Politecnico
di Torino

1859

Data Science Lab

Introduction to Python

Andrea Pasini
Flavio Giobergia

DataBase and Data Mining Group

Summary

- **Python engine**
  - Basic components and setup
- **Python language**
  - Data types, object oriented programming
- **NumPy library**
  - Computation with multi-dimensional arrays
- **Pandas library**
  - Tabular data and data preprocessing
- **scikit-learn library**
  - Machine learning and data science tools

- Python language
  - Clean and concise syntax
    - No semi-colons to end instructions
    - No braces to define if clauses and for loops
    - No need to specify variable types
    - …

Java

```
List<String> l = new LinkedList<>();
for (int i=0; i<10; i++) {
    l.add(i);
}
```

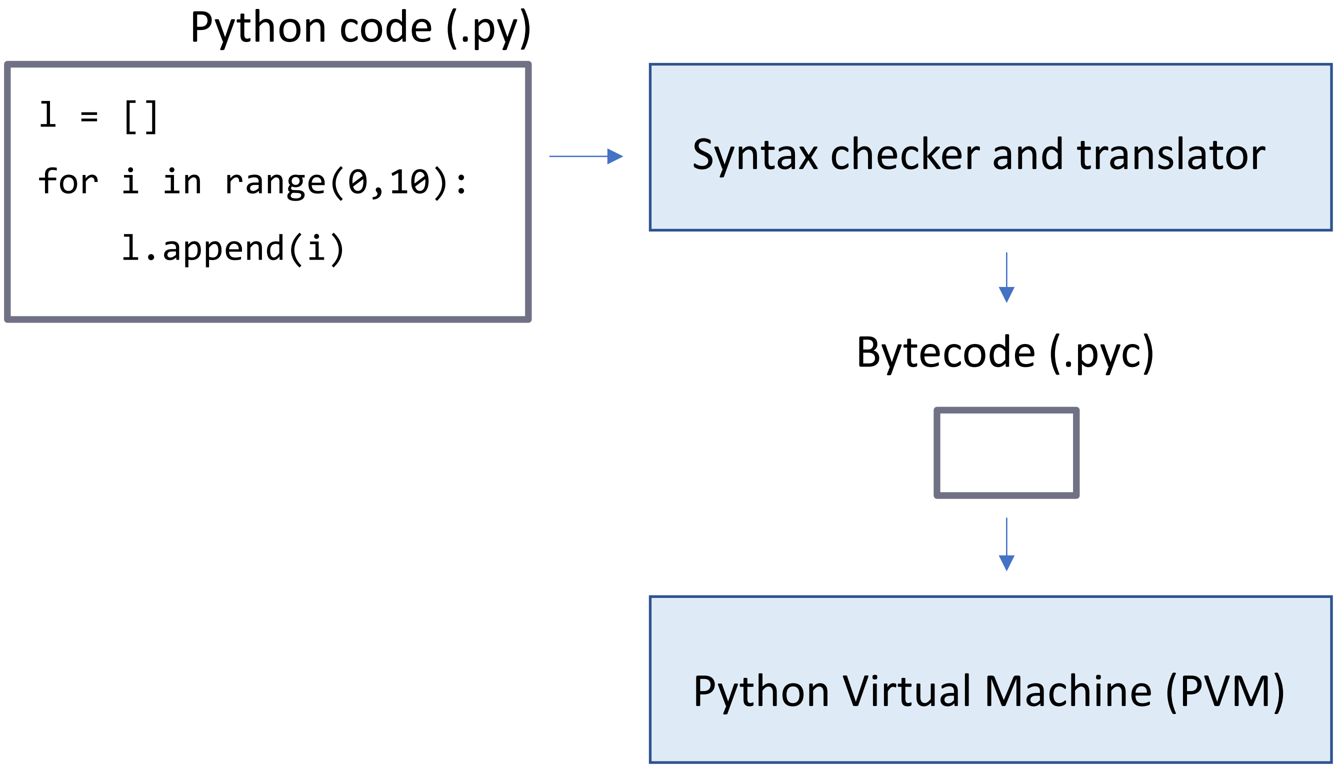
Python

```
l = []
for i in range(0,10):
    l.append(i)
```

Introduction to Python

- Python is an **interpreted** language
  - Code is not compiled to machine language
  - However the source code is compiled to an intermediate level, called **bytecode**
  - For this reason, to run Python programs, you need an **interpreter** that is able to execute the bytecode

- Sequence of operations executed by the interpreter



Python code (.py)

