Artificial Intelligence

Algorithms and Applications with Python

Chapter 4

Dr. Dominik Jung
dominik.jung42@gmail.com

Outline

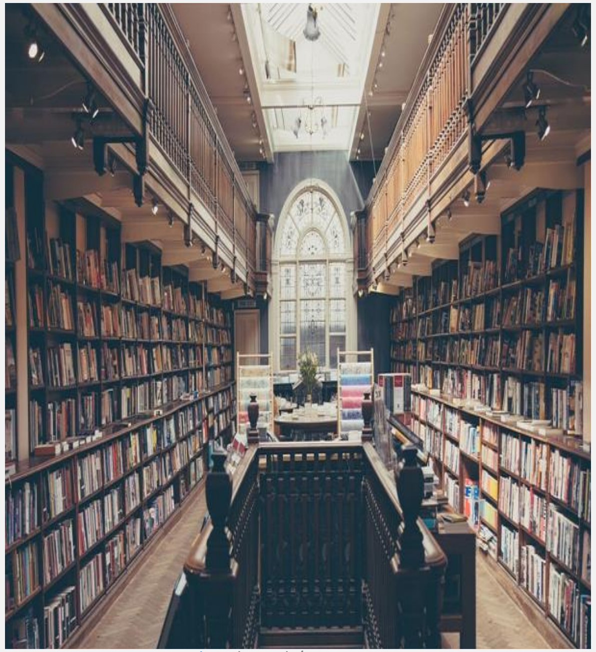


Image source: ↗ Pixabay (2019) / ↗ CC0

▶ **What we will learn:**
- How data and knowledge is handled in AI-based information systems
- Most common methods for exploring the data to get better domain understanding
- Learn to prepare and transform data to make it ready for your AI project
- Deepen your Python programming skills using popular Python modules like Pandas and Scikit-learn for data engineering

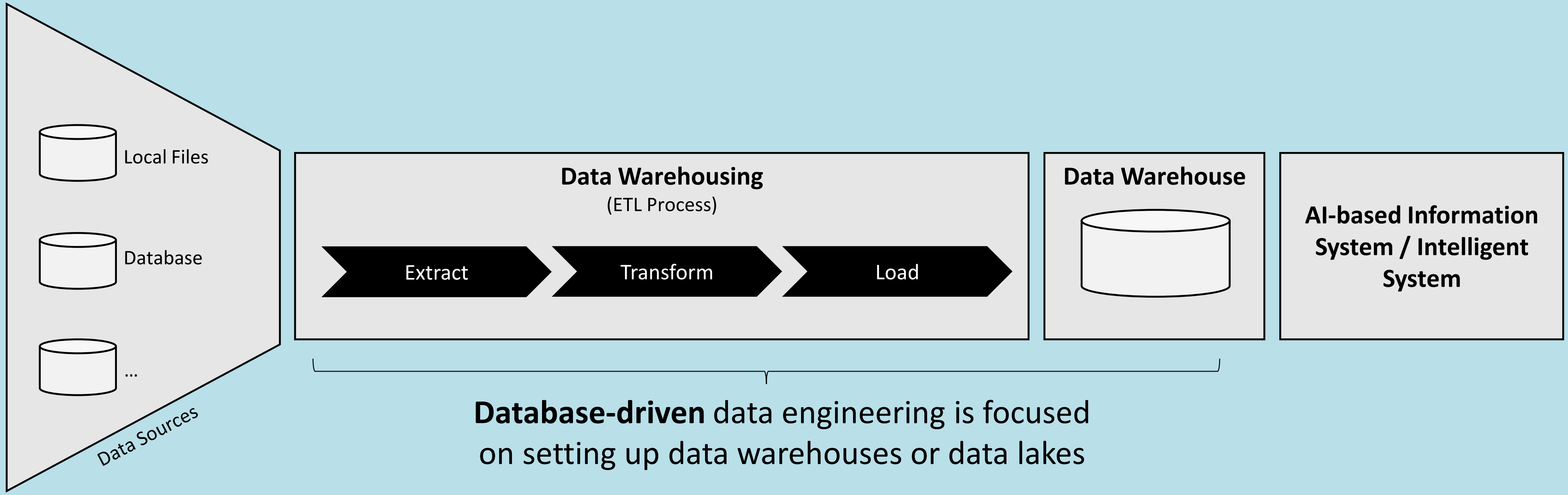
▶ **Duration:**
- 135 min

▶ **Relevant for Exam:**
- 4.1-4.5

Why Data Understanding is so important…

Adobe Analytics Cloud
https://www.youtube.com/watch?v=iANv_0ZQKDY

4.1 Data Engineering Build Data Systems in AI Projects I

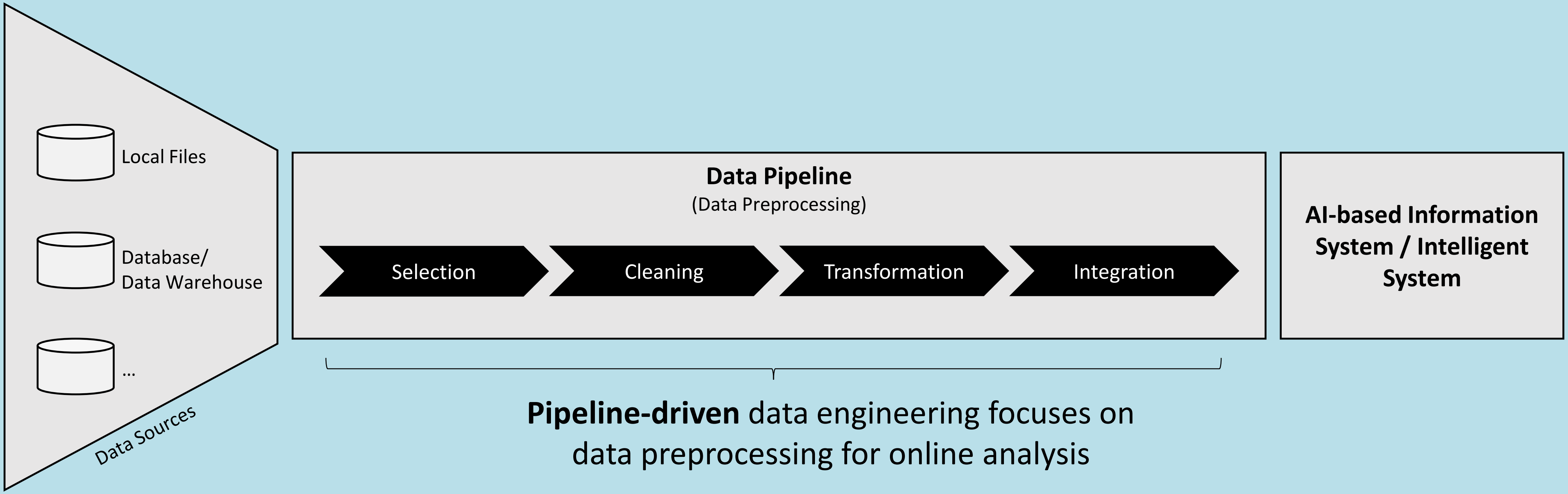


Database-driven data engineering is focused on setting up data warehouses or data lakes

D **Data Engineering**

Designing, building and maintain data systems (data warehouse, data pipelines etc.)

4.1 Data Engineering Build Data Systems in AI Projects II

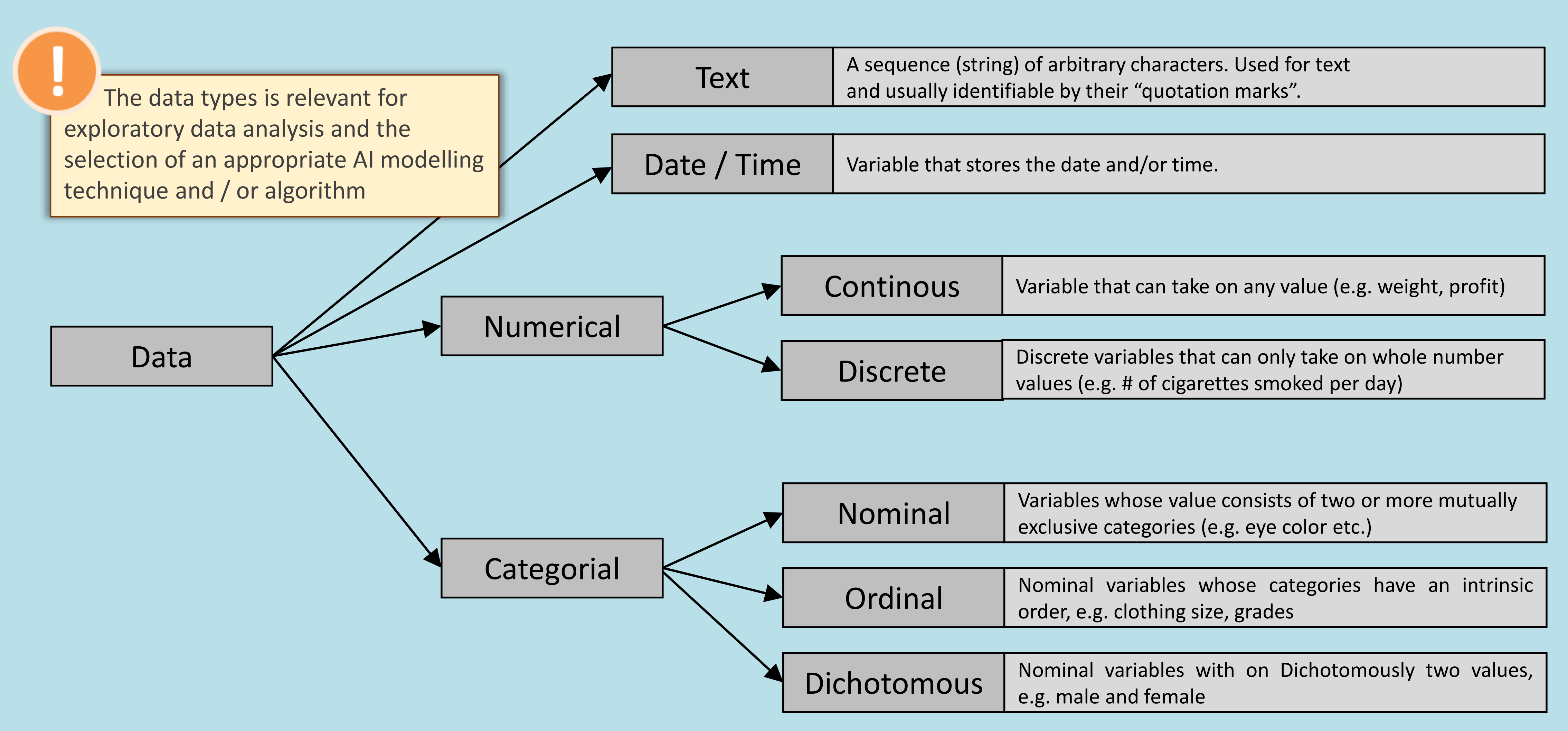


Pipeline-driven data engineering focuses on data preprocessing for online analysis

D **Data Engineering**

Designing, building and maintain data systems (data warehouse, data pipelines etc.)

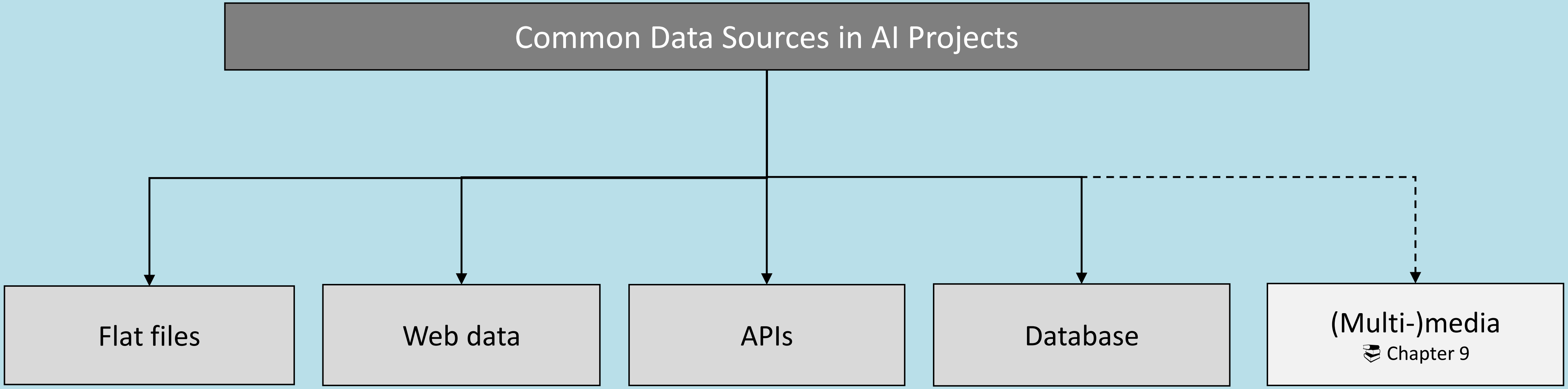
4.1 Data Types



4.1 Data Types and Related Variable Types in Python

Data	Data/Variable Type Python	Code Example	Storage (in Bytes)
Text	`string`	`var = "AI is my favorite lecture!"`	*Depends on assigned character length (set in database)*
Date / Time	`date`	`Import datetime var = date(year = 2018, month = 7, day = 12)`	*1- 8*
Continuous	`float, double`	`var = 42.42`	*4, 8*
Discrete	`integer, short, long`	`var = 42`	*2, 2, 4*
Nominal, Ordinal	`string, character, encoded integer`	`var = 1`	*Variable, 1, 2*
Dichotomous	`boolean, string, character, encoded integer`	`var = True`	*2, variable, 1, 2*

4.1 Common Data Sources



4.1 File Objects

```
fobj = open("database_extract.txt", mode = "r")
```

- There are many other specification you can set when you read a file:

r	Open a file for reading *(default)*
w	Open a file for writing. Creates a new file if it does not exist or truncates the file if it exists
x	Open a file for exclusive creation. If the file already exists, the operation fails.
a	Open for appending at the end of the file without truncating it. Creates a new file if it does not exist
t	Open in text mode *(default)*
b	Open in binary mode
+	Open a file for updating (*reading and writing*)

Adapted from the Offical Python Documentation (2019): https://docs.python.org

4.1 File Objects - Examples

- Read file

```
read_file = open("text.txt", mode="r")
```

- (Over)write file

```
new_file = open("text.txt", mode="w")
```

- Add content to file (appending)

```
updated_file = open("text.txt", mode="a")
```

- Do not forget to close your file, when you are done

```
_file.close()
```

Offical Python Documentation (2019): https://docs.python.org

4.1 Read Whole File Objects

- If you have generated the file object you can iterate over the lines

```
for line in file:
    do_something()
```

4.1 Delimited Files

- A delimited text file represents tabulated data by using specific characters (delimiters) to separate rows and columns

Model	Comment	UserID
911	„i like the sound of the motor"	1
718	„best car ever"	4

```
Model | Comment | UserID
911 | "i like the sound of the motor" | 1
718 | "best car ever" | 4
```

- Most popular type in data science is *.CSV: comma-separated-values

Offical Python Documentation (2019): https://docs.python.org

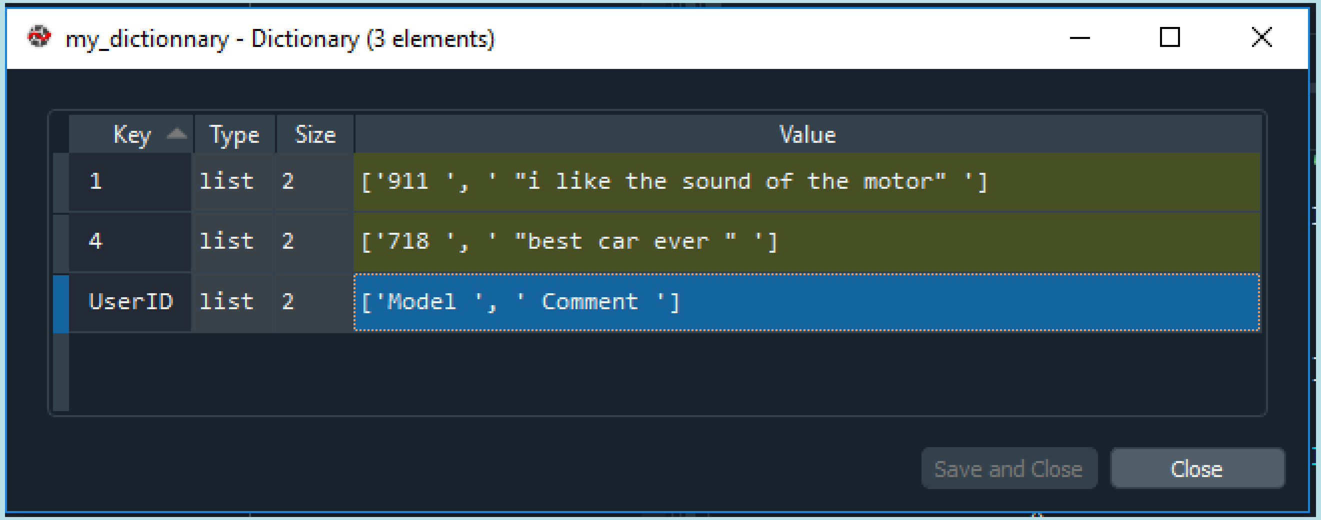
4.1 Read Delimited Files

- If you have generated the file object you can iterate over the lines

```
import csv

read_file = open("delimited_file.txt", mode="r")
reader = csv.reader(read_file, delimiter="|")

my_dictionnary = {}

for row in reader:
    model = row[0]
    comment = row[1]
    user = row[2]
    my_dictionnary[user] = [model, comment]

read_file.close()
```

Offical Python Documentation (2019): https://docs.python.org

4.1 Read Delimited Files – Skip Header



Key	Type	Size	Value
1	list	2	['911 ', ' "i like the sound of the motor" ']
4	list	2	['718 ', ' "best car ever " ']
UserID	list	2	['Model ', ' Comment ']

If we run the code, we see that the reader also reads the header line.

```
next(reader)

for row in reader:
    model = row[0]
    comment = row[1]
    user = row[2]
    my_dictionnary[user] = [model, comment]
```

We just command the reader to skip the first line before he starts to iterate over the lines

4.1 Read Delimited Files – Final Code

- If you have generated the file object you can iterate over the lines

```
import csv

read_file = open("delimited_file.txt", mode="r")
reader = csv.reader(read_file, delimiter="|")

my_dictionnary = {}

next(reader)
for row in reader:
    model = row[0]
    comment = row[1]
    user = row[2]
    my_dictionnary[user] = [model, comment]

read_file.close()
```

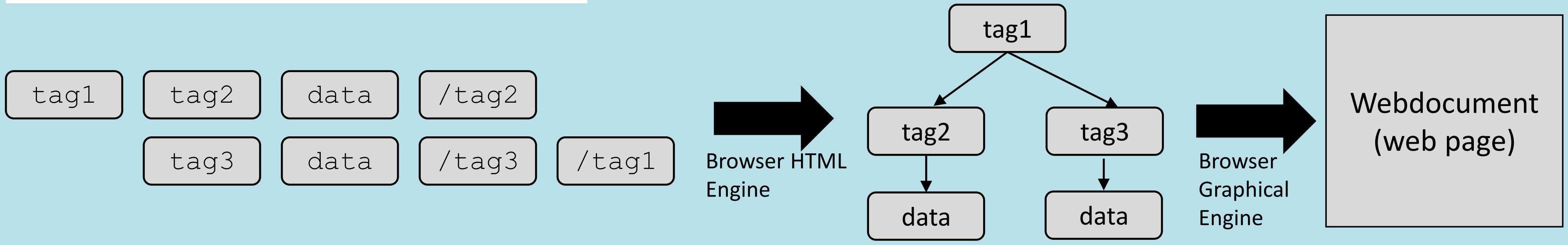
! Instead of skipping you can store the header in an other variable, and use it later for the column names

Offical Python Documentation (2019): https://docs.python.org

4.1 Scraping the Web

- Web data is a very popular data source for many AI applications
- The de-facto standard in the web for documents is HTML
- HTML has a tree-like structure, the so called document object model. It says that the distinct elements of an document have to be defined as follows:

```
<element> data </element>
```



Offical Python Documentation (2019): https://docs.python.org

4.1 Parser

- Python has build-in HTML parser to parse web documents and use them in your applications.
- If you want to use it, you have to handle start tags, data, and the end tag

```
from html.parser import HTMLParser

class Parse(HTMLParser):
    def handle_starttag(self, tag, attrs):
        pass

    def handle_data(self, data):
        pass

    def handle_endtag(self, tag):
        pass
```

Offical Python Documentation (2019): https://docs.python.org

```
from html.parser import HTMLParser

class MyHTMLParser(HTMLParser):
    def handle_starttag(self, tag, attrs):
        print("Encountered a start tag:", tag)

    def handle_endtag(self, tag):
        print("Encountered an end tag :", tag)

    def handle_data(self, data):
        print("Encountered some data  :", data)

parser = MyHTMLParser()
```

```
parser.feed()
```

Offical Python Documentation (2019): https://docs.python.org

4.1 Scraping the Web

```
>>> parser.feed('<html><head><title>Test</title></head>'
            '<body><h1>Parse me!</h1></body></html>')

Encountered a start tag: html
Encountered a start tag: head
Encountered a start tag: title
Encountered some data  : Test
Encountered an end tag : title
Encountered an end tag : head
Encountered a start tag: body
Encountered a start tag: h1
Encountered some data  : Parse me!
Encountered an end tag : h1
Encountered an end tag : body
Encountered an end tag : html
```

