ANALYSIS OF TOURISM DATA

PROF. DR. ROBERT KELLER*

* *Research Professor for Information Management and Digitalization at the Faculty for Tourism Management at the University of Applied Sciences Kempten, Germany, ORCID: 0000-0001-7097-1724*

Analysis of Tourism Data
First Edition

You can find the code at `https://github.com/robberdk/AnalysisOfTourismData`.

Please cite this work as follows:
Keller, R. (2025) Analysis of Tourism Data. 1st Edition. Kempten. DOI: 10.60785/opus-2954

I used Gemini, ChatGPT, and Mistral to improve the quality of the language of the book.

CONTENTS

LEARNING OBJECTIVES OF THE BOOK

In an era where technology drives innovation and shapes the world around us, the ability to think critically and solve problems has never been more essential. Programming serves as a powerful tool to develop these skills, and Python, with its simplicity and versatility, is the perfect language to get started.

This book is designed as a practical guide towards the general language of digitalization, focusing on structured problem-solving through Python programming. It combines clear explanations, real-world examples, and hands-on exercises to help readers develop the analytical mindset required to tackle complex challenges.

The concepts learned can also be easily transferred by the students to other application areas. Furthermore, the course conveys ...

- ...the foundations of python programming
- ...basic concepts of the development of data management, data analysis
- ...solution-oriented, structured approaches to professional questions

Let's embark on this journey to unravel the art of problem-solving and discover how Python can transform your ideas into solutions!

1 FOUNDATIONS

1.1 Installation Instructions

There are two recommended ways to use Python and Jupyter effectively in this course:

1. Google Colab - this requires a Google account and an internet connection, but no installation
2. Anaconda - a local installation of the programs, but significantly more complicated to use

COLAB: You can use Jupyter Notebook online without installation. This is also the only way you can use Jupyter on a tablet. However, you will need to accept Google's privacy policy. In this course, however, we will not be processing any personal data.

Below are links to the corresponding applications and instructions: `https://colab.research.google.com/`

After clicking the link, you can simply create a new document. The data will be automatically saved to your Google Drive.

ANACONDA: Anaconda is a platform that provides easy access to programming and all the required programs. However, Anaconda is also the tool that "professionals" use, so it may seem a bit complex and overwhelming at first.

Below are links to the corresponding applications and instructions: `https://www.anaconda.com/products/individual`

After installation, you can launch the program "Jupyter Notebook." Your browser will open. Then, you can navigate through your local folder structure in the browser and find a suitable folder for your documents. After that, you can click on "New" and "Python 3" to create a new window.

In this book, we use the programming language Python and gain experience with the programming environment Jupyter. The tasks can also be completed without your own notebook, but this is not recommended. On tablets, entering special characters is much more complicated, and on paper, you miss the automatic error checking.

1.2 Variables, Operators, and Data Types

1.2.1 A Few Basic Terms Upfront

In this course, we begin with our first small program. However, before that, we need to establish an important rule:

No excessive respect or fear of the topic "Programming"!

At first, "Programming" may sound like a completely different world about which one knows nothing. But that is not true. In this course, we start from ZERO. All you need is a little logical thinking!

What does this mean specifically?

- We need to abstract the real-world problem to make it understandable for the programming language.
- We must solve the problems in such a way that the solution is reproducible.
- We need to learn the basic commands of programming languages.
- This knowledge can then also be transferred to other business science applications with programming languages like Java, R, VBA (e.g., for Excel), ...

Before we begin, let's briefly introduce our tools:

PYTHON: Python is a general-purpose, typically interpreted, high-level programming language aimed at readable and concise programming style. A programming language is a formal language for formulating data structures and algorithms, i.e., computational rules that can be executed by a computer. Python is one of the most commonly used programming languages, it is free, open-source, and has over 200,000 add-on packages.

JUPYTER NOTEBOOK AS AN IDE: Jupyter Notebook is an online development environment offered by the open-source project Jupyter, providing a graphical user interface for all programming languages in the field of data science. An integrated development environment (IDE) is a collection of computer programs designed to handle the tasks of software development with minimal media breaks.

SPECIAL CHARACTERS Please remark, that you must use several characters, especially brackets, e.g. (), [], or {}. Please make sure that you know how to use them.

Jupyter Notebook is particularly suitable for displaying both code and results on the same page. For this, you can now use two different types of elements:

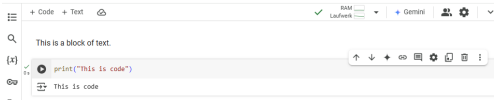

Figure 1: Screenshot of the Jupyter IDE

- On the one hand, you can insert **text blocks** to add descriptions and structure your notebook.
- On the other hand, you can insert **code blocks** that contain the code. When you execute these code blocks, the results will be displayed afterward.

1.2.2 Our First Program

Start your Jupyter Notebook and enter the following:

```
3+4
```

When you click on "Run," your IDE will display the correct result: 7.

This also works with other small calculations. The IDE even handles the order of operations.

```
3+4/2
```

As expected, the IDE now shows the correct result: 5.

With the "#" symbol, you can add **comments** that the program will not execute. This symbol is used in Python to add comments to the code, which help other programmers (and yourself) understand the code. This is especially important when writing a more complex program that you want to understand later.

```
#COMMENT
```

Here, nothing happens. Python recognizes the comment and skips that line of code.

PYTHON HACK #1.1: When you press "Ctrl" + "Enter" (Windows) or "Command" + "Enter" (Mac), the selected code block is executed.

Python can also **print text**. Using the keyword "print()", the program recognizes that something should be output. The text to be printed is written in quotation marks between the parentheses.

```
print("Hello World!")
```

1.2.3 Variables

Frequently used information can be stored in so-called **variables**. A variable is a single word without special characters. Only "_" is allowed.

Assigning information to a variable is done with the "=" sign. Variables and their contents always follow the form "Variable Name" "=" "Content of the Variable."

```
a = "Hello"
pi = 3.14

print(a)
print(pi)
```

In this example, the information "Hello" is stored in the variable "a". The value "3.14" is stored in the variable "pi". The contents of the variables can be printed using the Print statement. Note that the variable in the Print statement does not have quotation marks.

The information in the variables is stored in a specific form, such as text or numbers. We will discuss this in more detail later.

PYTHON HACK #1.2: The Python programming language follows the English number notation. Floating-point numbers are separated with a ".".

1.2.4 Arithmetic Operators

In the previous examples, we already performed some basic arithmetic operations without explicitly discussing them. In Python, you can perform all conceivable

mathematical operations. There are also various constants that are already built into the language.

To call certain operators, you need to import a package into a code block beforehand. We will discuss what this means in more detail later. The following table introduces some operators:

Table 1: Arithmetic Operators

Command	Description
+, -, *,	Addition, Subtraction, Multiplication, Division
** , math.sqrt()	Power, Square Root
abs(), math.log(), round()	Absolute Value, Natural Logarithm, Round
math.pi, math.inf, -math.inf	Pi, Infinity, Negative Infinity

EXERCISE 1.1 Please try to execute the following calculations in Python:

1. 563476 + 7493176
2. 543.4 − 47.34
3. 3076 ÷ 12
4. Round the number 3.14 to the nearest whole number
5. Find the square root of 144

1.2.5 Data Types

There are several ways to differentiate between various types of data. An obvious distinction is between numbers and letters. However, there are many other data types used by Python. Below are a few commonly encountered examples:

Table 2: Common Data Types

Data Type	Description
int	Integer numbers
float	Floating-point numbers
string	Strings (Text)
boolean	Boolean values (True, False)
list	List of elements of a data type
dictionaries	A key is associated with one or more elements

You can find out which variable type applies to a specific variable. You can also convert between different types of variables, such as from text to numbers or vice versa.

```
type(3)

type(a)

type("a")

str(5)

int("3")
```

EXERCISE 1.2: Please try to execute the following calculations in Python:

1. Assign the value 85 to a variable "k"
2. Multiply "k" by 2, then subtract 5, and store the result in the variable "p"
3. Print the variable "p"
4. What data type does the variable "p" have?
5. What value does the variable "u" have if you execute the following operations and assignments: a=1, b=2, c=3, d=b, p=a. p+d+a=u

Finally, regarding data types, we now turn to the topic of Boolean values. These are important for executing program logic, for example, to determine whether something is true or false. To do this, we often need to compare values or statements, such as "Is it raining right now?". A valid answer to the question can only be "True" or "False". Additionally, we can later filter data in lists using these values, such as "All people who have made annual sales of over 500 euros in our webshop".

To check such comparisons or statements, we need some operators, which we can use for this purpose:

Table 3: Logical Operators

Operator	Description
>, >=, <, <=	greater, greater than or equal to, less than, less than or equal to
==, !=	equal (2x= to avoid confusion with assignment), not equal
not	Negation (logical NOT)
and, or	logical AND, logical OR
true, false	TRUE, FALSE (central result of a comparison)

Using these operators, we can now make various comparisons:

```
print(3>=4) #False

print((3<2) or (4==(3+1))) #True

print(True and False) #False
```

The results of Statement 1 and Statement 2 are logical. But what happened in the third one? In Python, "false" is equivalent to 0 and "true" is equivalent to 1. Accordingly, you can perform arithmetic operations with Boolean values. For example, you can check how often a certain condition has been met. For instance, how many times the last name == "Müller" appears in the address database.

Now, it is important to understand how the combination of true and false behaves with OR and AND. This is not immediately intuitive.

Table 4: OR-Operation

Input 1	Input 2	Result
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

In the case of OR, the statement is always true if one element is true. Only in the case of FALSE OR FALSE is the statement false. In the case of AND, the statement is always true when the combined elements match (TRUE AND TRUE, FALSE AND FALSE) and false when they contradict each other.

Table 5: AND-Operation

Input 1	Input 2	Result
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

PYTHON HACK #1.3: As in mathematics, the placement of parentheses is crucial. You can try this out yourself with a few simple examples.

EXERCISE 1.3: Please try to execute the following calculations in Python:

1. Assign the value 65 to a variable "k"
2. Check if (k7/3) is greater than (k8/5).
3. Is (TRUE AND TRUE) OR (FALSE OR FALSE) true?
4. What happens if you leave out the parentheses?

1.2.6 The Three Sisters

The three sisters Clara, Hanna, and Anna have taken over their family's vineyard. In addition to the traditional wine business, the sisters also sell beauty and care products. Furthermore, they have established a hotel on the vineyard and regularly host wine tastings and similar events.

Clara is responsible for marketing and sales of the produced goods. Hanna oversees production, and Anna is in charge of the hotel and events.

The sisters recognized early on that, in such a complex business as theirs, information is the key to success. They want to use Python and Jupyter Notebook to make the most out of their data!

EXERCISE THREE SISTERS 1.4: Clara would like to store the prices of the various products in variables with the same names. Currently, the sisters offer the following products:

- Wine "Sorelle Rosso" 8.90 €
- Wine "Sorelle Bianco" 7.50 €
- Soap "Sapone Allgäu" 4.50 €
- Cream "Crema Allgäu" 6.35 €

Additionally, shipping costs of 4.50 € for customers from Germany and 6.50 € for customers from Austria, Switzerland, and Italy should be recorded in a variable.

PYTHON HACK #1.4: Come up with shorter yet descriptive variable names. Keep in mind the rules for naming variables.

PYTHON HACK #1.5: If you want to save your current work, click on "File" in the top left and select "Save as". You can then give the notebook a name. It will be saved as *.ipynb, where the * represents the name you assign.

EXERCISE THREE SISTERS 1.5: A regular customer and family friend named Leo, from the same village, orders 4 bottles of Bianco, 6 bottles of Rosso, and tries one of each of the beauty products, just like every year at this time.

A new customer from Switzerland, who happened to see the advertisement on the sisters' social media channels, also tries all the products except the soap.

How much revenue can the sisters expect from both customers? Please note that shipping costs are not part of the revenue!

1.2.7 Solutions

SOLUTIONS TO EXERCISE 1.1: Below you will find the solution to the exercises:

```
import math #important for sqrt function

563476 + 7493176

543.4 - 47.34 #US number notation

3076/12

round(3.14)

math.sqrt(144)
```

SOLUTIONS TO EXERCISE 1.2: Below you will find the solution to the exercise:

```
k = 85

p = k*2 - 5

print(p)

type(p)

a=1
b=2
c=3
d=b
p=a
u=p+d+a

print(u)
```

SOLUTIONS TO EXERCISE 1.3: Below you will find the solution to the exercise:

```
k = 65

print((k*7/3)>(k*8/5))

print((True and True) or (False or False))

print(True and True or False or False)
```

SOLUTION TO EXERCISE 1.4: Below you will find the solution to the exercise:

```
WR = 8.90 #Sorelle Rosso
WB = 7.50 #Sorelle Bianco
SA = 4.50 #Sapone Allgaeu
CA = 6.35 #Crema Allgaeu

VD = 4.50 #Shipping Germany
VA = 6.50 #Shipping Abroad
```

SOLUTION TO EXERCISE 1.5: Below you will find the solution to the exercise:

```
Leo = 4 * WB+6 * WR+SA+CA
print(Leo)

NK1 = WB+WR+CA
print(NK1)

Revenue = NK1 + Leo
print(Revenue)
```

Thank you for working through the first section. Did you think you could learn so much about programming so quickly?

1.3 Conditions

1.3.1 Decisions

In everyday life, we often face decisions. Do we want pizza or pasta tonight? Or would we prefer salad instead? This decision depends on many influencing factors. How hungry are we? How much money do we have available? ...

We can break down complex questions into simpler ones that are easy to answer and have a finite set of possible answers. The complex question "What should I wear today?" can be simplified through many smaller questions. Is it warm? Is it raining? Do I need to exercise? Business or casual? ...

We can formally represent these questions, and thus the underlying decision, using a **decision tree**. A decision tree consists of nodes and edges. In **nodes**, in the chosen representation, these are boxes, the specific question is asked. The **edges**, i.e., the lines between the nodes, represent the different possible answers. At the end, the final decision answer is presented.

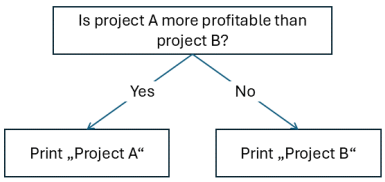

Figure 2: A simple decision tree

In the depicted example, the question is whether Project A is more profitable than Project B. If Project A is more profitable, "Project A!" will be displayed on the screen. Otherwise, "Project B!" will be displayed.

1.3.2 IF/ELSE Statements

Such decisions can be represented in Python using IF ... ELSE statements. Since Python, as seen with floating-point numbers, is influenced by English, this is also reflected in conditions. Translated, IF means "if" and ELSE means "else". With this, we can now, for example, represent the following: If I am very hungry, I want pizza, otherwise, I will eat salad.

