JAN BISCHOFF
DATA SCIENCE & VISUALIZATION

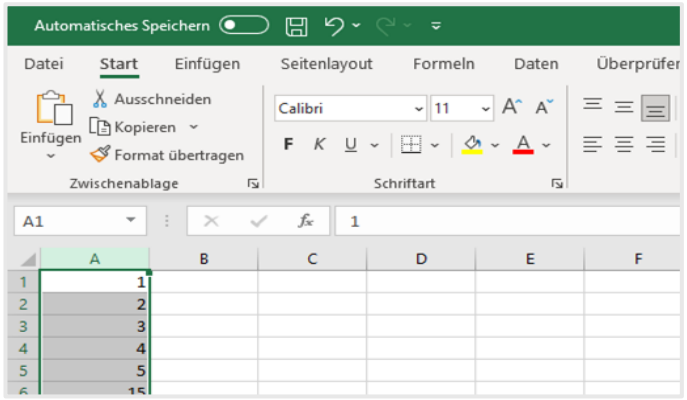
Introduction to Data Science with Python

Organisation of the course

- Alternation between theory sessions and hands-on programming sessions in Google Colab
- Focus on practical application rather than theory
- Q&A after each programming session
- Lunch break at 12:00
- Coffee breaks in the morning and afternoon

The Need to Learn Programming

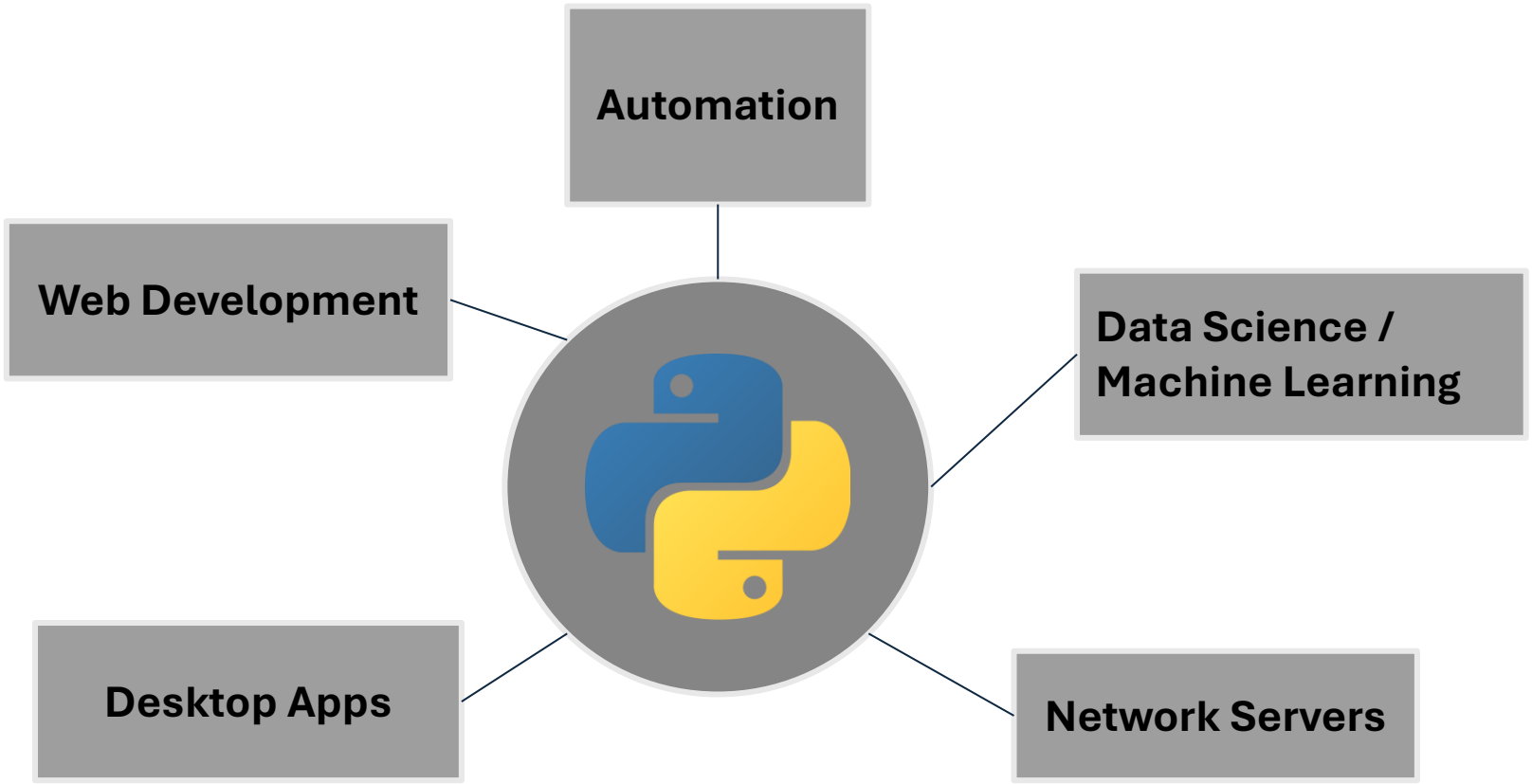
1. Select data
2. Click on buttons :)



1. Write code in editor
2. Execute code with Python
3. Result will be returned

```
data = pd.read_csv(file)
mean = data.mean()
print(mean)
```

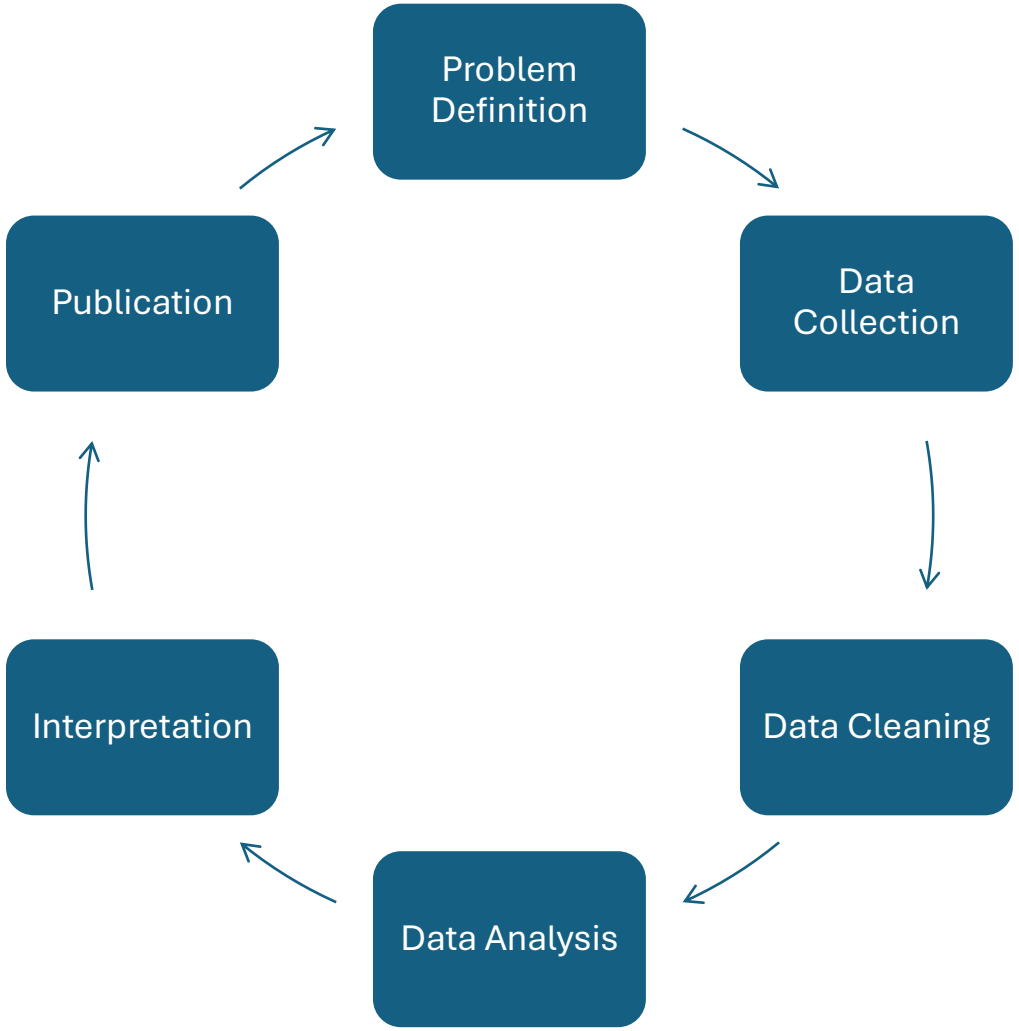
Python – A General Purpose Language



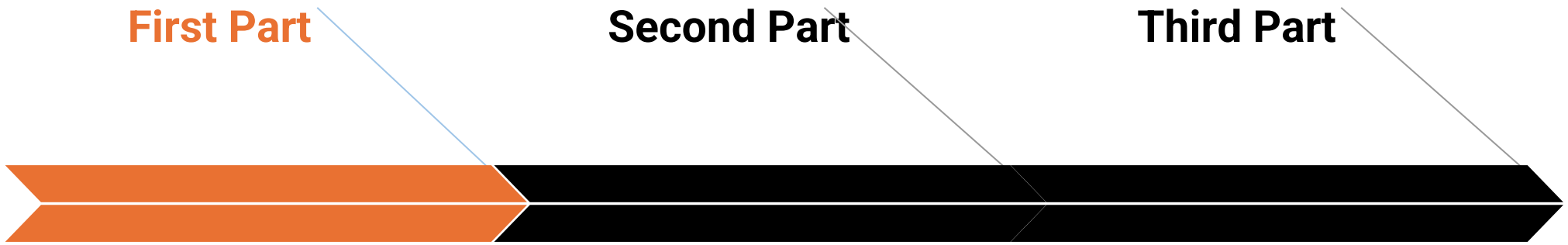
Google Colab

- Write and execute code
- Accessed via Browser (runs on Google Servers)
- No pre-configurations necessary
- Independent of your local machine
- Jupyter Notebook format heavily used in data science community

The Research / Data Science Lifecycle



Course Structure



Python Fundamentals

Basic concepts, Variables, basic data structures, functions

Data Wrangling & Simple visualizations

How to process data with pandas and visualize it with matplotlib

Visualizations & Modelling

More plots with matplotlib and seaborn and an introduction to modelling

Python Fundamentals

Data Types

How is data stored and processed ?

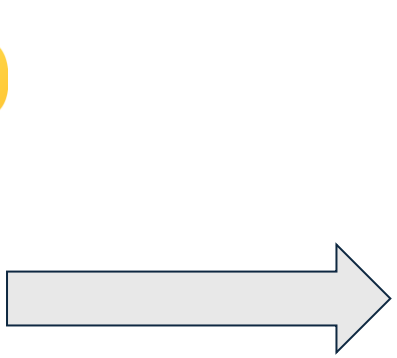
- Values are stored in **variables**
- The four most important data types in Python:

```
integer = 10
float = 2.8
string = "This is a string"
boolean = True
```

How is data stored and processed ?

- We can compute with these variables

```
a = 10
b = 5
c = a + b

print(c)
```



```
Output: 15
```

What kind of data type is this : `"27-03-2021"`

a) integer b) float c) string d) date

What kind of data type is this : `"27-03-2021"`

a) integer b) float c) string d) date

Python Fundamentals

Data Structures

Data Structures - Lists

We can combine values in lists

```
a = [5, 3, 9, 7, 4, 10, 3]
b = ["Justus", "Peter", "Bob"]
```

Data Structures - Lists

Value	5	3	9	7	4	10	3
Index	0	1	2	3	4	5	6

Data Structures - Lists

Access the data with an index

```
a = [5, 3, 9, 7, 4, 10, 3]
b = ["Justus", "Peter", "Bob"]

print(a[0])  # Output: 5
```

Data Structures - Lists

Access the data with an index

```
a = [5, 3, 9, 7, 4, 10, 3]
b = ["Justus", "Peter", "Bob"]

a[3]  # Output: 7
b[1]  # Output: Peter
```

Data Structures - Lists

```
a[start:stop:step_size]
```

Data Structures - Lists

Value	5	3	9	7	4	10	3
Index	0	1	2	3	4	5	6

```
a[1:4]
```

Data Structures - Lists

Value	5	3	9	7	4	10	3
Index	0	1	2	3	4	5	6

```
a[1:4:2]
```

Data Structures - Dictionaries

```
translate = {"Eins":"One",
             "Zwei":"Two",
               "Ja":"Yes"}
translate["Eins"]  # Output: "One"
```

Quick - Summary

Data types

```
integer    2
float      2.32
string    "Text"
boolean    True/False
```

Data structures

```
lists:  a = [1,2,3]
dictionary: b={“a”:1}
```

Exercise Time

```
Data structures - Hints
lists:
    create:  a = [1,2,3]
    access: a[0]
dictionary:
    create: b={“a”:1}
    access: b[“a”]
```

Relational operators

- Compare variables

  a == b → is a equal to b?

  returns **True / False**

Relational operators

- Compare variables

  a == b → is a equal to b?

  returns **True / False**

==	is equal
<	smaller than
>	greater than
<=	smaller or equal than
>=	greater or equal than
!=	not equal to

Relational operators

- Compare variables

  a == b → is a equal to b?

  returns **True / False**

- Combine operators with **“and” / “or”**

  “and”: (a >= b) and (a<=c)

  “or”: (a >= b) or (a<=c)

==	is equal
<	smaller than
>	greater than
<=	smaller or equal than
>=	greater or equal than
!=	not equal to

```
a = 1
b = 2
c = 2

    (a>b) or (a<=c)
```

```
a = 1
b = 2
c = 2

    (a>b) or (a<=c)
    False or True
```

```
a = 1

b = 2

c = 2

        (a>b) or (a<=c)

        False or True →  True
```

Very important for filtering

Name	Gender	Age
“Tim”	“M”	20
“Nina”	“F”	24
“John”	“M”	26

Select all Names with following condition:

(Gender == “F”) **&** (Age > 20)

