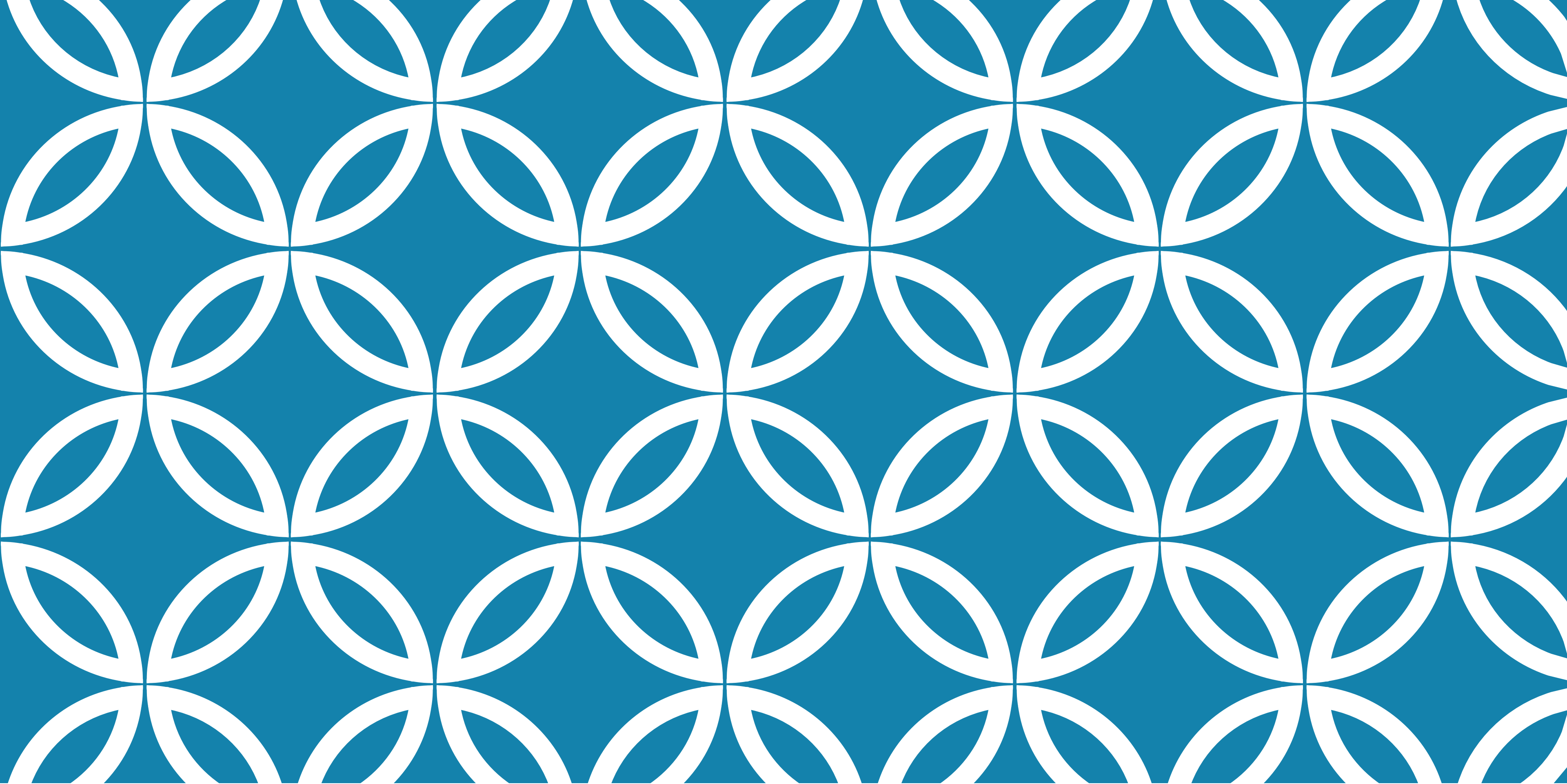