If the condition behind the IF statement is true, the application block is executed; if it is false, the content of the code block after the ELSE is executed.

```
hunger = False

if hunger == True:
   print("Eat pizza!")
else:
   print("Eat salad!")
```

Such decisions can also be nested. In the following, "..." represents further code, and CONDITION represents any condition/question.

```
hunger = False

if CONDITION:
   if CONDITION:
      ...
   else:
      ...
```

```
else:
   if CONDITION:
      ...
   else:
      ...
```

PYTHON HACK #1.6: In Python, programming blocks are identified by indentation (other languages often use brackets).

However, an ELSE doesn't always have to follow an IF if there is no alternative case. For example, customers can choose whether they want to receive a confirmation email or not.

```
bestaetigungsmail = False

if bestaetigungsmail == True:
   print("Send an email!")

print("Execute order.")
```

PYTHON HACK #1.7: The syntax of programming languages is like the grammar of a language. In a language, there are fixed rules for when to place a comma or which words should follow each other. The same applies to programming languages. For example, indentations or line breaks are crucial.

EXERCISE 1.6 The controlling department of your company has developed the cost function for the current variable and fixed costs. From a market study, you also learned the costs of a competitor. Compare your costs (5*279+3500.8) with those of the competitor (4*408+3034.4) and display the company with the higher costs on the screen. Use the branching construct presented in this section. Implement the given scenario in Jupyter Notebook with Python.

EXERCISE 1.7 How can the project with different profit values be evaluated using a nested condition? Try it out yourself. In this task, assume a profit of 211. Assume that an "excellent project" generates a profit of 500. A "very good project" must generate at least 250, and a "good project" must generate at least 0. Projects that generate less are "not profitable".

1.3.3 The Three Sisters

After Clara, Hanna, and Anna celebrated their first successes with Python, they are already facing the next challenge: With the profits from the last season, they plan to make a new investment. Hanna, who is responsible for production, wants to invest in a new wine press and believes this will help her get more yield from the same amount of grapes. Anna, on the other hand, believes that a sauna in the basement of the winery hotel would attract wealthier guests. Clara understands the suggestions but would rather hire an influencer for a small advertising campaign. How should the three sisters decide?

With what we've learned, we should be able to help the sisters decide between the investments, right?

EXERCISE THREE SISTERS 1.8: The wine press has a one-time price of 3000 euros but should help generate 487 euros in profit each year for the next 20 years. For the sauna, they can charge an additional 30 euros per guest and visit, but there are maintenance costs of 1 euro per guest and visit. The sauna costs 30,000 euros,

including installation, and can probably be used for 4 years. The sisters expect 400 guests per year. The influencer charges 9,430 euros plus expenses of 645 euros per day he spends at the winery. He stays for 4 days. Clara expects an increase in profit of 3,333 euros over the next 4 months.

Should the three sisters invest in the wine press, the sauna, or the influencer?

1.3.4 Solutions

SOLUTION TO EXERCISE 1.6 Below is the solution to the exercise:

```
eigenkosten = 5*279+3500.8
fremdkosten = 4*408+3034.4

if (eigenkosten > fremdkosten):
   print("Our company has higher costs")
else:
   print("The competitor has higher costs!")
```

SOLUTION TO EXERCISE 1.7 Below is the solution to the exercise:

```
Ga1 = 211
if(Ga1>500):
   print("Excellent project")
else:
   if(Ga1>250):
      print("Very good project")
   else:
      if(Ga1>0):
         print("Good project")
      else:
         if(Ga1<0):
            print("Unprofitable project")
         else:
            print("The input is not a numeric value")
```

SOLUTION TO EXERCISE 1.8 Below is the solution to the exercise:

```
GewinnWeinpresse = -3000+20*487
GewinnSauna = -30000+4*400*(30-1)
GewinnInfluencer = -(9430+4*645)+4*3333

if (GewinnWeinpresse > GewinnSauna):
   if (GewinnWeinpresse > GewinnInfluencer):
      print("Investment should be made in the wine press")
else:
   if (GewinnSauna > GewinnInfluencer):
      print("Investment should be made in the sauna")
   else:
      print("Investment should be made in the influencer")
```

1.4 Simple Data Structures

1.4.1 Basics

In this course, you will learn how to store multiple different elements in a list. For this, there are "Lists" and "Numpy Arrays." In this course, we will focus on the latter! Numpy **Arrays** can hold multiple **elements** and enable additional functions such as mathematical operations. Numpy arrays thus form the foundation for data analysis.

Below are some important properties of Numpy Arrays:

- Numpy Arrays have a fixed size
- Numpy Arrays enable fast and efficient mathematical operations
- To perform calculations the Numpy Array must contain elements of the same data type
- Arithmetic operations are applied to the entire Numpy Array

Numpy Arrays can thus be considered as vectors or matrices. It is important to note that we need to import numpy as a package. There are also different ways to create Numpy Arrays. For example, values can be entered directly (see a1), or defined through a range (a2 or a3). Another possibility is creating a Numpy Array filled with zeros. The Numpy Arrays can easily be printed using the print() function.

```
import numpy as np  # Import the package

a1 = np.array([1,2,3]) # Create a Numpy Array with specific values
             # When entering values, additional square brackets are required
print(a1)

a2 = np.arange(1,10) # Create a Numpy Array from 1 to 9 (exclusive of 10)
print(a2)

a3 = np.arange(1,11,2) # As before, with a step size of 2
print(a3)

a4 = np.zeros(4)  # Create a Numpy Array with 4 zeros
print(a4)
```

In addition to printing the entire Numpy Array, you can also access individual values or ranges. You can either access a single element or an entire range of the Numpy Array.

It is important to note that when selecting elements, counting always starts at 0. This is called the **index**. The first value of the Numpy Array has index 0, the second value has index 1, and so on. The n-th value has index n-1.

Table 6: Index and value using the previous example

Index	0	1	2
Value	1	2	3

When outputting a range, the index is used again. However, it is important to note that 1:4 refers to indices 1 to 3, inclusive. Mathematically, this is expressed as $[1:4[$, meaning the set from 1 up to, but not including, 4.

```
print(a2[2])    # Access the element with index 2
print(a2[1:4])   # Access the elements with index 1 up to and including 3
```

In addition to printing values, they can also be modified. Using the assignment operator =, a value at a specific index or over a range of indices can be overwritten.

```
a2[1] = 50
a2[3:5] = 100

print(a2)
```

1.4.2 Calculations with Numpy Arrays

Various calculations can be performed with Numpy Arrays. They can be added, subtracted, and multiplied. They can also be calculated with a scalar. The best way to understand this is through an example.

```
import numpy as np

a1 = np.arange(1,10)   # Numpy array from 1 to 9
a2 = np.arange(1,10)
a3 = a1 + a2      # Element-wise addition of the two arrays
print(a3)
print(a1*3)       # Element-wise multiplication with a scalar
```

The following table explains the arithmetic operators for working with Numpy Arrays as vectors in more detail.

Table 7: Calculations with Numpy Arrays as Vectors

Operator	Description
arr + arr	Addition of two arrays
arr - arr	Subtraction of two arrays
arr * arr	Multiplication of two arrays
arr / arr	Division of two arrays
1/arr arr*3	Calculation with a scalar
arr**3	Exponentiation
np.sqrt(arr)	Square root
np.exp(arr)	Exponential value
np.max(arr)	Maximum value of the array
arr.size	Length of the array

EXERCISE 1.9 Create a Numpy array with only even numbers from 2 to 22 without manually entering the numbers and then output it.

PYTHON HACK #1.8: Coding conventions are the etiquette of programming. They are guidelines for the programmability that...

- ... encourage programmers to write clean and readable source code
- ... make it easier to debug the source code later, both for yourself and others
- An example of this is clearly naming variables so that you can still understand them throughout the course of the project.

One can also perform matrix calculations with Numpy. You may have various, possibly positive, possibly negative memories from school. But we will definitely repeat the basics once more.

A matrix is a rectangular arrangement of numbers/values arranged in rows and columns. The dimension or order of a matrix describes how many rows n and columns k there are.

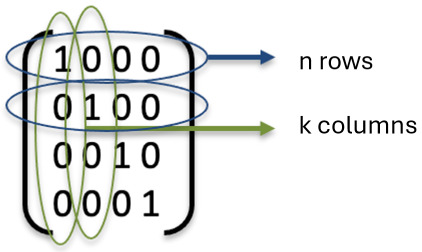

Figure 3: Rows and columns of a matrix

When working with matrices, you must be careful about the following. Only matrices of the same order can be added or subtracted. For two matrices A and B, the product $C = AB$ is defined only if the number of columns of A matches the number of rows of B: $C = AB$ is not the same as $D = BA$!

Matrix A: $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$

Matrix B: $\begin{pmatrix} 1 & 1 \\ 2 & 3 \\ 3 & 0 \\ 0 & 3 \end{pmatrix}$

The number of columns in A (2) and the number of rows in B (2) match. Therefore, the matrices can be multiplied, resulting in a 2x2 matrix. The result of the matrix multiplication is C: $A * B = C$

Matrix C: $\begin{pmatrix} 1 & 1 \\ 2 & 3 \end{pmatrix}$

In Python, a matrix is essentially several combined Numpy arrays. If you want to manually input the numbers, they are inserted row by row in square brackets. Important: the entire rows are enclosed again in square brackets at the end. The familiar automated functions also work with matrices. Below are two code examples:

```
matrix1 = np.array ([[1,2,3],[4,5,6]]) # Matrix based on 2 arrays
matrix2 = np.zeros((2,2))    # Create a 2x2 matrix with zeros

print(matrix1)
print(matrix2)
```

You can also output individual elements of a matrix again. In this case, the first value in the square brackets is the index of the row (the index starts at 0!). The second value corresponds to the index of the column. If you write a ":" instead of a number in the index, the entire content, for example, all the elements of the rows or columns, will be output.

```
print(matrix1[0,1])  # Access an element: Row index 0, Column index 1

print(matrix1[0,:])  # Output only the row with index 0, all columns
print(matrix1[:,0])  # Output only the column with index 0, all rows
```

Changing values or performing operations with matrices works analogously to operations with vectors.

```
matrix1[0,1]=10   # Replace an element with a new value
print(matrix1.sum()) # Output the sum of the matrix
```

EXERCISE 1.10 Create a matrix Q.

Matrix Q: $\begin{pmatrix} 35 & 12 & 48 & 66 \\ 43 & 88 & 46 & 44 \\ 22 & 13 & 46 & 99 \\ 74 & 67 & 66 & 91 \end{pmatrix}$

Now perform the following operations:

1. Output the 3rd row
2. Output the 2nd column
3. Replace the number "67" with "40" and output the last row
4. What is the result of Q[1,2] – Q [3,2] ?

1.4.3 Excursus: Packages

Python packages contain various functions and sometimes pre-made datasets. They thus expand the range of standard functions that Python offers. You have already used a few different packages, such as "math" and "numpy," without even realizing it.

Before using the packages, you first need to download and install them. Luckily, in Colab, the important Python packages are automatically installed. To use a Python package, only the following step is necessary:

Activate it once at the beginning of the code with:

```
import package-name as package-abbreviation
```

For simple package names, you can import without abbreviation:

```
import package-name
```

Here are a few Python packages that cover the most commonly used functions, with a brief list and description:

```
import numpy as np     # Tool for vectors, matrices, and mathematical operations
import pandas as pd    # Tool for managing data in, e.g., DataFrames
import matplotlib.pyplot as plt # Tool for creating simple graphics
import seaborn as sns    # Tool for complex graphics and distributions
import scikit-learn    # Tool for machine learning
...
```

The import of packages often takes place together at the beginning of the code. However, you can also import just before using the imported function. For space and clarity reasons, we will not always mention the import of packages. Please import the required packages on your own.

Sometimes, packages are not installed in your environment. In this case, you get an error message at your import statement. You can solve this issue by installing the package with a simple command:

```
pip install PACKAGENAME
```

If you want to install the package numpy, you can write:

```
pip install numpy
```

Your computer will answer, that the package is already installed.

1.4.4 The Three Sisters

Clara, Hanna, and Anna are excited about how well they can work with Python. They can not only perform calculations but also model complex decision processes using IF-statements and automate them!

This saves the three sisters a lot of work. To make better use of these possibilities, they are wondering if they can also store numbers or word sequences in Python without always typing them individually.

Now, the three sisters can take a look at the orders from the previous sections:

- A regular customer and family friend named Leo from the same village orders, as he does every year at this time, 4 bottles of Bianco, 6 bottles of Rosso, and tries two of the care products each.
- A new customer from Switzerland, who happened to see the advertisement on the sisters' social media channels, tries 2 bottles of each wine and 3 cans of cream.

Here are the prices of the individual products again:

- Wine "Sorelle Rosso" 8.90 €
- Wine "Sorelle Bianco" 7.50 €
- Soap "Sapone Allgäu" 4.50 €
- Cream "Crema Allgäu" 6.35 €

EXERCISE THREE SISTERS 1.11 Can the price lists be recorded together in a matrix?

$$\begin{pmatrix} \texttt{SorelleRosso} & \texttt{SaponeAllgaeu} \\ \texttt{SorelleBianco} & \texttt{CremaAllgaeu} \end{pmatrix}$$

Can the individual and total revenues of all customers per product be represented as a matrix?

1.4.5 Solutions

SOLUTION TO EXERCISE 1.9 Below is the solution to the exercise:

```
array = np.arange(2,23,2)
print(array)
```

SOLUTION TO EXERCISE 1.10 Below is the solution to the exercise:

```
Q = np.array([[35, 12, 48, 66],[43, 88, 46, 44],[22, 13, 46, 99],[74, 67, 66,
    91]])

print(Q[2,:])

print(Q[:,1])


Q[3,1]=40
print(Q[3,])

Q[1,2] - Q [3,2]
```

SOLUTION TO EXERCISE 1.11 Below is the solution to the exercise:

```
# In the first column of the price matrix pm are the wines, and in the second
    column are the care products.

pm = np.array([[8.9, 4.5],[7.5, 6.35]])
print(pm)

# Below are several solution paths. Here are two examples.
# Example 1

Leo_m = np.array([[0.0, 0.0],[0.0, 0.0]])
Leo_m[0,0] = pm[0,0]*6
Leo_m[0,1] = pm[0,1]*2
Leo_m[1,0] = pm[1,0]*4
Leo_m[1,1] = pm[1,1]*2

print(Leo_m)

ch_m = np.zeros((2,2))
ch_m[0,0] = pm[0,0]*2
ch_m[0,1] = pm[0,1]*0
ch_m[1,0] = pm[1,0]*2
ch_m[1,1] = pm[1,1]*3

print(ch_m)

gesamt_m = Leo_m + ch_m
print(gesamt_m)

print(gesamt_m.sum())



# Example 2

leo_matrix = np.array([[6, 2],[4, 2]])
```

```
leo_matrix = leo_matrix * pm
ch_matrix = np.array([[2, 0],[2, 3]])
ch_matrix = ch_matrix * pm

gesamt_matrix = leo_matrix + ch_matrix
print(gesamt_matrix)
```

1.5 Loops

1.5.1 Preliminary Considerations on Loops

In this section, we will explore how repetitive processes can be represented in code. One example of this is depreciation. Below is a brief excursion on this topic.