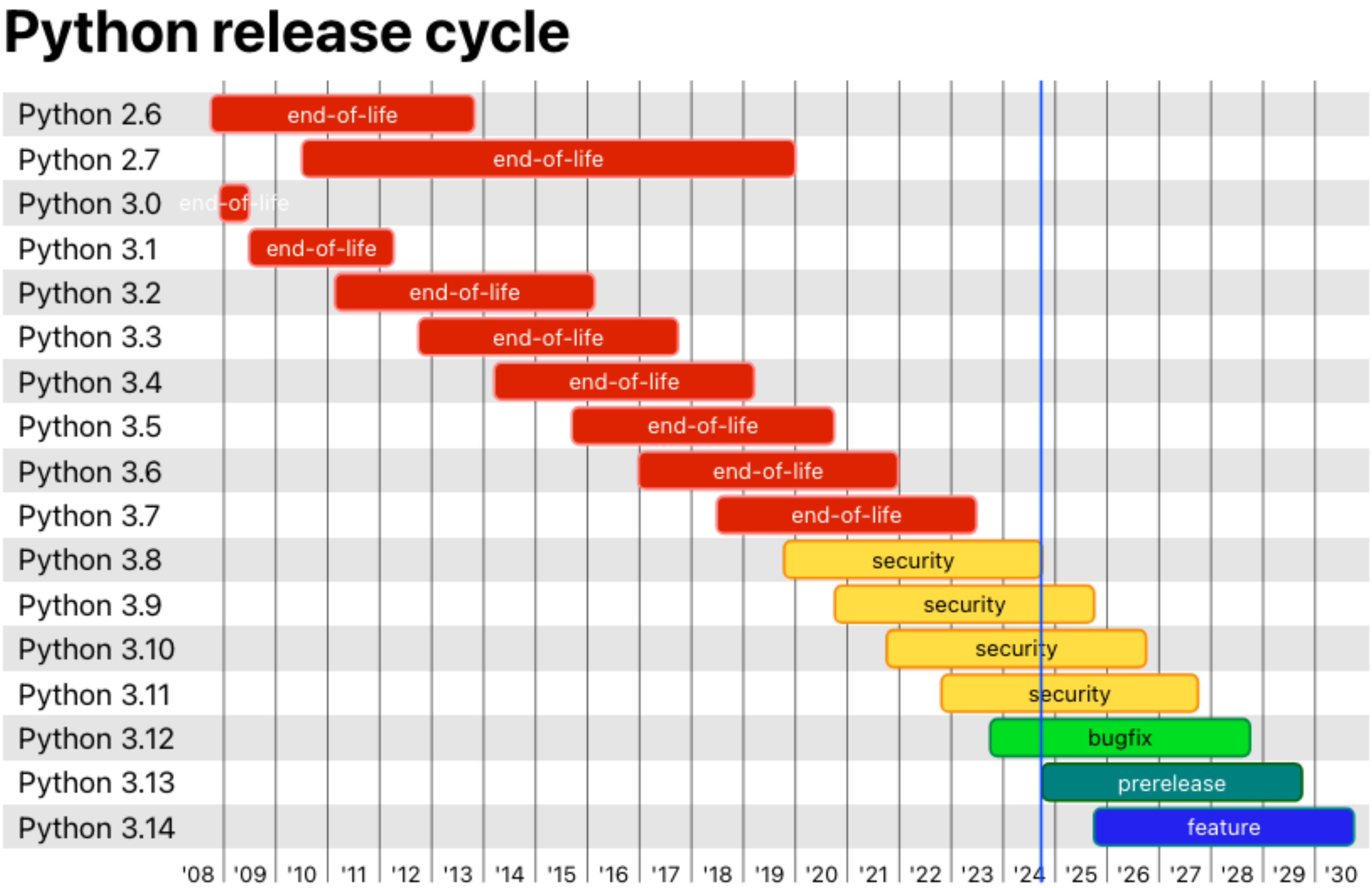
```
l = []
for i in range(0,10):
    l.append(i)
```

Syntax checker and translator

Bytecode (.pyc)

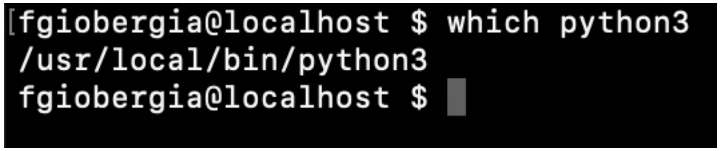
Python Virtual Machine (PVM)