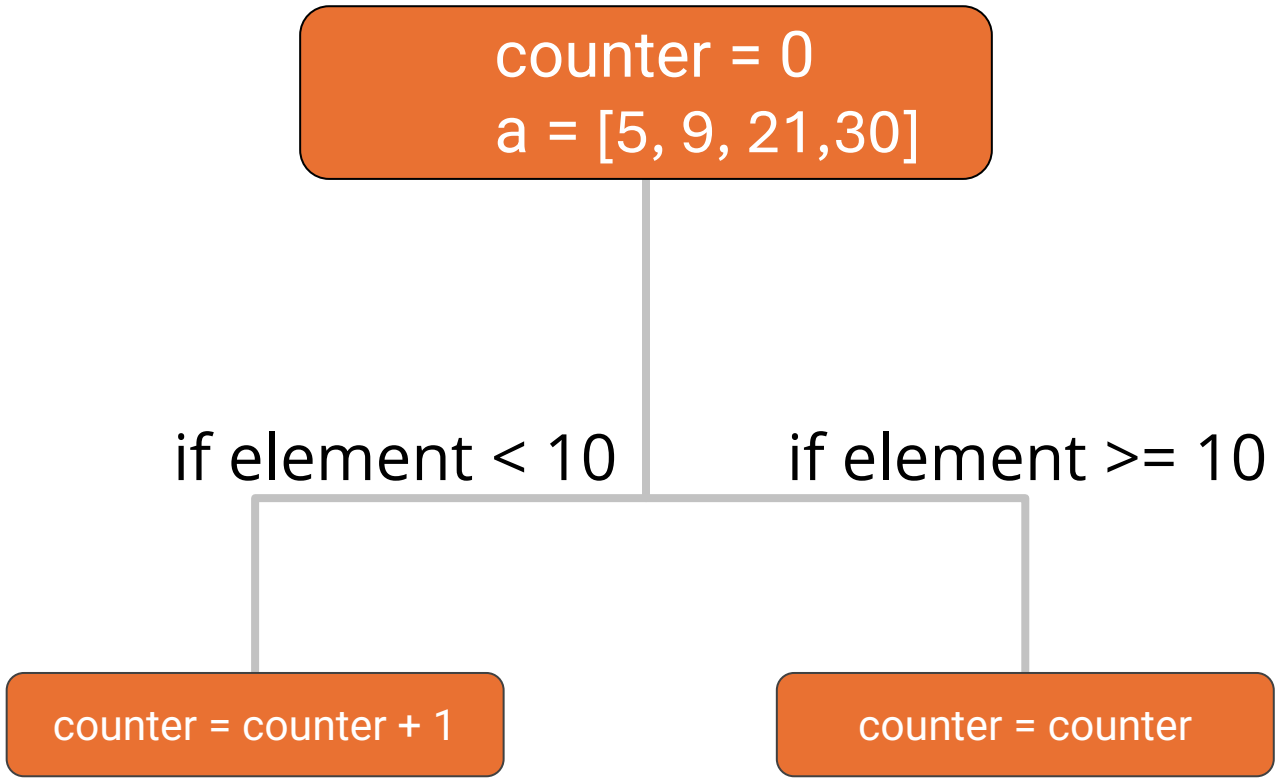
Fundamentals

Control Flow

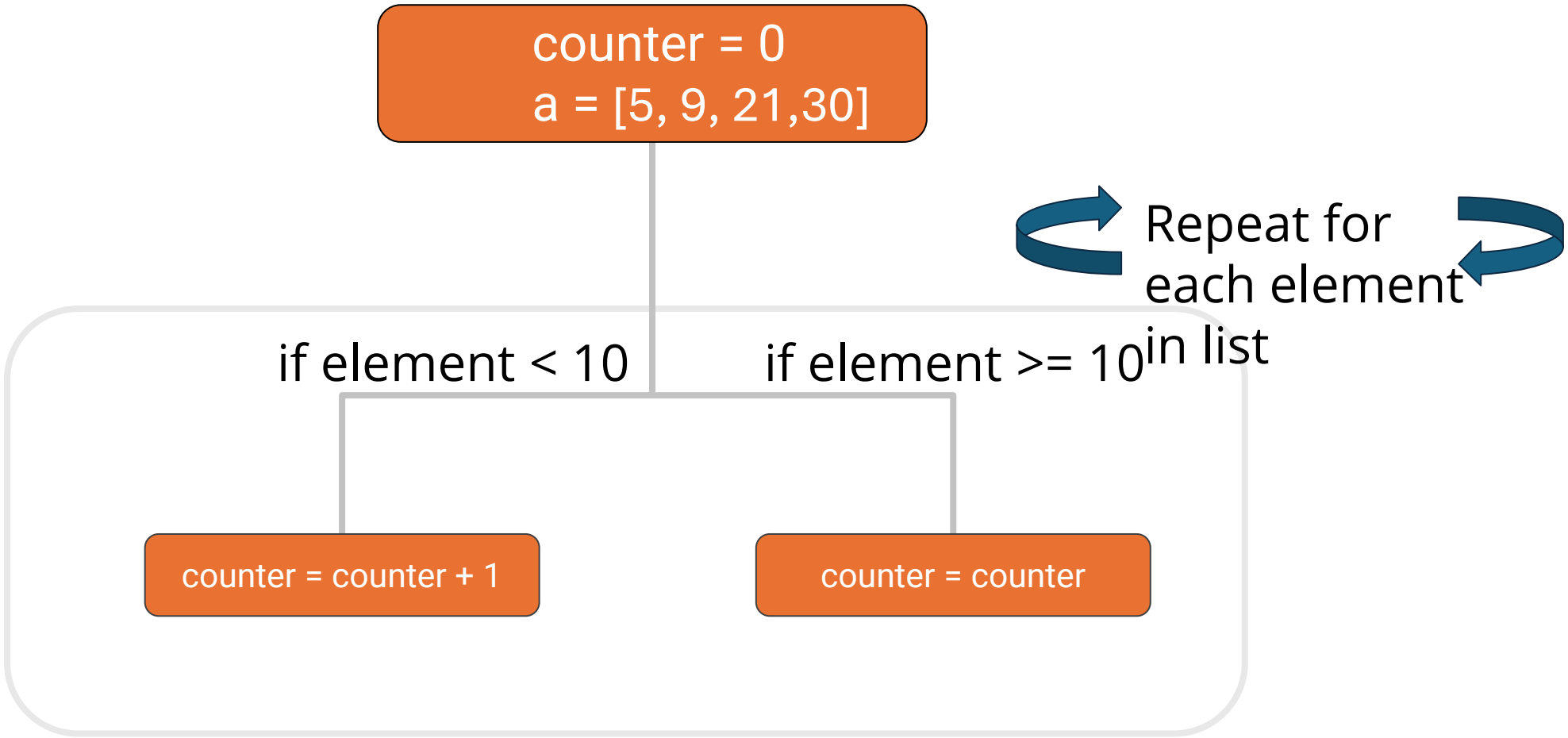
Count numbers smaller than 10 in a list

counter = 0
a = [5, 9, 21,30]

Count numbers smaller than 10 in a list



Count numbers smaller than 10 in a list



Control Flow - if / else

- Control which block of code will be executed
- Blocks defined by indentation

```
if BOOLEAN-CONDITION:
    print("A")
else:
    print("B")
```

Control Flow - if / else

- Control which block of code will be executed
- Blocks defined by indentation

```
if a>2:
    print("A")
else:
    print("B")
```

Control Flow - for-loop

- Repeat blocks of your code
- Use different values in each loop

```
for element in [1,2,3,4]:
    print(element)
```

Exercise Time

Count amount of numbers in a list which are smaller than 5

```
for element in [1,2,3,4]:
    print(element)
```

```
if a>2:
    print("A")
else:
    print("B")
```

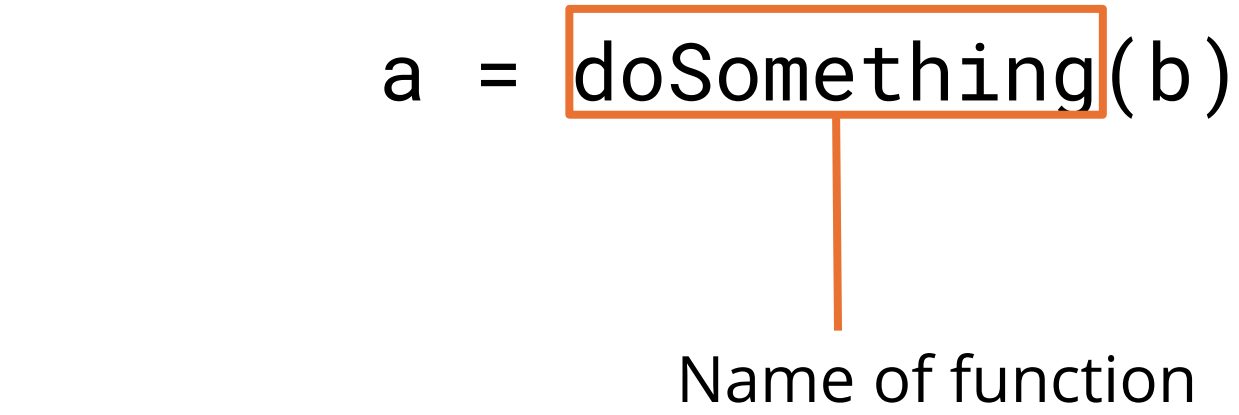
Fundamentals

Functions & Libraries

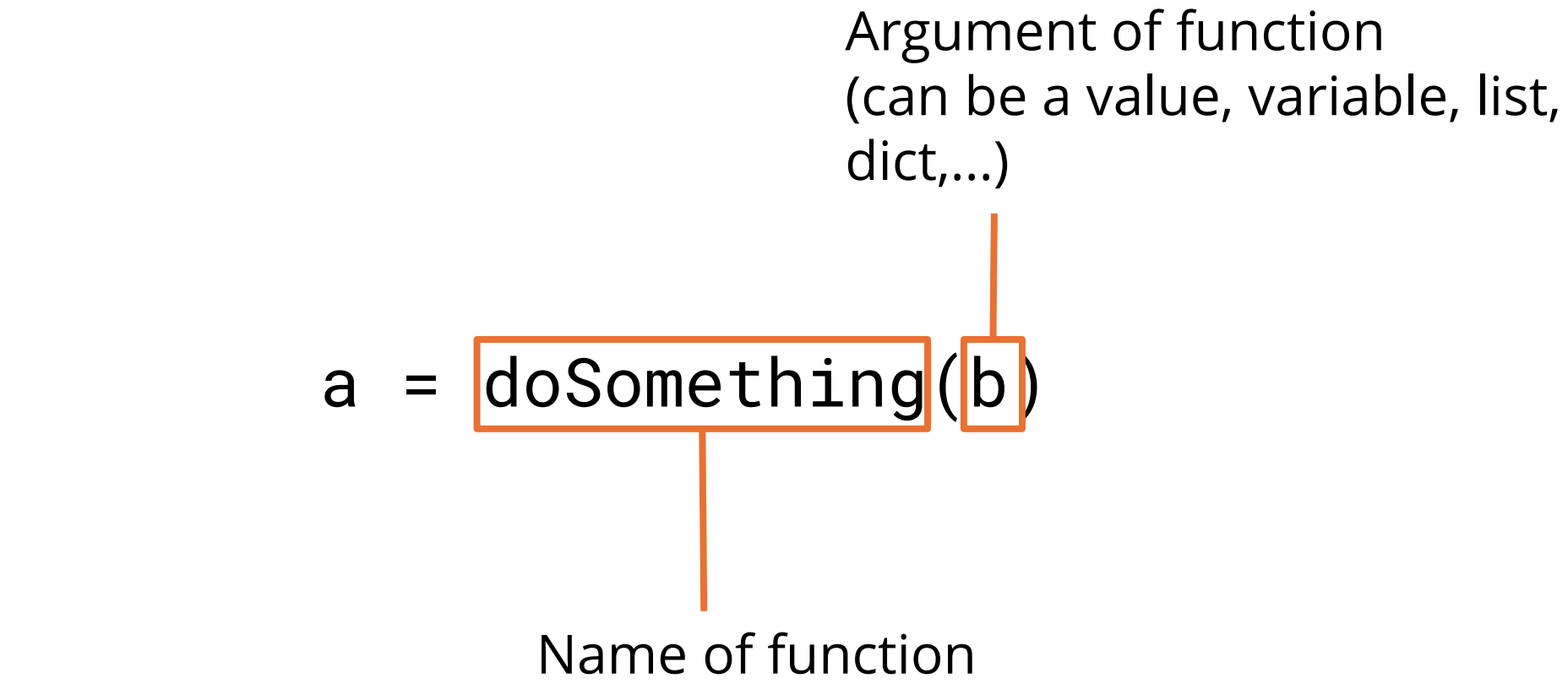
Functions

```
a = doSomething(b)
```

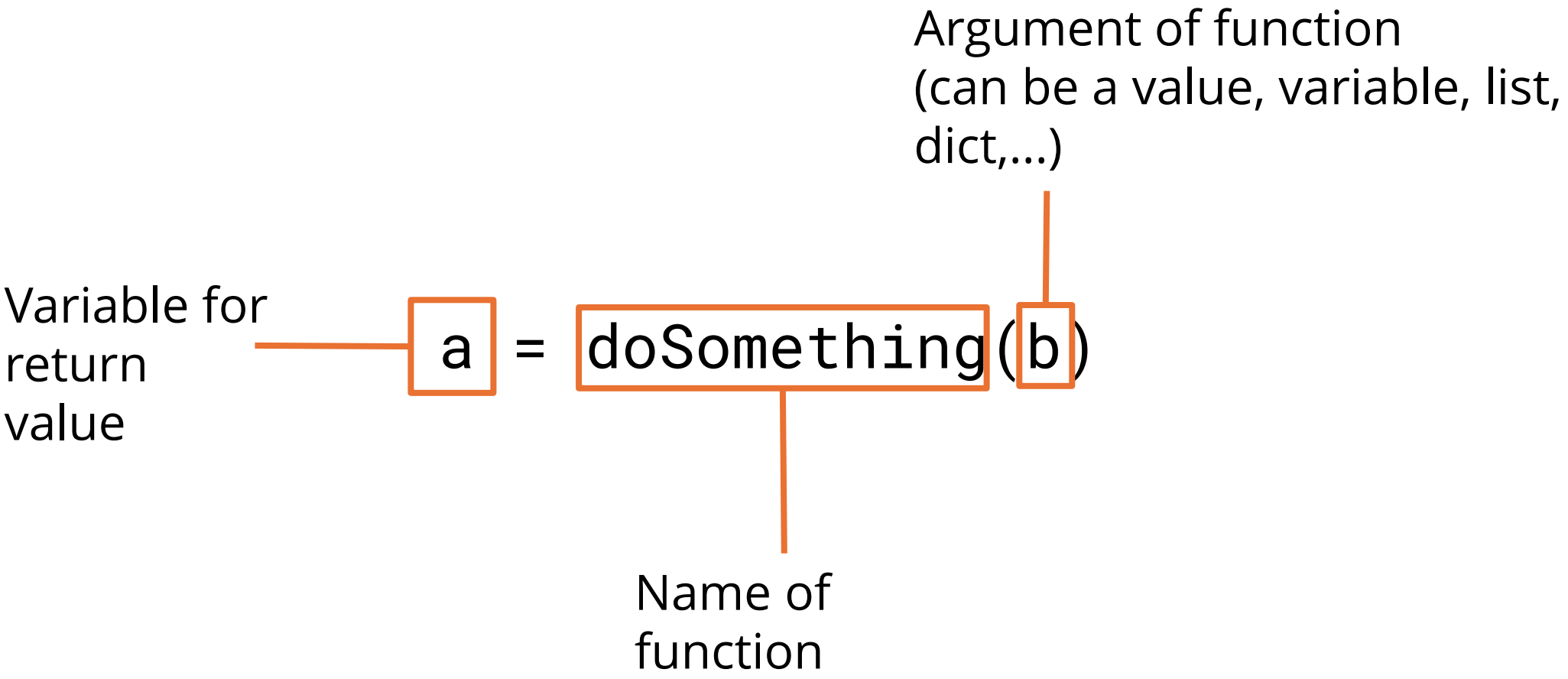
Functions



Functions


Argument of function
(can be a value, variable, list, dict,...)
a = doSomething(b)
Name of function

Functions


Argument of function
(can be a value, variable, list, dict,...)
Variable for return value
a = doSomething(b)
Name of function

Functions - round function

```
b = 5.2
a = round(b)
→ a = 5.0
```

Built-in Functions

```
print()  sum()
round()  abs()
min()    range()
max()    sorted()
```

```
round(2.34)   == ?
abs(-2)     == ?
a = [0,4,1,3,2]
max(a)          == ?
sum(a)          == ?
len(a)          == ?
sorted(a)     == ?
```

```
round(2.34)   == 2.0
abs(-2)     == 2
a = [0,4,1,3,2]
max(a)          == 4
sum(a)          == 10
len(a)          == 5
sorted(a)     == [0,1,2,3,4]
```

Create your own functions

- Define own functions for repeating tasks
- reduce amount of code lines

```
def my_function(a,b):
    c = ...
    return c
```

Create your own functions

- Define own functions for repeating tasks
- reduce amount of code lines

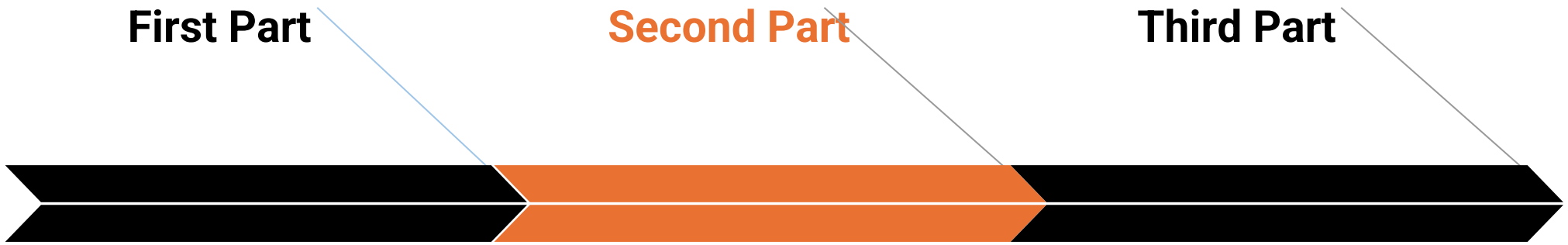
```
def my_function(a,b):
    return a + b
my_function(1,2) # 3
```

Exercise Time

Convert your code which counts amount of numbers smaller than 5 into a function

```
def smaller_than(numbers, value):
    # your code here
    return counter
```

Course Structure



Python Fundamentals

Basic concepts, Variables, basic data structures, functions

Data Wrangling & Simple visualizations

How to process data with pandas and visualize it with matplotlib

Visualizations & Modelling

More plots with matplotlib and seaborn and an introduction to modelling

What you learn in this Chapter...

