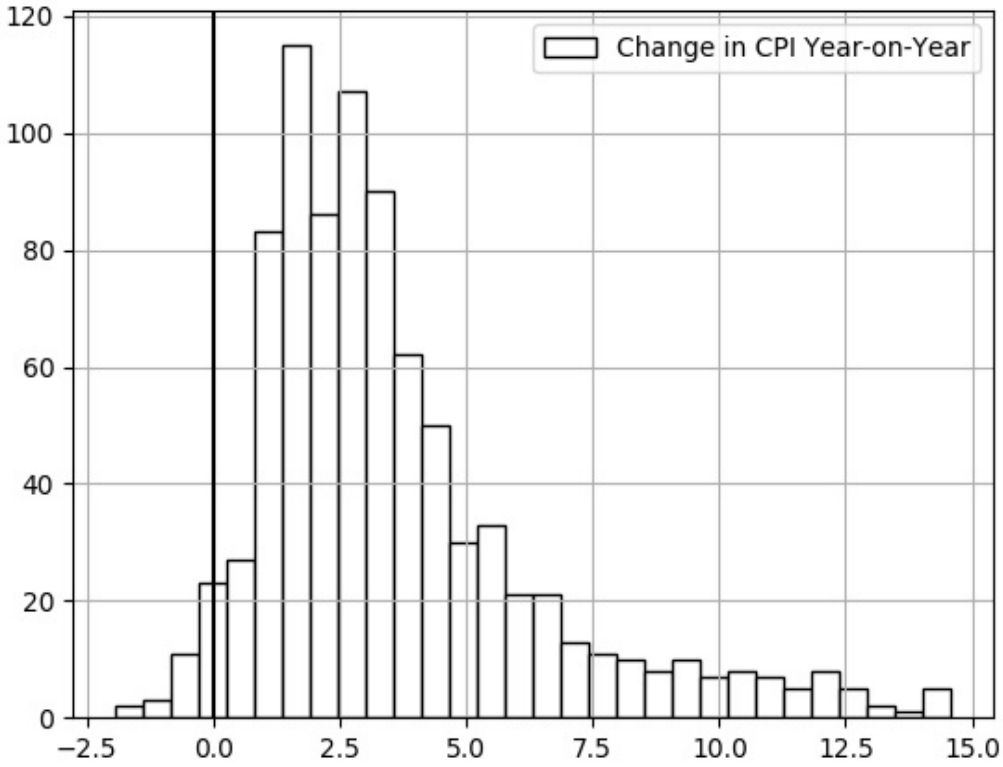


*Figure 3-13. Histogram plot of US CPI*

```
# Creating the chart
fig, ax = plt.subplots()
ax.hist(cpi['CPIAUCSL'], bins = 30, edgecolor = 'black', color =
'white', label = 'Change in CPI Year-on-Year',)

# Add vertical lines for better interpretation
ax.axvline(0, color='black')

# Calling the grid function for better interpretability
plt.grid()

# Calling the legend function to show the labels
plt.legend()

# Showing the plot
plt.show()
```

Notice how the bar plot is charted against the time axis while the histogram does not have a time horizon because it is a group of values with the aim of showing

the overall distribution points. Visually, you can see the positive skewness of the distribution.

> **NOTE**
>
> An example of a *categorical variable* is gender while an example of a *continuous variable* is a commodity's price.

Another classic plotting technique in statistics is the famous *box and whisker plot*. It used to visualize the distribution of continuous variables while including the median and the quartiles, as well as the outliers. The way to understand the box and whisker plot is as follows:

- The box represents the IQR. The box is drawn between the first quartile and the third quartile. The height of the box indicates the spread of the data in this range.
- The line inside the box represents the median.
- The whiskers extend from the top and bottom of the box to the highest and lowest data points that are still within 1.5 times the IQR. These data points are called *outliers* and are represented as individual points on the plot.

Figure 3-14 shows a box and whisker plot on the US CPI since 1950:

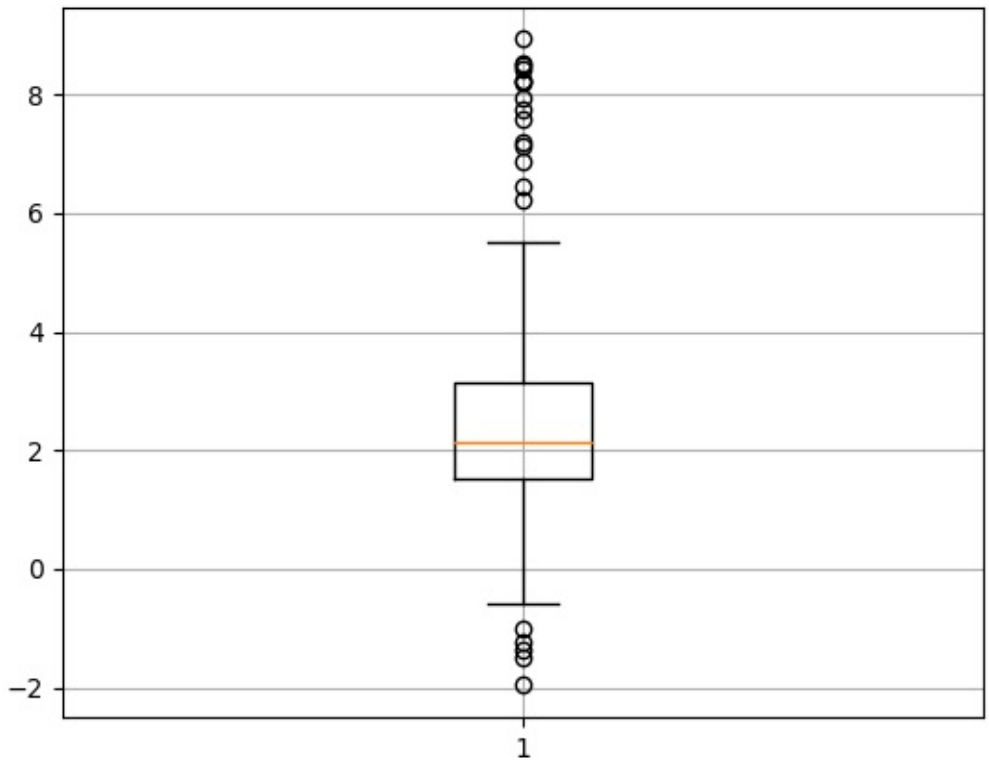


*Figure 3-14. Box and whisker plot of US CPI*

You can also plot it without the outliers (any value that lies more than one and a half times the length of the box from either end of the box). To create Figure 3-14, you can use the following code snippet:

```
# Taking the values of the last twenty years
cpi_last_ten_years = cpi.iloc[-240:]

# Creating the chart
fig, ax = plt.subplots()
ax.boxplot(cpi_last_ten_years['CPIAUCSL'])

# Calling the grid function for better interpretability
plt.grid()

# Calling the legend function to show the labels
plt.legend()

# Showing the plot
plt.show()
```

To remove the outliers from the plot, you simply use the following tweak:

```
# Replace the corresponding code line with the following
ax.boxplot(cpi_last_ten_years['CPIAUCSL'], showfliers = False)
```

Which will give you Figure 3-15:

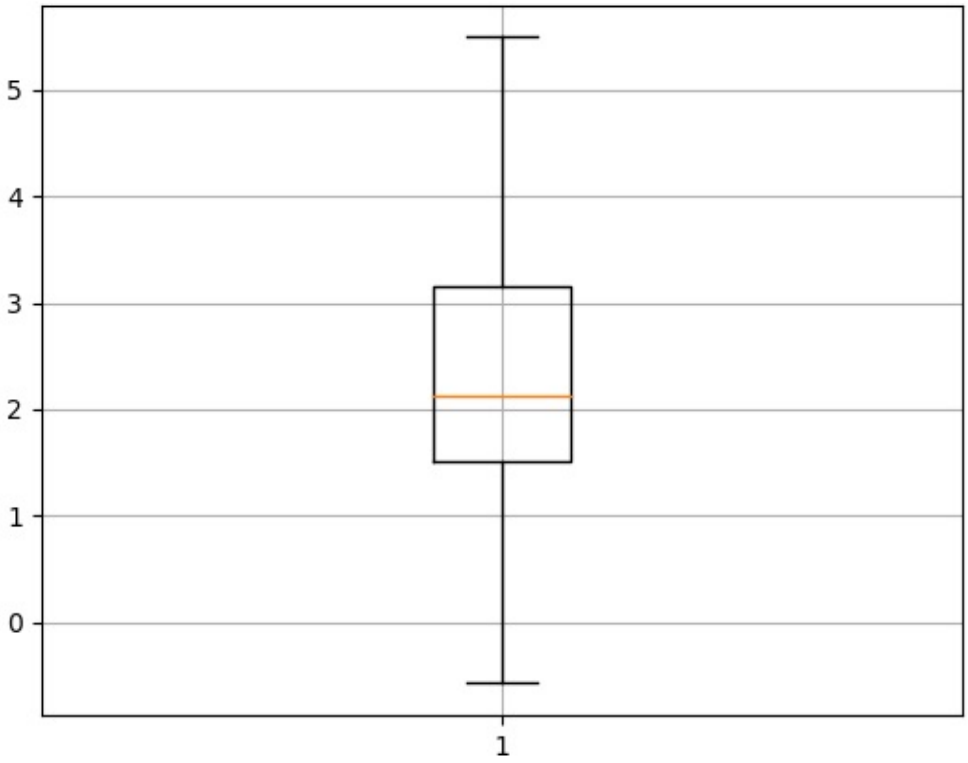


*Figure 3-15. Box and whisker plot of US CPI with no outliers*

Many more data visualization techniques exist such as *heatmaps* (commonly used with correlation data and temperature mapping) and *pie charts* (commonly used for budgeting and segmentation). It all depends on what you need to understand and what fits better with your needs. For example, a line plot is better suited for time series that only have one feature (for example, only the close price of a certain security is available). A histogram plot is better suited with probability distribution data.

**NOTE**

Let’s do a summary of everything you need to retain:

- Data visualization depends on the type of analysis and interpretation you want to do. Some plots are better suited with certain types of data.
- Data visualization helps with an initial interpretation of data before confirming it numerically.
- You are more likely to use line plots and candlestick plots when dealing with financial time series.

## Correlation

*Correlation* is a measure used to calculate the degree of the linear relationship between two variables. It is a number between -1.0 and 1.0 with -1.0 designating a strong negative relationship between the variables and 1.0 designating a strong positive relationship.

A value of zero indicates that there is no linear association between the variables. However, correlation does not imply causation. Two variables are said to be correlated if they move in the same direction, but this does not imply that one causes the other to move or that they move as a result of the same events.

Most people agree that some assets have natural correlations. For instance, because they are both part of the same industry and are affected by the same trends and events, the stocks of Apple and Microsoft are positively connected (which means their general trend is in the same direction). Figure 3-16 shows the chart between the two stocks. Notice how they move together.

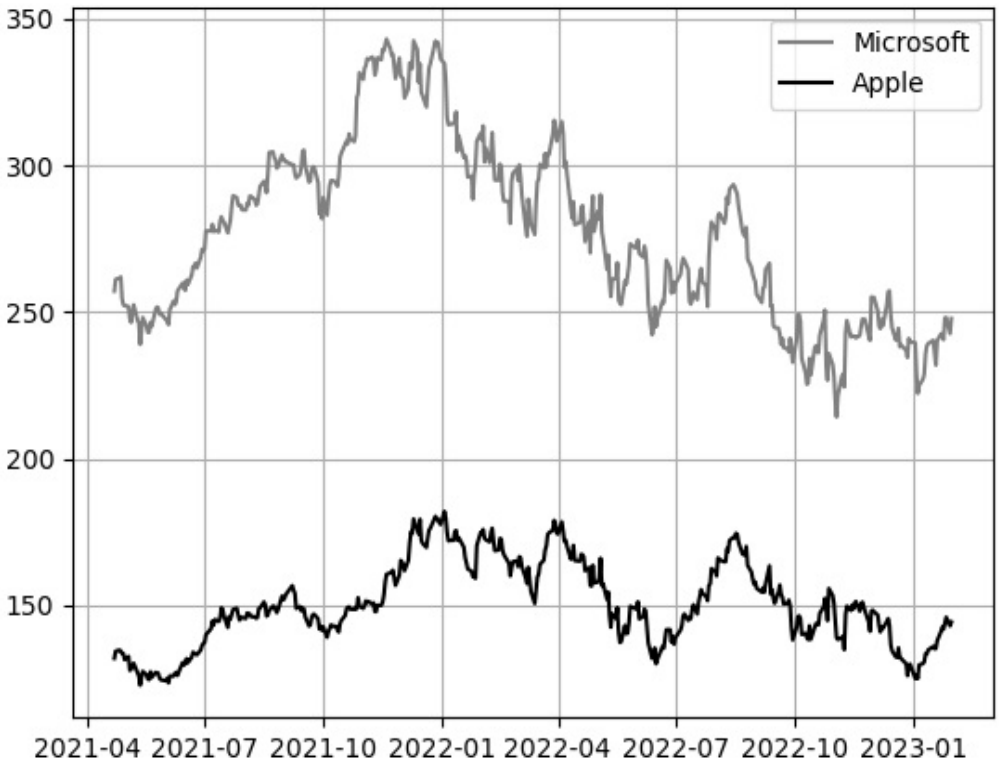


*Figure 3-16. Apple and Microsoft stock prices since 2021*

The tops and bottoms of both stocks occur at almost the exact same time. Similarly, as the United States and the UK have similar economic drivers and impacts, they are also likely to have positively correlated inflation numbers.

Checking for correlation is done through visual interpretation and mathematical formulae. Before seeing an example, let's deeply understand the roots of calculating correlation so that you know where it comes from and what are its limitations.

> **NOTE**
>
> Simply put, to calculate correlation, you need to measure how close the points in a scatter plot of the two variables are to a straight line. The more they look like a straight line, the more they are positively correlated, hence the term, *linear correlation*.

There are two main[1] ways to calculate correlation, it is either by using the Spearman method or the Pearson method.

The *Pearson* correlation coefficient, is a measure of the linear association between two variables calculated from the standard deviation and the covariance between two variables. But, what is covariance?

*Covariance* calculates the average of the difference between the means of the two variables. If the two variables have a tendency to move together, the covariance is positive and if the two variables typically move in opposite directions, the covariance is negative. It ranges between infinity and negative infinity with values close to zero representing no linear correlation.

The formula for calculating the covariance between variables *x* and *y* is as follows:

$$cov_{xy} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{n}$$

Therefore, covariance is the sum of the products of the average deviations between the variables and their respective means (which measures the degree of their association). An average is taken to normalize this calculation. The Pearson correlation coefficient is calculated as follows:

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \overline{y})^2}}$$

Simplifying the previous correlation formula gives you the following:

$$r_{xy} = \frac{cov_{xy}}{\sigma_x \sigma_y}$$

Therefore, Pearson correlation coefficient is simply the covariance between two variables divided by the product of their standard deviation. Let's calculate the correlation between the US CPI year-on-year changes and the UK CPI year-on-year changes. The intuition is that the correlation is above zero as economically, the UK and the US are related. The following code block calculates the Pearson correlation coefficient for the two time series:

```
# Importing the required libraries
import pandas_datareader as pdr
import pandas as pd
```

```
# Setting the beginning and end of the historical data
start_date = '1995-01-01'
end_date   = '2022-12-01'

# Creating a dataframe and downloading the CPI data using its code
name and its source
cpi_us = pdr.DataReader('CPIAUCSL', 'fred', start_date, end_date)
cpi_uk = pdr.DataReader('GBRCPIALLMINMEI', 'fred', start_date,
end_date)

# Dropping the NaN values from the rows
cpi_us = cpi_us.dropna()
cpi_uk = cpi_uk.dropna()

# Transforming the US CPI into a year-on-year measure
cpi_us = cpi_us.pct_change(periods = 12, axis = 0) * 100
cpi_us = cpi_us.dropna()

# Transforming the UK CPI into a year-on-year measure
cpi_uk = cpi_uk.pct_change(periods = 12, axis = 0) * 100
cpi_uk = cpi_uk.dropna()

# Joining both CPI data into one dataframe
combined_cpi_data = pd.concat([cpi_us['CPIAUCSL'],
cpi_uk['GBRCPIALLMINMEI']], axis = 1)

# Using pandas' correlation function to calculate the measure
combined_cpi_data.corr(method = 'pearson')
```

The output is as follows:

```
                 CPIAUCSL  GBRCPIALLMINMEI
CPIAUCSL         1.000000         0.732164
GBRCPIALLMINMEI  0.732164         1.000000
```

The correlation between the two is a whopping 0.73. This is in line with the expectations. Pearson correlation is usually used with variables that have proportional changes and are normally distributed. This may be an issue as financial data is not normally distributed. Therefore, it is interesting to discuss Spearman correlation.

*Spearman correlation* is a non-parametric rank correlation that measures the strength of the relationship between the variables. It is suitable for variables that do not follow a normal distribution.

**NOTE**

Remember, financial returns are not normally distributed but are sometimes treated that way for simplicity.

Unlike Pearson correlation, the Spearman rank correlation takes into account the order of the values, rather than the actual values. To calculate Spearman correlation, follow these steps:

1. Rank the values of each variable. This is done by inputing 1 instead of the smallest variable and inputing the length of the dataset instead of the largest number.
2. Calculate the difference in ranks. Mathematically, the difference in ranks is represented by the letter *d* in the mathematical formula to come. Then, calculate their squared differences.
3. Sum the squared differences you have calculated from step 2.
4. Use the following formula to calculate Spearman correlation.

$$\rho = 1 - \frac{6 \sum_{i=1}^{n} d_i^2}{n^3 - n}$$

As with Pearson correlation, Spearman correlation also ranges from -1.00 to 1.00 with the same interpretation.

**NOTE**

Strong positive correlations are generally upwards of 0.70, while strong negative correlations are generally downwards of -0.70.

The following code block calculates the Spearman rank correlation coefficient for the two time series:

```
# Importing the required libraries
import pandas_datareader as pdr
import pandas as pd
```

```
# Setting the beginning and end of the historical data
start_date = '1995-01-01'
end_date   = '2022-12-01'

# Creating a dataframe and downloading the CPI data using its code
name and its source
cpi_us = pdr.DataReader('CPIAUCSL', 'fred', start_date, end_date)
cpi_uk = pdr.DataReader('GBRCPIALLMINMEI', 'fred', start_date,
end_date)

# Dropping the NaN values from the rows
cpi_us = cpi_us.dropna()
cpi_uk = cpi_uk.dropna()

# Transforming the US CPI into a year-on-year measure
cpi_us = cpi_us.pct_change(periods = 12, axis = 0) * 100
cpi_us = cpi_us.dropna()

# Transforming the UK CPI into a year-on-year measure
cpi_uk = cpi_uk.pct_change(periods = 12, axis = 0) * 100
cpi_uk = cpi_uk.dropna()

# Joining both CPI data into one dataframe
combined_cpi_data = pd.concat([cpi_us['CPIAUCSL'],
cpi_uk['GBRCPIALLMINMEI']], axis = 1)

# Using pandas' correlation function to calculate the measure
combined_cpi_data.corr(method = 'spearman')
```

The output is as follows:

```
                 CPIAUCSL  GBRCPIALLMINMEI
CPIAUCSL         1.000000         0.472526
GBRCPIALLMINMEI  0.472526         1.000000
```

Let’s answer a very important question after getting this difference in results. Why are the two measures so different?

The first thing to keep in mind is what they measure. Pearson correlation measures the linear relationship (trend) between the variables while Spearman rank correlation measures the monotonic trend. The word *monotonic* refers to moving in the same direction but not exactly at the same rate or magnitude. Also, Spearman correlation transforms the data to an ordinal type (through the ranks) as opposed to Pearson correlation which uses the actual values.

*Autocorrelation* (also referred to as serial correlation) is a statistical method used to look at the relationship between a given time series and a lagged version of it. It is generally used to predict future values through patterns in data, such as seasonality or trends. Autocorrelation is therefore the values' relationship with the previous values. For example, comparing each day's Microsoft stock price to the preceding day and see if there is a discernible correlation there. Algorithmically speaking, this can be represented in table 3-2:

*Table 3-1. Lagged values table*

| t | | t-1 | |
|---|---|---|---|
| $ | 1.25 | $ | 1.65 |
| $ | 1.77 | $ | 1.25 |
| $ | 1.78 | $ | 1.77 |
| $ | 1.25 | $ | 1.78 |
| $ | 1.90 | $ | 1.25 |

Each row represents a time period. Column *t* is the current price and column *t-1* is the previous price put on the row that represents the present. This is done when creating machine learning models so as to understand the relationship between the current values and the previous ones at every time step (row).

Positive auto correlation frequently occurs in trending assets and is associated with the idea of persistence (trend following). On the other hand, ranging markets exhibit negative auto correlation, which is associated with the idea of anti-persistence (mean reversion).

**NOTE**

Measures of short-term correlation are typically computed using returns on prices rather than real prices. However, it is possible to utilize the prices directly to identify long-term trends.

The following code block calculates the autocorrelation of the US CPI year-on-year:

```
# Importing the required libraries
import pandas_datareader as pdr
import pandas as pd

# Setting the beginning and end of the historical data
start_date = '1950-01-01'
end_date   = '2022-12-01'

# Creating a dataframe and downloading the CPI data using its code
name and its source
cpi = pdr.DataReader('CPIAUCSL', 'fred', start_date, end_date)

# Dropping the NaN values from the rows
cpi = cpi.dropna()

# Transforming the US CPI into a year-on-year measure
cpi = cpi.pct_change(periods = 12, axis = 0) * 100
cpi = cpi.dropna()

# Transforming the data frame to a series structure
cpi = cpi.iloc[:,0]

# Calculating autocorrelation with a lag of 1
print('Correlation with a lag of 1 = ', round(cpi.autocorr(lag = 1),
2))

# Calculating autocorrelation with a lag of 6
print('Correlation with a lag of 6 = ', round(cpi.autocorr(lag = 6),
2))

# Calculating autocorrelation with a lag of 12
print('Correlation with a lag of 12 = ', round(cpi.autocorr(lag = 12),
2))
```

A lag of 12 means that every data is compared to the data from twelve periods ago and then a calculation measure is calculated. The output of the code is as follows:

```
Correlation with a lag of 1 =  0.99
Correlation with a lag of 6 =  0.89
Correlation with a lag of 12 =  0.73
```

Now, before proceeding to the next section, let’s revert back to information

theory and discuss one interesting correlation coefficient that is able to pick-up on non-linear relationships. One of these ways is the maximal information coefficient (MIC).

The *maximal information coefficient* (MIC) is a non-parametric measure of association between two variables designed to handle large and complex data. It is generally seen as a more robust alternative to traditional measures of correlation, such as Pearson correlation and Spearman rank correlation. Introduced by *Reshef et al.*, the MIC uses concepts from information theory that you have seen in Chapter 2.

The MIC measures the strength of the association between two variables by counting the number of cells in a contingency table that are maximally informative about the relationship between the variables. The MIC value ranges from 0 to 1, with higher values indicating stronger associations. It can handle high-dimensional data and can identify non-linear relationships between variables. It is however non-directional which means that values close to 1 only suggest a strong correlation between the two variables but it does not say whether the correlation is positive or negative.

**NOTE**

In other words, the mutual information between the two variables within each bin is calculated after the range of each variable has been divided into a set of bins.

The strength of the association between the two variables is then estimated using the maximum mutual information value across all bins.

Let’s check out a practical example that showcases the strength of the MIC in detecting non-linear relationships. The following example simulate a Sinus and Cosinus time series. Intuitively, looking at Figure 3-17, it seems that there is a lag-lead relationship between the two.

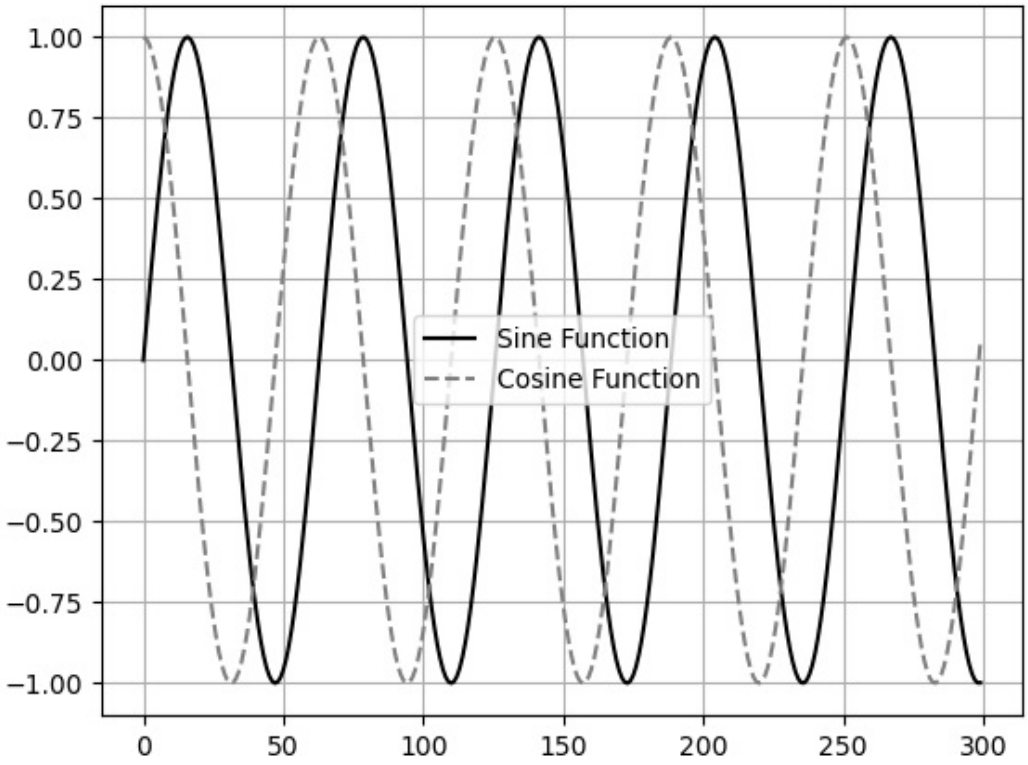


*Figure 3-17. The two wave series showing a form of non-linear relationship*

The following Python code snippet creates the two time series and plots Figure 3-17:

```
# Importing the required libraries
import numpy as np
import matplotlib.pyplot as plt

# Setting the range of the data
data_range = np.arange(0, 30, 0.1)

# Creating the sine and the cosine waves
sine = np.sin(data_range)
cosine = np.cos(data_range)

# Plotting
plt.plot(sine, color = 'black', label = 'Sine Function')
plt.plot(cosine, color = 'grey', linestyle = 'dashed', label = 'Cosine
Function')
plt.grid()
plt.legend()
```

Now, the job is to calculate the three correlation measures and analyze their results. This can be done using the following code:

```
# Importing the libraries
from scipy.stats import pearsonr
from scipy.stats import spearmanr
from minepy import MINE

# Calculating the linear correlation measures
print('Correlation | Pearson: ', round(pearsonr(sine, cosine)[0], 3))
print('Correlation | Spearman: ', round(spearmanr(sine, cosine)[0],
3))

# Calculating the MIC
mine = MINE(alpha = 0.6, c = 15)
mine.compute_score(sine,cosine)
MIC = mine.mic()
print('Correlation | MIC: ', round(MIC, 3))
```

Note that since the code creates an array (and not a data frame), it is mandatory to import the required libraries before calculating the measures. This will be made clear in the next chapter. The following is the output of the code:

```
Correlation | Pearson:  0.035
Correlation | Spearman:  0.027
Correlation | MIC: 0.602
```

Let’s interpret the results:

- *Pearson correlation*: Notice the absence of any type of correlation here due to it missing out on the non-linear association.
- *Spearman correlation*: The same situation applies here with an extremely weak correlation.
- *MIC*: The measure returned a strong relationship of 0.60 between the two which is closer to reality. It seems that the MIC states that both waves have a strong relationship albeit non-linear.

**NOTE**

You may need to update Microsoft Visual C++ (at least version 14.0 or greater) to avoid any errors in

trying to run `minepy`library.

The MIC is very useful in economic analysis, financial analysis, and even finding trading signals if used properly. Non-linear relationships are abundant in such complex fields and being able to detect them may give a sizable edge.

**NOTE**

The main takeaways from the correlation section are the following points:

- Correlation is a measure used to calculate the degree of the linear relationship between two variables. It is a number between -1.0 and 1.0 with -1.0 designating a strong negative relationship between the variables and 1.0 designating a strong positive relationship.
- There are two main types of correlation measures, Spearman and Pearson. Both have their advantages and limitations.
- Autocorrelation is the correlation of the variable with its own lagged values. For example, if the autocorrelation of Nvidia's stock returns is positive, it denotes a trending configuration.
- Correlation measures can also refer to non-linear relationships when you use the right tool, for example, the MIC.

## The Concept of Stationarity

Stationarity is one of the key concepts in statistical analysis and machine learning. *Stationarity* is when the statistical characteristics of the time series (mean, variance, etc.) are constant over time. In other words, no discernable trend is detectable when plotting the data across time.

The different learning models rely on data stationarity as it is one of the basics of statistical modelling and this is mainly for simplicity. In finance, price time series are not stationary as they show trends with varying variance (volatility). Take a look at Figure 3-18 and see if you can detect a trend. Would you say that this time series is stationary?

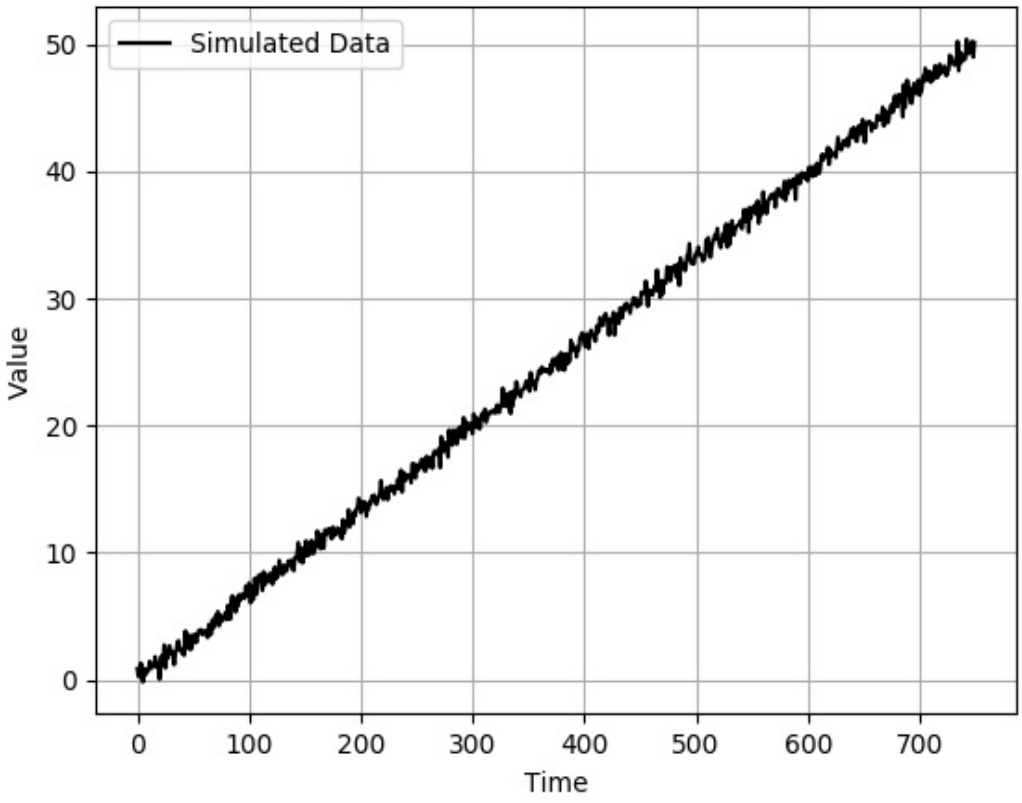


*Figure 3-18. Simulated data with a varying mean across time*

Naturally, the answer is no as a rising trend is clearly in progress. States like this are undesirable for statistical analyses and machine learning. Luckily, there are transformations that you can apply to the time series to make them stationary. But first, let's see how to check for stationarity the mathematical way as the visual way does not prove anything. The right process to deal with data stationarity problem is to follow these steps:

1. Check for stationarity using the different statistical tests that you will see in this section.
2. If the tests show data stationarity, you are ready to use the data for the different learning algorithms. If the tests show that the data is not stationary, you have to proceed to the next step.
3. Apply the price transformation technique that you will see in this section.
4. Re-check for stationarity using the same tests on the new transformed data.

5. If the test shows data stationarity, then you have successfully transformed your data. Otherwise, re-do the transformation and check again until you have stationary data.

> **NOTE**
>
> Ascending or descending time series have varying mean and variances through time and are therefore most likely non-stationary. There are exceptions to this and you will see later why.

Remember, the aim of stationarity is stable and constant mean and variance over time. Therefore, when you look at Figure 3-19, what can you say about it?