EXCURSION ON DEPRECIATION: Depreciation allows companies to write down the remaining book value of their assets. There are two different types of depreciation: geometric (or declining balance) and linear depreciation.

Geometric depreciation is now only possible in special cases. A specific percentage is deducted from the remaining book value.

Table 8: Geometric Depreciation

Period	1	2	3	4
Book Value	3000	1800	1080	...
Depreciation	3000*0.4=1200	1800*0.4=720	...	...
Remaining Book Value	1800	1080	...	...

In linear depreciation, a fixed amount is deducted each period.

Table 9: Linear Depreciation

Period	1	2	3	4
Book Value	3000	2700	2400	...
Depreciation	3000*0.1=300	3000*0.1=300	...	...
Remaining Book Value	2700	2400	...	...

To represent such depreciation in code, we need a construct that can model these repetitive calculations. In each iteration, the values from the previous iteration must be known in order to continue working with them. Additionally, the repetition must stop when a certain condition is met, such as a specific remaining book value or a specific number of iterations.

In this section, you will learn about two types of loops:

- for-loop, also called a counting loop, for a known number of iterations
- while-loop for programs with an unknown number of iterations

Both types of loops follow a general structure:

1. Check a condition before each loop iteration
2. If the result is "True": Execute the block of statements
3. If the result is "False": End the loop

1.5.2 for-loops

The first loop we will look at more closely is the for-loop. Let's first look at the syntax.

```
for i in range(1,3):
  STATEMENTS
```

Here, we start with the keyword "for." The loop can be read naturally as: "For the variable i in the range from 1 to exclusive 3." After the "for," a variable is defined, which is then incremented within a specified range. Important: As with other number ranges, the last number is exclusive, so in this case, $[1,3[$. For each increment, the loop's statement block is executed once.

Indented statements follow, which are repeated for each iteration. Once the indentation is finished, the program continues with the normal code that is not repeated.

Here is an example, where we first initialize a loop that runs twice (i = 1, 2). In the statement block, the value of i is printed. Afterward, the loop ends, and the normal code continues.

```
for i in range(1,3): # Initialize the for-loop
  print(i)   # Loop content
print("Now the code continues after the for-loop!")
```

PYTHON HACK #1.9: Variables can be initialized outside of the loop and modified inside the loop.

In the following example, a variable k is defined first. This variable is then used in the following for-loop for calculations. What's interesting is the order of the print() output and the calculation $k = k - 50$. Since the print() is executed first and the calculation happens afterward, k will have a lower value after the loop than was printed during the loop. The iteration of the loop is displayed with `print(i)`.

```
k = 200

for i in range(0,3):
  print(i)
  print(k)
  k = k - 50

print("The value of k after the loop:")
print(k)
```

For-loops are particularly useful when working with Numpy arrays. These arrays have a defined length. If we want to process each element, we can easily achieve this with a for-loop.

PYTHON HACK #1.10: Using NAME.size, we can print the size of an array.

The following example shows how this can be used. First, a Numpy array is created. The length of the array is then calculated with x.size. We can use this value as the end value for the "range" of the array. Important: Since this number is not considered in the loop, we must start counting from 0 or count up to $x.size + 1$. Then we will print the squared values of x. WARNING: The values will only be squared in the output. The array values themselves do not change!

```
import numpy as np

x = np.arange(0,11, 2)
print(x)
print(x.size)

for i in range(0, (x.size)):
  print(x[i]**2)

print(x)
```

EXERCISE 1.12 Theo has just started his tourism studies at the University of Kempten. Unfortunately, Theo does not have a usable laptop and has to make do with his sister's old device. Theo plans to buy a better one before the next semester starts. He has calculated that he can save around 135 € each month, and there are still 5 months until the new semester. Can you use a for-loop to calculate how much money Theo will save up?

In the following example, a variable k is defined first. This variable is then used in the following for-loop for calculations. What's interesting is the order of the print() output and the calculation $k = k - 50$. Since the print() is executed first and the calculation happens afterward, k will have a lower value after the loop than was printed during the loop. The iteration of the loop is displayed with `print(i)`.

```
k = 200

for i in range(0,3):
  print(i)
  print(k)
  k = k - 50

print("The value of k after the loop:")
print(k)
```

1.5.3 *while-Loops*

The second loop is the while-loop. Again, let's take a look at the syntax.

```
VALUE_FOR_CONDITION

while(CONDITION == True):
  STATEMENTS
```

At the beginning, we have the keyword "while". The loop can be read naturally as: "While the condition is true, the statement block is executed." The condition is checked before the statements are executed. If the condition is true, one iteration of the statement block begins. It is important to note that the while-loop does not contain variable declaration. The variable must be initialized before the loop, otherwise the value will be reset every time the loop runs. Also, the condition being checked must eventually become false. To achieve this, there must be a way to change the condition within the statement block. For example, the condition $i < 5$ can be changed by setting a variable i=0 before the loop, and in the statement block, it is incremented by 2 ($i = i + 2$). The loop will continue to restart until the condition is no longer true.

Another application example:

```
i = 0      #We set the initial value to 0

while (i < 5):   #Exit the while-loop once i > 5
  i = i + 1   #In each iteration, we increase i by 1
  print(i)

print("Now the loop is finished!")
```

While-loops are used whenever the exit condition is more complex, or when the number of iterations is unknown. For example, in complex calculations.

```
bool = True
value = 2

print("All powers of 2 up to the number 100")

while (bool == True):
  print(value)
  value = value**2
  if value > 100:
    bool = False
```

```
print("The first power of 2 over 100 is:")
print(value)
```

1.5.4 The Three Sisters

Now that the three sisters Clara, Hanna, and Anna can efficiently store customer, product, and shopping cart information, they can turn their attention to some bookkeeping tasks. Our new acquisitions, the wine press and the sauna, now need to be depreciated year by year. Perhaps Python can help us with this!

EXERCISE THREE SISTERS 1.13: THE WINE PRESS Fortunately, depreciating the wine press is easy. It cost €3000 and can be depreciated by 10% annually. The sisters quickly realize that it will take 10 years to fully depreciate the wine press. Please implement this depreciation in a Python program using a for-loop.

EXERCISE THREE SISTERS 1.14: THE SAUNA To apply the declining balance depreciation, we use the investment in the sauna, which has acquisition costs of €30,000. The depreciation rate is 40%. Note: Once the book value reaches €25, the asset can be fully depreciated immediately.

1.5.5 Solutions

SOLUTION TO EXERCISE 1.12 Below is the solution to the exercise:

```
er = 0        #the savings start at 0
sb = 135    #the monthly savings amount is 135

for i in range(0,5):
  er= er + sb
  print(er)
```

SOLUTION TO EXERCISE 1.13 Below is the solution to the exercise:

```
bw = 3000  #book value
ab = 3000*0.1 #depreciation amount

for i in range(0,10):
  bw = bw - ab
  print(bw)
```

SOLUTION TO EXERCISE 1.14 Below is the solution to the exercise:

```
bw = 30000  #the book value is 30000 at acquisition
jas = 0.4  #the annual depreciation rate is 40 percent

while (bw > 25):
 aw = bw * jas
 bw = bw - aw
 print(bw)
```

1.6 Application Session

In this section, we will only review what has been learned in this section.

1.6.1 Multiple Choice

EXERCISE 1.15 Each question has exactly one correct answer. Please mark this answer. You may only mark one answer per question.

Question 1: In which case is a while-loop more appropriate than a for-loop?

1. When I want to program linear depreciation in Python
2. When I do not know exactly how many times the loop should run
3. When I know how many times the loop should run

Question 2: Why are code conventions important?

1. Adherence to code conventions is a legally prescribed DIN standard
2. Code conventions are like the handwriting of a programmer, allowing you to recognize who wrote a particular piece of code
3. Code conventions help keep the code organized and understandable by following generally accepted rules

Question 3: What is crucial when creating Numpy arrays?

1. They must always have the same length
2. They must always contain the same data type to perform calculations
3. Once created, they cannot be modified

1.6.2 Text assignment

All text assignments should be solved by creating code. No points will be awarded for results without code.

EXERCISE 1.16 Due to the currently prevailing inflation, the prices of products are rising sharply. The three sisters would like to gain clarity about the possible future price development of their own products, assuming that prices increase by 4.5% each year.

The current prices of the four products are listed below.

- Wine "Sorelle Rosso" 7.20 €
- Wine "Sorelle Bianco" 6.80 €
- Soap "Sapone Allgäu" 3.59 €
- Cream "Crema Allgäu" 4.80 €

Calculate the prices of all products in 1, 2, and 3 years. Create a Numpy array for each year and print these values.

EXERCISE 1.17 After the sisters have typed everything in, they wonder if the price lists could not be filled more easily with a loop, especially if they want to model more than 3 years. Can you help the three sisters create and print price lists for the next 10 years? The output should start with the unchanged current prices.

Table 10: Orders

Year	1	2	3
Sorelle Rosso	12	16	13
Sorelle Bianco	18	17	17
Sapone Allgaeu	20	32	43
Crema Allgaeu	23	45	55

EXERCISE 1.18 Now that we know how much the individual products cost each year, we can calculate the revenue for each year. To do this, we will use the attached order overview:

What is the revenue for each year?

1.6.3 Debugging

In debugging, you need to find errors contained in the code. For example, missing quotation marks count as a single error, even if they occur in two places. The correct number of errors is provided in the task description. Please mark only that many errors.

EXERCISE 1.19 The three sisters wanted to write a for-loop to calculate how much revenue they can expect in the next 10 years. They assume that they will generate 500€ in the 2nd year and then achieve a 12% annual revenue increase.

Unfortunately, their code doesn't work! Can you help them find the 5 hidden errors in the code?

```
umsatz = 500  # Revenue in the first year is 500
steigerung = 0,12 # Revenue increases by 12% each year

for i in range(0:10)
   print(i)
   print(Umsatz)
   = umsatz + steigerung*umsatz
```

EXERCISE 1.20 Find the 7 errors in the following code.

```
import as np

Einzahlungen = ([20, 22, 24])
Auszahlungen = ([10, 11, 12])
Gewinn = 0

for i in range[0,3]:
   Gewinn = Gewinn + Einzahlungen[j] - Auszahlungen[j]

   print(Gewin)
```

1.6.4 Code Understanding

This task is about printing the outputs of the presented code. The code may contain some small traps, such as code that is never executed, or loops that run more or less often than they should. So, you need to be attentive!

EXERCISE 1.21 Below you have a Python code. What is the output of the print statements?

```
import numpy as np
```

```
a = 1
b = a
c = 2
d = "bc91"

v= np.array([2,3,4])

print (v)
print ("We start!")

for i in range(0,3):
   a = a + 2
   print (a)
   print (v[i])

if(a==c):
   print(b)

D = c
print(d)
```

EXERCISE 1.22 The three sisters have now set themselves a new, significantly higher revenue goal of 3,000 euros. They want to use the following code to find out when they will reach this goal.

Can you read the code and write down the output of the print statements?

```
import numpy as np

umsatz = 1250
umsatzziel = 3000
steigerung = 0.35
i = 0
v_jahr= np.arange(2018,2024)
print(v_jahr)

while umsatz < 3000:
   umsatz = umsatz+umsatz*steigerung
   i=i+1
   print("Year")
   print(i)
   print(v_jahr[i])
   print("Revenue")
   print(umsatz)

   if(umsatz > umsatzziel):
      print("Yay, we have reached our revenue target")
   else:
      diff = umsatzziel - umsatz
      print("We still need the following amount to reach the revenue target")
      print(diff)
```

PYTHON HACK #11: Always try to note the current values of the variables. In the task "Code understanding", the values of variables change regularly. These have a significant impact on the conditions in the while loop and the if statement. Thus, you should write them down seperately to track their current value.

1.6.5 Solutions

SOLUTION TO EXERCISE 1.15 Below is the solution to the exercise:
Question 1: In which case is a while-loop more appropriate than a for-loop?

1. When I want to program linear depreciation in Python
2. When I do not know exactly how many times the loop should run
3. When I know how many times the loop should run

Question 2: Why are code conventions important?

1. Adherence to code conventions is a legally prescribed DIN standard
2. Code conventions are like the handwriting of a programmer, which allows you to recognize who wrote a specific part of the code
3. Code conventions help keep the code organized and understandable by following generally accepted rules

Question 3: What is crucial when creating Numpy arrays?

