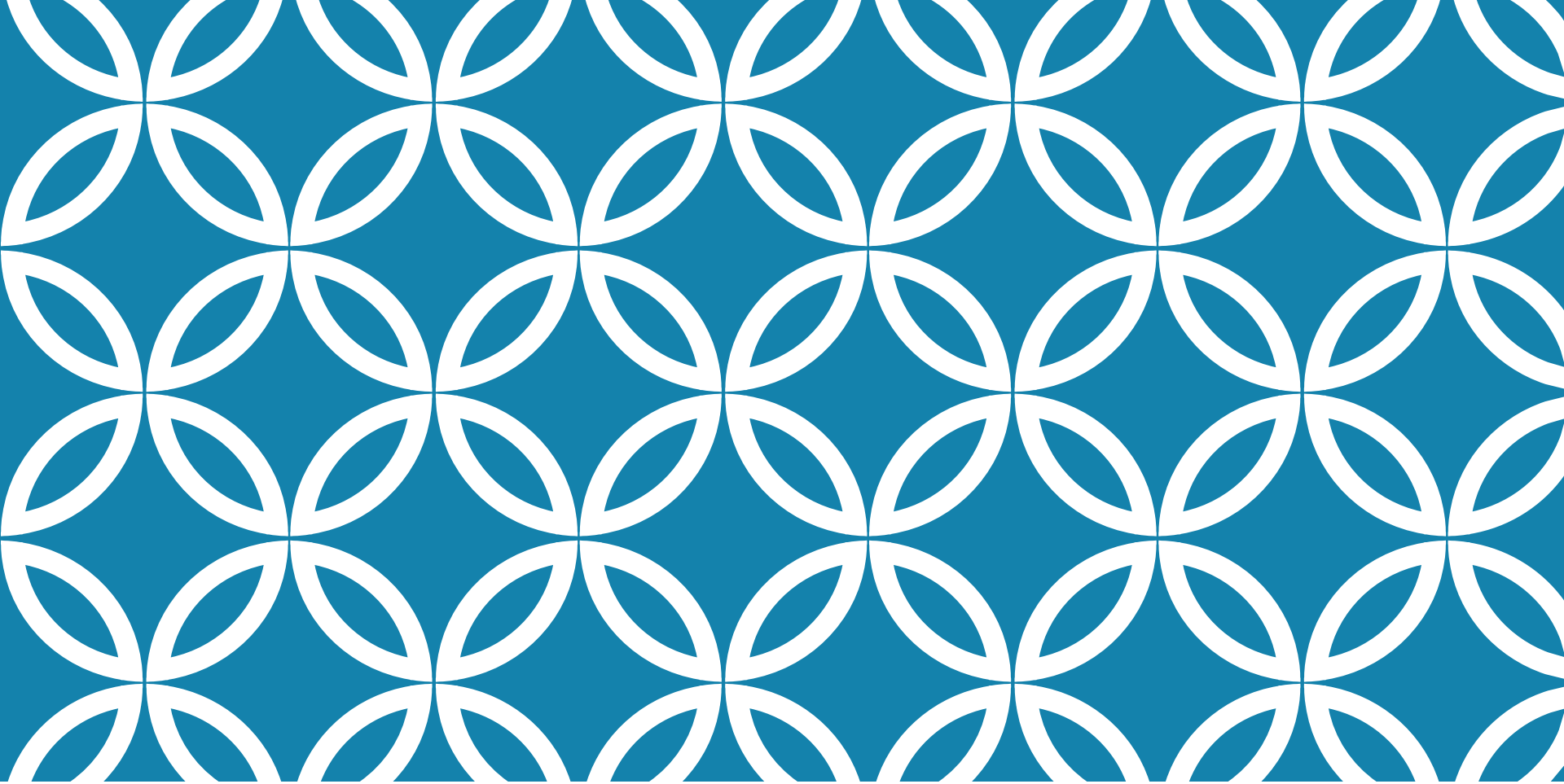


DESIGNING THE PROJECT PIPELINE

Antonio Luca Alfeo

CODE QUALITY

CODE QUALITY: PEP8 STANDARD

The primary focus of **PEP8** (Python Enhancement Proposal) standard is to improve the readability and consistency of Python code. Examples:

Use 4 spaces per indentation level. Use spaces, not tabs.

Limit all lines to a maximum of 79 characters.

Imports should be as specific as possible

Surround top-level functions and classes with two blank lines.

Surround method definitions inside classes with a single blank line.

Use blank lines sparingly inside functions to show clear steps.

Surround the following binary operators with a single space on either side:

- Assignment operators (=, +=, -=, and so forth)
- Comparisons (==, !=, >, <. >=, <=) and (is, is not, in, not in)
- Booleans (and, not, or)

PEP8 - NAMING

Function: Use a lowercase word or words. Separate words by underscores to improve readability. I.e. function, my_function

Variable: Use a lowercase single letter, word, or words. Separate words with underscores to improve readability. I.e. x, var, my_variable

Class: Start each word with a capital letter. Do not separate words with underscores. This style is called camel case. I.e. Model, MyClass

Method: Use a lowercase word or words. Separate words with underscores to improve readability. I.e. class_method, method

Constant: Use an uppercase single letter, word, or words. Separate words with underscores to improve readability. I.e. CONSTANT, MY_CONSTANT, MY_LONG_CONSTANT

Module: Use a short, lowercase word or words. Separate words with underscores to improve readability. I.e. module.py, my_module.py

Package: Use a short, lowercase word or words. Do not separate words with underscores

AUTOMATIC QUALITY CHECKING

Pylint is a source-code, bug and quality checker for the Python programming language. It checks:

if the code is compliant with Python's **PEP8** standard

if each module is properly imported and used

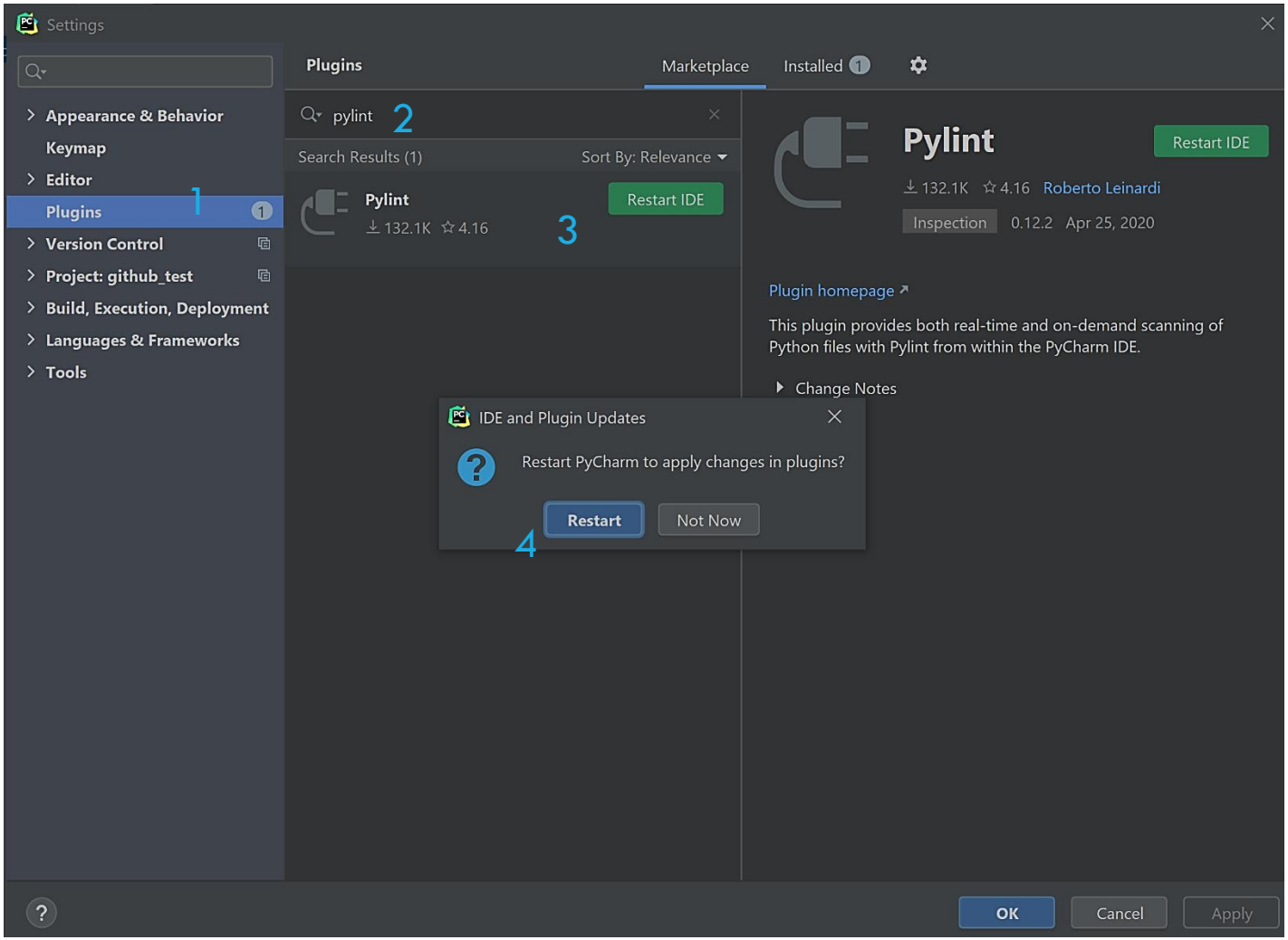
if declared interfaces are truly implemented

if there is duplicated code

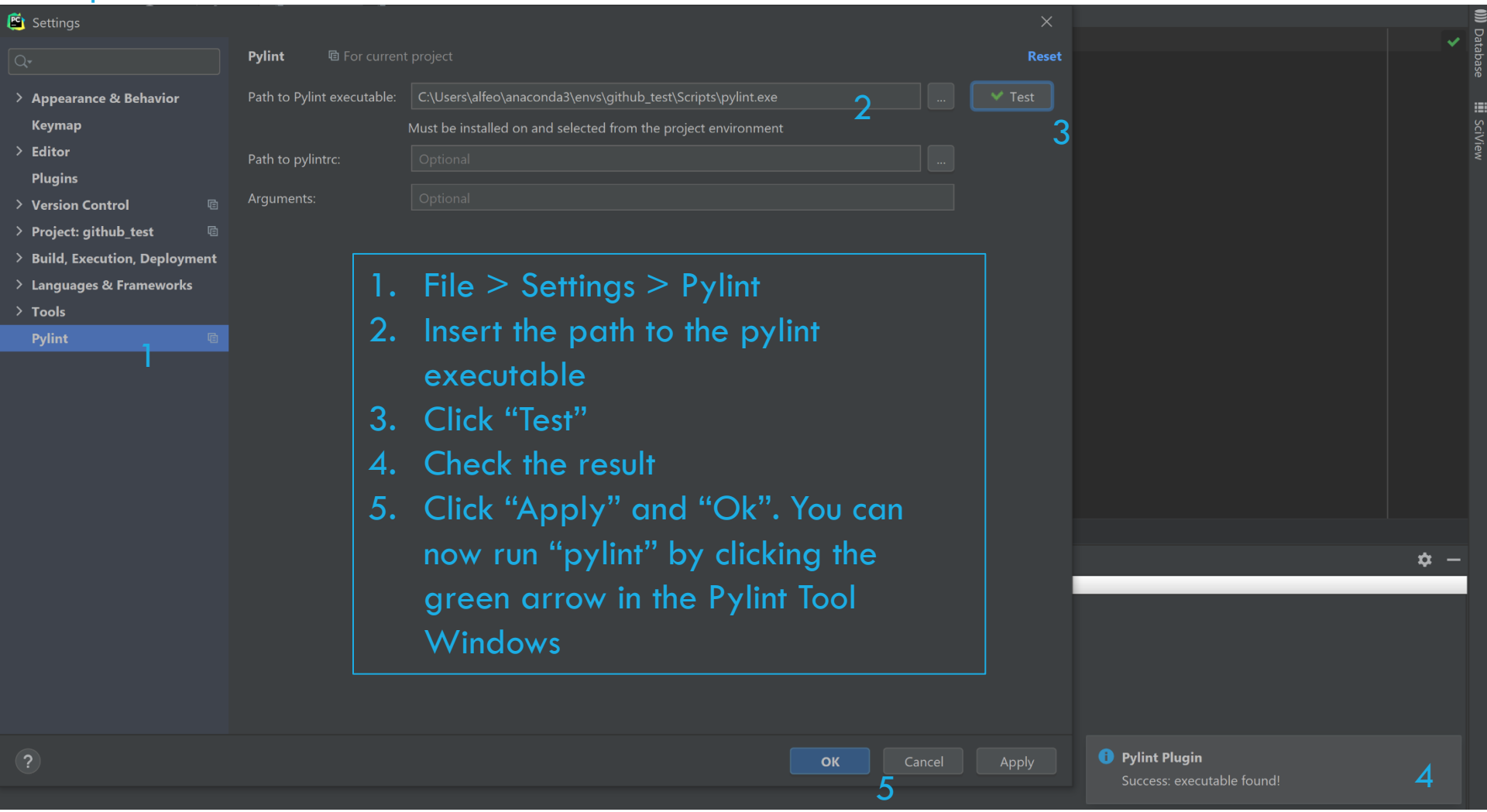
Pylint plugin is fully customizable: modify your *pylintrc* to customize which errors or conventions are important to you.