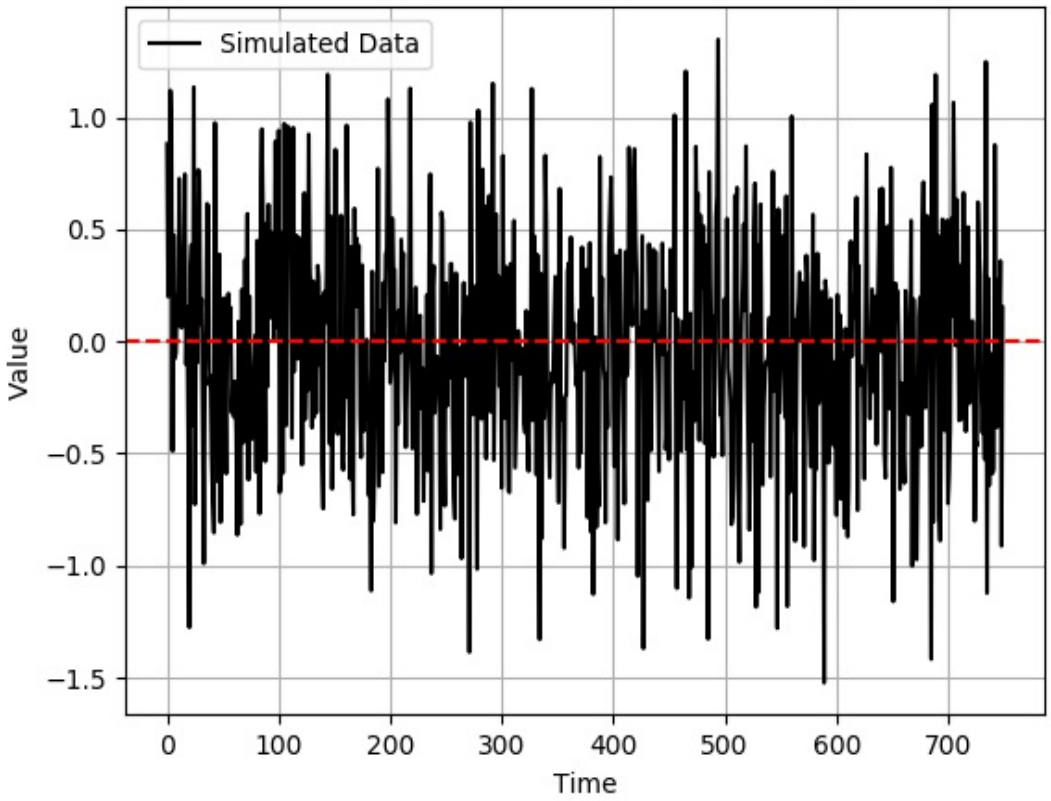


*Figure 3-19. Simulated data with a mean around zero across time*

Visually, it looks like the data does not have a trend and it also looks like it fluctuates around a stable mean with stable variance around it. The first impression is that the data is stationary. Of course, this must be proved by

statistical tests.

The first and most basic test is the *Augmented-Dickey-Fuller* (ADF) test. The ADF tests for stationarity using hypothesis testing.

The ADF test searches for a unit root in the data. A *unit root* is a property of non-stationary data and in the context of time series analysis, refers to a characteristic of a stochastic process where the series has a root equal to 1. In simpler terms, it means that its statistical properties, such as the mean and variance, change over time. Here's what you need to know:

- The null hypothesis assumes the presence of a unit root. This means that if you are trying to prove that the data is stationary, you are looking to reject the null hypothesis (as seen in the hypothesis testing section from Chapter 2).
- The alternative hypothesis is therefore the absence of a unit root and the stationarity of the data.
- The p-value obtained from the test must be less than the significance level chosen (in most cases, it is 5%).

Let's take the US CPI year-on-year data and test it for stationarity. The following code snippet checks for stationarity using the ADF test:

```
# Importing the required libraries
from statsmodels.tsa.stattools import adfuller
import pandas_datareader as pdr

# Setting the beginning and end of the historical data
start_date = '1950-01-01'
end_date   = '2022-12-01'

# Creating a dataframe and downloading the CPI data using its code
name and its source
cpi = pdr.DataReader('CPIAUCSL', 'fred', start_date, end_date)

# Dropping the NaN values from the rows
cpi = cpi.dropna()

# Transforming the US CPI into a year-on-year measure
cpi = cpi.pct_change(periods = 12, axis = 0) * 100
cpi = cpi.dropna()
```

```
# Applying the ADF test on the CPI data
adfuller(cpi)
print('p-value: %f' % adfuller(cpi)[1])
```

The output of the code is as follows:

```
p-value: 0.0152
```

Assuming a 5% significance level, it seems that it is possible to accept that the year-on-year data is stationary (however, if you want to be more strict and use a 1% significance level, then the p-value suggests that the data is non-stationary). In any way, even looking at the chart can make you scratch your head. Remember that in Figure 3-11, the yearly changes in the US CPI seem to be stable but does not resemble stationary data. This is why numerical and statistical tests are used.

Now, let's the do the same thing but omit taking the yearly changes. In other words, applying the code on the raw US CPI data and not taking year-on-year changes. Here's the code:

```
# Importing the required libraries
from statsmodels.tsa.stattools import adfuller
import pandas_datareader as pdr

# Setting the beginning and end of the historical data
start_date = '1950-01-01'
end_date   = '2022-12-01'

# Creating a dataframe and downloading the CPI data using its code
name and its source
cpi = pdr.DataReader('CPIAUCSL', 'fred', start_date, end_date)

# Dropping the NaN values from the rows
cpi = cpi.dropna()

# Applying the ADF test on the CPI data
adfuller(cpi)
print('p-value: %f' % adfuller(cpi)[1])
```

The output of the code is as follows:

```
p-value: 0.999
```

Clearly, the p-value is greater than all significance levels which means that the time series is non-stationary. Let's sum up these results:

- It seems that you can reject the null hypothesis using a 5% significance level when it comes to the year-on-year changes on the US CPI. The dataset is assumed to be stationary.
- It seems that you cannot reject the null hypothesis using a 5% significance level when it comes to the raw values of the US CPI. The dataset is assumed to be non-stationary.

This becomes obvious when you plot the raw values of the US CPI, as shown in Figure 3-20.

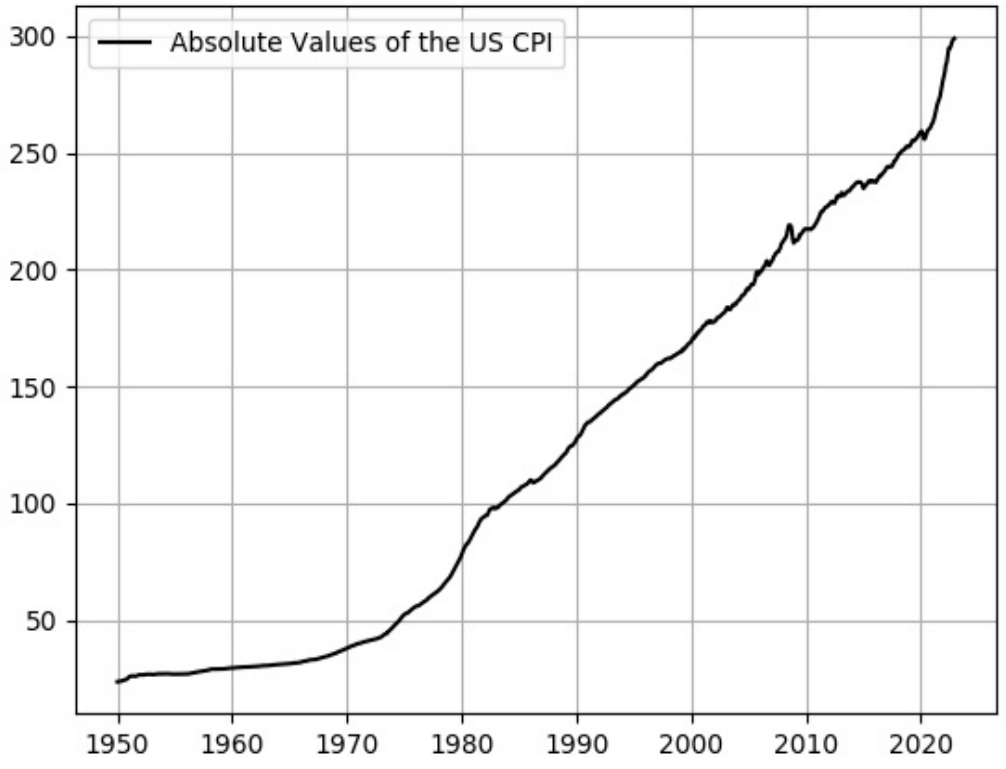


*Figure 3-20. Absolute values of the US CPI showing a clearly trending nature*

The other test that you must be aware of is called the The *Kwiatkowski-Phillips-Schmidt-Shin* (KPSS) which is also a statistical test with the aim of determining whether the time series is stationary or non-stationary. However, the KPSS test

can detect stationarity in trending time series which makes it a powerful tool.

Trending time series can actually be stationary on the condition they have a stable mean.

> **WARNING**
>
> The ADF test has a a null hypothesis that argues for non-stationarity and an alternative hypothesis that argues for stationarity. The KPSS test on the other hand, has a null hypothesis that argues for stationarity and an alternative hypothesis that argues for non-stationarity.

Before analyzing the inflation data, let's see how can a trending time series be stationary. Remember that stationarity refers to a stable mean and standard deviation so if somehow, you have a gradually ascending or descending time series with stable statistical properties, it may be stationary. The next code snippet simulates a sine wave and then adds a touch of a trending nature to it:

```
# Importing the required libraries
import numpy as np
import matplotlib.pyplot as plt

# Creating the first time series using sine waves
length = np.pi * 2 * 5
sinewave = np.sin(np.arange(0, length, length / 1000))

# Creating the second time series using trending sine waves
sinewave_ascending = np.sin(np.arange(0, length, length / 1000))

# Defining the trend variable
a = 0.01

# Looping to add a trend factor
for i in range(len(sinewave_ascending)):

    sinewave_ascending[i] = a + sinewave_ascending[i]

    a = 0.01 + a
```

Plotting the two series as shown in Figure 3-21 shows how the trending sinewave seems to be stable. But let's prove this through statistical tests.

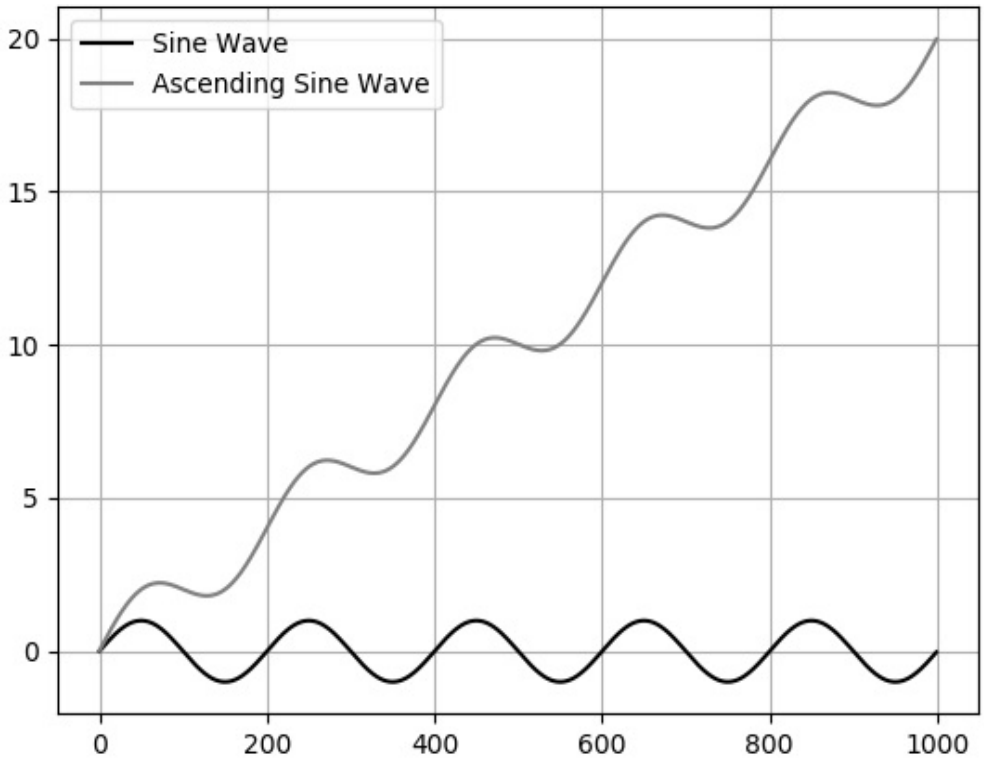


*Figure 3-21. A normal sine wave simulated series with a trending sine wave*

Figure 3-21 is generated using the following code (make sure you have defined the series using the previous code block):

```
# Plotting the series
plt.plot(sinewave, label = 'Sine Wave', color = 'black')
plt.plot(sinewave_ascending, label = 'Ascending Sine Wave', color =
'grey')

# Calling the grid function for better interpretability
plt.grid()

# Calling the legend function to show the labels
plt.legend()

# Showing the plot
plt.show()
```

Let’s try the ADF test on both series and see what the results are:

```
# ADF testing | Normal sine wave
adfuller(sinewave)
print('p-value: %f' % adfuller(sinewave)[1])

# ADF testing | Ascending sine wave
adfuller(sinewave_ascending)
print('p-value: %f' % adfuller(sinewave_ascending)[1])
```

The output is as follows:

```
p-value: 0.000000 # For the sine wave
p-value: 0.898635 # For the ascending sine wave
```

Clearly, the ADF test is consistent with the idea that trending markets cannot be stationary. But what about the KPSS test? The following code uses the KPSS on the same data to check for stationarity:

```
# Importing the KPSS library
from statsmodels.tsa.stattools import kpss

# KPSS testing | Normal sine wave
kpss(sinewave)
print('p-value: %f' % kpss(sinewave)[1])

# KPSS testing | Ascending sine wave
kpss(sinewave_ascending)
print('p-value: %f' % kpss(sinewave_ascending)[1])

# KPSS testing while taking into account the trend | Ascending sine
wave
kpss(sinewave_ascending, regression = 'ct')
print('p-value: %f' % kpss(sinewave_ascending, regression = 'ct')[1])

'''
The 'ct' argument is used to check if the dataset is stationary
around a trend. By default, the argument is 'c' which is is used
to check if the data is stationary around a constant.
'''
```

The output is as follows:

```
p-value: 0.10 # For the sine wave
p-value: 0.01 # For the ascending sine wave without trend
consideration
p-value: 0.10 # For the ascending sine wave with trend consideration
```

Remember that the null hypothesis of the KPSS test is that the data is stationary, therefore if the p-value is greater than the significance level, the data is considered stationary since it is not possible to reject the null hypothesis.

The KPSS statistic when taking into account the trend, states that the ascending sine wave is a stationary time series. This is a basic example on how you can find stationary data in trending time series.

Let's take the US CPI year-on-year data and test it for stationarity. The following code snippet checks for stationarity using the KPSS test:

```
# Importing the required libraries
from statsmodels.tsa.stattools import kpss
import pandas_datareader as pdr

# Setting the beginning and end of the historical data
start_date = '1950-01-01'
end_date   = '2022-12-01'

# Creating a dataframe and downloading the CPI data using its code
name and its source
cpi = pdr.DataReader('CPIAUCSL', 'fred', start_date, end_date)

# Dropping the NaN values from the rows
cpi = cpi.dropna()

# Transforming the US CPI into a year-on-year measure
cpi = cpi.pct_change(periods = 12, axis = 0) * 100
cpi = cpi.dropna()

# Applying the KPSS (no trend consideration) test on the CPI data
kpss(cpi)
print('p-value: %f' % kpss(cpi)[1])

# Applying the KPSS (with trend consideration) test on the CPI data
kpss(cpi, regression = 'ct')
print('p-value: %f' % kpss(cpi, regression = 'ct')[1])
```

The output of the code is as follows:

```
p-value: 0.036323 # without trend consideration
p-value: 0.010000 # with trend consideration
```

It seems that the results from the KPSS test are in contradiction with the results from the ADF test. This may happen from time to time and differencing may

solve the issue (bear in mind, that the year-on-year data is already a differenced time series from the absolute CPI values but some time series may need more than one differencing to become stationary and it also depends on the period of differencing). The safest solution in contradiction is to transform once more the data.

Before finishing this section on stationarity, let's discuss a complex topic that you will later see in action in Chapter 7. Transforming the data may cause an unusual problem that is, *memory loss*. In his book, *Marco Lopez de Prado* proposed a technique called fractional differentiation with the aim of making data stationary while preserving some memory.

When a non-stationary time series is differenced in the aim of making it stationary, memory loss occurs which is another way of saying that the autocorrelation between the values is greatly reduced, thus removing the trend component and the DNA of the underlying asset. The degree of differencing and the persistence of the autocorrelation structure in the original series determines how much memory loss occurs.

**NOTE**

This section has presented many complex concepts. You should retain the following:

- Stationarity refers to the concept of stable mean and variance through time. It is a desired characteristic as most machine learning models rely on it.
- Financial price time series are most likely non-stationary and require a first order transformation to become stationary and ready for statistical modelling. Some may even require a second order transformation to become stationary.
- The ADF and KPSS tests check for stationarity in the data with the latter being able to check for stationarity in trending data, thus being more thorough.
- Trending data may be stationary. Although this characteristic is rare, the KPSS is able to detect the stationarity as opposed to the ADF test.

## Regression Analysis and Statistical Inference

*Inference,* as Oxford Languages defines it, is a conclusion reached on the basis of evidence and reasoning. Therefore, as opposed to descriptive statistics, inferential statistics use the data or a sample of the data to make inferences

(forecasts). The main tool in statistical inference is linear regression.

*Linear regression* is a basic machine learning algorithm you will see in this book as of Chapter 7 with the other machine learning algorithms. Hence, let's present the intuition of regression analysis in this section.

The most basic form of a linear regression equation is as follows:

$$y = \alpha + \beta x + \epsilon$$

$$y\ is\ the\ dependent\ variable,\ it\ is\ what\ you\ want\ to\ forecast$$

$$x\ is\ the\ independent\ variable,\ it\ is\ what\ you\ use\ as\ an\ input\ to\ forecast\ y$$

$$\alpha\ is\ the\ expected\ value\ of\ the\ dependent\ variable\ when\ the\ independent\ variables\ are\ equal\ to\ zero$$

$$\beta\ represents\ the\ change\ in\ the\ dependent\ variable\ per\ unit\ chang\ in\ the\ independent\ variable$$

$$\epsilon\ is\ the\ residual\ or\ the\ unexplained\ variation$$

The basic linear regression equation states that a dependent variable (what you want to forecast) is explained by a constant, a sensitivity-adjusted variable, and a residual (error term to account for unexplained variations). Consider table 3-3:

*Table 3-2. Prediction table*

| y | x |
|---|---|
| 100 | 49 |
| 200 | 99 |
| 300 | 149 |
| 400 | 199 |
| ? | 249 |

The linear equation to predict *y* given *x*, is as follows:

$$y_i = 2 + 2x_i$$

Therefore, the latest *y* given *x* = 249 should be 500:

$$y_i = 2 + 2x_i = 2 + (2 \times 249) = 500$$

Notice how linear regression perfectly captures the linear relationship between the two variables since there is no residual (unexplained variations). When a linear regression perfectly captures the relationship between two variables, it means that their coordinate points are perfectly aligned on a linear line across the x-axis.

Multiple linear regression can take the following form:

$$y_i = \alpha + \beta_1 x_1 + \ldots + \beta_n x_n + \epsilon_i$$

This basically means that the dependent variable *y* may be impacted by more than one variable. For instance, if you want to estimate housing prices, you may want to take into account the number of rooms, the surface area, the neighborhood, and any other variable that is likely to impact the price. Similarly, if you want to predict commodity prices, you may want to take into account the different macroeconomic factors, currency values, and any other alternative data.

It is important to understand what every variable refers to. Make sure to memorize the previous formula. Linear regression has a few assumptions:

*Linear relationship*

> The relationship between the dependent variable and the independent variable(s) should be linear, meaning that a straight line across the plane can describe the relationship. This is rare in real life when dealing with complex financial variables.

*Independence of variables*

> The observations should be independent of each other, meaning that the value of one observation does not influence the value of another observation.

*Homoscedasticity*

The variance of the residuals (the difference between the predicted and actual values of the dependent variable) should be constant across all levels of the independent variable(s).

*Normality of the residuals*

The residuals should be normally distributed, meaning that the majority of the residuals are close to zero and the distribution is symmetrical.

In case of a multiple linear regression, you can add a new assumption, that is the absence of *multicollinearity*. The independent variables should not be highly correlated with each other, otherwise it can make it difficult to determine the unique effect of each independent variable on the dependent variable. In other words, this prevents redundancy. You will see linear regression in detail with more in-depth examples in Chapter 7 as this section only introduces it as part of the statistics field.

**NOTE**

Now, you should have a solid understanding in the key concepts of statistics. Let’s do a summary of everything you need to retain:

- Linear regression is part of the inferential statistics field and it is a linear equation that describes the relationship between variables.
- Linear regression interprets and predicts data following an equation that you obtain when you train past data and expect the relationship to hold in the future.

## Summary

Being able to perform data analysis is key towards deploying the right algorithms in order to predict the future values of the time series. Understanding data is done through a wide selection of tools coming from the statistics world.

Make sure you understand what stationarity and what correlation are as they offer extremely valuable insights in modeling.

---

[1] Among others but these two ways are the most popular representations.

# Chapter 4. Linear Algebra and Calculus for Deep Learning

### A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 4th chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at *ccollins@oreilly.com*.

Algebra and calculus are pillars of data science, especially the learning algorithms which are based on concepts from these two mathematical fields. This chapter presents some key algebra and calculus topics in a way that everyone can understand.

It helps to know why you're learning something. This way, you gain the motivation to continue and you know which way to point your focus.

*Algebra* is the study of operations and relational rules, as well as the constructions and ideas that result from them. Algebra covers topics such as linear equations and matrices. You can consider algebra as the first step towards calculus.

*Calculus* is the study of curve slopes and rates of change. Calculus covers topics such as derivatives and integrals. It is heavily used in many fields such as economics and engineering. Different learning algorithms rely on the concepts of calculus to perform their complex operations.

The distinction between the two is that while calculus works with ideas of change, motion, and accumulation, algebra deals with mathematical symbols and

the rules for manipulating those symbols. Calculus focuses on the characteristics and behavior of changing functions, while algebra offers the foundation for solving equations and comprehending functions.

# [Heading to Come]

## Vectors and Matrices

A *vector* is an object that has a magnitude (length) and a direction (arrowhead). The basic representation of a vector is an arrow with coordinates on the axis. But first, let's see what an axis is.

The x-axis and y-axis are perpendicular lines that specify a plane's boundaries and the locations of different points within them in a two-dimensional Cartesian coordinate system. The x-axis is horizontal and the y-axis is vertical.

These axes may represent vectors, with the x-axis representing the vector's horizontal component and the y-axis representing its vertical component.

Figure 4-1 shows a simple 2-dimensional Cartesian coordinate system with both axes.

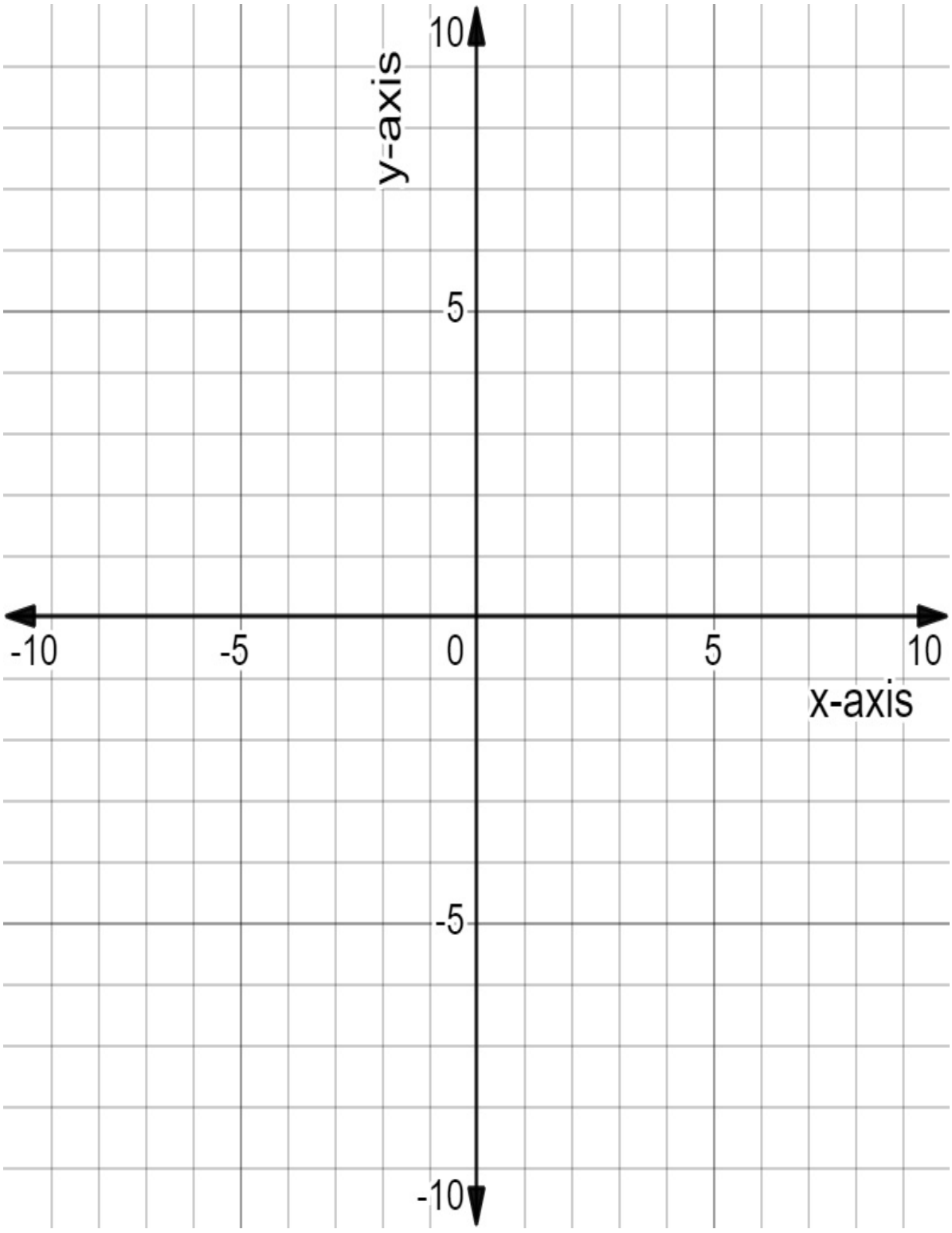


*Figure 4-1. 2-dimensional Cartesian coordinate system*

The 2-dimensional Cartesian coordinate system uses simple parentheses to show the location of different points following this order:

$$Point\ coordinates = (horizontal\ location\ (x), vertical\ location\ ($$

Therefore, if you want to draw point A which has (2, 3) as coordinates, you are likely to look at a graph from point zero, move two points to the right and from there, three points upwards. The result of the point should look like Figure 4-2.

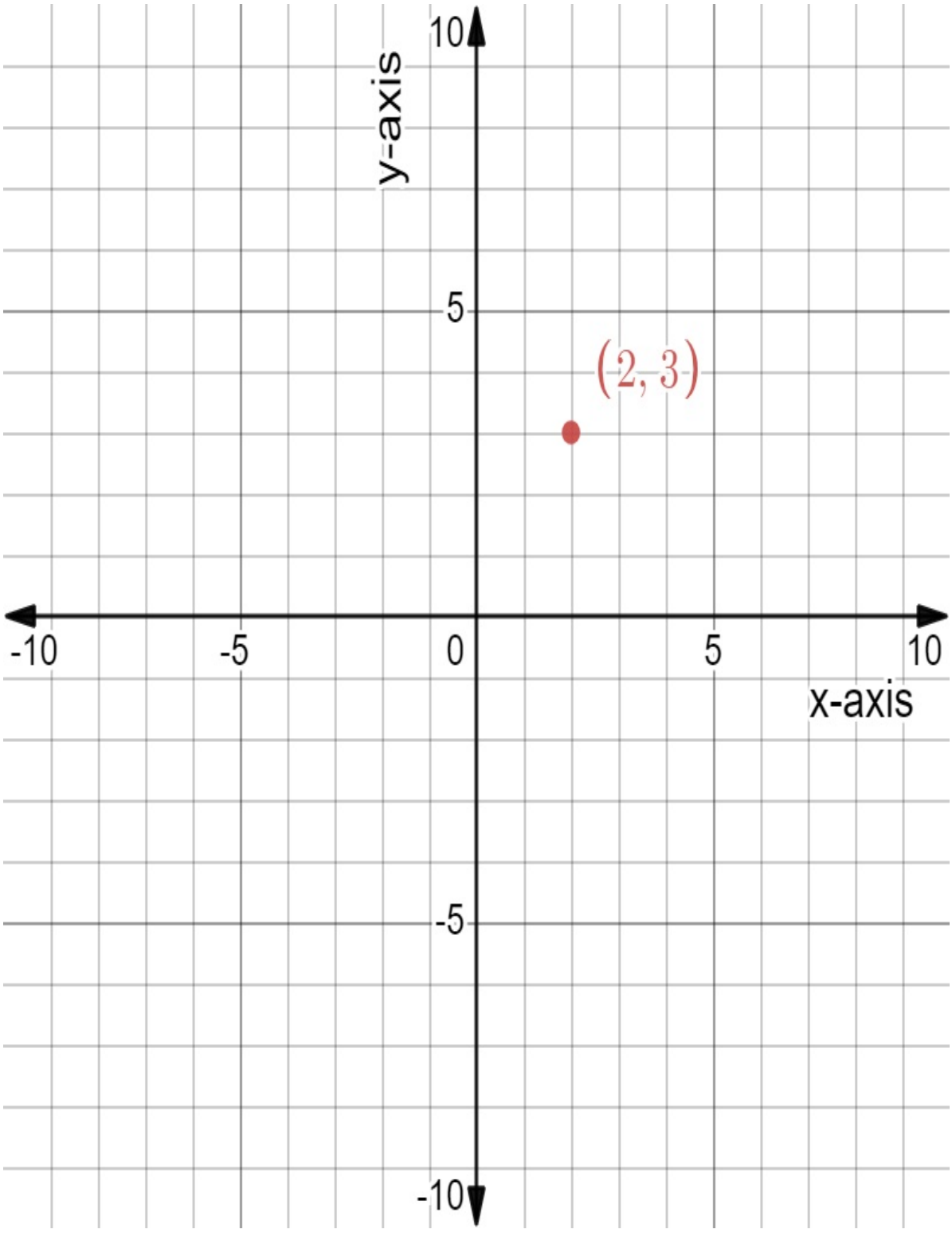


*Figure 4-2. The location of A on the coordinate system*

Let’s now add another point and draw a vector between them. Suppose you have also point B with (4, 5) as coordinates. Naturally, as the coordinates of B are

both higher than the coordinates of A, you would expect vector AB to be upwards sloping. Figure 4-3 shows the new point B and vector AB.