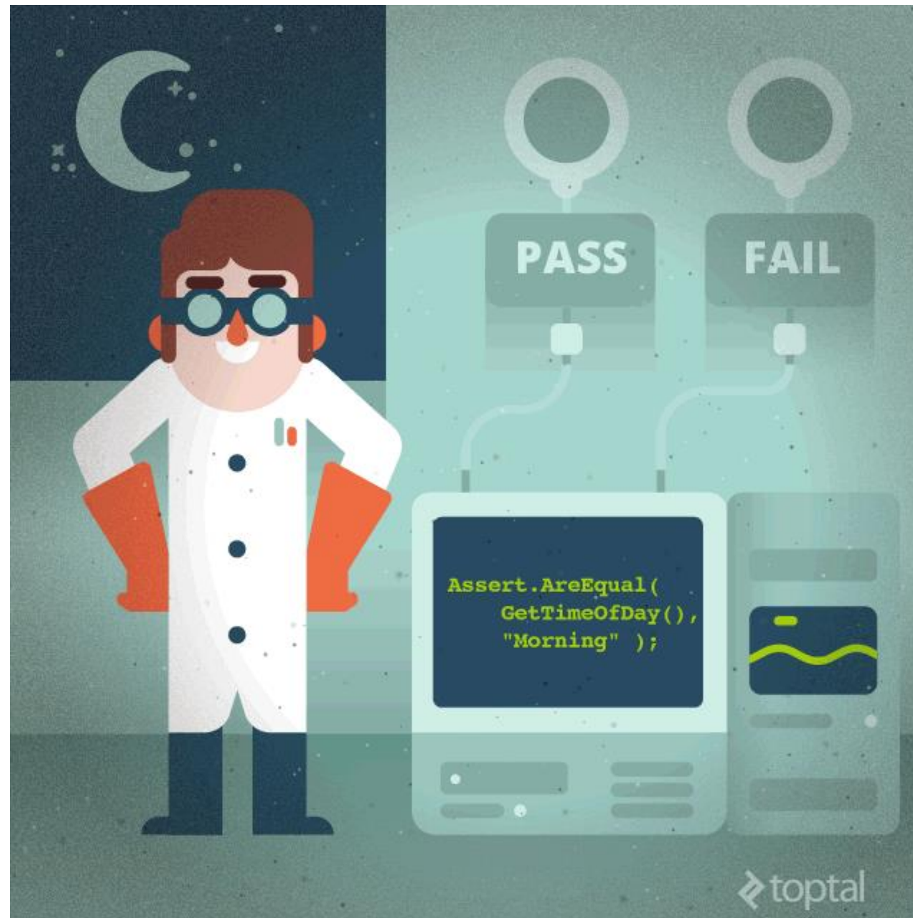