Offical Python Documentation (2019): https://docs.python.org

4.1 Scraping the Web for Specific Files

```
class Parser(HTMLParser):
    def __init__(self):
        HTMLParser.__init__(self)
        self.is_wikitable = False
        self.is_row = False
        self.table = []

    def handle_starttag(self, tag, attrs):
        if tag == "table":
            for name, value in attrs:
                if name == "class" and value == "wikitable":
                    self.is_wikitable = True
        if tag == "tr":
            self.is_row = True

    def handle_endtag(self, tag):
        if tag == "table":
            self.is_wikitable = False
        if tag == "tr":
            self.is_row = False

    def handle_data(self, data):
        if(self.is_wikitable):
            if(self.is_row):
                self.table.append(data.strip())
```

- Identify specifc tags and parse them, then store them in a list for further analysis
- Pythons build-in HTML parser has problems with HTML that are not perfectly formed, hence I recommend to use more state-of-the-art frameworks

4.1 Scraping the Web like a Pro

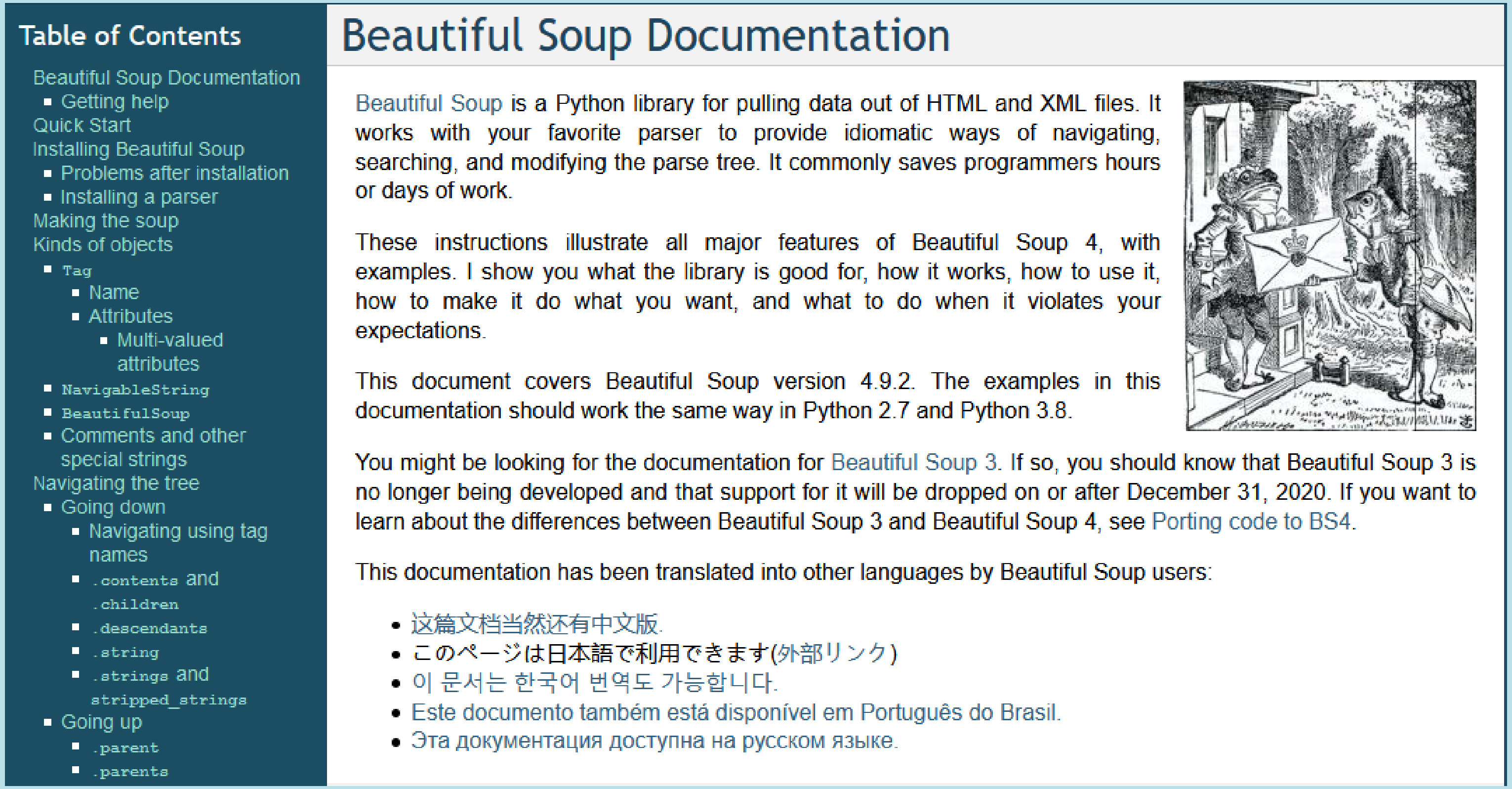


Table of Contents

- Beautiful Soup Documentation
  - Getting help
- Quick Start
- Installing Beautiful Soup
  - Problems after installation
  - Installing a parser
- Making the soup
- Kinds of objects
  - Tag
    - Name
    - Attributes
      - Multi-valued attributes
  - NavigableString
  - BeautifulSoup
  - Comments and other special strings
- Navigating the tree
  - Going down
    - Navigating using tag names
    - .contents and .children
    - .descendants
    - .string
    - .strings and stripped_strings
  - Going up
    - .parent
    - .parents

Beautiful Soup Documentation

Beautiful Soup is a Python library for pulling data out of HTML and XML files. It works with your favorite parser to provide idiomatic ways of navigating, searching, and modifying the parse tree. It commonly saves programmers hours or days of work.

These instructions illustrate all major features of Beautiful Soup 4, with examples. I show you what the library is good for, how it works, how to use it, how to make it do what you want, and what to do when it violates your expectations.

This document covers Beautiful Soup version 4.9.2. The examples in this documentation should work the same way in Python 2.7 and Python 3.8.

You might be looking for the documentation for Beautiful Soup 3. If so, you should know that Beautiful Soup 3 is no longer being developed and that support for it will be dropped on or after December 31, 2020. If you want to learn about the differences between Beautiful Soup 3 and Beautiful Soup 4, see Porting code to BS4.

This documentation has been translated into other languages by Beautiful Soup users:

- 这篇文档当然还有中文版.
- このページは日本語で利用できます(外部リンク)
- 이 문서는 한국어 번역도 가능합니다.
- Este documento também está disponível em Português do Brasil.
- Эта документация доступна на русском языке.

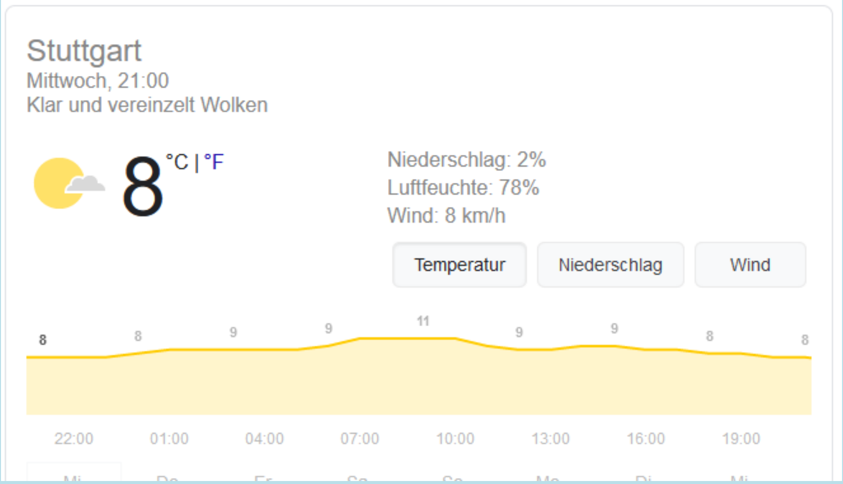
- Python library for pulling data out of HTML and XML files

Beautiful Soup Project

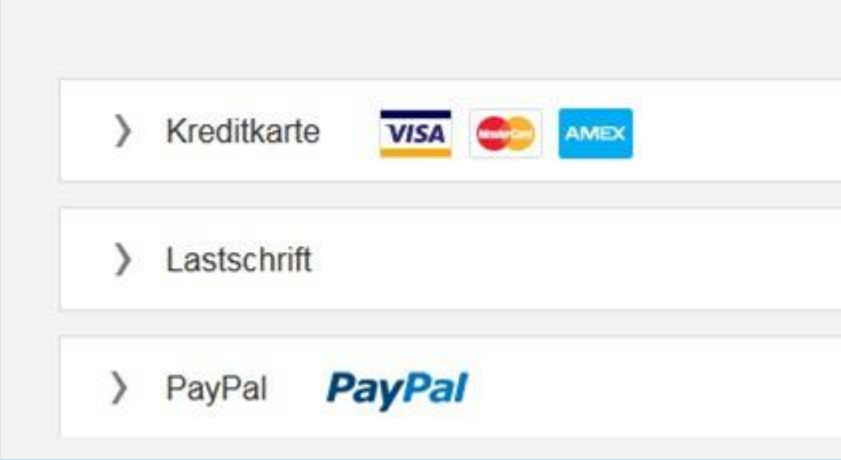
4.1 Application Programming Interfaces (API)

- Acronym for Application Programming Interface
- Software interface that enables that two or more programs can interact with each other by specifying requests, their schema and used data formats etc.

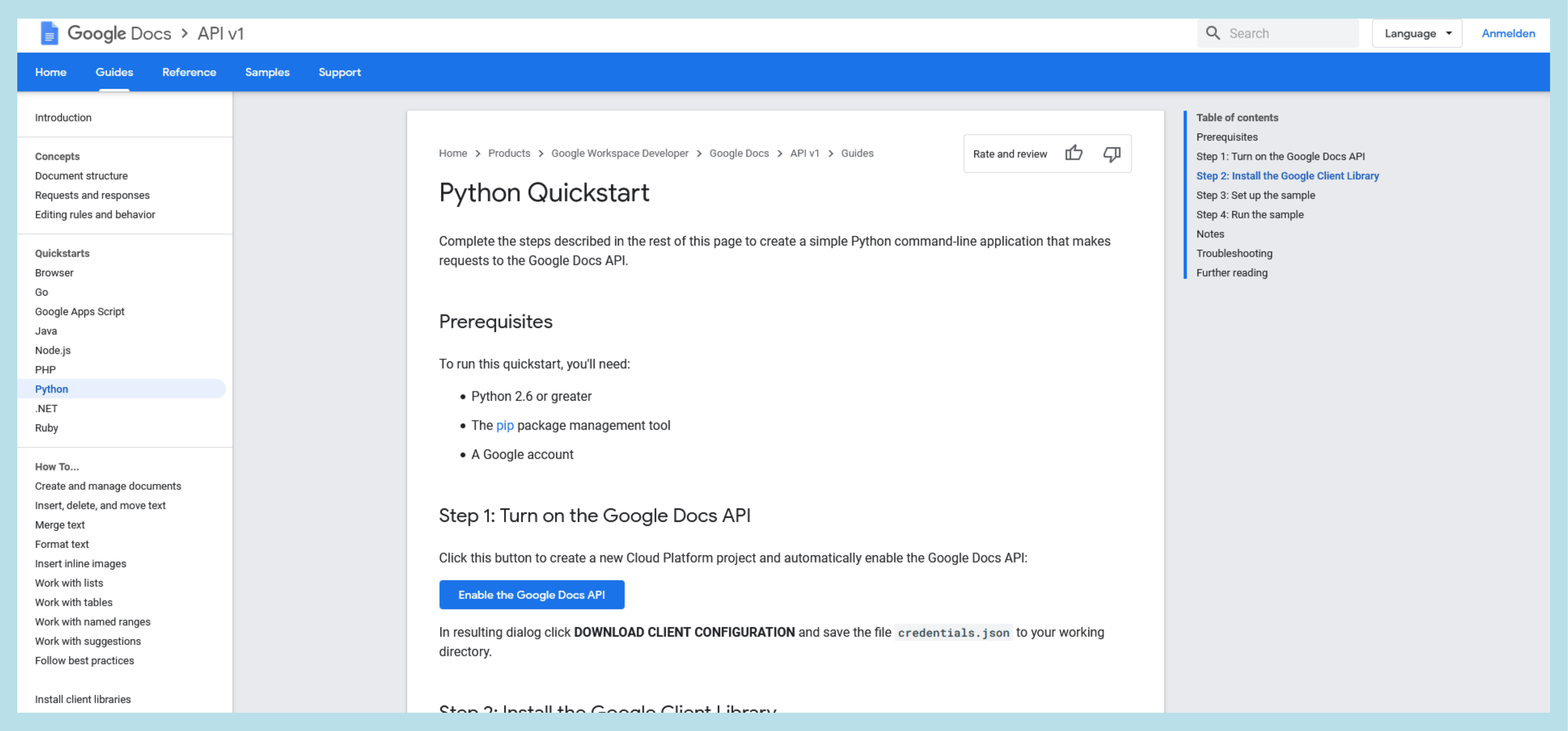
Google Weather



Online Payment in E-Shops



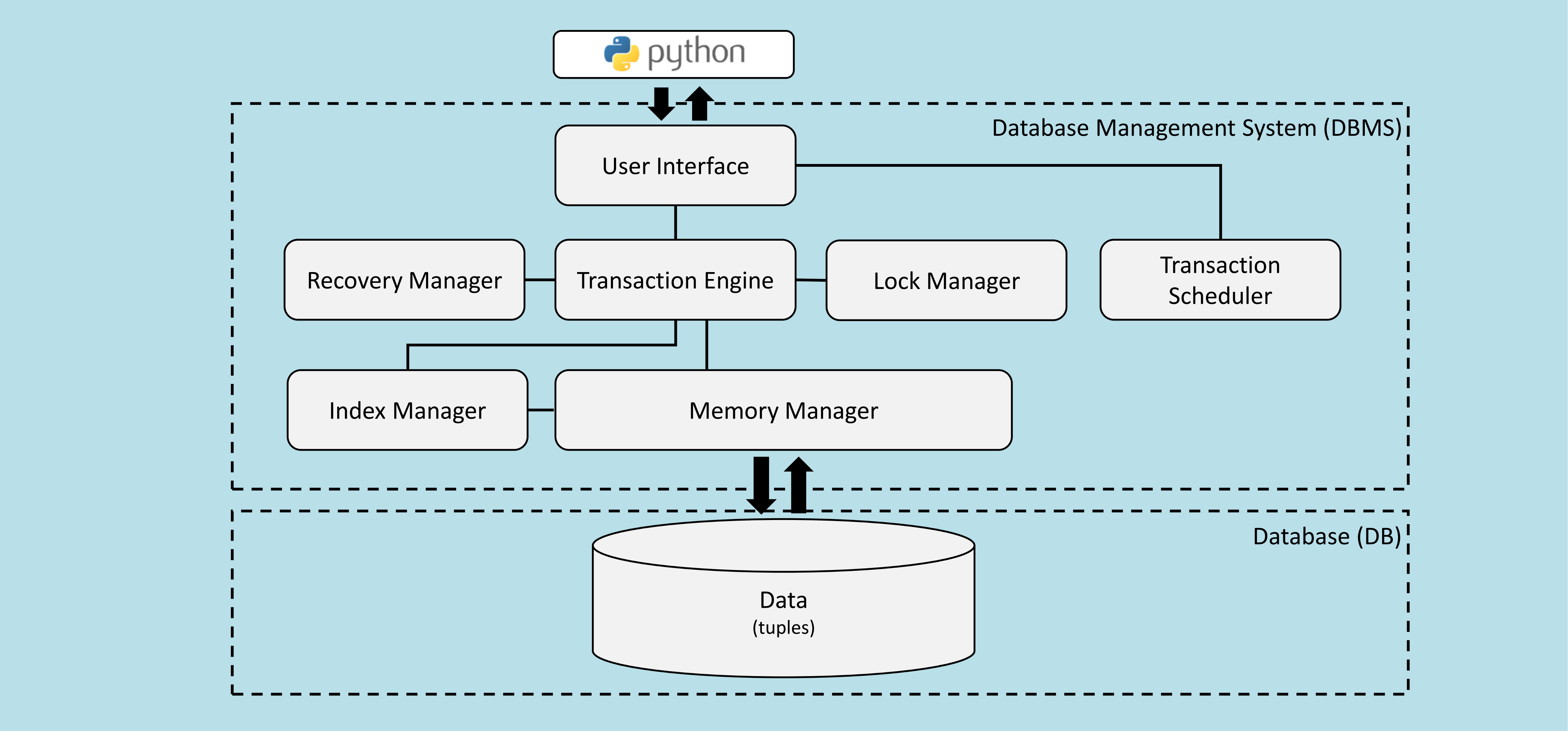
4.1 Example – Google Docs API



4.1 Example – Google Docs API in Action

```
import gspread
from oauth2client.service_account import ServiceAccountCredentials

# Create a client to interact with the Google Drive API
scope = ["https://spreadsheets.google.com/feeds"]
credentials = ServiceAccountCredentials.from_json_keyfile_name("client_secret.json", scope)
client = gspread.authorize(credentials)

sheet = client.open("Agent Secret data").sheet1

# Print content
content = sheet.get_all_records()
print(content)
```

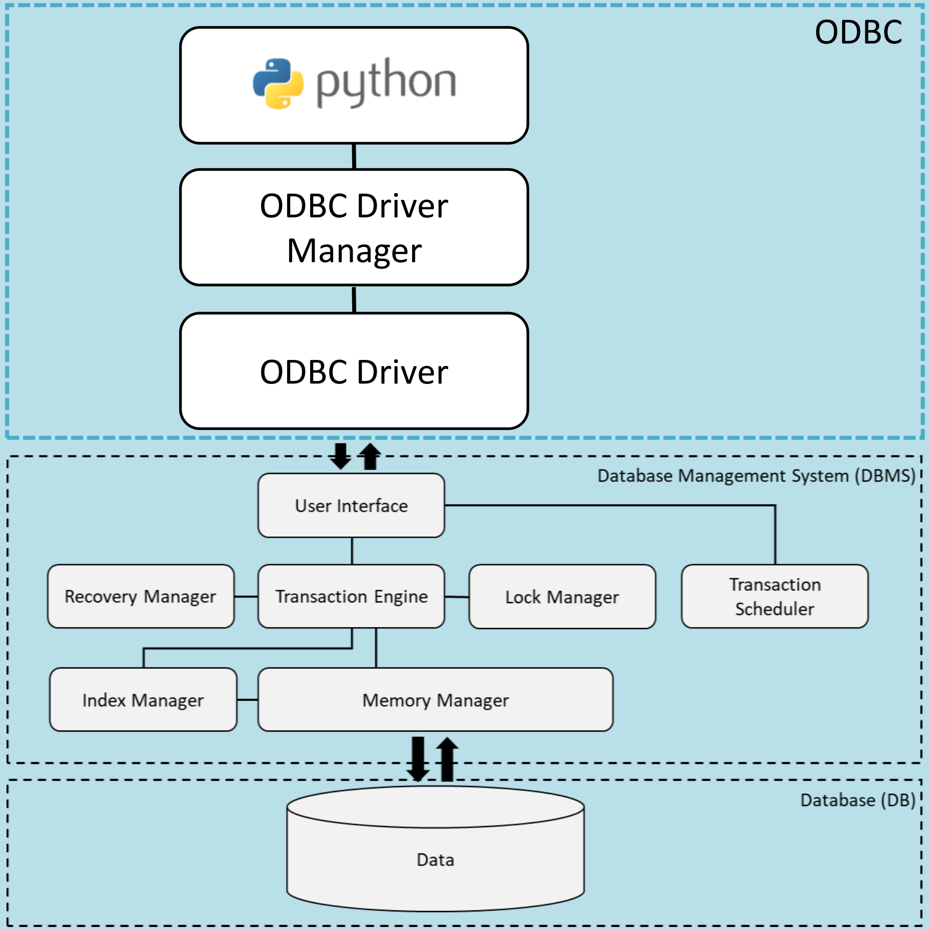
4.1 General Architecture of Database Systems (DBS)



4.1 Using Databases with Python

- There exists two ways to access data in a database management system (DBMS)
  - via database language (direct access)
  - via application program interface (API)
- Best practice is to access your database with low-level APIs (ODBC, JDBC, XDBC etc.)
- Microsoft's ODBC (Open DataBase Connectivity) API is probably the most widely used programming interface
- **ODBC**: API for accessing DBMS

Python Database Access with ODBC



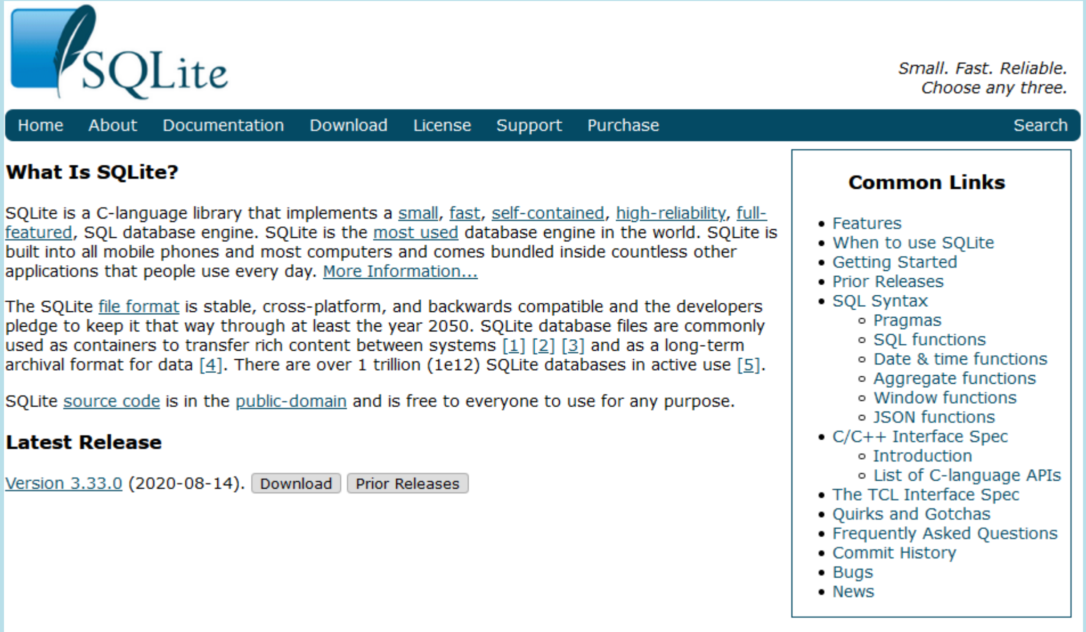
4.1 Database System: SQLite

Characteristics

- Small, fast, self-contained, high-reliability, full-featured, SQL database engine
- SQLite is the most used database engine in the world.