Python 3



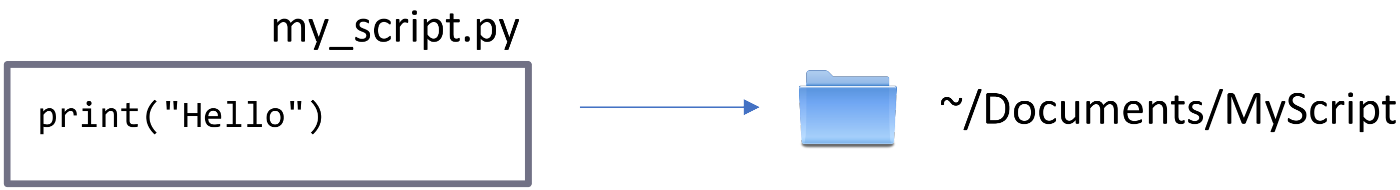
https://devguide.python.org/versions/

- A common Python 3 setup on a **Linux** System
- Typically in the /usr/bin folder:
  - “**python**” executable: run Python programs
  - “**pip**” executable: install Python packages
  - “**ipython**” executable: run programs line by line
  - “**jupyter**” executable: run a jupyter notebook
  - **“<name>3**” if your system defaults to Python 2
    - (hopefully it does not)
- To find where your python commands live:
  - `which <command>`



```
[fgiobergia@localhost $ which python3
/usr/local/bin/python3
fgiobergia@localhost $ █
```

- Executing a Python program



my_script.py

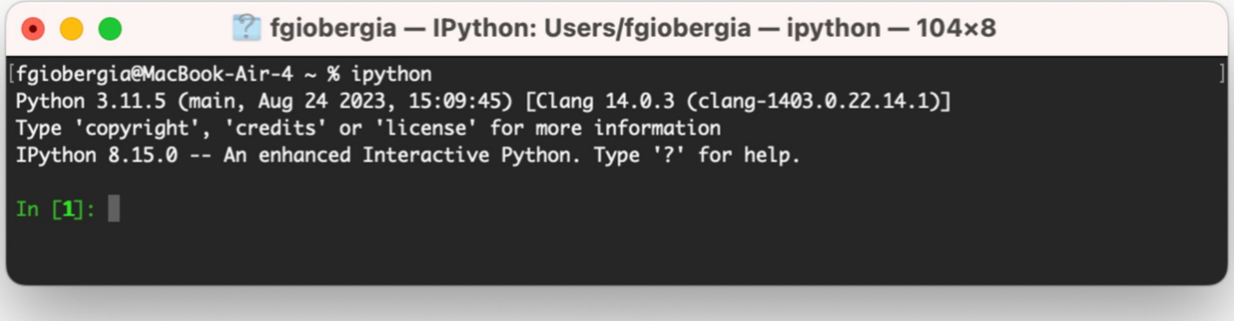
```
print("Hello")
```

~/Documents/MyScript

- Type in your terminal:

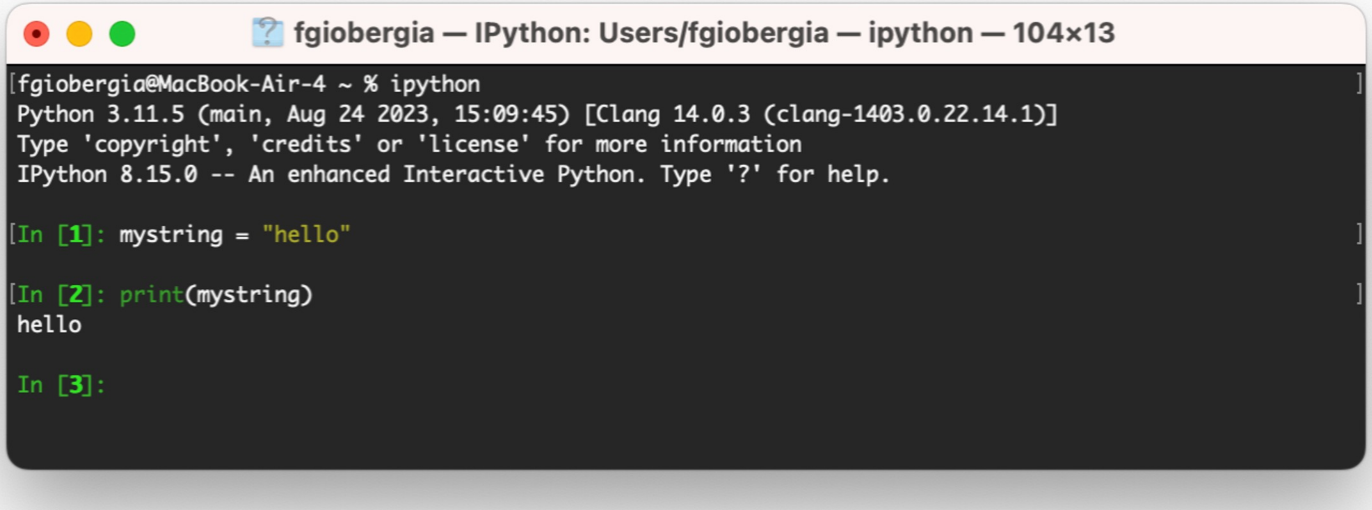
```
cd ~/Documents/MyScript
python my_script.py
```