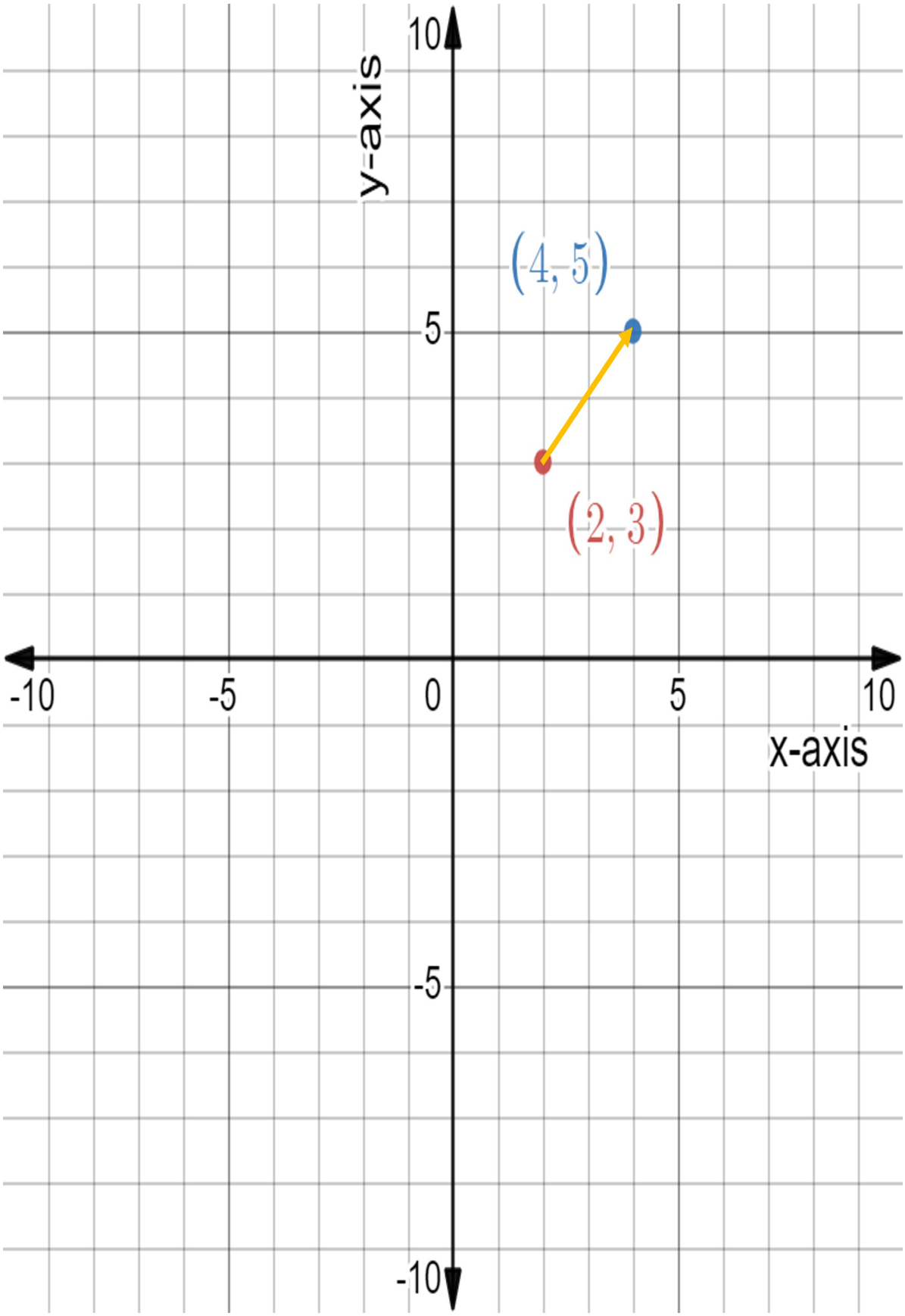
10
y-axis
(4, 5)
5
(2, 3)
-10
-5
0
5
10
x-axis
-5
-10

*Figure 4-3. Vector AB joining A and B points together in magnitude and direction*

However, having drawn the vector using the coordinates of both points, how would you refer to the vector? Simply put, vector AB has its own coordinates that represent it. Remember that the vector is a representation of the movement from point A to point B. This means the 2-point movement along the X-axis and the Y-axis is the vector. Mathematically, to find the vector, you should subtract the two coordinate points from each other while respecting the direction. Here's how:

- *Vector AB* means that you are going from A to B, therefore, you need to subtract the coordinates of point B from the coordinates of point A:

$$\overrightarrow{AB} = < 4 - 2{,}5 - 3 >$$

$$\overrightarrow{AB} = < 2{,}2 >$$

- *Vector BA* means that you are going from B to A, therefore, you need to subtract the coordinates of point A from the coordinates of point B:

$$\overrightarrow{BA} = < 2 - 4, 3 - 5 >$$

$$\overrightarrow{BA} = < -2, -2 >$$

To interpret AB and BA vectors, you think in terms of movement. AB vector represents going from point A to point B, two positive points horizontally and vertically (right and upwards respectively). BA vector represents going from point B to point A, two negative points horizontally and vertically (left and downwards respectively).

**NOTE**

Vectors AB and BA are not the same thing even though they share the same slope. But what is a slope anyway?

The *slope* is the ratio of the vertical change between two points on the line to the horizontal change between the same two points. You calculate the slope using this mathematical formula:

$$Slope = \frac{(\Delta Y)}{(\Delta X)}$$

$Slope\ of\ \overrightarrow{AB} = \frac{2}{2} = 1$

$Slope\ of\ \overrightarrow{BA} = \frac{-2}{-2} = 1$

If the two vectors were simply lines (with no direction), then, they would be the same object. However, adding the directional component makes them two distinguishable mathematical objects.

Figure 4-4 sheds more light on the concept of the slope, as *x* has shifted two points to the right and *y* has shifted two points to the left.

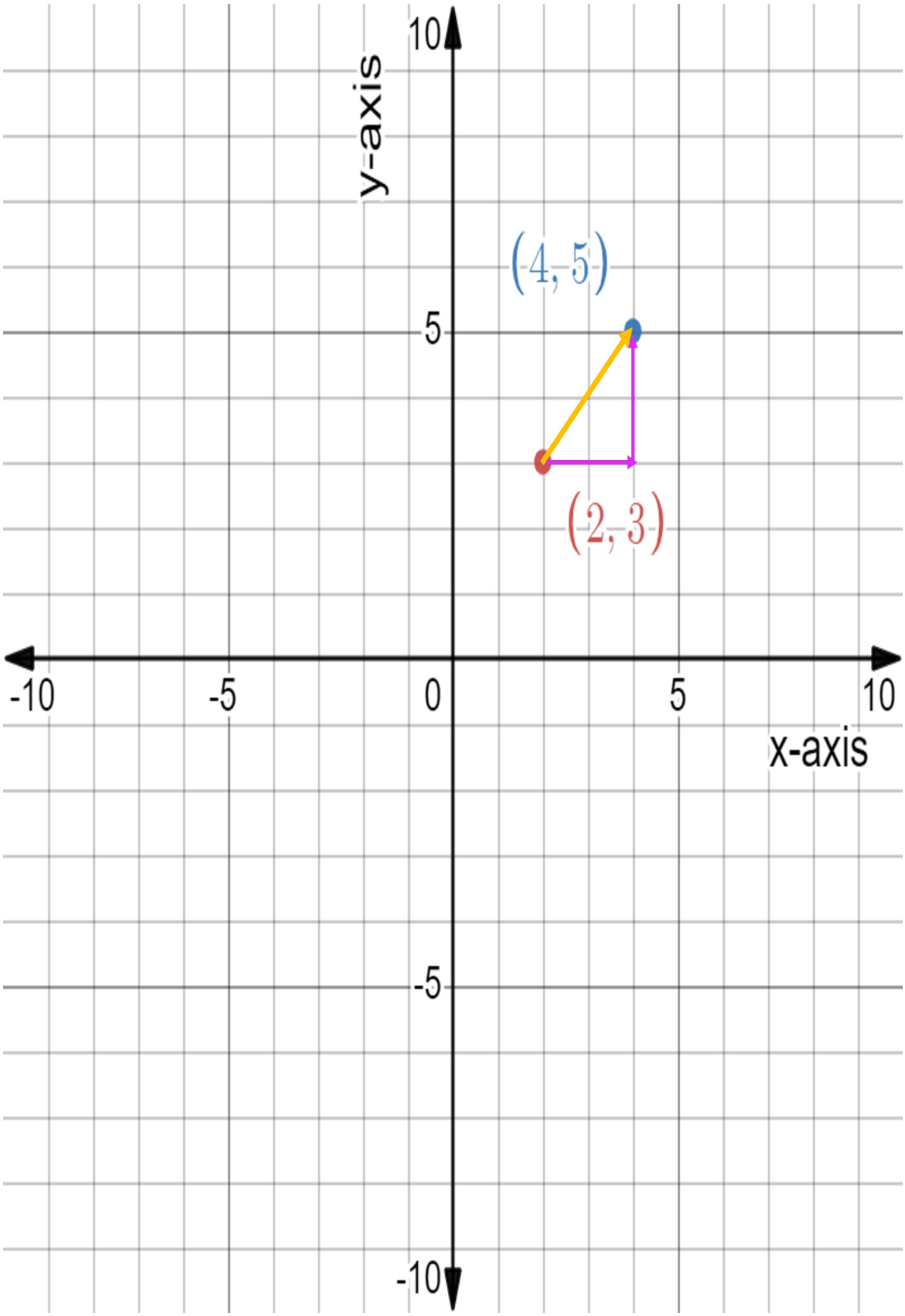
10
y-axis
(4, 5)
5
(2, 3)
-10
-5
0
5
10
x-axis
-5
-10

*Figure 4-4. The change in x and the change in y for vector AB*

Figure 4-5 shows the change in *x* and the change in *y* in the case of vector BA.

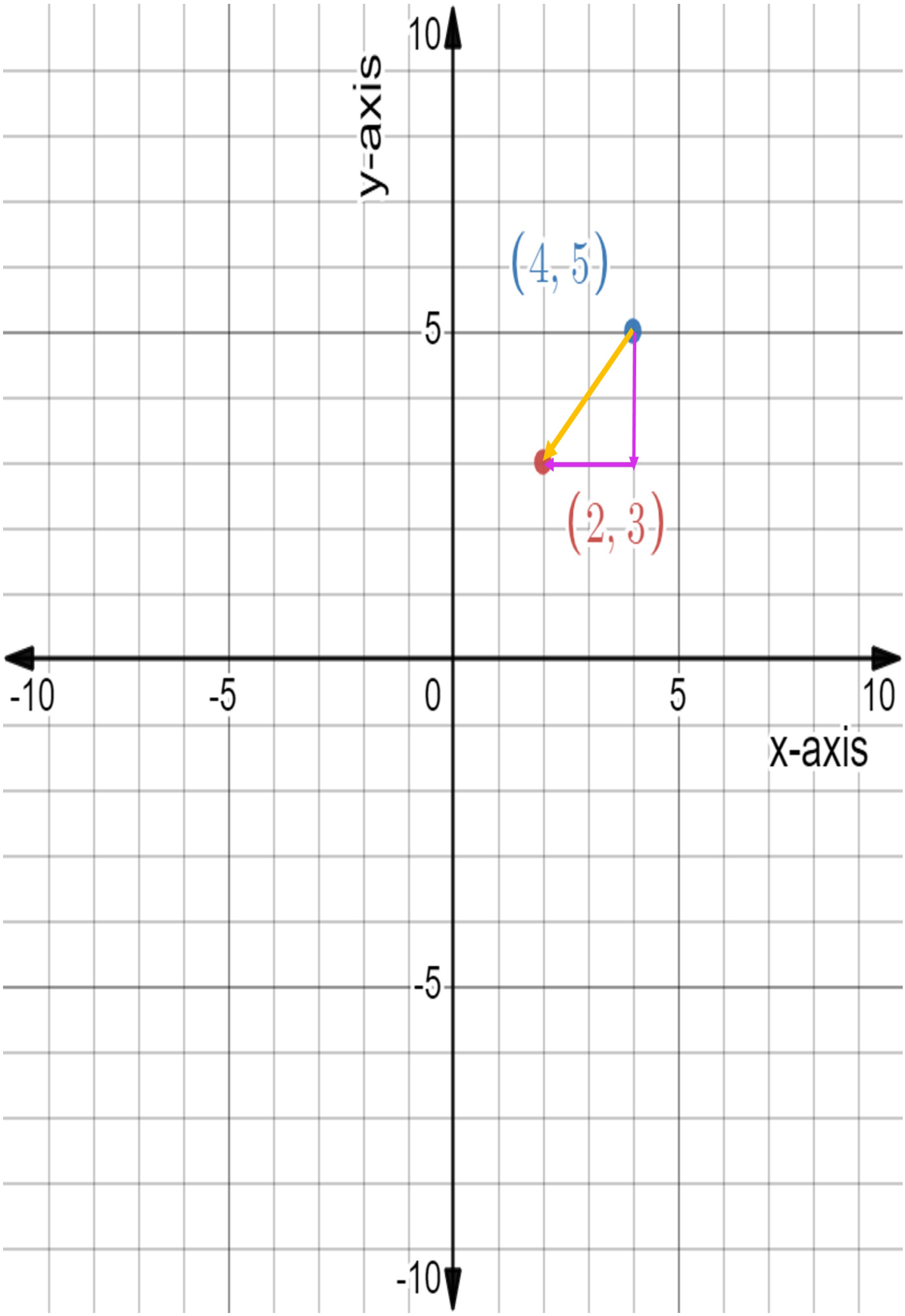
y-axis
10
5
-5
-10
-10
-5
0
5
10
x-axis
(4, 5)
(2, 3)

*Figure 4-5. The change in x and the change in y for vector BA*

> **NOTE**
>
> A vector that has a magnitude of 1 is referred to as a *unit vector*.

Researchers typically use vectors as representations of speed especially in engineering. Navigation is one field that heavily relies on vectors. It allows navigators to determine their positions and plan their destinations. Naturally, magnitude represents speed and the direction represents the destination.

You can add and subtract vectors from each other and from scalars. This allows for a shift in direction and magnitude. What you should retain from the previous discussion is that vectors indicate directions between different points on the axis.

> **NOTE**
>
> A *scalar* is a value with magnitude but no direction. Scalars, as opposed to vectors, are used to represent elements like temperature and prices. Basically, scalars are numbers.

In machine learning, the *x*-axis and *y*-axis respectively represent data and the model's results. The independent (predictor) variable is represented by the *x*-axis in a scatter plot, and the dependent (forecast) variable is represented by the *y*-axis.

A *matrix* is a row-and-column-organized rectangular array containing numbers[1]. Matrices are useful in computer graphics and other domains as well as to define and manipulate linear systems of equations. What differentiates a matrix from a vector? The simplest answer is that a vector is a matrix with a single column or a single row. Here's a a basic example of a 3 x 3 matrix:

$$\begin{bmatrix} 5 & 2 & 9 \\ -8 & 10 & 13 \\ 1 & 5 & 12 \end{bmatrix}$$

Matrices' sizes are referred to using their rows and columns respectively. A row

is a horizontal line and a column is a vertical line. The following representation is a 2 x 4 matrix:

$$\begin{bmatrix} 5 & 2 & 1 & 3 \\ -8 & 10 & 9 & 4 \end{bmatrix}$$

The following representation is another example of a matrix. This time it is a 4 x 2 matrix:

$$\begin{bmatrix} 5 & 2 \\ -8 & 10 \\ 8 & 22 \\ 7 & 3 \end{bmatrix}$$

**NOTE**

Matrices are heavily used in machine learning. Rows generally represent time and columns represent features.

The summation of different matrices is straightforward but must be used only when the matrices match in size (which means they have the same number of columns and rows). For instance, let's add the two following matrices:

$$\begin{bmatrix} 1 & 2 \\ 5 & 8 \end{bmatrix} + \begin{bmatrix} 3 & 9 \\ 1 & 5 \end{bmatrix} = \begin{bmatrix} 4 & 11 \\ 6 & 13 \end{bmatrix}$$

You can see that to add two matrices, you simply have to add the numbers in the same positions. Now, if you try to add the two following matrices, you won't be able to do it as there is a mismatch in what to add:

$$\begin{bmatrix} 8 & 3 \\ 3 & 2 \end{bmatrix} + \begin{bmatrix} 3 & 9 \\ 1 & 5 \\ 5 & 4 \end{bmatrix}$$

The subtraction of matrices is also straighforward and follows the same rules as the summation of matrices. Let's take the following example:

$$\begin{bmatrix} 5 & 2 \\ -8 & 10 \end{bmatrix} - \begin{bmatrix} 3 & 9 \\ -1 & -5 \end{bmatrix} = \begin{bmatrix} 2 & -7 \\ -9 & 15 \end{bmatrix}$$

Evidently, subtraction of matrics is also a summation of matrices with a change of signals in one of them.

Matrix multiplication by a scalar is quite simple. Let's take the following example:

$$3 \times \begin{bmatrix} 5 & 2 \\ 8 & 22 \end{bmatrix} = \begin{bmatrix} 15 & 6 \\ 24 & 66 \end{bmatrix}$$

So basically, you are multiplying every element in the matrix by the scalar. Matrix multiplication by another matrix is a bit more complicated since they use the *dot product* method. First of all, in order to multiply two matrices together, they must satisfy this condition:

$$Matrix_{xy} \times Matrix_{yz} = Matrix_{xz}$$

This means that the first matrix must have a number of columns equal to the number of rows in the second matrix, and the resulting matrix from the dot product is a matrix that has the number of rows of the first matrix and the number of columns of the second matrix. The dot product is explained in this example representation of a 1 x 3 and 3 x 1 matrix multiplication (notice the equal number of columns and rows):

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \times \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} (1 \times 3) + (2 \times 2) + (3 \times 1) \end{bmatrix} = \begin{bmatrix} 10 \end{bmatrix}$$

Let's take another example of a 2 x 2 matrix multiplication:

$$\begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} 3 & 0 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 7 & 2 \\ 2 & 1 \end{bmatrix}$$

There is a special type of matrix called the *identity matrix* which is basically the number 1 for matrices. It is defined as follows for a 2 x2 dimension:

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

And as follows for a 3 x 3 dimension:

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Multiplying any matrix by the identity matrix yields the same original matrix. This is why it can be referred to as the 1 of matrices (multiplying any number by 1 yields the same number). It is worth noting that matrix multplication is not commutative, which means that the order of multiplication changes the result such as:

$$AB \neq BA$$

*Matrix transposing* is a process that involves changing the rows of into columns and vice versa. The transpose of a matrix is obtained by reflecting the matrix along its main diagonal:

$$\begin{bmatrix} 4 & 6 & 1 \\ 1 & 4 & 2 \end{bmatrix}^T = \begin{bmatrix} 4 & 1 \\ 6 & 4 \\ 1 & 2 \end{bmatrix}$$

Transposing is used in some machine learning algorithms and is not an uncommon operation when dealing with such models. If you are wondering about the role of matrices in data science and machine learning, you can refer to this non-exhaustive list:

*Representation of data*

> Matrices often represent data with rows representing samples and columns representing features. For example, a row in a matrix can present OHLC data in one time step.

*Linear algebra*

> Matrices and linear algebra are intertwined, and many learning algorithms use the concepts of matrices in their operations.

*Data relationship matrices*

If you remember from Chapter 3, covariance and correlation measures are often represented as matrices. These relationship calculations are important concepts in machine learning.

**NOTE**

You should retain the following key concepts from this section:

- A vector is an object that has a magnitude (length) and a direction (arrowhead). Multiple vectors grouped together form a matrix.
- A matrix can be used to store data. It has its special ways of performing operations.
- Matrix multiplication uses the dot product method.
- Transposing a matrix means to swap its rows and its columns.

## Introduction to Linear Equations

You have already seen an example of a linear equation in the section that discusses linear regression and statistical inference from Chapter 3. *Linear equations* are basically formulae that present an equality relationship between between different variables and constants. In the case of machine learning, it is often a relationship between a dependent variable (the output) and the independent variable (the input). The best way to understand linear equations is through examples.

**NOTE**

The aim of linear equations is to find an unknown variable, usually denoted by the letter *x*.

Let’s see a very basic example which you can consider as a first building block towards more advanced calculus concepts you will see later on. The following examples requires finding the value of *x*:

$$10x = 20$$

You should understand the equation as "*10 times which number equals 20?”*. When a constant is directly attached to a variable such as *x*, it refers to a multiplication operation. Now, to solve for *x* (so, finding the value of *x* that equalizes the equation), you have an obvious solution which is to get rid of 10 so that you have x on one side of the equation and the rest on the other side.

Naturally, to get rid of 10, you divide by 10 so that what remains is 1 which if multiplied by the variable *x* does nothing. However, keep in mind two important things:

- If you do a mathematical operation on one side of an equation, you must do it on the other side as well. This is why they are called equations.
- For simplicity, instead of dividing by the constant to get rid of it, you should multiply it by its reciprocal.

The *reciprocal* of a number is one divided by that number. Here’s the mathematical representation of it:

$$Reciprocal\,(x) = \frac{1}{x}$$

Now, back to the example, to find *x*, you can do the following:

$$\left(\frac{1}{10}\right) 10x = 20 \left(\frac{1}{10}\right)$$

Performing the multiplication and simplifying gives the following result:

$$x = 2$$

This means that the solution of the equation is 2. To verify this, you just need to plug it into the original equation as follows:

$$10 \times 2 = 20$$

Therefore, it takes two 10’s to get 20.

> **NOTE**
>
> Dividing the number by itself or multiplying it by its reciprocal is the same thing.

Let’s take another example of how to solve *x* through linear techniques. Consider the following problem:

$$\frac{8}{6}x = 24$$

Performing the multiplication and simplifying gives the following result:

$$\left(\frac{6}{8}\right)\frac{8}{6}x = 24\left(\frac{6}{8}\right)$$

$$x = 18$$

This means that the solution of the equation is 18. To verify this, you just need to plug it into the original equation as follows:

$$\frac{8}{6} \times 18 = 24$$

Let's dig in a little deeper, because typically, linear equations are not this simple. Sometimes they contain more variables and more constants, which need more detailed solutions, but let's keep taking it step by step. Consider the following example:

$$3x - 6 = 12$$

Solving for *x* requires rearranging the equation a little bit. Remember, the aim is to leave *x* on one side and the rest on the other. Here, you have to get rid of the constant 6 before taking care of 3. The first part of the solution is as follows:

$$3x - 6(+6) = 12(+6)$$

Notice how you have to add 6 to both parts of the equation. The part on the left will cancel itself out while the part on the right will add up to 18:

$$3x = 18$$

Finally, you're all set to multiply by the reciprocal of the constant attached to the variable *x*:

$$\left(\frac{1}{3}\right)3x = 18\left(\frac{1}{3}\right)$$

Simplifying and solving for *x* leaves the following solution:

$$x = 6$$

This means that the solution of the equation is 6. To verify this, just plug it into the original equation as follows:

$$(3 \times 6) - 6 = 12$$

By now, you should start noticing that linear algebra is all about using shortcuts

and quick techniques to simplify equations and find unknown variables. The next example shows how sometimes the variable *x* can occur in multiple places:

$$6x + x = 27 - 2x$$

Remember, the main focus is to have *x* on one side of the equation and the rest on the other side:

$$6x + x + 2x = 27$$

Adding the constants of *x* gives you the following:

$$9x = 27$$

The final step is dividing by 9 so that you only have *x* remaining:

$$x = 3$$

You may now verify this by plugging 3 in the place of *x* in the original equation. You will notice that both sides of the equation will be equal.

**NOTE**

Even though this section is quite simple, it contains the basic foundations you need to start advancing in algebra and calculus. The main points to retain before going further are as follows:

- A linear equation is a representation in which the highest exponent on any variable is one. This means that there are no variables that are raised to the power of two and above.
- A linear equation line is straight when plotted on a chart.
- The application of linear equations in modeling a wide range of real-world occurrences makes them crucial in many branches of mathematics and research. They are also widely utilized in machine learning.
- Solving for *x* is the process of finding its value that equalizes both sides of the equation.
- When performing an operation (such as adding a constant or multiplying by a constant) on one side of the equation, you have to do it on the other side as well.
- Reciprocals are useful to simplify equations.

## Systems of Equations

A *system of equation* is when there are two or more equations working together to solve a variable or more. Therefore, instead of the usual single equation like the following:

$$x + 10 = 20$$

Systems of equations resemble the following:

$$x + 10 = 20$$
$$y + 2x = 10$$

There are methods that solve them and particularities, which are discussed in this section. Systems of equations are useful in machine learning and are used in many of its aspects.

Let’s look at the previous system of equation from the beginning of this section and solve it graphically. Plotting the two functions can actually give the solution directly. The point of intersection of linear equations is the solution. Therefore, the coordinates of the intersection (*x*, *y*) refer to the solutions of the *x* and *y* respectively.

From Figure 4-6, it seems that *x* = 10 and *y* = -10. Plugging these values into their respective variables gives the correct answer:

$$10 + 10 = 20$$
$$(-10) + (2 \times 10) = 10$$

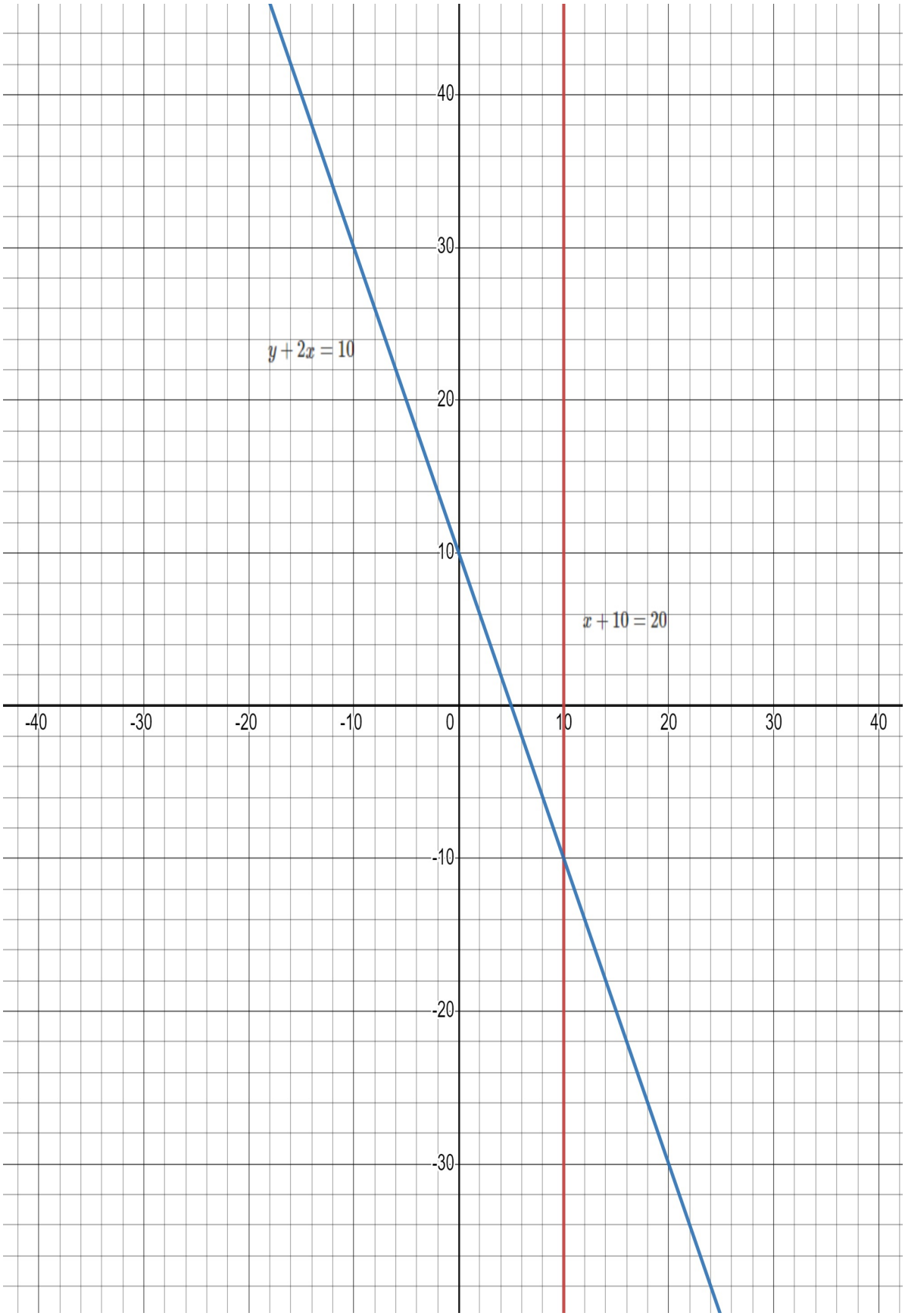
40
30
20
10
-10
-20
-30
-40
-30
-20
-10
0
10
20
30
40
$y + 2x = 10$
$x + 10 = 20$

*Figure 4-6. A graph showing the two functions and their intersection (solution)*

As the functions are linear, there can be three cases to solving them:

1. There is only one solution for each variable.
2. There is no solution. This occurs when the functions are *parallel* (this means that they never intersect).
3. The is an infinite number of solutions. This occurs when it's the same function (since all points fall on the straight line).

Before moving on to solving systems of equations using algebra, let's visually see how can there be no solution and how can there be an infinite number of solutions. Consider the following system:

$$2x = 10$$
$$4x = 20$$

Figure 4-7 charts the two together. Since they are exactly the same equation, they fall on the same line. In reality, there are two lines in Figure 4-7, but since they are the same, they are indistinguishable. For every *x* on the line, there is a corresponding *y*.

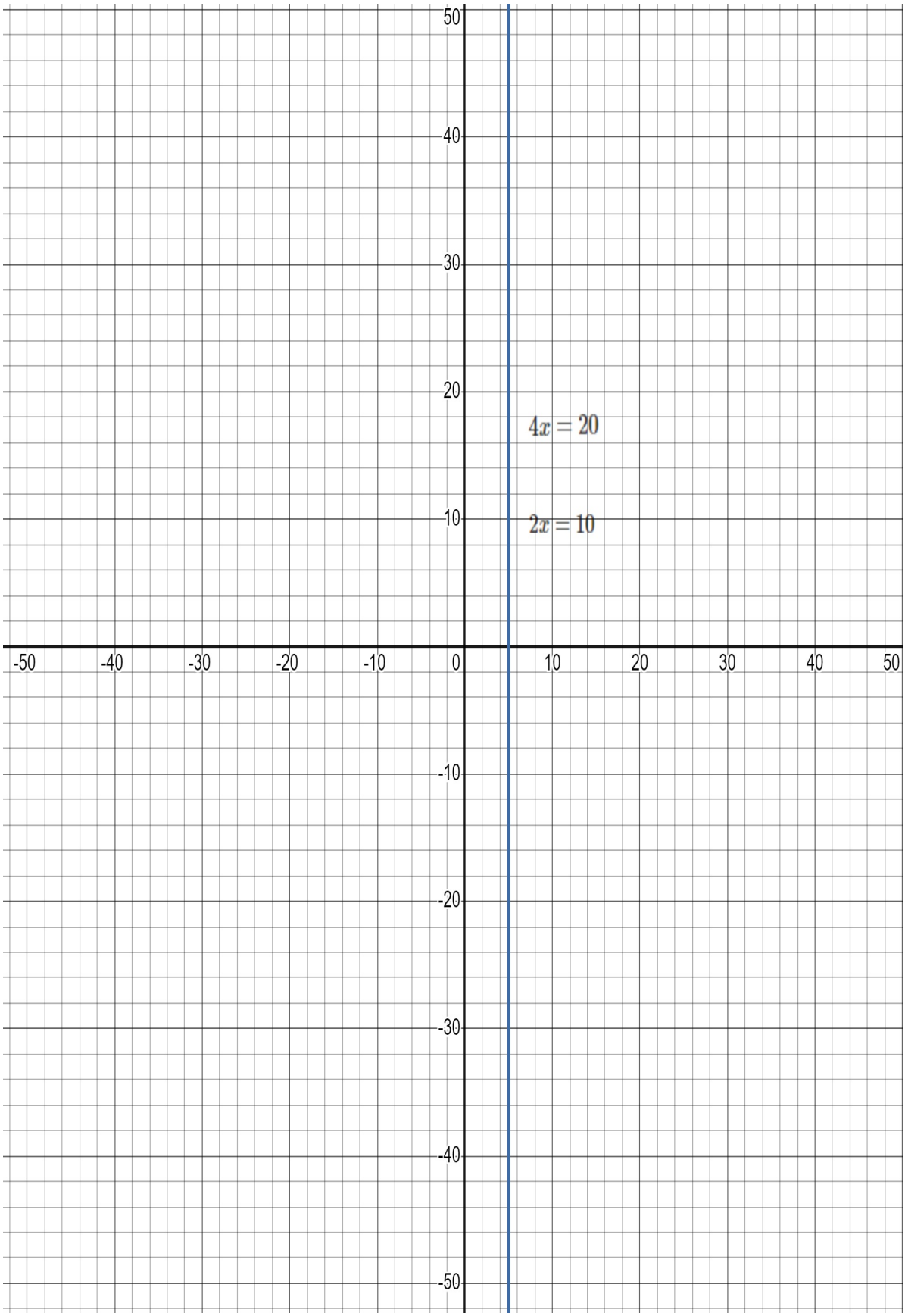
50
40
30
20
10
-50
-40
-30
-20
-10
0
10
20
30
40
50
-10
-20
-30
-40
-50
$4x = 20$
$2x = 10$

*Figure 4-7. A graph showing the two functions and their infinite intersections*

Now, consider the following system:

$$3x = 10$$
$$6x = 10$$

Figure 4-8 shows how they never intersect which is intuitive as you cannot multiply the same number (represented by the variable *x*) with different numbers and expect to get the same result.