PYLINT PLUGIN INSTALLATION

1. File > Settings > Plugin
2. Search for “pylint”
3. Install Pylint plugin
4. Restart Pycharm



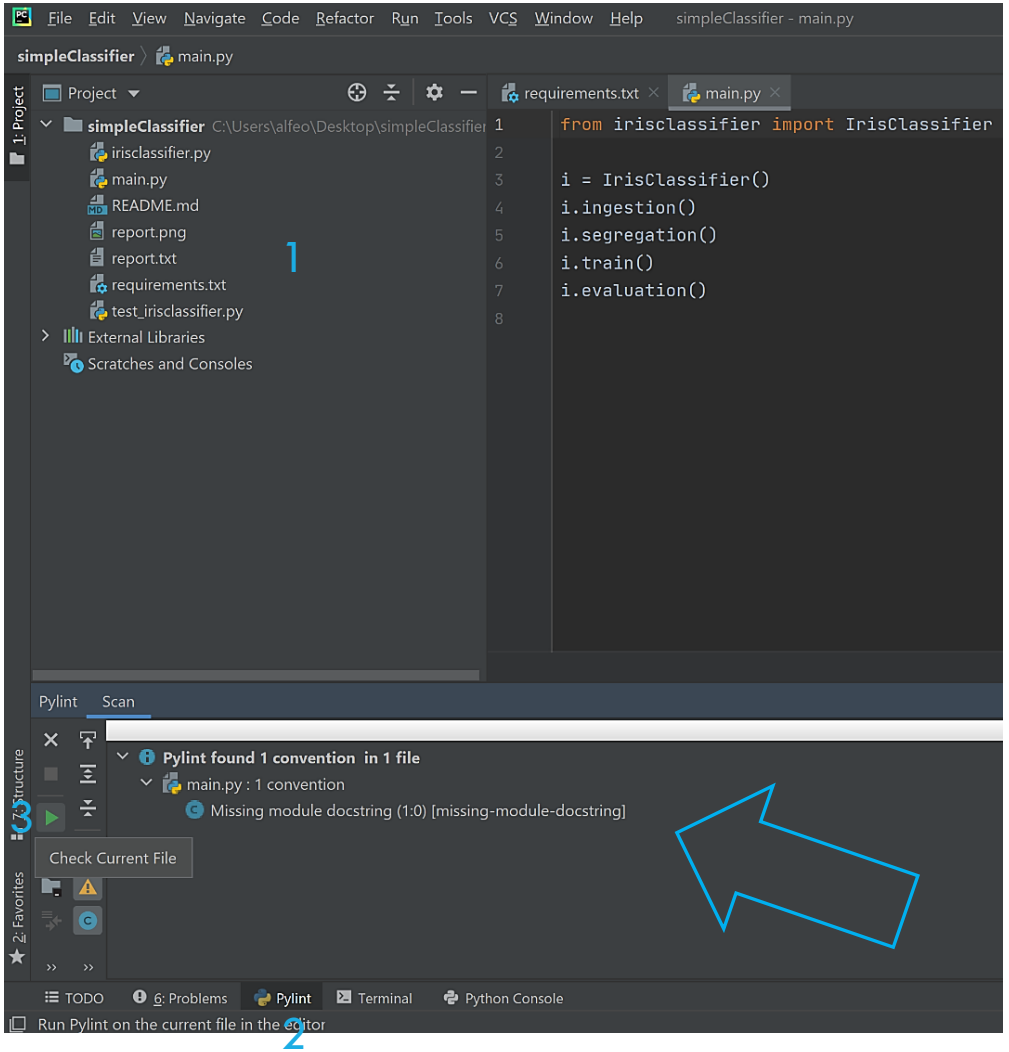
PYLINT PLUGIN SETUP



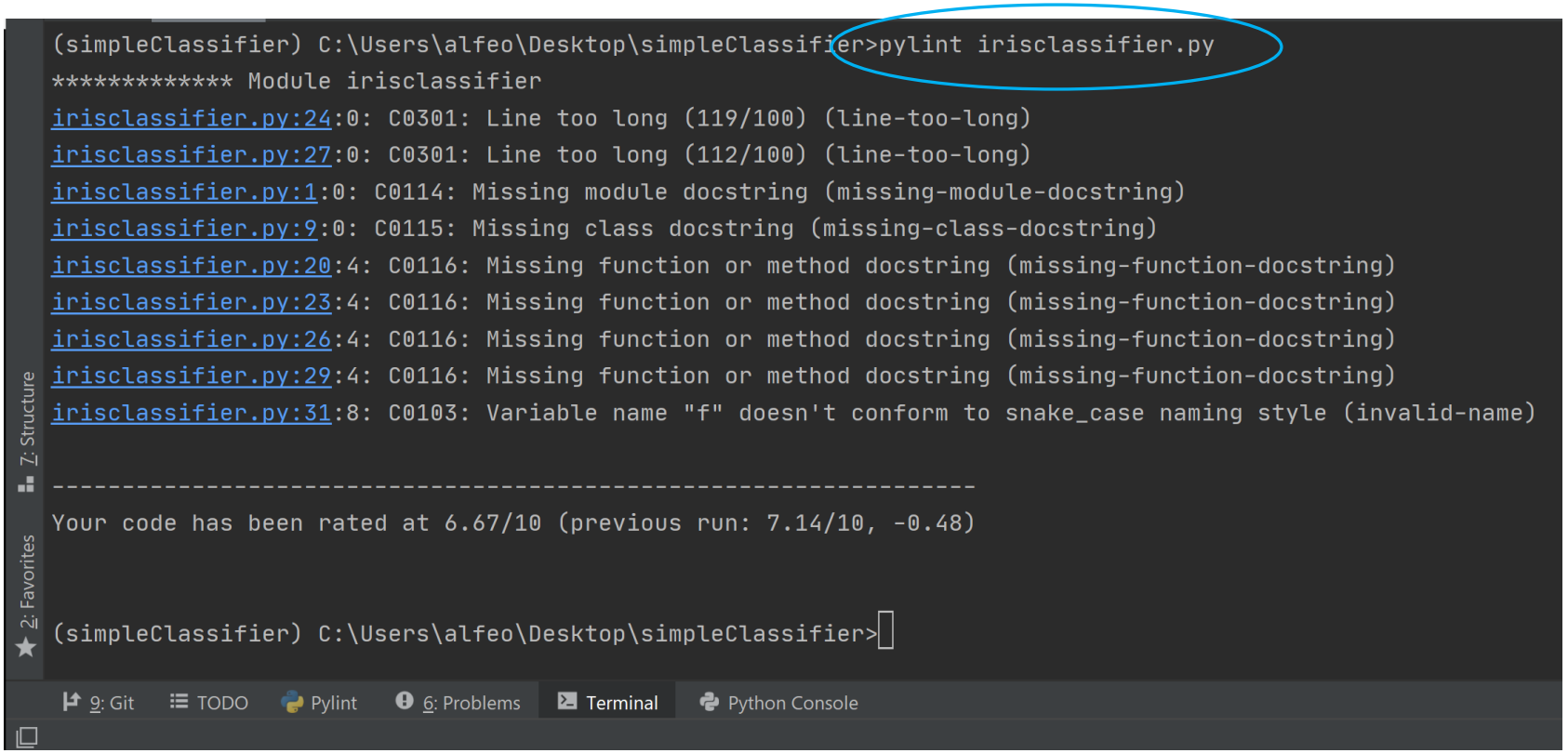
PYLINT TOOL WINDOW USAGE

1. Double click on a Python file in the Project View
2. Open the Pylint Tool Windows
3. Click the green arrow

P.S.= if you are using virtualenv with PyCharm you must also install pylint in the virtualenv so that you can use it from the IDE's dedicated window, regardless of whether pylint has been installed "globally." Of course, once installed, simply open the virtualenv in a terminal and use commands such as "where pylint.exe" to find the path to provide to the IDE in the project settings.