... how to use Python libraries

... work with data in a DataFrame

... filter, merge and group your data

... visualize data with simple plots

Data Wrangling & Visualization

Libraries

Libraries

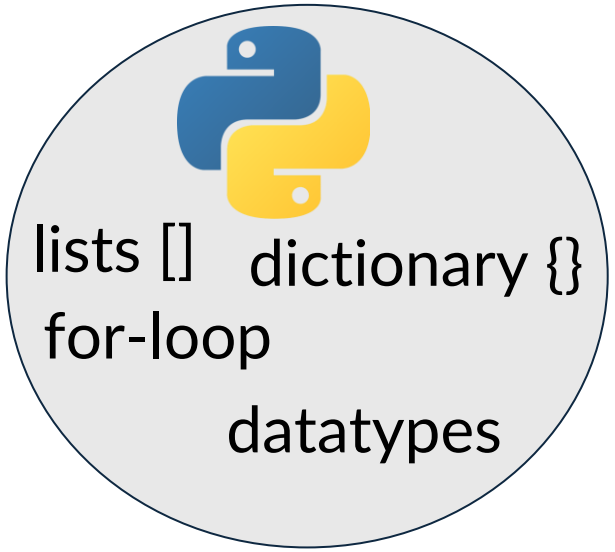
- A collection of functions is bundled in a **library**
- we import these libraries and can use the defined functions
- Some libraries come with a Python installation, some need to be installed

… for plotting and visualization

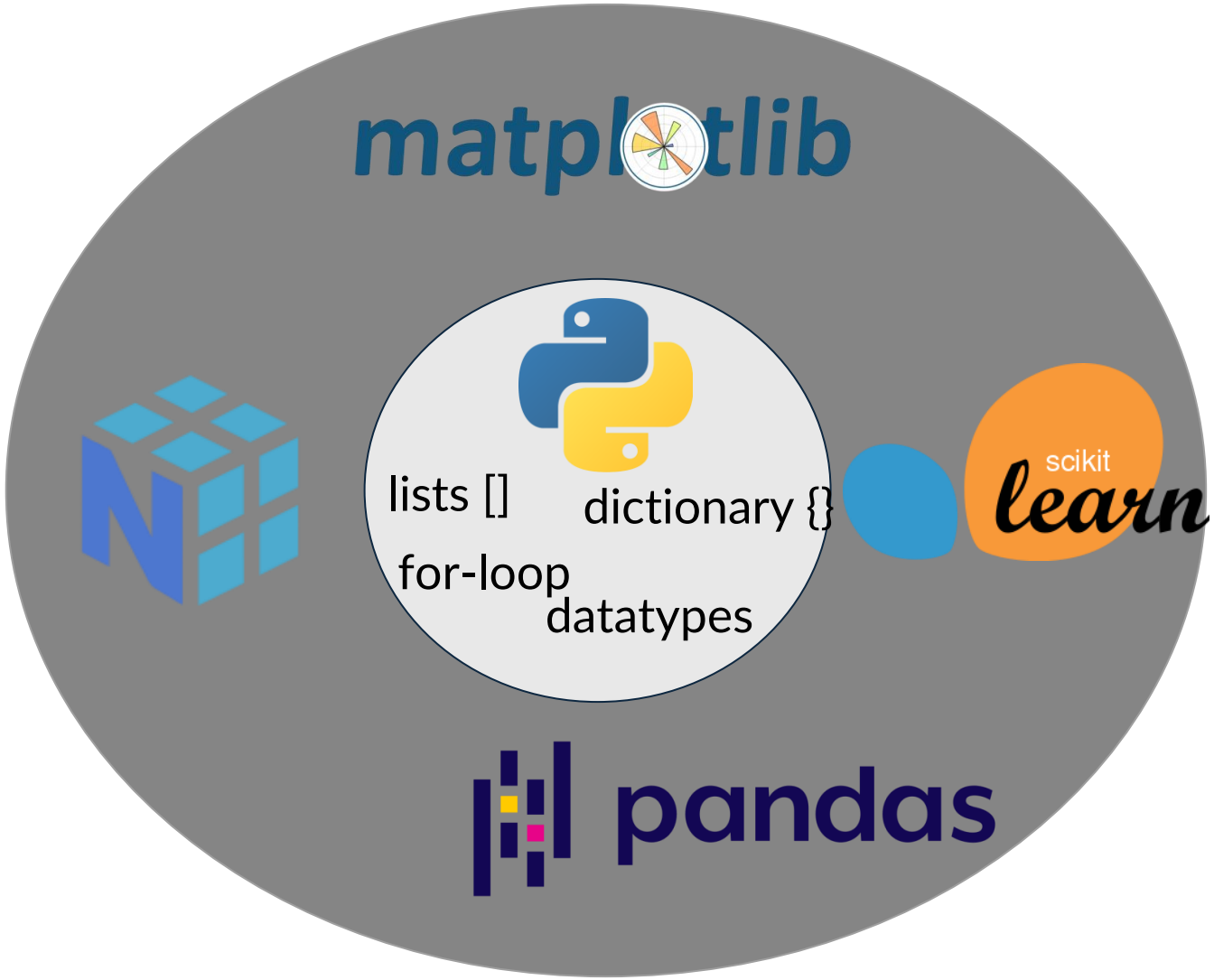
… for working with tabular data (Excel-files, csv-files,…)

… creating machine learning models

Libraries



Libraries


matplotlib
lists []
dictionary {}
for-loop
datatypes
scikit
learn
pandas

Function & Methods

```
a = function_name(parameter)
```

Function & Methods

Functions

```
a = function_name(parameter)
```

Methods

```
a = "a string!"
a = a.upper() # ‘A STRING!’
```

Import Libraries

```
import library

a = library.function_name()
```

Import Libraries

```
import library as l

a = l.function_name()
```

Data Wrangling & Visualization

Arrays with Numpy

NumPy - library

- Library for scientific computing
- Work with lists, matrices or higher dimensional structures
- NumPy lists have much more functionality than usual lists

```
import numpy as np

a = np.array([1,2,3,4])
```

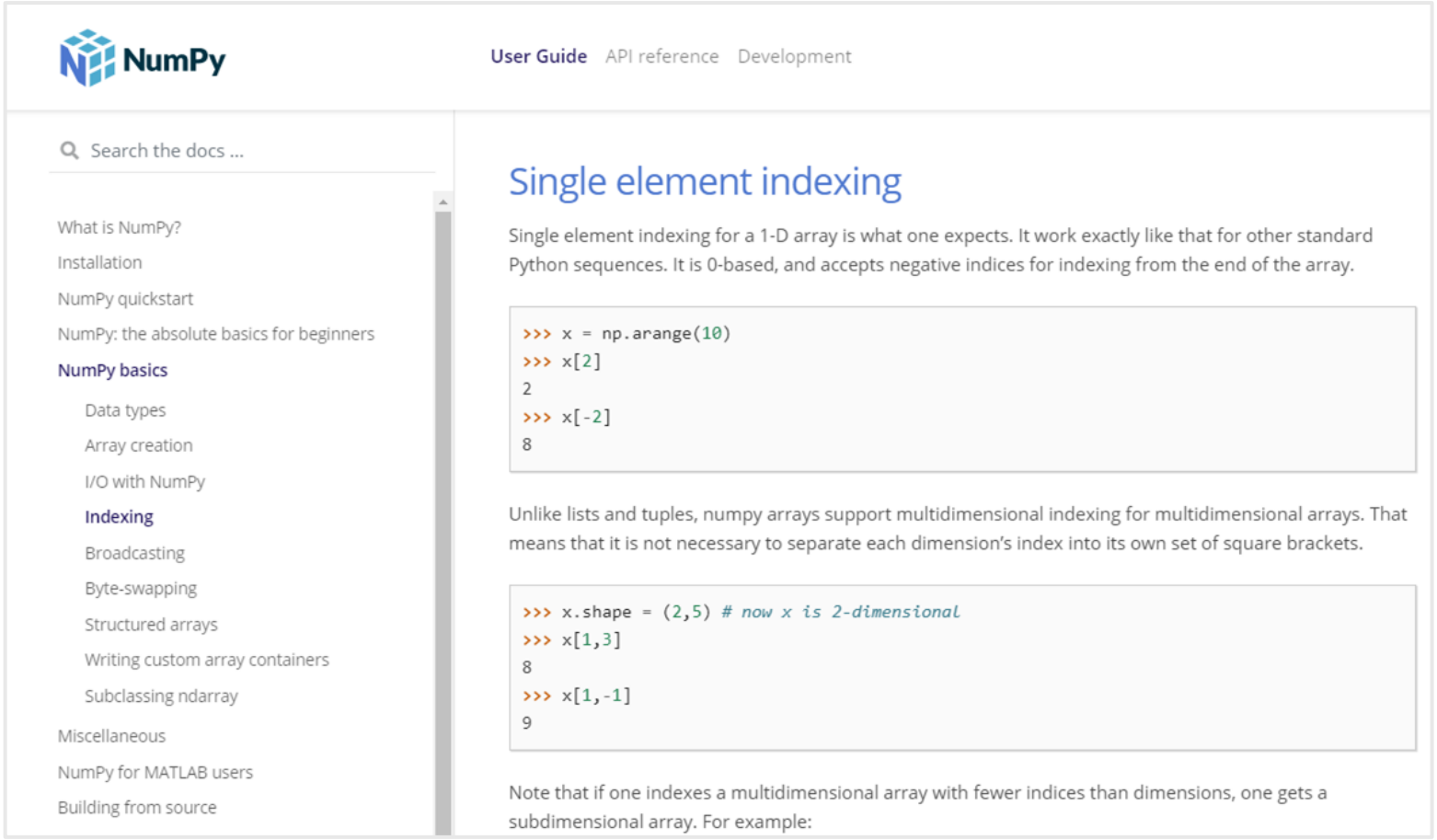
NumPy - library

- Library for scientific computing
- Work with lists, matrices or higher dimensional structures
- NumPy lists have much more functionality than usual lists

```
import numpy as np

a = np.array([1,2,3,4])
a.sum() # 10
a.mean() # 2.5
a.std() # 1.118...
```

Documentation

- Explanations to the methods and functions
- Often include examples and tutorials
- https://numpy.org/doc/



NumPy - Append

```
import numpy as np

a = np.array([1,2,3,4])
np.append(a, [5,6,7]) # array([1, 2, 3, 4, 5, 6, 7])
```

NumPy - Append

```
import numpy as np

a = np.array([1,2,3,4])
np.append(a, [5,6,7]) # array([1, 2, 3, 4, 5, 6, 7])
a = np.append(a, [5,6,7])
```

Compute with arrays - Broadcasting

```python
import numpy as np
a = np.array([1,2,3])
a * 2 # array([2, 4, 6])
a ** 2 # array([1, 4, 9])
a - 1 # array([0, 1, 2])
```

Compute with arrays

```
import numpy as np
a = np.array([1,2,3])
b = np.array([1,1,1])
a + b  # array([2, 3, 4])
```

Indexing

```
import numpy as np

a = np.array([1,2,3,4])
a[0:2] # array([1,2])
```

Boolean Indexing

```
import numpy as np

a = np.array([1,2,3,4])
a[[True,True,False,False]] # array([1,2])
```

Boolean Indexing

```
import numpy as np

a = np.array([1,2,3,4])
a <= 2 # [True, True, False, False]
```

Boolean Indexing

```python
import numpy as np

a = np.array([1,2,3,4])
a <= 2 # [True, True, False, False]
a[a<=2] # array([1,2])
```

Exercise Time

```
import numpy as np

a = np.array([1,2,3,4])
a = np.append(a, [5,6,7])# [1,2,3,4,5,6,7]
a[a<=2] # [1,2] (boolean indexing)
a.sum() # 28
```

Data Wrangling & Visualization

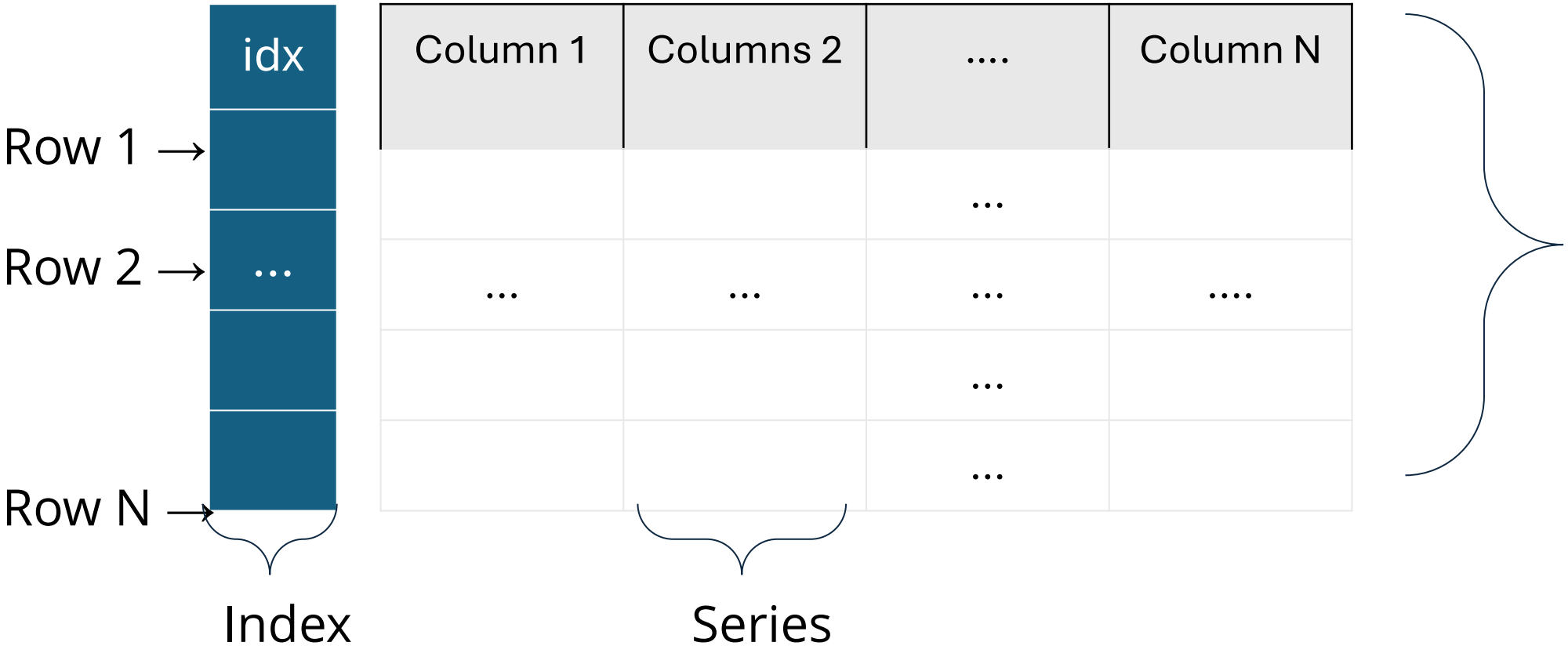
Tabular Data with Pandas

pandas - library

```
import pandas as pd

data = {
"Name":["Clara","Tom","Sarah","John"],
"Age" :[20,24,19,21]}

df = pd.DataFrame(data)
```