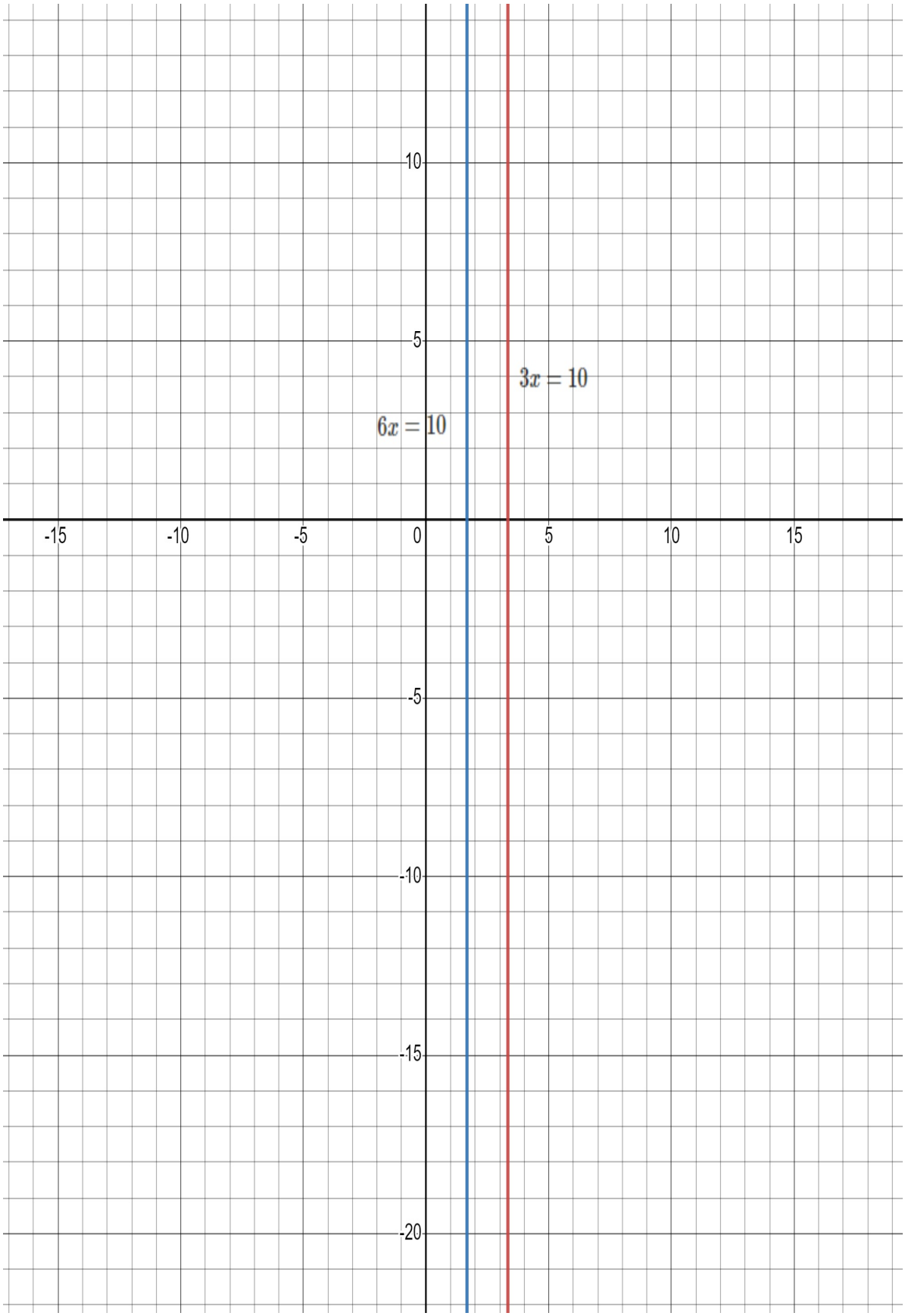
10
5
-5
-10
-15
-20
-15
-10
-5
0
5
10
15
$3x = 10$
$6x = 10$

*Figure 4-8. A graph showing the two functions and their impossible intersection*

Algebraic methods are used when there are more than two variables since they cannot be solved through graphs. This mainly entails two methods, substitution and elimination.

*Substitution* is used when you can replace the value of a variable in one equation and plug it in the second equation. Consider the following example:

$$x + y = 2$$

$$10x + y = 10$$

The easiest method is to rearrange the first equation so that you have *y* in terms of *x*:

$$y = 2 - x$$

$$10x + (2 - x) = 10$$

Solving for *x* in the second equation becomes simple:

$$10x + (2 - x) = 10$$

$$10x + 2 - x = 10$$

$$10x - x = 10 - 2$$

$$9x = 8$$

$$x = \frac{8}{9}$$

$$x = 0.8889$$

Now that you have found the value of *x*, you can easily find *y* by plugging the value of *x* in the first equation:

$$0.8889 + y = 2$$

$$y = 2 - 0.8889$$

$$y = 1.111$$

To check if your solution is correct, you can plug in the values of x and y in both formulas:

$$0.8889 + 1.111 = 2$$

$$(10 \times 0.8889) + 1.111 = 10$$

Graphically this means that the two equations intersect at (0.8889, 1.111). This technique can be used with more than two variables. Follow the same process until the equations are simplified enough to give you the answers. The issue with substitution is that it may take some time when you're dealing with more than two variables.

*Elimination* is a faster alternative. It is about eliminating variables until there is only one left. Consider the following example:

$$2x + 4y = 20$$

$$3x + 2y = 10$$

Noticing that there is 4*y* and 2*y*, it is possible to multiply the second equation by 2 so that you can subtract the equations from each other (which will remove the *y* variable):

$$2x + 4y = 20$$

$$6x + 4y = 20$$

Subtracting the two equations from each other gives the following result:

$$-4x = 0$$

$$x = 0$$

Therefore, $x = 0$. Graphically, this means that they intersect whenever $x = 0$ (exactly at the vertical *y* line). Plugging in the value of *x* in the first formula gives $y = 5$:

$$(2 \times 0) + 4y = 20$$

$$4y = 20$$

$$y = 5$$

Similarly, elimination can also solve equations with three variables. The choice between substitution and elimination depends on the type of equations.

**NOTE**

Key takeaways from this section can be summed up as follows:

- Systems of equations solve variables together. They are very useful in machine learning and are

used in some algorithms.

- Graphical solutions are preferred for simple systems of equations.
- Solving systems of equations through algebra entails the use of substitution and elimination methods.
- Substitution is preferred when the system is simple but elimination is the way to go when the system is a bit more complex.

## Trigonometry

*Trigonometry e*xplores the behavior of what is known as *trigonometric functions* that relate the angles of a triangle to the lengths of its sides. The most-used triangle is the right-angled triangle which has one angle at 90°. Figure 4-9 shows an example of a right-angled triangle.

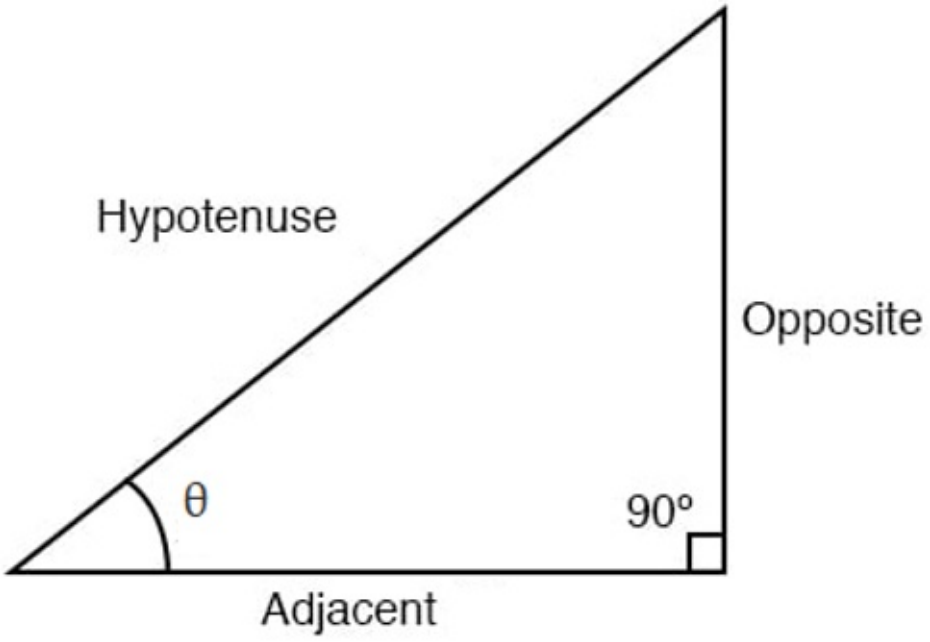


*Figure 4-9. A right-angled triangle*

Let’s define the main characteristics of a right-angled triangle:

- The longest side of the triangle is called a *hypotenuse.*
- The angle in front of the hypotenuse is the right angle (the one at 90°).

- Depending on the other angle (θ) you choose (from the two that remains), the line between this angle and the hypotenuse is called the *adjacent* and the other line is called the *opposite*.

Trigonometric functions are simply the division of a line by the other. Remember that you have three lines in a triangle (hypotenuse, opposite, and adjacent). The trigonometric functions are found as follow:

$$sin\left(\theta\right) = \frac{Opposite}{Hypotenuse}$$

$$cos\left(\theta\right) = \frac{Adjacent}{Hypotenuse}$$

$$tan\left(\theta\right) = \frac{Opposite}{Adjacent}$$

From the previous three trigonometric functions, it is possible to extract a trigonometric identity which reaches *tan* from *sin* and *cos* using basic linear algebra:

$$tan\left(\theta\right) = \frac{sin(\theta)}{cos(\theta)}$$

*Hyperbolic functions* are similar to trigonometric operations but are defined using exponential functions.

> **NOTE**
>
> This part on hyperbolic functions is extremely important as it forms the basis of what is known as *activation functions*, a key concept in neural networks, the protagonists of deep learning models. You will see them in detail in Chapter 8.

Euler's number (denoted as *e*) is one of the most important numbers in mathematics. It is an *irrational number*, which is a real number that cannot be expressed as a fraction. The word *irrational* comes from the fact that there is no *ratio* to express it; it has nothing to do with its personality. Euler's number *e* is also the base of the natural logarithm *ln* and the first digits of it are 2.71828. One of the best approximations to get *e* is the following formula:

$$e = \left(1 + \frac{1}{n}\right)^n$$

By increasing *n* in the previous formula, you will approach the value of *e*. It has many interesting properties, notably the fact that its slope is its own value. Let's take for example the following function (also called the natural exponent function):

$$f(x) = e^x$$

At any point, the slope of the function is the same value. Take a look at Figure 4-10.

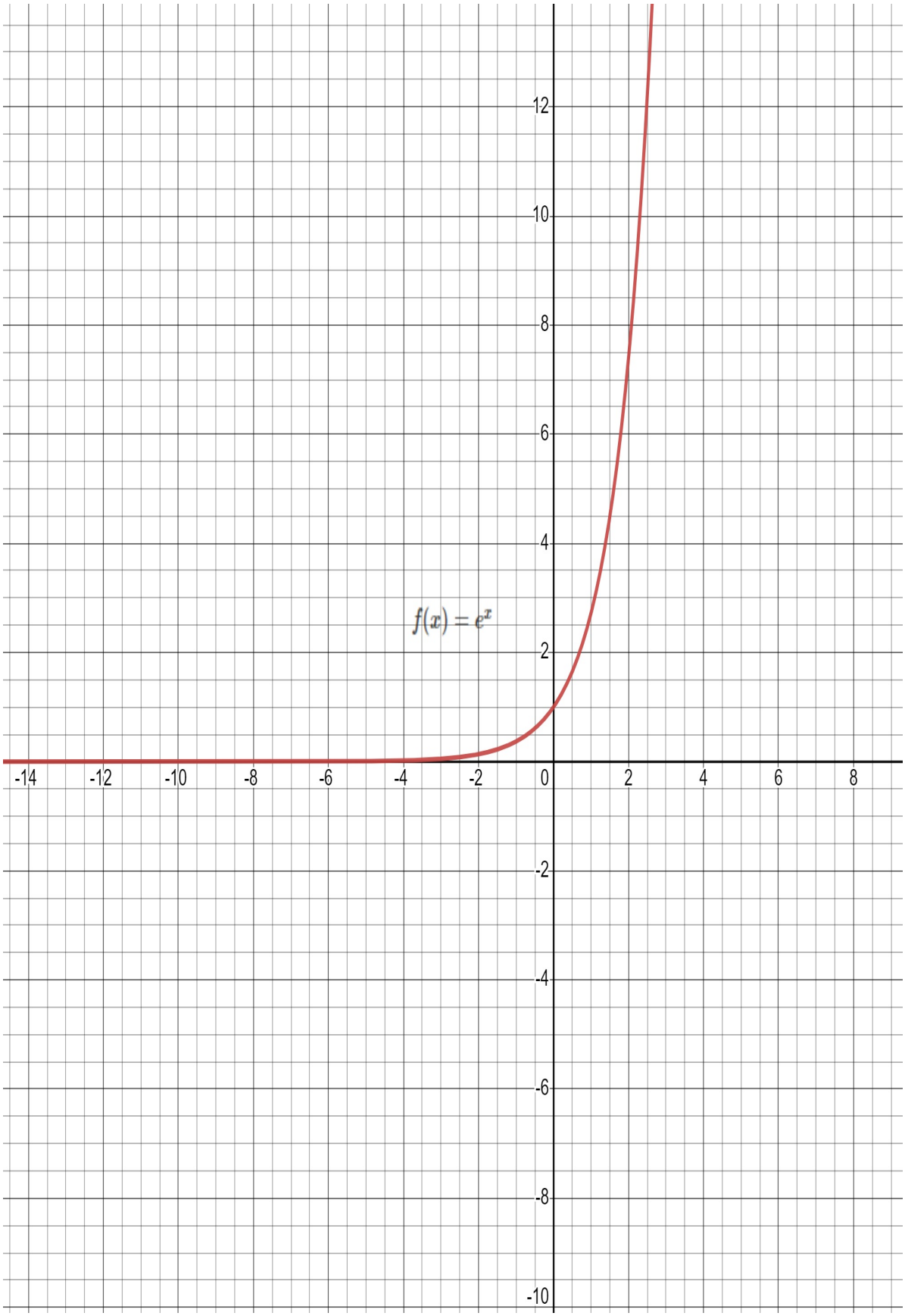
12
10
8
6
4
2
-14
-12
-10
-8
-6
-4
-2
0
2
4
6
8
-2
-4
-6
-8
-10
$f(x) = e^x$

*Figure 4-10. The graph of the natural exponent function*

**NOTE**

You may be wondering about the use of explaining exponents and logarithms in this book. There are mainly two reasons for this:

- Exponents and more importantly Euler's number are used in hyperbolic functions where *tanh(x)* is one of the main activation functions for neural networks, a type of machine and deep learning model.
- Logarithms are very useful in *data normalization* but also in *loss functions*, concepts that you will see in later chapters.

Therefore, having a deep understanding of what they refer to is primordial in building up expertise in the subsequent models.

Hyperbolic functions use the natural exponent function and are defined as follows:

$$sinh\,(x) = \frac{e^{x} - e^{-x}}{2}$$

$$cosh\,(x) = \frac{e^{x} + e^{-x}}{2}$$

$$tanh\,(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

Among the key characteristics of *tanh(x)* is non-linearity, the limitation between [-1, 1], and the fact that it is centered at zero. Figure 4-11 shows the graph of *tanh(x)*.

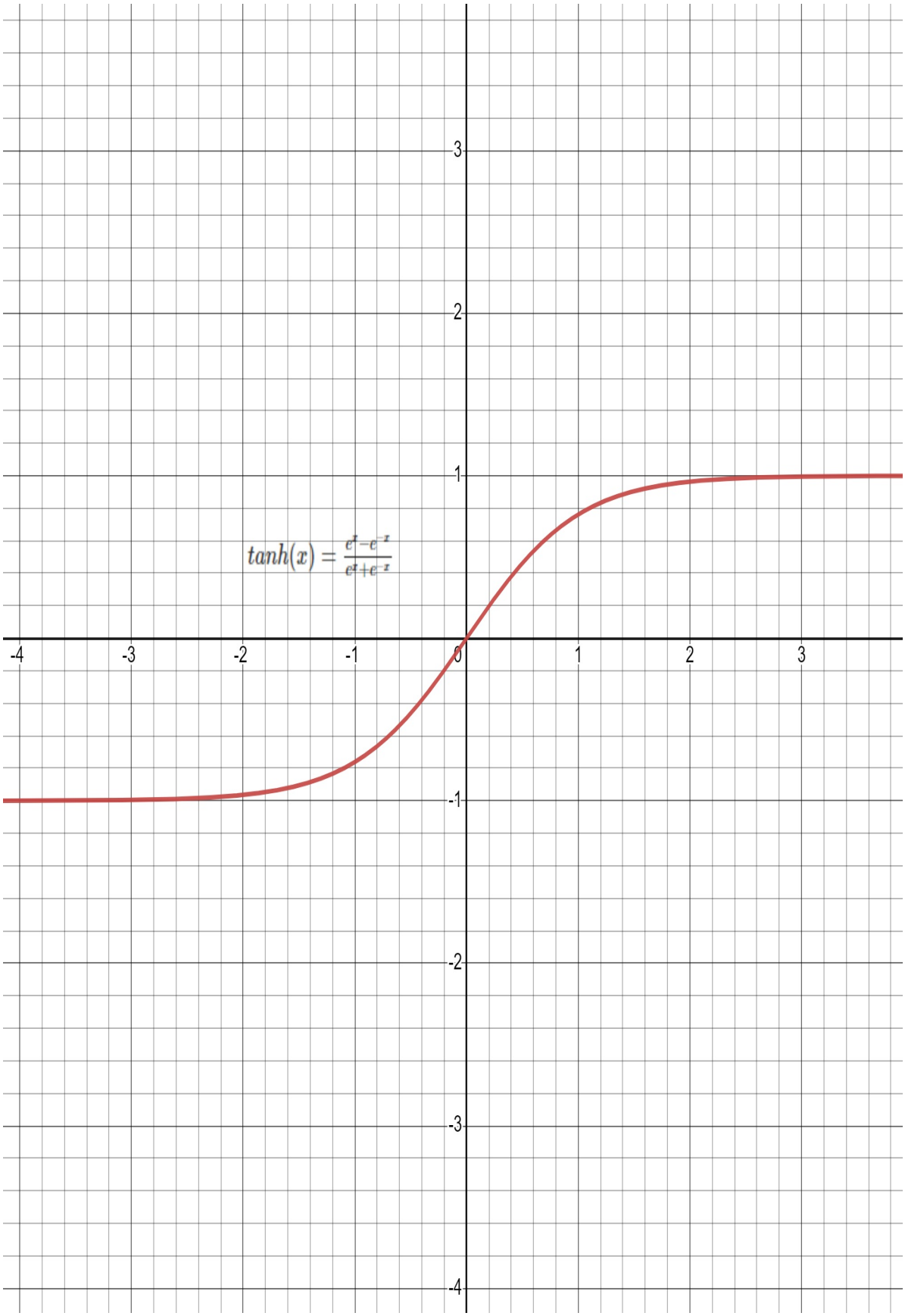
$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
3
2
1
-1
-2
-3
-4
-4
-3
-2
-1
0
1
2
3

*Figure 4-11. The graph of tanh(x) showing how it's limited between -1 and 1*

**NOTE**

Key concepts to retain from this section are summed up as follows:

- Trigonometry is a field that explores the behavior of trigonometric functions which relate the angles of a triangle to the lengths of its sides.
- A trigonometric identity is a shortcut that relates the trigonometric functions with each other.
- Eurler's number *e* is irrational and is the base of the natural logarithm. It has many applications in exponential growth and in hyperbolic functions.
- Hyperbolic functions resemble trigonmetric functions but are not the same thing. While trigonometric functions relate to a triangles and circles, hyperbolic functions relate to hyperbolas.
- The hyperbolic tangent function is used in neural networks, a deep learning algorithm.

## Limits and Continuity

*Calculus works by making visible the infinitesimally small.*

*- Keith Devlin*

Let's move now to calculus after seeing the major topics of linear algebra. Limits don't have to be nightmarish. I have always found them to be misunderstood. They are actually quite easy to get. But first, you need motivation and this comes from knowing the added value of learning limits.

Understanding limits is very important in machine learning models for many reasons:

*Optimization*

In optimization methods like gradient descent, limits can be used to regulate the step size and guarantee convergence to a local minimum (a concept you will learn in Chapter 8).

*Feature Selection*

Limits can be used to rank the significance of various model features and perform feature selection, which can make the model simpler and perform better.

*Sensitivity analysis*

A machine learning model's sensitivity to changes in input data and its capacity to generalize to new data can be used to examine a model's behavior.

Also, limits are used in more advanced calculus concepts you will run across in the coming pages.

The main aim of limits is to know the value of a function when it's undefined. But what is an undefined function? When you have a function that gives a solution that is not possible (such as dividing by zero), limits help you bypass this issue in order to know the value of the function at that point. So the aim of limits is to solve functions even when they are undefined.

Remember that the solution to a function that takes *x* as an input is a value in the *y* axis. Figure 4-12 shows a linear graph of the following function:

$$f(x) = x + 2$$

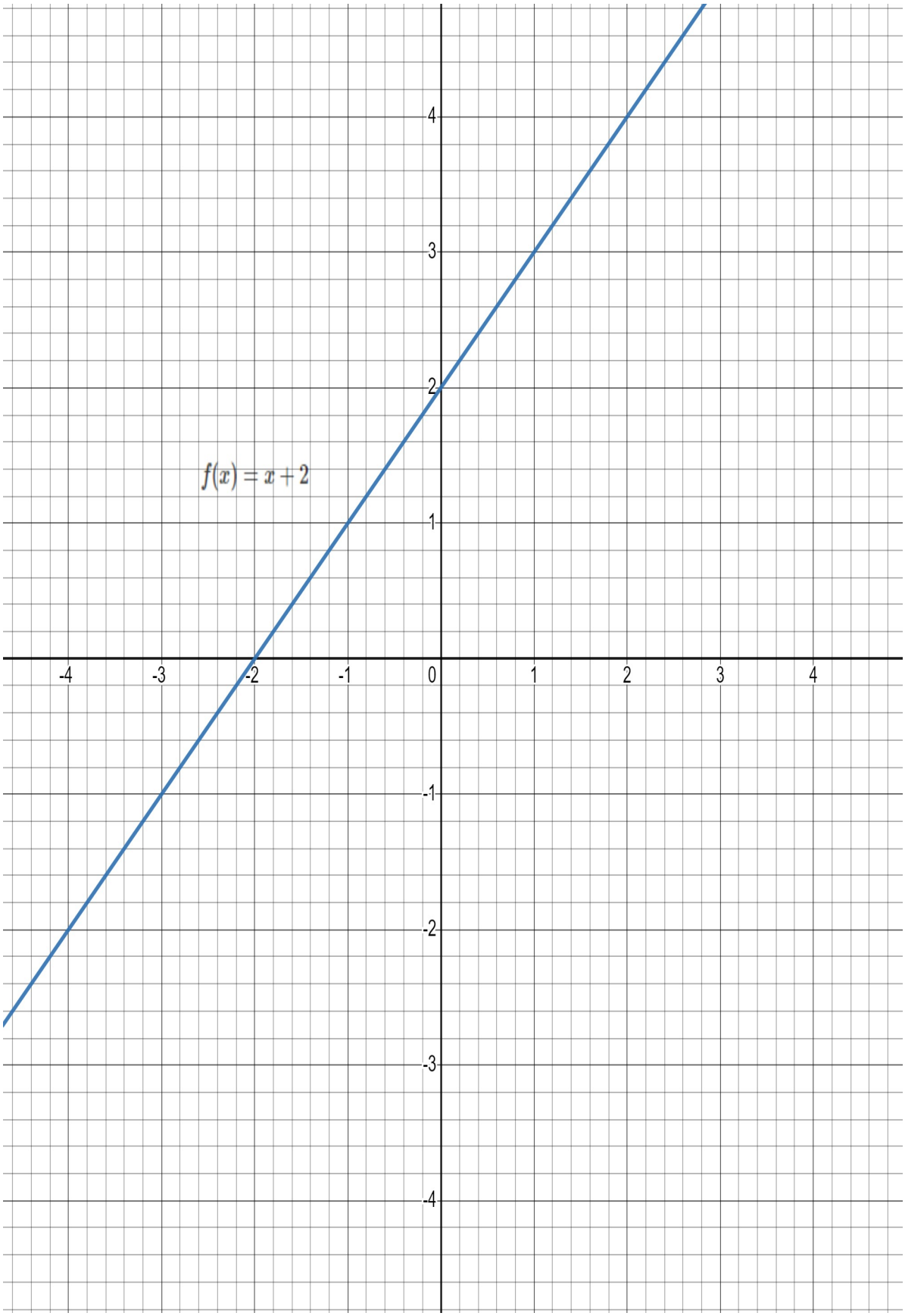
$f(x) = x + 2$
4
3
2
1
-1
-2
-3
-4
-4
-3
-2
-1
0
1
2
3
4

*Figure 4-12. The graph of the function f(x) = x + 2*

The solution of the function in the graph is the one that lies on the linear line taking into account the value of *x* every time.

What would be the solution of the function (the value of *y*) when *x* = 4? Clearly, the answer is 6, as substituting the value of *x* by 4 gives 6.

$$f(4) = 4 + 2 = 6$$

Thinking of this solution in terms of limits would be saying, what is the solution of the function as *x* approaches 4 from both sides (the negative side and the positive side)? Table 4-1 simplifies this dilemma:

*Table 4-1. Finding x*

| *f(x)* | *x* |
|---|---|
| 5.998 | 3.998 |
| 5.999 | 3.999 |
| 6.000 | 4 |
| 6.001 | 4.001 |
| 6.002 | 4.002 |

Approaching from the negative side is the equivalent of adding a fraction of a number while below 4 and analyzing the result every time. Similarly, approaching from the positive side is the equivalent of removing a fraction of a number while above 4 and analyzing the result every time. The solution seems to converge to 6 as x approaches 4. This is the solution to the limit.

Limits in the general form are written following this convention:

$$\lim_{x \to a} f(x) = L$$

The general form of the limit is read as follows: as you approach *a* along the *x*-axis (whether from the positive or the negative side), the function *f(x)* gets closer

to the value of *L*.

> **NOTE**
>
> The idea of the limit states that as you lock-in and approach a number from either side (negative or positive), the solution of the equation approaches a certain number, and the solution to the limit is that number.

As mentioned previously, limits are useful when the exact point of the solution is undefined using the conventional way of substitution.

A one-sided limit is different from the general limit. The left-hand limit is where you search for the limit going from the negative side to the positive side, and the right-hand limit is when you search for the limit going from the positive side to the negative side. The general limit exists when the two one-sided limits exist and are equal. Therefore, the previous statements are summarized as follows:

- The left-hand limit exists.
- The right-hand limit exists.
- The left-hand limit is equal to the right-hand limit.

The left hand limit is defined as follows:

$$\lim_{x \to a^-} f(x) = L$$

The right hand limit is defined as follows:

$$\lim_{x \to a^+} f(x) = L$$

Let’s take the following equation:

$$f(x) = \frac{x^3 - 27}{x - 3}$$

What is the solution of the function when x = 3? Substitution leads to the following issue:

$$f(3) = \frac{3^3 - 27}{3 - 3} = \frac{27 - 27}{3 - 3} = \frac{0}{0} = Undefined$$

However, thinking about this in terms of limits as Table 4-2 shows, it seems that as you approach *x = 3*, either from the left side or the right side, the solution

tends to approach 27.

*Table 4-2. Finding x*

| f(x) | x |
|---|---|
| 2.9998 | 26.9982 |
| 2.9999 | 26.9991 |
| 3.0000 | Undefined |
| 3.0001 | 27.0009 |
| 3.0002 | 27.0018 |

Graph-wise, this can be seen as a discontinuity in the chart along both axes. The discontinuity exists on the line around the coordinate (3, 27).

Some functions do not have limits. For example, what is the limit of the following function as *x* approaches 5?

$$\lim_{x \to 5} \frac{1}{x-5}$$

Looking at Table 4-3, it seems that as *x* approaches 5, the results highly diverge when approaching from both sides. For instance, approaching from the negative side, the limit of 4.9999 is -10,000 and from the positive side, the limit of 5.0001 is 10,000.

| f(x) | x |
|---|---|
| 4.9998 | -5000 |
| 4.9999 | -10000 |
| 5.0000 | Undefined |
| 5.0001 | 10000 |

| | |
|---|---|
| 5.0002 | 5000 |

Remember that for the general limit to exist, both one-sided limits must exist and must be equal, which is not the case here. Graphing this gives Figure 4-13, which may help you understand why the limit does not exist.

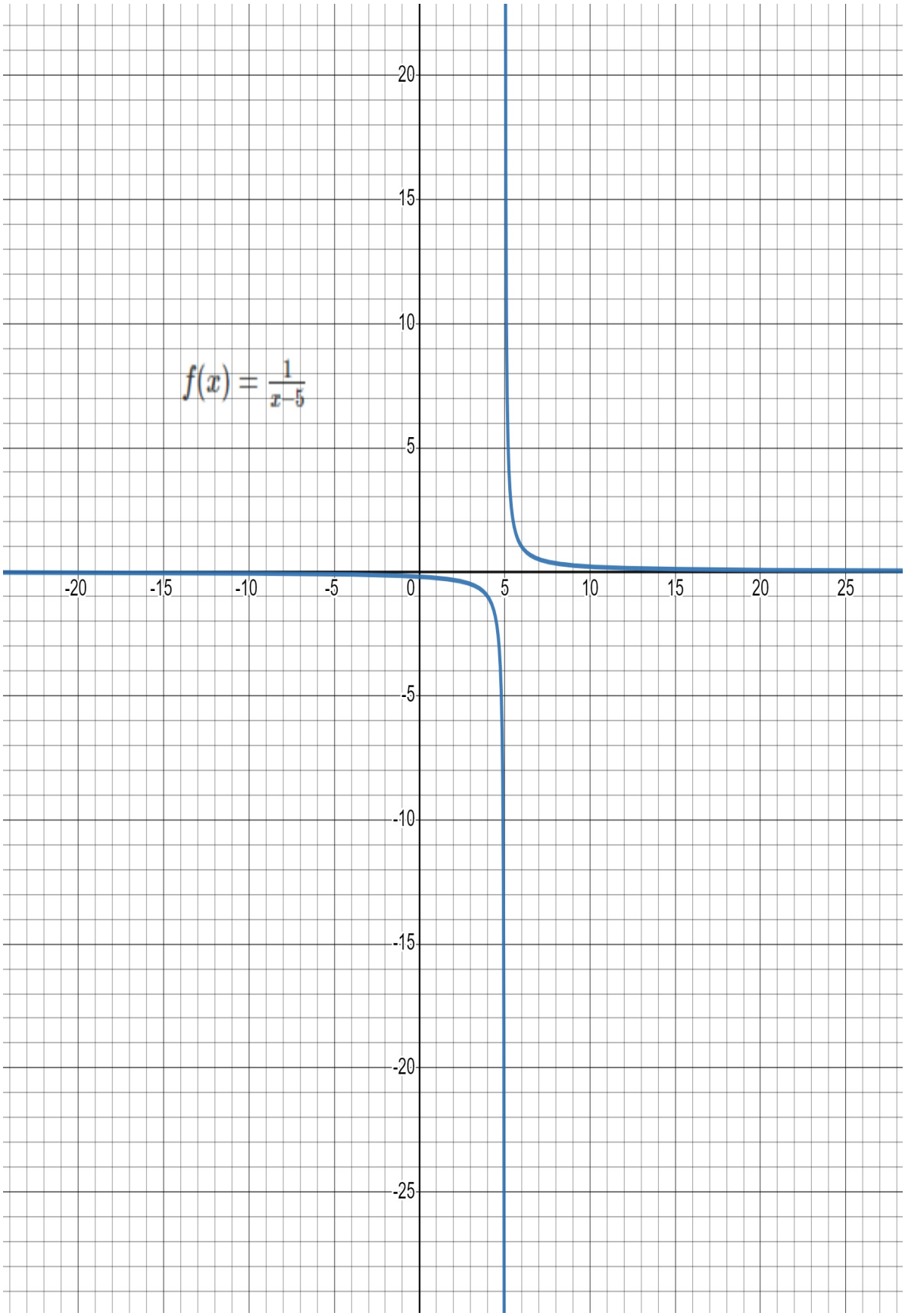
$f(x) = \frac{1}{x-5}$
20
15
10
5
-5
-10
-15
-20
-25
-20
-15
-10
-5
0
5
10
15
20
25

*Figure 4-13. The graph of the function proving that the limit does not exist*

But what if the function that you want to analyze looks like this:

$$\lim_{x \to 5} \frac{1}{|x-5|}$$

Looking at Table 4-3, it seems that as x approaches 5, the results rapidly accelerate as they diverge to a certain very big number referred to as infinity (∞). Take a look at Table 4-4:

$$f(x) = \frac{1}{|x-5|}$$

*Table 4-3. Finding x*

| f(x) | x |
|---|---|
| 4.99997 | 33333.33 |
| 4.99998 | 50000 |
| 4.99999 | 100000 |
| 4.9999999 | 10000000 |
| 5 | Undefined |
| 5.0000001 | 10000000 |
| 5.00001 | 100000 |
| 5.00002 | 50000 |
| 5.00003 | 33333.33 |

See how at every tiny step *x* approaches 5, *y* approaches positive infinity. The answer to the limit question is therefore, positive infinity (+∞). Figure 4-14 shows the graph of the function. Notice how they both rise in value as x approaches 5.