MODULES TESTING AND COMMUNICATION

Antonio Luca Alfeo

CODE TESTING



CODE CORRECTNESS CHECKING

PyCharm supports **pytest**, providing many testing features such as:

1. Dedicated test runner.
2. Code completion for test subject.
3. Code navigation.
4. Detailed failing assert reports.
5. Multiprocessing test execution.

..A QUICK NOTE

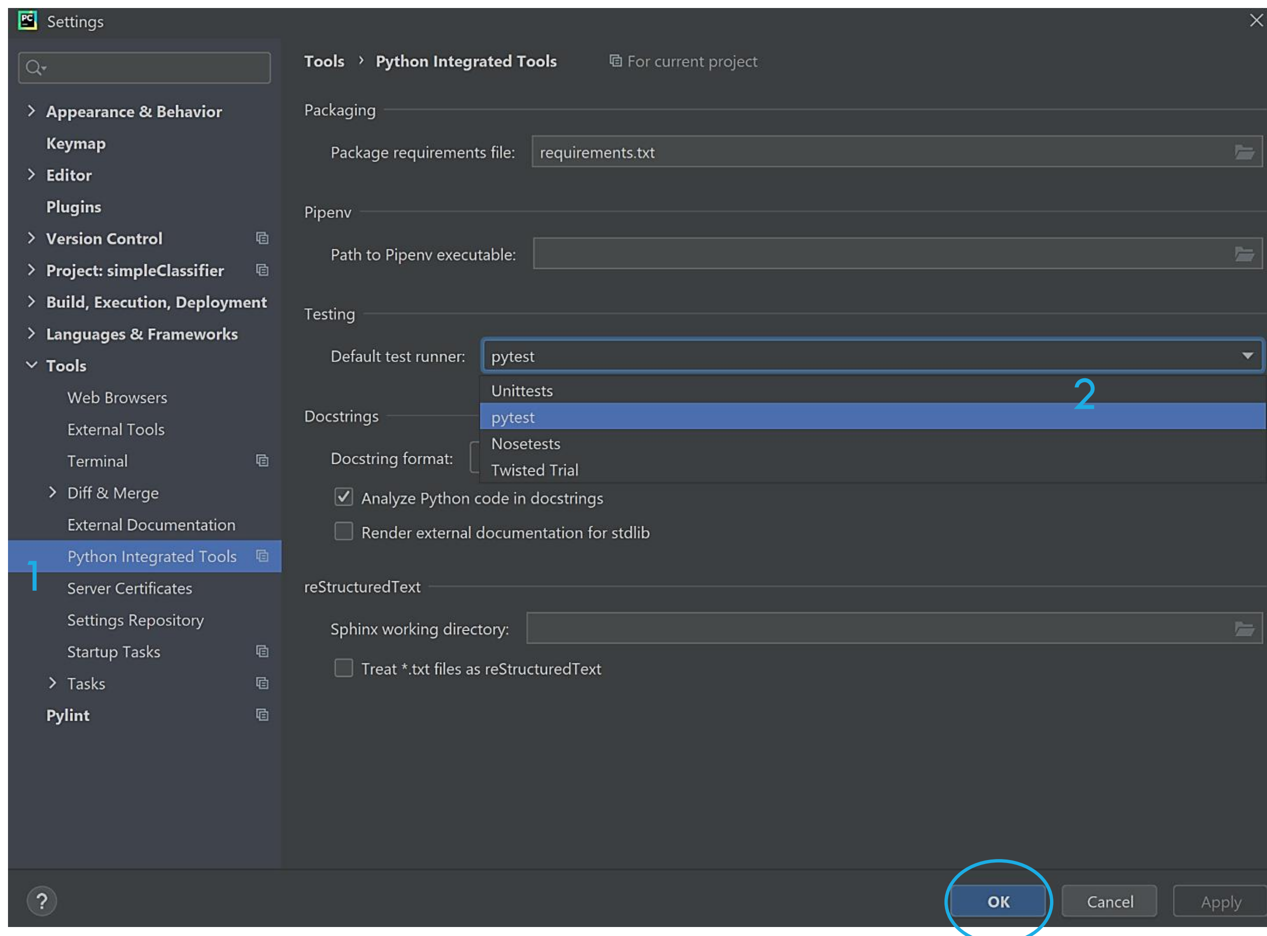
In the next slides, the functions described in the previous lessons are represented by a hypothetical class "irisClassifier" and its methods:

1. Ingestion: allows data to be uploaded via .csv and combine them
2. Segregation: prepare training and testing sets
3. Train: train the machine learning (ML) module
4. Evaluate: assess the performance of the ML module

In order to perform the next steps, prepare a “evaluate” function in which your trained model can be tested to obtain its score as result.