	Name	Age
0	Clara	20
1	Tom	24
2	Sarah	19
3	John	21

DataFrame - data structure



Read data

- Pandas can read data from files
- Various data formats possible

```
df = pd.read_csv('path_to_file')
df = pd.read_excel('path_to_file')
df = pd.read_sql_table('postgres://db')
```

First look at the data

```
df.head()
```

	id	price	neighbourhood_group_cleansed	latitude
0	28684898	$50.00	Neukölln	52.473978
1	22607348	$10.00	Treptow - Köpenick	52.468095
2	21019199	$35.00	Neukölln	52.481810
3	21919556	$99.00	Pankow	52.537269
4	4820648	$39.00	Friedrichshain-Kreuzberg	52.491483

```
df.describe()
```

	id	price	latitude	longitude	bathrooms
count	1.353100e+04	13531.000000	13531.000000	13531.000000	13508.000000
mean	1.573089e+07	70.082625	52.509956	13.405871	1.095203
std	8.580394e+06	255.451132	0.030773	0.058517	0.335469
min	2.695000e+03	0.000000	52.346203	13.103557	0.000000
25%	8.041528e+06	30.000000	52.489082	13.374950	1.000000
50%	1.697254e+07	45.000000	52.509229	13.416764	1.000000
75%	2.264464e+07	70.000000	52.532808	13.439258	1.000000
max	2.986735e+07	9000.000000	52.651670	13.721671	8.500000

Select data in a DataFrame

idx	Name	Age
0	Clara	20
1	Tom	24
2	Sarah	19
3	John	21

```
df = pd.DataFrame(data)
```

Select data in a DataFrame

idx	Name	Age
0	Clara	20
1	Tom	24
2	Sarah	19
3	John	21

```
df = pd.DataFrame(data)
df["Name"]
```

Select data in a DataFrame

idx	Name	Age
0	Clara	20
1	Tom	24
2	Sarah	19
3	John	21

```
df = pd.DataFrame(data)
df.iloc[0:3]
```

Add data to a DataFrame

idx	Name	Age	Grade
0	Clara	20	1
1	Tom	24	2
2	Sarah	19	3
3	John	21	4

```
df["Grade"] = [1,2,3,4]
```

Compute with data in a DataFrame

idx	Name	Age	Grade
0	Clara	20	2
1	Tom	24	3
2	Sarah	19	4
3	John	21	5

```
df["Grade"] = df["Grade"] + 1
```

Drop data

idx	Name	Age	Grade
0	Clara	20	1
1	Tom	24	2
2	Sarah	19	3
3	John	21	4

```python
# Drop column
df = df.drop(column="Grade")
```

Drop data

idx	Name	Age	Grade
2	Sarah	19	3
3	John	21	4

```python
# Drop column
df = df.drop(column="Grade")
# Drop row
df.drop([0,1], inplace=True)
```

Filter data - select a subset of the data

idx	Name	Age	Grade
0	Clara	20	2
2	Sarah	19	4

```
keep_rows = [True, False,
                   True,False]

df[keep_rows]
```

Filter data - select a subset of the data

idx	Name	Age	Grade
0	Clara	20	2
1	Tom	24	3
2	Sarah	19	4
3	John	21	5

```
df["Age"] <= 20
→ [True, False, True, False]
```

Filter data - select a subset of the data

idx	Name	Age	Grade
0	Clara	20	2
2	Sarah	19	4

```
df[ df["Age"] <= 20 ]
```

```
[True, False, True, False]
```

Filter data - select a subset of the data

idx	Name	Age	Grade
0	Clara	20	2

```
df[
    (df["Age"] <= 20) &
    (df["Grade"] < 3)
  ]
```

Filter data - select a subset of the data

idx	Name	Age	Grade
0	Clara	20	2
2	Sarah	19	4

```
df[
    (df["Age"] == 20) |
    (df["Grade"] == 4)
  ]
```

Combine filtering with other methods

idx	Name	Age	Grade
0	Clara	20	2
2	Sarah	19	4

```
df[df["Age"] <= 20]
```

This is again a DataFrame

Combine filtering with other methods

idx	Name	Age	Grade
0	Clara	20	2
2	Sarah	19	4

```
df[df["Age"]<=20]["Age"].mean()
```

Exercise Time

```
import pandas as pd

df = pd.DataFrame(data) # create dataframe
df['new_column'] = [1,2,3]
df.sort_values(by='column_name') # sort
df[df['age']<=20] # filter data
```

Apply pandas methods

- Pandas has large amount of commonly used methods
- Can be applied to single column or whole data frame

```
df["Grade"].mean()
df["Grade"].std()
df["Grade"].sum()
```

Often used methods

```
df["Grade"].mean()
df["Grade"].sum()
df["Grade"].value_counts()
df.sort_values(by="Grade")
df.groupby(by="Grade").sum()
df1.merge(df2)
df.drop(colums=["Grade"])
df["Grade"].replace(5,"Failed")
```

Group-By

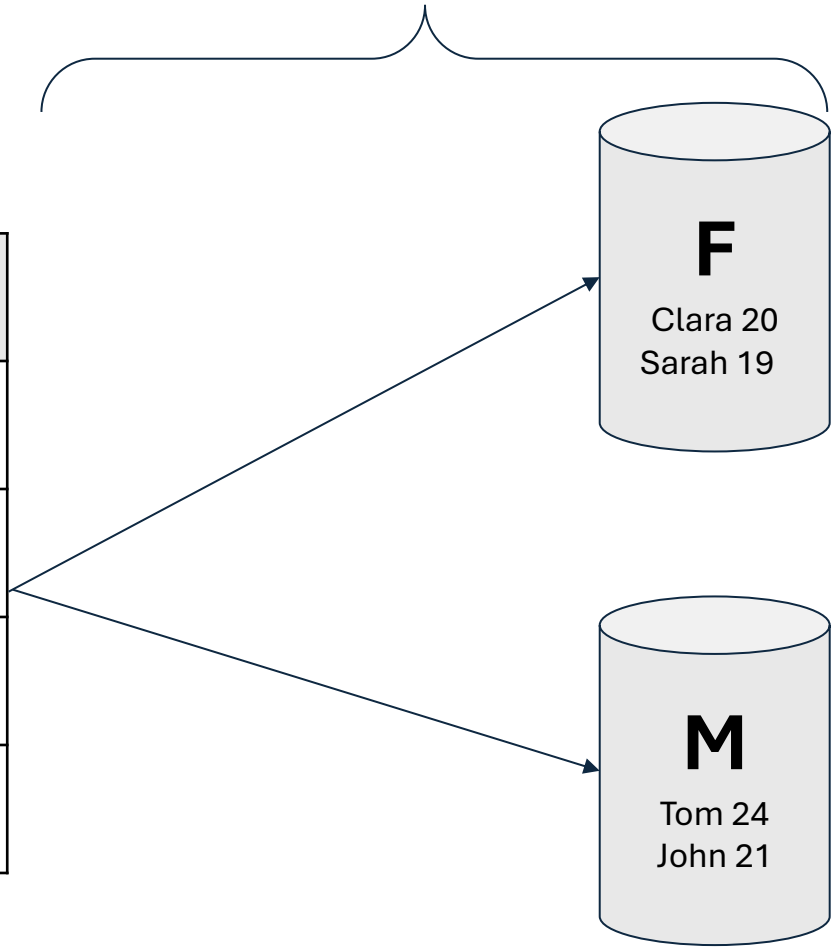
idx	Name	Age	Gender
0	Clara	20	F
1	Tom	24	M
2	Sarah	19	F
3	John	21	M

What is the average age per gender?

Group-By

```
df.groupby(by='Gender').mean()
```

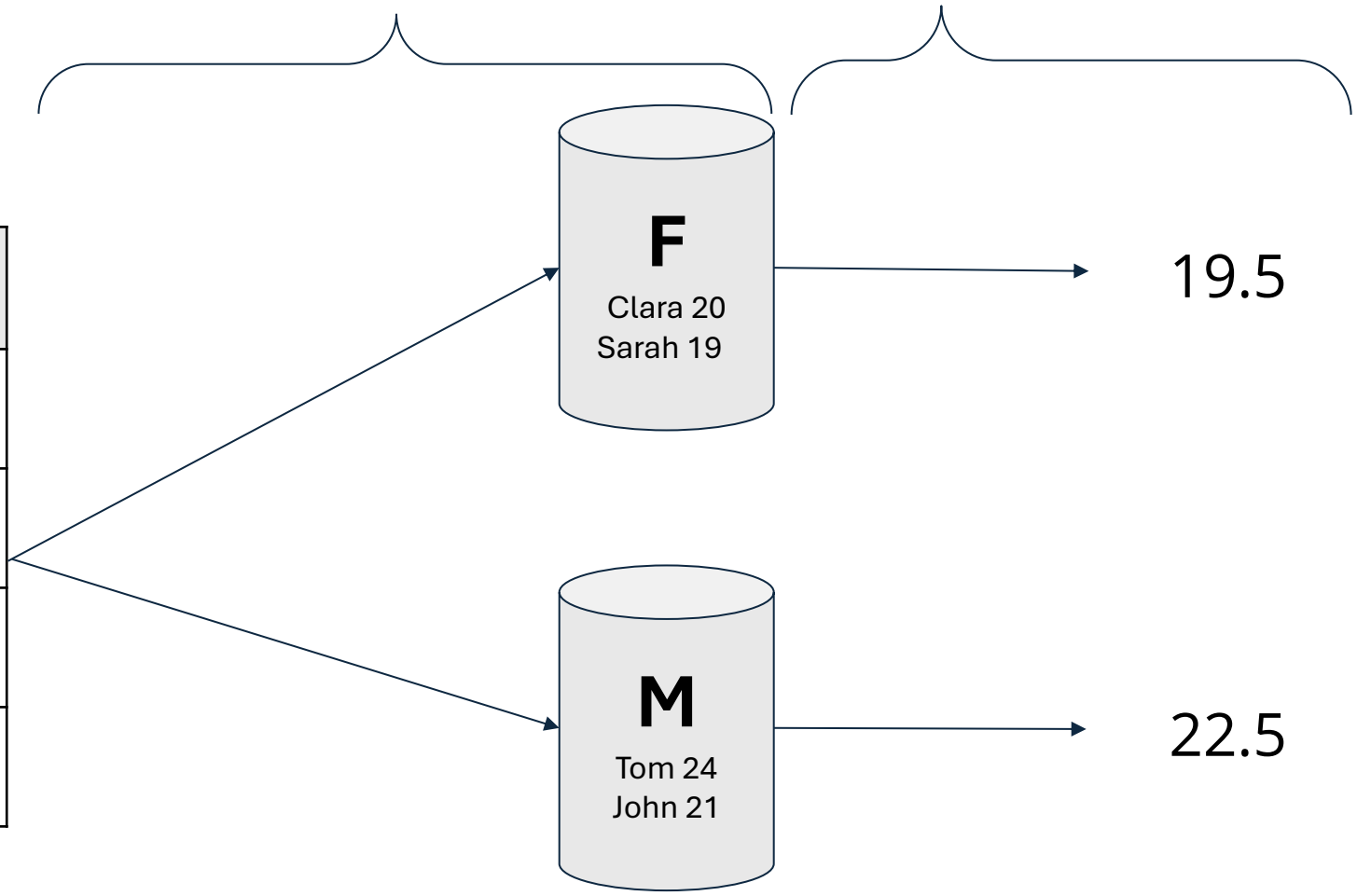
idx	Name	Age	Gender
0	Clara	20	F
1	Tom	24	M
2	Sarah	19	F
3	John	21	M



Group-By

```
df.groupby(by='Gender').mean()
```

idx	Name	Age	Gender
0	Clara	20	F
1	Tom	24	M
2	Sarah	19	F
3	John	21	M



Merge DataFrame's

Name	Age	Grade
Clara	20	2
Tom	24	3
Sarah	19	4

Name	Subject
Sarah	Physics
Tom	Politics
John	English

Merge DataFrame's

Name	Age	Grade
Clara	20	2
Tom	24	3
Sarah	19	4

Name	Subject
Sarah	Physics
Tom	Politics
John	English

Name	Age	Grade	Subject
Tom	24	3	Politics
Sarah	19	4	Physics

Merge DataFrame's

Name	Age	Grade
Clara	20	2
Tom	24	3
Sarah	19	4

```
df1.merge(df2, on='Name')
```

Name	Subject
Sarah	Physics
Tom	Politics
John	English

Name	Age	Grade	Subject
Tom	24	3	Politics
Sarah	19	4	Physics

Exercise Time

```
import pandas as pd

df = df.groupby(by='column_name')
df = df.merge(df2,on='column_name',how='left')
```

Data Wrangling & Visualization

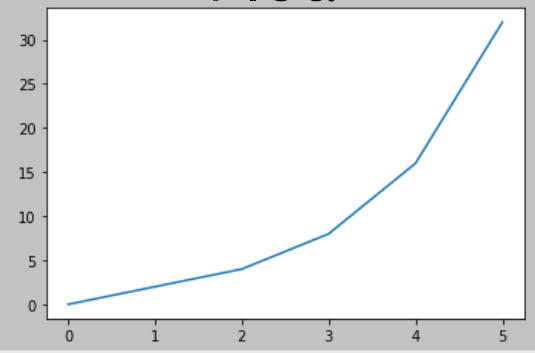
Visualizations with Matplotlib

matplotlib - library

- data visualization tool
- generate highly customizable plots
- good integration with pandas

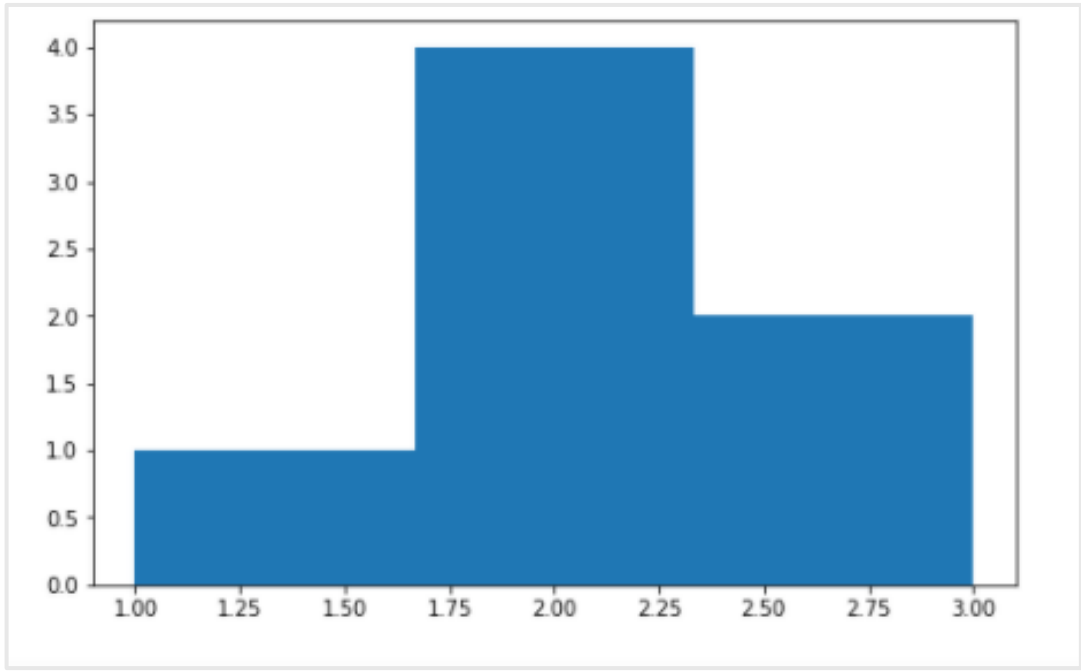
```
import matplotlib.pyplot as plt
x = [0,1,2,3,4,5]
y = [0,2,4,8,16,32]
plt.plot(x,y)
```