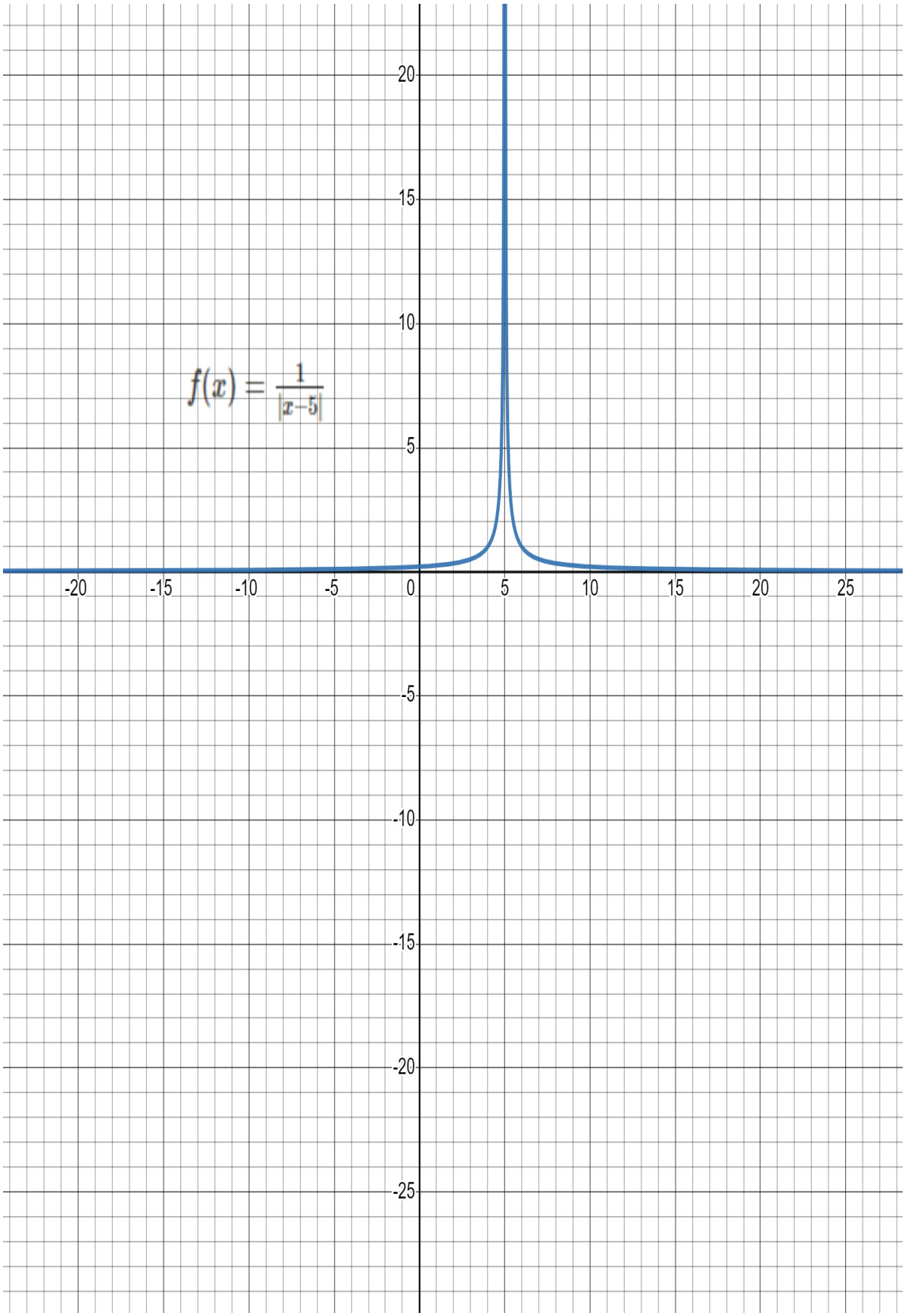
$f(x) = \frac{1}{|x-5|}$
20
15
10
5
-5
-10
-15
-20
-25
-20
-15
-10
-5
0
5
10
15
20
25

*Figure 4-14. The graph of the function proving that the limit exists as x approaches 5*

### TO INFINITY AND BEYOND

In this optional note, you can understand what infinity represents in terms of mathematics. *Infinity* is an idea or a concept rather than a number. The symbol (∞) is often referred to as a *lemniscate*.

Positive infinity (∞) and negative infinity (-∞) are both concepts that exist across the axis where the former tends towards the left and the latter tends towards the right.

Interestingly, infinity is often thought of as an ever-growing measure, but it is not expanding or getting bigger. It is already what it is.

Mathematical operations including the concept of infinity may be hard to grasp and have many considerations. One of the most interesting examples is the answer to why the result of 1 divided by zero is undefined.

Imagine dividing an apple by ten people; normally every person will get an equal fraction of that apple until the apple is consumed. But what if you want to divide the apple by zero people? The number of people required to consume the apple will tend towards infinity (the logic is hard to grasp, but mathematically you can think of infinitesimally small parts of the apple requiring a huge number of people to consume them).

Therefore, following this logic, you can say that 1 divided by 0 is infinity and is actually defined. So, why is it generally considered as undefined? The issue is that the infinity described in the apples' example is the positive one, and if you want to follow the example of dividing 1 by infinitesimally small negative numbers that tend towards zero, then you will also say that 1 divided by 0 is negative infinity.

So which is it? Positive or negative infinity? Because of this conflict, the result is undefined.

*Continuous* functions are ones that are drawn without gaps or holes in the graph, while *discontinuous* functions contain such gaps and holes. This usually means that the latter contain points where the solution of the functions is undefined and

may need to be approximated by limits. Therefore, continuity and limits are two related concepts.

Let's proceed to solving limits; after all, you are not going to create a table every time and analyze the results subjectively to find the limits. There are three ways to solve limits:

- *Substitution*: This is the simplest rule and is generally used first.
- *Factoring*: This comes after substitution does not work.
- *Conjugate methods*: This solution comes after the first two ways do not work.

The *substitution* way is simply plugging in the value which *x* approaches. Basically, these are functions that have solutions where the limits are used. Take the following example:

$\lim_{x\to 5} x + 10 - 2x$

Using the substitution way, the limit of the function is found as follows:

$$\lim_{x\to 5} x + 10 - 2x = 5 + 10 - (2 \times 5) = 5$$

Therefore, the answer to the limit is 5.

The *factoring* way is the next option when substitution does not work (for example, the limit is undefined after plugging in the value of *x* in the function). *Factoring* is all about changing the form of the equation using factors in a way that it is not undefined anymore when using the substitution way. Take the following example:

$\lim_{x\to -6} \frac{(x+6)(x^2-x+1)}{x+6}$

If you try the substitution way, you will get an undefined value as follows:

$$\lim_{x\to -6} \frac{(x+6)\left(x^2-x+1\right)}{x+6} = \frac{(-6+6)\left((-6)^2-(-6)+1\right)}{-6+6} = \frac{0}{0}$$
$$= Undefined$$

Factoring may help in this case. For example, the nominator is multiplied by

(x+6) and then divided by (x+6). Simplifying this by canceling the two terms could give a solution:

$$\lim_{x\to -6} \frac{(x+6)(x^2-x+1)}{x+6} = \lim_{x\to -6} x^2 - x + 1$$

Now that factoring is done, you can try substitution once again:

$$\lim_{x\to -6} x^2 - x + 1 = (-6)^2 - (-6) + 1 = 43$$

The limit of the function as *x* tends towards -6 is therefore 43.

The *conjugate* way is the next option in case substitution and factoring do not work. A conjugate is simply the changing of signs between two variables. For example, the conjugate of *x* + *y* is *x* - *y*. The way to do this in the case of a fraction is to multiply the nominator and the denominator by the conjugate of one of them (with a preference to use the conjugate of the term that has a square root since it will get canceled out). Take the following example:

$$\lim_{x\to 9} \frac{x-9}{\sqrt{x}-3}$$

By multiplying both terms by the conjugate of the denominator, you will have started to use the conjugate way to solve the problem:

$$\lim_{x\to 9} \frac{x-9}{\sqrt{x}-3}\left(\frac{\sqrt{x}+3}{\sqrt{x}+3}\right)$$

Taking into account the multiplication and simplifying gives the following:

$$\lim_{x\to 9} \frac{(x-9)(\sqrt{x}+3)}{(\sqrt{x}-3)(\sqrt{x}+3)}$$

You will be left with the following familiar situation:

$$\lim_{x\to 9} \frac{(x-9)(\sqrt{x}+3)}{x-9}$$

$$\lim_{x\to 9} \sqrt{x} + 3$$

Now, the function is ready for substitution:

$$\lim_{x\to 9} \sqrt{9} + 3 = 3 + 3 = 6$$

The solution to the function is therefore 6. As you can see, sometimes work needs to be done on preparing the equations in order to be ready for substitution.

**NOTE**

The main key points of this section on limits are as follows:

- Limits help find solutions for functions that may be undefined in certain points.
- For the general limit to exist, the two one-sided limits must exist and must be equal.
- There are ways to find the limit of a function, notably substitution, factoring, and the conjugate way.
- Limits are useful in machine learning such as sensitivity analysis and optimization.

## Derivatives

A *derivative* measures the change in a function given a change of one or more of its inputs. In other words, it is the rate of change of a function at a given point.

Having a solid understanding of derivatives is important in building machine learning models for multiple reasons:

*Optimization*

In order to minimize the loss function (a concept you will see in Chapter 8), optimization methods employ derivatives to ascertain the direction of the steepest descent and modify model parameters. Gradient descent is one of the most frequently used optimization techniques in machine learning.

*Backpropagation*

To execute gradient descent in deep learning, the backpropagation technique uses derivatives to calculate the gradients of the loss function with respect to the model's parameters.

*Hyperparameter tuning*

To improve the performance of the model, derivatives are used for sensitivity

analysis and tuning of hyperparameters (another concept you will perfectly grasp in Chapter 8).

Do not forget what you have learned from the previous section on limits, as you will be needing them for this section as well. Calculus mainly deals with derivatives and integrals. This section discusses derivatives and their uses.

You can consider derivatives as functions that represent (or model) the slope of another function at some point. A *slope* is a measure of a line's position relative to a horizontal line. A positive slope indicates a line moving up, while a negative slope indicates a line moving down.

Derivatives and slopes are related concepts, but they are not the same thing. Here's the main difference between the two:

- The slope measures the steepness of a line. It is the ratio of the change in the y-axis to the change in the *x*-axis. You have already seen this in the section that discusses linear algebra.
- The derivative describes the rate of change of a given function. As the distance between two points on a function approaches zero, the derivative of that function at that point is the limit of the slope of the tangent line.

Before explaining derivatives in layperson's terms and seeing some examples, let's see their formal definitions (which means their mathematical representation in their default form):

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

The equation forms the basis of solving derivatives, although there are many shortcuts that you will learn and understand where they come from. Let's try finding the derivative of a function using the formal definition. Consider the following equation:

$$f(x) = x^2 + 4x - 2$$

To find the derivative, plug *f(x)* inside the formal definition and then solve the limit:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

To simplify things, let's find *f(x + h)* so that plugging it in the formal definition becomes easier:

$$f(x+h) = (x+h)^2 + 4(x+h) - 2$$

$$f(x+h) = x^2 + 2xh + h^2 + 4x + 4h - 2$$

Now, let's plug f(x + h) into the definition:

$$f'(x) = \lim_{h \to 0} \frac{x^2 + 2xh + h^2 + 4x + 4h - 2 - x^2 - 4x + 2}{h}$$

Notice how there are many terms that can be simplified so that the formula becomes clearer. Remember, you are trying to find the limit for the moment, and the derivative is found after solving the limit:

$$f'(x) = \lim_{h \to 0} \frac{2xh + h^2 + 4h}{h}$$

The division by *h* gives further potential for simplification since you can divide all the terms in the numerator by the denominator *h*:

$$f'(x) = \lim_{h \to 0} 2x + h + 4$$

It's now time to solve the limit. Because the equation is simple, the first attempt is by substitution, which is, as you have guessed, possible. By substituting the variable *h* and making it zero (according to the limit), you are left with the following:

$$f'(x) = 2x + 4$$

That is the derivative of the original function *f(x)*. If you want to find the derivative of the function when *x* = 2, you simply have to plug it in to the derivative function:

$$f'(2) = 2(2) + 4 = 8$$

Take a look at the graph of the function that you have just solved. Figure 4-15 shows the original function's graph with the derivative (the straight line). Notice how *f'(2)* lies exactly at 8. The slope of *f(x)* when *x* = 2 is 8.

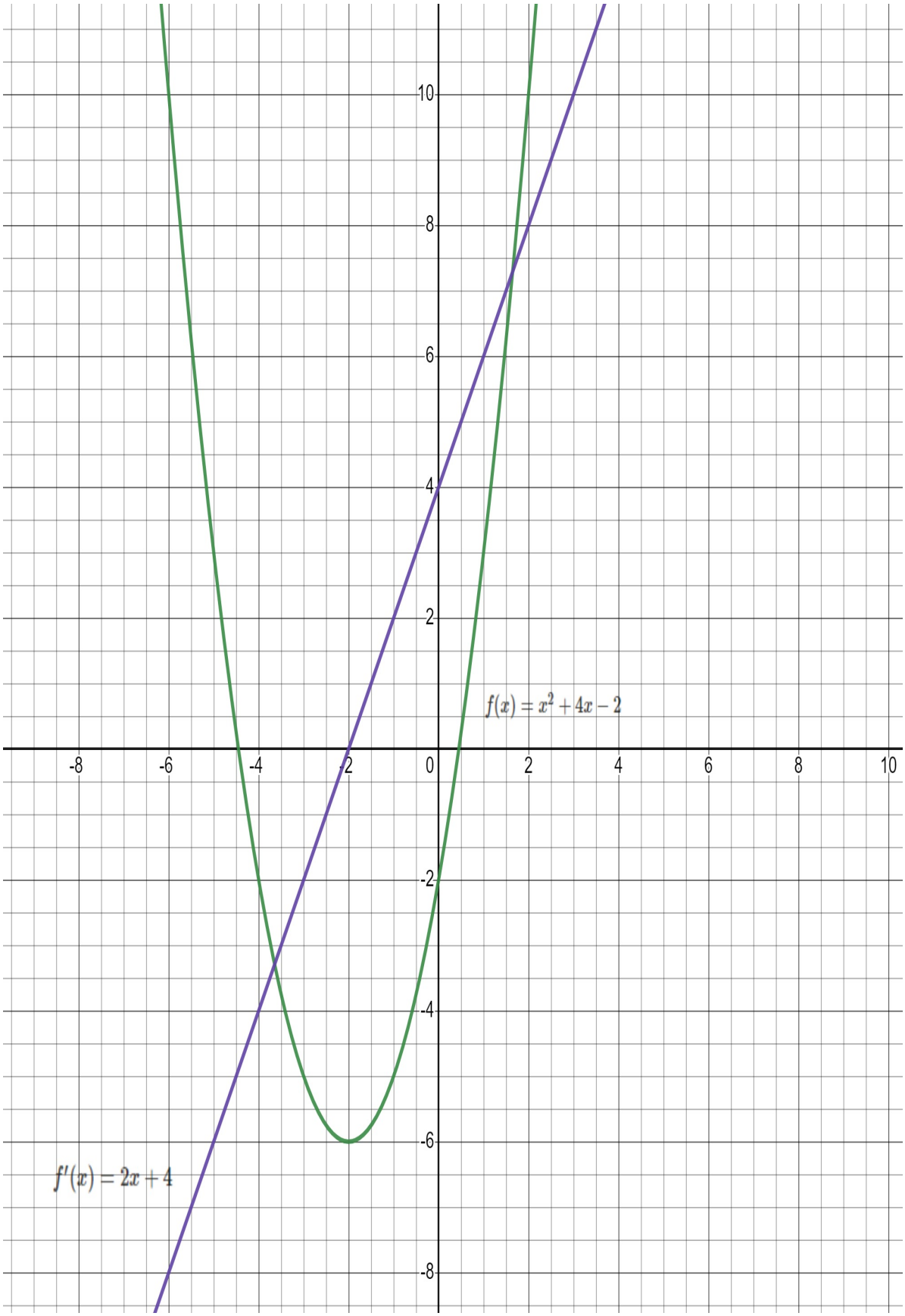
10
8
6
4
2
-2
-4
-6
-8
-8
-6
-4
-2
0
2
4
6
8
10
$f(x) = x^2 + 4x - 2$
$f'(x) = 2x + 4$

*Figure 4-15. The original f(x) with its derivative f'(x)*

> **NOTE**
>
> Notice that when *f(x)* hits the bottom and starts rising, *f'(x)* crossed the zero line at -2. This is a concept you will later in this chapter.

You are unlikely to use the formal definition every time you want to find a derivative (which can be used on every function). There are derivative rules that allow you to save a lot of time through shortcuts. The first rule is referred to as the *power rule*, which is a way to find the derivative of functions with exponents.

It is common to also refer to derivatives using this notation (which is the same thing as *f'(x)*):

$$\frac{dy}{dx}$$

The power rule for finding derivatives is as follows:

$$\frac{dy}{dx}(ax^n) = (a.n)\,x^{n-1}$$

Basically, what this means is that the derivative is found by multiplying the constant by the exponent and then subtracting 1 from the exponent. Here's an example:

$$f(x) = x^4$$

$$f'(x) = (1 \times 4)\,x^{(4-1)} = 4x^3$$

Remember that if there is no constant attached to the variable, it means that the constant is equal to 1. Here's a more complex example with the same principle:

$$f(x) = 2x^2 + 3x^7 - 2x^3$$

$$f'(x) = 4x + 21x^6 - 6x^2$$

It is worth noting that the rule also applies to constants even though they do not satisfy the general form of the power rule. The derivative of a constant is zero. However, it helps to know why, but first, you must be aware of this

mathematical concept:

$$x^0 = 1$$

With this being said, you can imagine constants as always being multiplied by *x* to the power of zero (since it does not change their value). Now, if you want to find the derivative of 17, here's how it would go:

$$17 = 17x^0 = (0 \times 17)\, x^{0-1} = 0x^{-1} = 0$$

As you know, anything multiplied by zero returns zero as a result. This gives the constants rule for derivatives as follows:

$$\frac{dy}{dx}(a) = 0$$

You follow the same logic when encountering fractions or negative numbers in the exponents.

The *product rule* of derivatives is useful when there are two functions multiplied by each other. The product rule is as follows:

$$\frac{dy}{dx}\left[f\left(x\right)g\left(x\right)\right] = f'\left(x\right)g\left(x\right) + f\left(x\right)g'\left(x\right)$$

Let's take an example and find the derivative using the product rule:

$$h\left(x\right) = \left(x^2 + 2\right)\left(x^3 + 1\right)$$

The equation can clearly by segmented into two terms, *f(x)* and *g(x)*, like this:

$$f\left(x\right) = \left(x^2 + 2\right)$$

$$g\left(x\right) = \left(x^3 + 1\right)$$

Let's find the derivatives of the two terms before applying the product rule. Notice that finding the derivative of *f(x)* and *g(x)* is easy once you have understood the power rule:

$$f'\left(x\right) = 2x$$

$$g'\left(x\right) = 3x^2$$

When applying the product rule, you should get the following:

$$h'\left(x\right) = \left(x^2 + 2\right)\left(3x^2\right) + \left(2x\right)\left(x^3 + 1\right)$$

$$h'(x) = 3x^4 + 6x^2 + 2x^4 + 2x$$

$$h'(x) = 5x^4 + 6x^2 + 2x$$

Figure 4-16 shows the graph of *h(x)* and *h'(x)*.

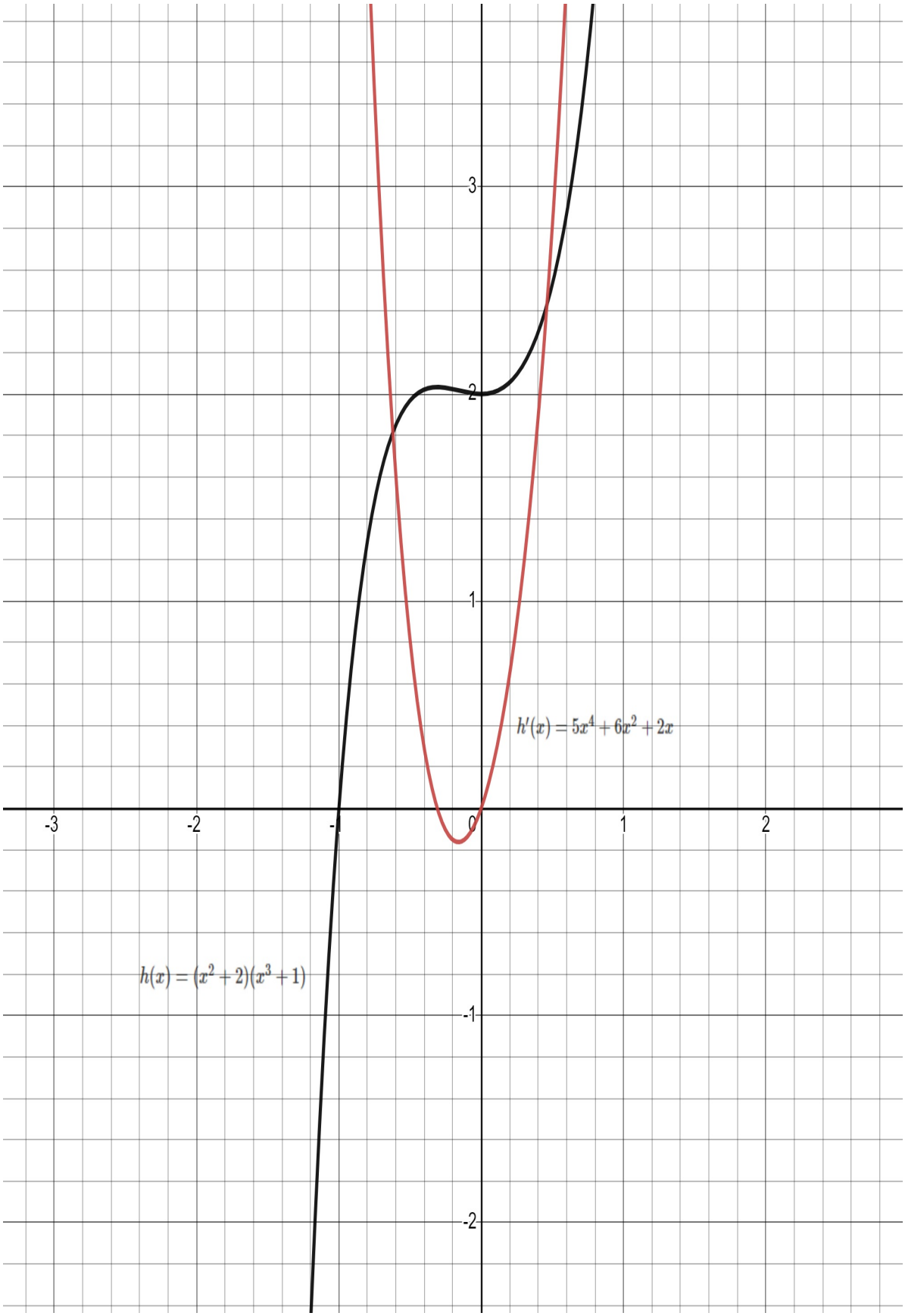
3
2
1
-1
-2
-3
-2
-1
0
1
2
$h'(x) = 5x^4 + 6x^2 + 2x$
$h(x) = (x^2 + 2)(x^3 + 1)$

*Figure 4-16. The original h(x) with its derivative h'(x)*

The next step is to see the *quotient rule*, which deals with the division of two functions. The formal definition is as follows:

$$\frac{dy}{dx}\left[\frac{f(x)}{g(x)}\right] = \frac{f'(x)g(x) - f(x)g'(x)}{[g(x)]^2}$$

Let's apply it into the following function:

$$f(x) = \frac{x^2 - x + 1}{x^2 + 1}$$

As usual, it's better to start by finding the derivatives of *f(x)* and *g(x)* which in this case are clearly separated, with *f(x)* being the nominator and *g(x)* being the denominator. When applying the quotient rule, you should get the following:

$$f'(x) = \frac{(2x-1)(x^2+1) - (x^2 - x + 1)(2x)}{(x^2+1)^2}$$

$$f'(x) = \frac{2x^3 + 2x - x^2 - 1 - 2x^3 + 2x^2 - 2x}{(x^2+1)^2}$$

$$f'(x) = \frac{x^2 - 1}{(x^2+1)^2}$$

*Exponential derivatives* deal with power rule applied to constants. Take a look at the following equation -- how would you find its derivative?

$$f(x) = a^x$$

Instead of the usual variable-base-constant-exponent, it is constant-base-variable-exponent. This is treated differently when trying to calculate the derivative. The formal definition is as follow:

$$\frac{dy}{dx}a^x = a^x\ (\ \ln\ a)$$

The following example shows how this is done:

$$\frac{dy}{dx}4^x = 4^x\ (\ \ln\ 4)$$

Euler's number, mentioned earlier, has a special derivative. When it comes to finding the derivative of *e*, the answer is interesting:

$$\frac{dy}{dx}e^x = e^x\ (\ \ln\ e) = e^x$$

This is because the natural log function and the exponential function are inverses of one another, so, the term *ln e* equals to 1. Therefore, the derivative of the exponential function function *e* is itself.

In parallel, let's discuss logarithmic derivatives. By now, you should have known what exponents and logarithms are. The general definition for both types of logarithms is as follows:

$$\frac{dy}{dx}\ \log_a x = \frac{1}{x \ln a}$$

$$\frac{dy}{dx}\ \ln\ x = \log_e x = \frac{1}{x \ln e} = \frac{1}{x}$$

Notice how in the second derivative function of the natural logarithm, the term *ln e* is once again encountered, thus making simplification quite easy since it is equal to 1.

Take the following example:

$$f\left(x\right) = 7log_2\left(x\right)$$

Using the formal definition, the derivative of this logarithmic function is as follows:

$$f'\left(x\right) = 7\left(\frac{1}{x \ln 2}\right) = \frac{7}{x \ln 2}$$

**NOTE**

Remember that the logarithm *log* has a base of 10, but the natural logarithm *ln* has a base of *e* (~2.7182)

The natural logarithm and the log function are actually linearly related through simple multiplication. If you know the log of the constant *a*, you can find its natural logarithm *ln* by multiplying the log of *a* by 2.303.

One major concept in derivatives is the *chain rule*. Let's back up to the power rule, which deals with exponents on variables. Remember the following formula to find the derivative:

$$\frac{dy}{dx}\left(ax^n\right) = \left(a.n\right)x^{n-1}$$

This is a simplified version because there is only *x*, but the reality is that you must multiply by the derivative of the term under the exponent. Until now, you

have seen only *x* as the variable under the exponent. The derivative of *x* is 1, which is why it is simplified and rendered invisible. However, with more complex functions such as this one:

$$f(x) = (4x + 1)^2$$

The derivative of the function is found by following these two steps:

1. Find the derivative of the outside function without touching the inside function.
2. Find the derivative of the inside function and multiply it by the rest of the function.

The solution is therefore as follows (knowing that the derivative of *4x + 1* is just 4):

$$f'(x) = 2(4x + 1).4$$

$$f'(x) = 8(4x + 1)$$

$$f'(x) = 32x + 8$$

The same applies with the exponential functions. Take the following example:

$$f(x) = e^x$$

$$f'(x) = e^x(1) = e^x$$

The chain rule can actually be considered as a master rule as it applies anywhere even in the product rule and the quotient rule.

There are more concepts to master in derivatives, but as this book is not meant to be a full calculus masterclass, you should at least know the meaning of a derivative, how it is found, what does it represent, and how can it be used in machine and deep learning.

**NOTE**

The key points of this section on derivatives are as follow:

- A derivative measures the change in a function given a change of one or more of its inputs.
- The power rule is used to find the derivative of a function raised to a power.

- The product rule is used to find the derivative of two functions that are multiplied together.
- The quotient rule is used to find the derivative of two functions that are divided by each other.
- The chain rule is the master rule used in differentiating (which means the process of finding the derivative). Due to simplicity, it is often overlooked.
- Derivatives play a crucial role in machine learning such as enabling optimization techniques, aiding model training, and enhancing the interpretability of the models.

## Integrals and the Fundamental Theorem of Calculus

An *integral* is an operation that represents the area under a curve of a function given an interval. It is the inverse of a derivative, which is why it is also called an *anti-derivative*.

The process of finding integrals is called *integration*. Integrals can be used to find areas below a curve and they are also heavily used in the world of finance such as risk management, portfolio management, probabilistic methods, and even option pricing.

The most basic way of understanding an integral is by thinking of calculating an area below the curve of a function. This can also be done by manually, calculating different changes in the *x*-axis, but it won't be accurate (accuracy increases as you add up much smaller slices). Therefore, as the size of the slices approaches zero, the accuracy of the area gets better. Since this is a tedious process, integrals are here for the rescue.

Keep in mind that an integral is the inverse of a derivative. This is important because it implies a direct relationship between the two. The basic definition of an integral is as follows:

$$\int f(x)\,dx = F(X) + C$$

$$The\ \int\ symbol\ represents\ the\ integration\ process$$

$$f(x)\ is\ the\ derivative\ of\ the\ general\ function\ F(x)$$

$$C\ represents\ the\ lost\ constant\ in\ the\ differentiation\ process$$

$$dx\ represents\ slicing\ along\ x\ as\ it\ approaches\ zero$$

What the preceding equation means is that the integral of *f(x)* is the general

function *F(x)* plus a constant *C* which was lost initially in the initial differentiation process. Here's an example to better explain the need to put in the constant:

Consider the following function:

$$f\left(x\right) = x^2 + 5$$

Calculating its derivative, you get the following result:

$$f'\left(x\right) = 2x$$

Now, what if you wanted to integrate it so that you go back to the original function (which in this case is represented by the capital letter *F(x)* instead of *f(x)*)?

$$\int 2x\, dx$$

Normally, having seen the differentiation process (which means taking the derivative), you would return 2 as the exponent, which gives you the following answer:

$$\int 2x\, dx = x^2$$

This does not look like the original function. It's missing the constant 5. But you have no way of knowing that or even if you knew there was a constant, what is it? 1? 2? 677? This is why a constant *C* is added in the integration process so that it represents the lost constant. Therefore, the answer to the integration problem is as follows:

$$\int 2x\, dx = x^2 + C$$

**NOTE**

Up until now, the discussion has been limited to *indefinite integrals* where the integration symbol is *naked* (which means there are no boundaries to it). You will see what this means right after defining the necessary rules to complete the integration.

For the power function (just like the previous function), the general rule for integration is as follows:

$$\int x^a \, dx = \frac{x^{a+1}}{a+1} + C$$

This is much simpler than it looks. You are simply reversing the power rule you saw earlier. Consider the following example:

$$\int 2x^6 \, dx$$

$$\int 2x^6 \, dx = \frac{2x^7}{7} + C$$

$$\int 2x^6 \, dx = \frac{2}{7}x^7 + C$$

To verify your answer, you can find the derivative of the result (using the power rule):

$$F\left(x\right) = \frac{2}{7}x^7 + C$$

$$f'\left(x\right) = \left(7\right)\frac{2}{7}x^{7-1} + 0$$

$$f'\left(x\right) = 2x^6$$

Let's take another example. Consider the following integration problem:

$$\int 2 \, dx$$

Naturally, using the rule, you should find the following result:

$$\int 2 \, dx = 2x + C$$

Let's move on to *definite integrals,* which are integrals with numbers on top and bottom that represent intervals below a curve of a function. Hence, *indefinite* integrals find the area under the curve everywhere but definite integrals are bounded within an interval given by point *a* and point *b*. The general definition of indefinite integrals is as follows:

$$\int_a^b f\left(x\right) \, dx = F\left(B\right) - F\left(A\right)$$

This is as simple as it gets. You will solve the integral, then plug in the two numbers and subtract the two functions from each other. Consider the following evaluation of an integral (integral solving is commonly referred to as *evaluating* the integral):

$$\int_0^6 3x^2 - 10x + 4 \, dx$$

The first step is to understand what is being asked. From the definition of integrals, it seems that the area between [0, 2] on the *x*-axis is to be calculated using the given function:

$$F(x) = \left(\left[x^3 - 5x^2 + 4x + C\right]\right)\Big|_0^6$$

To evaluate the integral at the given points, simply plug in the values as follows:

$$F(x) = \left(\left[6^3 - 5(6)^2 + 4(6) + C\right]\right) - \left(\left[0^3 - 5(0)^2 + 4(0) + C\right]\right)$$

$$F(x) = ([216 - 180 + 24 + C]) - ([0 - 0 + 0 + C])$$

$$F(x) = ([60 + C]) - ([0 + C])$$

$$F(x) = (60 - 0)$$

$$F(x) = 60$$

> **NOTE**
>
> The constant *C* will always cancel out in definite integrals so you can leave it out in this kind of problems.