Application

- SQLite has been used with great success as the on-disk file format for desktop applications such as version control systems, financial analysis tools, media cataloging and editing suites, CAD packages, record keeping programs, etc.
- You can easily share and save your application data, and even use it as application file format



+ Cross-platform, and backwards compatible, guaranteed support at least to the year 2050

- Lack of multi-user capabilities, no security capabilities beyond encrypting the database file itself

4.1 SQLite Write Data

- Let us create a database containing some data

```
connection = sqlite3.connect("cars.db")
cursor = connection.cursor()

sql = "CREATE TABLE car_table(" \
      "ID INTEGER PRIMARY KEY, " \
      "model TEXT, " \
      "price REAL, " \
      "acceacceleration TEXT, " \
      "ps REAL)"
cursor.execute(sql)

sql = "INSERT INTO car_table VALUES('1', " \
      "'718 Cayman', 54000, '5.1', 300)"
cursor.execute(sql)
connection.commit()

…

sql = "INSERT INTO car_table VALUES('3', " \
      "'911 Turbo S', 212000, '2.7', 650)"
cursor.execute(sql)
connection.commit()
connection.close()
```

Connect to your database

Run standard SQL on the database system

Commit your changes

Close connection

4.1 SQLite Read Data

- Then you can access your data in your AI application ☺

```
>>> sql = "SELECT * FROM car_table"
>>> cursor.execute(sql)

>>> print(cursor.fetchall())

[(1, '718 Cayman', 54000.0, 4.7, '300'),
(2, '718 Cayman S', 67000.0, 4.6, '350'),
(3, '911 Turbo S', 212000.0, 2.7, '650')]
```

```
>>> sql = "SELECT model, MAX(acceacceleration) FROM car_table"
>>> cursor.execute(sql)
>>> print(cursor.fetchall())

[('718 Cayman', 4.7)]
```

Your turn!

Task

Please setup the SQLite database from our example. You can use the code from git or copy it from the lecture slides. Then solve the following tasks:

- Connect to your database and add a new row `(4, "Taycan Turbo S", 181000.0, 2.8, "761")`
- Print the new table in the console

Outline

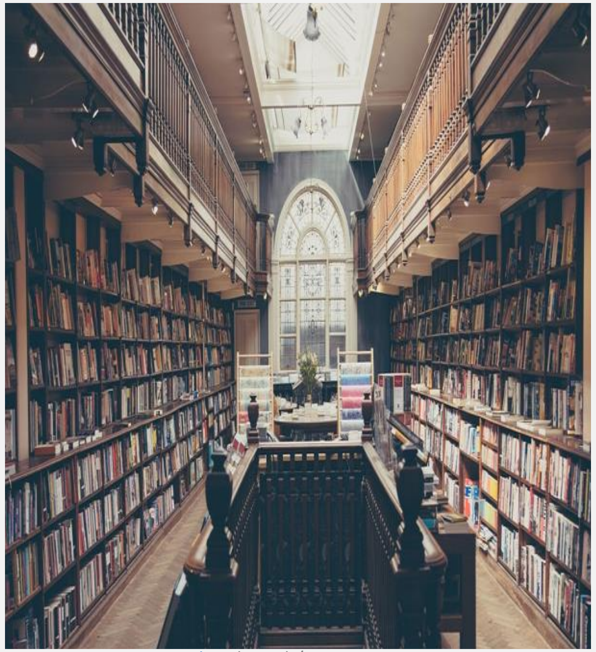


Image source: ↗ Pixabay (2019) / ↗ CC0

▶ **What we will learn:**
- How data and knowledge is handled in AI-based information systems
- Most common methods for exploring the data to get better domain understanding
- Learn to prepare and transform data to make it ready for your AI project
- Deepen your Python programming skills using popular Python modules like Pandas and Scikit-learn for data engineering

▶ **Duration:**
- 135 min

▶ **Relevant for Exam:**
- 4.1-4.5

4.2 Data Exploration

	One Variable	**Multiple Variables**
Descriptive Statistics	▪ Summary statistics	▪ Correlation Coefficient
Visualization	▪ Histogram ▪ Box plot ▪ Bar chart	▪ Correlation matrix ▪ Scatter plot ▪ Time Series line chart ▪ Conditional box plot

Which one of the various data exploration methodologies to use, largely depends on what variable type we are looking at.

4.2 Summary Statistics

```
# summary statistics
import pandas as pd
import numpy as np

df = pd.DataFrame(<your data>)
print df.describe()
```

	Var1	Var2
count	12.00	20.00
mean	45.80	37.00
std	5.00	9.65
min	25.00	10.00
25%	30.25	15.00
50%	39.00	36.50
75%	55.25	40.00
max	60.00	72.02

Measures

- Arithmetic mean:

$$\bar{x} = \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)$$

- Standard Deviation:

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

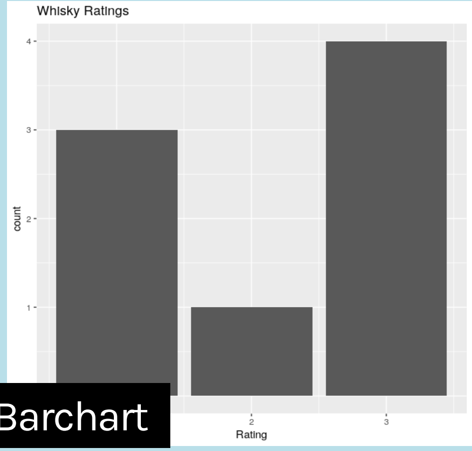
- p-Percentile: $p\%$ of the data is bellow this value

4.2 Compute Summary Statistics

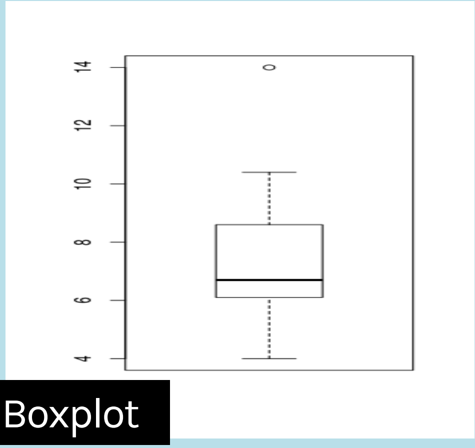
```
data = [1,2,3,4,5]

largest_value = max(data)
smallest_value = min(data)

sorted_values = sorted(data)
second_smallest = sorted_values[1]

mean_value = mean(data)
```

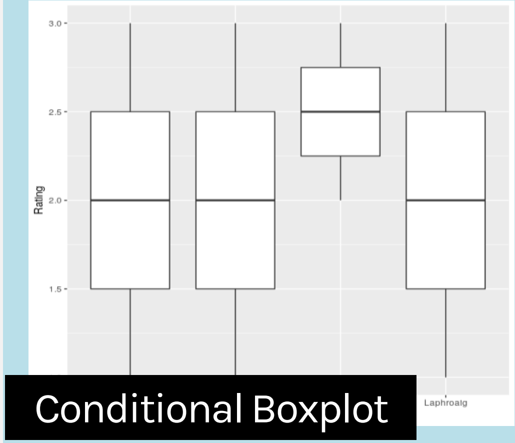
Adapted from matplotlib.org

4.2 Visualizations

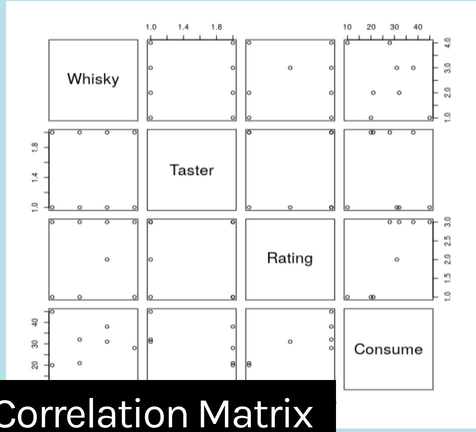


Barchart

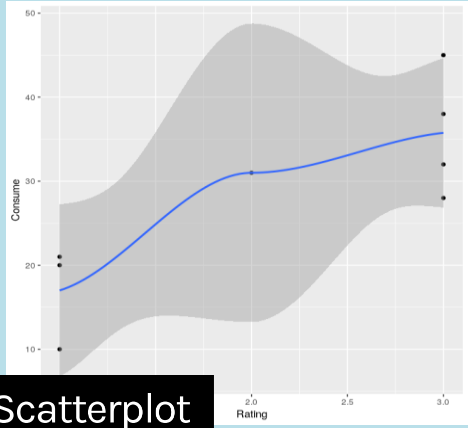

Boxplot



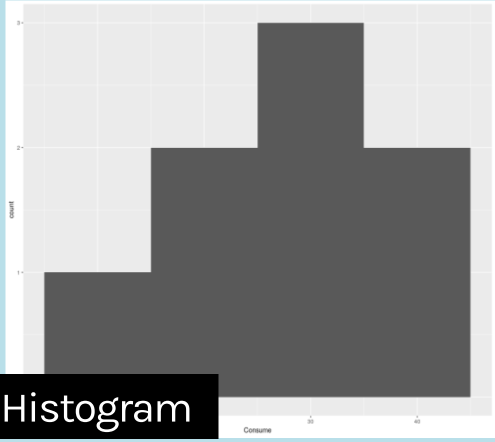
Conditional Boxplot



Correlation Matrix



Scatterplot

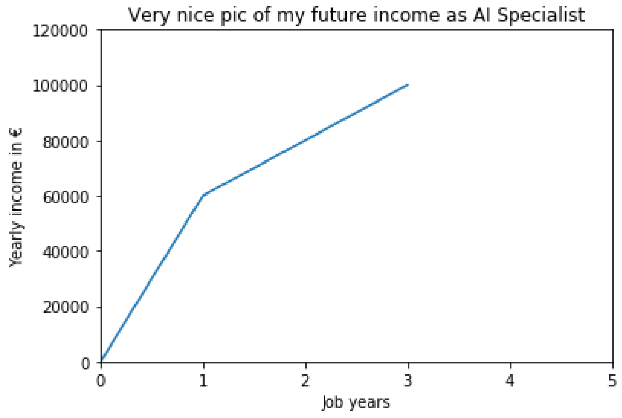


Histogram

4.2 Template: Generating Plots with Matplotlib

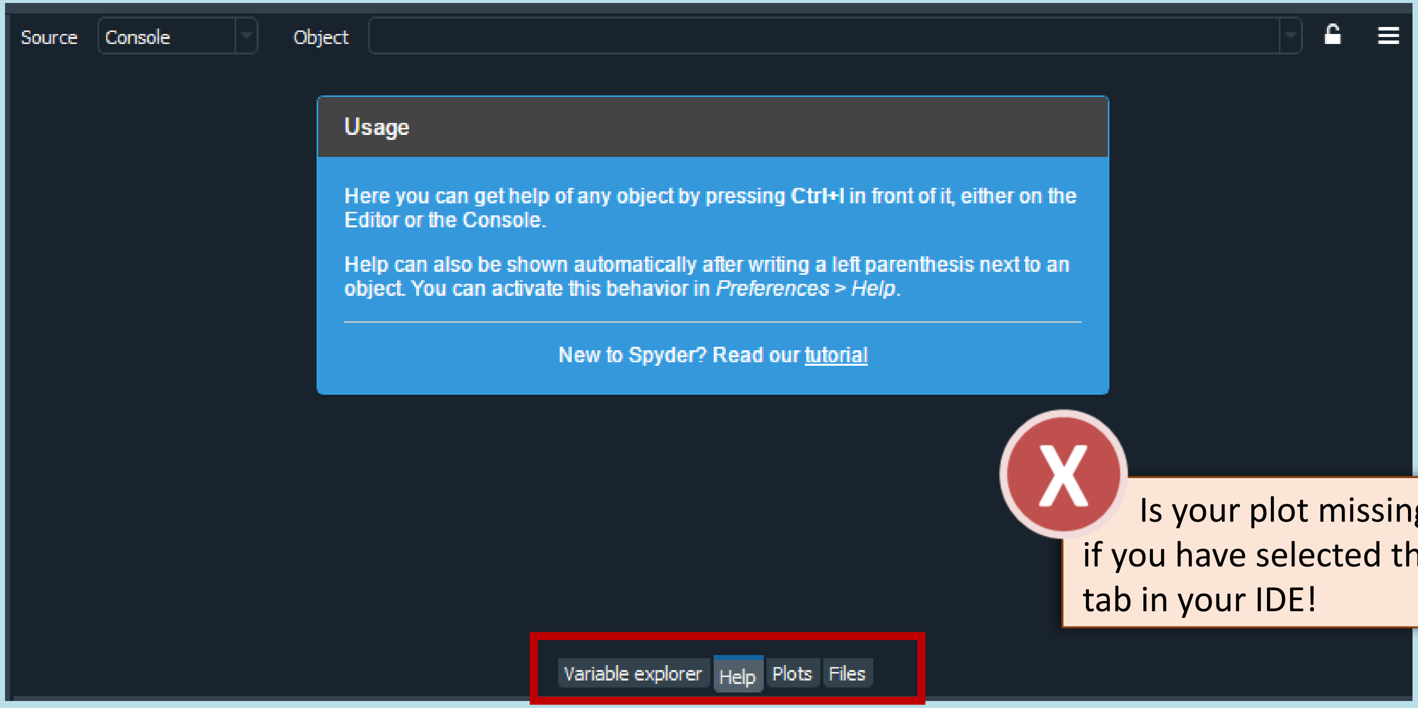
- Matplotlib graphs your data on figures, which can be expanded easily

```
import matplotlib.pyplot as plt
import numpy as np

x = [0, 1, 2, 3]
y = [0, 60000, 80000, 100000]
plt.plot(x,y)

plt.axis([0, 5, 0, 120000])
plt.title("Very nice pic of my future income as AI Specialist")
plt.xlabel("Job years")
plt.ylabel("Yearly income in €")

plt.show()
```

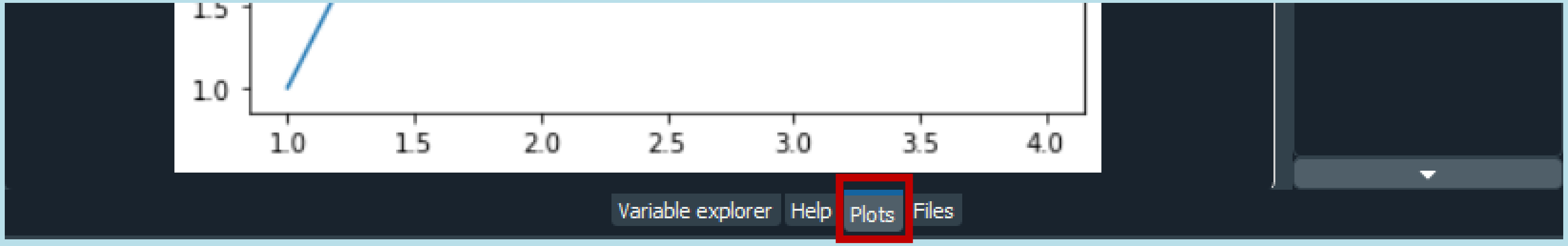


Adapted from matplotlib.org

4.2 Show Plots in Spyder IDE



Is your plot missing? Check out if you have selected the correct tab in your IDE!

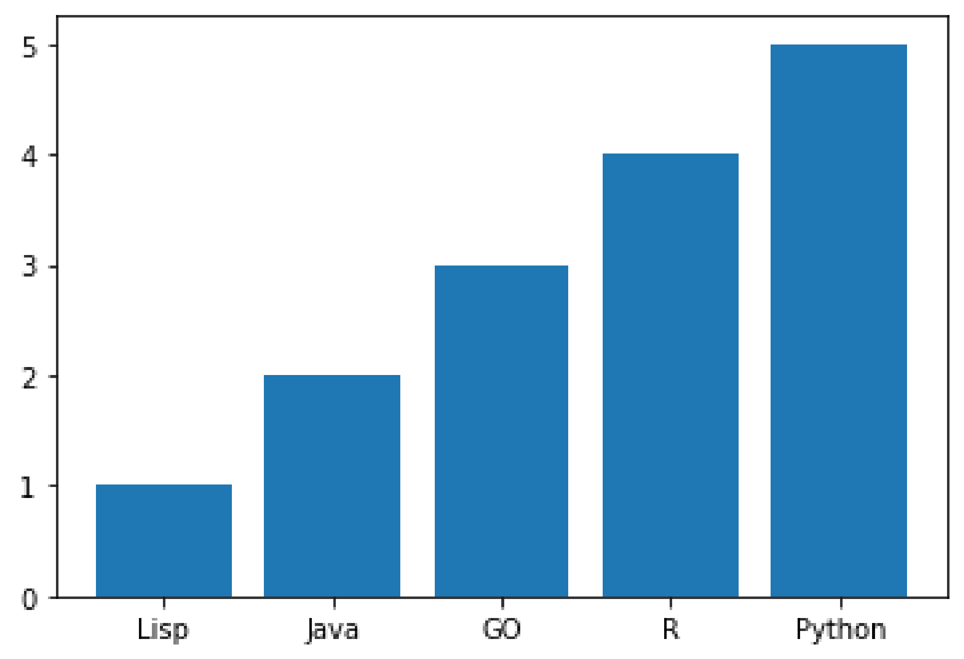


4.2 Bar Chart

Python

```
import numpy as np
import matplotlib.pyplot as plt

x = [1,2,3,4,5]
l = ['Lisp', 'Java', 'GO', 'R','Python']
plt.bar(l, x, align='center')
plt.show()
```

```
# show distinct values of variable
from collections import Counter
c = Counter( dataset["XYZ"])
bar(c.keys(), c.values())
plt.show()
```

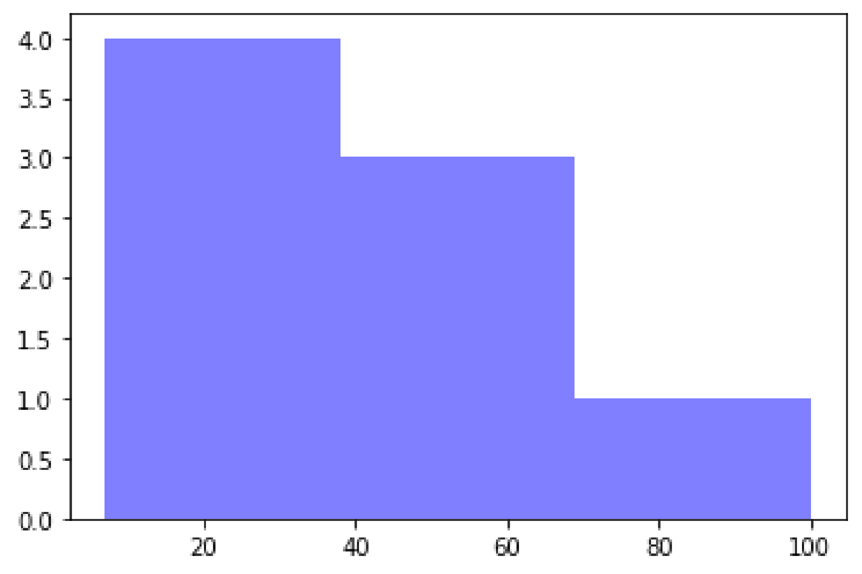


- Frequency of discrete values, the number of items specified by the X-axis
- Most basic visualization for discrete values (like scatterplot for continuous variables)