Plot:



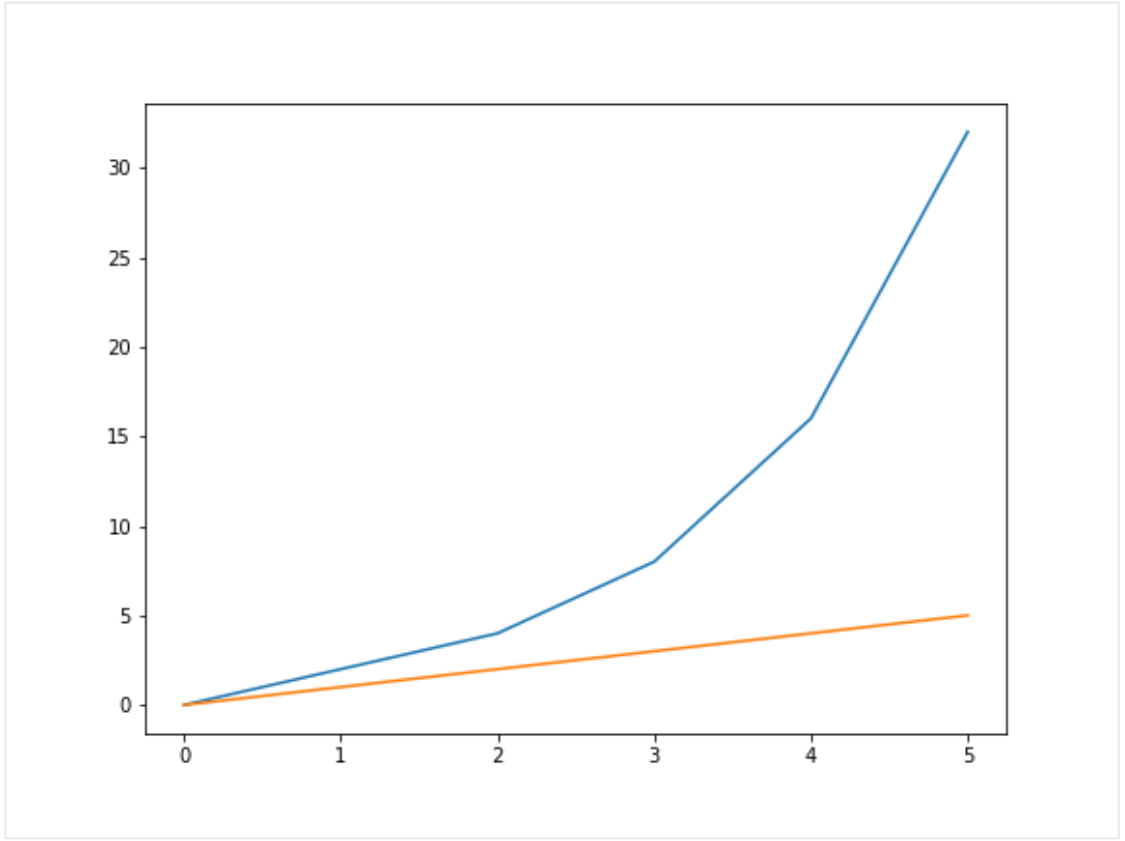
Histogram

```
import matplotlib.pyplot as plt
x = [1,2,2,2,2,3,3]
plt.hist(x, bins=3)
```



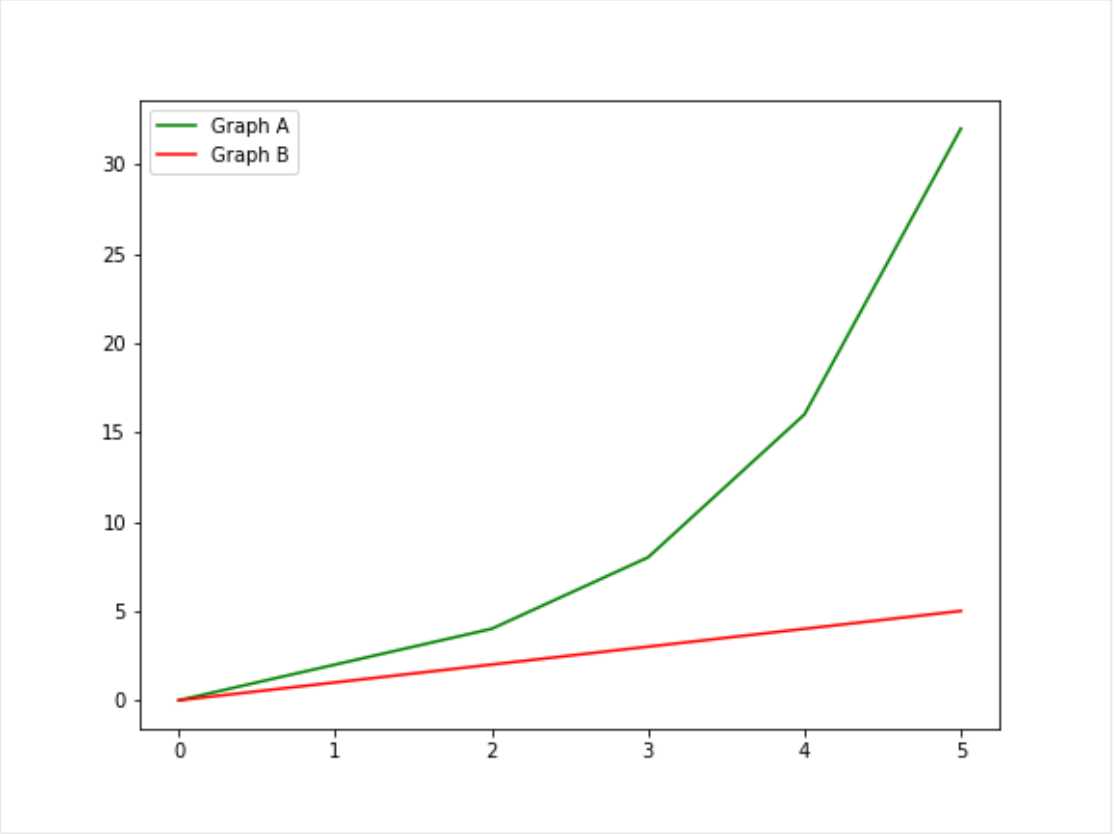
Customize plots

```
import matplotlib.pyplot as plt
x = [0,1,2,3,4,5]
y = [0,2,4,8,16,32]
y2 = [0,1,2,3,4,5]

plt.plot(x,y)
plt.plot(x,y2)
```



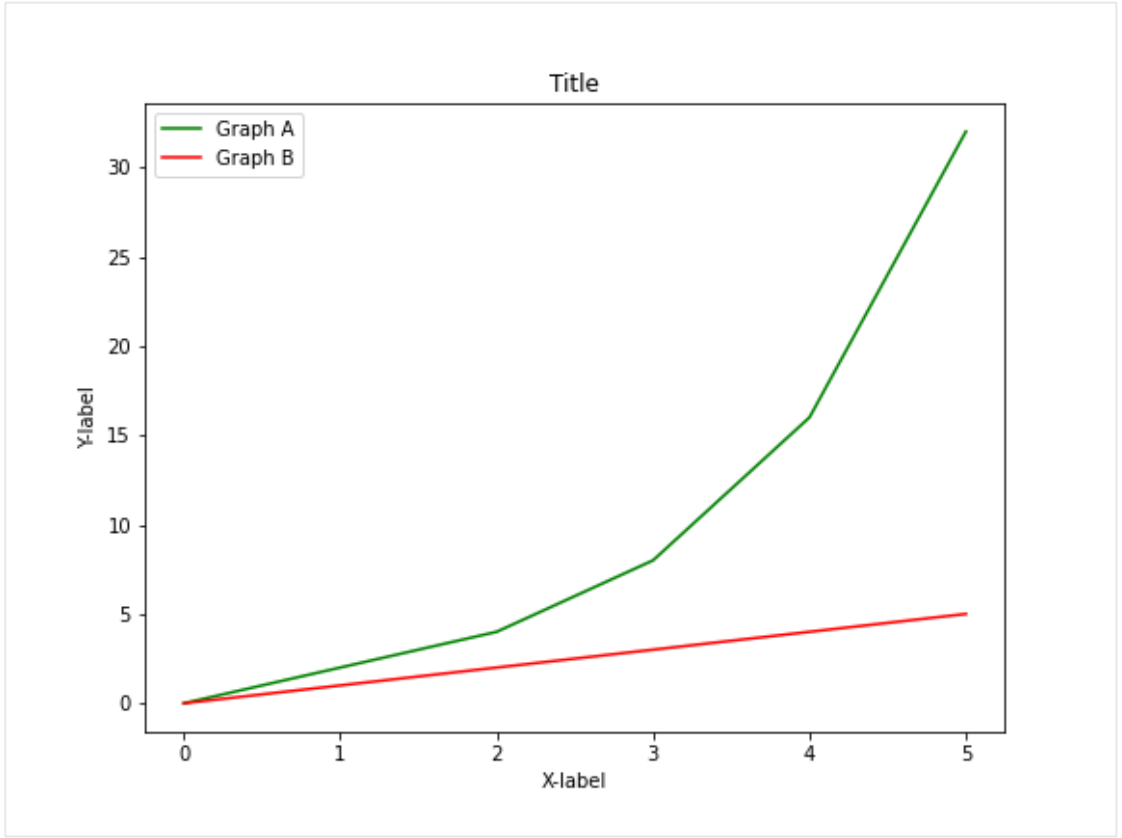
Customize plots

```
import matplotlib.pyplot as plt
x = [0,1,2,3,4,5]
y = [0,2,4,8,16,32]
y2 = [0,1,2,3,4,5]

plt.plot(x,y, color='green', label='Graph A')
plt.plot(x,y2, color='red', label='Graph B')

plt.legend()
```



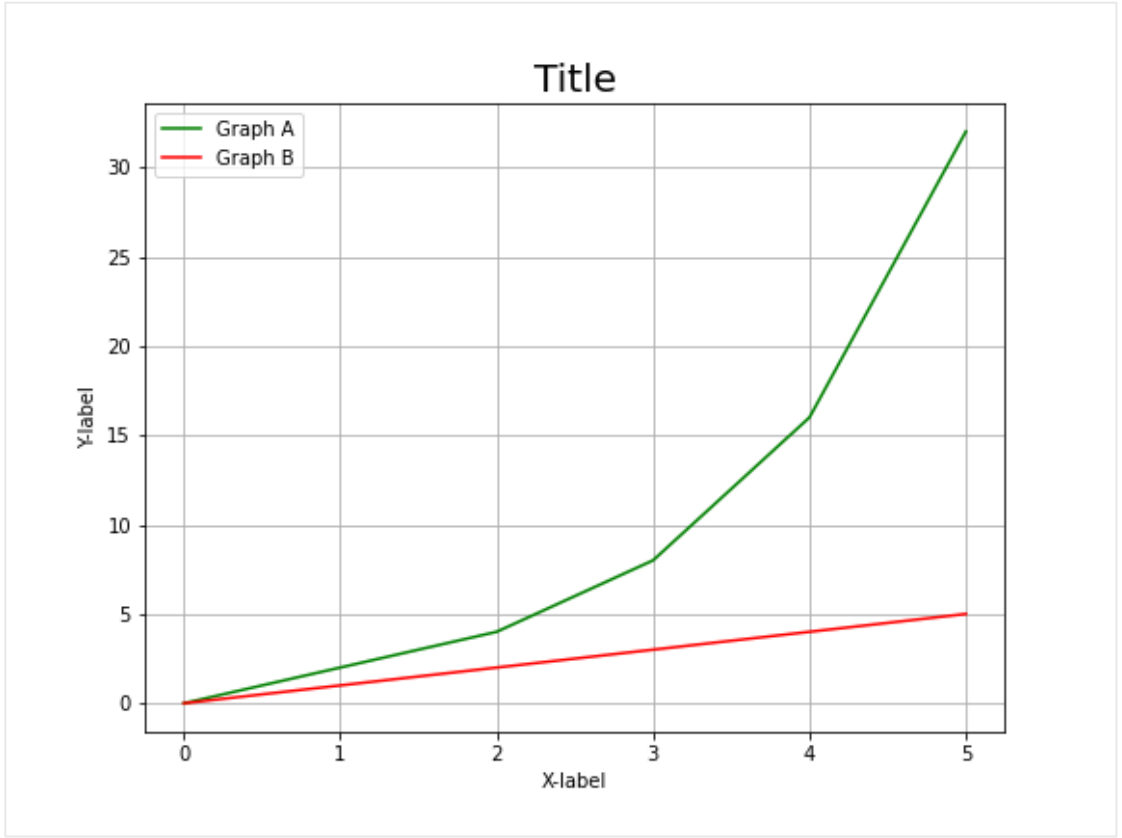
Customize plots

```
import matplotlib.pyplot as plt
x = [0,1,2,3,4,5]
y = [0,2,4,8,16,32]
y2 = [0,1,2,3,4,5]

plt.plot(x,y, color='green', label='Graph A')
plt.plot(x,y2, color='red', label='Graph B')

plt.legend()
plt.xlabel("X-label")
plt.ylabel("Y-label")
plt.title("Title")
```



Customize plots

```
import matplotlib.pyplot as plt
x = [0,1,2,3,4,5]
y = [0,2,4,8,16,32]
y2 = [0,1,2,3,4,5]

plt.plot(x,y, color='green', label='Graph A')
plt.plot(x,y2, color='red', label='Graph B')

plt.legend()
plt.xlabel("X-label")
plt.ylabel("Y-label")
plt.title("Title",
{'fontname':'DejaVu Sans', 'size':'20'})
plt.grid()
```



pandas & matplotlib

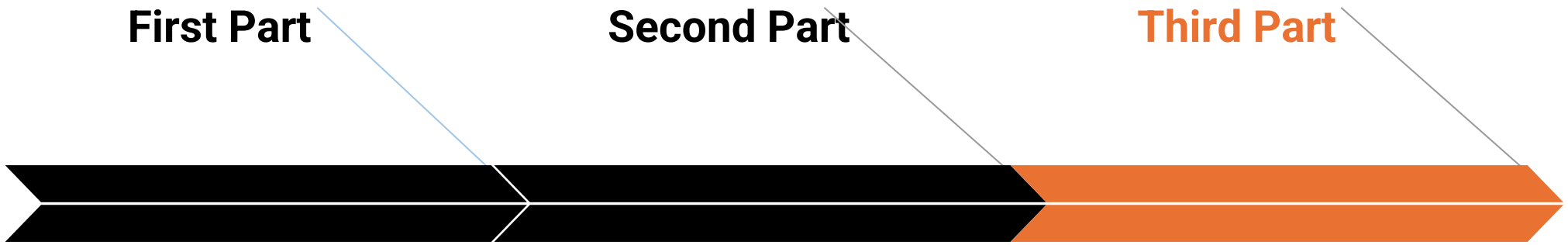
- Pandas and matplotlib work very well together
- We can pass columns of a DataFrame to matplotlib

```
import matplotlib.pyplot as plt
df = pd.DataFrame(data)
plt.hist(df["Age"])
```

Exercise Time

```
import matplotlib.pyplot as plt
df = pd.DataFrame(data)
plt.hist(df["Age"])
```

Course Structure



Python Fundamentals

Basic concepts, Variables, basic data structures, functions

Data Wrangling & Simple visualizations

How to process data with pandas and visualize it with matplotlib

Visualizations & Modelling

More plots with matplotlib and seaborn and an introduction to modelling

Advanced Visualisations & Modelling

Clean Data with Pandas

Clean data

Name	Town
Clara	Frankfurt a.M.
Sarah	Frankfurt am Main
John	Berlin

```
df1['Town'] = df1['Town'].str.replace(
                          'a.M.','am Main')
```

Clean data

Name	Subject
Clara	Physics
Sarah	physics
John	Math

```
df['Subject'] = df['Subject'].str.lower()
```

Fill missing values

Name	Score
Clara	10
Sarah	5
John	NaN

```
df['Score'] = df['Score'].fillna(0)
```