Therefore, the area below the graph of *f(x)* and above the *x*-axis, as well as between [0, 6] on the *x*-axis, is equal to 60 square units. The following shows a few rules of thumb on integrals (after all, this chapter is supposed to refresh your knowledge or to give you a basic understanding of a few key mathematical concepts):

- To find the integral of a constant:

$$\int a\,dx = ax + C$$

- To find the integral of a variable:

$$\int x\,dx = \tfrac{1}{2}x^2 + C$$

- To find the integral of a reciprocal:

$$\int \tfrac{1}{x}\,dx = \ln|x| + C$$

- To find the integral of an exponential:

$$\int a^x \, dx = \frac{a^x}{\ln(a)} + C$$

$$\int e^x \, dx = e^x + C$$

The *fundamental theorem of calculus* links derivatives with integrals. This means that it defines derivatives in terms of integrals and vice versa. The fundamental theorem of calculus is actually made up of two parts:

*Part I*

> The first part of the fundamental theorem of calculus states that if you have a continuous function *f(x)*, then the original function *F(x)* defined as the antiderivative of *f(x)* from a fixed starting point *a* up to *x*, is a function that is differentiable everywhere from *a* to *x*, and its derivative is simply *f(x)* evaluated at *x*.

*Part II*

> The second part of the fundamental theorem of calculus states that if you have a function *f(x)* that is continuous over a certain interval [*a*, *b*], and you define a new function *F(x)* as the integral of *f(x)* from *a* to *x*, then the definite integral of *f(x)* over that same interval [*a*, *b*] can be calculated as *F(b) - F(a)*.

The theorem is useful in many fields including physics and engineering, but optimization and other mathematical models also benefit from it. Some examples of using integrals in the different learning algorithms can be summed up as follows:

*Density estimation*

> Integrals are used in *density estimation*, a part of many machine learning algorithms, to calculate the probability density function.

*Reinforcement learning*

Integrals are used in reinforcement learning to calculate expected values of reward functions. Reinforcement learning is covered in Chapter 10.

*Bayesian models*

Integrals are used in *Bayesian inference*, a statistical framework for modeling uncertainty.

**NOTE**

The key points of this section on integrals are as follow:

- Integrals are also known as antiderivatives and they are the opposite of derivatives.
- Indefinite integrals find the area under the curve everywhere while definite integrals are bounded within an interval given by point *a* and point *b*.
- The fundamental theorem of calculus is the bridge between derivatives and integrals.
- Integrals are used in machine learning for modeling uncertainty, making predictions, and estimating expected values.

## Optimization

Several machine and deep learning algorithms depend on optimization techniques to decrease error functions. This section discusses a primordial concept in the different learning algorithms.

*Optimization* is the process of finding the best solution among the possible solutions' universe. Optimization is all about finding the highest and lowest points of a function. Figure 4-17 shows the graph for the following formula:

$$f(x) = x^4 - 2x^2 + x$$

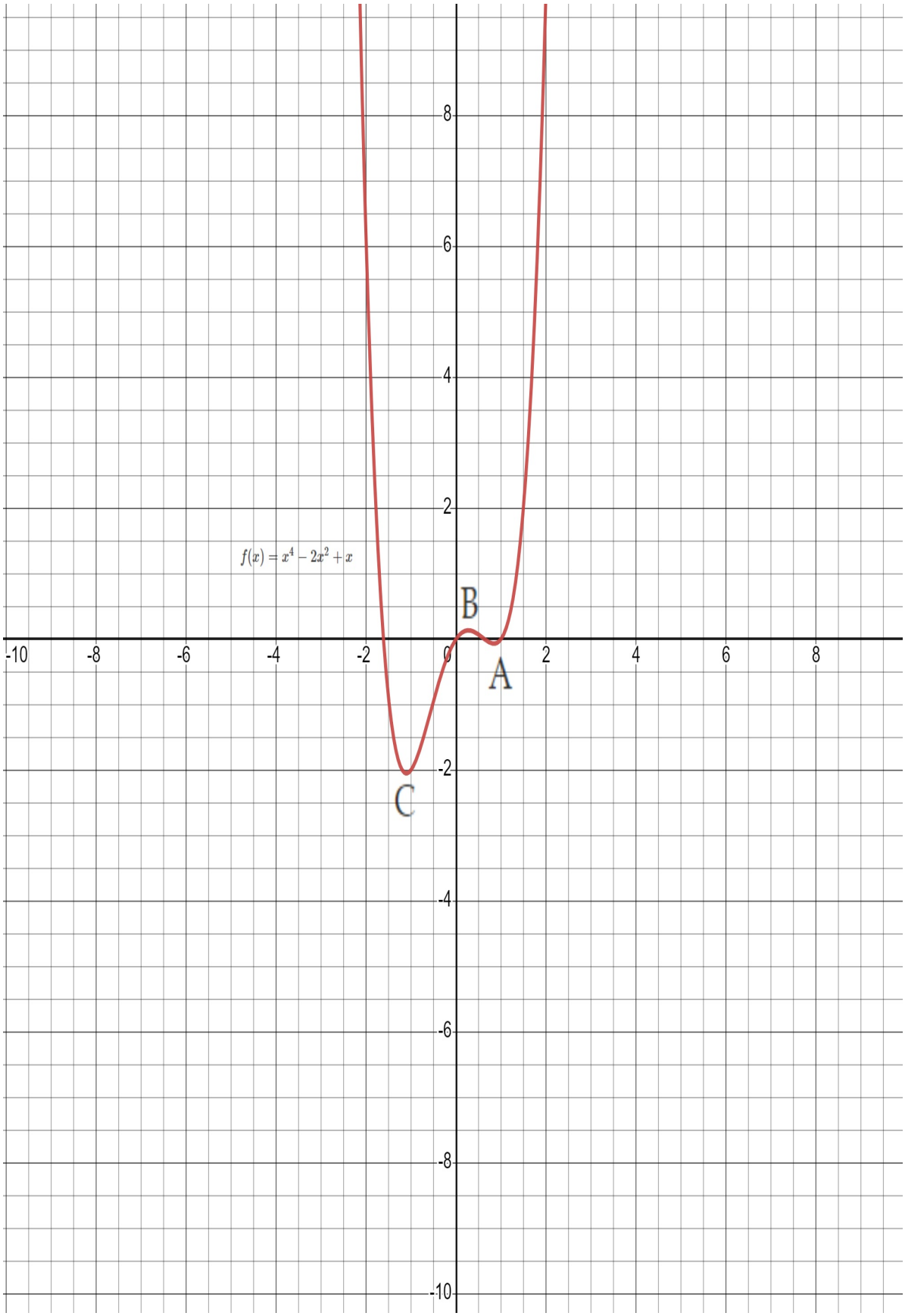
$f(x) = x^4 - 2x^2 + x$
A
B
C
-10
-8
-6
-4
-2
0
2
4
6
8
-10
-8
-6
-4
-2
2
4
6
8

*Figure 4-17. The graph of the function*

A *local minimum* exists when values on the right of the *x*-axis are decreasing until reaching a point where they start increasing. The point does not have to necessarily be the lowest point in the function, hence the name *local*. In Figure 4-18, the function has a local minimum at point A.

A *local maximum* exists when values on the right of the x-axis are increasing until reaching a point where they start decreasing. The point does not have to necessarily be the highest point in the function. In Figure 4-18, the function has a local maximum at point B.

A *global minimum* exists when values on the right of the x-axis are decreasing until reaching a point where they start increasing. The point must be the lowest point in the function hence the name global. In Figure 4-18, the function has a global minimum at point C.

A *global maximum* exists when values on the right of the x-axis are increasing until reaching a point where they start decreasing without. The point must be the highest point in the function. In Figure 4-18, there is no global maximum, as the function will continue infinitely without shaping a top. You can clearly see how the function accelerates upwards.

When dealing with machine and deep learning models, the aim is to find model parameters (or inputs) that minimize what is known as a *loss function* (a function that gives the error of forecasts). If the loss function is convex, optimization techniques should find the parameters that tend towards the global minimum where the loss function is minimized. Furthermore, if the loss function is non-convex, the convergence is not guaranteed, and the optimization may only lead towards approaching a local minimum, which is a part of the aim, but this leaves the global minimum which is the final aim.

But how are these minima and maxima found? Let’s look at it step by step:

1. The first step is to perform the first derivative test (which is simply calculating the derivative of the function). Then, setting the function equal to zero and solving for *x* will give what is known as critical points. *Critical points* are the points where the function changes direction (the values stop going in one direction and start going in another). Therefore, these points

are maxima and minima.

2. The second step is to perform the second derivative test (which is simply calculating the derivative of the derivative). Then, setting the function equal to zero and solving for *x* will give what is known as inflection points. *Inflection points* give where the function is concave up and where it is concave down.

In other words, critical points are where the function changes direction and inflection points are where the function changes its concavity. Figure 4-19 shows the difference between a concave up function and a concave down function.

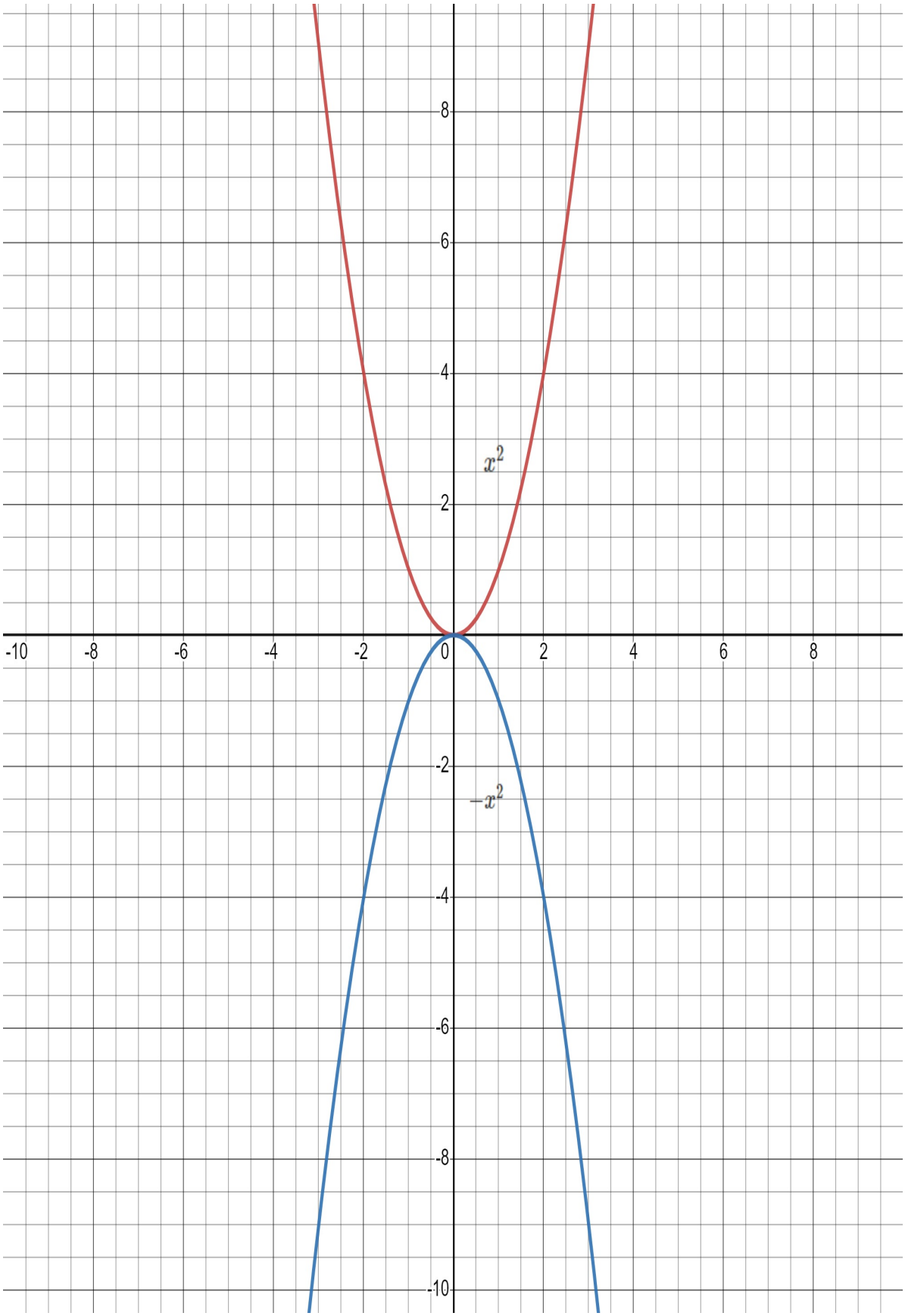
8
6
4
$x^2$
2
-10
-8
-6
-4
-2
0
2
4
6
8
-2
$-x^2$
-4
-6
-8
-10

*Figure 4-18. A concave up versus a concave down functions*

$$Concave\ up\ function = x^2$$

$$Concave\ down\ function = -x^2$$

The steps to find the extrema are as follows:

1. Find the first derivative and set it to zero.
2. Solve the first derivative to find *x*. The values are called critical points and they represent the points where the function changes the direction.
3. Plug in values in the formula that are either below or above the critical points. If the result of the first derivative is positive it means that it's increasing around that point and if it's negative, then it means that it's decreasing around that point.
4. Find the second derivative and set it to zero.
5. Solve the second derivative to find *x*. The values, called inflection points, represent the points where concavity changes from up to down and vice versa.
6. Plug in values in the formula that are either below or above the inflection points. If the result of the second derivative is positive, it means there is a minimum at that point, and if it's negative it means there is a maximum at that point.

It is important to understand that the first derivative and second derivative tests relate to critical points as opposed to the second derivative test relating to inflection points. The following example finds the extrema of the function:

$$f(x) = x^2 + x + 4$$

The first step is to take the first derivative, set it to zero, and solve for *x*:

$$f'(x) = 2x + 1$$

$$2x + 1 = 0$$

$$x = -\frac{1}{2}$$

Therefore, there is a critical point at that value. Now, the next step is to find the

second derivative:

$$f''(x) = 2$$

Next, the critical point must be plugged into the second derivative formula:

$$f''\left(-\frac{1}{2}\right) = 2$$

The second derivative is positive at the critical point. This means that there is a local minimum at that point.

In the next chapters, you will see more complex optimization techniques such as the gradient descent and the stochastic gradient descent, which are fairly common in machine learning algorithms.

**NOTE**

The key points of this section on optimization are as follow:

- Optimization is the process of finding the function's extrema
- Critical points are the points where the function changes direction (the values stop going in one direction and start going in another)
- Inflection points give where the function is concave up and where it is concave down.
- A loss function is a function that measures the error of forecasts in predictive machine learning. It needs to be minimized in order to increase the accuracy of the model. Optimization of the loss function can be done through the discussed ways or through what is known as a gradient, a technique out of scope of the book.

## Summary

Chapters 2, 3, and 4 have presented the main numerical concepts that you need to start understanding basic machine and deep learning models. I have made all reasonable efforts to simplify as much as possible the technical requirements, I do encourage you to read these three chapters at least twice so that everything you have learned becomes a second nature.

Naturally, such a complex field requires more in-depth knowledge in mathematics, but I believe that with the concepts seen in this chapter, you may start discovering and building the models in Python. After all, they come pre-

built from packages and libraries, and the aim of this chapter is to understand what you are working with It is unlikely that you will build the models from scratch using archaic tools.

By now, you should have gained a certain understanding of data science and the mathematical requirements that will get you started comfortably.

---

[1] Matrices can also contain symbols and expressions but for the sake of simplicity, let's stick to numbers.

# Chapter 5. Introducing Technical Analysis

> **A NOTE FOR EARLY RELEASE READERS**
>
> With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.
>
> This will be the 5th chapter of the final book. Please note that the GitHub repo will be made active later on.
>
> If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at *ccollins@oreilly.com*.

Technical analysis presents many types of inputs that you can use in your deep learning models. This chapter introduces this vast field so that you are equipped with the necessary knowledge to create technical-based learning models in the chapters to follow.

*Technical analysis* relies on the visual interpretation of the price action's history to determine the likely aggregate direction of the market. It relies on the idea that the past is the best predictor of the future. There are several types of techniques within the vast field that is technical analysis, notably the following:

*Charting analysis*

> This is where you apply subjective visual interpretation techniques onto charts. You generally use methods like drawing support and resistance lines as well as retracements to find inflection levels that aim to determine the next move.

*Indicator analysis*

This is where you use mathematical formulas to create objective indicators that can be either trend following or contrarian. Among known indicators are *moving averages* and the *relative strength index* (RSI), both of which are discussed in greater detail in this chapter.

*Pattern recognition*

This is where you monitor certain recurring configurations and act on them. A *pattern* is generally an event that emerges from time to time and presents a certain theoretical or empirical outcome. In finance, it is more complicated, but certain patterns have been shown to add value across time, and this may partly be due to a phenomenon called *self-fulfilling prophecy* (a process by which an initial expectation leads to its confirmation).

Let's take a quick tour of the history of technical analysis so that you have a better idea of what to expect. Technical analysis relies on three principles:

*History repeats itself.*

You are likely to see clusters during trends and ranges. Also, certain configurations are likely to have a similar outcome most of the time.[1]

*The market discounts everything.*

It is assumed that everything (all fundamental, technical, and quantitative information) is included in the current price.

*The market moves in waves.*

Due to different time frames and needs, traders buy and sell at different frequencies, therefore creating trends and waves as opposed to a straight line.

Unfortunately, technical analysis is overhyped and misused by the retail trading community, which gives it a somewhat less than savory reputation in the

professional industry. Every type of analysis has its strengths and weaknesses, and there are successful fundamental, technical, and quantitative investors, but there are also failed investors from the three fields.

*Fundamental analysis* relies on economic and financial data to deliver a judgment on a specific security or currency with a long-term investment horizon, whereas *quantitative analysis* is more versatile and is more often applied to short-term data. It uses mathematical and statistical concepts to deliver a forecast.

Among other assumptions, technical analysis suggests that markets are not efficient, but what does that mean? *Market efficiency* states that information is already embedded in the current price and that price and value are the same thing. When you buy an asset, you are hoping that it is *undervalued* (in fundamental analysis jargon) or *oversold* (in technical analysis jargon), which is why you believe the price should go up to meet the value. Therefore, you are assuming that the value is greater than the price.

Market efficiency rebuffs any claims that the price does not equal the value and therefore suggests that any alpha trading must not result in above-average returns (*alpha trading* is the act of engaging in speculative operations to perform better than a benchmark, which is typically an index or a weighted measure).

The market efficiency hypothesis is the technical analyst’s greatest enemy, as one of its principles is that in the weak form of efficiency, you cannot earn excess returns from technical analysis. Hence, technical analysis gets shot down right at the beginning, and then fundamental analysis gets its share of the beating.

It is fair to assume that at some point in the future, markets will have no choice but to be efficient due to the number of participants and the ease of access to information. However, as political and abnormal events show us, markets tend to be anything but efficient.

**NOTE**

An example of a political event that triggers panic and irrationality in the markets is the Russian invasion of Ukraine. Similarly, an example of an abnormal economic event is an unexpected interest rate hike from a central bank.

## Charting Analysis

Before you can understand what charting analysis is, you need to know what you see when opening a chart, more specifically a candlestick chart.

Let’s assume that the market opens at $100. Some trading activity occurs. Let’s also record the high price ($102) and the low price ($98) printed during the hourly period. Also, record the hourly close price ($101). Recall that these four pieces of data are referred to as *open*, *high*, *low*, and *close* (OHLC). They represent the four basic prices that are necessary to create candlestick charts.

*Candlesticks* are extremely simple and intuitive. They are box-shaped chronological elements across the timeline that contain the OHLC data. Figure 5-1 shows everything you need to know about how a candlestick works.

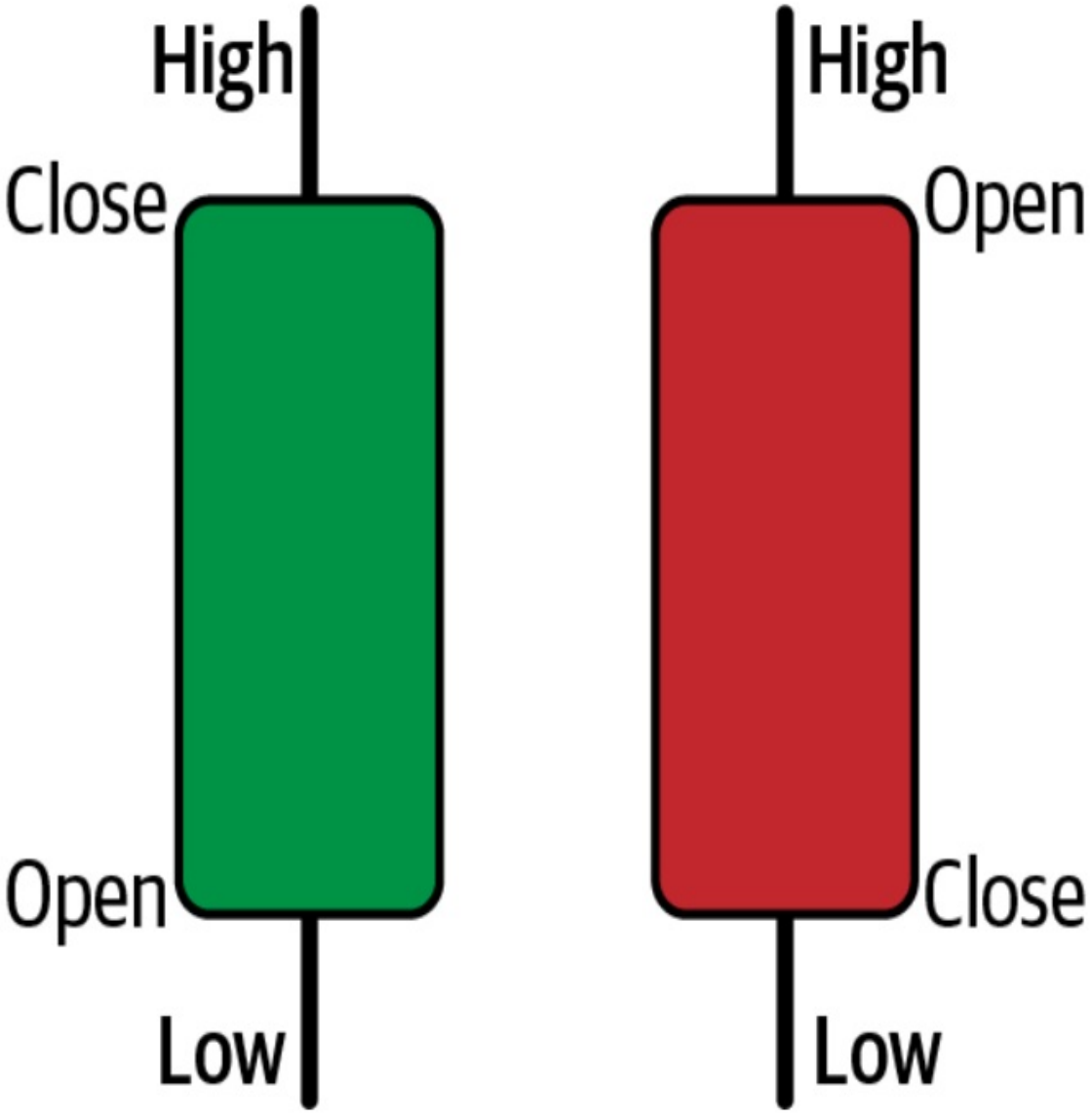


*Figure 5-1. On the left, a bullish candlestick; on the right, a bearish candlestick*

A *bullish* candlestick has a close price higher than its open price, whereas a *bearish* candlestick has a close price lower than its open price.

Candlestick charts are among the most famous ways to analyze financial time series. They contain more information than simple line charts and offer more visual interpretability than bar charts.

**NOTE**

A *line chart* is created by joining the close prices chronologically. It is the simplest way to chart an asset. It contains the least information among the three chart types since it shows only the close price.

*Charting analysis* is the task of finding support and resistance lines through subjective drawing. *Lines*, whether horizontal or diagonal, are the essence of finding levels to predict the market's reaction:

- *A support level* is a level from where the market should bounce, as it is implied that demand should be higher than the supply around it.
- *A resistance level* is a level from where the market should retreat, as it is implied that supply should be higher than the demand around it.

The asset's direction on a timeline axis can be threefold: *uptrend* where prices are making higher highs, *downtrend* where prices are making lower lows, and *sideways (or ranging)* where prices fluctuate around the same level for extended periods of time.

Figure 5-2 shows a support level on EURUSD close to 0.9850. Generally, traders start thinking about buying when a price is close to support. This is in anticipation of a reaction to the upside since the balance of power should shift more to the demand (positive) side, where traders accept to pay a higher price as they expect an even higher price in the future (remember the price-to-value argument discussed earlier). The implication here is that most traders see a price that is lower than the value.

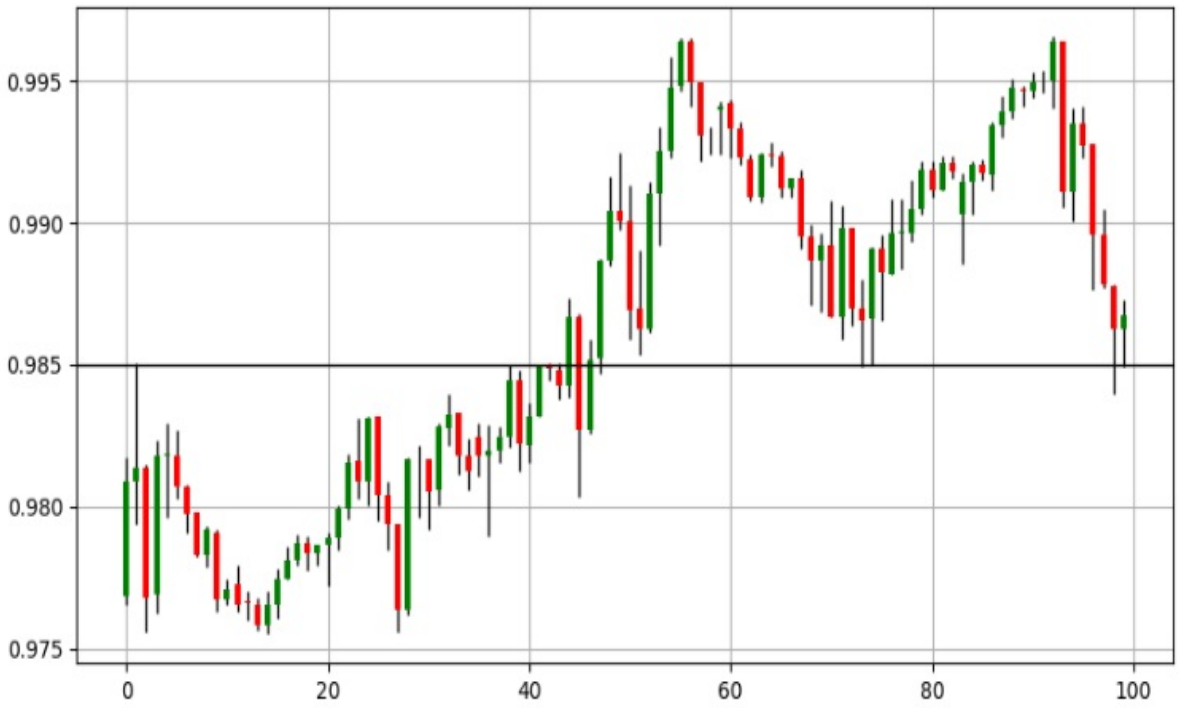


*Figure 5-2. Candlestick chart on EURUSD showing support at 0.9850*

Figure 5-3 shows a resistance level on EURUSD close to 0.9960. Generally, traders start thinking about shorting the market when it is close to resistance. This is in anticipation that a reaction to the downside should occur since the balance of power should shift more to the supply side. The implication here is that most traders see a price that is higher than the value.


*Figure 5-3. Candlestick chart on EURUSD showing resistance at 0.9960*

Ranging (sideways) markets give more confidence that horizontal support and resistance lines will work. This is because of the already implied general balance between supply and demand. Therefore, if there is excess supply, the market would adjust quickly, as demand should rise enough to stabilize the price.

Figure 5-4 shows a ranging market trapped between two horizontal levels; this is the case of EURUSD. Whenever the market approaches the resistance line in a ranging market, you should have more confidence that a drop will occur than you would in a rising market, and whenever it approaches support, you should have more confidence that a bounce will occur than you would in a falling market.

Furthermore, charting analysis is also applied on trending markets. This comes in the form of ascending and descending channels. They share the same inclination as horizontal levels but with a bias (discussed later).

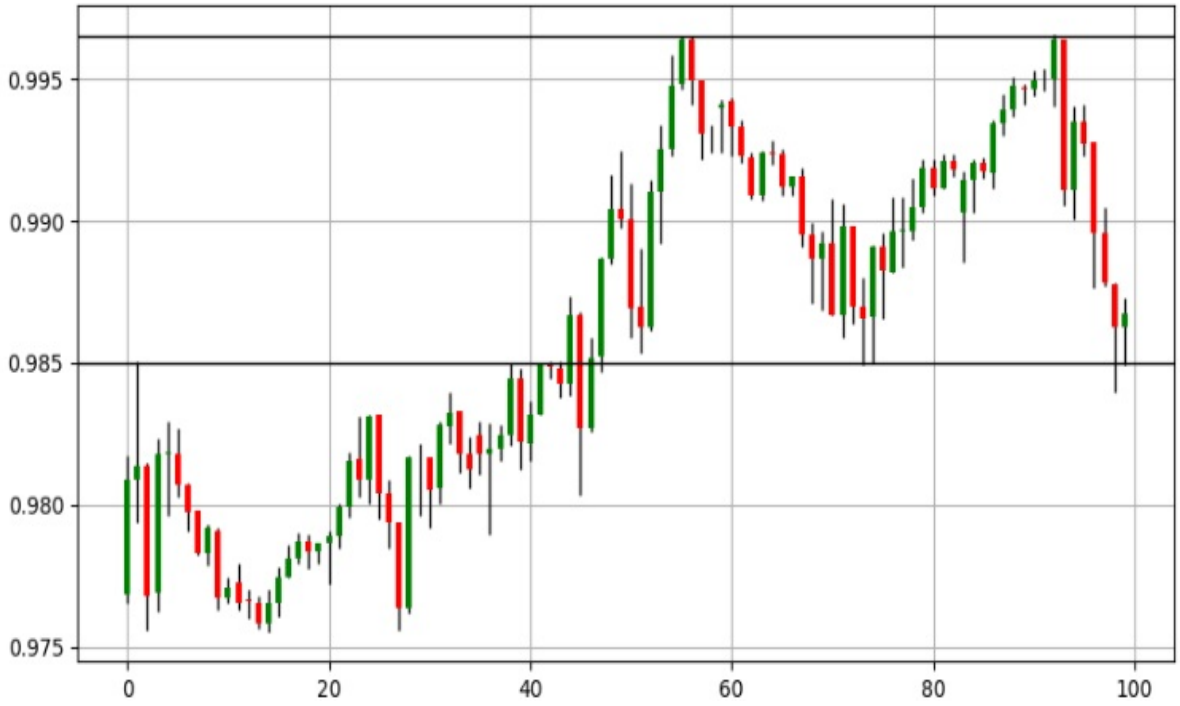


*Figure 5-4. Candlestick chart on EURUSD showing support at 0.9850 and resistance at 0.9960*

Figure 5-5 shows an *ascending channel* where support and resistance points rise over time to reflect the bullish pressure stemming from a steadily rising demand force.


*Figure 5-5. Candlestick chart on AUDUSD showing an ascending channel*

Traders seeing this would anticipate a bullish reaction whenever the market approaches the lower part of the ascending channel and would expect a bearish reaction whenever the market approaches the upper part of the channel.

This has no sound scientific backing because nothing says that the market must move in parallel, but the self-fulfilling prophecy may be why such channels are considered predictive in nature.

Figure 5-6 shows a descending channel where support and resistance points fall with time to reflect the bearish pressure coming from a steadily rising supply force. Generally, bearish channels tend to be more aggressive as fear dominates greed and sellers are more panicky than buyers are greedy.


*Figure 5-6. Candlestick chart on EURUSD showing a descending channel*

I mentioned a bias when dealing with ascending and descending channels. I refer to this bias as the *invisible hand*. Here's why:

"The trend is your friend." This saying, coined by Martin Zweig, means that with ascending channels, you need to be focusing more on buying whenever the market reverts to the support zone. That's because you want the invisible hand of the bullish pressure to increase your probability of a winning trade. Similarly, in the case of a descending channel, focus more on short selling whenever the

market reaches the upper limit. The full version of Zweig's axiom goes as follows: "The trend is your friend, until the end when it bends." This means that at any point in time, the market may change its regime, and any friendship with the trend gets terminated. In the end, charting analysis is subjective in nature and relies more on the experience of the trader or analyst.