4.2 Histogram

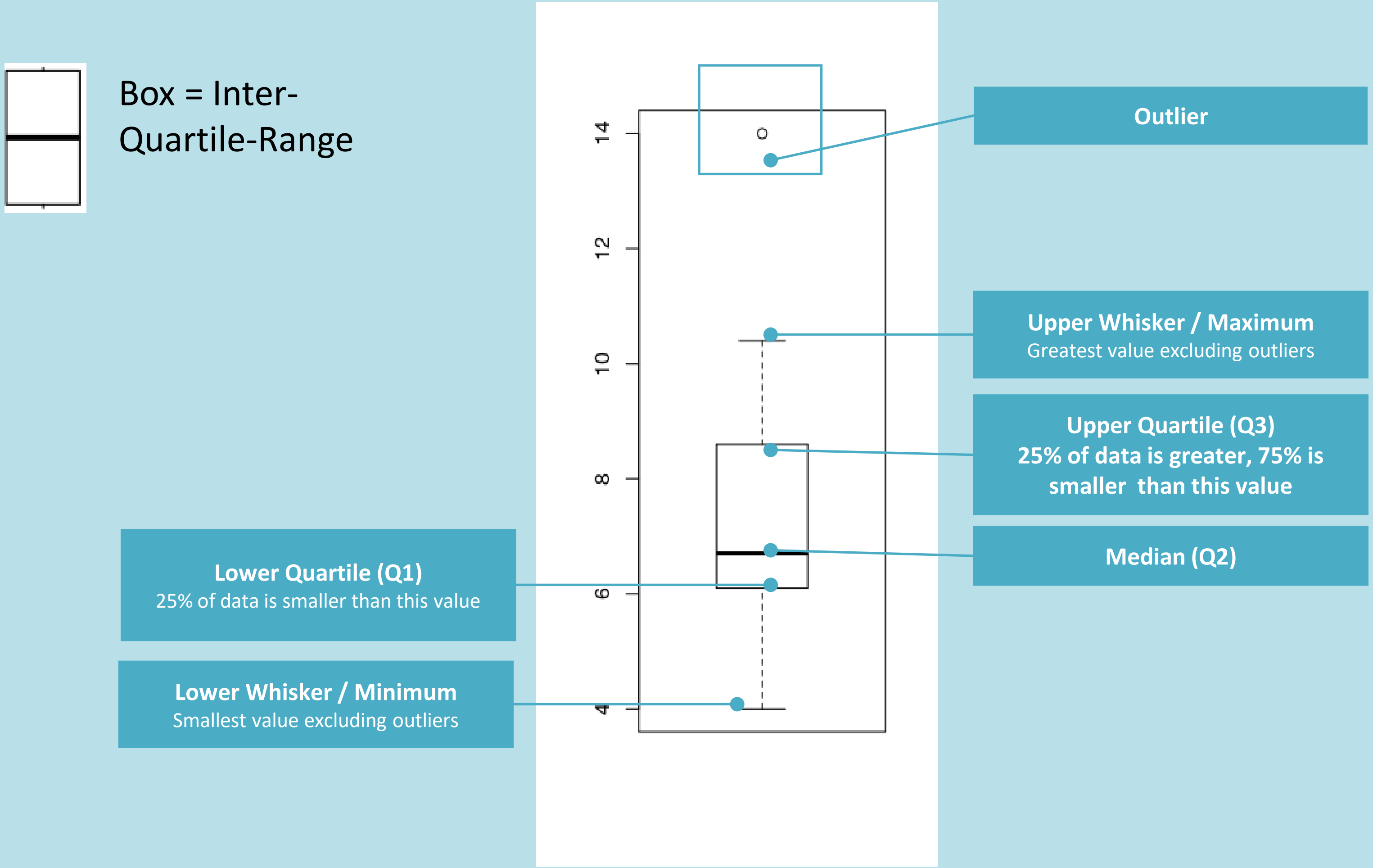
Python

```
import numpy as np
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt

x = [100, 22, 38, 14, 59, 7, 48, 20]
num_bins = 3
n, bins, patches = plt.hist(x, num_bins,
facecolor="blue", alpha=0.5)
plt.show()
```



- Univariate visualization of basic statistical class frequencies to visualize the summary statistics
- Consists of a set of rectangles that reflect the counts (frequencies) of the classes present.
- Normally, an alternative to the boxplot to visualize distributions of your data

4.2 Boxplot - Characteristics



- **Whiskers**: two lines outside the box extend to Minimum and Maximum
- **Inter-Quantile-Range (IQR)** = Q3 – Q1
- **Outlier**: More than 3/2 times of related quartile
- Outliers (Minimum) < Q1 – IQR * 1,5
- Outliers (Maximum) > Q3 + IQR * 1,5
- Minimum / maximum is only shown if value different from Q1/Q3

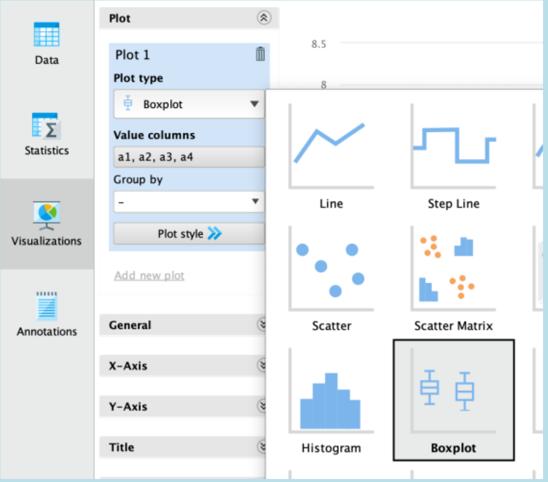
4.2 Boxplot

Python

```
import numpy as np
import matplotlib.pyplot as plt

x = [1,2,3,4]
plt.boxplot(x)
plt.show()
```

Rapidminer

1. Load a dataset
2. In the results view, go to Visualizations
3. Select Boxplot

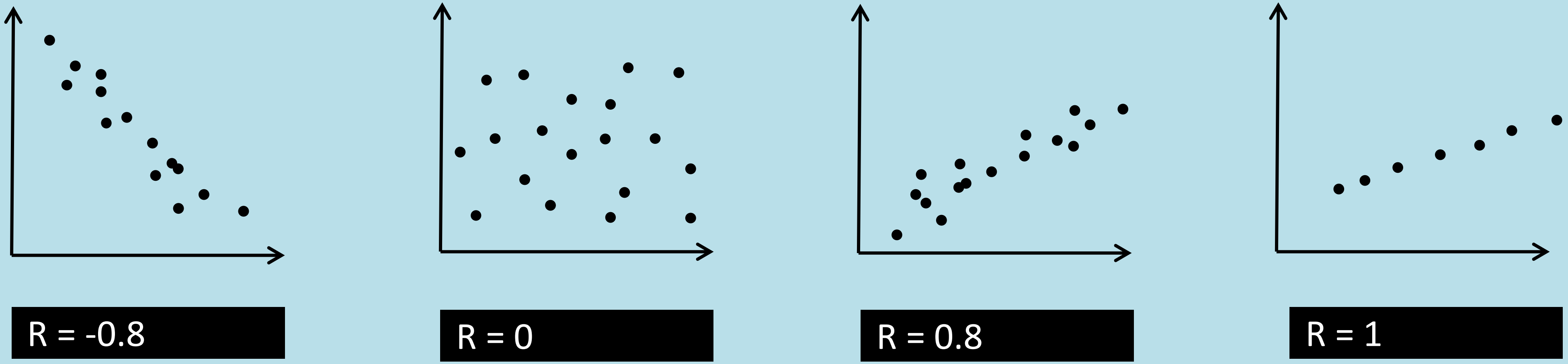


- Data is represented with a box
- The ends of the box are at the first and third quartiles, i.e., the height of the box is IQR
- The median is marked by a line within the box

4.2 Correlation Coefficient

Person Correlation

The (Pearson) correlation coefficient r measures the strength of the linear relationship between two numerical variables. It can take values from -1 (perfect negative linear relationship) and +1 (perfect positive linear relationship).



4.2 Correlation Matrix

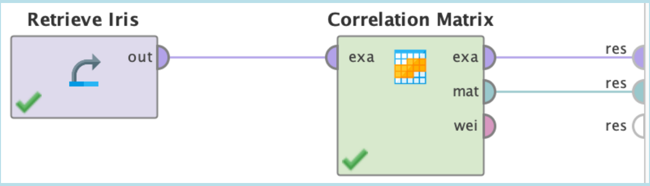
Python

```
import numpy as np
import pandas as pd

data = [[1,2,3,4],[4,3,2,1],[1,1,1,1]]
df = pd.DataFrame(data)

corr = df.corr()
corr.style.background_gradient(cmap='coolwarm')
```

Rapidminer

1. Use Correlation Matrix Operator

Process



Output

Attributes	a1	a2	a3	a4
a1	1	-0.109	0.872	0.818
a2	-0.109	1	-0.421	-0.357
a3	0.872	-0.421	1	0.963
a4	0.818	-0.357	0.963	1

- Visualize relationships between more than two variables. The names of the variables are in the cells of the main diagonal.
- Each off-diagonal cell shows the scatter plot for its row variable (on the y-axis) and its column variable (on the x-axis).

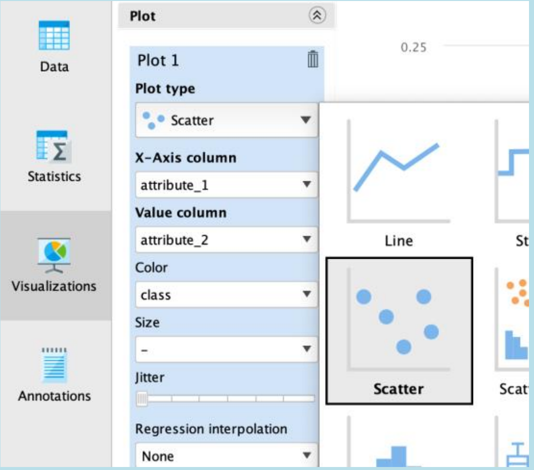
4.2 Scatterplot

Python

```
import numpy as np
import matplotlib.pyplot as plt

x = [1,2,3,4]
y = [4,3,2,1]

plt.scatter(x,y)
plt.show()
```

Rapidminer

1. Load a dataset
2. In the results view, go to Visualizations
3. Select Scatter or Scatter Matrix



- Scatter plots visualize the relationship between two numerical variables to see clusters of points, outliers, etc. Each pair of values is treated as a pair of coordinates and plotted as points in the plane
- help unveiling potential relationships and correlations between two numerical variables.

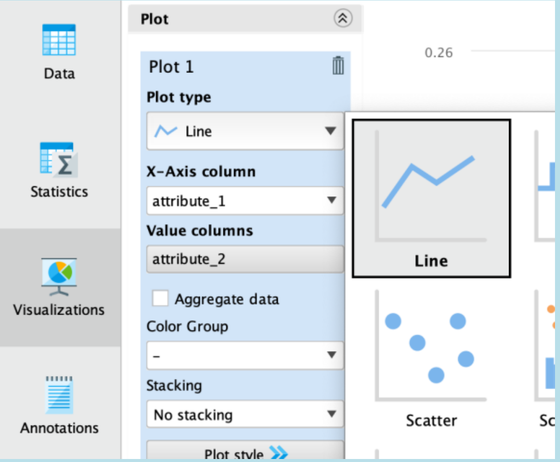
4.2 Time Series Line Chart

Python

```
import numpy as np
import matplotlib.pyplot as plt

x = [2015,2016,2017,2018]
Y = [80.2,67.7,70.5,84.3]

plt.plot(x,y)
plt.gca().set(title="Profit per Year",
xlabel="year", ylabel="profit")
plt.show()
```

Rapidminer

1. Load a dataset
2. In the results view, go to Visualizations
3. Select Line



- If one of the variables in a scatter plot is sorted in order of time, one can use line graphs
- They display the value of a given variable over time (y-axis)
- Time Series charts are a visualization for a numerical variables **tracked over time**. They therefore help identifying the temporal trends of a variable by seeing how it developed over time.

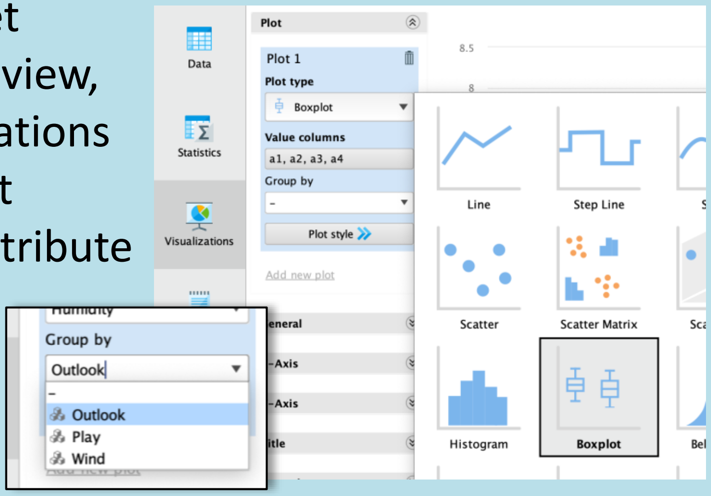
4.2 Conditional Boxplot

Python

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

x = [[22,"A"], [24, "A"], [37, "B"], [27,
"A"],
     [35, "B"], [18, "A"], [40, "B"]]
data = pd.DataFrame(x, columns=["profit",
"country"])
data.boxplot(column='profit',by='country')
plt.show()
```

Rapidminer

1. Load a dataset
2. In the results view, go to Visualizations
3. Select Boxplot
4. Indicate an attribute for Group by



- Conditional box plots extend box plots to a multivariate case
- They require one numerical and one categorical variable and display the box-plot per category
- Conditional box plots are box plots of a numerical variable conditioned on a categorical variable. They are ideal for seeing if there are differences between groups (e.g. are credit limits higher in country X than in country Y?).

Your turn!

Task

Please write or scatch a Python function `percentile(data, p)` that returns the pth-percentile value in the input tuple `data`.

Outline

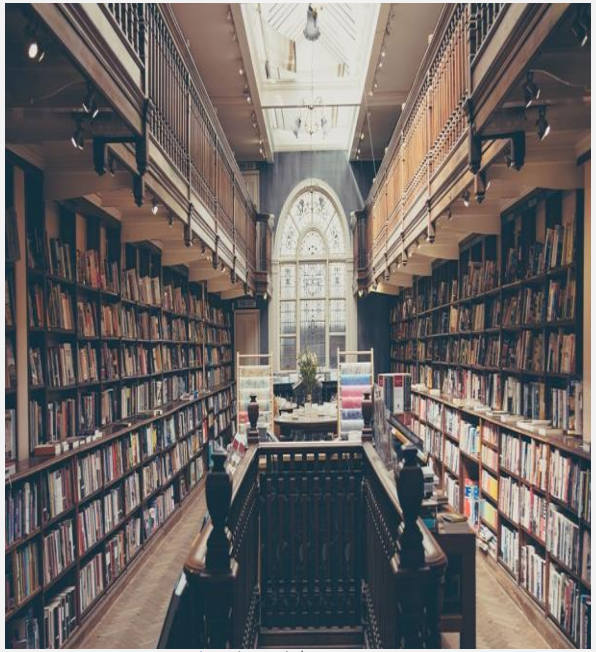


Image source: ↗ Pixabay (2019) / ↗ CC0

▶ **What we will learn:**

- How data and knowledge is handled in AI-based information systems
- Most common methods for exploring the data to get better domain understanding
- Learn to prepare and transform data to make it ready for your AI project
- Deepen your Python programming skills using popular Python modules like Pandas and Scikit-learn for data engineering

▶ **Duration:**

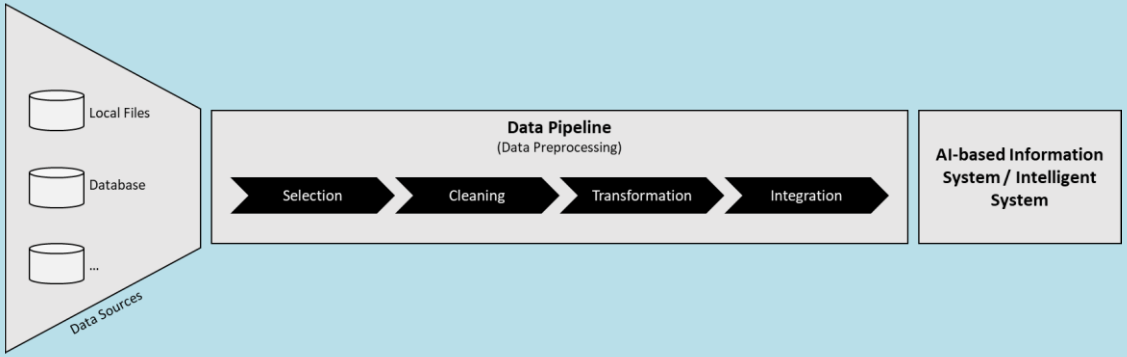
- 135 min

▶ **Relevant for Exam:**

- 4.1-4.5

4.3 Best Practices to Handle Different Data Sources

- As a data engineer you want to prepare the data for the AI application. But how should the data structure look like?



- **Best Practice**: Define a general data format and scheme to integrate datasets from different sources

- For that purpose you have to handle your data and make different aggregation, generalizations or other transformations

Tidy Data

Features (columns) ◄ ►

$$X_{nxm} = \begin{bmatrix} X_{11} & \dots & X_{1j} & \dots & X_{1m} \\ \dots & \dots & \dots & \dots & \dots \\ X_{i1} & \dots & X_{ij} & \dots & X_{im} \\ \dots & \dots & \dots & \dots & \dots \\ X_{n1} & \dots & X_{nj} & \dots & X_{nm} \end{bmatrix}$$

Objects (rows) ▲▼

Data Set: We usually need data in table or matrix form with the specific row-colum characteristics (**tidy data**)

Characteristics

- One **object per row** (or sample, observation, subject, record, element, case)
  - Objects are described by features (or attributes, variables, fields)
  - Indexed by i =1, …., n
  - ***Example***: *customers, store_items*
- One **feature per column**
  - Represents characteristics of an object
  - Indexed by j = 1, …, m
  - ***Example***: *customer_ID, name, age, gender, last_transaction_date, last_transaction_volume, …*

Adapted from Wickham H (2014)

4.3 Data Frame

D

Data Frame (Python 3)

A data frame is a two-dimensional data structure, where data is aligned in a tabular fashion in rows and columns.

```
pandas.DataFrame(data, index, columns, dtype, copy)
```

Features (columns) ◄ ►

ID	CAR	RATING
1	Model X	3
2	Mercedes S-class	4
4	Taycan	5
5	911 GT	5
8	BMW i3	3

Objects (rows) ▲▼

4.3 DataFrames with Pandas

DataFrame()

```
pandas.DataFrame(data, index, columns, dtype, copy)
```

Parameters

Data (Input)	Input data, which can have various forms like series, map, lists, dict, constants and also another dataframes.
index	You can set an individual type of index for the row labels, the default index if no index is passed is `np.arrange(n)`.
columns	For column labels, the optional default syntax is - `np.arrange(n)`.
dtype	Here you can specify the data type of each column.
copy	Use this for copying of data, however the default is `False`.

4.3 DataFrames with Pandas (Example)

- You retrieve values in a data structure by declaring an index inside a square bracket "[]" operator.

```
>>> import pandas as pd
>>> df = pd.DataFrame()
>>> print(df)

Empty DataFrame
Columns: []
Index: []
```

```
>>> import pandas as pd
>>> data = [4, 8, 15, 16, 23, 42]
>>> df = pd.DataFrame(data)
>>> print(df)

    0
0   4
1   8
2  15
3  16
4  23
5  42
```

- The same works for keys with dictionaries

? What is the difference between dictionaries and data frames? Why do we need data frames?

4.3 Pandas DataFrames Indexing

```
import pandas as pd
speed = [290, 330, 345]
car = [718, 911, 918]
df = pd.DataFrame()
df["CAR"] = car
df["SPEED"] = speed
```

INDEX	CAR	SPEED
0	718	290
1	911	330
2	918	345

Nam	Type	Size	
car	list	3	[718, 911, 918]
df	DataFrame	(3, 2)	Column names: CAR, SPEED
speed	list	3	[290, 330, 345]

```
df["SPEED"]
```

290
330
345

```
df[0:2]
```

CAR	SPEED
718	290
911	330

Pandas offers many ways of multi-axis indexing. However, the most popular one is .loc-indexing. Check out the package documention for other indexing methods like iloc.

