
Source: datacamp

Introduction to Python

Lecture: Computer Vision

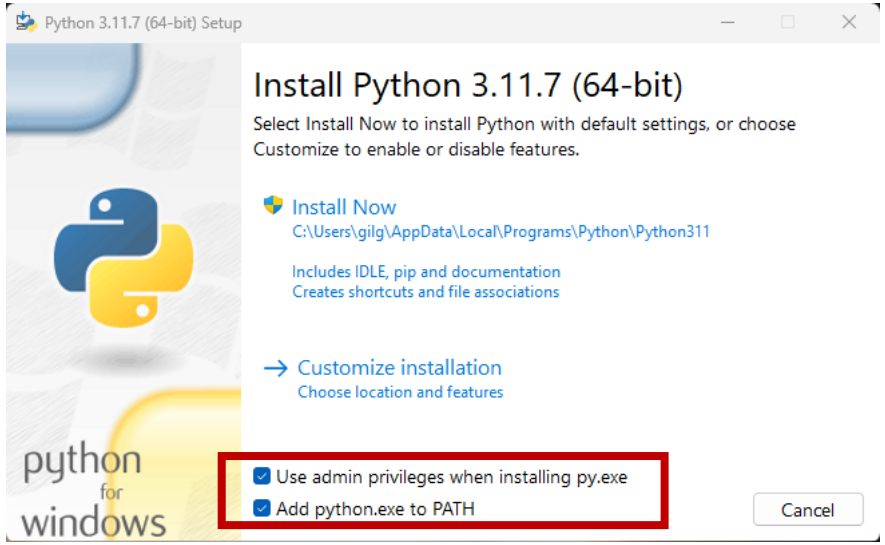
Dr.-Ing. Antje Muntzinger

Outline

1. Python Basics
   - Installation and Environments
   - Jupyter Notebooks
   - Introduction to Python
2. Python Modules
   - NumPy
   - Scikit-Learn
   - Matplotlib

Python-Installation

- Python version **3.11.7** is installed in PC room 2/011, you can use these computers for the assignments.
- Alternatively you can download and install python from here: https://www.python.org/downloads/release/python-3117/
- Further resources:
  - Python tutorial: https://docs.python.org/3.11/tutorial/index.html
  - Python documentation: https://docs.python.org/3.11/index.html



Check both options here!

Python interpreter

- You can try your python installation using the **python interpreter**.
- Start the python interpreter using the command „python“
  - directory with binaries must be added to your system‘s PATH variable

```
C:\Users\muntzinger>python
Python 3.11.7 (tags/v3.11.7:fa7a6f2, Dec  4 2023, 19:24:49) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- You can now enter python commands, e.g.

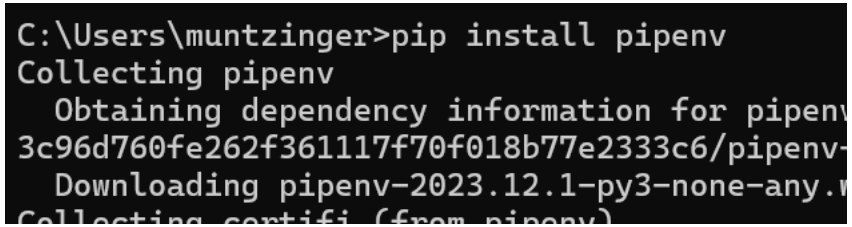
```
>>> print("Hello world!")
Hello world!
```

Virtual Environments

- **Virtual environments** make it possible to set up multiple parallel instances of the python interpreter.
- Each virtual environment can have its own specific packages and configurations, which are sometimes not compatible with the packages needed in other virtual environments for other projects.
- You can even use different python versions.
- Advantages: work on multiple projects in parallel; exchange code more easily; use specific package versions etc.

Installing pipenv

- To install packages, you can use the python package manager **pip.**
- First, we install the package **pipenv** to create a virtual environment as discussed above:



- Afterwards, we can use „pipenv“ instead of „pip“ to install packages

Creating a virtual environment

- Navigate to the folder where you want to store your python files
- Inside this folder, create a folder called „.venv“:

```
C:\Users\muntzinger\Documents\Arbeit_HFT\4_Code\Teaching\CV>mkdir .venv
```

- Initialize the virtual environment with „pipenv install“:

```
C:\Users\muntzinger\Documents\Arbeit_HFT\4_Code\Teaching\CV>pipenv install
Creating a virtualenv for this project...
Pipfile: C:\Users\muntzinger\Documents\Arbeit_HFT\4_Code\Teaching\CV\Pipfile
```

- Activate the environment with „pipenv shell“
  - i.e., packages are installed in this environment while activated, and Python can access the packages while activated only

```
C:\Users\muntzinger\Documents\Arbeit_HFT\4_Code\Teaching\CV>pipenv shell
Launching subshell in virtual environment...
Microsoft Windows [Version 10.0.22631.3296]
(c) Microsoft Corporation. Alle Rechte vorbehalten.

(.venv) C:\Users\muntzinger\Documents\Arbeit_HFT\4_Code\Teaching\CV>
```

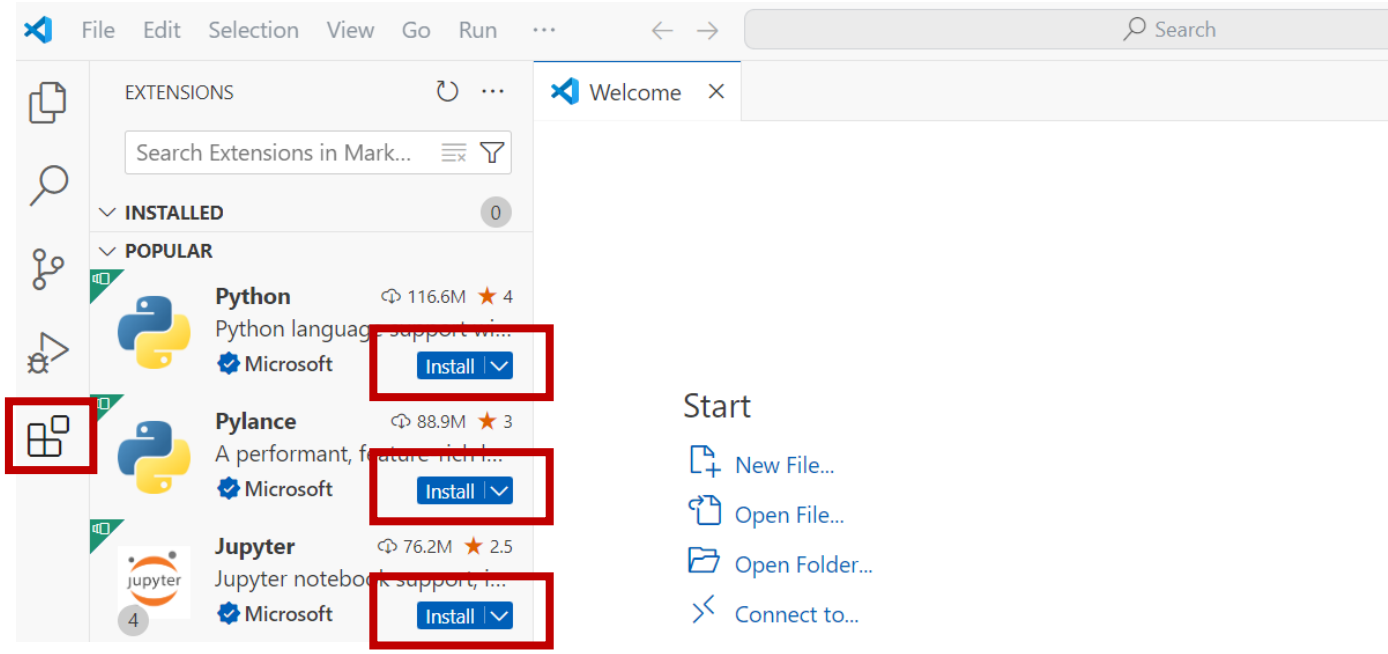
Pipfile

- **Pipfile** shows the installed packages of the virtual environment

- You can also specify packages to be installed in pipfile and then call „pipenv install“ to install them
  - pipfile replaces requirements.txt
- Best practice is to keep pipfile under version control

IDE: VS Code

- Download and install **VS Code** from https://code.visualstudio.com/
  - Alternatively, use another IDE of your choice
- Open VS Code and install **Python, Pylance** and **Jupyter extensions:**



Jupyter Notebook

- Install Jupyter:

```
(.venv) C:\Users\muntzinger\Documents\Arbeit_HFT\4_Code\Teaching\CV>pipenv install jupyter
Installing jupyter...
Resolving jupyter
```

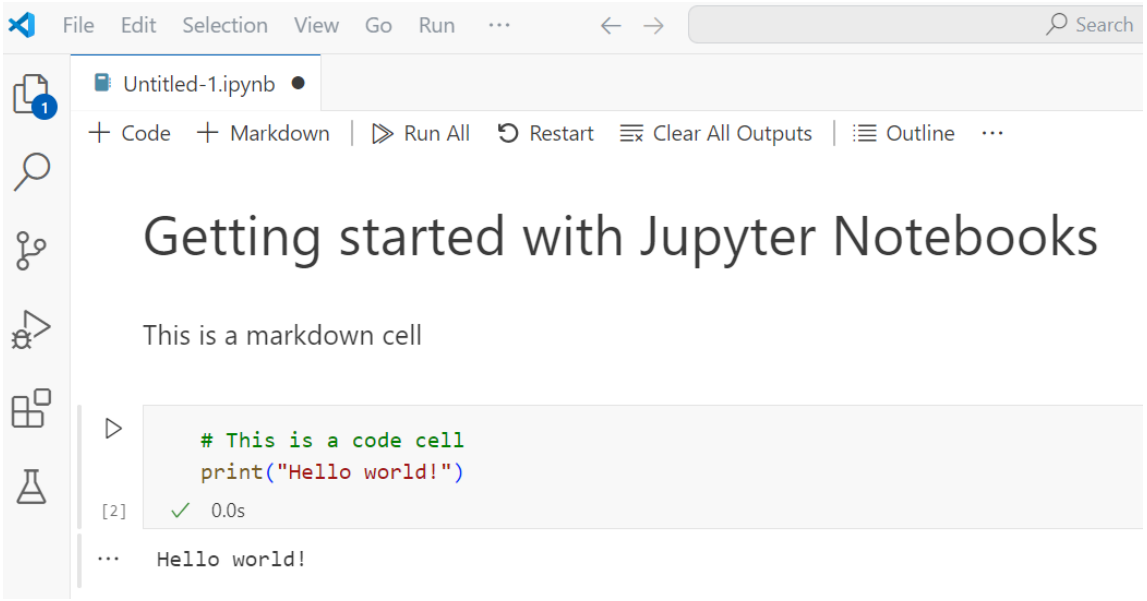
- Potentially install other packages, e.g.