1. They must always have the same length
2. They must always contain the same data type to perform calculations
3. Once created, they cannot be changed

SOLUTION TO EXERCISE 1.16 Below is the solution to the exercise:

```
import numpy as np

pm1 = np.array([7.20, 6.80, 3.59, 4.80])
pm2 = pm1 + pm1*0.045
pm3 = pm2 + pm2*0.045

print(pm1)
print(pm2)
print(pm3)
```

SOLUTION TO EXERCISE 1.17 Below is the solution to the exercise:

```
import numpy as np

pm1 = np.array([7.20, 6.80, 3.59, 4.80])

j=0

for i in range(0,10):

   print ("pm Year")
   j=j+1
   print (j)
   print(pm1)
   pm1 = pm1 *1.045
```

SOLUTION TO EXERCISE 1.18 Below is the solution to the exercise:

```
import numpy as np

# Creating the sales and prices
jahr1 = np.array([12, 18, 20, 23])
jahr2 = np.array([16, 17, 32, 45])
jahr3 = np.array([13, 17, 43, 55])

pm1 = np.array([7.20, 6.80, 3.59, 4.80])
pm2 = pm1 + pm1*0.045
pm3 = pm2 + pm2*0.045

print(jahr1)
print(jahr2)
print(jahr3)

print(pm1)
print(pm2)
print(pm3)

# Calculating the revenue for each year
um1 = pm1 * jahr1
um2 = pm2 * jahr2
um3 = pm3 * jahr3

print(um1)
print(um2)
print(um3)

# Calculating the total revenue
ug = um1 + um2 + um3
print(ug)

# There are alternative approaches to the solution, what matters is the correct
    final result
```

SOLUTION TO EXERCISE 1.19 Below is the solution to the exercise:

```
umsatz = 500   # Revenue in the first year is 500
steigerung = 0.12 # Revenue increases by 12% every year

for i in range(0,10):
   print(i)
   print(umsatz)
   umsatz = umsatz + steigerung*umsatz
```

SOLUTION TO EXERCISE 1.20 Below is the solution to the exercise:

```
import numpy as np

Einzahlungen = np.array([20, 22, 24])
Auszahlungen= np.array([10, 11, 12])
Gewinn = 0

for i in range(0,3):
   Gewinn = Gewinn + Einzahlungen[i] - Auszahlungen [i]

   print(Gewinn)
```

SOLUTION TO EXERCISE 1.21 Below is the solution to the exercise:

```
[2, 3, 4]
We start!
3
2
5
3
7
4
bc91
```

SOLUTION TO EXERCISE 1.22 Below is the solution to the exercise:

```
[2018 2019 2020 2021 2022 2023]
Year
1
2019
Revenue
1687.5
We still need the following amount to reach the revenue target
1312.5
Year
2
2020
Revenue
2278.125
We still need the following amount to reach the revenue target
721.875
Year
3
2021
Revenue
3075.46875
Yay, we have reached our revenue target
```

2 DATA MANAGEMENT

2.1 Data Structures and Data Manipulation

2.1.1 *Functions*

Functions are recurring processes. They help by automating frequently used procedures. When using a function in Python, various predefined processes are executed sequentially. We have also unconsciously used this in the past. Now, we want to explicitly address this and illustrate it using the example of "rounding numbers":

First, we tell Python which function we want to use, in this case with the command 'round()'. Then, we need to tell Python which number to round, here 2.34532, and how many decimal places to round to, here 3. Python will then decide whether to round the number up or down and when it does so. It sounds simple, but there are a few steps behind it, which you can execute with just one command.

```
number = 2.34532
print(round(number, 3))
```

In addition to the value we want to round, we also specify the number of decimal places we want to calculate. This is done through a second parameter after the comma. If no value is provided, the default is 0 decimal places. Specifying the argument names is often unnecessary. The following code works as well:

```
print(round(2.34532))
```

The meaning of the individual parameters can be found in the Python documentation. You will learn where to find this later. Below are some more examples of functions. There are many different functions; some are already included in Python by default, while others are downloaded via packages like "numpy." For example, "arange" is also a function.

```
import numpy as np

print(np.sqrt(16))
print(np.log(2.718))
print(round(2.781))

x = np.arange(1, 12, 2)
print(x)
```

Some functions have more than one parameter and some have optional parameters, e.g. round, where you can specify the number of decimal places.

2.1.2 *Regular Arrays*

We already know numpy arrays. There is a second type of array. In contrast to numpy arrays, it is not as good for calculations. However, they can also contain text. You can create the array as follows:

```
ARRAYNAME = ["Text", "Text"]

ARRAYNAME_TWO = [1,2,3]
```

This enables us to quickly create a list of names or a list of hobbies.

2.1.3 Dictionaries

A dictionary stores data in a "key : value" structure. We can imagine those structures like an actual dictionary. For example, you can see the translation of a German word into English.

Dictionaries are a standard functionality of python, are written with curly brackets, and have keys and values:

```
hobby = ["Tennis", "Painting"]

thisIsADict = {
  "Name": "Clara",
  "Age": 27,
  "Hobby": hobby
}
print(thisIsADict)
```

As you can see in the code, one can add different variable types to a data structure (text and numbers) and you can also assign the value of a variable to a dictionary (hobby).

2.1.4 DataFrames

So far, we have prepared data exclusively in arrays and matrices for clarity. Another option is provided by DataFrames. DataFrames are tables that closely resemble Excel tables. Rows and columns can also be labeled, which provides significantly more flexibility than the previous methods. In future sections, we will also learn how to load external data, such as from Excel, into our Python environment. These will then be stored as DataFrames by default.

To create a DataFrame, the package "pandas" is used. First, the columns x, y, z are created as np-arrays. These are then stored in a DataFrame named 'df'. In the function 'pd.DataFrame', the name "x" is assigned the content of the np-array x. Similarly, "y" is assigned to y and "z" to z. Remark that we use dictionaries to give our column a name.

DataFrames can also be easily displayed using the familiar 'print()' function.

Below is the code for creating a DataFrame:

```
import numpy as np
import pandas as pd

x = np.array([3,6,2,4])
y = np.array([4.4,2.32,7.6,9.2])
z = np.array([5,6,7,8])
df = pd.DataFrame({"x": x, "y": y, "z": z})
print(df)
```

Once created, DataFrames can also be displayed in rows and columns, much like Excel. However, DataFrames or Excel tables are often very detailed. To address this, you can use the 'head()' function to display the first rows of a DataFrame. For example, you can display the first 5 rows of the DataFrame 'df' with 'df.head(5)'.

```
df.head(5)
```

The DataFrame can now be further processed. For instance, individual columns, rows, or the value at a specific position in the DataFrame can be displayed.

DISPLAYING A COLUMN: To display a column named "x," we can use the following code. You can replace the column name as needed. In the example code below, the

values 3, 6, 2, 4 (column x) from the previously created DataFrame are displayed along with the column and row labels.

```
print(df["x"])
```

DISPLAYING A ROW: To display a row with the index "0," we can use the following code. The index counts from 0 (first row) to "number of rows - 1." You can also adjust the row index as needed. In the example code below, the values 3, 4.4, 5 (row with index 0) from the previously created DataFrame are displayed along with column and row labels.

```
print(df.loc[0])
```

DISPLAYING A VALUE: To display a value at an exact position, both the row and column need to be specified. You can adjust both the row index and the column name as needed. In the example code below, the value 4.4 (row with index 0 and column "y") from the previously created DataFrame is displayed WITHOUT column and row labels.

```
print(df.loc[0,"y"])
```

2.1.5 *Working with DataFrames*

In DataFrames, you can filter by specific values or sort by order, much like in Excel. The following examples illustrate some of these operations. To demonstrate these capabilities, we will recreate the DataFrame from the previous section.

```
import numpy as np
import pandas as pd

x = np.array([3,6,2,4])
y = np.array([4.4,2.32,7.6,9.2])
z = np.array([5,6,7,8])
df = pd.DataFrame({"x": x, "y": y, "z": z})
print(df)
```

DISPLAYING A SORTED SELECTION: Within a DataFrame, individual columns can be selected and displayed in a specific order. In the example below, only the columns "y" and "x" from the DataFrame are displayed in this order.

```
print(df[["y", "x"]])
```

Please remark, that we need a second pair of brackets (i.e. a list) to ask for more than one column.

ADDING OR REMOVING ROWS: A DataFrame can also be modified. For example, new columns can be added or removed. Additionally, calculations can be performed on columns, similar to operations with vectors and matrices discussed in previous sections. As an example, we create a column "w" in the DataFrame "df" and assign it the value of the sum of the columns "x" and "y."

```
df["w"] = df["x"] + df["y"]
```

Next, we delete the column "z".

```
df.drop(columns=["z"], inplace=True)
```

PYTHON HACK #2.1: If you use the parameter "inplace = True", deleting a column is **permanent**! You need to be very sure about what you want to do. Please proceed carefully! If you do not use this parameter and you just write "df.drop(columns=["z"])", then you only create a view for imediate printing or storing in a new variable. Rows can also be removed using "df.drop()".

After these transformations, we have modified our basic DataFrame. We removed the column "z" and added a new column "w." Consequently, we can no longer interact with the column "z." Below is the updated DataFrame:

```
print(df)
```

FILTERING WITH CONDITIONS: You can also output only parts of a DataFrame. For instance, we can display only the rows that satisfy a certain condition. For example, all rows where the value of "y" is greater than 5. This can be represented in the code as follows:

```
print(df[df["y"]>5.0])
```

You can use all possible conditions such as $>, <, ==, \ldots$ to filter. This allows you to selectively extract data from your DataFrame. For example, you can search for customers with a specific revenue range or similar criteria.

SORTING A DATAFRAME: Often, you may want to sort the contents of a DataFrame. For example, think of your phonebook, which you may want to sort by first name or last name. In Python, this can be done as follows. You can define the column by which to sort. In the example below, the DataFrame is sorted by "x." Since these are numeric values, they will be sorted by size. If you want to start with the smallest number or letter, you need to add the argument "ascending=True."

```
print(df.sort_values(by="x"))
```

COMBINING TWO DATAFRAMES: Sometimes, you may want to combine two datasets, such as the records from last year with those from the current year. To do this, the two DataFrames need to have the same columns. An example of combining two DataFrames is shown below:

```
x = np.array([2,1])
y = np.array([4.3,2.2])
w = np.array([7,8])
df2 = pd.DataFrame({"x": x, "y": y, "w": w})

df1und2 = pd.concat([df,df2])
print(df1und2)
```

FUNCTIONS OF DATAFRAMES: DataFrames also have various functions that can be accessed. For example, you can determine the number of elements, calculate the sum of a column, or display the minimum and maximum values of a column.

Please remark, that min and max also works for text. "min()" returns the first text in alphabetical order, "max()" returns the last text in alphabetical order.

EXERCISE 2.1 Create two separate arrays with the following values:

Table 11: Functions of DataFrames

Function	Description
df["ColumnName"].sum()	Sum of all values in the column
df["ColumnName"].min()	Smallest value in the column
df["ColumnName"].max()	Largest value in the column
df["ColumnName"].mean()	statistical mean of the (numeric) column
df["ColumnName"].count()	Number of elements in the column

Table 12: Exercise DataFrames

Age	Gender
17	f
20	m
18	f
18	f
19	m

1. Combine these into a DataFrame. Note that you are working with text, which must be enclosed in "quotes."
2. Calculate the average age of the individuals.

2.1.6 The Three Sisters

After the Three Sisters now understand what DataFrames are and how efficiently one can work with them, they want to collect information about their products in a DataFrame.

Table 13: The Three Sisters

Product Number	Product Code	Product Type	Product Name	Product Price
1	SR	Wine	Sorelle Rosso	8.90
2	SB	Wine	Sorelle Bianco	7.50
3	SA	Soap	Sapone Allgaeu	4.50
4	CA	Cream	Crema Allgaeu	6.35

EXERCISE THREE SISTERS 2.2 Please execute the following tasks:

1. Create a DataFrame with the products.
2. Sort the products by their price.
3. Please display only the distinct product codes and product names.
4. Can you display all products that cost less than €6.85?
5. The Three Sisters have reconsidered the DataFrame and would like to delete the "Product Number" column and add a column for the responsible sister. Clara is responsible for the wine, Hanna oversees the soap, and Anna manages the cream.
6. They would also like to add the order numbers from last year: SR: 438, SB: 342, SA: 246, CA: 294.
7. Could you also add a final column displaying the revenue for each product for the year?

2.1.7 Solutions

SOLUTION TO EXERCISE 2.1 Below is the solution to the exercise:

```
import numpy as np
import pandas as pd

age = np.array([17, 20, 18, 18, 19])
gender = np.array(["f", "m", "f", "f", "m"])
df = pd.DataFrame({"age": age, "gender": gender})
print(df)

average_age = df["age"].sum() / df["age"].count()
print("The average age is:")
print(average_age)
```

SOLUTION TO EXERCISE 2.2 Below is the solution to the exercise:

```
import numpy as np
import pandas as pd

product_number = np.array([1, 2, 3, 4])
product_code = np.array(["SR", "SB", "SA", "CA"])
product_type = np.array(["Wine", "Wine", "Soap", "Cream"])
product_name = np.array(["Sorelle Rosso", "Sorelle Bianco", "Sapone Allgaeu",
    "Crema Allgaeu"])
product_price = np.array([8.90, 7.50, 4.50, 6.35])

df_total = pd.DataFrame({"ProductNumber": product_number, "ProductCode":
    product_code, "ProductType": product_type, "ProductName": product_name,
    "ProductPrice": product_price})

print(df_total)

print(df_total.sort_values(by="ProductPrice"))

print(df_total[["ProductCode", "ProductName"]])

print(df_total[df_total["ProductPrice"] < 6.85]["ProductName"])

df_total.drop(columns=["ProductNumber"], inplace=True)

df_total["Responsibility"] = np.array(["Clara", "Clara", "Hanna", "Anna"])

df_total["Orders"] = np.array([438, 342, 246, 294])

df_total["Revenue"] = df_total["Orders"] * df_total["ProductPrice"]
```

2.2 Data Import and Data Preparation

2.2.1 *Reading Data*

In this section, we focus on importing and preprocessing data. This is the usual use case, as data is typically not manually entered into the code but generated using other tools. For example, we might analyze address data from CRM systems, booking data from an online store, social media account click counts, and more.

We receive data in various formats, such as Excel files, CSV files, or geospatial data files. In this section, we will work with CSV files (character-separated values). These files organize data into rows and columns, with rows separated by line breaks and columns separated by a specific character (e.g., a comma, semicolon, or tab).

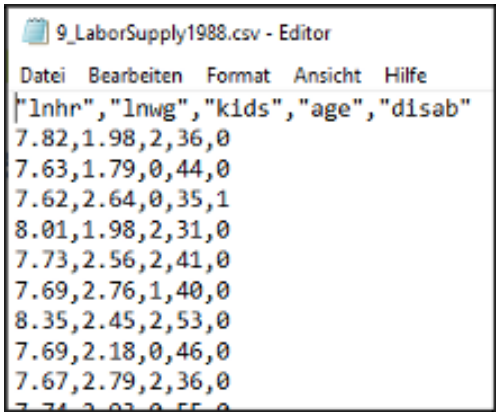

Figure 4: Example of a CSV file

The example image shows a CSV file opened in a text editor. It has a header row with column names ("lnhr", "lnwg",...) and rows with values (in American number formatting) separated by commas.

You can also create CSV files in Excel. To do this, open an Excel file, then go to "File" → "Save As", and in the dropdown menu, select the option "CSV (Comma delimited) (*.csv)".

PYTHON HACK #2.2: You can find out the path of your working directory in Python. This is called the **working directory**. To do so, use the 'getcwd()' function from the "os" package. Python will look for files to import or export in this location by default. In **Google Colab**, the working directory is "content". This is particularly important if you accidentally click ".." and suddenly see many folders.