```
df.loc[1:]
```

startindex

stop

INDEX	CAR	SPEED
1	911	330
2	918	345

```
df.loc[2]
```

INDEX	CAR	SPEED
2	918	345

```
df.loc[:, 'CAR']
```

CAR
718
911
918

```
df.loc[1:2, 'CAR':'SPEED']
```

CAR	SPEED
911	330
918	345

4.3 Data Handling Using Pandas

Create	Apply	Sort	Melt
View	Append	Clean	Pivot
Insert	Join	Fill	
Filter	Group	Rotate	

All coding examples are available online in the course repository.

4.3 Create New DataFrame

```
data = {"Model": ["718", … "911"],
      "Type": ["Cayman", … "Targa 4S"],
      "PS": [300, … 385],
      "Price":[54900, … 140000]}

df = pd.DataFrame(data=data,
columns=["Type","Model", "PS","Price"])
```

```
 Type     Model    PS    Price
0       Cayman    718   300    54900
1     Cayman S    718   350    67000
2       Boxter    718   300    56900
3     Boxter S    718   350    69000
4      Carrera    911   385   103000
5   Carrera 4S    911   450   126000
6     Targa 4S    911   385   140000
```

Model	Type	PS	Price
718	*Cayman*	*300*	*54900*
718	*Cayman S*	*350*	*67000*
718	*Boxter*	*300*	*56900*
718	*Boxter S*	*350*	*69000*
911	*Carrera*	*385*	*103000*
911	*Carrera 4S*	*450*	*126000*
911	*Targa 4S*	*385*	*140000*

- Create new Pandas DataFrames

4.3 View DataFrame

```
>>> df.head()

 Type  Model    PS    Price
0     Cayman   718   300    54900
1   Cayman S   718   350    67000
2     Boxter   718   300    56900
3   Boxter S   718   350    69000
4    Carrera   911   385   103000
```

```
>>> df.tail()

 Type  Model    PS    Price
2        Boxter   718   300    56900
3      Boxter S   718   350    69000
4       Carrera   911   385   103000
5   Carrera 4S   911   450   126000
6     Targa 4S   911   385   140000
```

Model	Type	PS	Price
718	*Cayman*	*300*	*54900*
718	*Cayman S*	*350*	*67000*
718	*Boxter*	*300*	*56900*
718	*Boxter S*	*350*	*69000*
911	*Carrera*	*385*	*103000*
911	*Carrera 4S*	*450*	*126000*
911	*Targa 4S*	*385*	*140000*

- View the first 5 or the last 5 entries of your dataframe

4.3 Insert New Column in DataFrame

```
df["Wishlist"] = [1,0,0,0,0,1,0]
```

```
>>> df

         Type Model   PS   Price  Wishlist
0      Cayman   718  300   54900         1
1    Cayman S   718  350   67000         0
2      Boxter   718  300   56900         0
3    Boxter S   718  350   69000         0
4     Carrera   911  385  103000         0
5  Carrera 4S   911  450  126000         1
6    Targa 4S   911  385  140000         0
```

Model	Type	PS	Price	Wishlist
718	*Cayman*	*300*	*54900*	*1*
718	*Cayman S*	*350*	*67000*	*0*
718	*Boxter*	*300*	*56900*	*0*
718	*Boxter S*	*350*	*69000*	*0*
911	*Carrera*	*385*	*103000*	*0*
911	*Carrera 4S*	*450*	*126000*	*1*
911	*Targa 4S*	*385*	*140000*	*0*

- Insert new column into the existing dataframe

4.3 Filter DataFrame

```
>>> df["Wishlist"] == 1

0     True
1    False
2    False
3    False
4    False
5     True
6    False
Name: Wishlist, dtype: bool
```

```
>>> df[df["Wishlist"] == 1]

Type Model    PS    Price   Wishlist
0       Cayman   718  300    54900         1
5  Carrera 4S   911  450   126000         1
```

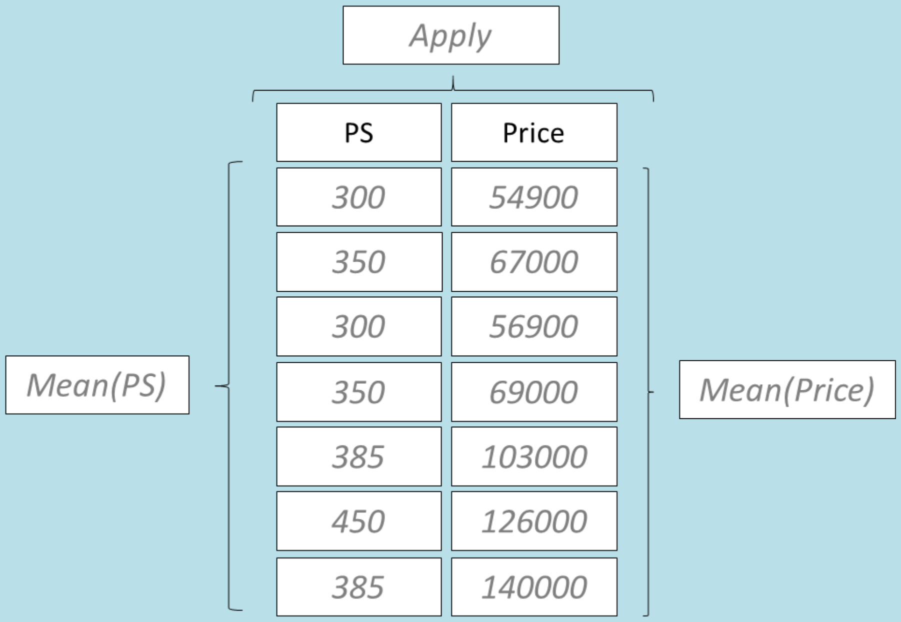
Model	Type	PS	Price	Wishlist
718	*Cayman*	*300*	*54900*	*1*
911	*Carrera 4S*	*450*	*126000*	*1*

- Filter dataframe based on variable values

4.3 `Apply` Functions on Columns in `DataFrame`

```
def calculate_mean(row_col):
    return row_col.mean()

df[["Price","PS"]].apply(calculate_mean,axis=0)
```

```
Price    88114.285714
PS         360.000000
dtype: float64
```



- Apply a function on specific columns to compute new values/features or aggregations

4.3 Append a New Row

```
new_data = [["Taycan Turbo S", "Taycan",
761, 181000]]

new_row = pd.DataFrame(data=new_data,
columns=["Type", "Model", "PS","Price"])

df.append(new_row, ignore_index=True)
```

```
>>> df.append(new_row, ignore_index=True)

             Type   Model   PS   Price  Wishlist
0          Cayman     718  300   54900       1.0
1        Cayman S     718  350   67000       0.0
2          Boxter     718  300   56900       0.0
3        Boxter S     718  350   69000       0.0
4         Carrera     911  385  103000       0.0
5      Carrera 4S     911  450  126000       1.0
6        Targa 4S     911  385  140000       0.0
7  Taycan Turbo S  Taycan  761  181000       NaN
```

Model	Type	PS	Price	Wishlist
718	*Cayman*	*300*	*54900*	*1*
718	*Cayman S*	*350*	*67000*	*0*
718	*Boxter*	*300*	*56900*	*0*
718	*Boxter S*	*350*	*69000*	*0*
911	*Carrera*	*385*	*103000*	*0*
911	*Carrera 4S*	*450*	*126000*	*1*
911	*Targa 4S*	*385*	*140000*	*0*

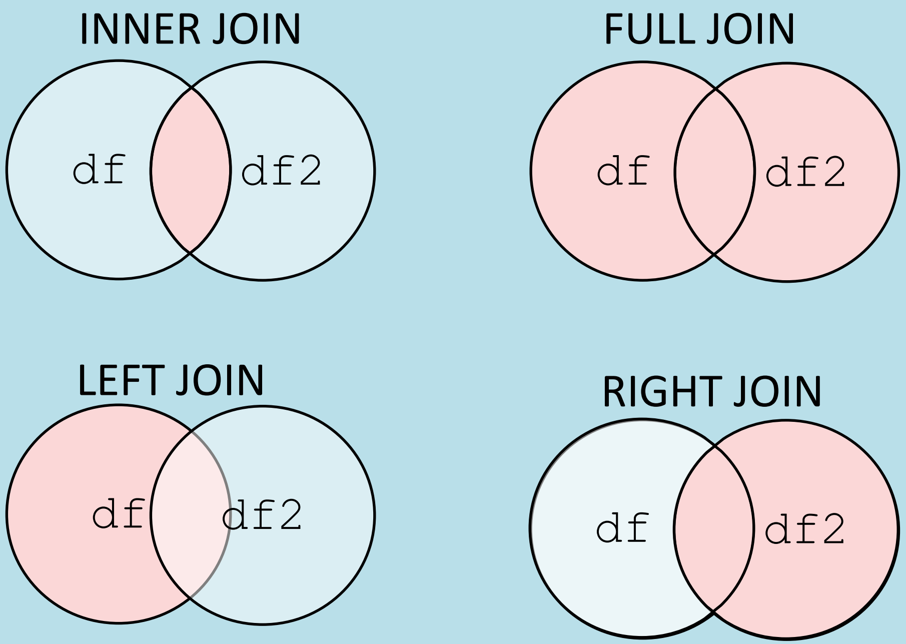
+

Taycan	*Taycan Turbo S*	*761*	*181000*	*NaN*

- Append new rows at the end of the dataframe, missing columns get `NaN` values

4.3 Join different DataFrames

```
data2 = {"Type" : ["Carrera", "Carrera
4S", "Targa 4S", "Panamera"],
        "Ranking" : [1,2,3,4]}

df2 = pd.DataFrame(data=data2,
columns=["Type","Ranking"])

pd.merge(df, df2, on="Type")
pd.merge(df, df2, on="Type", how="inner")
pd.merge(df, df2, on="Type", how="outer")
pd.merge(df, df2, on="Type", how="right")
```



- Pandas supports to join different dataframes

4.3 Group Data

```
>>>
df[["Model","Price","PS"]].groupby("Model
").mean()

         Price          PS
Model
718      61950.0  325.000000
911     123000.0  406.666667
```

Model	PS	Price
718	300	54900
	350	67000
	300	56900
	350	69000
911	385	103000
	450	126000
	385	140000

- Group data by specific column-based groups to compute new values

4.3 Sort DataFrame based on Columns

```
df.sort_values(by="PS")

         Type Model   PS   Price  Wishlist
0      Cayman   718  300   54900         1
2      Boxter   718  300   56900         0
1    Cayman S   718  350   67000         0
3    Boxter S   718  350   69000         0
4     Carrera   911  385  103000         0
6    Targa 4S   911  385  140000         0
5  Carrera 4S   911  450  126000         1
```

```
df.sort_values(by="PS", ascending=False)

         Type Model   PS   Price  Wishlist
5  Carrera 4S   911  450  126000         1
4     Carrera   911  385  103000         0
6    Targa 4S   911  385  140000         0
1    Cayman S   718  350   67000         0
3    Boxter S   718  350   69000         0
0      Cayman   718  300   54900         1
2      Boxter   718  300   56900         0
```

Model	Type	PS	Price	Wishlist
718	Cayman	[illegible]	54900	1
718	Cayman S	[illegible]	67000	0
718	Boxter	[illegible]	56900	0
718	Boxter S	[illegible]	69000	0
911	Carrera	[illegible]	103000	0
911	Carrera 4S	[illegible]	126000	1
911	Targa 4S	[illegible]	140000	0

4.3 Simple Data Cleaning with `dropna()`

```
data2 = {"Type" : ["Carrera", "Carrera 4S",
"Targa 4S", "Panamera"], "Ranking" : [1,2,3,4]}

df2 = pd.DataFrame(data=data2,
        columns=["Type","Ranking"])

df_new = pd.merge(df, df2, on="Type",
        how="outer")

df_new.dropna()
```

Model	Type	PS	Price	Wishlist	Ranking
718	Cayman	300	54900	1	NaN
718	Cayman S	350	67000	0	NaN
718	Boxter	300	56900	0	NaN
718	Boxter S	350	69000	0	NaN
911	Carrera	385	103000	0	0
911	Carrera 4S	450	126000	1	1
911	Targa 4S	385	140000	0	0
Panamera	NaN	NaN	NaN	4.0	[illegible]

Model	Type	PS	Price	Wishlist	Ranking
911	Carrera	385	103000	0	0
911	Carrera 4S	450	126000	1	1
911	Targa 4S	385	140000	0	0

4.3 `Fill` Missing Data in `DataFrame`

```
df_new.fillna(0)
```

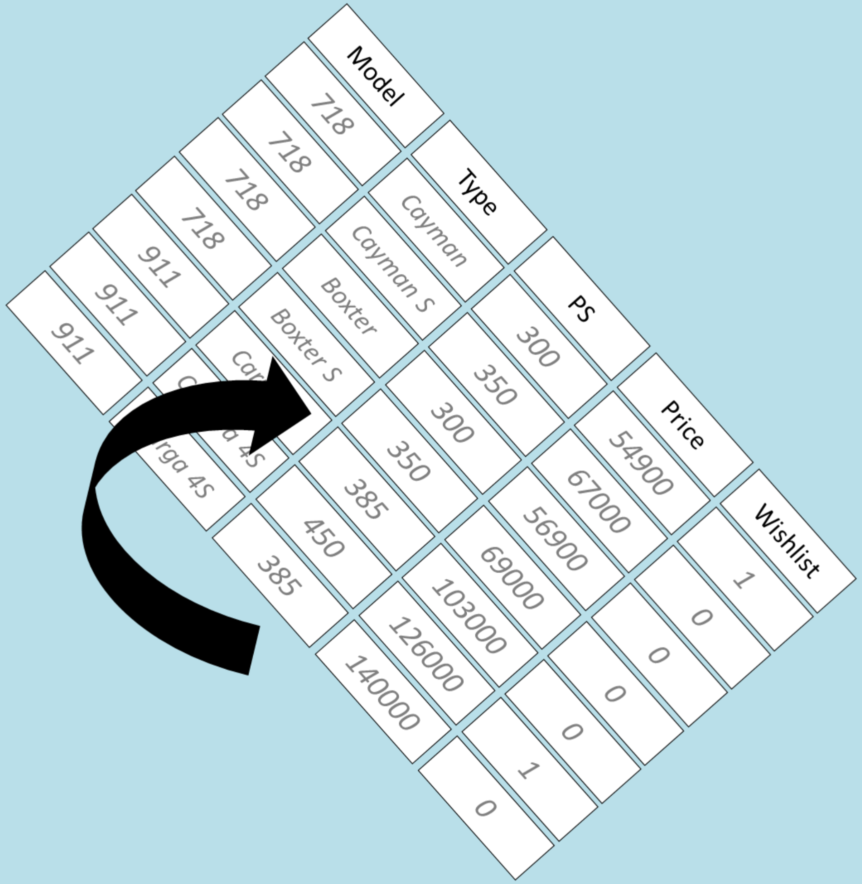
Model	Type	PS	Price	Wishlist	Ranking	
718	Cayman	300	54900	1	NaN	X
718	Cayman S	350	67000	0	NaN	X
718	Boxter	300	56900	0	NaN	X
718	Boxter S	350	69000	0	NaN	X
911	Carrera	385	103000	0	0	✓
911	Carrera 4S	450	126000	1	1	✓
911	Targa 4S	385	140000	0	0	✓
Panamera	NaN	N	NaN	4.0	4.0	X

Model	Type		Price	Wishlist	Ranking
718	Cayman		54900	1	0
718	Cayman S	350	67000	0	0
718	Boxter	300	56900	0	0
718	Boxter S	350	69000	0	0
911	Carrera	385	103000	0	0
911	Carrera 4S	450	126000	1	1
911	Targa 4S	385	140000	0	0
Panamera	0	0	0	4.0	4.0

4.3 Rotate Data in DataFrame

```
>>> df.T

               0          1        2          3       4            ...
Type       Cayman  Cayman S   Boxter  Boxter S ...
Model         718        718      718        718     911
PS            300        350      300        350     385
Price       54900      67000    56900      69000  103000
Wishlist        1          0        0          0       0
```

Generally speaking: Do not use this function, you will transform your tidy data into a useless format. We only use it for mathematical computation!



- Switch rows and columns

4.3 Melt DataFrame

```
df["ID"] = [1,2,3,4,5,6,7]

# Single ID
df_melted = pd.melt(df, id_vars=["ID"],
                    value_vars=["PS", "Price"],
                    var_name="Characteristics",
                    value_name="Value")

# Multiple ID vars
df_melted = pd.melt(df,
id_vars=["Model","Type"],
                    value_vars=["PS", "Price"],
                    var_name="Characteristics",
                    value_name="Value")
```

```
print(df_melted)   # single ID

  ID Characteristics   Value
0    1                PS      300
1    2                PS      350
2    3                PS      300
```

ID	PS	Price
1	300	54900
2	350	67000
3	300	56900
....	...	...
x	385	103000
y	450	126000
...	...	...

ID	Charact	Value
1	PS	300
2	PS	350
3	PS	300
	...	...
1	Price	54900
2	Price	67000
...	...	...

- Transform wide to long data

4.3 Pivot DataFrame

```
df_unmelted = df_melted.pivot(index=["ID"],
columns=["Characteristics"])

df_melted.pivot_table(index=["Model","Type"],
columns=["Characteristics"],
values=["Value"])
```

ID	PS	Price
1	*300*	*54900*
2	*350*	*67000*
3	*300*	*56900*
....	*...*	*...*
x	*385*	*103000*
y	*450*	*126000*
...	*...*	*...*

ID	Charact	Value
1	PS	*300*
2	PS	*350*
3	PS	*300*
	...	*...*
1	Price	*54900*
2	Price	*67000*
...	*...*	*...*

- Transform long to wide data

Your turn!

Task

Please discuss with your neighbors:

- What is a dataframe, and how does it relate to “tidy” data
- How can you combine different data source into a dataset for your AI application? Which Pandas “commands” will you need for this?

Outline

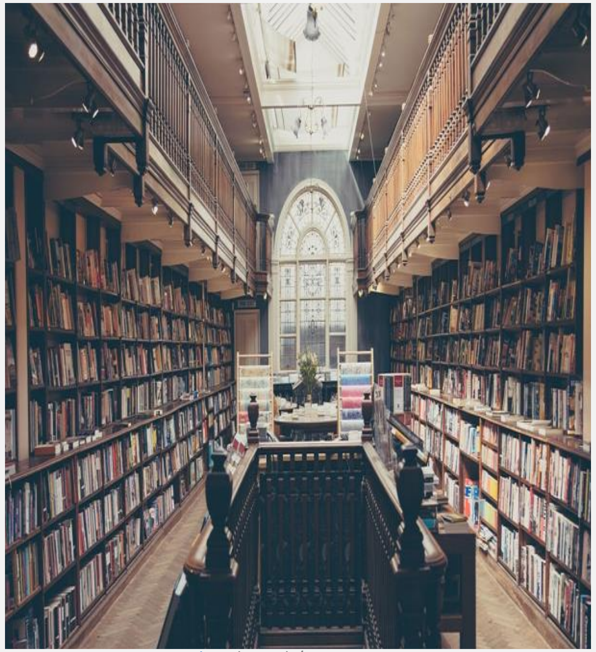


Image source: ↗ Pixabay (2019) / ↗ CC0

▶ **What we will learn:**

- How data and knowledge is handled in AI-based information systems
- Most common methods for exploring the data to get better domain understanding
- Learn to prepare and transform data to make it ready for your AI project
- Deepen your Python programming skills using popular Python modules like Pandas and Scikit-learn for data engineering

▶ **Duration:**

- 135 min

▶ **Relevant for Exam:**

- 4.1-4.5

4.4 Why Data Preprocessing and Cleaning?

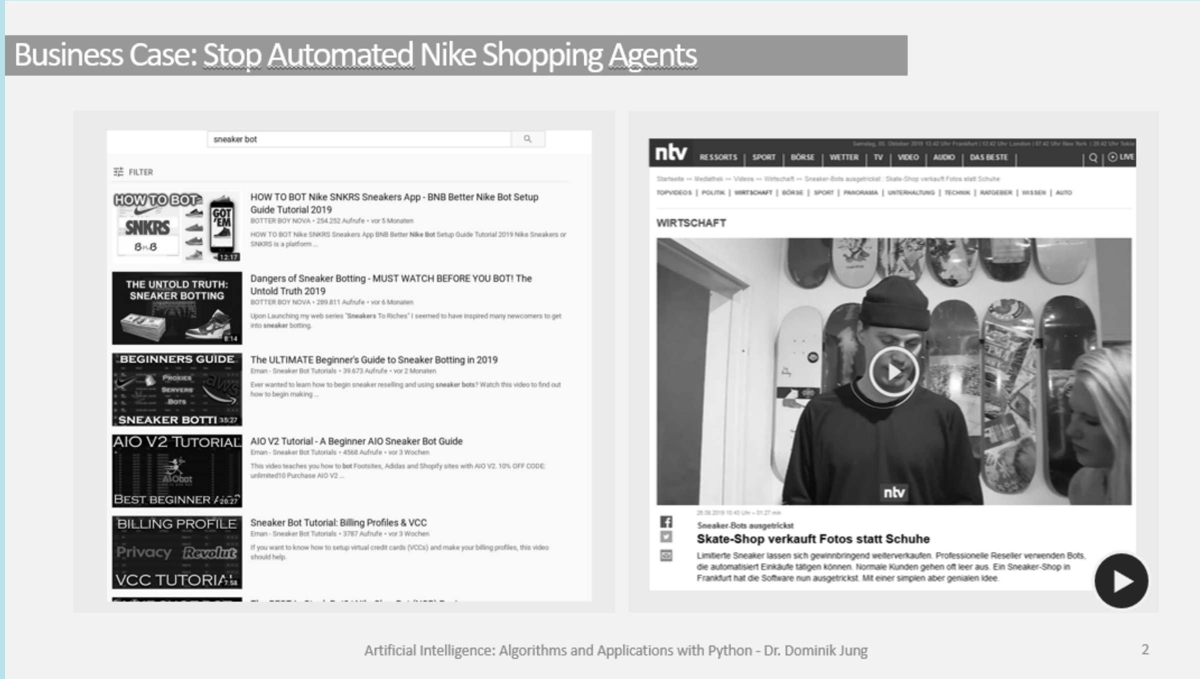
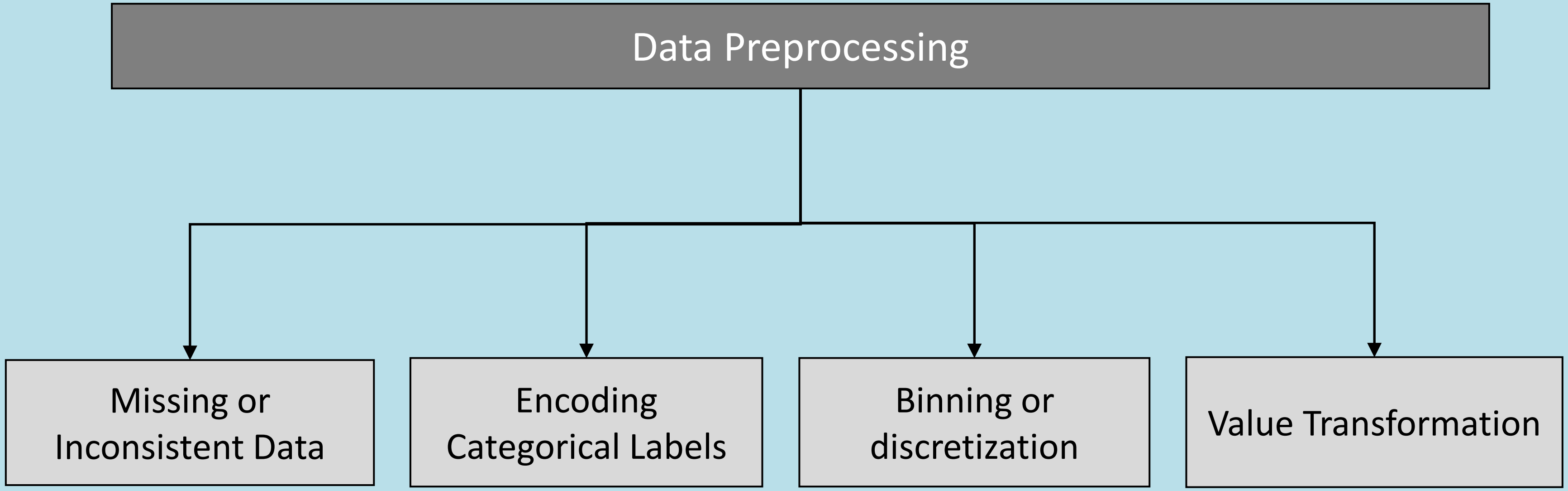


Image source: ↗ [Pixabay](https://pixabay.com) (2019) / ↗ [CC0]()

- **GIGO-principle**: Messy input data produces useless output
- Relying on the GIGO-principle, data cleaning is the process of removing errors and inconsistencies in your database.

„Garbage“ → „Agent“ → „Garbage“

4.4 General Data Pre-processing Challenges



4.4 Straight forward Stratgies to handle missing values

In real world datasets, we often find many missing values. There are three strategies to handle missing values:

- Ignoring the tuple
  - Ignore the row/column with missing values.
- Fill the values manually or by a constant
  - Fill in missing values based on your business knowledge
  - Taking the most frequent
- Fill by a computed value
  - Use the attribute mean, or the class mean
  - Use the most probably value for an attribute derived by some learning algorithm from other attributes (e.g. the default)

4.4 Handle Missing or Inconsistent Data

- The SimpleImputer class from scikit-learn provides basic strategies for handling missing values.
- For instance, missing values can be estimated by using the statistics (e.g. mean) of each column in which the missing values are located.