```
(.venv) C:\Users\muntzinger\Documents\Arbeit_HFT\4_Code\Teaching\CV>pipenv install matplotlib numpy pandas scipy scikit-learn
```

- Open VS Code and generate new Jupyter Notebook using file -> new file -> Jupyter Notebook

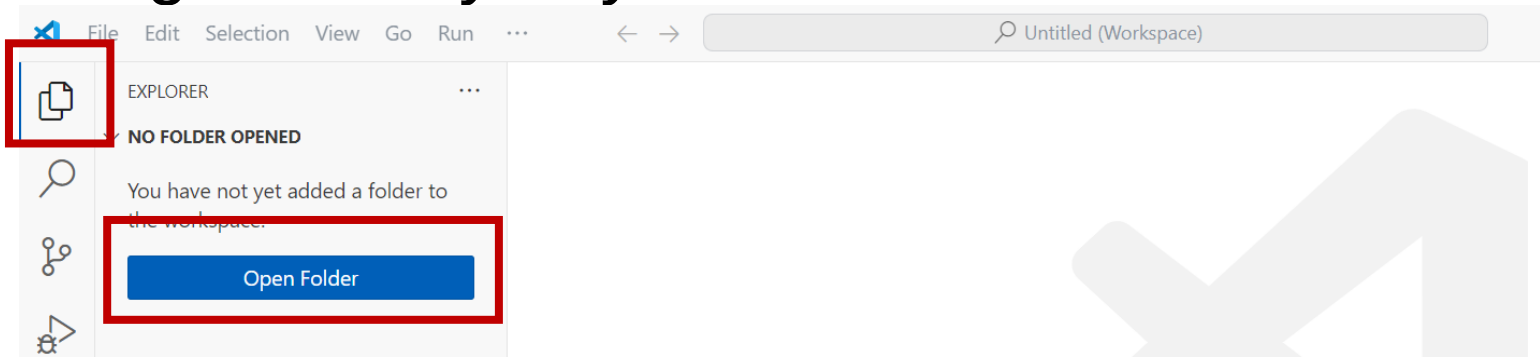
Jupyter Notebook

- A Jupyter notebook contains a list of cells formatted as code or markdown (text). You can select either format via the GUI.
- You can execute a cell using the play-button or shift + enter.
- Further info: https://jupyter-notebook.readthedocs.io/en/latest/

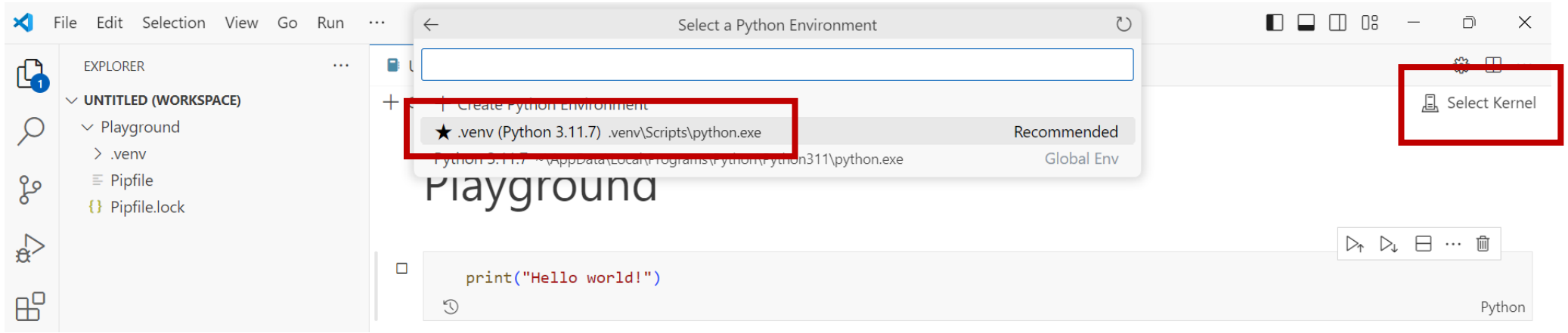


Selecting environment kernel in VS Code

- Open working directory of your virtual environment in VS Code:

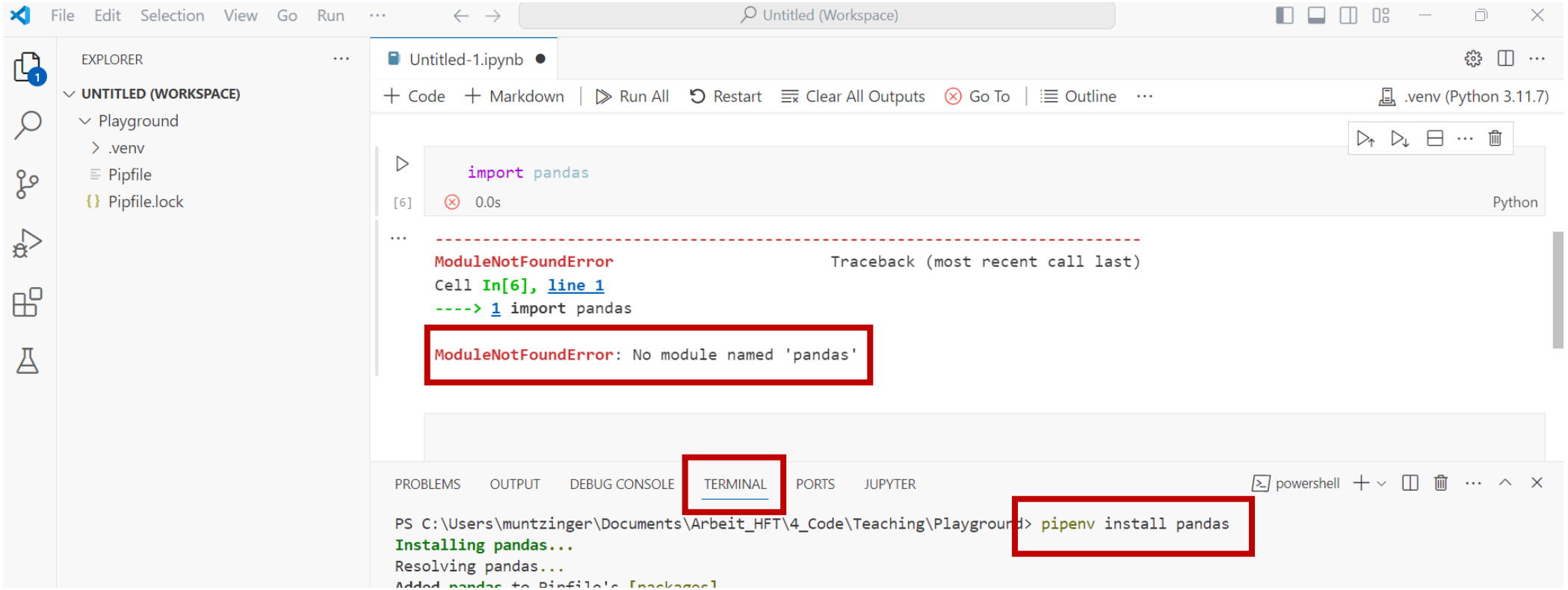


- Select the .venv kernel of your virtual environment:



Installing missing packages

- Missing packages: pipenv install <package>
  - You can use the built-in terminal in VS Code



Export Notebook as pdf (via html)

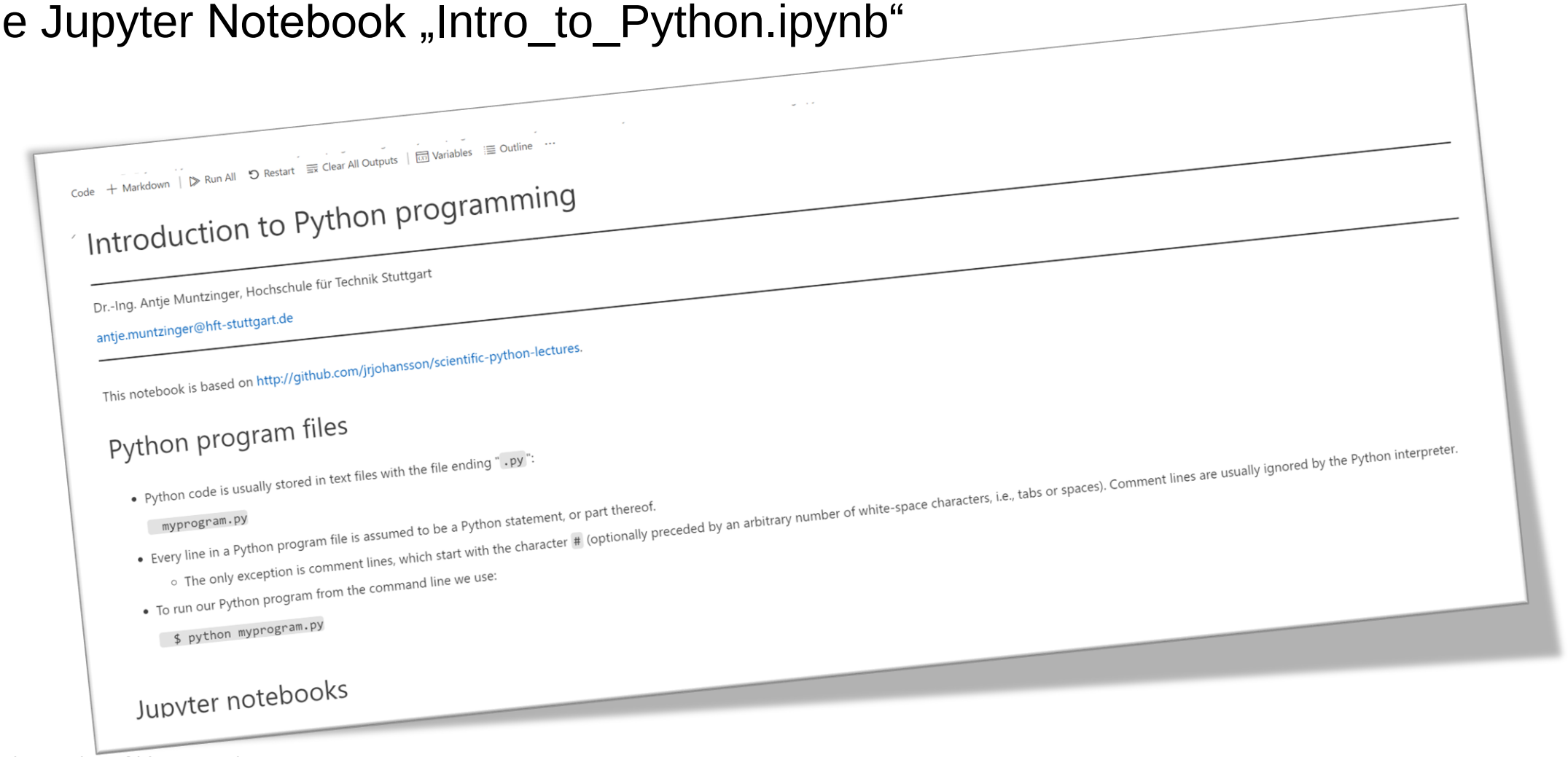
- Run **pipenv install nbconvert[webpdf]** in a terminal to install the nbconvert package
  - updated pipfile:

```
[packages]
matplotlib = "*"
numpy = "*"
pandas = "*"
scipy = "*"
scikit-learn = "*"
jupyter = "*"
nbconvert = {extras = ["webpdf"], version = "*"}
```

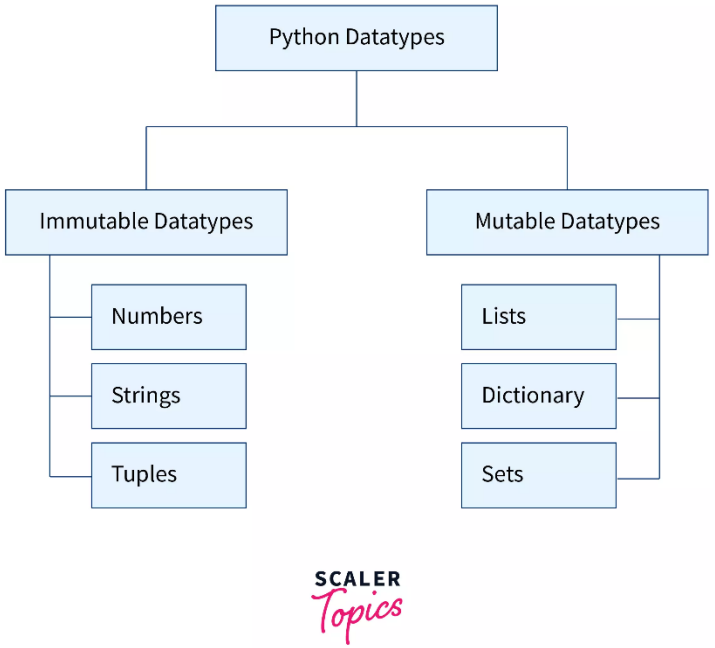
- Run **jupyter nbconvert --to webpdf --allow-chromium-download your-notebook-file.ipynb** in a terminal to create pdf
  - after the first run, you can use the following command instead:
  - **jupyter nbconvert --to webpdf --no-input your-notebook-file.ipynb**

Introduction to Python

- See Jupyter Notebook „Intro_to_Python.ipynb“



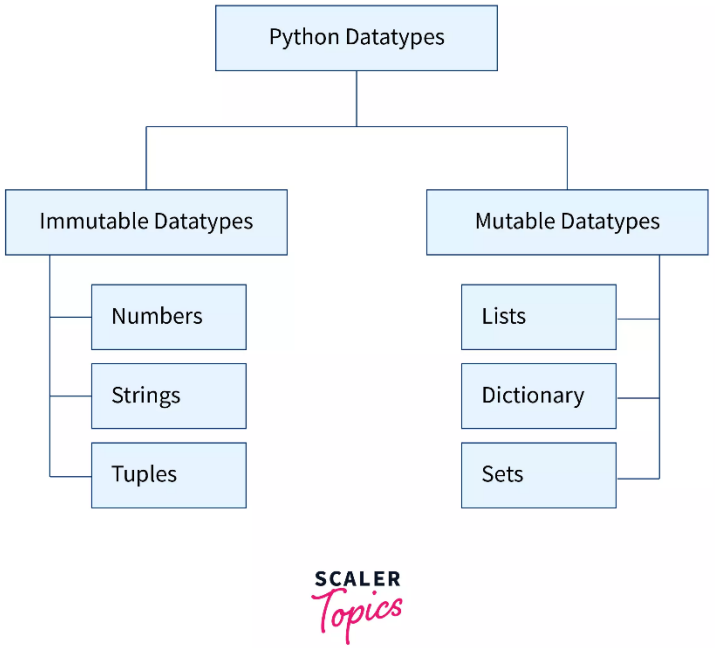
Mutable vs. Immutable Datatypes



- Mutable:
  - values of variables can be changed
  - preferable in case of variable size or content of data
- Immutable:
  - Changing requires overwriting data completely
  - preferable in case of fixed data size or content

Bild: https://www.scaler.com/topics/mutable-data-types-in-python/

Example: list (mutable)



Example 1:

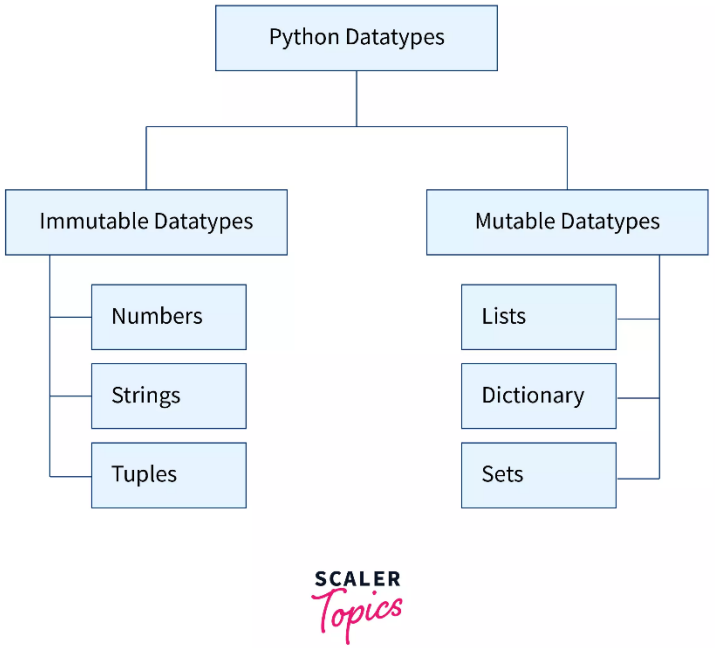
```
# example to demonstrate list is a mutable data type in python