- Running Python line by line with IPython
- Type in your terminal:
  - `ipython` (or `ipython3`, depending on your installation)



```
fgiobergia@MacBook-Air-4 ~ % ipython
Python 3.11.5 (main, Aug 24 2023, 15:09:45) [Clang 14.0.3 (clang-1403.0.22.14.1)]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.15.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
```

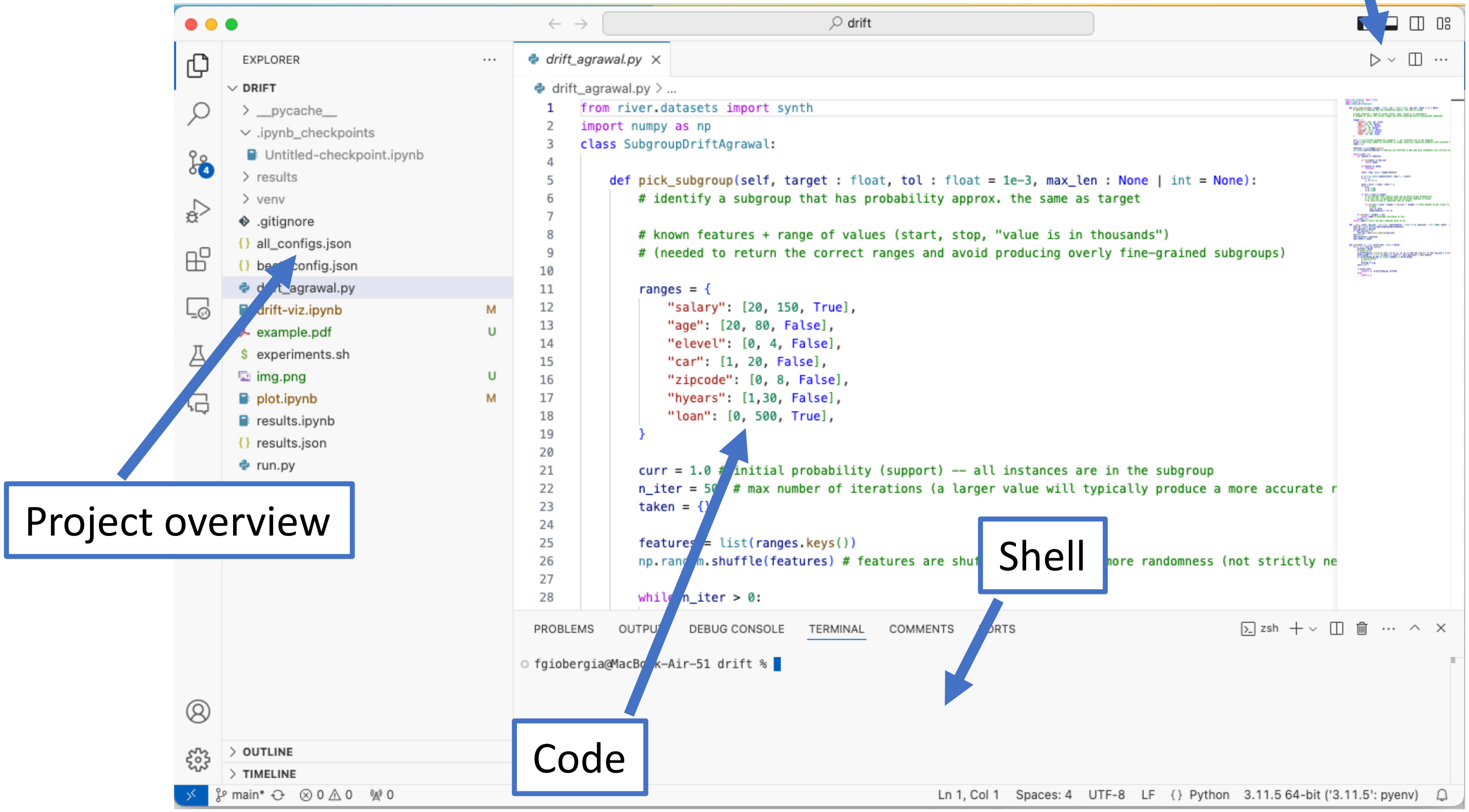
- Write your program line by line to see the results step by step…



```
fgiobergia@MacBook-Air-4 ~ % ipython
Python 3.11.5 (main, Aug 24 2023, 15:09:45) [Clang 14.0.3 (clang-1403.0.22.14.1)]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.15.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: mystring = "hello"

In [2]: print(mystring)
hello

In [3]:
```