```
import numpy as np
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
```

```
imputer = SimpleImputer(strategy="most_frequent")
```

4.4 Encoding Categorial Labels

- Usually our data is not always continuous, our values appear as categorical in textual type like: Route {Yellowstone, Grand Canyon, etc.}

- Encoding or continuization is the transformation of categorical variables to binary or numerical counterparts so that computers (or our agents) can handle them

- The most common are:
  - Binary
  - Target-based Encoding
  - One-hot Encoding

4.4 Binary Encoding

- Numerization of categorical variables by taking the values 0 or 1 to indicate the absence or presence of each category.
- If the categorical variable has *k* categories we would need to create *k* binary variables (technically speaking, *k-1* would suffice).

Car
911
718
Taycan
718

➡

Order
1
2
3
2

➡

Binary
01
10
11
10

➡

ENCODED

Car_1	Car_2
0	1
1	0
1	1
1	0

? What is the main problem of this approach?

4.4 Binary Encoding

- You can run binary encoding in Python with:

```
from sklearn.preprocessing import BinarayEncoder

encoder = BinarayEncoder()
encoder.fit_transform(data)
```

4.4 Target-based Encoding

- Target-based encoding is numerization of categorical variables via target.
- In this method, we replace the categorical variable with just one new numerical variable and replace each category of the categorical variable with its corresponding probability of the target (if categorical) or average of the target (if numerical).
- The main drawbacks of this method are its dependency to the distribution of the target, and its lower predictability power compare to the binary encoding method.

4.4 Target-based Encoding

Car	Target
911	1
718	1
Taycan	1
718	0
Taycan	0
Taycan	1

	Target		
Car	**0**	**1**	**Probability**
911	0	1	100 %
718	1	1	50 %
Taycan	1	2	66 %

Target-based encoding via categorical target

Car	Target	Target_encoded
911	10	10
718	20	25
Taycan	60	60
718	30	25
Taycan	40	60
Taycan	80	60

Car	Average
911	10
718	25
Taycan	60

Target-based encoding via numerical target

4.4 Target-based Encoding

- To run the `targetBasedEncoding` from `scikit-learn` use the following code:

```
from sklearn.preprocessing import TargetEncoder

encoder = TargetEncoder()
encoder.fit_transform(data)
```

4.4 OneHotEncoding

- One-hot Encoding transforms each categorical feature with n possible categories into n binary features, with one of them 1, and all others 0.

Car	Category
911	1
718	2
Taycan	3

➡

911	718	Taycan
1	0	0
0	1	0
0	0	1

4.4 OneHotEncoding

- The `OneHotEncoder` from `scikit-learn` helps us to encode our data
- It transforms each categorical feature into binary features with the related number of categories, with one of them 1, and all others 0.

```
from sklearn.preprocessing import OneHotEncoder

encoder = OneHotEncoder()
encoder.fit_transform(data)
```

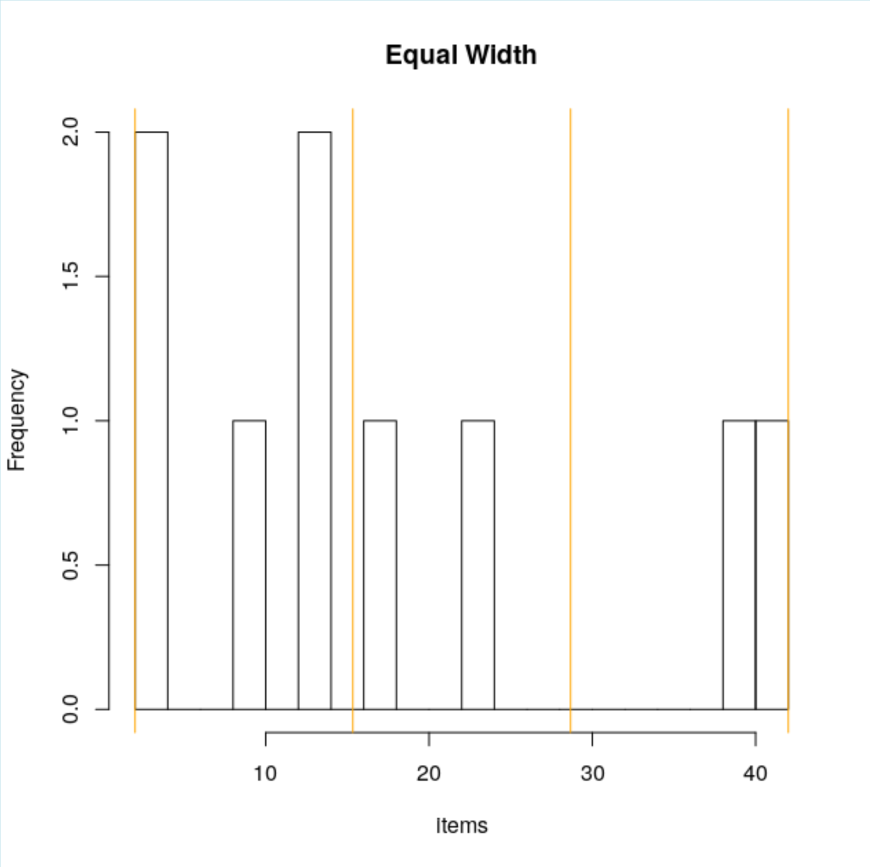
4.4 Binning or discretization

- Binning or discretization: the process of transforming numerical variables into categorical counterparts

- If the knowledge about the hierarchies of concepts in data is present, we can discretize the values at lower levels to values at highest levels: e.g. street > city > state > country

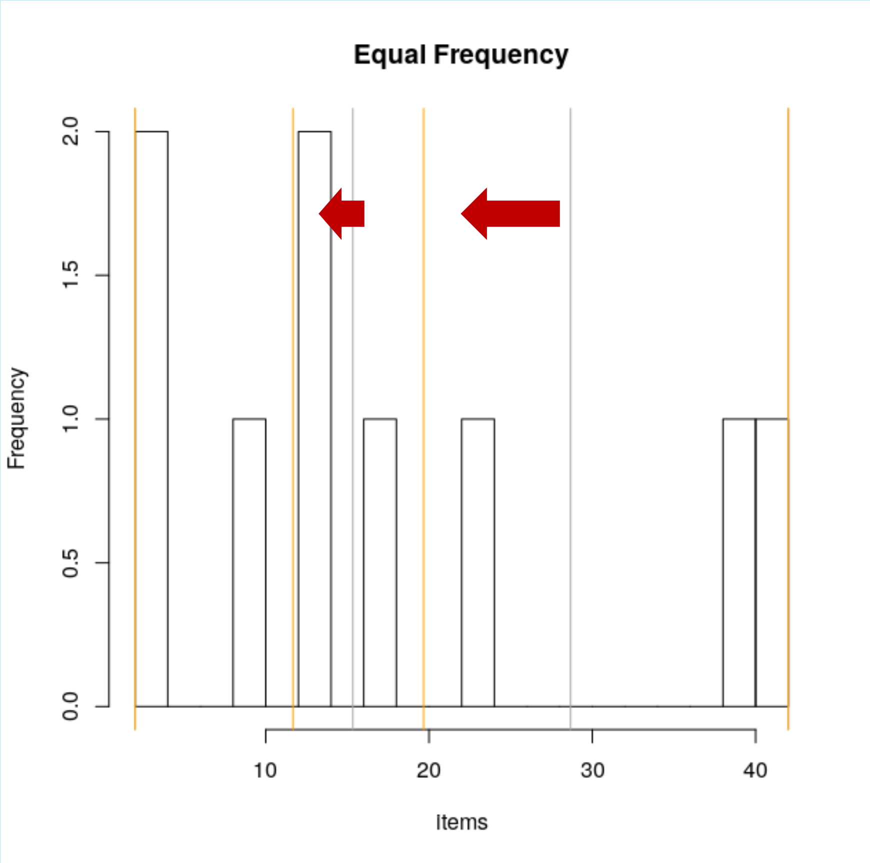
- Otherwise, we have to use specific binning algorithms:
  - Equal Width Binning
  - Equal Frequency Binning

4.4 Equal Width Binning



- The algorithm divides the data into k intervals of equal size.
- The width of intervals is $w = \frac{(\max - \min)}{k}$
- The interval boundaries are: $\min + w, \min + 2w, \ldots, \min + (k-1)\,w$

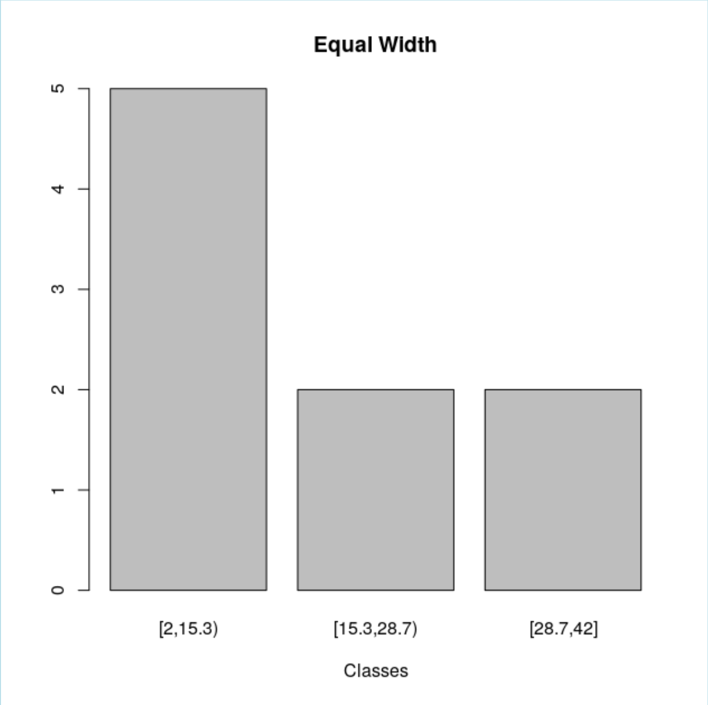
4.4 Equal Frequency Binning

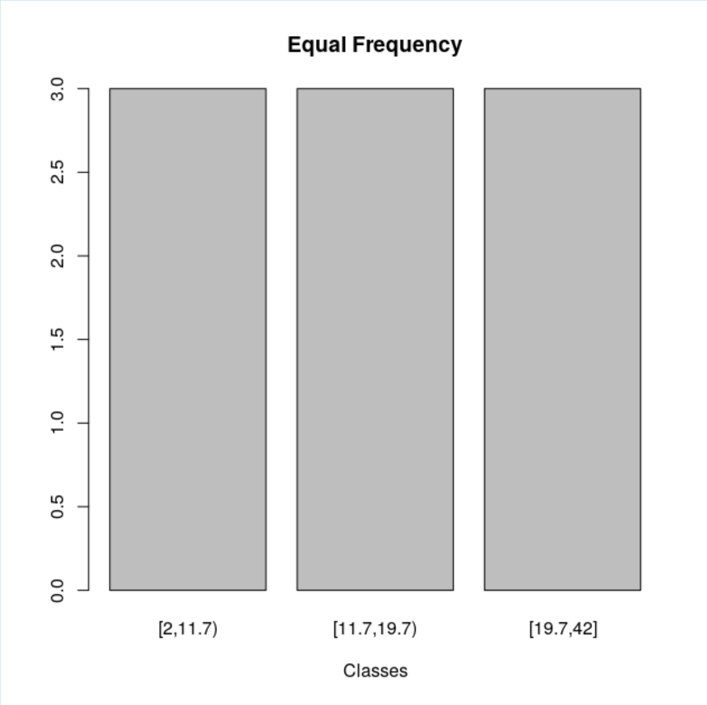


- The algorithm divides the data into *k* groups which each group contains approximately same number of values
- For the both methods, the best way of determining *k* is by looking at the histogram and try different intervals or groups.

4.4 Equal Frequency vs. Equal Width

```
x = (2, 4, 9, 13, 13, 18, 23, 40, 42)
```





4.4 Value Transformation

- If you have different „types“ of features or when your input data set has large differences between their ranges value transformation comes into play
- The most popular transformation is the Z-score (zero-mean) normalization or so called “Standardization”

$$x' = \frac{x - \bar{x}}{\sigma_x}$$

- **Effect**: Standardization rescales data to have a mean of 0 and standard deviation of 1

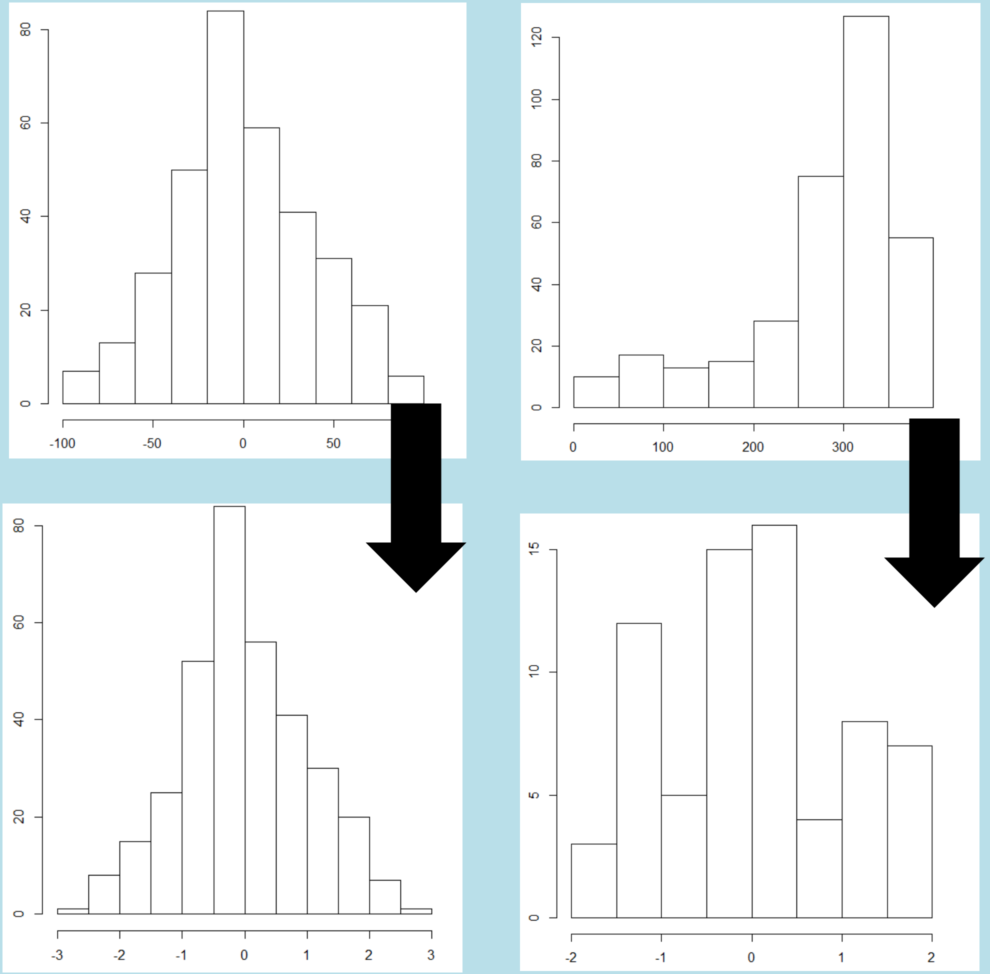
4.4 Standardization

```
from sklearn import preprocessing

preprocessing.scale(data)
```

```
inverse_transform(x_sc)
```

- `Scikit learn` provides standard value transformation in his module preprocessing

! Some people without statistical education say that the z-scores are normally distributed. This is wrong, standardization does **NOT** change the form of the distribution, it only shifts/expands/compresses it to zero mean and standard deviation of 1



4.4 Normalization

- Normalization is a value transformation method. It scales your data to have a unit norm. The most popular is the min-max normalization

- Min-max normalization

$$x' = \frac{x - min_x}{max_x - min_x}$$

- You can run it with Python:

```
from sklearn.preprocessing import Normalizer

normalizer = Normalizer()
normalizer.fit_transform(data)
```

Your turn!

Task

Managing your data source is a key-success factor in Data Engineering and AI Development. Please discuss with your neighbor what could be **potential steps and challenges** in the data processing **building automated trading agents for crypto markets**?

Image source: ↗ Pixabay (2019) / ↗ CC0

Outline

4	Data and Feature Engineering with Python
4.1	Data Imports with Python
4.2	Exploratory Data Analysis with Matplotlib
4.3	Data Handling with Pandas
4.4	Data Preprocessing with Scikit-learn
4.5	Feature Engineering and the Curse of Dimensionality

Lectorial 2: Staff Planning with Genetic Algorithms

▶ **What we will learn:**

- How data and knowledge is handled in AI-based information systems
- Most common methods for exploring the data to get better domain understanding
- Learn to prepare and transform data to make it ready for your AI project
- Deepen your Python programming skills using popular Python modules like Pandas and Scikit-learn for data engineering

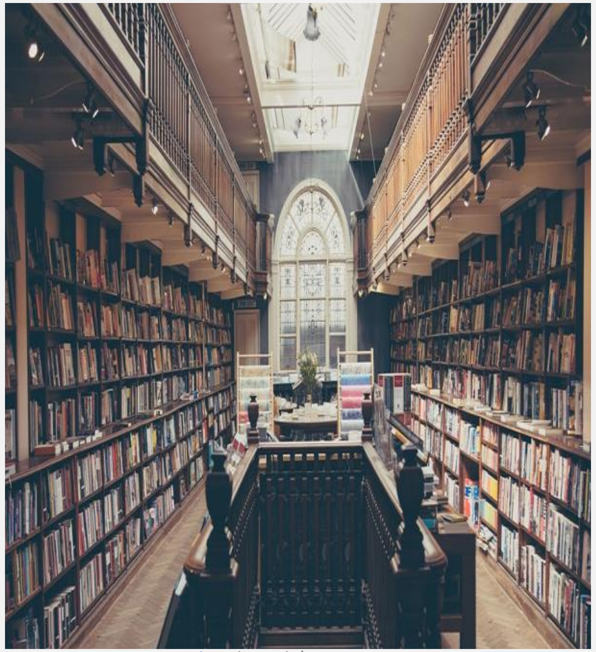


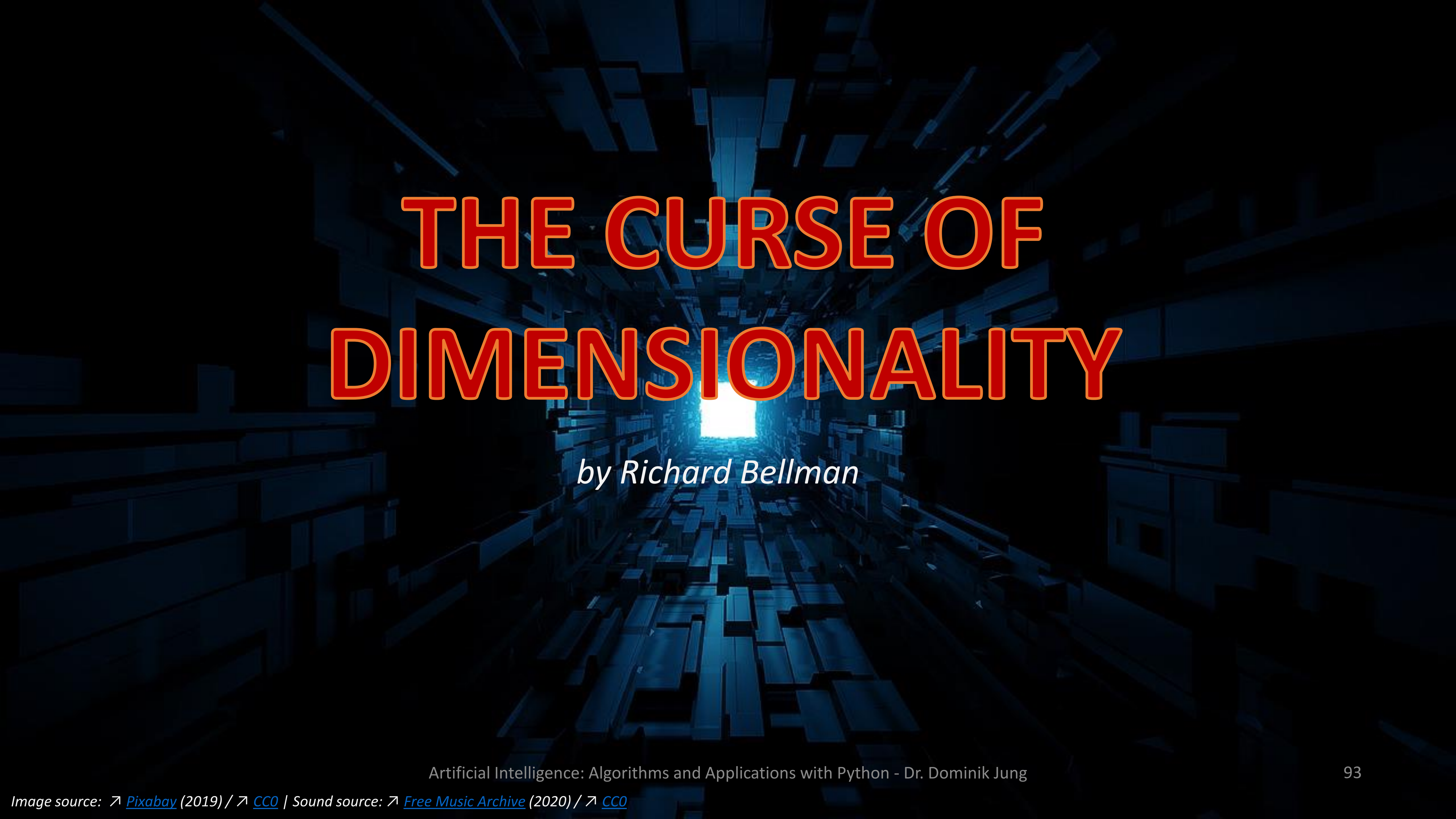
Image source: ↗ Pixabay (2019) / ↗ CC0

▶ **Duration:**

- 135 min

▶ **Relevant for Exam:**

- 4.1-4.5



THE CURSE OF DIMENSIONALITY

by Richard Bellman

Image source: ↗ Pixabay (2019) / ↗ CC0 | Sound source: ↗ Free Music Archive (2020) / ↗ CC0

4.5 The Curse of Dimensionality – Example

- Let us assume a default predictive maintance scenario

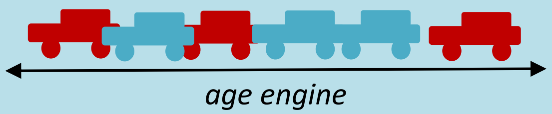
- We want to build an agent that predicts if a car will have a technical failure before a customer returns to the car station for the next checkup

Technical problem

Car is fine

Task: Avoid that the customer has a technical failure before the next inspection

4.5 The Curse of Dimensionality – Example

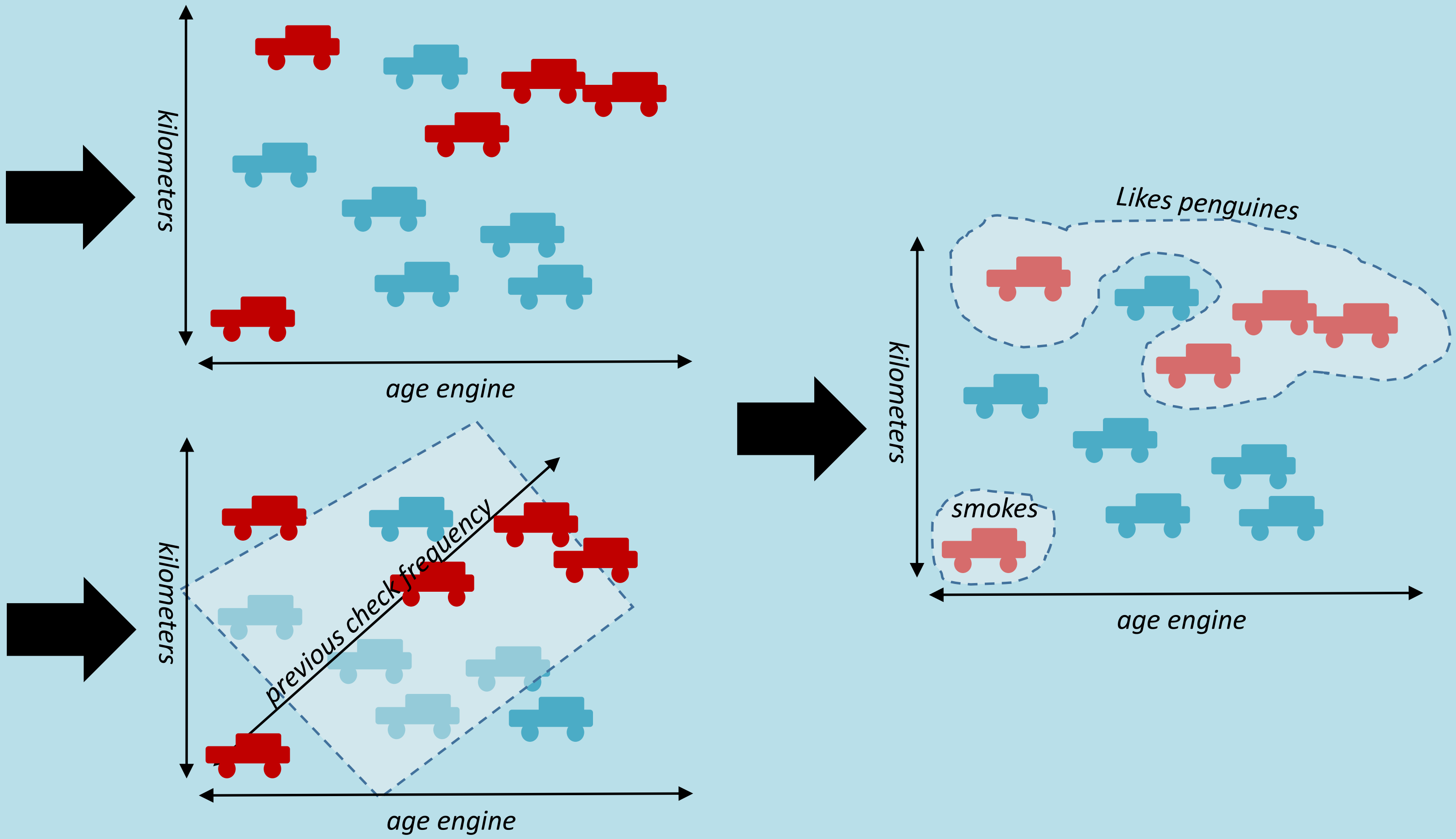


Task: Avoid that the customer has a technical failure before the next inspection

- Add new features for class separation

Technical problem

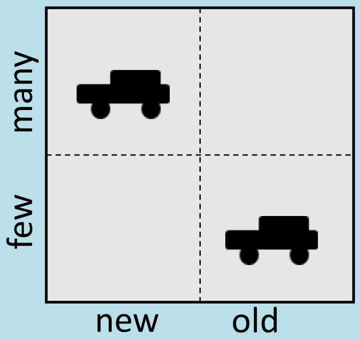
Car is fine



D **Curse of Dimensionality**

As the number of dimensions (features) grows, the amount of data we need to generalize accordingly grows exponentially.

Visualization

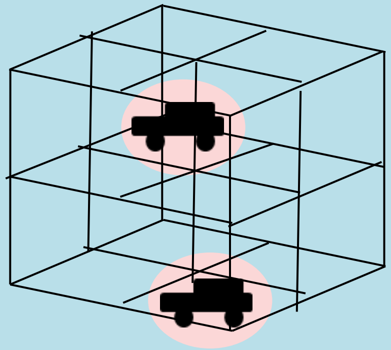


Variables:
- **age**: new, old
- **Kilometers**: ≤ Ø, > Ø

Total: 4 categories

2^{N-1}

The existing data fills 50 % of the feature spaces. The required number of sample data to fill 50 % grows expontentially with the number of dimensions.



Variables:
- **age**: new, old
- **kilometers**: ≤ Ø, > Ø
- **check frequency**: ≤ Ø, > Ø

Total: 8 categories

- The required data points for machine learning often grows expontentially with the number of features (dimenisons)

Bellman, R. E. (1957)

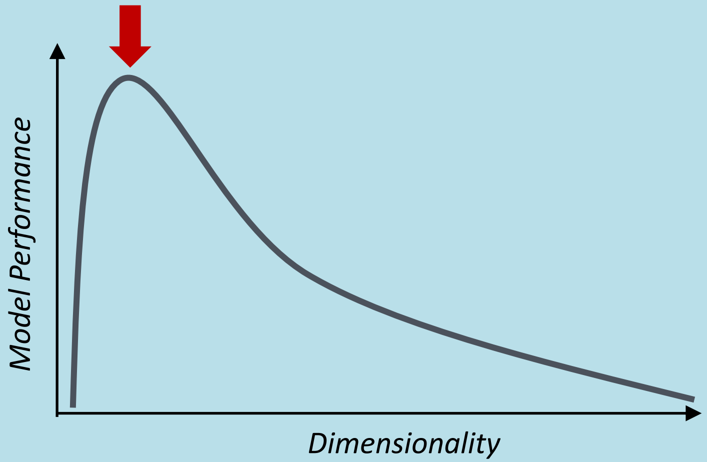
4.5 Motivation Feature Selection

- As we have seen a high number of features is associated with many computional and theoretical problems like that we need exponentially more data (and more performance, have harder interpretability etc.)

- However, the complexity of learning models is implicitly defined by the features, because they are used as attributes in the learning algorithms

- We have to find a way to select the right set of features

> **!** Feature selection is a key success criterion for many AI and machine learning problems, optimal results are very difficult to achiev!



4.5 Feature Engineering Methods for Dimensionality Reduction

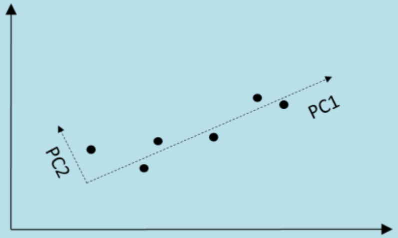
Feature Extraction

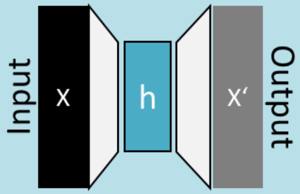
An algorithm uses human-prepared training data to learn the relationship between given inputs and a given outcome.

Feature Extraction

Linear Methods

Non-Linear Methods





- Extract a **set** of features or **linear combination** of them
- Extract a **set** of features or **non-linear combination** of them

Feature Selection

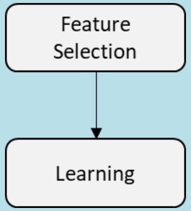
An algorithm examines input data without knowing about attributes and possible results.

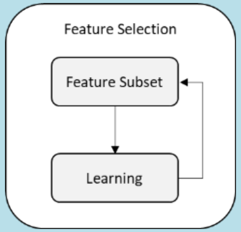
Feature Selection

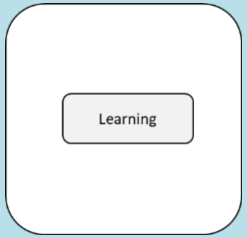
Filter Methods

Wrapper Methods

Embedded Methods







- Methods which **do not incorporate a specific** machine learning **algorithm**
- Evaluate a **specific machine learning** algorithm to find best features
- Observe each training phases to **embed features during model building** process

4.5 Linear Feature Reduction

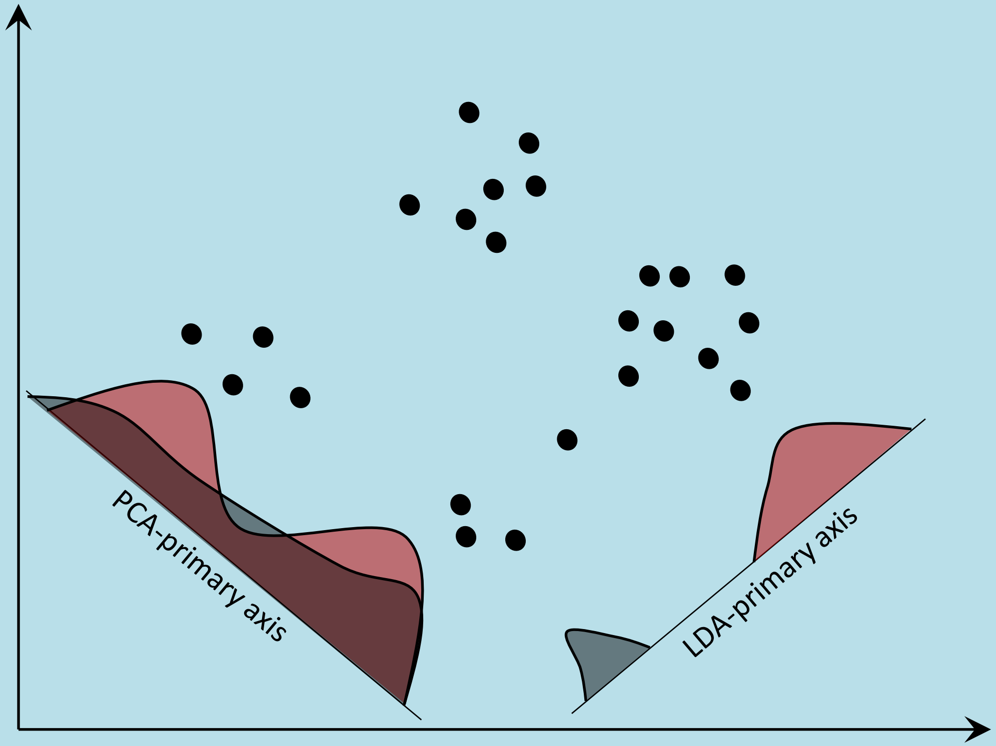
PCA: Principle Component Analysis

- Linear protection that maximizes the variance among the data points

LDA: Linear Discriminant Analysis

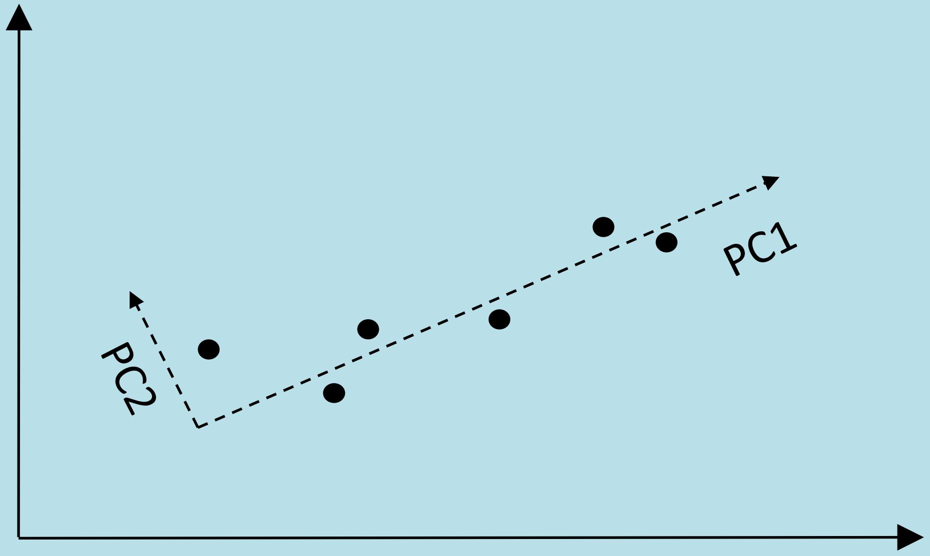
- Linear protection that separates the classes with the maximum distance between the class means

Graphical Interpretation



4.5 Feature Extraction: Prinicipal Components Analysis (PCA)

▶ Search for a lower-dimensional space that best represents the data



- Transform the coordinate system in a way that the first axis is placed in the direction of the greatest variance. Similarly for the other axes
- Removes correlated features; all the principal components are independent of one another.

Example
- Media editing, quality control, portfolio analysis

\+ Reduces overfitting and hence algorithm performance

\- Variables become less interpretable, number of prinicipal components directly related to information loss

Adapted from Smith, L. I. (2002)

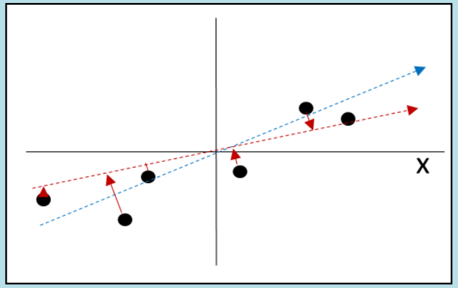
4.5 PCA Geometric Illustration

Initial dataset (scaled!)

Object	O1	O2	On
O1	0	1	4
O2	2	0	3
...	...	...	...

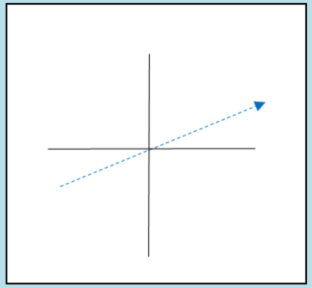
The first axis is placed in the direction of the greatest variance. Repeat for the other axes

Compute the variance and the related sum of squared distances for each line (between the projected points and the origin)



$SS(distances) = d_1^2 + d_2^2 + \dots + d_n^2$

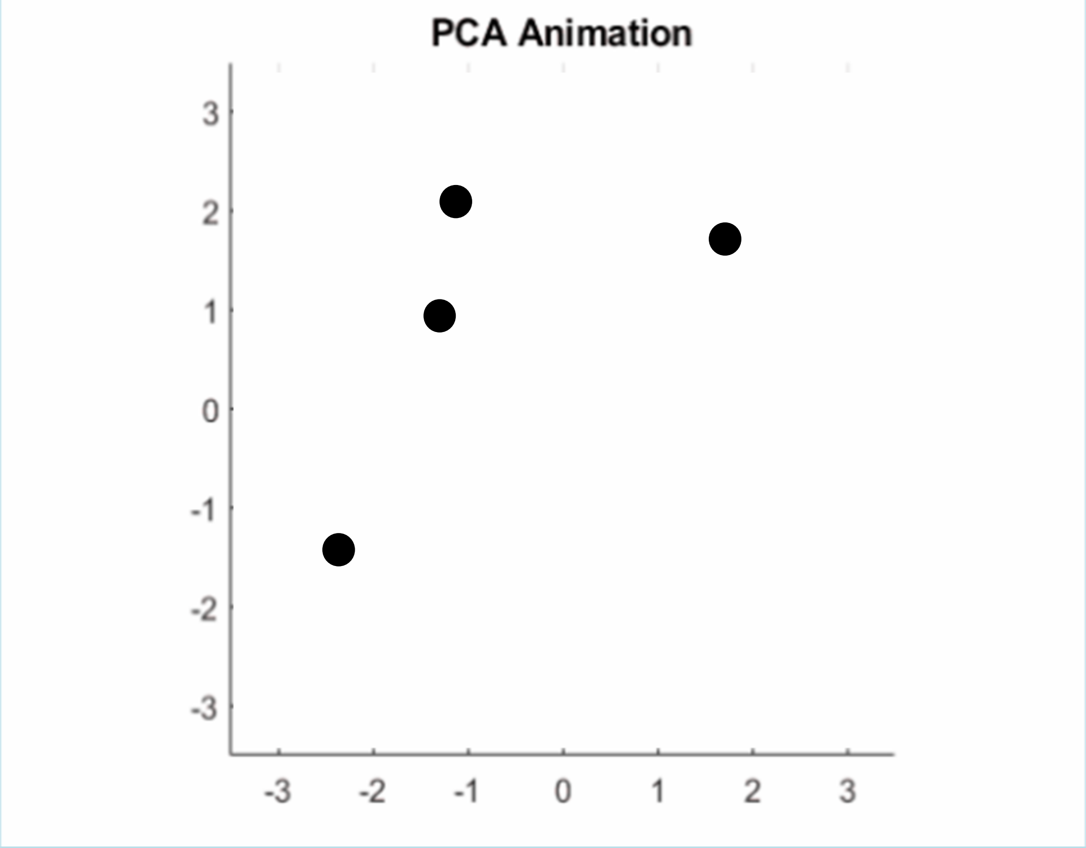
→ maximizing the sum of the squared distances from the projected points to the origin



Repeat until you find the line with the largest SS

Repeat until each axis is found

Visualization



Adapted from Smith, L. I. (2002) | Image sources: Adapted from Amoeba from stackexchange (2020)

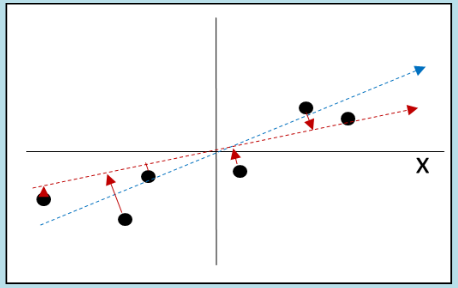
4.5 PCA Procedure

Initial dataset (scaled!)

Object	O1	O2	On
O1	0	1	4
O2	2	0	3
...	...	...	...

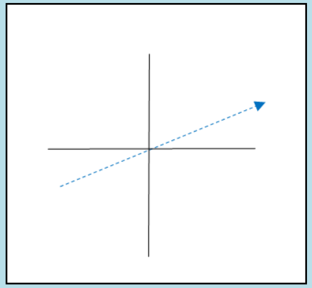
The first axis is placed in the direction of the greatest variance. Repeat for the other axes

Compute the variance and the related sum of squared distances for each line (between the projected points and the origin)



$$SS(distances) = d_1^2 + d_2^2 + \dots + d_n^2$$

→ maximizing the sum of the squared distances from the projected points to the origin



Repeat until you find the line with the largest SS

Repeat until each axis is found

Adapted from Smith, L. I. (2002)

PCA Computation

1. Substract the mean or standardize your data
2. Calculate the covariance matrix
3. Calculate the eigenvectors and eigenvalues of the corvariance matrix
4. Compute the feature vector
5. Recast the data along the principal component

There is a very illustrative tutorial on pca computation from Otago University. Take a look at it ↗ here.

4.5 PCA in Python

- You can run a PCA easily with `scikit-learn`