# our current list
my_list = [1,2,3,4,5]

# using append operation in our list
my_list.append(10)
# printing our list after the operation
print("List after appending a value = ",my_list)

# using extend operation in our list
my_list.extend([6,11,23])
# printing our list after the operation
print("List after extending a list = ",my_list)

# after removing a value from our list
my_list.remove(3)
# printing our list after the operation
print("List after removing a value = ",my_list)
```

Output:

```
List after appending a value =  [1, 2, 3, 4, 5, 10]
List after extending a list =  [1, 2, 3, 4, 5, 10, 6, 11, 23]
List after removing a value =  [1, 2, 4, 5, 10, 6, 11, 23]
```


Bild: https://www.scaler.com/topics/mutable-data-types-in-python/

Example: list (mutable)



Example 2:

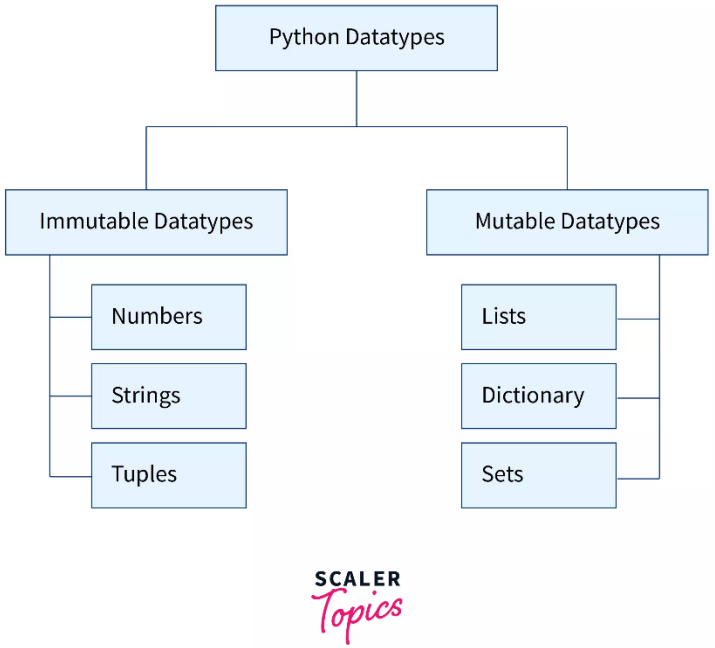
```
# Example to demonstrate we can
# also change the value of the list
# by using the assignment operator
my_list = [1,2,3,4,5]
my_list[4] = 100
print("List after changing value using indexing = ", my_list)
```

Output:

```
List after changing value using indexing =  [1, 2, 3, 4, 100]
```


Bild: https://www.scaler.com/topics/mutable-data-types-in-python/

Example: set (mutable)



Example:

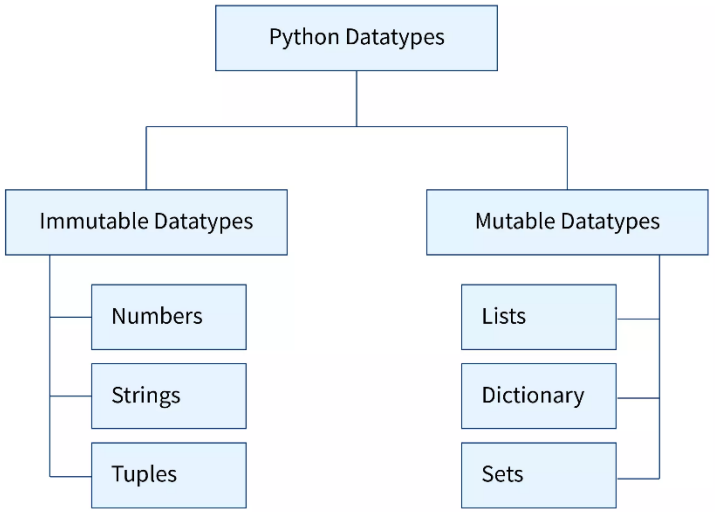
```
# example to demonstrate
# set is a mutable data type in python

# our current set
my_set = {1,2,6,5,7,11}

# adding an element in our set
my_set.add(16)
# printing our set after the operation
print("Set after adding a value : ",my_set)

# adding multiple elements in our set
# multiple elements (such as a list) can be added using update
my_set.update([9,78,100])
# printing our set after the operation
print("Set after updating some values : ",my_set)

# removing element from our set
my_set.remove(2)
# printing our set after the operation
print("Set after removing a value : ",my_set)
```

Output:

```
Set after adding a value :  {1, 2, 5, 6, 7, 11, 16}
Set after updating some values :  {1, 2, 100, 5, 6, 7, 9, 11, 78, 16}
Set after removing a value :  {1, 100, 5, 6, 7, 9, 11, 78, 16}
```


Bild: https://www.scaler.com/topics/mutable-data-types-in-python/

Example: dictionary (mutable)



Example:

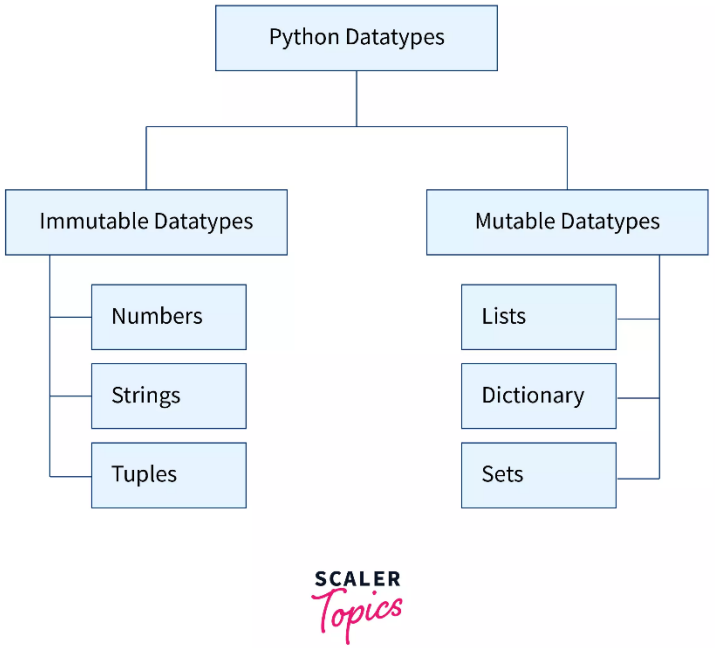
```
# example to demonstrate dictionary is a mutable data type in python