ENABLE PYTEST

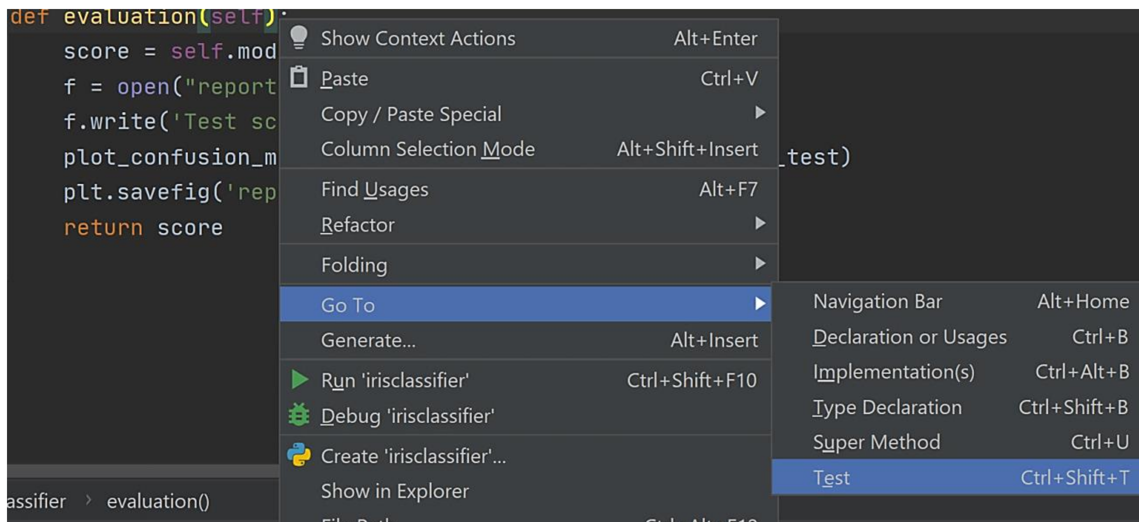
1. File > Settings > Python Integrated Tools
2. Set “pytest” as the “Default test runner”

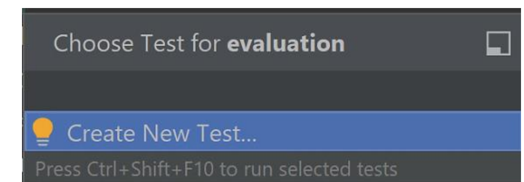


CREATE A NEW TEST

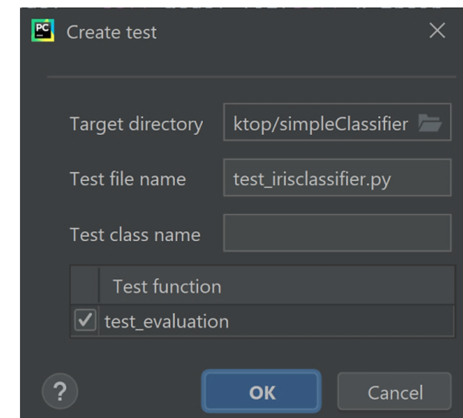
1. Put the cursor on the bracket of a method declaration, right-click and select Go To > Test
2. Create a new test script
3. Give it a directory, a name, and a method to test