TRY PYLINT VIA COMMAND LINE

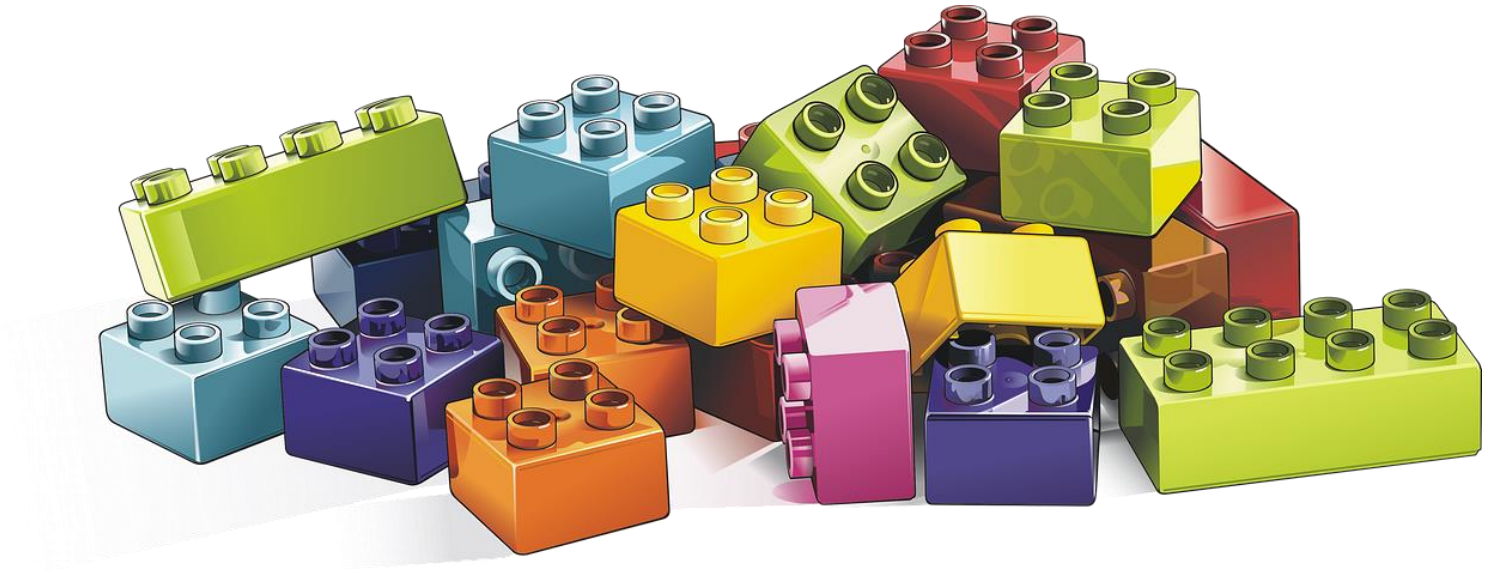


MORE ON CODING ASSISTANCE

Pycharm provides a number of functionalities aimed at helping you while writing your code, such as:

- **Python refactoring**:
  - renaming
  - extract methods
  - introduce variables and constants
- **Coding assistance and analysis**:
  - code completion
  - syntax and error highlighting
  - quick fixes
  - guided import
  - integrated Python debugger
  - code testing and code coverage

INTRODUCTION TO PACKAGES FOR YOUR PROJECT



Based on previous lecture by A. L. Alfeo

WORKSPACE FOLDER ORGANIZATION

The project folder and subfolder organization is up to you, but needs to be simple and clear, and allow to understand the processing put in place by each module.

DATA ANALYSIS IN PYTHON

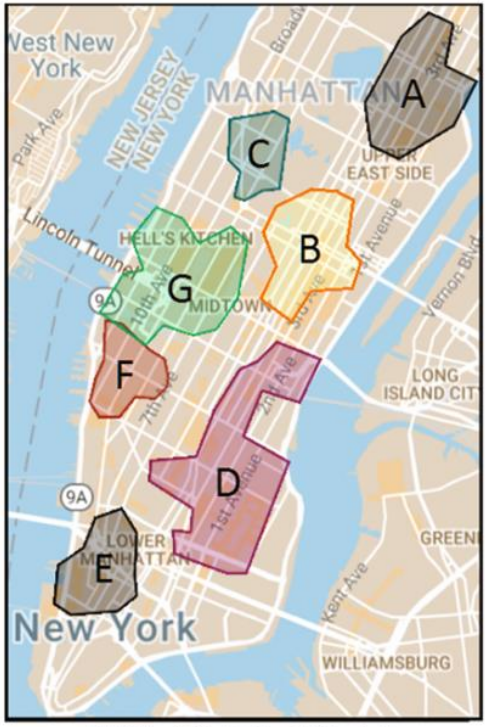
So many packages available:

- Pylint: to produce readable code
- Pandas: read data from a broad range of sources like CSV, and Excel files.
- Json and Joblib: to save/read configurations and models to/from files.
- Sci-kit learn: machine learning library with a broad range of classification algorithms. It provides also data pre-processing, pipelining and performance analysis modules.
- Matplotlib: graphic visualization libraries.

…and much more.

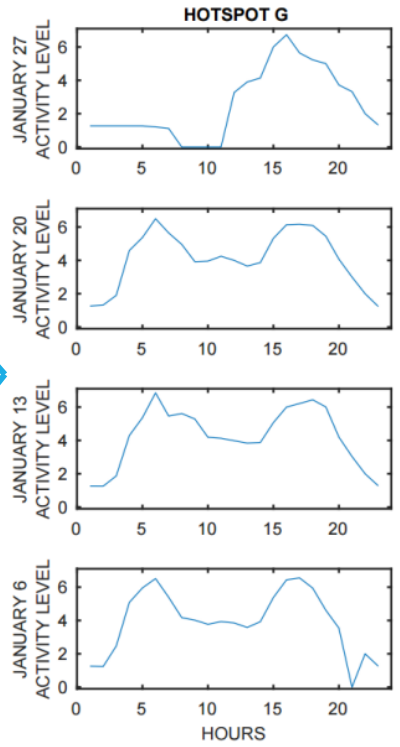
AN EXAMPLE…

Let's start with a very simple example (click to download the data).



HOTSPOT SET

By looking at mobility dynamics in the hotspot it's easy to distinguish between weekdays and weekend, or even to detect traffic anomalies (i.e. an parade, a storm).