# our current dictionary
my_dict = {"state":"WB", "Capital":"Kolkata"}

# adding new key-value pair to our dictionary
my_dict['Country'] = "India"
# printing our dictionary after the operation
print("Dictionary after adding a new key-value pair = ",my_dict)

# updating key-value pair in our dictionary
my_dict['state'] = "West Bengal"
# printing our dictionary after the operation
print("Dictionary after updating an existing key value pair = ",my_dict)

# removing key-value pair in our dictionary
my_dict.pop('Capital')
# printing our dictionary after the operation
print("Dictionary after popping out a key value pair = ",my_dict)

# removing key-value pair in our dictionary
my_dict.clear()
print("After clearing the whole dictionary = ",my_dict)
```

Output:

```
Our original dictionary =  {'state': 'WB', 'Capital': 'Kolkata'}

Dictionary after adding a new key value pair = {'state': 'WB', 'Capital': 'Kolkata', 'Country':

Dictionary after updating an existing key value pair = {'state': 'West Bengal', 'Capital': 'Kol

Dictionary after popping out a key value pair = {'state': 'West Bengal', 'Country': 'India'}

After clearing the whole dictionary =  {}
```


Bild: https://www.scaler.com/topics/mutable-data-types-in-python/

Example: string (immutable)



- Changing the content creates completely new object:

```
>>> greeting = "Hello!"
>>> id(greeting)
4391270704

>>> greeting = "Hello, World!"
>>> id(greeting)
4391910000
```

- Changing of single items is not possible:

```
>>> greeting[1] = "E"
Traceback (most recent call last):
    ...
TypeError: 'str' object does not support item assignment
```


Quelle: https://realpython.com/python-mutable-vs-immutable-types/#immutable-built-in-data-types-in-python

Careful with mutable datatypes: Reference vs. copy in Python

- Example 1:

```
a = [1, 2, 3]
b = a
b[2] = 11

print("list a:", a)
print("list b:", b)
```

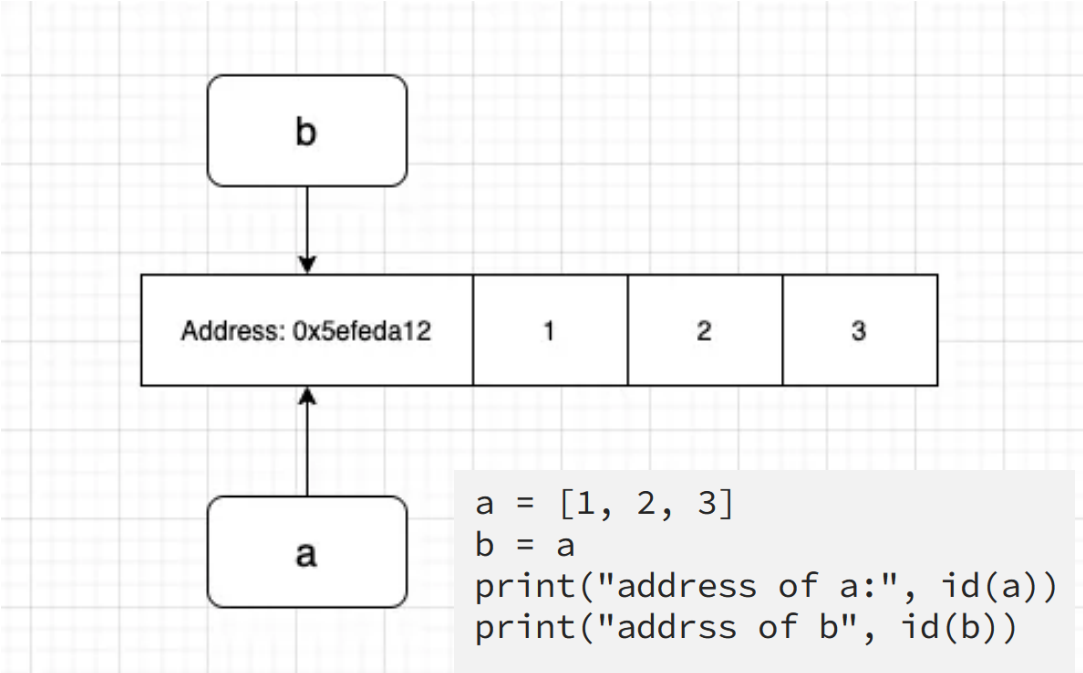
- Example 2:

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

def func(input_list):
    input_list[2] = 11
    return input_list

b = func(a)
print("list a:", a)
print("list b:", b)
```

- Result: Changing variables accidentially through reference

```
list a: [1, 2, 11]
```

- Reason: Both variables point to the same address -> changing one variable also changes the other:




```
a = [1, 2, 3]
b = a
print("address of a:", id(a))
print("addrss of b", id(b))

##############################
address of a: 4385545984
addrss of b 4385545984
```

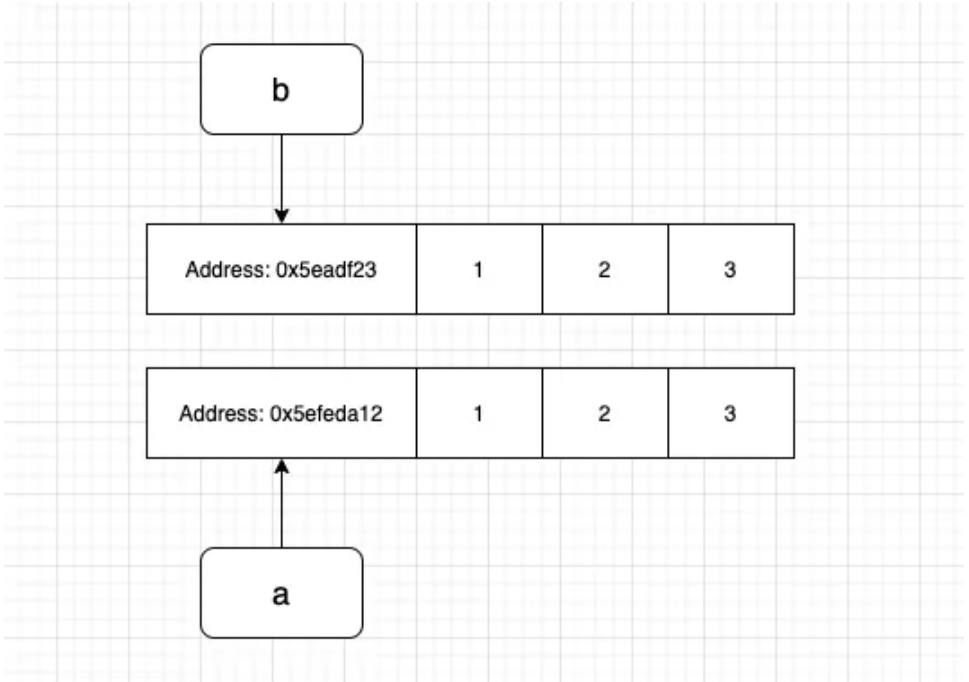
Quelle: https://levelup.gitconnected.com/understanding-reference-and-copy-in-python-c681341a0cd8