1





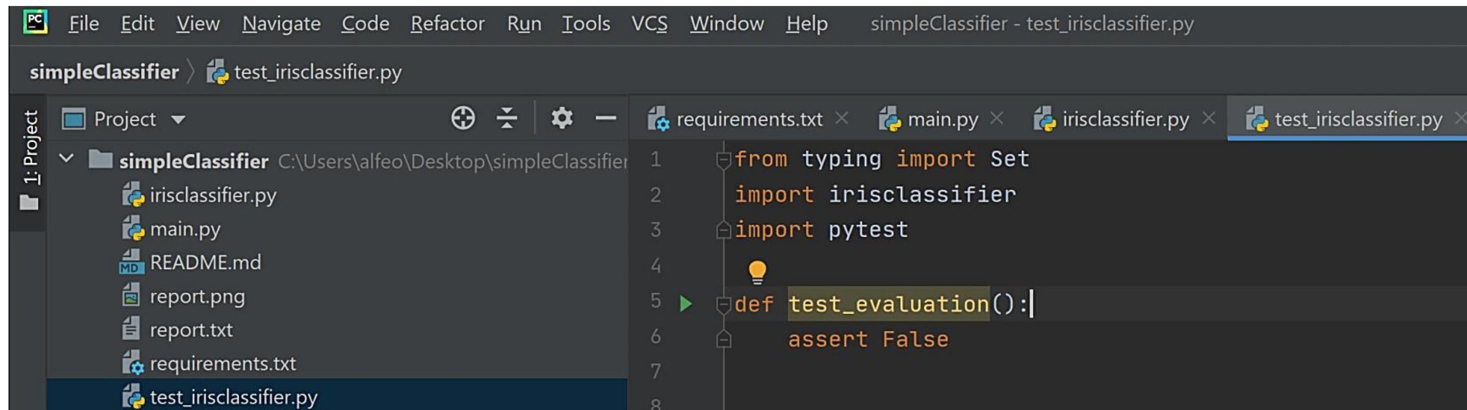
2



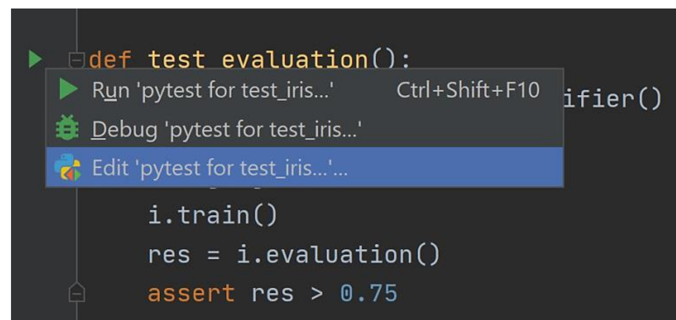
3

BUILD AND RUN A TEST 1/3

1. Fill the test function and the **assert** statement. If this statement is true the test is PASSED!



2. Click “Edit pytest for …”



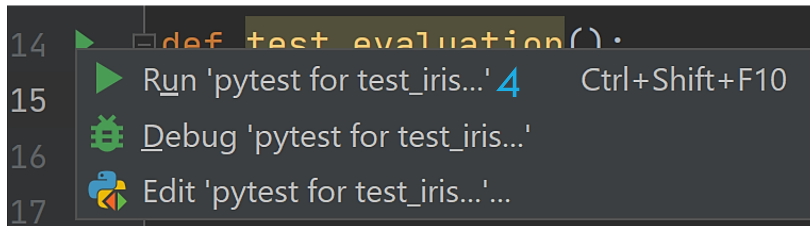
BUILD AND RUN A TEST 2/3

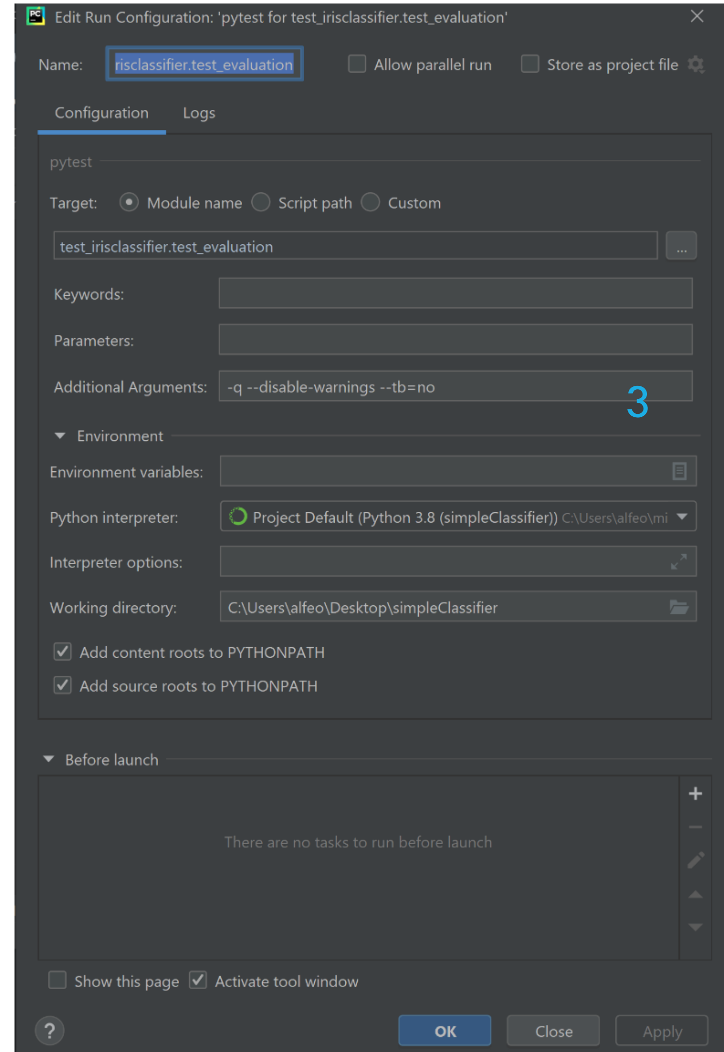
3. Reduce the verbosity of the pytest outcome to improve readability. Put the following Additional Arguments:

" -q --disable-warnings --tb=no "

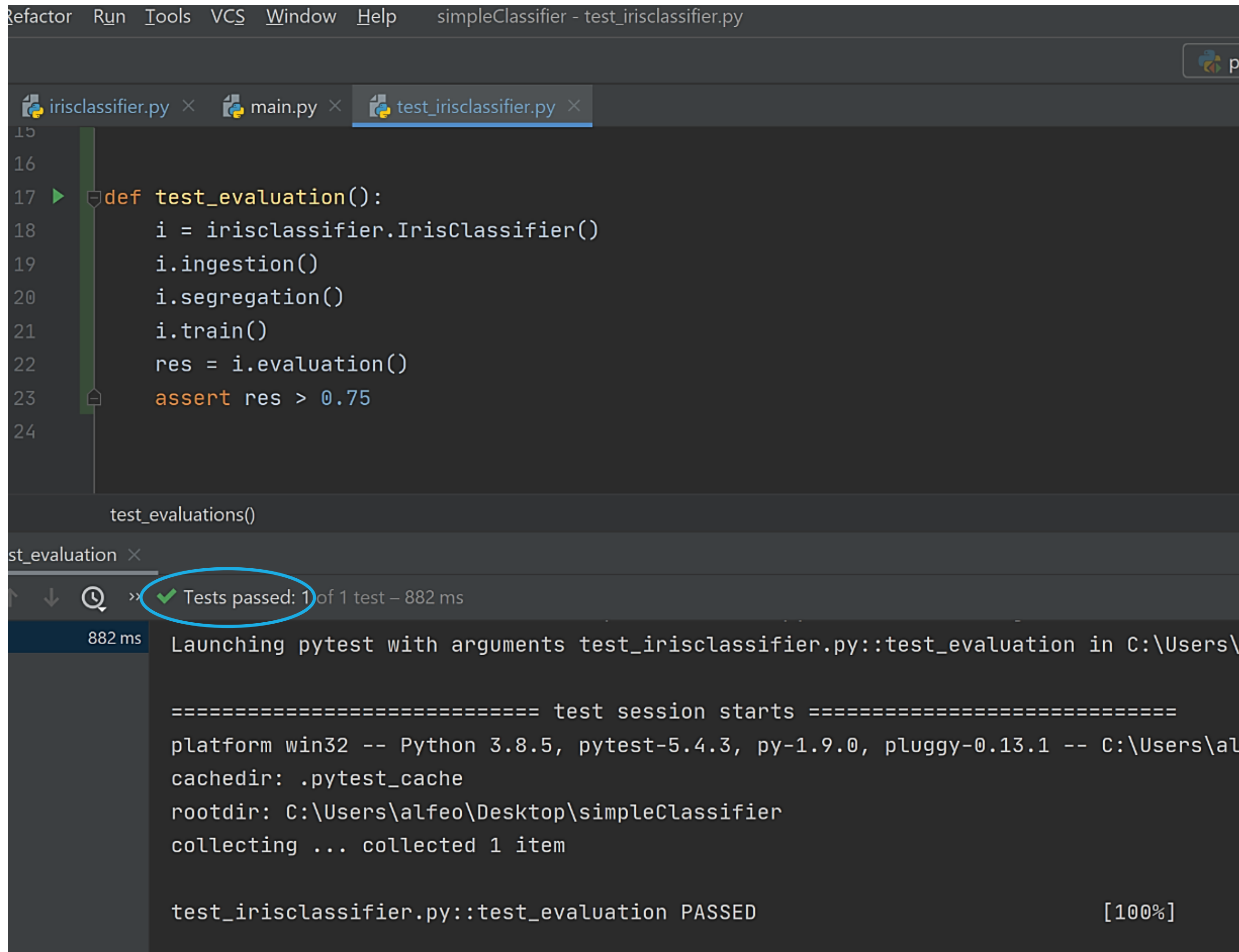
- -q, quiet output
- --disable-warnings, don't show most of the warnings
- --tb=no, don't show any traceback