- **Python** and **IPython** programs are the core for executing scripts, but…
- There are two typical scenarios:
  1. Develop your Python **project** with an Integrated Development Environment (**IDE**)
     - Example: Visual Studio Code, PyCharm
     - **Debug** and **run** your code inside the IDE
  2. Develop and test a Python **script** with **Jupyter notebook**
     - Inspect **step by step** the results
     - Keep the history of the output of the script

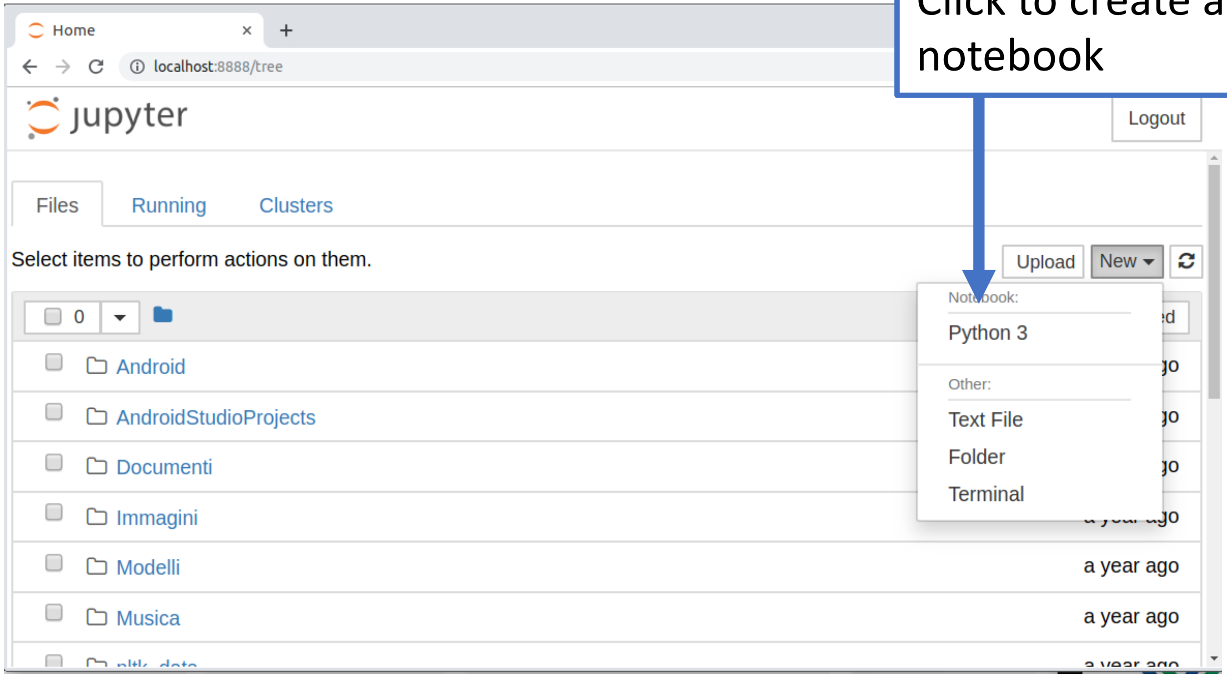
- **Scenario 1: Visual Studio Code (IDE)**
  - The Python suite can be installed from the *Extensions* tab



```python
from river.datasets import synth
import numpy as np
class SubgroupDriftAgrawal:

    def pick_subgroup(self, target : float, tol : float = 1e-3, max_len : None | int = None):
        # identify a subgroup that has probability approx. the same as target

        # known features + range of values (start, stop, "value is in thousands")
        # (needed to return the correct ranges and avoid producing overly fine-grained subgroups)

        ranges = {
            "salary": [20, 150, True],
            "age": [20, 80, False],
            "elevel": [0, 4, False],
            "car": [1, 20, False],
            "zipcode": [0, 8, False],
            "hyears": [1,30, False],
            "loan": [0, 500, True],
        }

        curr = 1.0 # initial probability (support) -- all instances are in the subgroup
        n_iter = 50 # max number of iterations (a larger value will typically produce a more accurate r
        taken = {}

        features = list(ranges.keys())
        np.random.shuffle(features) # features are shuf[illegible]more randomness (not strictly ne

        while n_iter > 0:
```

```
fgiobergia@MacBook-Air-51 drift % 
```