Careful with mutable datatypes: Reference vs. copy in Python

- Solution: „deepcopy“

```
import copy

a = [1, 2, [1, 2]]
b = copy.deepcopy(a)
b[2][0] = 11

print("list a:", a)
print("list b:", b)

print("address of a[2]:", id(a[2]))
print("address of b[2]:", id(b[2]))
```

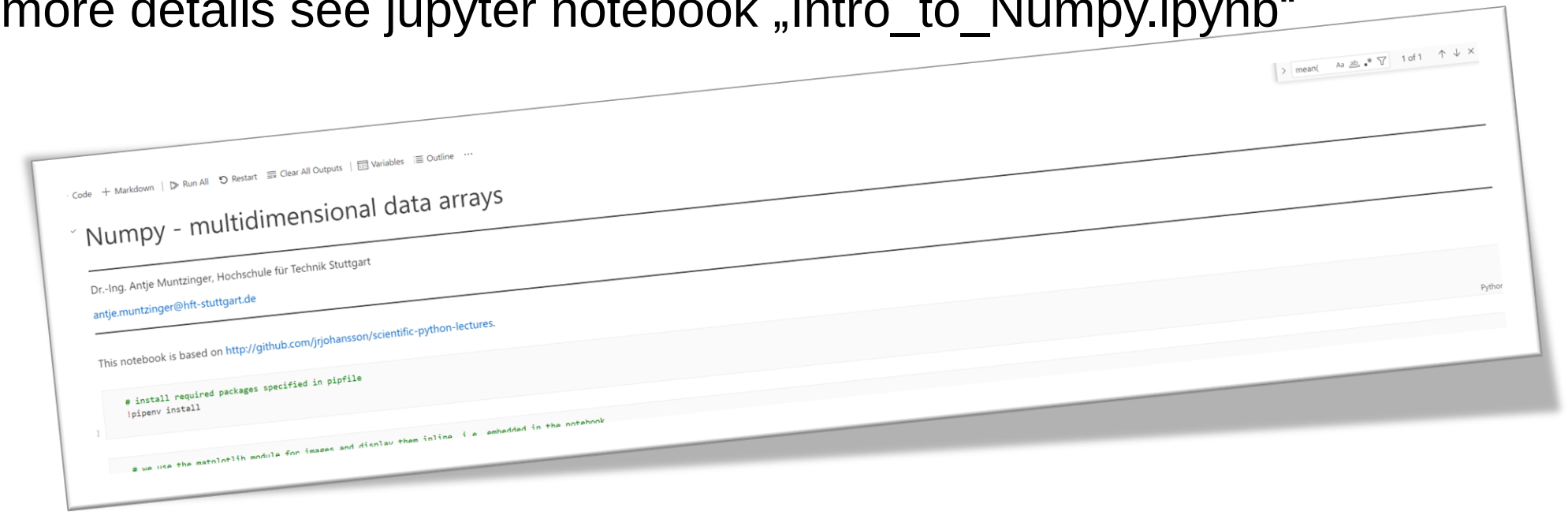
```
list a: [1, 2, [1, 2]]
list b: [1, 2, [11, 2]]

address of a[2]: 4507899200
address of b[2]: 4515136448
```



Quelle: https://levelup.gitconnected.com/understanding-reference-and-copy-in-python-c681341a0cd8

1. NumPy

- NumPy is a package for effective vector and matrix manipulation (called **numpy arrays**), e.g. matrix multiplication, dot product
- Advantage compared to list data type: static typing, i.e. data type is defined during compilation and is consistent within numpy array
- For more details see jupyter notebook „Intro_to_Numpy.ipynb“



2. Scikit-Learn

- Scikit-Learn (sklearn) is a Python library for ML and data science
- Further infos: https://scikit-learn.org/stable/
- Components:
  - **Estimators**: estimate parameters based on data
    - fit() function with data / data+labels as argument
  - **Transformers**: transform data (e.g. imputer: replace NaNs)
    - transform() function
    - fit_transform() function (= fit(), then transform())
  - **Predictors**: estimators that make predictions based on a data set (e.g. LinearRegression: predict happiness based on income)
    - predict() function: predict using new data
    - score() function: measure quality