Fill missing values

Name	Score
Clara	10
Sarah	5
John	NaN → 0

```
df['Score'] = df['Score'].fillna(0)
```

Fill missing values

Name	Score
Clara	10
Sarah	5
John	NaN → 7.5

```
df['Score'] = df['Score'].fillna(df.mean())
```

Drop missing values

Name	Score
Clara	10
Sarah	5
John	NaN

```
df = df.dropna()
```

Working with dates

What kind of data type is this : `"27-03-2021"`

a) integer b) float c) string d) date

Transform string to datetime object

```
df['Birthday'] = pd.to_datetime(df['Birthday'])
```

Name	Birthday
Clara	“10/10/1995”
Sarah	“01/10/1999”
John	“03/05/2001”

Transform string to datetime object

```
df['Birthday'] = pd.to_datetime(df['Birthday'], format="%d/%m/%Y")
```

Name	Birthday
Clara	“10/10/1995”
Sarah	“01/10/1999”
John	“03/05/2001”

Transform string to datetime object

```
df['Birthday'] = pd.to_datetime(df['Birthday'], format="%m-%d-%Y")
```

Name	Birthday
Clara	“10-10-1995”
Sarah	“10-01-1999”
John	“05-03-2001”

Transform string to datetime object

```
df['day'] = df['Birthday'].dt.day
df['weekday'] = df['Birthday'].dt.weekday
df['month'] = df['Birthday'].dt.month
```

Name	Birthday	day	weekday	month
Clara	“10-10-1995”	10	1	10
Sarah	“10-01-1999”	1	4	10
John	“05-03-2001”	3	3	5

Set date as index

date	City	Temperature
“2021-04-20”	‘Frankfurt’	10
“2021-04-21”	‘Frankfurt’	11
“2021-04-22”	‘Frankfurt’	12

```
df = df.set_index(['date'])
```

Select timespans

When the index is in datetime format, you can access the data in the following way:

date	City	Temperature
“2021-04-20”	‘Frankfurt’	10
“2021-04-21”	‘Frankfurt’	11
“2021-04-22”	‘Frankfurt’	12

```
df['2021'] # all data from 2021
df['2021-04':'2021-05']
df['2021':]
```

Exercise Time

```
df['date'] = pd.to_datetime(df['date'])
df = df.set_index(['date'])
df['2021-04':'2021-05']
df[df['col_name'] == 'some condition']


plt.plot(df['col_name']['2000'])
```

pandas – Documentation & Scope

- The official documentation of pandas has more than 3000 pages
- work with missing values
  - fill with mean value
  - interpolate between values
  - fill with last value
- Read data from various sources
  - Excel, CSV, SQL, Stata, SPSS, SAS, HTML Tables from websites
- Windowing functions
  - Moving average,...

Advanced Visualisations & Modelling

Advanced Visualizations with Seaborn

seaborn - library

- statistical data visualization tools
- based on matplotlib
- makes it easy to create more sophisticated plots
- best to visualize relations between columns of a dataset

```
import seaborn as sns
```

The penguins dataset

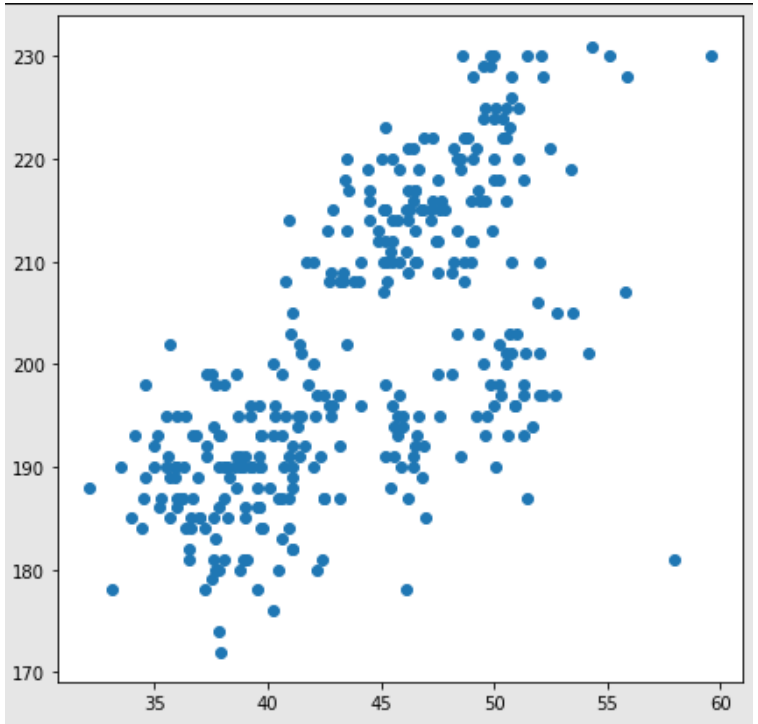
	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	Male
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	Female
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	Female
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	Female

The penguins dataset

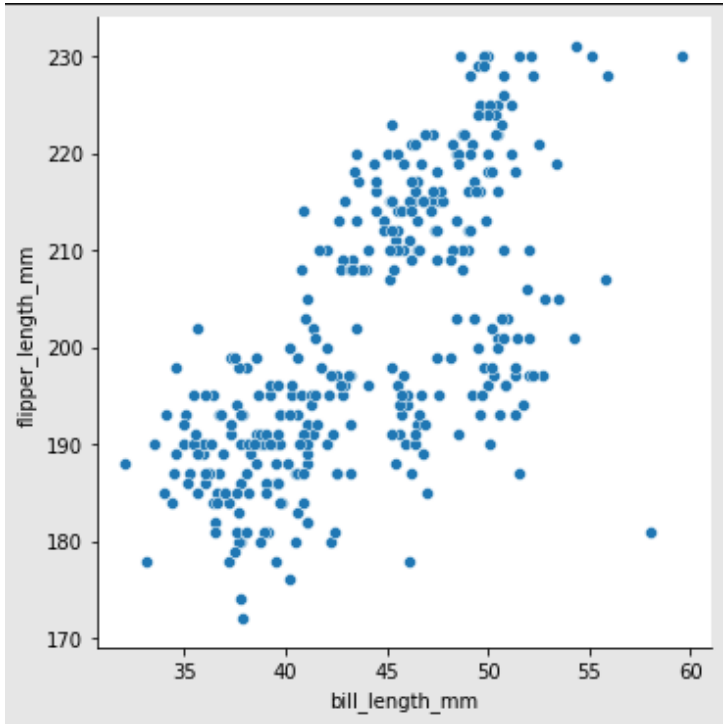
	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	Male
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	Female
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	Female
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	Female

Matplotlib vs Seaborn

```
plt.scatter(df['bill_length_mm'],df['flipper_length_mm'])
```




```
sns.relplot(df['bill_length_mm'],df['flipper_length_mm'])
```

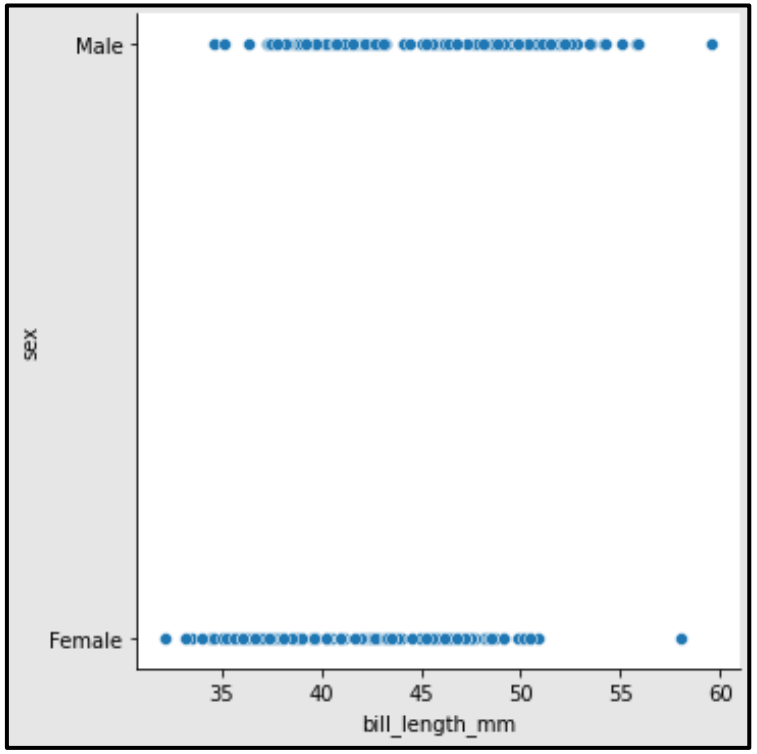


Matplotlib vs Seaborn

```
plt.scatter(df['bill_length_mm'],df[sex])
```

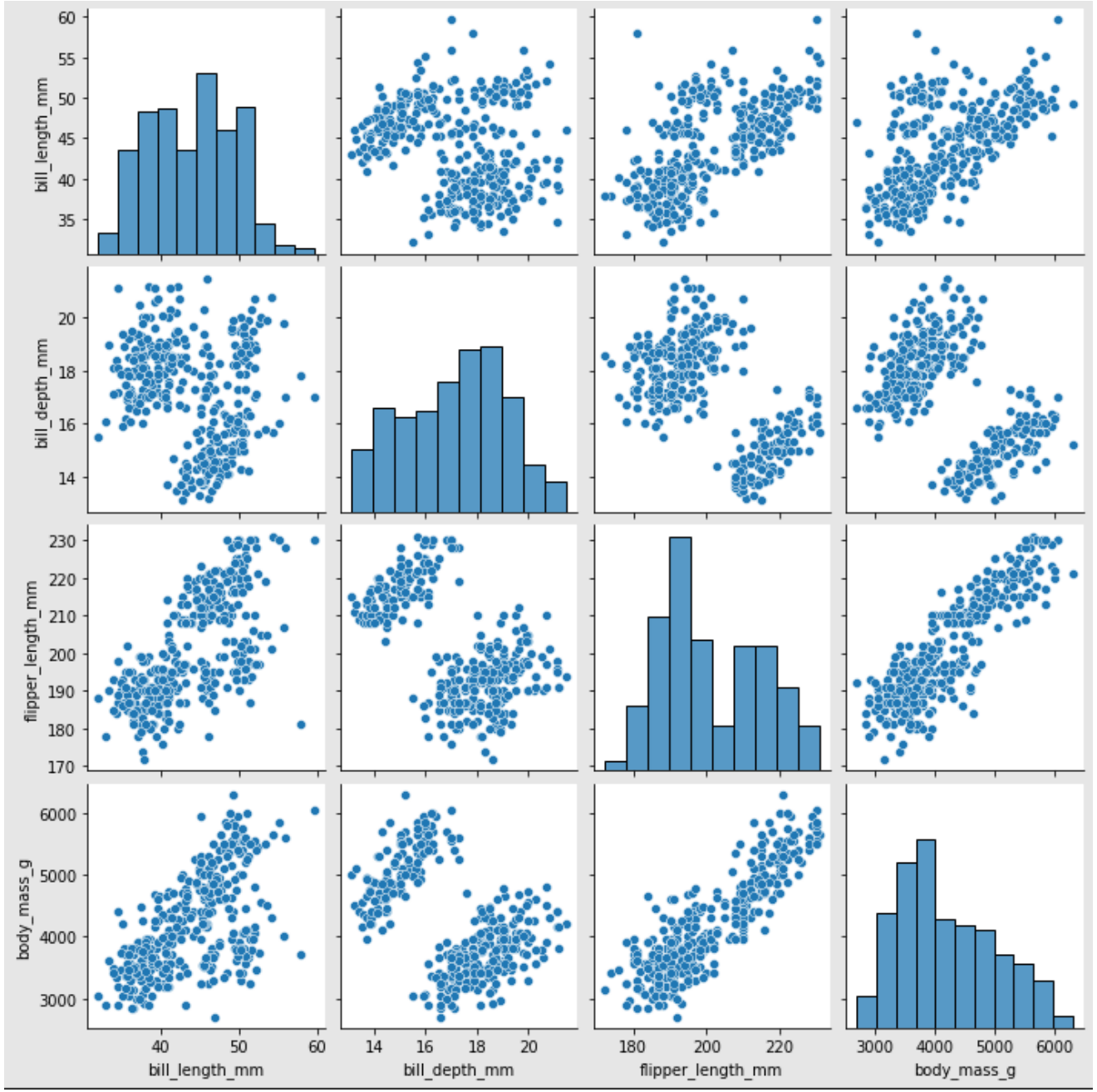
```
TypeError
```

```
sns.relplot(df['bill_length_mm'],df[sex])
```



Pairplot

```
sns.pairplot(df)
```

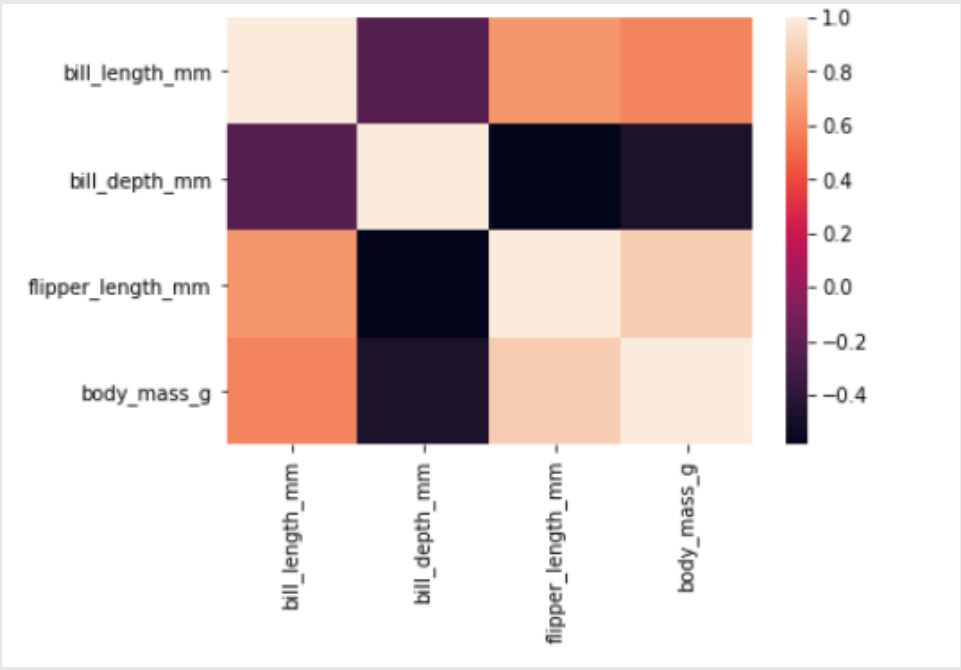


Visualize correlations

	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
bill_length_mm	1.000000	-0.235053	0.656181	0.595110
bill_depth_mm	-0.235053	1.000000	-0.583851	-0.471916
flipper_length_mm	0.656181	-0.583851	1.000000	0.871202
body_mass_g	0.595110	-0.471916	0.871202	1.000000

```
df.corr()
```

Visualize correlations




```
sns.heatmap(df.corr())
```

Exercise Time

```
sns.pairplot(df)
sns.pairplot(df, hue='col_name')
```

Matplotlib & Seaborn Gallery

- Documentation contains example plots
- Each plot comes with the corresponding code
- https://matplotlib.org/stable/gallery/index.html
- https://seaborn.pydata.org/examples/index.html

Advanced Visualisations & Modelling

Modelling Basics

Linear Regression

Source: https://avantecture.com/p/phoenixsee/

How expensive is this house?

Based on information like:

- amount of rooms
- location
- size

Housing regression

Old collected data

House	Location	Rooms	Price
A	Berlin	6	500k €
B	Frankfurt	8	600k €
C	Berlin	7	300k €

New data

House	Location	Rooms	Price
D	Frankfurt	8	?
E	Frankfurt	5	?
F	Berlin	4	?