UPLOAD/SAVE TABULAR DATA WITH PANDAS.DATAFRAME

```
from pandas import read_csv

# Read csv
data1 = read_csv('data/hotspotUrbanMobility-1.csv')
# Print dataframe shape
print(data1.shape)
# Read csv
data2 = read_csv('data/hotspotUrbanMobility-2.csv')
# Print dataframe shape
print(data2.shape)
# Append dataframes
data1 = data1.append(data2, ignore_index=True)
print(data1.shape)
# Save the new datafame as csv
data1.to_csv('data/completeDataset.csv', index=False)
```

Path of the csv to read

Dataframe shape consist of its numer of rows and column

Append 2 dataframes and ignore the indexes that pandas provides

Save the dataframe

DATAFRAME MANIPULATION WITH PANDAS.DATAFRAME

```
from pandas import read_csv

# Read csv
data = read_csv('data/completeDataset.csv')
# Check the content of the dataframe
print(data.describe())
# Remove column with constant values
data = data.drop('h24', axis=1)
# Remove anomalous instances
data = data.loc[data['Anomalous'] < 1]
print(data.describe())
# Save as csv
data.to_csv('data/preprocessedDataset.csv', index=False)
```

Provides statistics for each column in the dataframe

Drop the column (axis=1) whose label is 'h24'

Loc select instances by using a boolean array.

TRY THE LAST 2 SLIDES YOURSELF!

```
(198, 28)
(167, 28)
(365, 28)
        Anomalous     Cluster         Day  ...         h22         h23    h24
count  365.000000  365.000000  365.000000  ...  365.000000  365.000000  365.0
mean     0.101370    0.569863   15.720548  ...    0.472210    0.384617    1.0
std      0.302232    0.729045    8.808321  ...    0.373362    0.373992    0.0
min      0.000000    0.000000    1.000000  ...    0.000000    0.000000    1.0
25%      0.000000    0.000000    8.000000  ...    0.137218    0.130619    1.0
50%      0.000000    0.000000   16.000000  ...    0.403655    0.225237    1.0
75%      0.000000    1.000000   23.000000  ...    0.998552    0.783687    1.0
max      1.000000    2.000000   31.000000  ...    1.000000    1.000000    1.0

[8 rows x 28 columns]
       Anomalous     Cluster         Day  ...         h21         h22         h23
count      328.0  328.000000  328.000000  ...  328.000000  328.000000  328.000000
mean         0.0    0.530488   15.640244  ...    0.678014    0.455213    0.370639
std          0.0    0.716121    8.654346  ...    0.279946    0.371232    0.374739
min          0.0    0.000000    1.000000  ...    0.000000    0.000000    0.000000
25%          0.0    0.000000    8.000000  ...    0.467768    0.134583    0.129144
50%          0.0    0.000000   15.500000  ...    0.702473    0.381617    0.222655
75%          0.0    1.000000   23.000000  ...    1.000000    0.998032    0.774066
max          0.0    2.000000   31.000000  ...    1.000000    1.000000    1.000000

[8 rows x 27 columns]

Process finished with exit code 0
```

CREATING DATABASES WITH SQLITE3

SQLite is a database engine that makes it simple to store and work with relational data. In Python it can be used via **sqlite3**!

```
import sqlite3

con = sqlite3.connect('example.db')
cur = con.cursor()
# Create table and insert a row of data
cur.execute('CREATE TABLE <table_name> (<col_1>, <col_2>, <...>)')
cur.execute('INSERT INTO <table_name> VALUES ((<val_1>, <val_2>, <...>)')
# Save the changes
con.commit()
# Return all results of query
cur.execute('SELECT <...> FROM <...> WHERE <...> ')
cur.fetchall()
# Return first result of query
cur.execute('SELECT <...> FROM <...> WHERE <...>')
cur.fetchone()
# Close the connection
con.close()
```

PANDAS FOR SQLITE3 DATABASES

Once you have a working connection to the SQL database, you can use read_sql to query it or obtain a dataframe out of it and to_sql to save a dataframe as an SQL table.

```
import sqlite3
import pandas as pd
from sklearn.datasets import load_iris
from pandas import read_csv

con = sqlite3.connect('example.db')
# Obtain a dataframe
data = read_csv('data/preprocessedDataset.csv')
# Save it to the database
data.to_sql("mobility", con, if_exists="replace")
# Check the database
print(pd.read_sql("select * from mobility;", con))
# Save the changes and close the connection
con.commit()
con.close()
```

TRY THE LAST 2 SLIDES YOURSELF!

	index	Anomalous	Cluster	Day	...	h20	h21	h22	h23
0	0	0	1	2	...	1.000000	1.000000	1.000000	0.938944
1	1	0	1	3	...	1.000000	1.000000	0.998011	0.794555
2	2	0	2	4	...	0.610982	0.000000	0.252500	0.206939
3	3	0	0	5	...	0.464342	0.317294	0.139459	0.000000
4	4	0	0	6	...	0.772373	0.461212	0.000000	0.226510
..	...	...	...	...	...	...	...	...	...
323	323	0	0	23	...	0.742926	0.530848	0.484248	0.000000
324	324	0	2	27	...	0.730543	0.494042	0.000000	0.287482
325	325	0	0	28	...	0.599699	0.510087	0.578106	0.289053
326	326	0	0	29	...	0.989600	0.774025	0.523705	0.484244
327	327	0	0	30	...	1.000000	0.853960	0.563657	0.494982

[328 rows x 28 columns]

Process finished with exit code 0

MORE ON TABULAR DATA WITH PANDAS

- **Normalize JSON and convert them into dataframe**
- Load csv as a dataframe, save a dataframe as csv
- Shape, head and quick description of a dataframe
- Dataframe concatenation and append
- Data sort by index or by value
- Data resampling and augmentation
- Dataframe selection by value or index
- Dataframe columns/row addition and elimination
- Replace by value, drop duplicate rows, group by value
- Drop NAN values , fill NAN value
- Data interpolation and signal smoothing via moving average

DATA SEGREGATION WITH PANDAS AND SCIKIT-LEARN

```
from pandas import read_csv
from sklearn.model_selection import train_test_split

# Read csv
data = read_csv('data/preprocessedDataset.csv')
print(data.describe())
# Random split 75% as training and 25% as testing
training_data, testing_data, training_labels, testing_labels =
                        train_test_split(data.iloc[:, 4:len(data.columns)], data.iloc[:, 1])
# Save as csv
training_data.to_csv('data/trainingData.csv', index=False)
testing_data.to_csv('data/testingData.csv', index=False)
training_labels.to_csv('data/trainingLabels.csv', index=False)
testing_labels.to_csv('data/testingLabels.csv', index=False)
```

Split randomly 75% and 25% of the dataset (default), once it knows which are the target and the data.

iloc provides integer-location based indexing

Scikit-learn allows you also to discretize and normalize your data via minmax or standard scaler procedure. **Those are very important if you are going to use an artificial neural network!**