To import data, we will work with an example dataset called "Labor Supply 1988". In Colab, specify just the file name, whereas in Anaconda, you need to provide the full or relative (to your working directory) file path where the document is located. We will save the dataset in Python as a "dataframe" with the abbreviation "df". Additionally, we will specify the delimiter (","), decimal separator ("."), and character encoding used in the document ("latin-1", i.e., the standard European encoding).

```
import pandas as pd

df = pd.read_csv("2_2_LaborSupply.csv", sep =",", decimal=".",
    encoding="latin-1")

df.head(5)
```

PYTHON HACK #2.3: Using the correct decimal separators is essential, especially because Python typically handles these differently than what you may be accus-

tomed to. By default, Python uses the period as a decimal separator. However, if you create a document using German Excel, it will use a "," as the decimal separator and ";" as the delimiter. Choosing the wrong characters will make the dataframe appear strange. You can use ‘df.head(5)‘ to display the first five rows of the dataset.

2.2.2 Saving Data

Of course, data can also be saved. We'll demonstrate this with an example using the (unchanged) Labor Supply 1988 dataset. You need to decide on the format in which the file will be saved—in this case, as a CSV file. Then, specify the table to save—here, "df". Finally, provide a file name.

```
import pandas as pd

df.to_csv("output.csv")
```

The file will be saved in the working directory.

2.2.3 Working with Text

JOINING TEXT: The Python command ‘join‘ allows you to concatenate two or more strings. For example, the two strings "Hello" and "you" can be joined into the string "Hello you" using ‘join‘. Below are two examples:

```
words = ["Hello", "you", "!"]
print(" ".join(words))

print("-".join(["I", "separate", "with", "hyphens"]))
```

REPLACING CHARACTERS: You can also replace characters in strings. For instance, all "a"s can be replaced with "e"s. If you want to store the modified text to make further changes, you need to save it in a variable.

```
text = "Hello, I am a Python program."

print(text.replace("a", "e"))

text = text.replace("a", "i")

print(text)
```

CASE CONVERSION: Strings can also be converted to upper or lower case. The method ‘upper()‘ converts everything to uppercase, and ‘lower()‘ converts everything to lowercase.

```
text = "Hello, I am a Python program."

print(text.upper())

print(text.lower())
```

2.2.4 The Three Sisters

The three sisters have been very busy lately. Several new products are now available, and alongside their small shop, they have also established an online shipping service.

EXERCISE THREE SISTERS 2.3 Please execute the following tasks:

1. Can you load both the order list and the product list into your Python environment? The respective CSV files are named accordingly.
2. What do you notice when you examine the data more closely?
3. Once a year, the three sisters need to prepare a summary for the company "Poypol" showing how many payment transactions were processed for them. Can you create a separate dataframe containing only the orders processed through this payment method?
4. Additionally, the list must then be exported and saved. Can you also handle this task for the three sisters?

2.2.5 Solutions

SOLUTION FOR SISTERS 2.3 Below is the solution for the exercise:

```
bestellungen = pd.read_csv("08_Bestellungen.csv", sep=";", encoding="latin-1")

produkte = pd.read_csv("08_Produkte.csv", sep=";", decimal=",",
    encoding="latin-1")
```

OBSERVATIONS: The decimal separator in the file was changed from a comma to the dot commonly used in Python. This only happened because we specified it with the 'decimal' argument. Umlauts seem to work fine within the entries, but in the column headers of "Orders" ('Bestellungen'), they cause issues. Additionally, previously empty fields in "Orders" now display "NA" in gray.

```
poypol = bestellungen[bestellungen["Zahlungsart"] == "Poypol"]

poypol.to_csv("08_poypol.csv")
```

2.3 Data Quality

Data quality metrics are measurable criteria used to assess the reliability, accuracy, and usability of data. They help organizations evaluate how well their data supports decision-making processes and operational needs. One can distinguish in technical and business metrics:

- Technical metrics focus on the structural and system-level characteristics of data, such as completeness, consistency, accuracy, and timeliness. These metrics ensure the data is technically sound and aligned with predefined standards.
- On the other hand, business metrics assess the relevance and value of data in achieving organizational goals, such as customer satisfaction, operational efficiency, or compliance with regulations.

Together, these metrics provide a comprehensive view of data quality from both technical and strategic perspectives.

Technical metrics include ...

- Completeness: Measures the proportion of data fields that are populated compared to the total required fields. For example, if a customer database has missing phone numbers or email addresses, completeness is affected.
- Consistency: Assesses whether data values across different datasets or systems align and do not conflict. For instance, a customer's address should match across billing and shipping systems.
- Accuracy: Evaluates whether the data correctly represents the real-world entity or event it describes. For example, ensuring that product prices in a database match those listed on a retailer's website.
- Timeliness: Measures how up-to-date the data is relative to when it is needed. For example, in financial reporting, timely data ensures transactions are recorded without delay.
- Uniqueness: Checks for duplicate records or values within a dataset. For instance, verifying there are no duplicate customer IDs in a database.
- Integrity: Assesses whether relationships between data elements are properly maintained, such as ensuring that foreign key references in a database are valid.

These technical metrics ensure data is technically reliable and can be effectively used for operational and analytical purposes.

Business metrics, on the other hand, are more case specific. Data must align with business goals or decision-making needs. For instance, ensuring customer segmentation data is suitable for targeted marketing campaigns. Furthermore, the data must adhere to legal, regulatory, or organizational standards. For example, ensuring customer records meet GDPR requirements for data privacy. Thus, business metrics focus on the practical and strategic impact of data on business performance, ensuring that data drives meaningful outcomes.

2.4 Application Session

In this section, we will only review what we have learned in this section.

2.4.1 Multiple Choice

EXERCISE 2.4 For each question, there is exactly one correct answer. Please mark the correct answer. You may only select one answer per question.

Question 1: What should be considered when importing data?

1. The resulting DataFrame must have the same name as the source file.
2. Only files in CSV format should be used.
3. It is important to know which column and decimal delimiters are used in the source file.

Question 2: What should be considered when exporting files?

1. A file saved in CSV format cannot be opened with Excel.
2. The file is saved by default in the "Downloads" folder.
3. The "pandas" package is required for exporting files.

Question 3: What is possible in Python when working with strings?

1. Strings cannot be modified after they have been created.
2. The lower() function can convert uppercase letters in a string to lowercase letters.
3. The replace() function can be used to combine two strings.

2.4.2 Text Problems

All text problems must be solved by creating code. Its important to write actual code to train your skills.

EXERCISE 2.5 Import the two datasets for the first and second quarters, which contain the orders. In a second step, the two datasets should be combined. Which function would be suitable for this? Please create a code that successfully uses this function.

PYTHON HACK #2.4 The function "print(DATAFRAME)" does not format DataFrames very well. With "display(DATAFRAME)", they are displayed as a nicely formatted table.

EXERCISE 2.6 Let's take a closer look at the dataset. Can you display only the "Name" and "Payment Type" columns to get a better overview?

EXERCISE 2.7 Unfortunately, the overall overview is still not very helpful. Since the "Delivery Address" column contains almost no entries, it should be deleted. You can also now output the contents as a table (use "display()"). Please write down the corresponding commands.

EXERCISE 2.8 Now, we would like to take a closer look at specific customer groups.

1. Please display only the orders from customers whose customer number is less than 100. These should be our regular customers.
2. It would also be useful to know how many customers come from Kempten. Can you store the customers who have indicated "Kempten" as their location in a separate DataFrame?

EXERCISE 2.9 Please join the following words with a ":" as the delimiter: "Mit" "Doppelpunkt" "Trenne" "ich" "gerne".

EXERCISE 2.10 You want to write a break gong for your little siblings for school. The numbers from 45 to 1 should be counted down. Finally, "Gong" should be printed.

EXERCISE 2.11 You bought a notebook. This should be depreciated linearly over 5 years. The notebook cost 1500 euros. Please print the year for each year, followed by the depreciation amount and the residual value.

2.4.3 Debugging

In debugging, the goal is to find errors in the code. For example, missing quotation marks count as an error, even if they occur in two places. The correct number of errors is provided in the task description. Please mark exactly as many errors as indicated. If you mark more errors, they will not be counted.

EXERCISE 2.12 Find the 5 errors in the following code.

```
Import_Daten =read_csv("", sep=";")
print(Import_Daten ["Zahlungsart"]
Import_Daten.to_csv(Import_Daten_C1", Index=False)
```

EXERCISE 2.13 Find the 5 errors in the following code.

```
Ga1 = 344
if(Ga1>1000):
   print("excellent project")
else:
   if(Ga1>900):
         print(sehr gutes Projekt)
   else
      if(Ga1=500):
            print("good project")
      else:
            prind("unprofitable project")
  )
```

EXERCISE 2.14 Find the 5 errors in the following code.

```
Zahl = 1
Kopf = 0
print("We are tossing a coin)
iff(Zahl=1):
   print("Heads")
else:
   print("Something is wrong")
if(Kopf == 0):
   prind("Tails")
else
   print("Something is wrong")
```

2.4.4 Code Understanding

In this task, you need to read the code and write down the output of the print statements. The code may contain small traps, such as code that is never executed or loops that run more or less often than they should. So, you need to pay attention!

EXERCISE 2.15 Can you read the code and write down the output of the print statements?

```
produktkuerzel = np.array(["SRo", "SdB", "CA"])
produktart = np.array(["Wine", "Wine", "Cream"])
produktpreis = np.array([8.90, 7.50, 6.35])
produktname = np.array(["Sorelle Rosso", "Sapone Allgaeu", "Cream"])
verantwortung = np.array(["Clara", "Hanna", "Anna"])

produkte_tabelle_neu = pd.DataFrame({"ProductCode": produktkuerzel,
    "ProductType": produktart, "Price":produktpreis,
    "ProductName":produktname, "Responsibility":verantwortung})
print(produkte_tabelle_neu)
print("-".join(produktart))
```

EXERCISE 2.16 Can you read the code and write down the output of the print statements?

```
i=0
y=2
mat = np.array ([[6,8,3,6],[4,77,4,7],[5,8,2,9],[1,4,78,37]])

while i < 4:
   if (i<2):
      mat[i,i]=1
      print(i)
      i =i+1
   else:
      mat[i,i]=y
      print(i)
      i =i+1
      y=y+1
print(mat)
```

EXERCISE 2.17 Can you read the code and write down the output of the print statements?

```
import numpy as np

Zahl = 1
zahl = 2
emil = "Hello"
hallo = "Emil"
arr = np.array([3,2,1])

for i in range(1,2):
   print (hallo)
   print (emil)
print(arr[zahl])
if(2<1):
   print(Zahl)

if(1<2):
```

```
   print(zahl)
else:
   print(Hello)
```

2.4.5 Solutions

SOLUTION TO EXERCISE SHEET 2.4 Below you will find the solution to the exercise:

Question 1: What should be considered when importing data?

1. The resulting DataFrame must have the same name as the source file.
2. Only files in CSV format should be used.
3. It is important to know which column and decimal separators are used in the source file.

Question 2: What should be considered when exporting files?

1. A file saved in CSV format cannot be opened with Excel.
2. The file is saved by default in the "Downloads" folder.
3. The "pandas" package is required for exporting files.

Question 3: What is possible in Python when working with strings?

1. Strings cannot be modified after they have been created.
2. The lower()-function can be used to convert uppercase letters in a string to lowercase.
3. The replace()-function can be used to combine two strings.

SOLUTION TO EXERCISE SHEET 2.5 Below you will find the solution to the exercise:

```
import pandas as pd

bestellungen_q1 = pd.read_csv("2_4_Orders_Q1.csv", sep=";", encoding="latin-1")
bestellungen_q2 = pd.read_csv("2_4_Orders_Q2.csv", sep=";", encoding="latin-1")
bestellungen_q1_q2 = pd.concat([bestellungen_q1,bestellungen_q2])

display(bestellungen_q1_q2)
```

SOLUTION TO EXERCISE SHEET 2.6 Below you will find the solution to the exercise:

```
display(bestellungen_q1_q2 [["Name","Zahlungsart"]])
```

SOLUTION TO EXERCISE SHEET 2.7 Below you will find the solution to the exercise:

```
bestellungen_q1_q2.drop(columns=["Lieferadresse"], inplace=True)
display(bestellungen_q1_q2)
```

SOLUTION TO EXERCISE SHEET 2.8 Below you will find the solution to the exercise:

```
print(bestellungen_q1_q2[bestellungen_q1_q2["Kundennummer"]<100])

bestellungen_kempten = bestellungen_q1_q2[bestellungen_q1_q2["Ort"]== "Kempten"]
display(bestellungen_kempten)
```

SOLUTION TO EXERCISE SHEET 2.9 Below you will find the solution to the exercise:

```
print(":".join(["Mit","Doppelpunkt", "trenne", "ich", "gerne"]))
```

SOLUTION TO EXERCISE SHEET 2.10 Below you will find the solution to the exercise:

```
i = 45
while (i>0):
   print(i)
   i=i-1
print("Gong")
```

SOLUTION TO EXERCISE SHEET 2.11 Below you will find the solution to the exercise:

```
Restwert = 1500
Betrag = 500
Jahr = 0
while (Restwert > 0):
   Restwert = Restwert - Betrag
   Jahr=Jahr+1
   print("Year:")
   print(Jahr)
   print("Residual value:")
   print(Restwert)
   print("Depreciation amount")
   print(Betrag)
```

SOLUTION TO EXERCISE SHEET 2.12 Below you will find the solution to the exercise:

```
Import_Daten = pd.read_csv("10_Bestellungen_Q1.csv", sep=";")
print(Import_Daten ["Zahlungsart"])
Import_Daten.to_csv("Import_Daten_C1", index=False)
```

SOLUTION TO EXERCISE SHEET 2.13 Below you will find the solution to the exercise:

```
Ga1 = 344
if(Ga1>1000):
   print("excellent project")
else:
   if(Ga1>900):
         print("very good project")
   else:
      if(Ga1==500):
            print("good project")
      else:
            print("unprofitable project")
```

SOLUTION TO EXERCISE SHEET 2.14 Below you will find the solution to the exercise:

```
Zahl = 1
Kopf = 0
print("We are tossing a coin")
if(Zahl==1):
   print("Heads")
else:
   print("Something is wrong")
if(Kopf == 0):
   print("Tails")
else:
   print("Something is wrong")
```

SOLUTION TO EXERCISE SHEET 2.15 Below you will find the solution to the exercise:

```
   ProductCode ProductType Price ProductName  Responsibility
1  SRo         Wine        8.90   Sorelle Rosso Clara
2  SdB         Wine        7.50   Sapone Allgaeu Hanna
3  CA          Cream       6.35   Cream        Anna
Wine-Wine-Cream
```

SOLUTION TO EXERCISE SHEET 2.16 Below you will find the solution to the exercise:

```
0
1
2
3
[[ 1 8 3 6]
 [ 4 1 4 7]
 [ 5 8 2 9]
 [ 1 4 78 3]]
```

SOLUTION TO EXERCISE SHEET 2.17 Below you will find the solution to the exercise:

```
Emil
Hello
1
2
# What is interesting here is what is not printed
```

3 DATA ANALYTICS

3.1 Basics of Data Analytics

3.1.1 What is Data Science?

Data Science is the process of examining, organizing, and interpreting data to uncover valuable insights, patterns, and trends. With the ever-increasing volume of data generated in various fields, from business and healthcare to social media and scientific research, data analytics has become an essential tool for decision-making and innovation. By transforming raw data into actionable knowledge, it empowers individuals and organizations to make informed decisions and solve complex problems. To achieve these goals, data analytics relies on various algorithms, which can be broadly categorized into the following classes:

- Descriptive Analysis: Understanding what has happened by summarizing historical data. These algorithms summarize data to provide an overview, such as calculating averages, variances, or frequencies. Examples include clustering and association rule mining.
- Predictive Analysis: Anticipating future trends or outcomes using statistical and machine learning models. These algorithms use historical data to predict future outcomes. Examples include regression, decision trees, and neural networks.
- Exploratory Algorithms: These algorithms identify hidden patterns or structures within data, such as principal component analysis (PCA) or k-means clustering.
- Prescriptive Analysis: Recommending actions or strategies to achieve desired outcomes based on data-driven insights. These focus on optimization and recommendation by analyzing the potential outcomes of different actions. Techniques include optimization algorithms and reinforcement learning.