- **Scenario 2: Jupyter notebook**
  - Type in your terminal
    - jupyter notebook
  - Jupyter will open on your browser

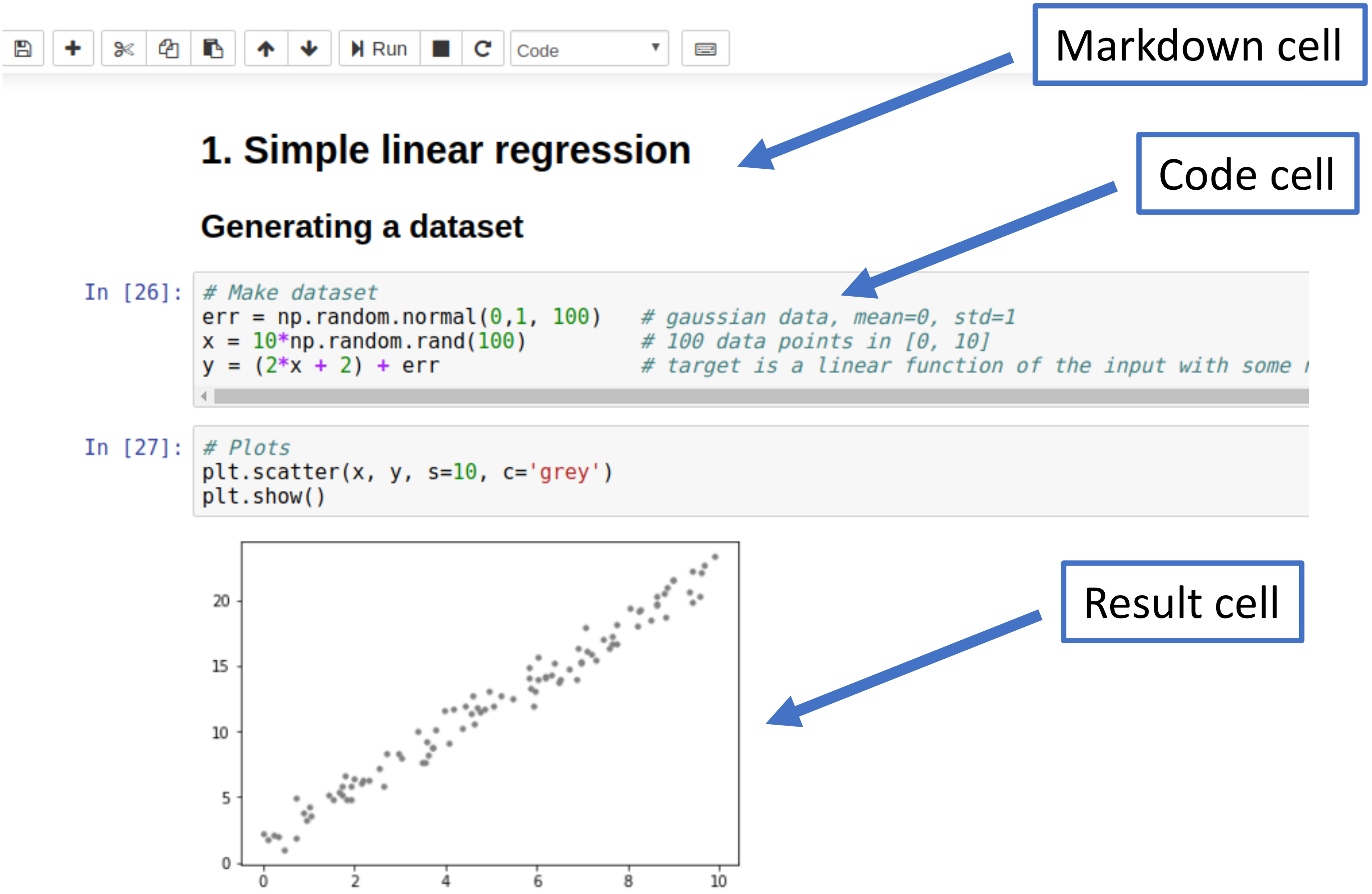


Click to create a new notebook

Select items to perform actions on them.