MODEL DESIGN 1\2

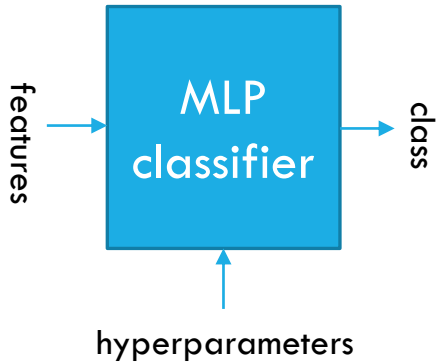
Scikit-learn provides an easy-to-use **classifier** based on artificial neural networks. Those use features, information extracted from an input data instance, to classify the instance itself. Here you can find the documentation to use them.

```
from sklearn.neural_network import MLPClassifier
from pandas import read_csv
from numpy import ravel
# Read the data
training_data = read_csv('data/trainingData.csv')
training_labels = read_csv('data/trainingLabels.csv')
testing_data = read_csv('data/testingData.csv')
testing_labels = read_csv('data/testingLabels.csv')
# Build and train the classifier
mlp = MLPClassifier(max_iter=100) .fit(training_data, training_labels)
mlp.predict(testing_data) # produce the predictions
mlp.score(testing_data, testing_labels) # compute the accuracy of the model
```

MODEL DESIGN 2\2

However, there is a number of **hyperparameters** to tune, such as:

- **Hidden layer number and size**
- **Batch size**
- **Max iterations**



Sometimes some recommendation can help you, but not always…

HYPER-PARAMETRIZATION VIA GRID SEARCH

```
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV
from pandas import read_csv
from numpy import ravel

# Read the data
training_data = read_csv('data/trainingData.csv')
training_labels = read_csv('data/trainingLabels.csv')
# setup Grid Search for MLP
mlp = MLPClassifier()
# values to test
parameters = {'max_iter': (100, 200, 300)}
# apply grid search
gs = GridSearchCV(mlp, parameters)
gs.fit(training_data, ravel(training_labels))
print(pd.DataFrame(gs.cv_results_)
        [['params','mean_test_score','rank_test_score']])
final_model = gs.best_estimator_
```

If "cv=None", GridSearch uses Cross-fold validation. Explicitly pass test and training sets with PredefinedSplit

The ML model

Max_iter is the name of the hyperparameter of MLPclassifier

Use ravel to transform dataframe to narray

Access all the results of each trial and the best model

CLASSIFIER EVALUATION

```
# compute labels on testing data
labels = final_model.predict(testing_data)
# evaluate accuracy score
score = accuracy_score(ravel(testing_labels), labels)
```

You can use different performances measures

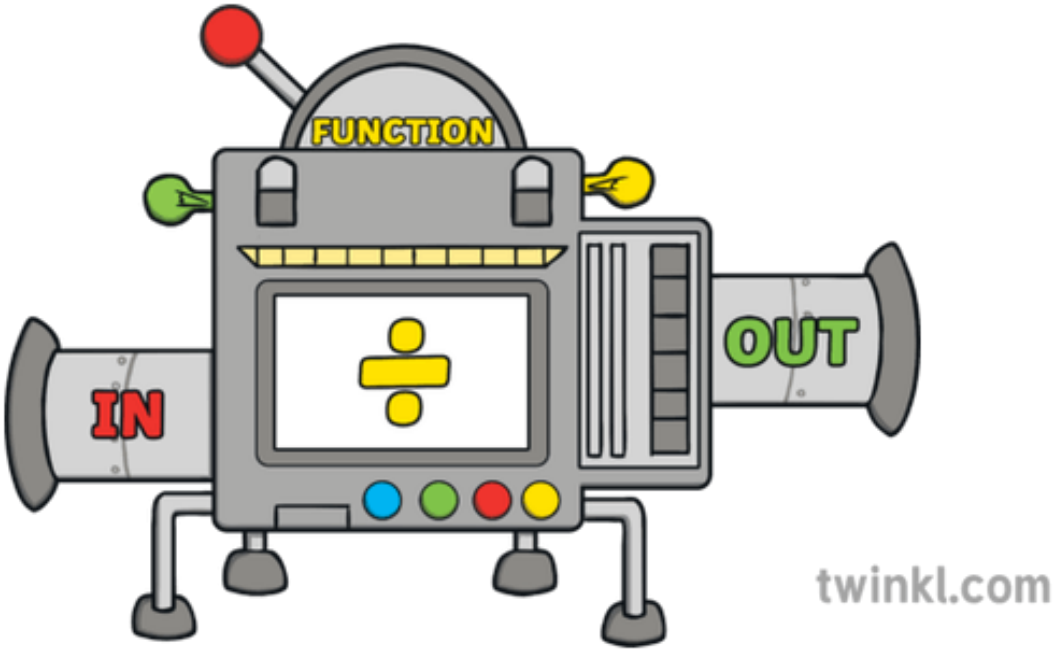
MORE ON MODEL EVALUATION…

- A lot of algorithms and performance metrics available on scikit-learn

- **How much time does an instruction take to execute?**

- **Performance visualization** with scikit-learn, and **how to write them on files**

INPUT/OUTPUT OF FUNCTIONAL MODULES



JSON TO SAVE THE BEST MODELS PARAMS

```
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV
from pandas import read_csv
from numpy import ravel
import json

# Read the data
training_data = read_csv('data/trainingData.csv')
training_labels = read_csv('data/trainingLabels.csv')
# setup Grid Search for MLP
mlp = MLPClassifier()
# values to test
parameters = {'max_iter': (100, 200, 300)}
# apply grid search
gs = GridSearchCV(mlp, parameters)
gs.fit(training_data, ravel(training_labels))
# save best configuration
config_path = 'config/modelConfiguration.json'
with open(config_path, 'w') as f:
    json.dump(gs.best_params_, f)
```

The ML model

Max_iter is the name of the hyperparameter of MLPclassifier

Use ravel to transform dataframe to narray

Save the best hyperparameter as JSON

FROM JSON TO OBJECT

```
import json

class Params():
    def __init__(self, M):
        self.max_iter = M

best_par = Params(60)
out_file = open("config/modelConfiguration.json", "w")
json.dump(best_par, out_file)
out_file.close()

# load best configuration
config_path = "config/modelConfiguration.json"
with open(config_path, "r") as f:
    params = json.load(f)

params_object = Params(**params)
```

Use a simple class to keep the model's parameters. You can use json.dump() to store it in a file.