By understanding the fundamentals of data analytics and its methodologies, one can gain the tools to harness the power of data and address challenges across diverse fields. In the following sections, we will dive into some types of algorithms.

3.1.2 Train and Test Data

Splitting data is a critical step in data science because it allows for robust model evaluation and prevents overfitting. One usually splits data into training, and testing sets. The training set is used to train the model, and the test set is reserved for final performance evaluation. The training set usually contains 60% and the test set typically 40% of the data. Splitting data ensures that the performance of a machine learning model is evaluated on unseen data. This helps to mimic real-world scenarios where the model encounters new data, ensuring it generalizes well beyond the training set. In this context, we speak of Overfitting. Overfitting occurs, if a model is trained and tested on the same dataset. Thus, it might simply memorize the data rather than learn patterns. This leads to overfitting, where the model performs well on training data but poorly on new, unseen data.

At some point in the following sections, we will draw back on this concept.

Some technical terms related to the topic:

PYTHON HACK #3.1: Underfitting occurs when a model cannot accurately capture the dependencies between data, usually due to its simplicity. This often results in a low R^2 with known data and poor generalization when applied to new data.

PYTHON HACK #3.2: Overfitting occurs when a model learns both the dependencies and the random noise in the data. In other words, the model learns the existing data too well. Complex models with many features or terms often tend to overfit. Such models usually produce high R^2 with known data but often fail to generalize, resulting in significantly lower R^2 when applied to new data.

3.1.3 Booking Data Set and Correlation

In the following sections, we will draw back on a booking data set. This data set is virtual and does not contain real data. It, however, enables us to apply some basic analytic tasks.

The data set does contain basic information on hotel bookings. It contains of the following information:

- Reservation ID: the ID of the booking
- Family Size: size of the family of the guests
- Rooms Booked: number of rooms the family booked
- Stay Duration (nights): the duration of the stay of the family
- Income (€): Income of the family
- Hotel: Name of the hotel
- Room Class: the room class which the family booked
- Cost of Stay (€): the cost of the family's stay

We can load the data set and get some initial information. For instance using "info()" to get information on the length and data types or "head()" to see some rows.

```
import pandas as pd

df = pd.read_csv("3_Hotel_Bookings_Dataset.csv")
df.info()
df.head()
```

Next, we can take a closer look at the correlation between the (numeric) data of the data set:

```

df.corr(numeric_only=True)
```

This code returns a table with all numeric variables as rows and columns. In the intersection between two variables, you can see their correlation. In this section, we draw back on the Bravais-Pearson-Correlation $R_{X,Y}$ with X,Y as random sample:

$$R_{X,Y} = \frac{\sum_{i=1}^{n}(X_i-\bar{X})(Y_i-\bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i-\bar{X})^2 \sum_{i=1}^{n}(Y_i-\bar{Y})^2}}$$

The correlation is a value between -1 and 1. 1 describes a perfectly positive correlation, -1 a perfectly negative correlation, and 0 indicates the lack of a correlation. Some further examples are illustrated in the following figure.

As you might remember from statistics, a variable has a perfect correlation with itself, resulting in a correlation of 1.

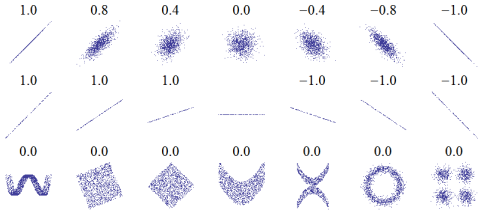

Figure 5: Some examples for correlation values from Wikipedia

3.2 Data Visualization

Data visualization is a crucial aspect of data analytics that helps communicate findings effectively. Data visualization encomasses visual representation, such as graphs and charts, which make complex data easier to understand.

On basis of a good data understanding, one can make informed decisions based on intuitive data representations. It also supports effective communication by well-designed visuals for storytelling, making it easier to convey insights to diverse audiences.

In Python, Matplotlib and Seaborn are two popular libraries for data visualization. Matplotlib provides a low-level, highly customizable framework for creating static, animated, and interactive visualizations. It allows for detailed control over plot elements, making it a powerful tool for technical and scientific visualizations. Seaborn, on the other hand, is built on top of Matplotlib and is designed for statistical data visualization. It offers a simpler interface and comes with built-in themes and functions to create more visually appealing and informative plots with minimal code. The main difference is that while Matplotlib provides fine-grained customization, Seaborn simplifies complex visualizations and integrates well with Pandas data structures.

In the following, we discuss some exemplary visualization methods. In the last section, you've seen some additional code. There are many different kind of charts, so that we cannot discuss all possible visualizations. This section, however, illustrates some of the basic concepts of visualization methods.

To visialize data, we require several packages you might have seen already:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

3.2.1 Count Plot

A count plot is a very basic plot. It is a type of bar plot used to visualize the count of categorical data. It is useful for showing how often each category appears in a dataset. Count plots are commonly used to understand the distribution of categorical variables. It uses bars to represent the frequency of each category. A Count Plot looks as follows:

Count Plots are usefull for the following reasons:

- When analyzing the distribution of categorical variables.
- When comparing the frequency of categories within a dataset.
- When detecting imbalanced data, especially in classification problems.

To create a Count Plot one must specify the data ("data=...") and variable within the data ("x=..."). Then you can select a title and "show()" the data.

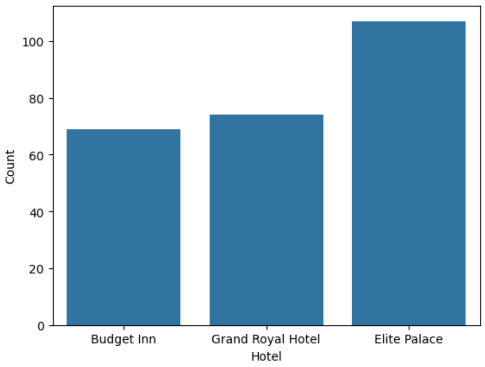

Figure 6: Exemplary illustration Count Plot

```
sns.countplot(data=df, x="Hotel")
plt.ylabel("Count")
plt.show()
```

3.2.2 Histogram Plot

A histogram is a type of bar chart used to represent the distribution of numerical data. It divides data into bins (intervals) and counts the number of observations that fall into each bin, helping to visualize the shape, spread, and central tendency of a dataset. A histogram represents numerical data by grouping it into intervals (bins). The x-axis represents the range of values, while the y-axis represents the frequency of observations in each bin. Helps to identify patterns, such as peaks, and gaps in data. Unlike a bar chart, which is used for categorical data, a histogram is for continuous data. A Histogram Plot looks as follows:

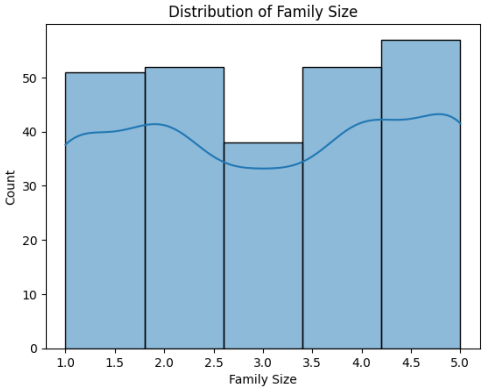

Figure 7: Exemplary illustration Histogram Plot

Histogram Plots are usefull for the following reasons:

- When analyzing the distribution of a single numerical variable.
- When detecting outliers, or gaps in the data.
- When determining whether data follows a normal distribution.

To create a violin plot one must specify the data variable within a dataframe ("df[...]"), a second visual line ("kde = True") and the number of intervals (bins). Then you can select a title and "show()" the data.

```
sns.histplot(df["Family Size"], kde=True, bins=5)
plt.title("Distribution of Family Size")
plt.show()
```

3.2.3 *Violin Plot*

A violin plot is a type of data visualization that combines aspects of a box plot and a density plot. It provides a way to visualize the distribution of a dataset while also displaying summary statistics.

The Violin Plot shows median, interquartile range, and outliers. It also includes a density estimation, which shows the distribution of the data. The wider sections of the violin indicate where more data points are concentrated, while the thinner sections indicate fewer data points. Violin Plots are useful for comparing multiple distributions in a compact space. A Violin Plot looks as follows:

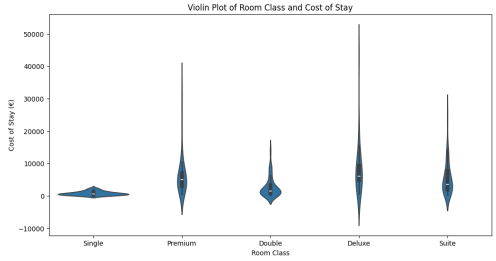

Figure 8: Exemplary illustration Violin Plot

Violin Plots are usefull for the following reasons:

- When you need to visualize the distribution of a dataset along with summary statistics.
- When comparing multiple categories or groups to see how their distributions differ.
- When you want more insight into the shape of the data distribution compared to a standard box plot.

To create a violin plot one must specify the x ("x=...") and y ("y=...") variable within a dataframe ("data=..."). Then you can select a title and "show()" the data.

```
sns.violinplot(x="Room Class", y="Cost of Stay (Euro)", data=df)
plt.title("Violin Plot of Room Class and Cost of Stay")
plt.show()
```

3.2.4 *Correlation Plot*

A correlation matrix plot is a data visualization tool used to display the correlation coefficients between multiple variables in a dataset. It helps in understanding the strength and direction of relationships between numerical variables. It presents a grid (matrix) where each cell represents the correlation coefficient between two variables. Typically, color gradients (e.g., blue for positive, red for negative) or circle sizes are used to enhance readability. Helps in identifying highly correlated variables, which is useful in feature selection for machine learning.

Correlation matrix plots are usefull for the following reasons:

- When analyzing the relationships between multiple numerical variables.
- When identifying redundant or highly correlated features in a dataset.

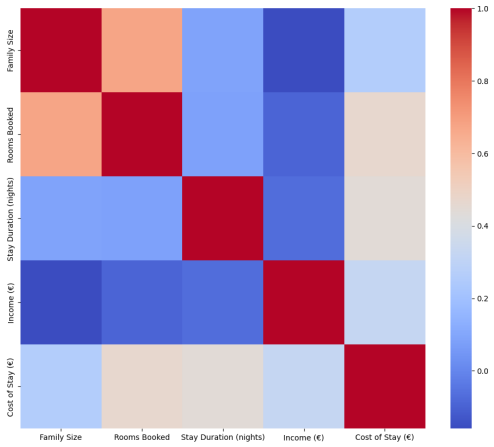

Figure 9: Exemplary illustration Correlation Matrix Plot

- When exploring patterns in data before applying statistical models.

To create a correlation matrix plot one uses the heatmap()-funktion of Seaborn. One must specify the data - in this case the correlation between the numeric variables in df. Furthermore, one must specify the color schema (cmap = 'coolwarm').

```
sns.heatmap(df.corr(numeric_only=True), cmap="coolwarm")
```

3.2.5 Scatter Plot

A scatter plot is a type of data visualization that represents individual data points on a two-dimensional graph, with one variable plotted on the x-axis and another on the y-axis. It is used to show relationships, correlations, or patterns between two numerical variables. Each point represents an observation with values for two variables. A scatter plot helps in identifying trends, clusters, and outliers in the data. A Scatter Plot looks as follows:

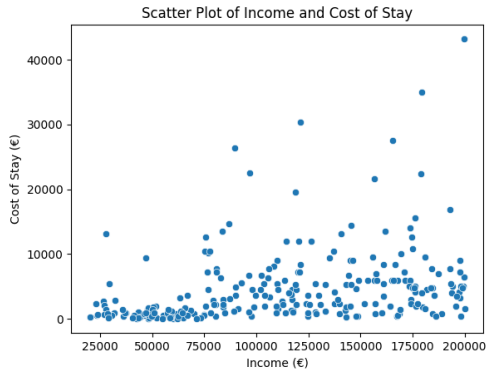

Figure 10: Exemplary illustration Scatter Plot

Scatter plots are usefull for the following reasons:

- When analyzing the relationship between two continuous variables.
- When identifying trends, clusters, or outliers in a dataset.
- When checking for correlation between variables before applying further statistical analysis.

To create a scatter plot one must specify the x ("x=...") and y ("y=...") variable within a dataframe ("data=..."). Then you can select a title and "show()" the data.

```
sns.scatterplot(x="Income (Euro)", y="Cost of Stay (Euro)", data=df)
plt.title("Scatter Plot of Income and Cost of Stay")
plt.show()
```

3.3 Linear Regression

In this section, we will work with data and focus on the topic of forecasting. Using linear regression, we aim to create predictions for future events. The goal is to understand how well a phenomenon (dependent variable) can be explained by other factors (independent variables). For instance, we could use past visitor numbers to predict future visitor numbers.

3.3.1 Linear Regression with One Input Variable

Linear regression is a tool from the field of machine learning that allows us to make statements about unknown potential phenomena. It can be performed with a single independent variable or multiple independent variables. In the case of one variable, the equation to solve is as follows:

$$\hat{y}_k = \beta_0 + \beta_1 * x_k$$

Here:

- k is the index of the observations,
- $\hat{y}_k$ is the estimated value $\hat{y}$ for observation k,
- x_k is the value of variable x for observation k,
- β_0 is the intercept of the regression line,
- β_1 is the slope of the regression line.