Housing regression

Old collected data

House	Location	Rooms	Price
A	Berlin	6	500k €
B	Frankfurt	8	600k €
C	Berlin	7	300k €

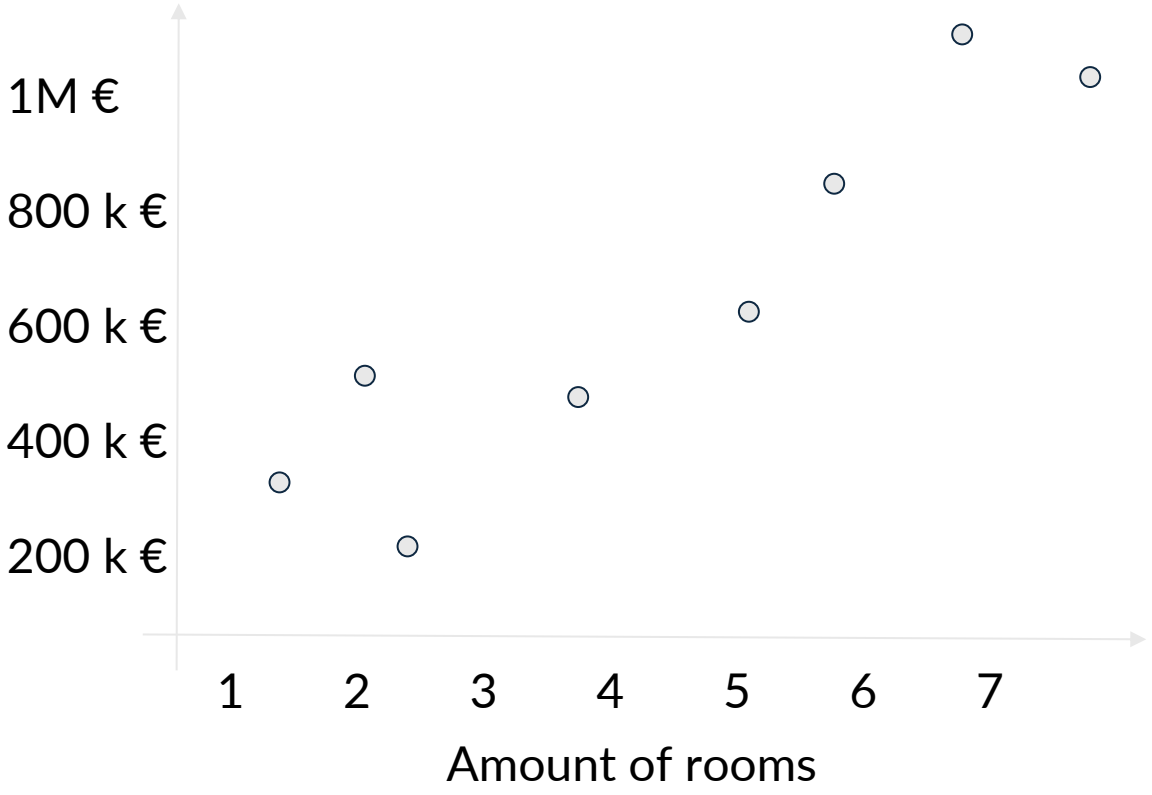
Features

Target column

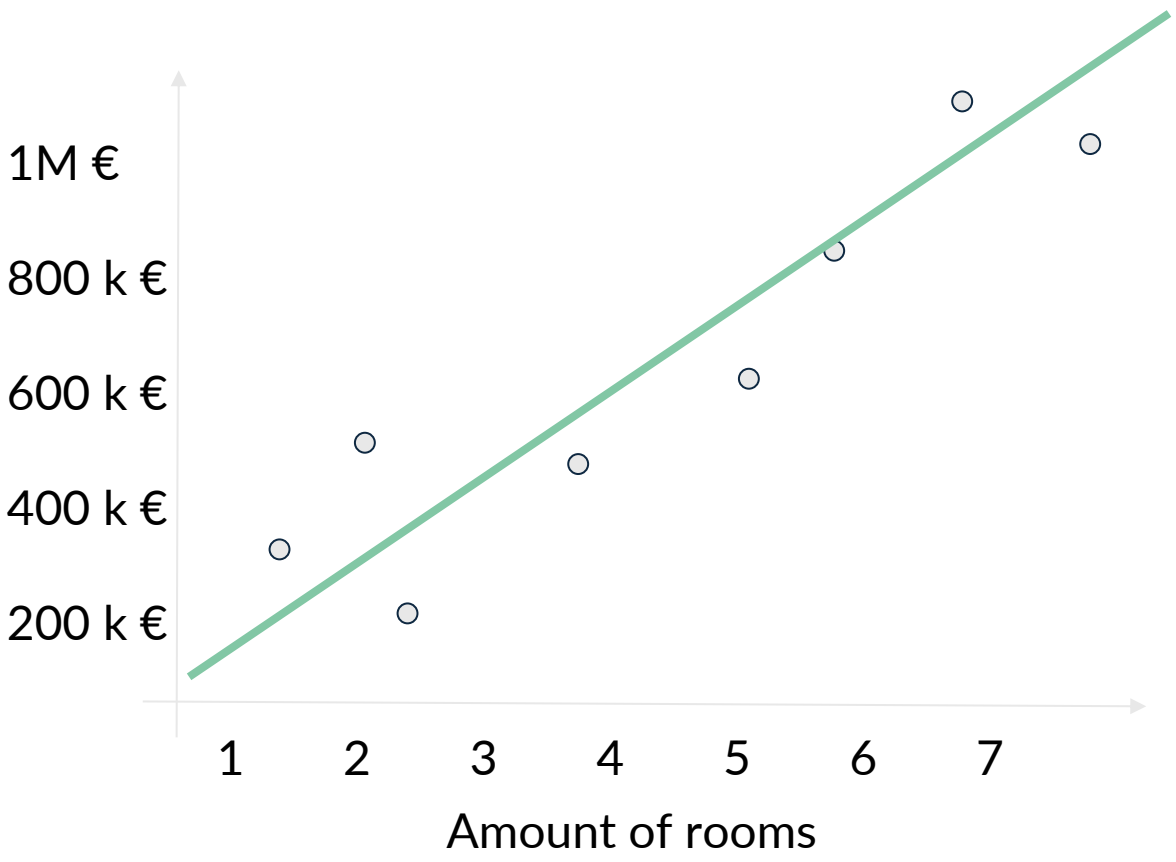
New data

House	Location	Rooms	Price
D	Frankfurt	8	?
E	Frankfurt	5	?
F	Berlin	4	?

Linear Regression

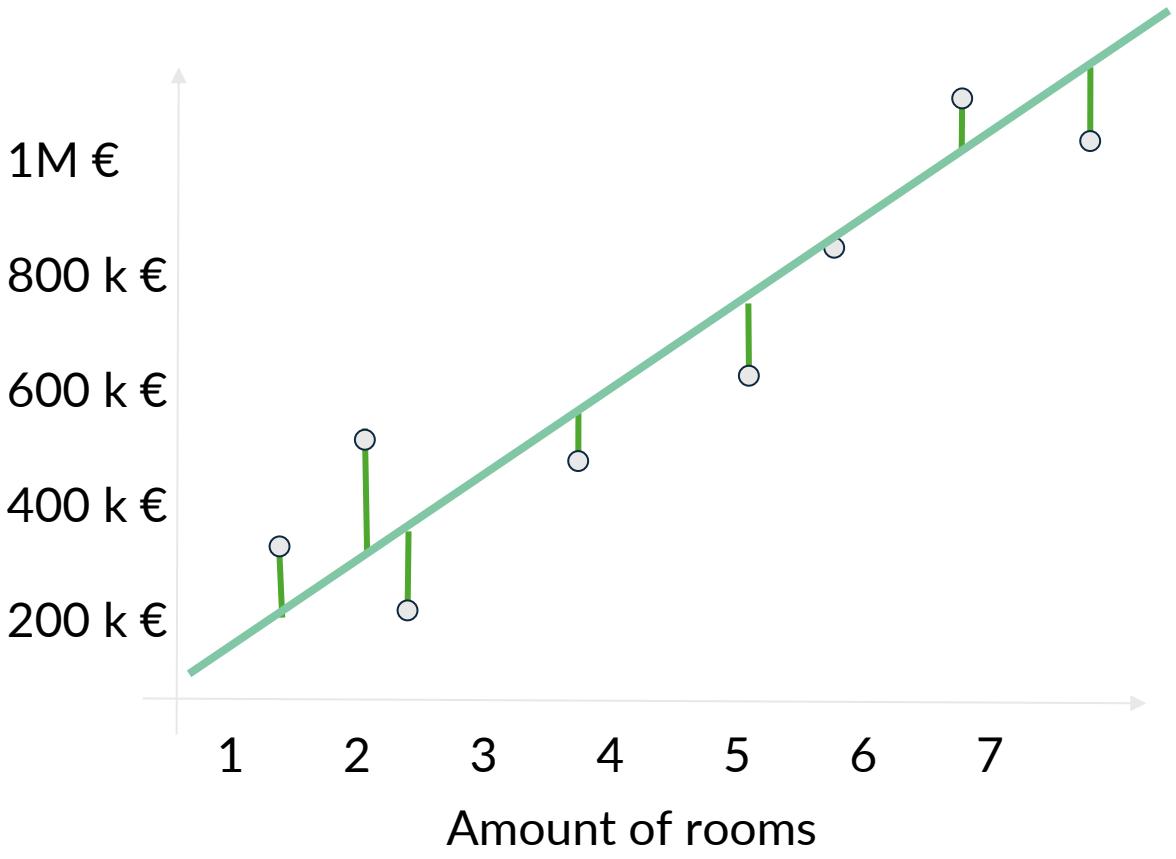


Linear Regression



$$price = a \cdot rooms + b$$

Linear Regression



$$price = a \cdot rooms + b$$

RMSE = $$\sqrt{\frac{1}{T}\sum_{i=1}^{T}(y_{i,predicted} - y_{i,true})^2)}$$

Linear Regression

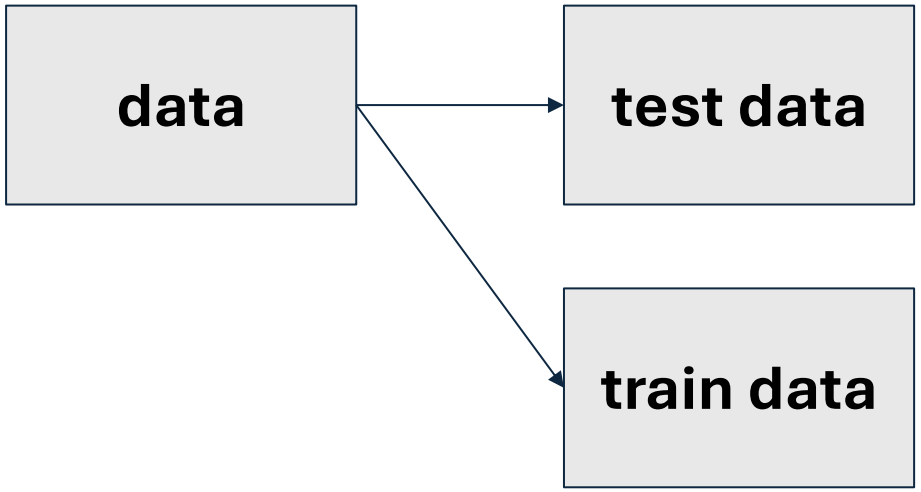
House	Bedrooms	Rooms	Price
A	1	6	500k €
B	3	8	600k €
C	2	7	300k €

$$price = a_1 \cdot rooms + a_2 \cdot size + a_3 \cdot bedrooms + b$$

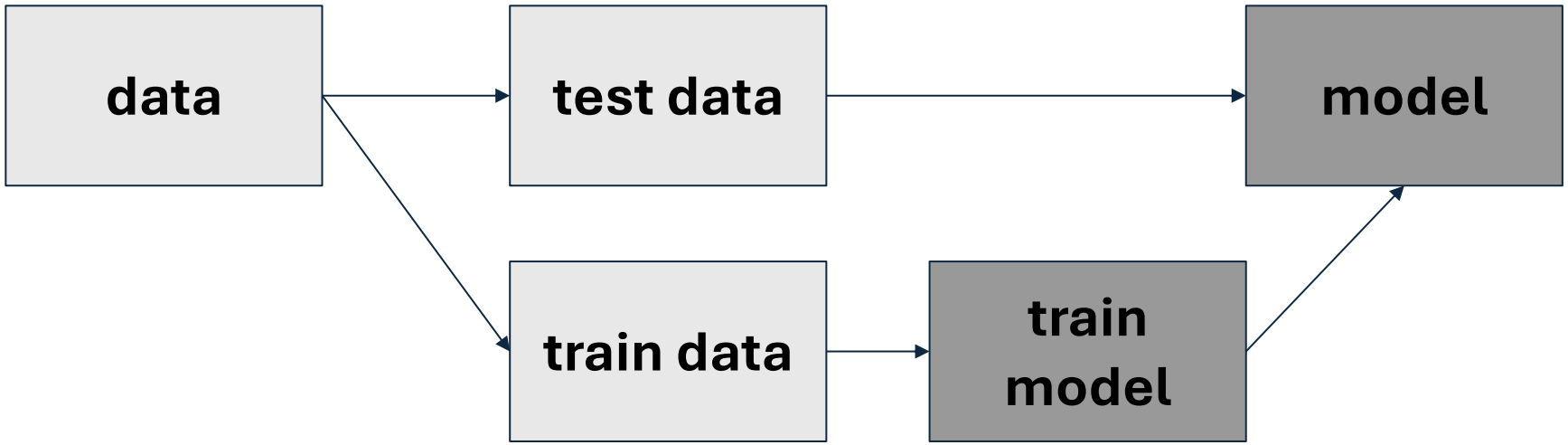
How good is your model?

How does the model perform on unseen data?