Load the JSON . Remember, it should be compliant to the expected json schema i.e. pass the validation

Create a new object, and pass the JSON dictionary as a map to convert JSON data into a custom Python Object

JSON TO LOAD AN OBJECT

```
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
from pandas import read_csv
from numpy import ravel
import json

data1 = read_csv('data/hotspotUrbanMobility-1.csv')
data2 = read_csv('data/hotspotUrbanMobility-2.csv')
data = data1.append(data2, ignore_index=True)
data = data.drop('h24', axis=1)
data = data.loc[data['Anomalous'] < 1]
data.to_csv('data/preprocessedDataset.csv', index=False)
data = read_csv ('data/preprocessedDataset.csv' )
training_data, testing_data, training_labels, testing_labels =
                    train_test_split(data.iloc[:,
                    4:len(data.columns)], data.iloc[:, 1])
# …try to use the default 70-30 split
mlp = MLPClassifier(random_state=0)
parameters = {'max_iter': (100, 200, 300)}
# …try to use all the hyper-params at slide 63
gs = GridSearchCV(mlp, parameters)
#...try PredefinedSplit…
gs.fit(training_data, ravel(training_labels))
#...or try to pick the first runner up config
config_path = 'config/modelConfiguration.json'
with open(config_path, 'w') as f:
   json.dump(gs.best_params_, f)
with open(config_path, "r") as f:
   params = json.load(f)
model = MLPClassifier(**params).fit(training_data,
                    ravel(training_labels))
labels = model.predict(testing_data)
score = accuracy_score(ravel(testing_labels), labels)
print(score)
```

What's the accuracy score of your classifier?

MODEL DEPLOYMENT WITH JOBLIB

```
from pandas import read_csv
from sklearn.neural_network import MLPClassifier
from numpy import ravel
import joblib

# Read the training data
training_data = read_csv('data/trainingData.csv')
training_labels = read_csv('data/trainingLabels.csv')
# train the model
initial_model = MLPClassifier(random_state=0).fit(training_data, ravel(training_labels))
# save the model
joblib.dump(initial_model, 'config/fitted_model.sav')
#[...]
# Read new data
testing_data = read_csv('data/testingData.csv')
testing_labels = read_csv('data/testingLabels.csv')
# load the model
model = joblib.load('config/fitted_model.sav')
# Evaluate the initial model on new data
print('Evaluation score:')
print(model.score(testing_data, ravel(testing_labels)))
```

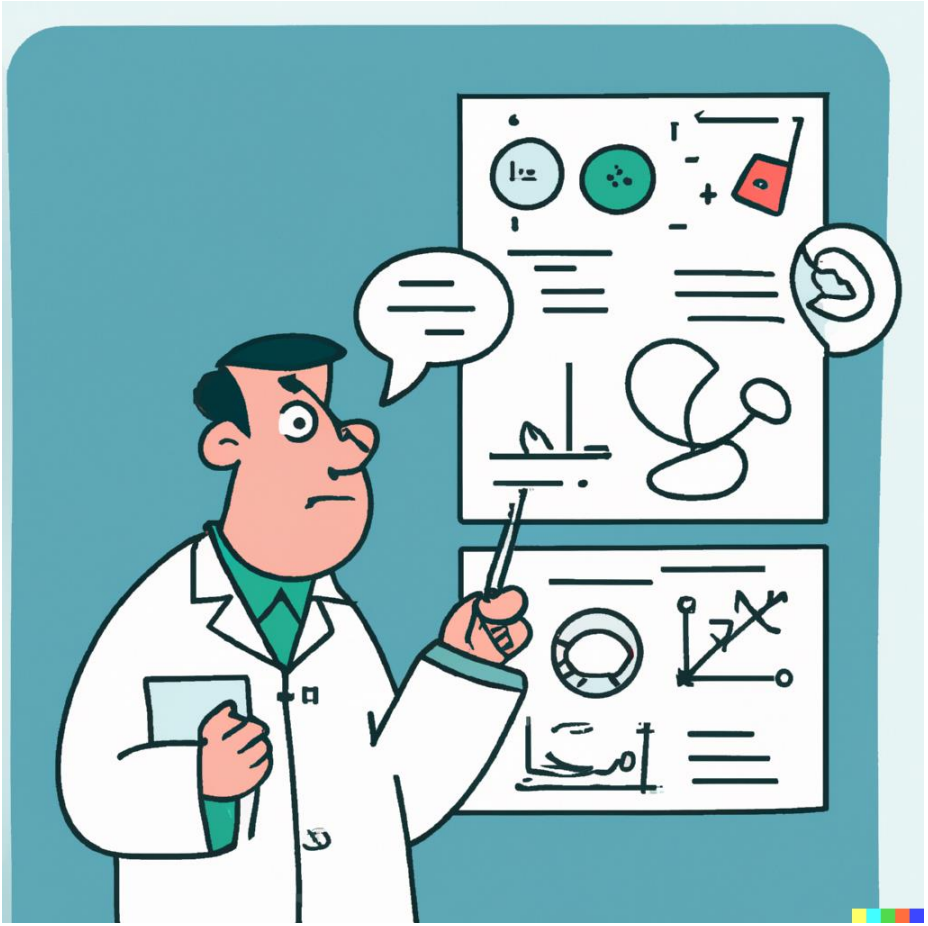
Random_state for reproducibility

Save a fitted model to file

Load a fitted model from file

Use the model with new data

PRESENT YOUR RESULTS



MATPLOTLIB

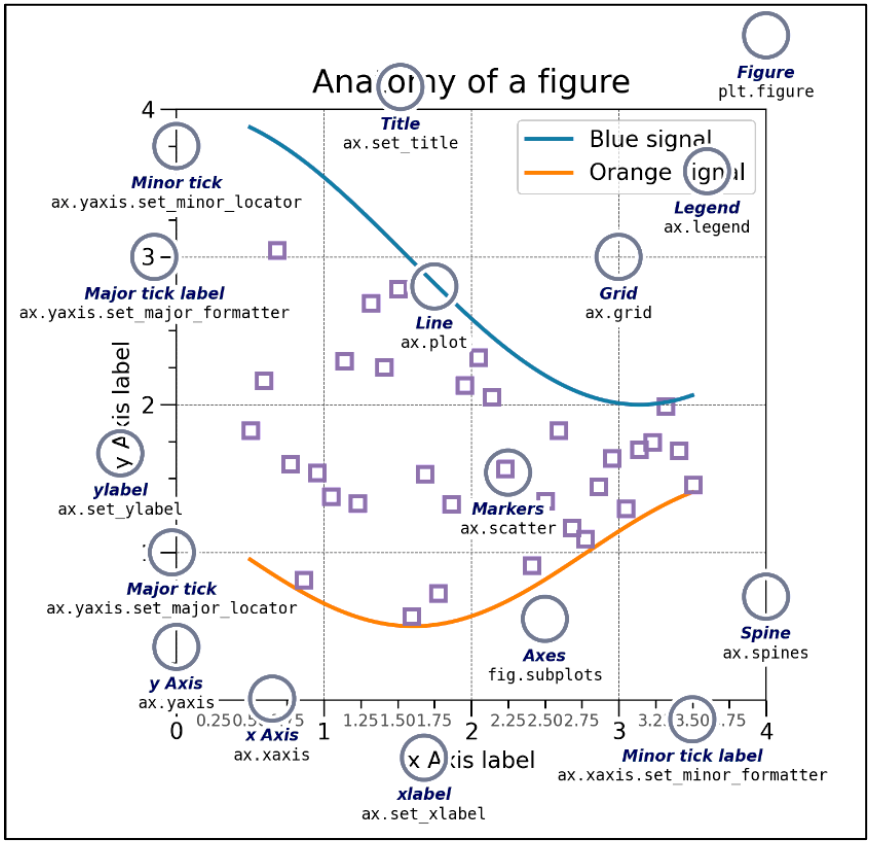
Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.