Android	
AndroidStudioProjects	
Documenti	
Immagini	a year ago
Modelli	a year ago
Musica	a year ago
nltk_data	a year ago

Scenario 2: Jupyter notebook



1. Simple linear regression

Generating a dataset

In [26]:

```
# Make dataset
err = np.random.normal(0,1, 100)   # gaussian data, mean=0, std=1
x = 10*np.random.rand(100)         # 100 data points in [0, 10]
y = (2*x + 2) + err                # target is a linear function of the input with some
```

In [27]:

```
# Plots
plt.scatter(x, y, s=10, c='grey')
plt.show()
```

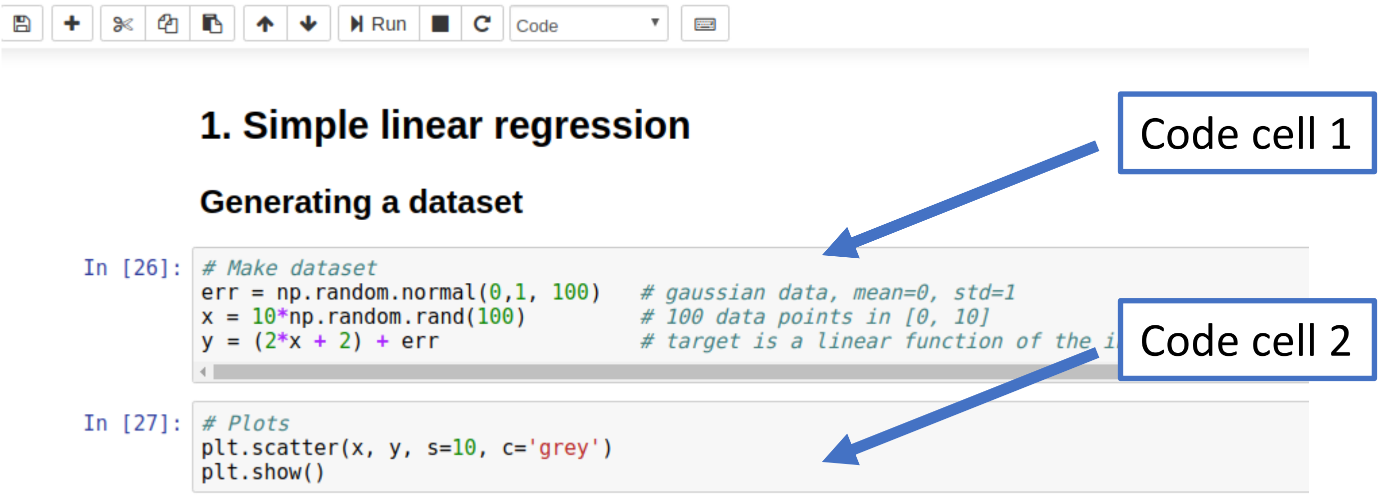
Markdown cell

Code cell

Result cell

Scenario 2: Jupyter notebook

- Based on **IPython** command
- Each code **cell** can be executed **separately** by pressing CTRL + ENTER



1. Simple linear regression

Generating a dataset

```
In [26]: # Make dataset
         err = np.random.normal(0,1, 100)   # gaussian data, mean=0, std=1
         x = 10*np.random.rand(100)         # 100 data points in [0, 10]
         y = (2*x + 2) + err                # target is a linear function of the i
```

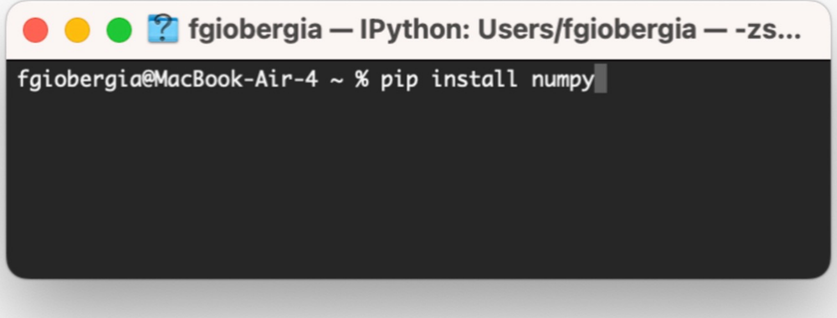
```
In [27]: # Plots
         plt.scatter(x, y, s=10, c='grey')
         plt.show()
```