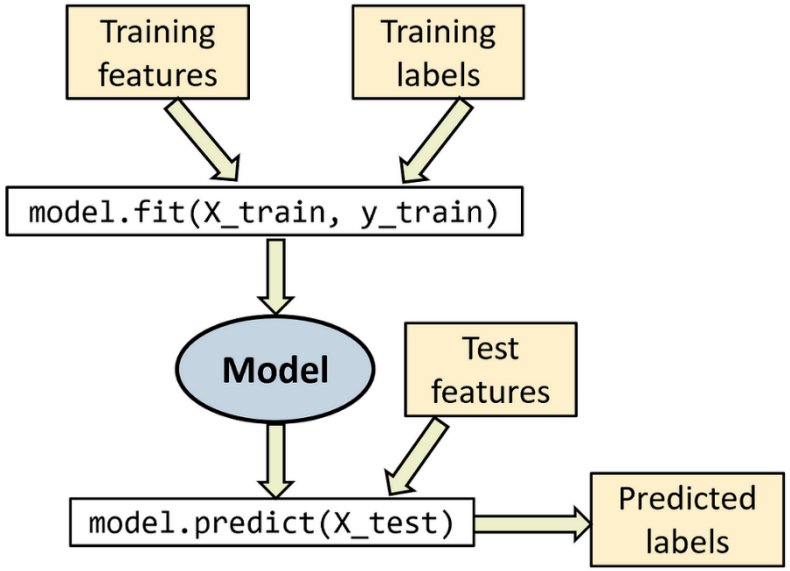
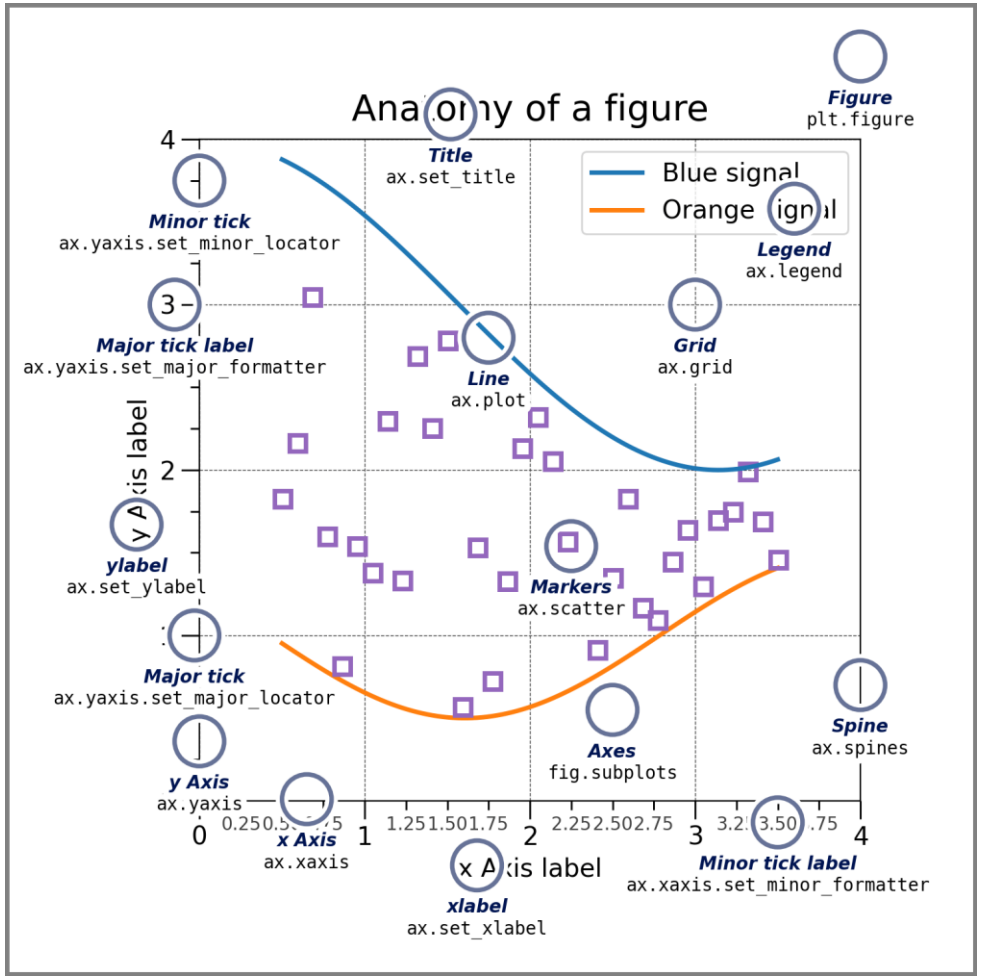


image: https://levelup.gitconnected.com/introduction-to-scikit-learn-1792dc6937e0

3. Matplotlib

- 2D plotting library for static, animated and interactive visualizations in python
- Further information: https://matplotlib.org/stable/users/explain/quick_start.html#



Matplotlib for beginners

Matplotlib is a library for making 2D plots in Python. It is designed with the philosophy that you should be able to create simple plots with just a few commands:

1 Initialize

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

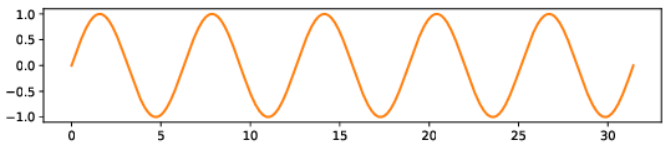
2 Prepare

```
X = np.linspace(0, 10*np.pi, 1000)
Y = np.sin(X)
```

3 Render

```
fig, ax = plt.subplots()
ax.plot(X, Y)
plt.show()
```

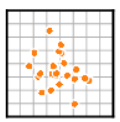
4 Observe



Choose

Matplotlib offers several kind of plots (see Gallery):

```
X = np.random.uniform(0, 1, 100)
Y = np.random.uniform(0, 1, 100)
ax.scatter(X, Y)
```



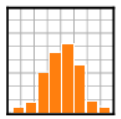
```
X = np.arange(10)
Y = np.random.uniform(1, 10, 10)
ax.bar(X, Y)
```

```
Z = np.random.uniform(0, 1, (8, 8))

ax.imshow(Z)
```

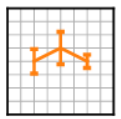
```
Z = np.random.uniform(0, 1, (8, 8))

ax.contourf(Z)
```

```
Z = np.random.uniform(0, 1, 4)

ax.pie(Z)
```

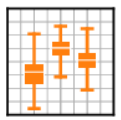
```
Z = np.random.normal(0, 1, 100)

ax.hist(Z)
```



```
X = np.arange(5)
Y = np.random.uniform(0, 1, 5)
ax.errorbar(X, Y, Y/4)
```



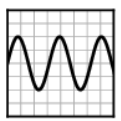
```
Z = np.random.normal(0, 1, (100, 3))

ax.boxplot(Z)
```



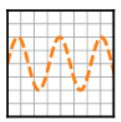
Tweak

You can modify pretty much anything in a plot, including limits, colors, markers, line width and styles, ticks and ticks labels, titles, etc.

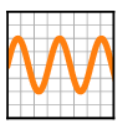
```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, color="black")
```



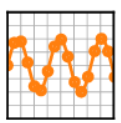
```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, linestyle="--")
```



```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, linewidth=5)
```



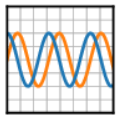
```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, marker="o")
```



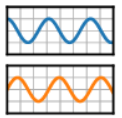
Organize

You can plot several data on the same figure, but you can also split a figure in several subplots (named *Axes*):

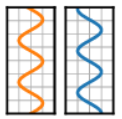
```
X = np.linspace(0, 10, 100)
Y1, Y2 = np.sin(X), np.cos(X)
ax.plot(X, Y1, X, Y2)
```



```
fig, (ax1, ax2) = plt.subplots(2, 1)
ax1.plot(X, Y1, color="C1")
ax2.plot(X, Y2, color="C0")
```

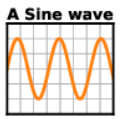


```
fig, (ax1, ax2) = plt.subplots(1, 2)
ax1.plot(Y1, X, color="C1")
ax2.plot(Y2, X, color="C0")
```

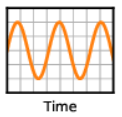


Label (everything)

```
ax.plot(X, Y)
fig.suptitle(None)
ax.set_title("A Sine wave")
```




```
ax.plot(X, Y)
ax.set_ylabel(None)
ax.set_xlabel("Time")
```



Explore

Figures are shown with a graphical user interface that allows to zoom and pan the figure, to navigate between the different views and to show the value under the mouse.

Save (bitmap or vector format)

```
fig.savefig("my-first-figure.png", dpi=300)
fig.savefig("my-first-figure.pdf")
```