```
from sklearn import decomposition

pca = decomposition.PCA(n_components=3)
principalComponents = pca.fit_transform(your_data)
```

- Note that PCA is effected by scale so you need to scale the features in your data before applying PCA (e.g. with the `StandardScaler()` from `scikit-learn`)

Adapted from Smith, L. I. (2002)

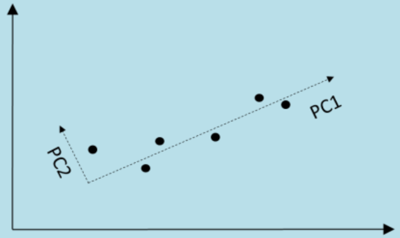
4.5 Feature Engineering Methods for Dimensionality Reduction

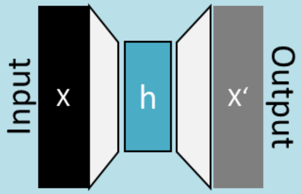
Feature Extraction

An algorithm uses human-prepared training data to learn the relationship between given inputs and a given outcome.

Feature Extraction

Linear Methods | Non-Linear Methods





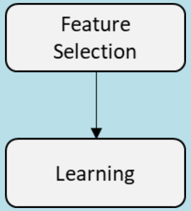
- Extract a **set** of features or **linear combination** of them
- Extract a **set** of features or **non-linear combination** of them

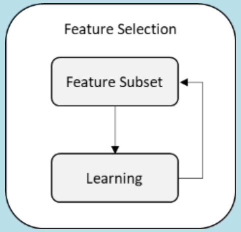
Feature Selection

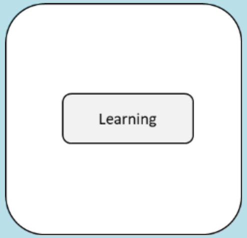
An algorithm examines input data without knowing about attributes and possible results.

Feature Selection

Filter Methods | Wrapper Methods | Embedded Methods





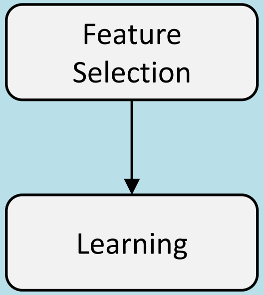


- Methods which **do not incorporate a specific** machine learning **algorithm**
- Evaluate a **specific machine learning** algorithm to find best features
- Observe each training phases to **embed features during model building** process

4.5 Filter Methods

- Select features based on quality criterion

- Possible critera are:
  - Information gain
  - Chi-squard test
  - Fisher score
  - Correlation coefficent
  - Variance threshold

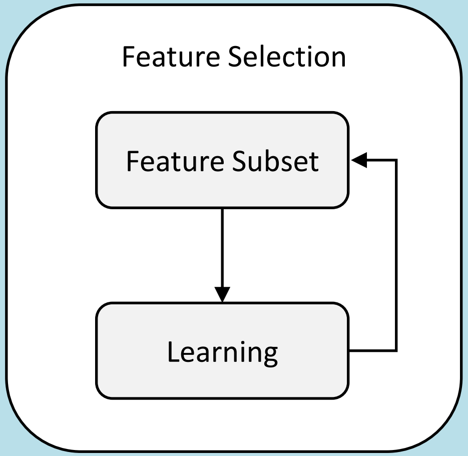


4.5 Wrapper Approach

- Select features based on evaluation criterion following a greedy search approach by evaluating all the possible combinations of features against the evaluation criterion

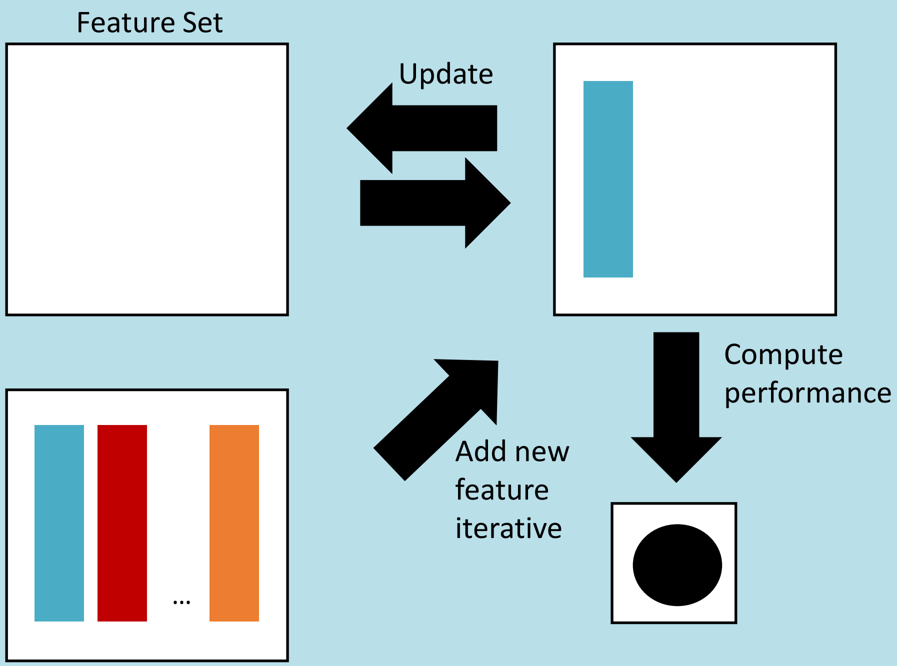
- As evaluation criterion you can take the performance measure which depends on the type of learning problem

- In general, there are three popular approaches to select the features:
  - Forward selection
  - Backward elimination
  - Genetic algorithms



4.5 Forward and Backward Selection

- Compute different models with your learning algorithm and different features, use the best one



Algorithm: Sequential Forward Selection

S, empty feature set

for each *Feature, S*
  *Select the next best feature f**
  $S \leftarrow ADD(f^*)$
  $P_i \leftarrow COMPUTE\text{-}PERFORMANCE(S)$
  if $(P_i < P_{i-1})$ *break*

- Backward selection: same as forward selection, but starts with full feature set and single features are step-wise removed

4.5 Forward and Backward Selection with Python

- This Sequential Feature Selector adds (forward selection) or removes (backward selection) features to form a feature subset in a greedy fashion.
- At each stage, this estimator chooses the best feature to add or remove based on the cross-validation score of the Akaide Information Criterion (AIC) as performance measure. If the score is reduced, the procedure stops.

SequentialFeatureSelector()

```
from sklearn.feature_selection import SequentialFeatureSelector
SequentialFeatureSelector(estimator, direction, n_features_to_select)
```

estimator	An unfitted estimator (your learning algorithm)
direction	Whether to perform forward selection or backward selection.
n_features	The number of features to select. If None, half of the features are selected

4.5 Recursive Feature Elimination (RFE) with Python

- Variant of backward feature selection, that computes all subsets and eliminations and then chooses the best subset
- Select features by recursively considering smaller and smaller sets of features: First, the estimator is trained on the initial set of features and the importance of each feature is used to compute the feature's importance

`RecursiveFeatureSelection()`

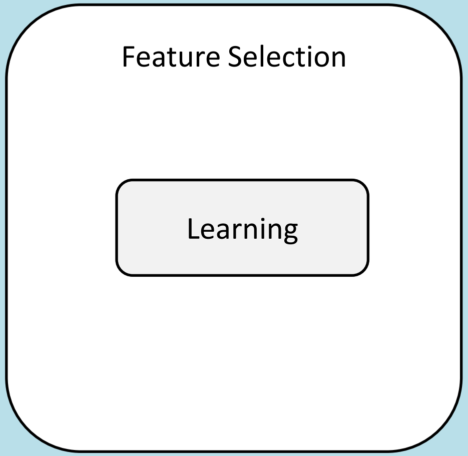
```
from sklearn.feature_selection import RFE
RFE(estimator, n_features_to_select)
```

`estimator`	An unfitted estimator (your learning algorithm)
`n_features`	The number of features to select. If None, half of the features are selected

Adapted from Guyon, I. et al. (2002) | Image source: ↗ Pixabay (2019) / ↗ CC0

4.5 Embedded Feature Selection

- Some algorithms embed (fix) features during model building process.
- Feature selection is done by observing each iteration of model training phase
- Examples: LASSO, elastic net, ridge regression etc.



4.5 Classroom Task

Previous
Exam Task!

Your turn!

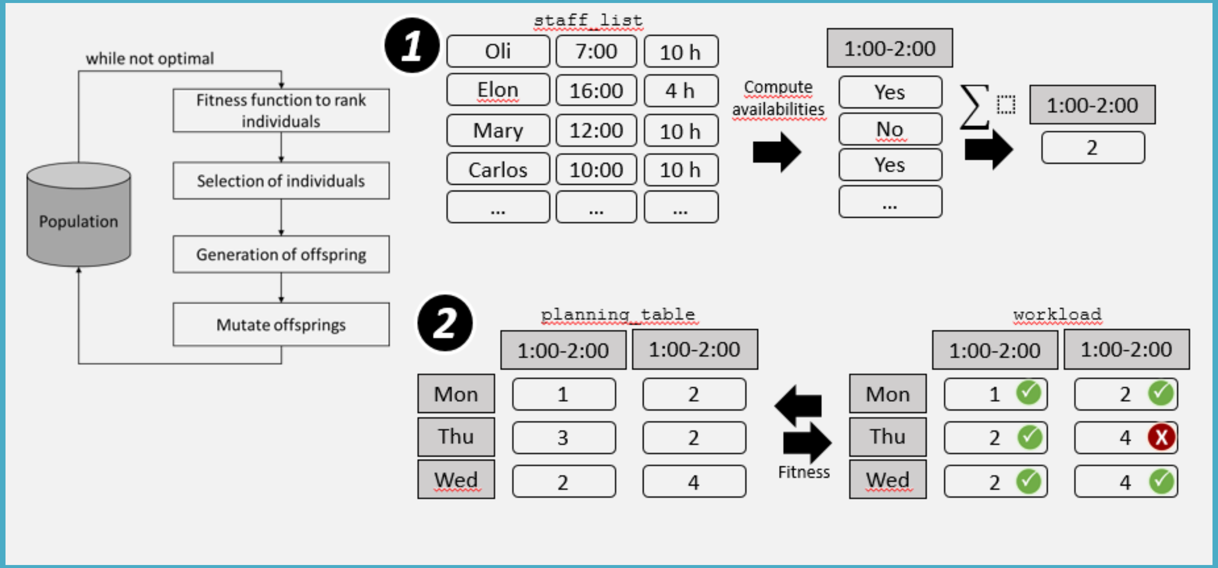
Task

Please discuss with your neighbor: What are criteria to classify data as “high dimensional” and why might this kind of data be a problem for AI in specific subdomains like machine learning, knowledge reasoning or natural language processing?

Consider the following aspects

- Model training
- Modelling techniques (e.g. clustering)

Your next project:
Staff Planning with Genetic Algorithms



Exercises

Workbook Exercises

- Read the chapter 8 of Géron, A. (2017). Find a learning-group and discuss the related exercises of each chapter. We will come back at the other chapters later in this lecture.

Coding Exercises

- Take a look at the code of the „Lectorial – Stuff Planning with Python". Try to understand the code and run it on your local computer.
- Increase the number of parents in one generation
- Adjust the script that the genetic algorithm uses the original `staff_list` as a starting point instead the random generated `staff_lists`

References

Literature

1. Géron, A. (2017). *Hands-on machine learning with Scikit-Learn and TensorFlow: Concepts, tools, and techniques to build intelligent systems.*
2. Guyon, I., Weston, J., Barnhill, S., & Vapnik, V. (2002). Gene selection for cancer classification using support vector machines. Machine learning, 46(1-3), 389-422.
3. Rusell, S., & Norvig, P. (2016). *Artificial Intelligence: A Modern Approach.* Global Edition.
4. Smith, L. I. (2002). *A tutorial on principal components analysis*. Online available at: http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf
5. Wickham, Hadley. "Tidy data." *Journal of Statistical Software* 59.10 (2014): 1-23.

References

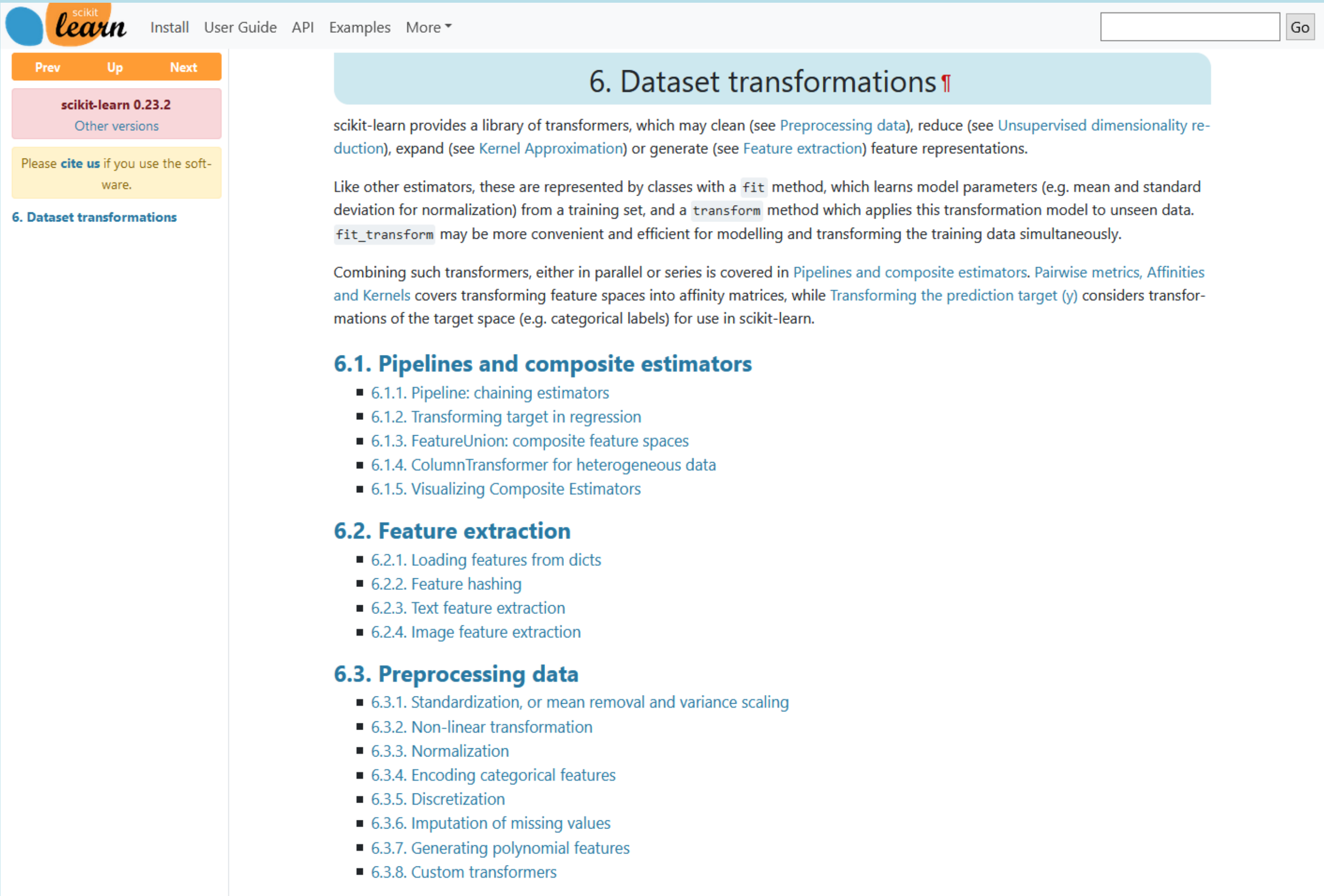
Images

All images that were not marked other ways are made by myself, or licensed ↗CC0 from ↗Pixabay.

Further reading

- Statistics is important if you want to become an AI specialist. Hence, it would be good to recapitulate your old lectures about statistics. Take them out, you will need profund knowledge in descriptive statistics, if you want to conduct sucessful data understanding and preprocessing. If your statistics lectures have not been that good, I can recommend to take a look at ↗Coursera.
- Besides SQLlite I can recommend to take a look at MySQL and MariaDB. There are pretty beginner-friendly database systems you can use for your further projects.
- There is a very illustrative example of PCA on scikit-learn, take a look at it (↗ here)
- I strongly want to encourage you to take a look at the scikit-learn userguide (↗ here). It illustrates the most popular methods for data and feature engineering.

Scikit-Learn Userguide



scikit learn Install User Guide API Examples More Go

Prev Up Next

scikit-learn 0.23.2
Other versions

Please **cite us** if you use the software.

6. Dataset transformations

6. Dataset transformations¶

scikit-learn provides a library of transformers, which may clean (see Preprocessing data), reduce (see Unsupervised dimensionality reduction), expand (see Kernel Approximation) or generate (see Feature extraction) feature representations.

Like other estimators, these are represented by classes with a `fit` method, which learns model parameters (e.g. mean and standard deviation for normalization) from a training set, and a `transform` method which applies this transformation model to unseen data. `fit_transform` may be more convenient and efficient for modelling and transforming the training data simultaneously.

Combining such transformers, either in parallel or series is covered in Pipelines and composite estimators. Pairwise metrics, Affinities and Kernels covers transforming feature spaces into affinity matrices, while Transforming the prediction target (y) considers transformations of the target space (e.g. categorical labels) for use in scikit-learn.

6.1. Pipelines and composite estimators

- 6.1.1. Pipeline: chaining estimators
- 6.1.2. Transforming target in regression
- 6.1.3. FeatureUnion: composite feature spaces
- 6.1.4. ColumnTransformer for heterogeneous data
- 6.1.5. Visualizing Composite Estimators

6.2. Feature extraction

- 6.2.1. Loading features from dicts
- 6.2.2. Feature hashing
- 6.2.3. Text feature extraction
- 6.2.4. Image feature extraction

6.3. Preprocessing data

- 6.3.1. Standardization, or mean removal and variance scaling
- 6.3.2. Non-linear transformation
- 6.3.3. Normalization
- 6.3.4. Encoding categorical features
- 6.3.5. Discretization
- 6.3.6. Imputation of missing values
- 6.3.7. Generating polynomial features
- 6.3.8. Custom transformers

Check out the scikit-learn user guide for more data and feature engineering algorithms (↗ here).

Glossary

Correlation (Pearson)	*The (Pearson) correlation coefficient r measures the strength of the linear relationship between two numerical variables*
Data Cleaning	*Relying on the GIGO-principle, data cleaning is the process of removing errors and inconsistencies in your database.*
Data Enrichment	*Sometimes you have to add further information or build features to increase the data quality. Doing so, the granularity of your data is often too high, and your data has to be on an more aggregated level*
Data Integration	*In this step all relevant data sources are connected to your AI application. For this purpose the data is integrated across data tables.*
Data Transformation	*Most AI applications (in particular the models) require the input data to be in a specific form. Furthermore, variable types have to be specified and/or changed to process data.*
EDA	*Exploratory Data Analysis, Methods of descriptive statistics and data visualization to understand your dataset*
Tidy Data	*A standard method of displaying a multivariate set of data is in the form of a data matrix in which rows correspond to sample individuals and columns to variables*