Code cell 1

Code cell 2

- **IDE vs Jupyter notebook**
  - IDE
    - For **complex** projects (many files)
    - More powerful debug commands
    - More powerful code editing tools
  - Jupyter notebook
    - For simple scripts and prototypes
    - Great **visualization** tool
      - Example: **report** with Python code and text for explanations

- **Installing libraries**
  - Python language is provided with many useful libraries:
    - Numpy, Pandas, Matplotlib, Scikit-learn, SciPy, …
  - To use any of them you first have to install it with the **pip** command: `pip install <package>`
    - `pip install numpy`
    - `pip install pandas`

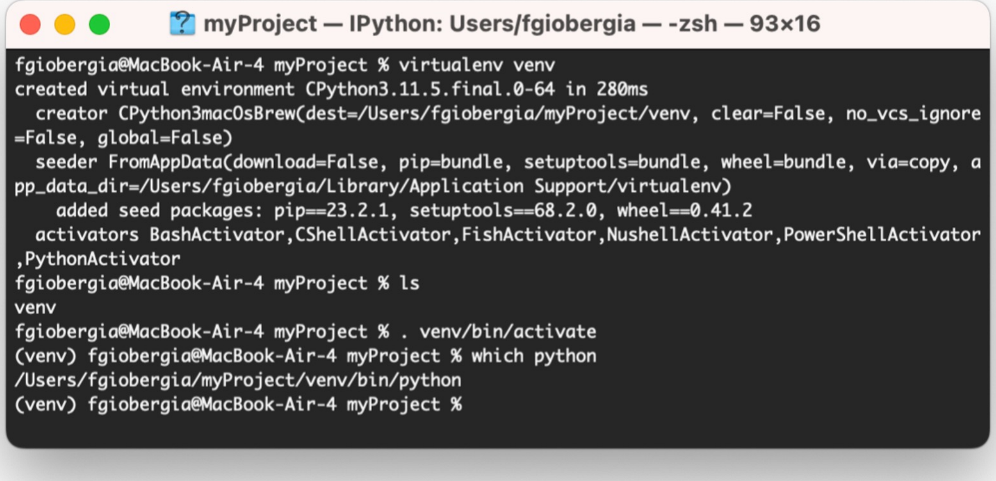


```
fgiobergia@MacBook-Air-4 ~ % pip install numpy
```

- **Virtual environments**
  - The pip command will associate the libraries to your **default Python installation**
  - A more powerful way of managing libraries is to use a Python **environment (virtualenv)**
    - Designed when you want to design **different projects** that use different libraries and **configurations (e.g. versions)**
      - Each projects is associated to a virtual environment

Virtual environments

- To create and use a new environment:
  - `cd ~/myProject` ➔ move to project directory
  - `virtualenv venv` ➔ *create* virtual environment called venv
  - `. venv/bin/activate` ➔ *activate* environment “venv”
- Python & libraries used will be from venv (not global)



```
fgiobergia@MacBook-Air-4 myProject % virtualenv venv
created virtual environment CPython3.11.5.final.0-64 in 280ms
  creator CPython3macOsBrew(dest=/Users/fgiobergia/myProject/venv, clear=False, no_vcs_ignore
=False, global=False)
  seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle, via=copy, a
pp_data_dir=/Users/fgiobergia/Library/Application Support/virtualenv)
    added seed packages: pip==23.2.1, setuptools==68.2.0, wheel==0.41.2
  activators BashActivator,CShellActivator,FishActivator,NushellActivator,PowerShellActivator
,PythonActivator
fgiobergia@MacBook-Air-4 myProject % ls
venv
fgiobergia@MacBook-Air-4 myProject % . venv/bin/activate
(venv) fgiobergia@MacBook-Air-4 myProject % which python
/Users/fgiobergia/myProject/venv/bin/python
(venv) fgiobergia@MacBook-Air-4 myProject %
```

- **Virtual environments**
  - After activation you can use the terminal to work within the environment
  - Install libraries in the *current* environment
    - `pip install my_library`
  - Execute a script/notebook within the environment
    - `python my_script.py`
    - `jupyter notebook`
  - To deactivate the environment
    - `deactivate`