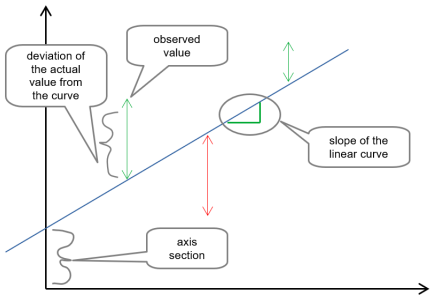

Figure 11: Graphical explanation of linear regression

The equation describes a straight line with an intercept and a slope. To determine the optimal regression line, we need to set β_0 and β_1 so that the squared deviations of the actual observed values y_k for each observed value x_k are as close as possible to the estimated values $\hat{y}_k$. This is referred to as minimizing the squared residuals (SSR for "Sum of Squared Residuals"). Squaring the deviations ensures that both positive and negative deviations are accounted for, as without squaring, they would cancel each other out.

If β_1 has a positive value, there is a positive relationship between x_k and y_k. Conversely, if β_1 is negative, there is a negative relationship, such as between precipitation levels and the number of people wearing shorts. An important assumption of linear regression is that it presumes a linear relationship between the independent variables and the target variable. If we now have an observation with $\hat{x}_k = potentialobservation$, we can substitute the value into the equation to obtain an estimated $\hat{y}_k$.

How regression works can best be explained with an example: Imagine you are the operator of a beer garden. The number of your guests depends heavily on the weather. For simplicity, let us assume that it does not rain in the observation period and that weekdays have no influence on the number of visitors. You have the following time series of past visitor numbers depending on the weather.

Table 14: Guests in the Beer Garden

Temperature x_k	28	23	32	35	29	30
Number of Guests y_k	200	60	440	360	180	410

QUESTION: According to the weather report, it is expected to be 33 degrees. Approximately how many guests should you expect?

Fortunately, we do not have to manually calculate the complex minimizations but can leave the task to the computer.

```
import numpy as np
from sklearn.linear_model import LinearRegression

# Defining the variables
x = np.array([28, 23, 32, 35, 29, 30]).reshape((-1, 1))
y = np.array([200, 60, 440, 360, 180, 410])

print(x)
print(y)

# Perform linear regression
model = LinearRegression().fit(x, y)

print(f"intercept: {model.intercept_}")
print(f"slope: {model.coef_}")

# Prediction
print("Now the prediction for 33 degrees:")
y_pred = model.predict(np.array([[33]]))
print(y_pred)
```

To calculate machine learning models, we need a special library. For this, we use "from sklearn.linear_model import LinearRegression". However, there are also other libraries that offer a similar range of models. After creating the two NumPy arrays for temperature (x) and the number of guests (y), we output them again for verification. Then, we create a model for linear regression and fit x and y. Next, we can output the intercept and slope of the estimated line.

Please remark, that "reshape((-1,1))" does transforms the row of the Numpy-Array to a column.

If one or more points on the line need to be estimated, we can use "model.predict (np.array([[VALUES]]))" to display them.

We can also visually illustrate the regression line for the case with two variables in a scatter plot. The temperature values are displayed on the x-axis, and the guest numbers on the y-axis. Our points from the NumPy arrays are plotted, where the first value of the x-array corresponds to the first value of the y-array, and so on. The red line shows the line with the smallest squared distance to the points. On this line lie our estimated values for unknown events.

Please also remark that you have learned how to include a regression line in a scatter plot.

```
# plot the linear regression
import matplotlib.pyplot as plt

plt.scatter(x, y)
plt.plot(x, model.predict(x), color="red")
plt.xlabel("x")
plt.ylabel("y")
plt.title("Linear Regression")
plt.show()
```

To generate a graphic, we first need to use the "matplotlib" library. This library enables us to create graphics using Python. First, we choose to create a scatter plot from the x and y values. Then, we can output the prediction of our model and color it red. Afterward, we label the x and y axes and give the graphic a title. The graphic is then displayed using `plt.show()`.

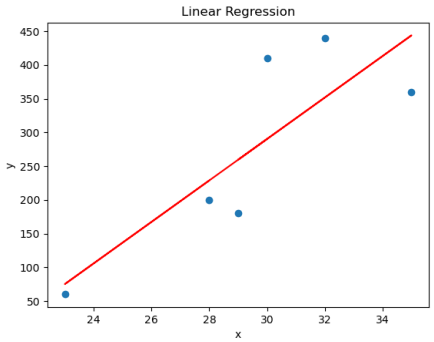

Figure 12: Plot of the Linear Regression

To evaluate a regression, the R^2 (R-Squared) metric is often used. Roughly speaking, this indicates the proportion of a phenomenon that can be explained by the observations. For a perfect regression, R^2 has a value of 1 (= 100%) and corresponds to an SSR of 0. The lower the value, the worse the regression.

```
r_sq = model.score(x, y)

print(f"coefficient of determination: {r_sq}")
```

The R^2 is calculated by asking the already computed model for its score for the two variables.

EXERCISE ON LINEAR REGRESSION 3.1 A few days have passed, and additional data about the beer garden's visitor numbers is now available. How has this data changed the model? What is the R^2 value of the new model? Also, plot the regression line.

Table 15: Guests in the Beer Garden

Temperature x_k	28	23	32	35	29	30	35	20	19
Number of Guests y_k	200	60	440	360	180	410	350	90	80

3.3.2 Linear Regression with Multiple Input Variables

Linear regression usually involves not just one observed variable but multiple variables. In our example, additional observed variables such as precipitation, holidays, or weekdays could be added alongside temperature. The more diverse information available, the better the explanatory power of the linear regression is, in general. For each observation k, every variable i must have a value $x_{i,k}$.

$$\hat{y}_k = \beta_0 + \beta_1 * x_{1,k} + ... + \beta_n * x_{n,k}$$

This can be further illustrated using our example above. This time, we also have counts of cyclists on a nearby bike path.

Table 16: Guests in the Beer Garden

Temperature x_k	28	23	32	35	29	30
Cyclists y_k	400	100	620	400	300	700
Number of Guests y_k	200	60	440	360	180	410

The code for this looks as follows:

```
import numpy as np
from sklearn.linear_model import LinearRegression

# Define the variables
x = np.array([[28,400], [23,100], [32,620], [35,400], [29,300], [30,700]])
y= np.array([200, 60, 440, 260, 180, 410])
print(x)
print(y)

# Perform linear regression
model = LinearRegression().fit(x, y)
print(f"intercept: {model.intercept_}")
print(f"slope: {model.coef_}")
```

The data for the model is stored as a NumPy array with two columns. For each observation point, the temperature x_1 and the number of cyclists x_2 are combined and stored in x. Then the model is calculated again. The output provides two values for the slope of X: one for x_1 and one for x_2, as well as a value for the intercept. If we had more variables in X, we would receive more values accordingly. A plot of the "line" is no longer possible, as it already involves three dimensions (x_1, x_2, y). As a result, we would need to represent a two-dimensional plane in a three-dimensional space. Since this is complex, it is not covered in the section. With three or more observations, spatial plotting is not possible anymore, as we would need 4 or more dimensions.

With our model, we can also make predictions again. For example, we can create a prediction for 33 degrees and an estimated 800 cyclists. For the cyclists, we must rely on estimated rather than observed values, as the number of cyclists in the future cannot be observed.

```
# Predictions
print("Now the prediction for 33 degrees and an estimated 800 cyclists:")
y_pred1 = model.predict(np.array([[33,800]]))
print(y_pred1)
```

The estimation looks very good at first glance. However, a regression is only valid for the observed range. We only have temperatures above 20 degrees. If we now try to estimate a rainy day, we get nonsensical results:

```
print("Now the prediction for 5 degrees and an estimated 40:")
```

```
y_pred2 = model.predict(np.array([[5,40]]))
print(y_pred2)
```

Accordingly, it is always necessary to check which data is available for the model and whether it is representative for new cases.

EXERCISE ON LINEAR REGRESSION 3.2 Test additional parameters. For which range do you believe the model is applicable?

PYTHON HACK #3.3: As the name suggests, linear regression attempts to model a linear relationship. However, this is not always the case. For example, if we have 50°C, noone will go to the beer garden. Linear regression is not suitable as a forecasting tool for such problems.

PYTHON HACK #3.4: There are many cases where no linear relationship exists—this can occur, for instance, when the phenomena being analyzed are simply unrelated. This is known as SpuriousCorrelations. Such correlations can lead to amusing models when the analyzed values happen to be similar. For example, the air pollution in Oklahoma City might correlate with the number of Google searches for "funny cat videos."

3.3.3 Application Example: Housing Prices

In this section, the knowledge on the topic of regression will be deepened. We will use an external dataset for this purpose. First, we will explore and analyze it. Accordingly, we will use all the tools from the previous sections to work with the data.

PYTHON HACK #3.5: You might encounter functions that you are not familiar with. However, the goal is to give you an impression of "Data Science at work." If you are unfamiliar with a function, the Python documentation will help you. You can find it at the following link: https://docs.python.org/3/

There, you can easily search for functions and terms you do not know. In the field of data science, you will spend 80% of your time searching for similar problems in documentation, on Google, StackOverflow, or GenAI Tools like ChatGPT. You will notice, however, that you will understand almost everything that happens today.

First, we will import everything we need for today. This includes a few new tools for visualization and data science.

```
# Import numpy and pandas

import numpy as np
import pandas as pd

# Data Visualization

import matplotlib.pyplot as plt
import seaborn as sns

# Data Science
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
```

Next, we load our dataset. Specifically, this time we want to help three sisters determine a price for their hotel. After all the work in the field of data science, they are now so motivated that they want to start their own IT company. Unfortunately, they now need to sell their hotel.

One of the sisters found a dataset online containing prices of houses in the United States. The sisters want to use this dataset as a reference. The data can be easily imported as a DataFrame, and the first rows can be displayed.

```
housing = pd.read_csv("3_3_Housing.xls")

housing.head()
```

Using `info()`, we can obtain additional information about the data. For instance, we can see how many missing values exist or what data types the variables have.

```
housing.info()
```

At this point, we could further process the data. However, because this is mostly manual effort, we simply remove the non-numeric variable "Adress".

```
housing = housing[["Price", "Avg. Area Income", "Avg. Area House Age", "Avg.
    Area Number of Rooms", "Avg. Area Number of Bedrooms", "Area Population"]]
```

We can now take a closer look at these variables. The "describe()" function gives us some statistical values for the variables, such as count, standard deviation, and various quantiles.

```
housing.describe()
```

Next, we can visualize the variables as a pairplot. For this, we use the "pairplot" of the "seaborn" library.

```
sns.pairplot(housing)
plt.show()
```

For each combination of variables, one variable is plotted on the X-axis and another on the Y-axis. The scatter plot indicates how the variables are correlated with each other. Please compare the section on correlation. When variables meet each other, there is a histogram, illustrating the distribution of the data.

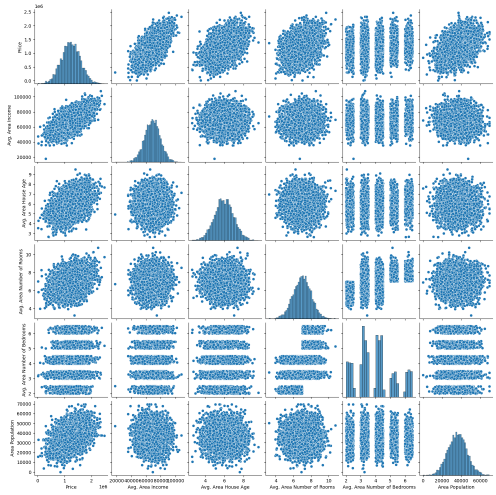

Figure 13: Pairplot of the data

After that, we select the price variable using "pop()" as the target variable for our linear regression. The pop function removes the value from the DataFrame. The remaining DataFrame is then used as the independent variable.

```
y = housing.pop("Price")
x = housing
```

With this data, we can now train the linear regression model.

```
lm = LinearRegression().fit(x, y)
```

Next, we can examine the coefficients.

```
#Coefficients of the regression
print(lm.intercept_)
print(lm.coef_)

#R-Squared
r_sq = lm.score(x, y)
print(f"coefficient of determination: {r_sq}")
```

EXERCISE 3.2 What do these coefficients mean? How can they be interpreted?

3.3.4 Application Example: Booking Data

In this section, we will analyze the booking data set for family vacations. This data enables us to build a prognosis model which can determine the cost of stay of family vacations.

First of all, we must import several packages. We will discuss the required functions in detail later.

```
#Import relevant packages
import numpy as np
import pandas as pd

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
```

Next, we import our data as DataFrame.

```
#Import and Describe the Data
df = pd.read_csv("3_Hotel_Bookings_Dataset.csv")

df.head()

df.info()
```

Next, we save an instance of the unprocessed data set. This allows us to access it again and again. Since we want to apply a linear regression, we must prepare our data. Only numerical values are allowed and we have to scale the data accordingly.

To process the text data, two approaches will be presented in the next step:

- First, we can remove the data. This is appropriate, if the variable contains unnecessary information. In our case, the data frame contains identifiers that are irrelevant.
- Second, we can add dummy variables for all instances of a variable. Since we do only have three different hotels in our data set, this is appropriate. Each hotel gets its own column which is "1", when the booking is for the respective hotel, and 0 in all other cases. The room class is processed similarly.

```
#Data Preperation
hotel_df = df.copy()

#Linear regression does not work with text data. Thus, we must handle
    "Reservation ID", "Hotel", and "Room Class".
#We can easily drop the reservation ID, since it is an Identifyer
hotel_df = hotel_df.drop(columns=["Reservation ID"])

#Split the variables "Hotel" and "Room Class" in dummy variables.
hotel_df = pd.get_dummies(hotel_df, columns=["Hotel", "Room Class"])

hotel_df.describe()
```

After that, we select the "Cost of Stay" variable using "pop()" as the target variable for our linear regression. The remaining DataFrame is then used as the independent variable.

Next, we split our data frame in test and training data. But why do we do this?

PYTHON HACK #3.6: In the last section, we discussed overfitting. Actually, we should test the developed regression on additional data to avoid overfitting. For this, we take the existing model and apply it to unknown data to see how well it performs.

For the split, we can draw back on on the function "train_test_split()" The function requires information on the data and the target variable as well as the split ratio.

```
#prepare the linear regression

y = hotel_df.pop("Cost of Stay (Euro)")
x = hotel_df

#Split the sample
x_train, x_test,y_train,y_test = train_test_split(x,y,test_size =0.4)
```

Next, we train our linear regression on our training data and examine its coefficients.

```
lm = LinearRegression().fit(x_train, y_train)

#Coefficients of the regression
print(lm.intercept_)
print(lm.coef_)

#R-Squared
r_sq = lm.score(x, y)
print(f"R-Squared: {r_sq}")
```

Finally, we use our regression to test our predictions on the test data set.

```
lm.predict(x_test)

lm.score(x_test, y_test)
```

When we take a closer look at our results, we see that our model is not very good.