Each **element** in a figure can be addressed and modified.

There are two ways to use Matplotlib:

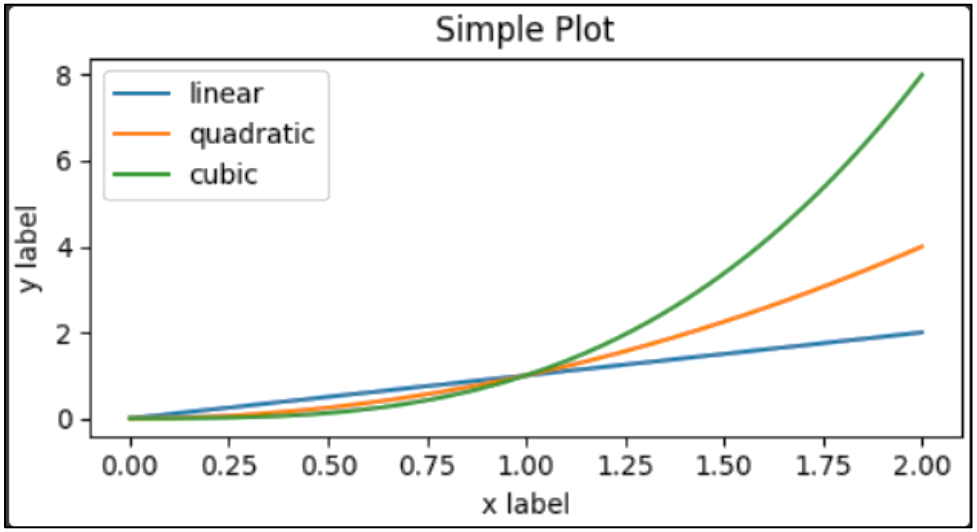
1. Explicit: explicitly create Figures and Axes, and call methods on them (the "object-oriented (OO) style").
2. **Implicit**: Rely on pyplot to implicitly create and manage the Figures and Axes, and use pyplot functions for plotting.

See Matplotlib Application Interfaces (APIs) for an explanation of the trade-offs between the implicit and explicit interfaces.



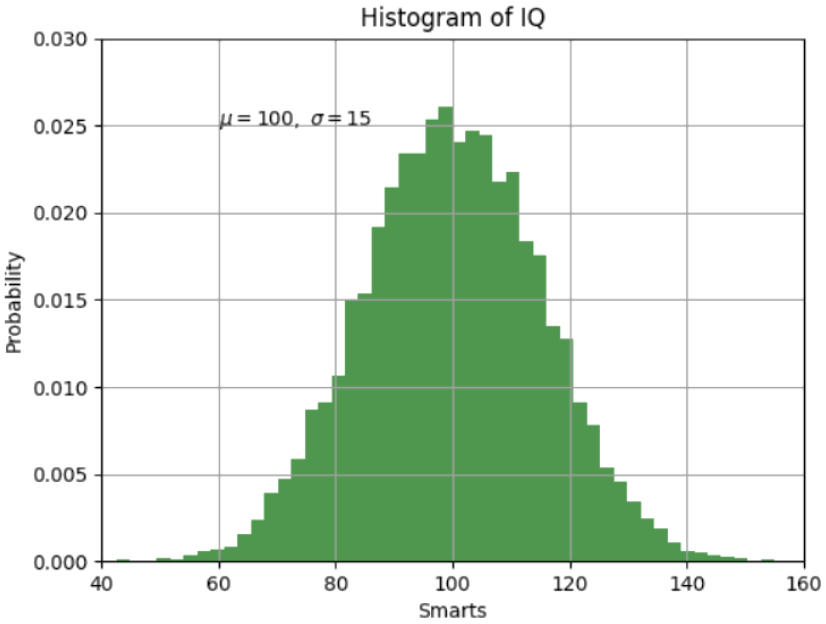
MULTIPLE LINES GRAPH

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

x = np.linspace(0, 2, 100)  # Sample data

plt.figure(figsize=(5, 2.7), layout='constrained')
plt.plot(x, x, label='linear')
plt.plot(x, x**2, label='quadratic')
plt.plot(x, x**3, label='cubic')
plt.xlabel('x label')
plt.ylabel('y label')
plt.title("Simple Plot")
plt.legend()
plt.show()
```



HISTOGRAMS

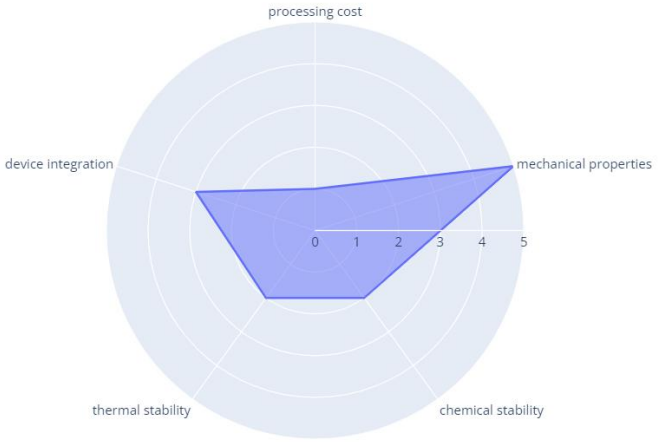
```
import matplotlib.pyplot as plt
import numpy as np

mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)

# the histogram of the data
n, bins, patches = plt.hist(x, 50,
density=True, facecolor='g', alpha=0.75)


plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title('Histogram of IQ')
plt.text(60, .025, r'$\mu=100,\ \sigma=15$')
plt.axis([40, 160, 0, 0.03])
plt.grid(True)
plt.show()
```



PLOTLY AND RADAR CHART

A Radar Chart (also known as a spider plot or star plot) displays multivariate data in the form of a two-dimensional chart of quantitative variables represented on axes originating from the center.
Use plotly.express to generate them!

```
import plotly.express as px
import pandas as pd
df = pd.DataFrame(
                  ( r=[1, 5, 2, 2, 3],
                    theta=['processing cost',
                           'mechanical properties','chemical stability',
                           'thermal stability', 'device integration']))
fig = px.line_polar(df, r='r', theta='theta', line_close=True)
fig.update_traces(fill='toself')
fig.show()
```



. . . MORE ON GRAPHS AND MATPLOTLIB.PYPLOT

- Plot and save a figure
- Histogram and barplot
- Boxplots
- Scatterplot
- Create single and multiple graphs
- Find out more on this TUTORIAL!!!

EXERCISE

Can you make an histogram out of the performances obtained by the grid search?

QUESTIONS?