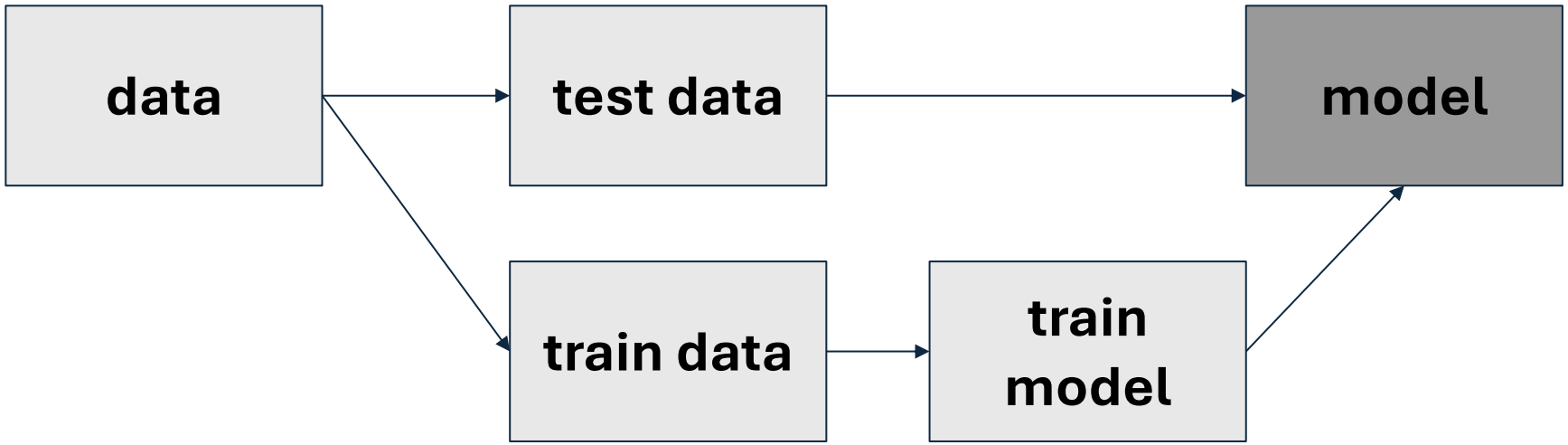
Split data in a train (70%) and test (30%) set to evaluate your model



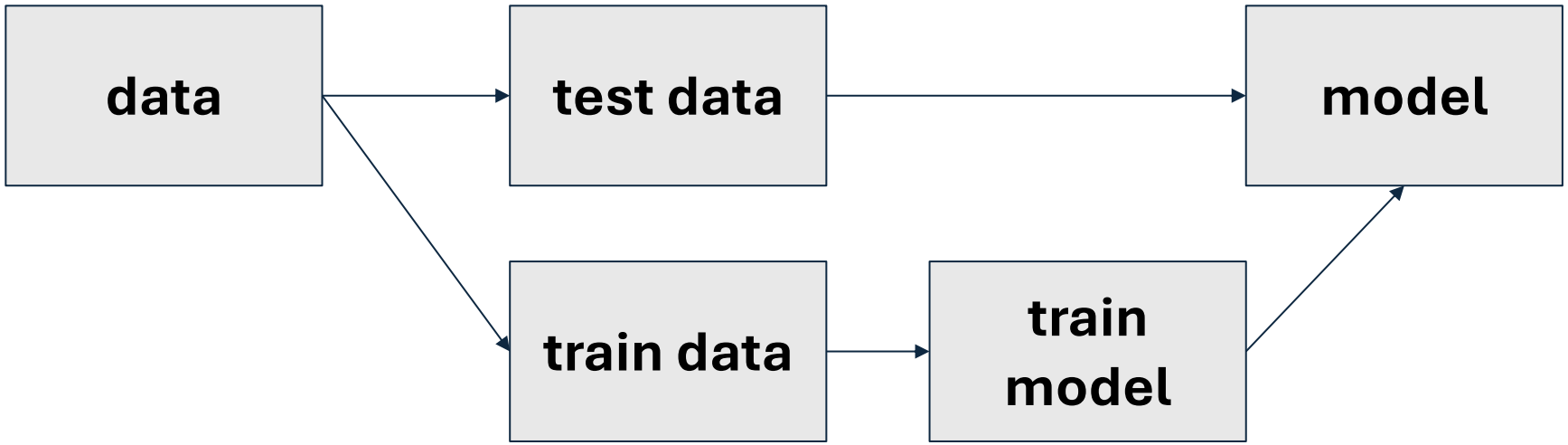
Typical workflow



Typical workflow



Typical workflow



Linear Regression with Python

```
import pandas as pd # data
from sklearn.model_selection import train_test_split # split data
from sklearn.linear_model import LinearRegression # create model
from sklearn.metrics import mean_squared_error # test model
```

Linear Regression with Python

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import root_mean_squared_error

df = pd.read_csv('/content/housing.csv')
X = df[['total_rooms','households']]
y_target = df['median_house_value']
```

Linear Regression with Python

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import root_mean_squared_error

df = pd.read_csv('/content/housing.csv')
X = df[['total_rooms','households']]
y_target = df['median_house_value']

X_train, X_test, y_train, y_test = train_test_split(X, y_target, test_size=0.3)
model = LinearRegression()
model.fit(X_train, y_train)
```

Linear Regression with Python

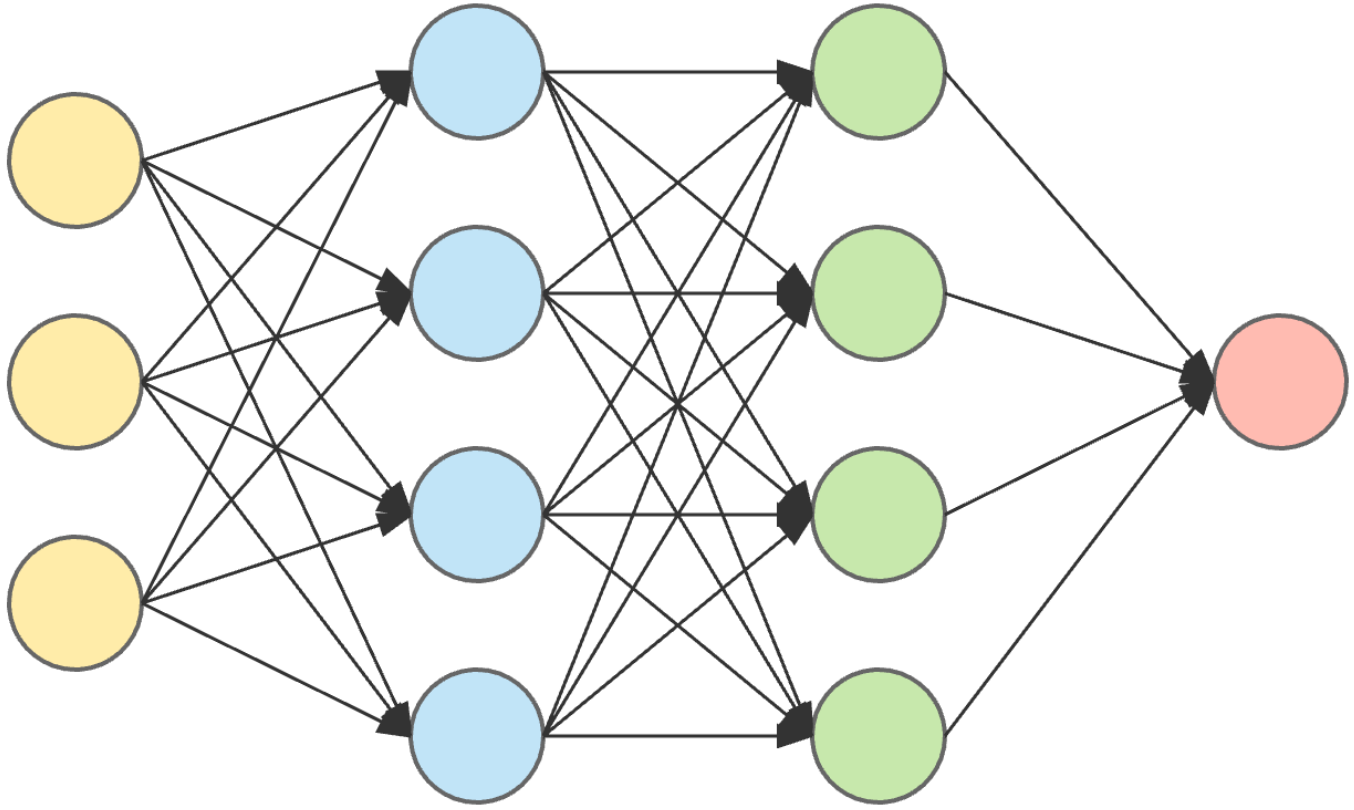
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import root_mean_squared_error

df = pd.read_csv('/content/housing.csv')
X = df[['total_rooms','households']]
y_target = df['median_house_value']

X_train, X_test, y_train, y_test = train_test_split(X, y_target, test_size=0.3)
model = LinearRegression()
model.fit(X_train, y_train)

y_predict = model.predict(X_test)
rmse = root_mean_squared_error(y_test, y_predict)
```

Exercise Time

Linear Regression with Python

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.metrics import median_absolute_error


rmse = mean_squared_error(y_test, y_predict, squared=False)
r2 = r2_score(y_test, y_predict)
mae = median_absolute_error(y_test, y_predict)
```

Deep Learning



input layer hidden layer 1 hidden layer 2 output layer

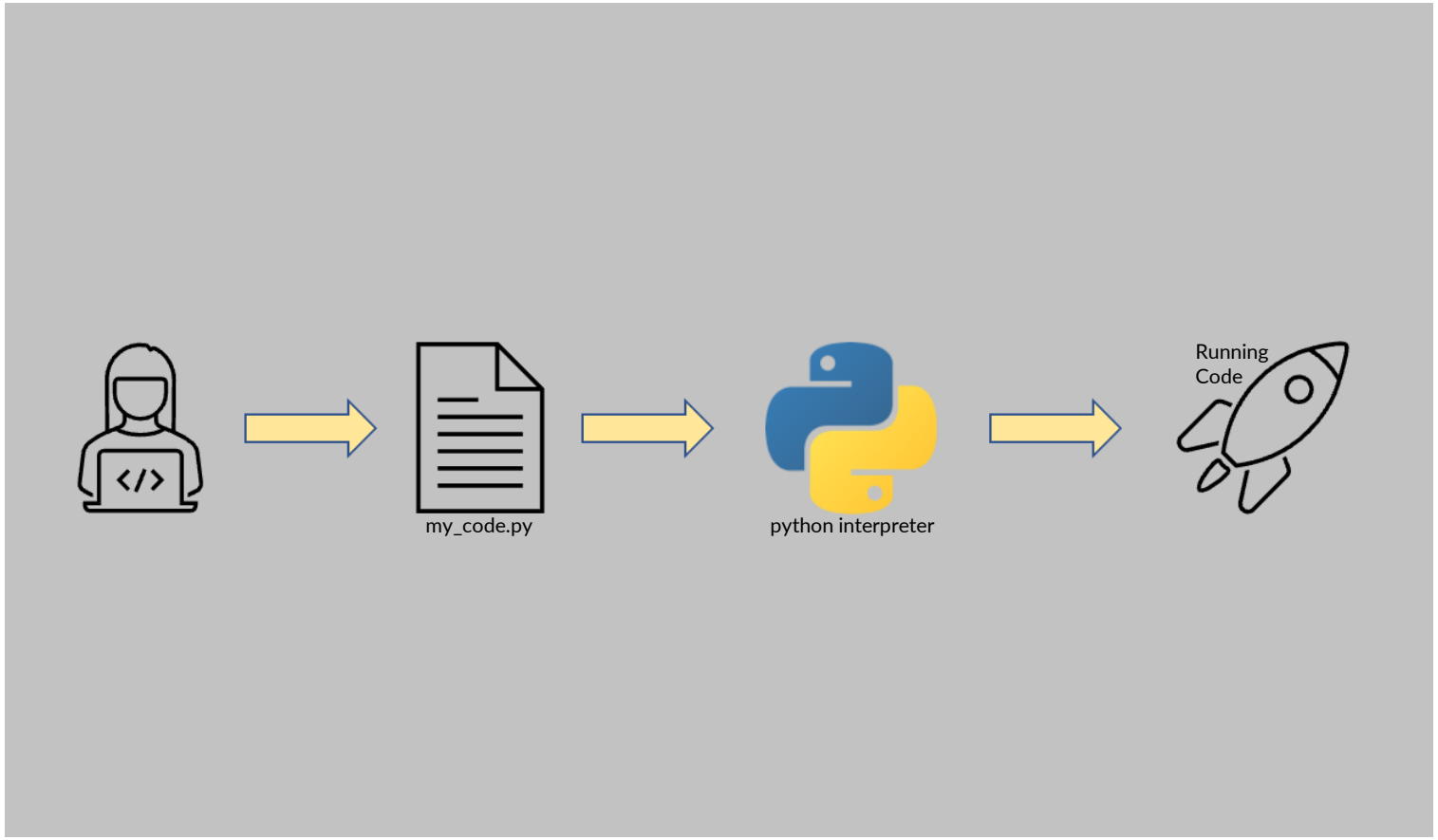
Deep Learning with Tensorflow

```
import tensorflow as tf

X = tf.constant([1,2,3,4,5,6,7])
y = tf.constant([2,3,4,5,6,7,8])
X_new = tf.constant([8])

model = tf.keras.models.Sequential([
  tf.keras.layers.Dense(4, activation="relu", input_shape=[1,]),
  tf.keras.layers.Dense(4, activation="relu"),
  tf.keras.layers.Dense(1 , activation="relu")])
model.compile(loss="mse", optimizer="adam")
model.fit(X,y, epochs=10)
model.predict(X_new)
```

Advanced Visualisations & Modelling

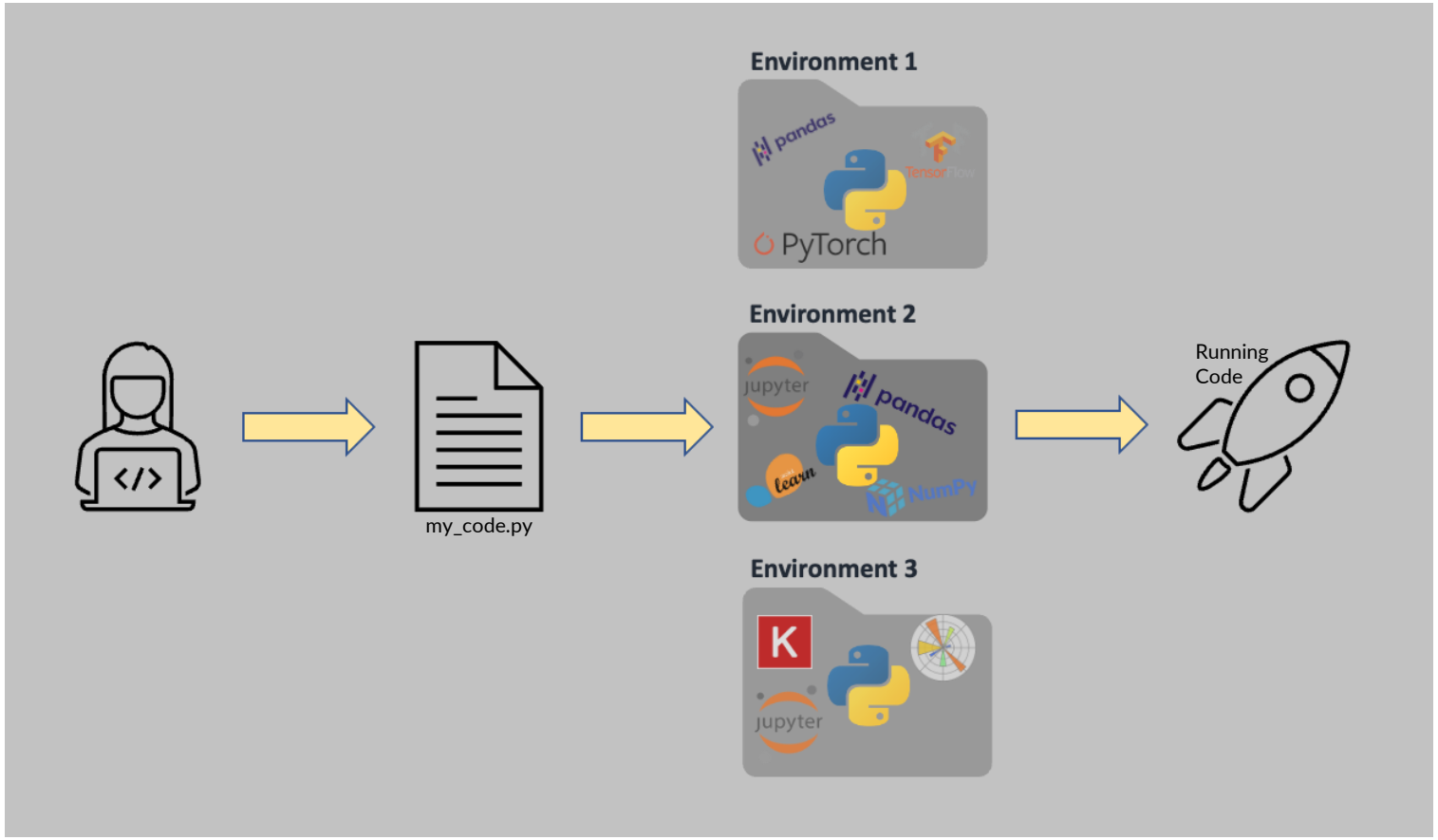
Python Local Installation

Code Execution



https://www.python.org/

Environments



- To install packages for a particular project

- Dependencies are installed in different directories

Anaconda distribution

- Python interpreter
- Jupyter Notebooks (similar to Colab, but local)
- Manage environments and packages
- Runs on Windows, Mac and Linux

Installation:
https://docs.anaconda.com/anaconda/install/

Getting started:
https://docs.anaconda.com/anaconda/user-guide/getting-started/

Pycharm

- Python IDE
- Free Educational License for Pro Version
- Many Alternatives: VS Code, Atom, ...
- AI native Alternatives: Cursur, Windsurf

Installation:
https://www.jetbrains.com/community/education/#students