EXERCISE 3.3 How can we improve our model?

3.4 Clustering

Sometimes we want to structure data. For example, we occasionally want to find similar customers in order to target them with a marketing campaign. Clustering algorithms search for similar patterns in the data to determine whether two instances are similar or not. There are several clustering algorithms. In the following, we will introduce K-Means-Clustering and illustrate its application.

3.4.1 K-Means-Clustering

One algorithm that attempts to determine the distance between individual points is K-Means clustering. This algorithm attempts to determine how close individual features are to each other. For example, for hotel prices, 110 Euros is closer to 100 Euros than 40 Euros. The algorithm attempts to find a specified number of clusters with a minimum distance.

An example: Please identify three clusters within the following accomodation prices of hotels:

Table 17: Accommodation Prices

Prices	20	30	40	70	70	80	110	120

This would obviously result in the three clusters $\{20, 30, 40\}$, $\{70, 70, 80\}$ and $\{110, 120\}$.

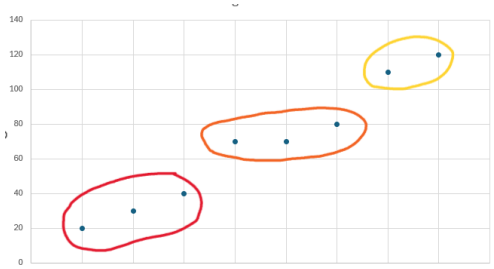

Figure 14: Graphical illustration of the clusters

Of course, this also works with more than one feature. We first specify how many clusters are to be formed. The algorithm then searches until it has found exactly as many clusters with a minimum distance between the individual points and the center.

The calculation of the distances looks as follows:

$$\min \sum_{i=1}^{k} \sum_{x \in S_i} \|x - \mu_i\|^2$$

With n observations $(x_1, x_2, ...x_n)$ and k desired clusters $(S_1, S_2, ...S_k)$. μ_i is the centroid, i.e. the mean of points in S_i:

$$\mu_i = \frac{1}{S_i} \sum_{x \in S_i} x$$

Please remark, that this approach requires numeric data with similar scales. If you would use data with a large scale and combine it with data of a small scale, the large scale data would dominate the small scale data in the results.

3.4.2 Application Example: Booking Data

In this section, we will draw back on our already introduced booking data set. We search for similar customers to target them with advertising.

First, we must import several packages.

```
#Import relevant packages
import numpy as np
import pandas as pd

from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import KMeans
```

In the first step, we import our data set as DataFrame.

```
#Import and Describe the Data
df = pd.read_csv("3_Hotel_Bookings_Dataset.csv")

df.head()

df.info()
```

Next, we save an instance of the unprocessed data set. This allows us to access it again and again. Since we want to apply K-Means clustering, we need to edit the data set. Only numerical values are allowed and we have to scale the data accordingly.

To process the text data, three approaches will be presented in the next step. In comparison to the last application section, we added one step:

- First, we can remove the data. This is appropriate, if the variable contains unnecessary information. In our case, the data frame contains identifiers that are irrelevant.
- Second, we can add dummy variables for all instances of a variable. Since we do only have three different hotels in our data set, this is appropriate. Each hotel gets its own variable which is "1", when the booking is for the respective hotel, and 0 in all other cases.
- Third, we can transform text data to an ordinal scale, if the data contains some differentiation. In case of our data set, this might be applied to the "Room Classes". However, one might argue, that the preferences of individual customers are not represented by our ranking.

Finally, we scale the data. Therefore, we use the "MinMaxScaler" of "sklearn". This algorithm is an appropriate approach do prepare our data for K-Means-Clustering.

```
#Data Preperation
hotel_data = df.copy()
hotel_data.head()

#K-Means does not accept text variables. Thus we must handle "reservation ID",
    "Hotel", and "Room Class".

#We can easily drop the reservation ID, since it is an Identifyer
hotel_data = hotel_data.drop(columns=["Reservation ID"])


#Split the variable "Hotel" in dummy variables
hotel_data = pd.get_dummies(hotel_data, columns=["Hotel"])

#We change the Room Classes from text to an ordinal scale with "Single" as
    worst and "Suite" as best category
```

```
hotel_data["Room Class"] = hotel_data["Room Class"].map({"Single": 1, "Double":
    2, "Premium": 3, "Deluxe": 4, "Suite": 5})

#Scale the data to avoid biases trough varying scales
scaler = MinMaxScaler()
hotel_scaled = pd.DataFrame(scaler.fit_transform(hotel_data),
    columns=hotel_data.columns)

hotel_scaled.describe()
```

In the next step, we choose the number of clusters. Then, we Initialize the K-Means Model and fit the data. Then, we add Labels to the data and add it to the data-set.

```
# Number of clusters
k = 3

# Initialize the K-Means model
kmeans = KMeans(n_clusters=k)

# Fit the model to the data
kmeans.fit(hotel_scaled)

# Get the cluster labels
labels = kmeans.labels_

# Add the cluster labels to the DataFrame
hotel_scaled["Cluster"] = labels
```

Finally, we can plot the data to illustrate the clustering. Remark, that we can only plot two dimensions of our 6-dimensional data. The combination of "Cost of Stay" and "Income" provides some interesting insights. Please remark, that we can use the labels "c=labels" to color them with "cmap=...".

```
# Plot the clusters depending of Income and Cost of Stay
plt.scatter(hotel_scaled.iloc[:, 3], hotel_scaled.iloc[:, 5], c=labels,
    cmap="viridis")
plt.ylabel("Cost of Stay (Euro)")
plt.xlabel("Income (Euro)")
plt.title("K-Means Clustering")
plt.show()
```

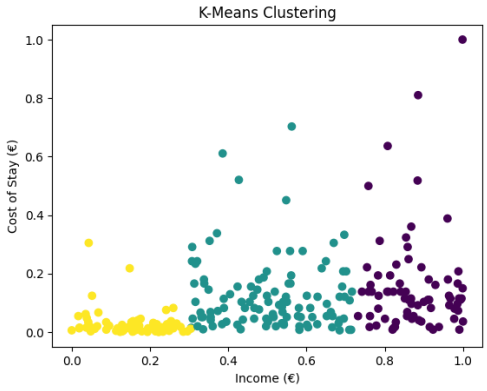

Figure 15: Graphical illustration of the clusters with colors

The clusters seem to be quite ok. Let's take a closer look at some numbers.

```
#In the following, we get some measures (SSE),
#the centers of the three clusters,
#and the number of iterations until we found the cluster

print("The loweset SSE Value ", kmeans.inertia_)
print("Final Cluster Centers: ", kmeans.cluster_centers_)
print("The number of iterations: ", kmeans.n_iter_)
```

The SSE value indicates, how good our clusters are. It computes the sum of the squared distance of all points to the epicenter of their cluster. The smaller an SSE value is, the better it is. However, it always depends on the problem how small it can be. For instance, for most problems, it cannot be 0.

The next line depicts the coordinates of the epicenters of the clusters. The last line indicates, how many iterations the algorithm required to come to its solution. This is important, when it comes to computing time.

3.4.3 The Ellbow Method

Apparently, the cluster algorithm seems to fit quite well. However, we somehow assumed, that three clusters are the best fit. However, this was just a smart guess. K-means requires you to specify the number of clusters in advance, but choosing the right K can be tricky.

The Elbow Method is a popular technique used to determine the optimal number of clusters (K) in K-means clustering. It prevents overfitting and underfitting and is quite simple to compute.

To use the ellbow method, we must install the package "kneed". Colab does not provide the package in it's environment. Thus, as explained in an earlier section, we use "pip" to install it:

```
pip install kneed
```

The following code illustrates how to implement the ellbow method:

```
# first we import the KneeLocator to locate the "ellbow point"
from kneed import KneeLocator

# you might not know what happens here. We define some arguments for the KMeans
    option and use it later using "**kmeans_kwargs"
kmeans_kwargs = {"init" : "random", "n_init" : 10, "max_iter" : 300,
    "random_state" : 42}

# We define a solution array, in which we store our results
sse = []

# We perform 10 iterations with 1 to 10 clusters. This enables us to identify a
    good amount of clusters
for k in range(1, 11):
   kmeans = KMeans(n_clusters=k, **kmeans_kwargs)
   kmeans.fit(hotel_scaled)
   sse.append(kmeans.inertia_)

# We plot the results
plt.style.use("fivethirtyeight")
plt.plot(range(1, 11), sse)
plt.xticks(range(1, 11))
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.show()

# We identify and plot the best elbow
kl = KneeLocator(range(1, 11), sse, curve="convex", direction="decreasing")
print(kl.elbow)
```

When we take a closer look at the result of the ellbow method, wee can see a clear kink, when we use 3 clusters.

PYTHON HACK #3.7: There are scenarios in which the elbow point is not clear, why one can use additional methods like the "silhouette score". However, those methods are out of scope of this book.

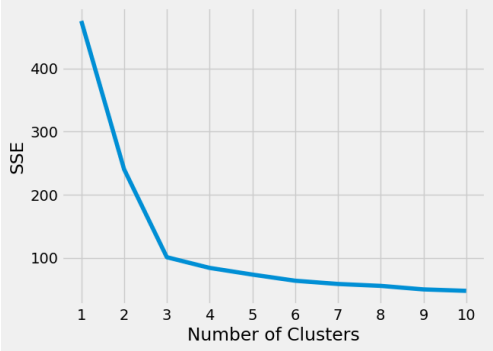

Figure 16: Graphical illustration of the Ellbow Method

3.5 Application Session

EXERCISE 3.2 You have not heard from the three sisters for a long time. They spent their time collecting data. Now, they are ready to contact you again and ask for a final job. In a dark room, they hand you over their customer data. The data set contains all their guests present during a specific week and their behavior on site.

Now you want to do the following tasks:

1. Import and explore the data set. How many customers are in the data set? Which variables are in the data set?
2. How many customers are male / female? Create a count plot.
3. How is the age distributed? Create a histogram.
4. You suspect that the age is positively correlated with the daily spending. Analyze this thought using a linear regression and plot your result in a scatterplot with a regression line.
5. You assume that there are clusters within your data. You assume 3 clusters. Use a k-means algrothm to get three clusters. Plot your results and color each of your clusters in a different color.

3.6 Solutions

SOLUTION TO EXERCISE 3.1 The solution to the exercise is as follows:

```
import numpy as np
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

# Define the variables
x = np.array([28, 23, 32, 35, 29, 30, 35, 20, 19]).reshape((-1, 1))
y = np.array([200, 60, 440, 360, 180, 410, 350, 90, 80])

print(x)
print(y)

# Perform linear regression
model = LinearRegression().fit(x, y)

print(f"intercept: {model.intercept_}")
print(f"slope: {model.coef_}")

# R-Squared
r_sq = model.score(x, y)

print(f"coefficient of determination: {r_sq}")

# Plot
plt.scatter(x, y)
plt.plot(x, model.predict(x), color="red")
plt.xlabel("x")
plt.ylabel("y")
plt.title("Linear Regression")
plt.show()
```

SOLUTION TO EXERCISE3.3 The solution to the exercise is as follows:

Import and explore the data set. How many customers are in the data set? Which variables are in the data set?

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv("data/Tourist_Spending_Patterns_Dataset.csv")
df.head()
```

```
df.info()
```

```
#drop tourist id since it is an identifyer
df.drop(columns="Tourist ID", inplace=True)

df.shape
```

How many customers are male / female? Create a count plot.

```
# Count Plot - Distribution of Sex
sns.countplot(data=df, x="Sex")
plt.ylabel("Count")
plt.show()
```

How is the age distributed? Create a histogram.

```
# Histogram - Distribution of Age
sns.histplot(df["Age"], kde=True, bins=7)
plt.title("Distribution of Age")
plt.show()
```

You suspect that the age is positively correlated with the daily spending. Analyze this thought using a linear regression and plot your result in a scatterplot with a regression line.

```
from sklearn.linear_model import LinearRegression

# Prepare the data
X = df[["Age"]].values
y = df["Daily Spending"].values

# Create and fit the model
model = LinearRegression()
model.fit(X, y)

# Get the coefficients
slope = model.coef_[0]
intercept = model.intercept_

print(f"Slope: {slope}")
print(f"Intercept: {intercept}")

# Calculate the R-squared value
r_squared = model.score(X, y)
print(f"R-squared: {r_squared}")

# Predict daily spending based on age
df["Predicted Daily Spending"] = model.predict(X)
df[["Age", "Daily Spending", "Predicted Daily Spending"]]
```

```
# Scatterplot - Age vs. Daily Spending with Regression Line
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x="Age", y="Daily Spending")
plt.plot(df["Age"], df["Predicted Daily Spending"], color="red")
plt.title("Age vs. Daily Spending with Regression Line")
plt.ylabel("Daily Spending in Euro")
plt.show()
```

You assume that there are clusters within your data. You assume 3 clusters. Use a k-means algrothm to get three clusters. Plot your results and color each of your clusters in a different color.

```
# Dummy Variables for Accommodation Type
df = pd.get_dummies(df, columns=["Accommodation Type"])

# K-Means Clustering
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# normalize the data
scaler = StandardScaler()
X = scaler.fit_transform(df[["Age", "Daily Spending", "Accommodation
    Type_Airbnb", "Accommodation Type_Hostel", "Accommodation Type_Hotel",
    "Accommodation Type_Resort"]])
```

```
# Create and fit the model
model = KMeans(n_clusters=3, random_state=42)

model.fit(X)

# Add the cluster labels to the DataFrame
df["Cluster"] = model.labels_

print(df.value_counts("Cluster"))

df.head()
```

```
# Scatterplot - Age vs. Daily Spending by Cluster
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x="Age", y="Daily Spending", hue="Cluster",
    palette="tab10")
plt.title("Age vs. Daily Spending by Cluster")
plt.ylabel("Daily Spending in Euro")
plt.show()
```

4 CLOSING WORDS

First of all, the most important thing: Thank you for taking the time and working through the book.

You have now learned everything you need to get started in the field of Data Science in tourism. You can now acquire further knowledge on various platforms.

I would particularly recommend the site https://www.kaggle.com/. There you will find various advanced tutorials, examples, and datasets. Please do not worry if you do not understand everything and do not always have a solution for your problem. Your most important helpers are Websearch and StackOverflow (https://stackoverflow.com/). Generative AI tools like GitHub Copilot (https://github.com/features/copilot) are also free for students.

```
print("And now, have fun on your future analytics journey!")
```