It is worth mentioning also that there are many ways of finding support and resistance levels other than drawing them through visual estimation:

*Fibonacci retracements*

This is where you use Fibonacci ratios to give out reactionary levels. Fibonacci retracements are usually calculated on up or down legs so that you know where the market will reverse if it touches one of these levels. The problem with this method is that it is very subjective and, as with any other technique, not perfect. The advantage is that it gives many interesting levels.

*Pivot points*

With pivot points you use simple mathematical formulas to find levels. Based on yesterday's trading activity, you use formulas to project today's future support and resistance levels. Then whenever the market approaches the levels, you try to fade the move by trading in the opposite direction.

*Moving averages*

These are discussed in the next section. They are dynamic in nature and follow the price. You can also use them to detect the current market regime.

**TIP**

The best way to find support and resistance levels is to combine as many techniques as possible so that you have a certain confluence of methods, which in turn will increase your conviction for the initial idea. Trading is a numbers game, and stacking as much odds as possible on your side should eventually increase your chances for a better-performing system.

# Indicator Analysis

*Indicator analysis* is the second-most used technical analysis tool. It generally accompanies charting to confirm your initial idea. You can consider *indicators* as assistants. They can be divided into two types:

*Trend-following indicators*

> Used to detect and trade a trending market where the current move is expected to continue. Therefore, they are related to the persistence of the move.

*Contrarian indicators*

> Used to fade the move[2] and best used in sideways markets[3] as they generally signal the end of the initial move. Therefore, they are related to the expected reversal of the move (and therefore to the anti-persistence of the move).

The next sections present two pillars of technical analysis: moving averages (trend following) and the relative strength index (contrarian).

> **NOTE**
>
> Indicators are important as you will use them as inputs in the different learning algorithms in the subsequent chapters.

## Moving Averages

The most famous trend-following overlay indicator is the *moving average*. Its simplicity makes it without a doubt one of the most sought-after tools. Moving averages help confirm and ride the trend. You can also use them to find support and resistance levels, stops, and targets, as well as to understand the underlying trend.

There are many types of moving averages, but the most common is the simple moving average where you take a rolling mean of the close price, as shown in the following formula:

$$\text{Moving average}_i = \frac{Price_i + Price_{i-1} + ... + Price_{i-n}}{n}$$

Figure 5-7 shows the 30-hour simple moving average applied on USDCAD. The term *30-hour* means that I calculate the moving average of the latest 30 periods in case of hourly bars.


*Figure 5-7. Candlestick chart on USDCAD with a 30-hour simple moving average*

Rules of thumb with moving averages include the following:

- Whenever the market is above its moving average, a bullish momentum is in progress, and you are better off looking for long opportunities.
- Whenever the market is below its moving average, a bearish momentum is in progress, and you are better off looking for short opportunities.
- Whenever the market crosses over or under its moving average, you can say that the momentum has changed and that the market may be entering a new regime (trend).

You can also combine moving averages so that they give out signals. For example, whenever a short-term moving average crosses over a long-term moving average, a bullish crossover occurs, and the market may continue to rise.

This is also referred to as a *golden cross*.

In contrast, whenever a short-term moving average crosses under a long-term moving average, a bearish crossover has occurred, and the market may continue to drop. This is also referred to as a *death cross*.

Figure 5-8 shows USDCAD with a 10-hour (closer to the market price) and a 30-hour moving average (further from the market price). Note that, at the beginning, a golden cross appeared and signaled a beginning of a bullish trend.


*Figure 5-8. Candlestick chart on USDCAD with a 30-hour and a 10-hour simple moving average*

## The Relative Strength Index

Let's now look at the contrarian indicator. First introduced by J. Welles Wilder Jr.,[4]the *relative strength index* (RSI) is one of the most popular and versatile bounded indicators. It is mainly used as a contrarian indicator where extreme values signal a reaction that can be exploited. Use the following steps to calculate the default 14-period RSI:

1. Calculate the change in the closing prices from the previous ones.
2. Separate the positive net changes from the negative net changes.

3. Calculate a smoothed moving average on the positive net changes and on the absolute values of the negative net changes.
4. Divide the smoothed positive changes by the smoothed absolute negative changes. Refer to this calculation as the *relative strength* (RS).
5. Apply this normalization formula for every time step to get the RSI:

$$RSI_i = 100 - \frac{100}{1+RS_i}$$

> **NOTE**
>
> The *smoothed* moving average is a special type of moving average developed by the creator of the RSI. It is smoother and more stable than the simple moving average.

Generally, the RSI uses a lookback period of 14 by default, although each trader may have their own preferences on this. Here's how to use this indicator:

- Whenever the RSI is showing a reading of 30 or less, the market is considered to be oversold, and a correction to the upside might occur.
- Whenever the RSI is showing a reading of 70 or more, the market is considered to be overbought, and a correction to the downside might occur.
- Whenever the RSI surpasses or breaks the 50 level, a new trend might be emerging, but this is generally a weak assumption and more theoretical than practical in nature.

Figure 5-9 shows EURUSD versus its 14-period RSI in the second panel. Indicators should be used to confirm long or short bias and are very helpful in timing and analyzing the current  market state.

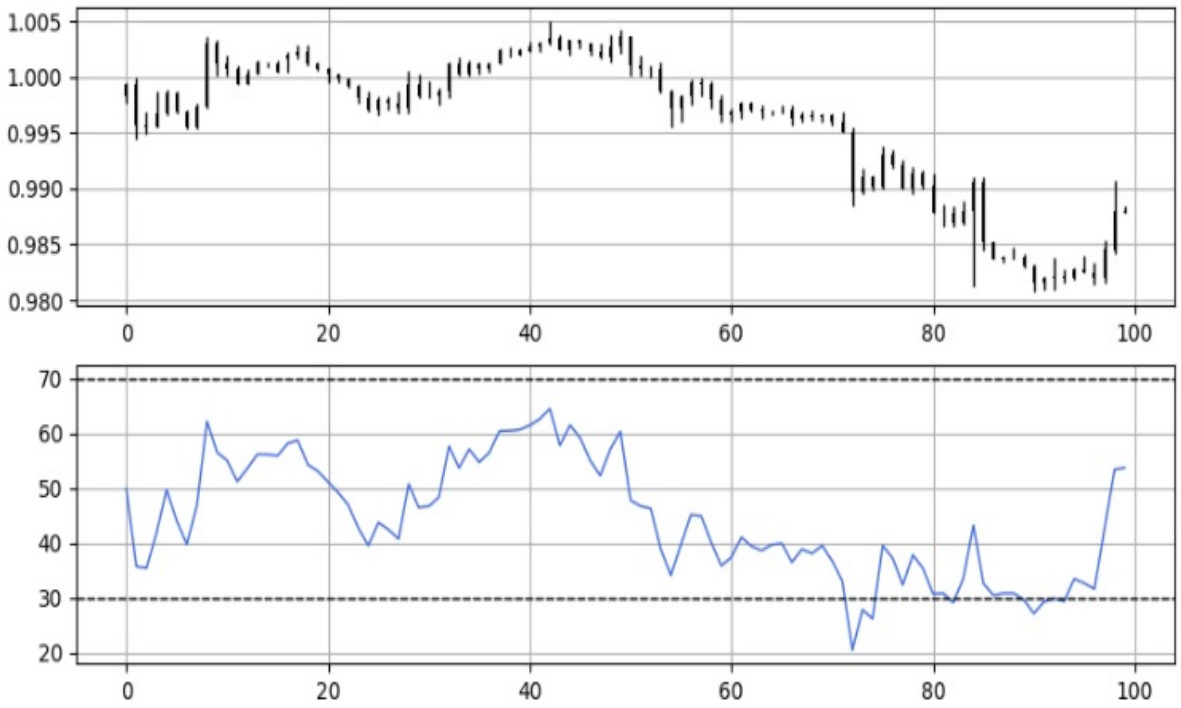


*Figure 5-9. Hourly EURUSD values in the top panel with the 14-period RSI in the bottom panel*

To summarize, indicators can be calculated in many ways. The two most commonly used ones are moving averages and the RSI.

## Pattern Recognition

*Patterns* are recurring configurations that show a specific prediction of the ensuing move. Patterns can be divided into the following types:

*Classic price patterns*

> These are known technical reversal price patterns, which are extremely subjective and can be considered unreliable due to the difficulty of back-testing them without taking subjective conditions. However, they are still used by many traders and analysts.

*Timing patterns*

> Based on a combination of timing and price, these patterns are less well known but can be powerful and predictive when used correctly.

*Candlestick patterns*

This is where OHLC data is used to predict the future reaction of the market. Candlesticks are one of the best ways to visualize a chart as they harbor many patterns that could signal reversals or confirm the move.

Classic price patterns refer to theoretical configurations such as double tops and rectangles. They are usually either reversal or continuation patterns:

*Continuation price patterns*

These are configurations that confirm the aggregate ongoing move. Examples include rectangles and triangles.

*Reversal price patterns*

These are configurations that fade the aggregate ongoing move. Examples include head and shoulders and double bottoms.

Old-school chartists are familiar with double tops and bottoms, which signal reversals and give the potential of said reversals. Despite their simplicity, they are subjective, and some are not visible like others.

This hinders the ability to know whether they add value or not. Figure 5-10 shows an illustration of a double top where a bearish bias is given right after the validation of the pattern, which is usually breaking the line linking the lows of the bottom between the two tops. This line is called the *neckline*.

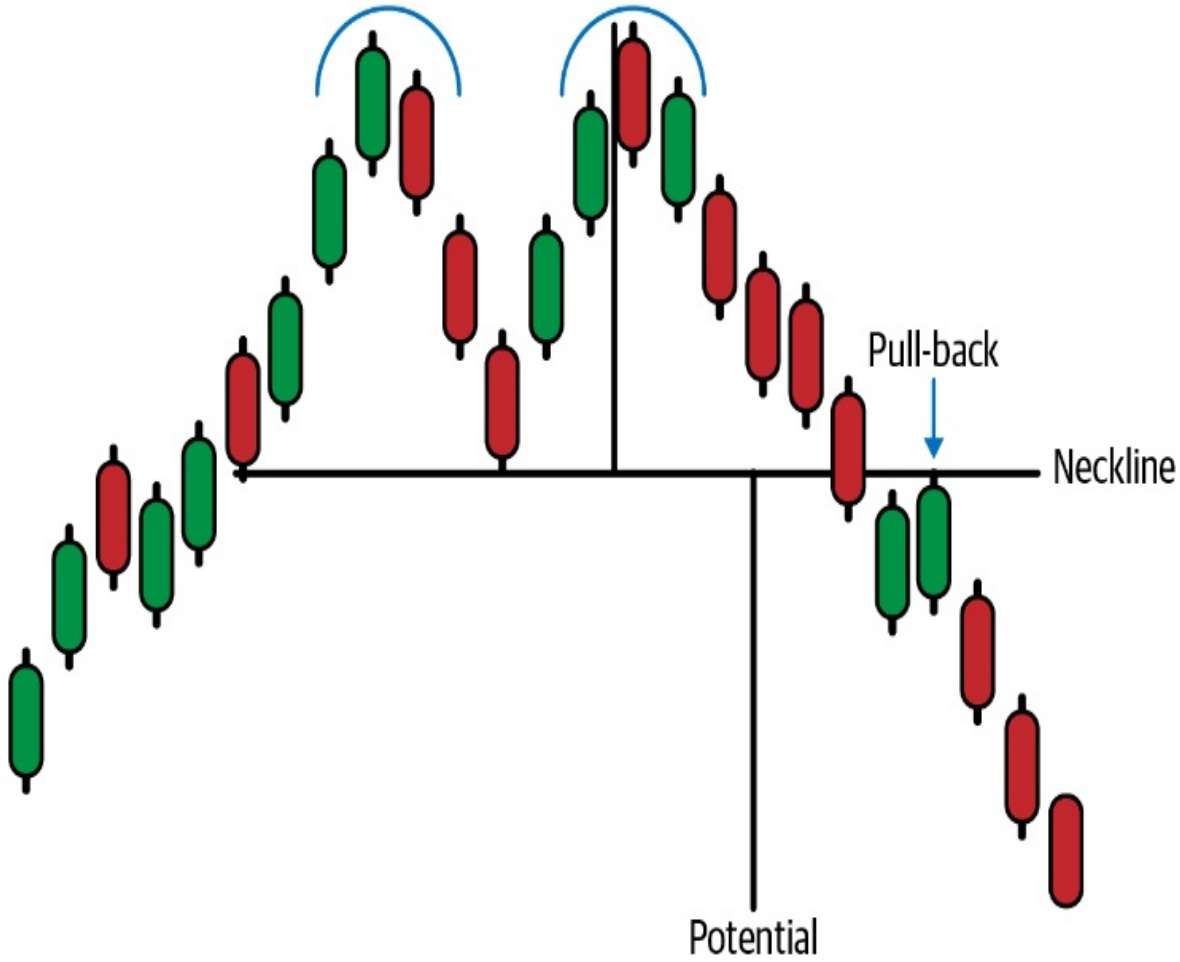


*Figure 5-10. Double top illustration*

Notice these three important elements in a double top:

*The neckline*

This is the line linking the lowest low between the two peaks and the beginning/end of the pattern. It serves to determine the pull-back level.

*The pull-back*

Having broken the neckline, the market should shape a desperate attempt toward the neckline but fails to continue higher as the sellers use this level as re-entry to continue shorting. Therefore, the pull-back level is the theoretical optimal selling point after validating a double top.

*The potential*

> This is the target of the double top. It is measured as the midpoint between the top of the pattern and the neckline projected lower and starting from the same neckline point.

The double top or bottom can have any size, but preferably it should be visible to most market participants so that its impact is bigger. Theoretically, the pattern's psychological explanation is that with the second top or bottom, the market has failed to push the prices beyond the first peak and therefore is showing weakness, which might be exploited by the seller.

There are other patterns that are more objective in nature; that is they have clear rules of detection and initiation. These are all based on clear objective conditions and are not subject to the analyst's discretion. This facilitates their back-testing and evaluation.

Before ending this chapter, I want to point out some misconceptions and best practices of technical analysis so that you start the right way.

## Common Pitfalls of Technical Analysis

Technical analysis can be misused, and this unfortunately fuels an everlasting debate about its utility and place relative to fundamental analysis. The important thing is to set expectations right and to remain within the means of logical thinking. This section talks about known pitfalls of technical analysis that you must make sure to avoid in order to maximize your survival rate in the financial jungle.

### Wanting to Get Rich Quickly

This pitfall is mostly psychological, as it deals with a lack of discipline and poor management. The need to make money as soon as possible to impress society pushes a trader to make emotional and bad decisions with regard to trades and trading-related activities.

This is also related to the fact that with the need to make money, you are likely to keep changing your trading plan because you believe that the new plan is a quicker way to wealth.

When you do not have enough faith in yourself and think that other people are better than you at making money, you are more likely to follow them, especially because of the abundance of information they provide. No one except you can change your future.

## Forcing the Patterns

A common psychological deficiency known as *confirmation bias* prevents traders from seeing signals that contradict their already established views.

Sometimes, you have an initial view on some market and therefore start looking for anything that agrees with the view, and this can also force patterns into existence even though there is no validity to them.

> **WARNING**
>
> You have to always be neutral in your analysis and approach elements with caution while retaining a maximum of objectivity. Of course, this is easier said than done, and the best alternative for absolute neutrality is an algorithmic approach that comes at the expense of the human intelligence factor.

## Hindsight Bias, the Dream Smasher

Technical analysis looks good in the past. Everything looked obvious and easy to predict, even with very basic strategies; however, when you apply the latter, you find deceiving results due to the harsh reality that your brain is wired to trick you into believing the past was perfectly predictable.

This is also why back-testing results almost always outperform forward-testing. When you look at past patterns and believe that they should have been easy to spot, you are exhibiting *hindsight bias*. To remedy this problem, you have to back-test using unbiased algorithms. Only then can you know for sure if the pattern adds value or not. Many retail traders fall into the trap of taking a general and simplistic look at the past to determine the validity of their strategies, which

then fail to perform.

### Assuming That Past Events Have the Same Future Outcome

You've heard the saying "History doesn't repeat itself, but it does often rhyme." This saying is key to understanding how the markets function. When you apply technical analysis, you must not expect exact outcomes from past signals and patterns. Rather, you must use them as guidelines and probabilistic elements that suggest markets may have a similar reaction that can be correlated with past reactions.

Trading is a numbers game, and you have to stack the odds in your favor. Humans sometimes do behave the same way when they are confronted with similar events. However, as different participants enter and leave the activity of buying and selling, with their motives changing all the time, you can be certain that the future reaction of the market after encountering a pattern similar to one in the past will not be exactly the same, although it might *rhyme* with the past, meaning it might have a correlated reaction on average.

This being said, do not expect to time the market perfectly every time you see a distinct pattern.

### Making Things More Complicated Than They Need to Be

Another saying is "Everything should be made as simple as possible, but no simpler." This is a perfect description of how you should do research and trade.

Financial markets are highly complex, semi-random environments that require more than simple strategies, yet strategies must not be so sophisticated that you fall into the trap of *overfitting*, a common issue where traders perfectly predict the past and assume it will behave exactly the same in the future, thus resulting in huge paper gains in the past but huge real losses in the future.

## Summary

Technical analysis offers a big selection of tools to analyze the markets either mathematically, graphically, or even psychologically (through patterns). This

chapter has marked the end of the warm-up exercise before starting the real aim of the book, machine and deep learning applications for trading and forecasting purposes. The learning outcome of this chapter should be in the form of a deep understanding what technical analysis is and what are its limitations. Similarly, you should also have a deep understanding in the two main technical indicators I have presented: moving averages and the RSI as they form key features in the coming models.

---

1 This assumes a nonrandom probability that over the long term shows deterministic characteristics.

2 Fading the move is a trading technique where you trade in the opposite direction of the on-going trend in the hopes that you are able to time its end.

3 Sideways markets are generally in equilibrium and no specific trend describes them. They tend to swing from top to bottoms that are close to each other.

4 See *New Concepts in Technical Trading Systems* by J. Welles Wilder Jr. (1978), published by Trend Research.

# Chapter 6. Introductory Python for Data Science

**A NOTE FOR EARLY RELEASE READERS**

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 6th chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at *ccollins@oreilly.com*.

One more stop to go before diving into the realm of machine and deep learning. This chapter is optional for experienced Python developers but is crucial for anyone without a solid programming background. Understanding the intuition behind the algorithms is a great advantage but it will not get you far if you fail to properly implement them. After all, these algorithms need code to work and do not function manually. Make sure to understand the basic syntax and how to manipulate data and transform it.

As the book is not meant to be an A-Z guide to programming in Python, this chapter gently brushes on some essentials and a few more techniques that should help you navigate smoothly the subsequent chapters.

## Downloading Python

*Coding* is defined as a set of instructions designed to be executed by a computer. Generally, specific syntax is required so that the computer applies the set of instructions without errors. There are many coding languages and are divided into two broad categories:

*Low-level coding languages*

> These are machine languages usually for operating systems and firmwares. They are very difficult to read. These languages have a sizable level of control over hardware. An example of a low-level language is assembly (with its various types).

*High-level coding languages*

> These are user-friendly languages (with a high level of abstraction). They are generally used to code programs and softwares. An example of a high-level language is Python and Julia.

The coding language used in this book is *Python*, a popular and versatile language with many advantages and wide adoption in the research and professional community. As you have seen from the chapter's name, you will see an introduction to Python with the necessary tools to start building your own scripts. But before that, let's see how to actually download Python.

A *Python interpreter* is a software used to write and execute code written using Python syntax. I use a software called *Spyder*. Some people may be more familiar with other interpreters such as *Jupyter* and *PyCharm*, but the process is the same. You can download *Spyder* from the official website or, even better, download it as part of a bigger package called Anaconda, which facilitates installation and offers more tools. Note that it is an open source and free-to-use software.

*Spyder's* interface is split into three windows, as you can see in Figure 6-1. The window on the left is used to write the code that is later executed (the algorithm is told to run and apply the code). Typically, you will see multiple lines of code in that area.

The window on the upper right is the *variable explorer*. Every time a variable is stored (defined), you can see it there. The window on the lower right is the *console* that shows the result of the code, whether it is an error or an output.

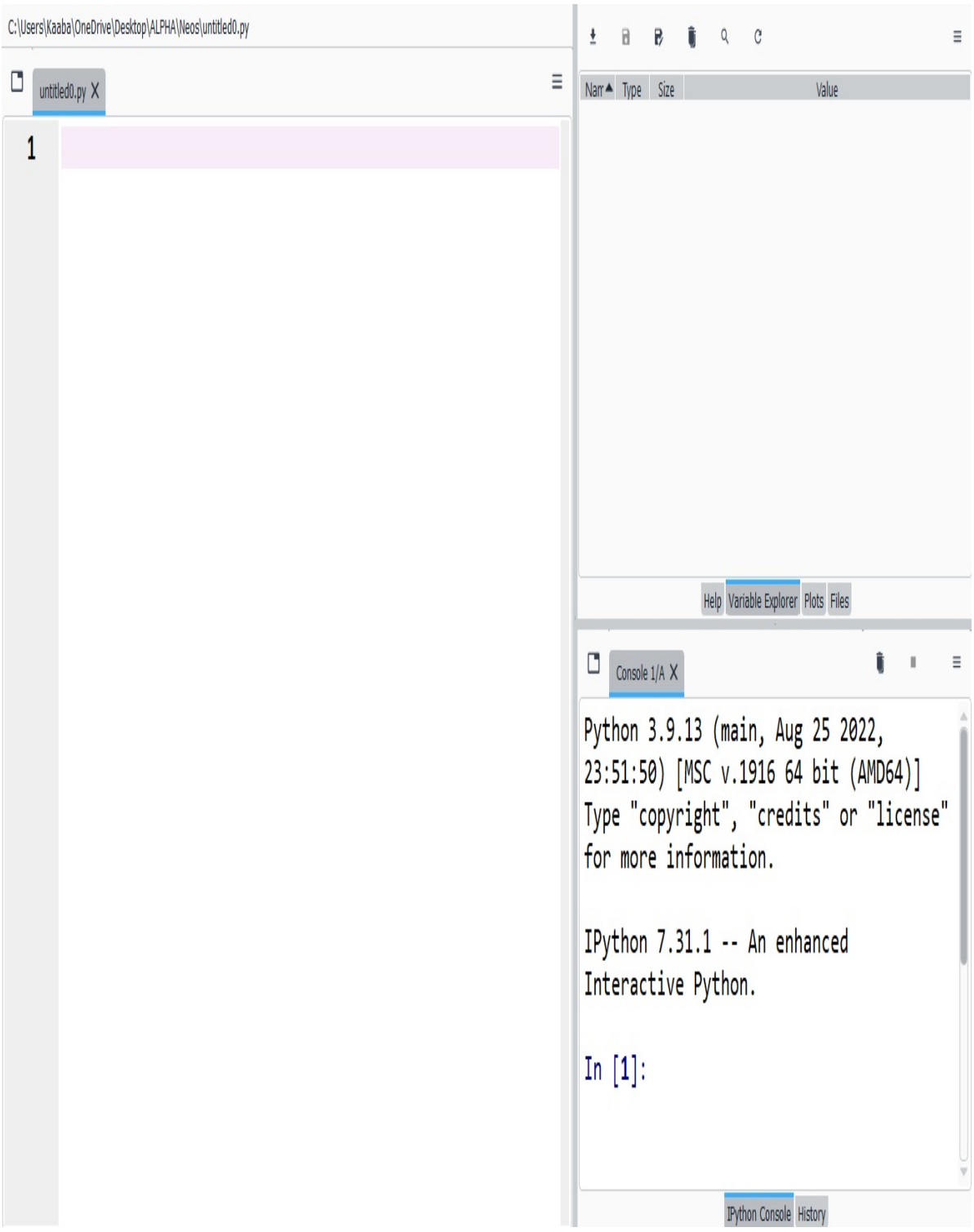


*Figure 6-1. Spyder's interface*

Python files have the extension *name.py* and they allow you to save the code and refer to it at a later stage. You can also open multiple files of code and navigate between them. The outline of this chapter is as follows:

1. Understand the language of Python and how to write error-free code.
2. Understand how to use control flow and its importance with time series analysis.
3. Understand what are libraries and functions and their role in facilitating coding.
4. Understand how to handle errors and their different types.
5. Understand how to use data manipulation libraries such as `numpy` and `pandas`.
6. Finally, see how to import historical financial time series data into Python so that it gets analyzed with the proper tools that you have seen in previous chapters but also in the coming chapters.

## Basic Operations and Syntax

*Syntax* is the proper way of writing error-free code, it is the structure of statements needed to write code that functions. When you are communicating with a computer, you have to make sure it understands you and therefore, having a solid understanding of syntax is important.

Understanding code comes with a useful action called comments. A *comment* is a non-executable code used to explain the executable code right after. This is used so other programmers understand the code. Comments are preceded by a hashtag #:

```
# This is a comment. Comments are ignored by the interpreter

# Comments explain the code or give more details about its use

# Comments are written in one line, otherwise, you have to re-write
'#'
```

**NOTE**

Make sure you understand that comments are non-executable. This means that when you run (execute) the code, they will be ignored by the interpreter and they will not return an error.

Sometimes you need to write documentation for your code which may require multiple lines of code (even paragraphs in some instances). Writing the hashtag symbol at every line can be tedious and cluttersome. This is why there is a way to write long comments. To do this you have to write your comments between three quotes at every end as follows:

```
'''

Python was created in the late 1980s by Guido van Rossum.

The name "Python" was inspired by the comedy group Monty Python.

'''
```

It is worth noting that triple quotes are considered *docstrings* and not really comments (according to the official Python documentation).

Let's discuss variables and constants. A *constant* is a fixed value that does not change while a *variable* takes on different values given an event. A constant can be the number 6 while a variable can be the letter *x* which takes on any number given a set of conditions or a state. A variable is defined using the '=' operator:

```
# Defining a variable
x = 10
y = 5

# Writing a constant
6
```

Running the previous code will store the variables *x* and *y* with their respective values (in the variable explorer). Simultaneously, the output of the code will be 6. Variables are case sensitive, therefore:

```
# Declaring my_variable
my_variable = 1

# Declaring My_variable
My_variable = 2

# The variable my_variable is different from My_variable
```

Variable declaration cannot start with a number but it can contain one in the middle or the end of its name:

```
# Returns a SyntaxError
1x = 5

# Valid declaration
x1 = 5

# Valid declaration
x1x = 5
```

Variables can also contain underscores but nothing else:

```
# Returns a SyntaxError
x-y = 5

# Valid declaration
x_y = 5
```

It is heavily recommended that variables are short and straightforward. For example, consider creating the variable that holds the lookback period of a certain moving average (a concept seen in Chapter 5):

```
# Recommended name
ma_lookback = 10

# Not recommended name
the_lookback_on_that_moving_average = 10
```

There are several different data types with different characteristics:

*Numerical data types*

This is simplest data type. It is formed exclusively from numbers. numerical data types are even further divided into integers, float numbers, and complex numbers. *Integers* are simple whole numbers (positive or negative). An example of an integer would be 6 and -19. *Float numbers* are more precise than integers as they incorporate the values after the comma. An example of a float number would be 2.7 and -8.09. *Complex numbers* include imaginary

numbers[1] and are less relevant than the other two.

*Strings*

As you have seen previously with comments and docstrings, it is possible to write text next to the code without it interfering with the execution process. *Strings* are text structures that represent sequences of characters. Strings can be inputs and arguments of functions and not necessarily just comments.

*Booleans*

This is a binary (true or false) data type used to evaluate the truth value of the given expression or condition. For example, you can use booleans to evaluate if the market price is above or below the 100-period moving average.

*Data collection*

These are sequences that contain multiple data with each having a different and unique usage. An *array* is a sequence of elements of the same type (mostly numerical). Arrays will be used frequently in this book (used with a Python library called `numpy` that is discussed in this chapter). A *data frame* is a two-dimensional table of structured data that is also frequently used in this book (used with a Python library called `pandas` that is discussed in this chapter). A *set* is a sequence of unordered elements. A *list* is an ordered collection of elements that can be of different data types. A *tuple* is an ordered, immutable collection of elements that may be of different data types. It is used for storing a fixed sequence of values. A *range* is a built-in Python function that returns a sequence of numbers. The range function is

mostly used in loops. A *dictionary* represents a collection of key-value pairs grouped together.

The following code snippet shows a few examples on the numerical data type:

```
# Creating a variable that holds an integer
my_integer = 1

# Creating a variable that holds a float number
my_float_number = 1.2

# Using the built-in Python function type() to verify the variables
type(my_integer)
type(my_float_number)
```

The output should be as follows (remember that anything after the hashtag symbol is a comment and will not be executed):

```
int # The output of type(my_integer)

float # The output of type(my_float_number)
```

Strings are simply text. The most famous example of explaining a string is the "*Hello World*" phrase as explained in the following code snippet:

```
# Outputting the phrase "Hello World"
print('Hello World')
```

The output should be as follows:

```
Hello World
```

Strings can also be used as arguments in functions, both concepts you will see later in this chapter.

Booleans as mentioned in the previous list are either true or false values. The following code snippet shows an example of using them:

```
# Make a statement that the type of my_integer is integer
type(my_integer) is int
```

```
# Make a statement that the type of my_float_number is float
type(my_float_number) is float

# Make a statement that the type of my_integer is float
type(my_integer) is float

'''
Intuitively, the two first statements will return True as they are
indeed true. The third statement is False as the variable my_integer
is an integer and not a float number

'''
```

The output of the previous code is as follows:

```
True
True
False
```

Let's discuss how operators work. You have actually already seen an example of an operator which is the assignement operator '=' used to defined variables. *Operators* perform special mathematical and other tasks between variables, constants, and even data structures. There are different types of operators. Let's start with *arithmetic operators* as shown in the following snippet:

```
# Arithmetic operator - Addition
1 + 1 # The line outputs 2

# Arithmetic operator - Subtraction
1 - 1 # The line outputs 0

# Arithmetic operator - Multiplication
2 * 2 # The line outputs 4

# Arithmetic operator - Division
4 / 2 # The line outputs 2.0 as a float number

# Arithmetic operator - Exponents
2 ** 4 # The line outputs 16
```

The next type of operators is the *comparison operators* which are used to compare different elements. They are mostly used in control flow events as

explained in the next section of this chapter. The following snippet shows a few comparison operators:

```
# Comparison operator - Equality
2 == 2 # The line outputs True

# Comparison operator - Non equality
2 != 3 # The line outputs True

# Comparison operator - Greater than
2 > 3 # The line outputs False

# Comparison operator - Greater than or equal to
2 >= 2 # The line outputs True

# Comparison operator - Less than
2 < 3 # The line outputs True

# Comparison operator - Less than or equal to
2 <= 2 # The line outputs True
```

*Logical operators* combine two or more conditions that are later evaluated. There are three logical operators: `and`, `or`, and `not`. The following code block shows an example of logical operators:

```
# Logical operator - and
2 and 1 < 4 # The line outputs True
2 and 5 < 4 # The line outputs False

# Logical operator - or
2 or 5 < 4 # The line outputs 2 which is the integer less than 4
```

Data collection structures (arrays an data frames) are discussed in a later section as they require an in-depth presentation due to their complexity and unique tools. Let's end this section with a code that englobes what has been discussed so far:

```
# Declaring two variables x and y and assigning them values
x = 10
y = 2.5

# Checking the types of the variables
type(x) # Returns int
type(y) # Returns float
```

```
# Taking x to the power of y and storing it in a variable z
z = x ** y # Returns 316.22

# Checking if the result is greater than or equal to 100
z >= 100 # Returns True as 316.22 >= 100
```

## Control Flow

Conditional statements form the first part of what is known as control flow (the second part is loops). *Conditional statements* are the ancestors of today's artificial intelligence as they only execute code if certain conditions are met.

Conditional statements are managed using `if`, `elif`, and `else`. Take the following code snippet as an example to clear things out:

```
# Declaring the variables
a = 9
b = 2

# First condition (specific)
if a > b:

    print('a is greater than b')

# Second condition (specific)
elif a < b:

    print('a is lower than b')

# Third condition (general)
else:

    print('a is equal to b')
```

Therefore, conditional statements start with `if`, then for every new unique and specific condition, `elif` is used, until it makes sense to use the rest of the probability universe as a condition on its own which is used by the `else` statement. Note that the `else` statement does not need a condition as it exist to cover the rest of the uncovered universe.

*Loops* are used to execute blocks of code repeatedly until a pre-defined condition is met. Loops are heavily used with time series to calculate indicators, verify states, and to back-test trading strategies.

Loops are managed using `for` (used to iterate over a finite and defined sequence or a range of elements) and `while` (used to continue the iteration until a condition is met) statements. Take as an example the following code that prints the values {1, 2, 3, 4} using a loop:

```
# Using a for loop
for i in range(1, 5):

    print(i)

# Using a while loop
i = 1
while i < 5:

    print(i)
    i = i + 1
```

The `for` loop when translated is simply saying that for every element which is called `i` (or any other letter depending on the coder) in the range that starts at 1 and ends at 5 (excluded), print the value of `i` at every loop (hence, in the first loop, the value of `i` is equal to 1 and in the second loop, it is equal to 2).

The `while` loop when translated is saying that starting from a value of `i = 1`, while looping, print its value and then add 1 to it before finishing the first loop. End the loop when `i` becomes greater than 4

> **NOTE**
>
> Theoretically, a `while` loop is infinite until told otherwise.

It is worth noting that `i = i + 1` can also be expressed as `i += 1`. The goal of an algorithm is the ability to apply many operations recursively in an objective way which makes loops extremly useful especially when combined with conditional statements. Let's take an example of a financial time series:

1. Create a range of values to simulate hypothetical daily close prices.
2. Loop through the range of the data while creating the condition that if the price rose from the last period, print 1. Similarly, if the price fell from the

last period, print -1. Lastly, print 0 if the price didn’t change from last period.

This can be done in the following code block:

```
# Creating the time series
time_series = [1, 3, 5, 2, 4, 1, 6, 4, 2, 4, 4, 4]

for i in range(len(time_series)):

    # The condition where the current price rose
    if time_series[i] > time_series[i - 1]:

        print(1)

    # The condition where the current price fell
    elif time_series[i] < time_series[i - 1]:

        print(-1)

    # The condition where the current price hasn't changed
    else:

        print(0)
```

The code defines a list of values (in this case, a time series called `time_series`), then loops around its length using the `len()` function to apply the conditions. Notice how at every loop, the current time step is referred to as `i` thus making the previous time step `i - 1`.

## Libraries and Functions

A *library* in Python is a group of pre-written code that offers functionality to make the creation of applications easier. *Modules*, which are individual Python files with reusable code and data that can be imported and used in other Python code, are commonly found in libraries. A module is therefore a single Python file that contains functions and other types of code that may be used and imported by other Python programs. Large code bases are often easier to manage and maintain by using modules to divide similar code into different files.

Coding is all about simplifying tasks and making them clearer. When you have a

recurring task such as calculating a moving average of a time series, you can use a function so that you do not have to write the moving average code all over again every time you want to use it. Instead, you define the function with the original code and then, you call it whenever you need to calculate the moving average. But what is a *function*? It is a block of reusable code that performs a specific task when called. It needs to be defined once.

Multiple functions form a module and multiple modules form a library. A library is generally theme-oriented. For example, in this book, `sklearn` library will be used with machine learning models. Similarly, data manipulation and importing is done using `numpy` and `pandas`, two libraries discussed in a later section of this chapter. Plotting and charting is done using `matplotlib` library.

Libraries must be imported first to the Python interpreter before being used. The syntax of doing this is as follows:

```
# The import statement must be followed by the name of the library
import numpy

# Optionally, you can give the library a shortcut for easier
references
import numpy as np
```

Sometimes, you need to import just one function or module from a library. For this, you don't need to import the totality of the library:

```
# Importing one function from a library
from math import sqrt
```

So, it is established that `math` is a Python library that harbors many mathematical functions, namely `sqrt` function which is used to find the square root of a given number. Let's see how to define a function. A function is defined using `def` followed by the name of the function and any optional arguments. Consider the following example that creates a function that sums any two given variables:

```
# Defining the function sum_operation and giving it two arguments
def sum_operation(first_variable, second_variable):

    # Outputing the sum of the two variables
```

```
    print(first_variable + second_variable)

# Calling the function with 1 and 3 as arguments
sum_operation(1, 3) # The output of this line is 4
```

**NOTE**

Calling a function means executing what it's supposed to do. In other words, calling a function is simply using it. The time line of a function is getting defined and then getting called.

Let's see how to import a function from a library and use its functions:

```
# Importing the library
import math

# Using the natural logarithm function
math.log(10)

# Using the exponential function (e)
math.exp(3)

# Using the factorial function
math.factorial(50)
```

As a side note, the *factorial* operation is a mathematical operation that used to calculate the product of all positive integers from 1 up to a certain number (which is the argument requested in `math.factorial()`).

Libraries may not be as easy as one plus one. Sometimes, external libraries require installation first before being able to be imported to the Python interpreter. Installation can be done through the prompt using the following syntax:

```
pip install library_name
```

Let's revert back to Chapter 3 where the MIC was discussed. Prior to the following already seen code of the MIC:

```
# Importing the library
from minepy import MINE
```

```
# Calculating the MIC
mine = MINE(alpha = 0.6, c = 15)
mine.compute_score(sine,cosine)
MIC = mine.mic()
print('Correlation | MIC: ', round(MIC, 3))
```

Importing the library directly will likely lead to an error as it has not been pip installed. Therefore, you must install it first using the following syntax on the prompt (not the Python interpreter):

```
pip install minepy
```

It is also important to read the documentation that comes with libraries in order to use them correctly. *Documentation* helps knowing the aim of the functions and what arguments go inside. Furthermore, it tells you what type of arguments the function can accept (for example, strings or numerics). Let's go back to functions now (notice how both are intertwined and discussing one may result in discussing the other).

Functions can have a `return` statement which allows the result to be stored in a variable so that it can be used in other parts of the code. Let's take two simple examples and then discuss them step-by-step:

```
# Defining a function to sum two variables and return the result
def sum_operation(first_variable, second_variable):

    # The summing operation is stored in a variable called final_sum
    final_sum = first_variable + second_variable

    # The result is returned
    return final_sum

# Create a new variable that holds the result of the function
summed_value = sum_operation(1, 2)

# Use the new variable in a new mathematical operation and store the
result
double_summed_value = summed_value * 2
```

The previous code defines the `sum_operation` function with two arguments, then stores the operation in a variable called `final_sum` before returning it so it can be stored externally. Afterwards, a new variable called `summed_value`

is defined as the output of the function. Finally, another variable is created under the name of `double_summed_value` and is the result of `summed_value` multiplied by 2. This is an example on how to use results from functions as variables in external operations. Now, let's consider another example (while keeping in mind the previously defined `sum_operation` function):

```
# Defining a function to square the result gotten from the
sum_operation function
def square_summed_value(first_variable, second_variable):

    # Calling the nested sum_operation function and storing its result
    final_sum = sum_operation(first_variable, second_variable)

    # Creating a variable that stores the square of final_sum
    squared_sum = final_sum ** 2

    # The result is returned
    return squared_sum

# Create a new variable that holds the result of the function
squared_summed_value = square_summed_value(1, 2)
```

The latest code snippet defines a function called `square_summed_value` which takes on two arguments. Furthermore, it uses a nested function inside which in this case is `sum_operation`. The result of the nested function is once again stored in a variable called `final_sum` which is used as an input in finding the `squared_sum variable`. The variable is found as `final_sum` to the power of two. This was an example on how to create functions out of other functions inside of them (in other words, nested functions).

Let's end the section with common libraries in Python and machine learning (other than `numpy` and `pandas`):

```
matplotlib # For plotting and visualizing data
sklearn    # For machine learning models
scipy      # For scientific computing and optimization
keras      # For neural networks
math       # For using mathematical tools such as square roots
random     # For generating random variables
requests   # For making HTTP requests used in web scraping
```

# Exceptions Handling and Errors

Quite often you will run into errors due to issues with the code during execution. In other words, errors occur when the code is executed and the interpreter finds an obstacle that prevents it from continuing further. The most basic error is *SyntaxError* which occurs when there are misspelled words or missing elements which make the code unintelligible:

```
# Will not output a SyntaxError if executed
my_range = range(1, 10)

# Will output a SyntaxError is executed
my_range = range(1, 10
```

As you can see from the previous code, there is a missing parenthesis at the end of the second code line, which is not understood by the interpreter. This type of error is likely to be the most common one. Another common error is *NameError* which occurs when failing to define a variable before executing a code that contains it. Consider the following example:

```
x + y
```

The previous code will give you a *NameError* because the interpreter does not know the value of `x` and `y` since they were not defined.

The *ModuleNotFoundError* occurs when the interpreter cannot find the library or module you are trying to import. This generally occurs when it is installed in the bad directory or when it is not properly installed. Common fixes for this issue include:

- Verifying if the module's name has been written correctly.
- Verifying if the module has been correctly pip installed.
- Verifying if the module is installed in the correct location.

Another type of common errors is *TypeError* and it occurs when you apply a certain operation on an incompatible element such as summing an integer with a string. The following operation raises a *TypeError*:

```
# Defining variable x
x = 1

# Defining variable y
y = 'Hello

# Summing the two variables which will raise a TypeError
x + y
```

In time series analysis, you will likely to encounter these four errors:

- *IndexError*: This is raised referring to an index that is out of range regarding the current array or data frame. Imagine having an array of 300 values (rows). If you want to loop through them and at each loop, you want to input the number 1 in the next cell (time step + 1), you are likely to encounter an *IndexError* as in the last loop, there is no next cell and the interpreter will raise this error.
- *ValueError*: This is raised when you try to call a function with an invalid argument. An example of this would to be try to pass an integer element as a string when calling a function.
- *KeyError*: This occurs when trying to access an element in a data frame that does not exist. For example, if you have three columns in the data frame and you refer to one that does not exist (maybe due to a syntax issue), you are likely to run into a *KeyError*.
- *ZeroDivisionError*: This error is intuitive and occurs when trying to divide a number by zero.

There are other types of errors that you may encounter. It is important to understand what they refer to so that you are able to fix them.

*Exceptions* on the other hand may be not fatal to the code in the sense that they only show a warning but not necessarily terminate the code. Therefore, exceptions occurs during the execution (as opposed to errors which occur because the interpreter is unable to execute).

**NOTE**

Understanding the error will help you fixing it and getting the code running again.

To ignore certain exceptions, the `try` and `except` keywords are used. This is useful when you are certain that handling the exception will not alter the output of the code. Let's take an example of creating a function that divides the first column of a time series by the next value of the second column. The first step is to define the time series as a data frame or as an array (or any other data collection structure):

```
# Importing the required library to create an array
import numpy as np

# Creating a two-column list with 8 rows
my_time_series = [(1, 3),
                  (1, 4),
                  (1, 4),
                  (1, 6),
                  (1, 4),
                  (0, 2),
                  (1, 1),
                  (0, 6)]

# Transforming the list into an array
my_time_series = np.array(my_time_series)
```

Now, let's write the division function which will take any value in the first column and divide it by the next value in the second column:

```
# Defining the function
def division(first_column, second_column):

    # Looping through the length of the created array
    for i in range(len(my_time_series)):

        # Division operation and storing it in the variable x
        x = my_time_series[i, first_column] / my_time_series[i + 1,
second_column]

        # Outputting the result
        print(x)

# Calling the function
division(0, 1)
```

Running the two previous code blocks will give an *IndexError* because in the last loop, the function cannot find the next value of the second column because it simply does not exist:

```
IndexError: index 8 is out of bounds for axis 0 with size 8
```

Fixing this through `try` and `except` will ignore the last calculation that is causing the problem and will return the expected results:

```
# Defining the function
def division(first_column, second_column):

    # Looping through the length of the created array
    for i in range(len(my_time_series)):

        # First part of the exception handling
        try:

            # Division operation and storing it in the variable x
            x = my_time_series[i, first_column] / my_time_series[i +
1, second_column]

            # Outputting the result
            print(x)

        # Exception handling of a specific error
        except IndexError:

            # Ignoring (passing) the error
            pass

# Calling the function
division(0, 1)
```

The output is as follows:

```
0.25
0.25
0.16666666666666666
0.25
0.5
0.0
0.16666666666666666
```

This section discussed errors and exceptions and how you can handle such issues. In the coming chapters, such issues will be handled using `try` and `except` blocks due to their simplicity and effectiveness.

## Data Structures in Numpy and Pandas

The words `numpy` and `pandas` may come familiar to you since I have used them in most of the code snippets in the previous chapters. Moreover, you now understand what a library is and therefore, you know that these two are the go-to libraries to manipulate, handle, and import data in Python. This section shows the differences between the two and their key functions that are definitely a great addition to your data analysis. But first, let's define these two libraries:

*numpy*

> NumPy (short for Numerical Python) is a Python library that allows working with multi-dimensional arrays and matrices. NumPy provides a powerful interface for performing various operations on arrays and matrices.

*pandas*

> Pandas (short for Panel Data) is a Python library that allows working with data frames (a type of tabular data). Pandas provides two main data structures: series and data frames. A *series* is a one-dimensional array-like object that can hold any data type. A *data frame* is a two-dimensional table-like structure that consists of rows and columns (similar to a spreadsheet).

Both libraries are very useful in analyzing time series data. Arrays hold only numerical type data and therefore, they do not really hold date type data as opposed to data frames. This may be one of the advantages of using `pandas` as opposed to `numpy` but both have the strengths and relative weaknesses. In the end, it is a matter of choice. This book will prioritize using numpy due to the simplicity and due to the fact that the machine learning models seen in the next chapter use `sklearn` library which is applied on arrays. This must not prevent you from using data frames until you are ready to apply the models.

> **NOTE**
>
> Switching between `numpy` and `pandas` requires converting the time series type. It is a relatively easy task but can sometimes cause loss of certain types of data (for example, date data).

Let's import both libraries before starting to see some of their potential:

```
import numpy as np
import pandas as pd
```

The following code creates two time series with two columns and three rows. The first time series is called `my_data_frame` and is created using the function `pd.DataFrame` of `pandas`. The second time series is called `my_array` and is created using the function `np.array` of `numpy`:

```
# Creating a data frame
my_data_frame = pd.DataFrame({'first_column' : [1, 2, 3],
                              'second_column' : [4, 5, 6]})

# Creating an array
my_array = np.array([[1, 4], [2, 5], [3, 6]])
```

As can be seen from Figure 6-2, data frames have real indexes and can have column names. Arrays are purely numerical and do not allow for this:

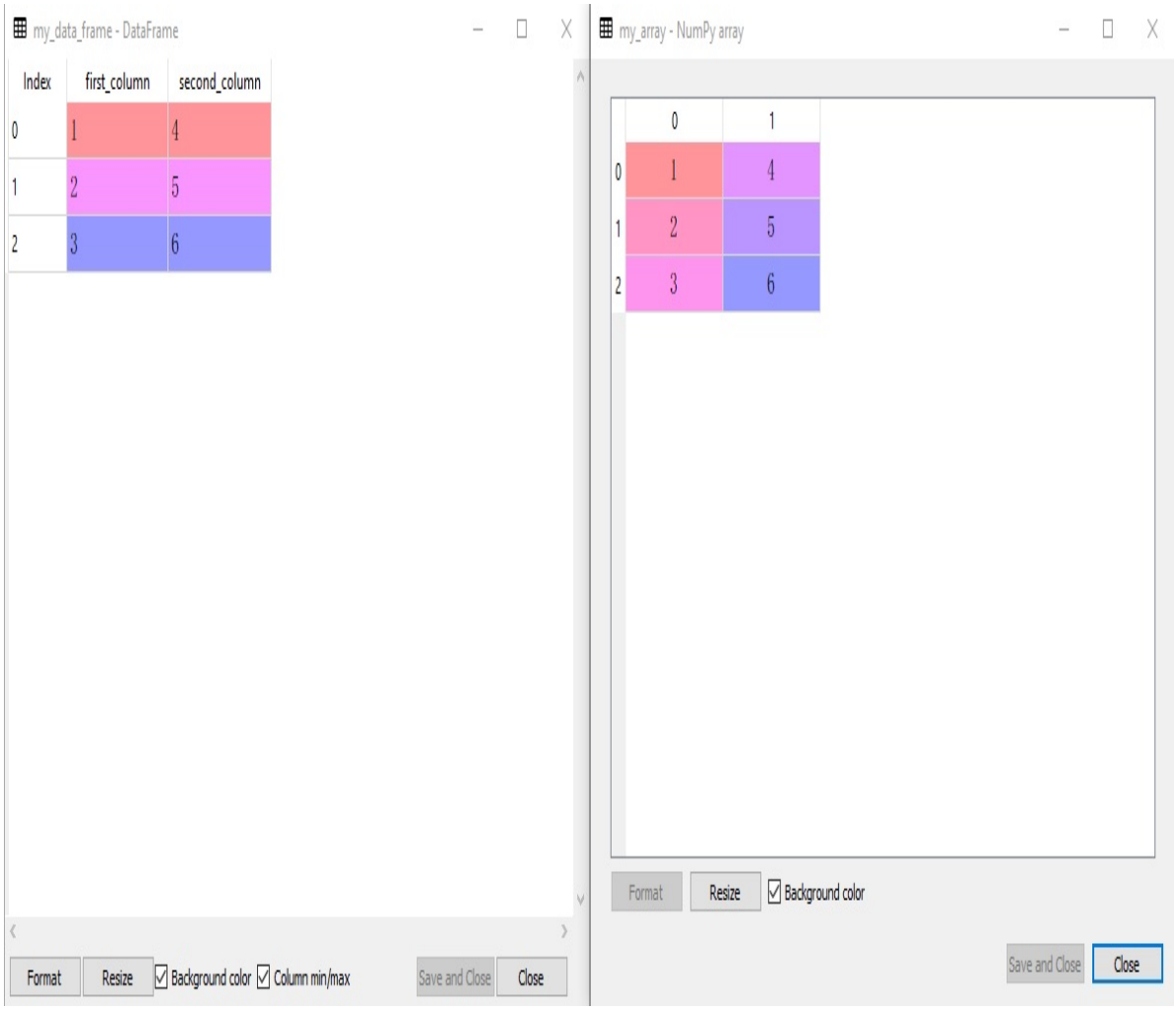


*Figure 6-2. On the left, a data frame (pandas) and on the right, an array (numpy)*

To switch between the two types of data, it is quite intuitive as you will be using the same two functions used in the previous code block:

```
# To transform my_data_frame into my_new_array
my_new_array = np.array(my_data_frame)

# To transform my_array into my_new_data_frame
my_new_data_frame = pd.DataFrame(my_array)
```

> **NOTE**
>
> You can notice that `my_new_data_frame` does not have column names.

Let’s now take a look at useful functions that will come in handy when dealing with models. Slicing, concatenating, and other tools are things that you must master in order to smoothly navigate through the data analysis part. Consider the following arrays:

```
first_array  = np.array([ 1,  2,  3,  5,   8,  13])
second_array = np.array([21, 34, 55, 89, 144, 233])
```

*Concatenation* is the act of fusing two datasets together either through rows (axis = 0) or through columns (axis = 1). Let’s do both of them:

```
# Reshaping the arrays so they become compatible in multidimensional
manipulation
first_array  = np.reshape(first_array, (-1, 1))
second_array = np.reshape(second_array, (-1, 1))

# Concatenating both arrays by columns
combined_array = np.concatenate((first_array, second_array), axis = 1)

# Concatenating both arrays by rows
combined_array = np.concatenate((first_array, second_array), axis = 0)
```

Now, let’s do the same thing for data frames. Consider the following data frames:

```
first_data_frame  = pd.DataFrame({'first_column'  : [ 1,  2,  3],
                                  'second_column' : [ 4,  5,  6]})
second_data_frame = pd.DataFrame({'first_column'  : [ 7,  8,  9],
                                  'second_column' : [10, 11, 12]})
```

Concatenation is useful when you want to combine data into one structure. This is how it can be done with data frames (notice how it’s simply a change of syntax and function source):

```
# Concatenating both data frames by columns
combined_data_frame = pd.concat([first_data_frame, second_data_frame],
axis = 1)

# Concatenating both data frames by rows
combined_data_frame = pd.concat([first_data_frame, second_data_frame],
axis = 0)
```

Remember that with time series, *rows* (horizontal cells) represent one time step (for example, hourly) with all the data inside while *columns* represent the different types of data (for example, open price and close price of a financial instrument). Now let's see slicing techniques for arrays:

```
# Defining a one-dimensional array
my_array = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

# Referring to the first value of the array
my_array[0] # Outputs 1

# Referring to the last value of the array
my_array[-1] # Outputs 10

# Referring to the fifth value of the array
my_array[6] # Outputs 7

# Referring to the first three values of the array
my_array[0:3] # Outputs array([1, 2, 3])
my_array[:3]  # Outputs array([1, 2, 3])

# Referring to the last three values of the array
my_array[-3:] # Outputs array([8, 9, 10])

# Referring to all the values as of the second value
my_array[1:] # Outputs array([2, 3, 4, 5, 6, 7, 8, 9, 10])

# Defining a multi-dimensional array
my_array = np.array([[ 1,  2,  3,  4,  5],
                     [ 6,  7,  8,  9, 10],
                     [11, 12, 13, 14, 15]])

# Referring to the first value and second column of the array
my_array[0, 1] # Outputs 2

# Referring to the last value and last column of the array
my_array[-1, -1] # Outputs 15

# Referring to the third value and second to last column of the array
my_array[2, -2] # Outputs 14

# Referring to the first three and fourth column values of the array
my_array[:, 2:4] # Outputs array([[3, 4], [8, 9], [13, 14]])

# Referring to the last two values and fifth column of the array
my_array[-2:, 4] # Outputs array([10, 15])

# Referring to all the values and all the columns up until the second
```

```
row
my_array[:2, ] # Outputs array([[ 1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

# Referring to the last row with all the columns
my_array[-1:, :] # Outputs array([[11, 12, 13, 14, 15]])
```

> **NOTE**
>
> It is important to know that Python indexing starts at zero. This means that to refer to the first element in a data structure, you refer to its index as index = 0. On another note, it is also worth noting that in ranges, the last element is excluded which means that the first three elements in a data structure are referred to as [0, 3] which will give the elements indexed at 0, 1, and 2.

Let's see the same thing for data frames so that this section becomes a sort of a mini encyclopedia whenever you want to manipulate data structures:

```
# Defining a one-dimensional data frame
my_df= pd.DataFrame({'first_column': [1, 2, 3, 4, 5,
                                      6, 7, 8, 9, 10]})

# Referring to the first value of the data frame
my_df.iloc[0]['first_column'] # Outputs 1

# Referring to the last value of the data frame
my_df.iloc[-1]['first_column'] # Outputs 10

# Referring to the fifth value of the data frame
my_df.iloc[6]['first_column'] # Outputs 7

# Referring to the first three values of the data frame
my_df.iloc[0:3]['first_column'] # Outputs ([1, 2, 3])

# Referring to the last three values of the data frame
my_df.iloc[-3:]['first_column'] # Outputs ([8, 9, 10])

# Referring to all the values as of the second value
my_df.iloc[1:]['first_column'] # Outputs ([2, 3, 4, 5, 6, 7, 8, 9,
10])

# Defining a multi-dimensional data frame
my_df  = pd.DataFrame({'first_column'  : [ 1,  6,  11],
                       'second_column' : [ 2,  7,  12],
                       'third_column'  : [ 3,  8,  13],

                       'fourth_column' : [ 4,  9,  14],
```

```
                        'fifth_column'  : [ 5,  10, 15]})

# Referring to the first value and second column of the data frame
my_df.iloc[0]['second_column'] # Outputs 2

# Referring to the last value and last column of the data frame
my_df.iloc[-1]['fifth_column'] # Outputs 15

# Referring to the third value and second to last column of the data
frame
my_df.iloc[2]['fourth_column'] # Outputs 14

# Referring to the first three and fourth column values of the data
frame
my_df.iloc[:][['third_column', 'fourth_column']]

# Referring to the last two values and fifth column of the data frame
my_df.iloc[-2:]['fifth_column'] # Outputs ([10, 15])

# Referring to all the values and all the columns up until the second
row
my_df.iloc[:2,] # Outputs ([[ 1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

# Referring to the last row with all the columns
my_df.iloc[-1:,]  # Outputs ([[11, 12, 13, 14, 15]])
```

**NOTE**

Try going back to the earlier chapters to execute the code given there. You should have a more solid understanding by now.

# Importing Financial Time Series in Python

This section presents a key part in deploying machine and deep learning algorithms. It deals with the historical OHLC data that is needed to run the models and evaluate their performance.

The first step is to prepare the environment and everything else necessary for the success of the algorithms. For this, you need two programs:

- A Python interpreter that you use to write and execute code. You have already completed this step.

- Charting and financial software that you use as a database. This part is covered in this section.

Throughout the book, I use *MetaTrader 5*, a benchmark charting program used by many traders around the globe. Follow these steps:

1. Download SPYDER and familiarize yourself with how it works.
2. Download the MetaTrader 5 software.
3. Use SPYDER to import historical prices from MetaTrader 5.

From the official website, download and install MetaTrader 5. You need to create a demo account, which is simply a virtual account with imaginary money. The word *demo* does not refer to a limited duration of use but to the fact that it is not using real money.

To open an account, select *File* > *Open an Account*, choose MetaQuotes Software Corp, and then click *Next*. Then, choose the first option to open a demo account; this will let you trade virtual money. Finally, enter some basic information such as name, email, and account type. You will not receive a verification request or any type of confirmation as the demo should launch directly, allowing you to see the charts.

Figure 6-3 shows the platform's interface. By default, MetaTrader 5 does not show all the markets it covers, so you need to make them accessible for import and visualization if necessary. Click *View*, click *Market Watch*, and then right-click any of the symbols shown in the new tab and choose *Show All*. This way you can see the extended list with more markets.


*Figure 6-3. MetaTrader's 5 interface*

Before proceeding to the coding part, you need to install the MetaTrader 5 Python integration library so you can use it later in Spyder. This is easy and requires one step. Open the Anaconda prompt and type:

```
pip install MetaTrader5
```

Installation is the bridge that allows you to use Python modules and functions designed for MetaTrader 5 in the interpreter.

The following code block uses the `import` built-in statement, which calls for internal (self-created) or external (created by third parties) libraries. A library is a store of functions, and thus, you need to import the libraries that are pertinent to

what you want to do. For demonstration purposes, import the following modules, packages, and libraries:

```
import datetime # Gives tools for manipulating dates and time
import pytz # Offers cross-platform time zone calculations
import MetaTrader5 as mt5 # Importing the software's library
import pandas as pd
import numpy as np
```

The next step is to create the universe of the time frames that you will be able to import. Even though I will be showing you how to analyze and back-test hourly data, you can define a wider universe, as shown in the following code snippet:

```
frame_M15 = mt5.TIMEFRAME_M15       # 15-minute time
frameframe_M30 = mt5.TIMEFRAME_M30 # 30-minute time frame
frame_H1 = mt5.TIMEFRAME_H1         # Hourly time frame
frame_H4 = mt5.TIMEFRAME_H4         # 4-hour time frame
frame_D1 = mt5.TIMEFRAME_D1         # Daily time frame
frame_W1 = mt5.TIMEFRAME_W1         # Weekly time frame
frame_M1 = mt5.TIMEFRAME_MN1        # Monthly time frame
```

> **NOTE**
>
> The full code is found in the GitHub repository under the name *Master_Function.py*

A *time frame* is the frequency with which you record the prices. With hourly data, you will record the last price printed every hour. This means that in a day, you can have up to 24 hourly prices. This allows you to see the intraday evolution of the price. The aim is to record the totality of the OHLC data within a specific period.

The following code defines the current time, which is used so that the algorithm has a reference point when importing the data. Basically, you are creating a variable that stores the current time and date:

```
now = datetime.datetime.now()
```

Let’s now proceed to defining the universe of the financial instruments you want to back-test. In this book, the back-tests will be done exclusively on the FX

market. Therefore, let's create a variable that stores some key currency pairs:

```
assets = ['EURUSD', 'USDCHF', 'GBPUSD', 'USDCAD']
```

Now that you have your time and asset variables ready, all you need is to create the structure of the importing algorithm. The `get_quotes()` function does this:

```
def get_quotes(time_frame, year = 2005, month = 1, day = 1,
               asset = "EURUSD"):

    if not mt5.initialize():

        print("initialize() failed, error code =", mt5.last_error())

        quit()

    timezone = pytz.timezone("Europe/Paris")

    time_from = datetime.datetime(year, month, day, tzinfo = timezone)

    time_to = datetime.datetime.now(timezone) +
datetime.timedelta(days=1)

    rates = mt5.copy_rates_range(asset, time_frame, time_from,
time_to)

    rates_frame = pd.DataFrame(rates)

    return rates_frame
```

Notice that in the `get_quotes()` function, you use the `pytz`and `pandas` libraries. The function starts by defining the *Olson* time zone, which you can set yourself. Here is a brief, nonexhaustive list of what you can enter depending on your time zone:

```
America/New_York
Europe/London
Europe/Paris
Asia/Tokyo
Australia/Sydney
```

Afterward, I define two variables called `time_from` and `time_to`:

- The `time_from` variable contains the datetime referring to the beginning of the import date (e.g., 01-01-2020).
- The `time_to` variable contains the datetime referring to the end of the import date which uses the `now` variable to represent the current time and date.

The next step is to create a variable that imports the financial data using the time periods you have specified. This is done through the `rates` variable using the `mt5.copy_rates_range()` function. Finally, using `pandas`, transform the data into a data frame. The final function required for the importing process is the `mass_import()` function. It lets you choose the time frame using the variable and then uses the `get_quotes()` function to import the data and format it to an array. The following code snippet defines the `mass_import()` function:

```
def mass_import(asset, time_frame):

    if time_frame == 'H1':
        data = get_quotes(frame_H1, 2013, 1, 1, asset = assets[asset])
        data = data.iloc[:, 1:5].values
        data = data.round(decimals = 5)

    return data
```

The `mass_import()` function automatically converts the data frame into an array, so you do not have to worry about conversion when using the automatic import.

**NOTE**

The algorithm imports a number of historical data limited by MetaTrader 5. Although that number is high, in time you may need to adjust the year argument higher in order to get the data. For instance, if you get an empty array using the `mass_import()` function, try putting a more recent year in the `get_quotes()` function ("2014" instead of "2013").

To import the historical hourly EURUSD data since beginning of 2014 to date, you may type the following (assuming `get_quotes()`, `now`, the frames, and

the libraries are already defined):

```
# Defining the universe of currency pairs
assets = ['EURUSD', 'USDCHF', 'GBPUSD', 'USDCAD']

# Re-defining the mass_import function to switch to a default 2014
def mass_import(asset, time_frame):

    if time_frame == 'H1':
        data = get_quotes(frame_H1, 2014, 1, 1, asset = assets[asset])
        data = data.iloc[:, 1:5].values
        data = data.round(decimals = 5)

# Calling the mass_import function and storing it into a variable
eurusd_data = mass_import(0, 'H1')
```

**NOTE**

Notice how the `return` statement is used in the `mass_import` function in order to store the historical data in chosen variables.

Even though there is a MAC version of MetaTrader 5, the Python library only works on Windows. It requires a Windows emulator on a Mac. For Mac users, you may want to try the manual import method.

Automatic import is a huge time saver but MAC users or even Windows users may run into frustrating errors. For this, I will show you the manual import way which you can use as a fix. In the GitHub link, you will find a folder called Historical Data. Inside the folder there is a selection of historical financial time series in the form of excel which you can download.

The manual way is to have an Excel file with OHLC data that you have downloaded from a third party (such as the excel files provided in the Github repository). In this case, you can use the `pandas` library to import it and transform it into an array.

Let's take an example of *eurusd_data*. Download and the file and store it on your desktop. You now have to make sure that the Spyder's directory is in the same place as the file (so, in the desktop). In layperson's terms, Spyder must search the desktop for the Excel file. To choose the right directory, you must

click the folder button next to the arrow, as shown in Figure 6-4:


*Figure 6-4. Directory tab*

You should get a separate window where you can choose the desktop location and then validate the choice. Having done this, the tab should look like Figure 6-5:


*Figure 6-5. Directory tab*

You must use the `read_excel()` function (built-in in `pandas` and accessible after importing it) to get the values inside the Excel file. Follow this syntax:

```
# Importing the excel file into the Python interpreter
my_data = pd.read_excel('eurusd_data.xlsx')
```

Right about now, you have a data frame called `eurusd_data` with four different columns representing open, high, low, and close prices. You generally have to enter the library's name before using a function that belongs to it; this is why `read_excel()` is preceded by `pd`.

> **NOTE**
>
> I recommend using the automatic way for Windows users and the manual way for macOS users due to compatibility issues.

You can get an array directly in one line by just adding `.values` to `pd.read_excel('eurusd_data.xlsx')`, thus becoming `pd.read_excel('my_data.xlsx').values` and resulting in an array instead of a data frame.

## Summary

Python, one of the stars of coding languages. It did enjoy and still enjoys a

widespread adoption by the developers' community. Mastering it is key to unlocking huge potential in the data science world.

The next chapter presents machine learning and different prediction algorithms. The main aim is to understand the intuition and be able to code the algorithms and run a back-test over financial data. You will see that once you start understanding the process, it becomes a matter of removing an algorithm an plugging another (in case they have the same assumptions). The warm-up chapters are over and it's time to start coding.

---

1 Imaginary numbers are a type of complex number that represent the square root of a negative number.

## About the Author

**Sofien Kaabar** is a financial author, trading consultant, and institutional market strategist specializing in the currencies market with a focus on technical and quantitative topics. Sofien's goal is to make technical analysis objective by incorporating clear conditions that can be analyzed and created with the use of technical indicators that rival existing ones.

Having elaborated many successful trading algorithms, Sofien is now sharing the knowledge he has acquired over the years to make it accessible to everyone.