4. Click OK and Run the test





BUILD AND RUN A TEST 3/3



TEST PARAMETRIZATION

The Pytest framework makes it easy to write tests to support multiple and complex functional testing for applications and libraries. Pytest enables test parametrization at several levels, such as:

@pytest.fixture provide a fixed baseline (e.g., an instance of a class) upon which tests can reliably and repeatedly execute. Moreover, via scope controls it is possible to check how often a fixture gets called.

@pytest.mark.parametrize allows the definition of multiple sets of arguments and fixtures for testing purpose.

BUILD AND RUN MANY TESTS

1. Prepare a set of values to test
2. Test those by passing them to the test function
3. Now you have one evaluation for each test value!

```
performance_thresholds = {0.50, 0.75, 0.95}
```

1

```
@pytest.mark.parametrize('th', performance_thresholds)
def test_evaluations(th):
    i = irisclassifier.IrisClassifier()
    i.ingestion()
    i.segregation()
    i.train()
    res = i.evaluation()
    assert res > th
```

2

```
======================= 3 passed, 3 warnings in 1.30s =======================

Process finished with exit code 0
PASSED                          [ 33%]Test score:  0.8947368421052632
PASSED                          [ 66%]Test score:  0.8947368421052632
PASSED                          [100%]Test score:  0.9736842105263158
```

3

USE PYTEST VIA COMMAND LINE

```
Terminal:  Local  +
Microsoft Windows [Version 10.0.19041.508]
(c) 2020 Microsoft Corporation. All rights reserved.

(simpleClassifier) C:\Users\alfeo\Desktop\simpleClassifier>pytest test_irisclassifier.py
========================================================================================= test session starts =========================================================================================
platform win32 -- Python 3.8.5, pytest-5.4.3, py-1.9.0, pluggy-0.13.1
rootdir: C:\Users\alfeo\Desktop\simpleClassifier
collected 6 items

test_irisclassifier.py ......                                                                                                                                                                  [100%]

========================================================================================== warnings summary ==========================================================================================
test_irisclassifier.py::test_evaluations[100]
test_irisclassifier.py::test_evaluations[200]
test_irisclassifier.py::test_evaluations[300]
test_irisclassifier.py::test_evaluations[50]
test_irisclassifier.py::test_evaluations[500]
test_irisclassifier.py::test_evaluation
  C:\Users\alfeo\miniconda3\envs\simpleClassifier\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:582: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (500
) reached and the optimization hasn't converged yet.
    warnings.warn(

-- Docs: https://docs.pytest.org/en/latest/warnings.html
================================================================================= 6 passed, 6 warnings in 8.69s ==================================================================================

(simpleClassifier) C:\Users\alfeo\Desktop\simpleClassifier>
```

9: Git   TODO   Pylint   6: Problems   Terminal   Python Console   Event Log

LET'S WORK AS A TEAM!

Based on previous lecture by A. L. Alfeo

REST API AS INTERFACES 1\3

REST APIs are becoming the standard de-facto for machine-to-machine communication, and can be used to communicate among the different modules of your project. For instance you can:

- Generate some end-points for the functionalities to expose
- Use POST method to create a new resource (e.g. build the training set, pre-process it)
- Use GET method to retrieve a resource from different components of the application (e.g. the configuration file, or the JSON version of a configuration object)
- Implement DELETE and PUT functionalities, that's up to you!

REST API AS INTERFACES 2\3

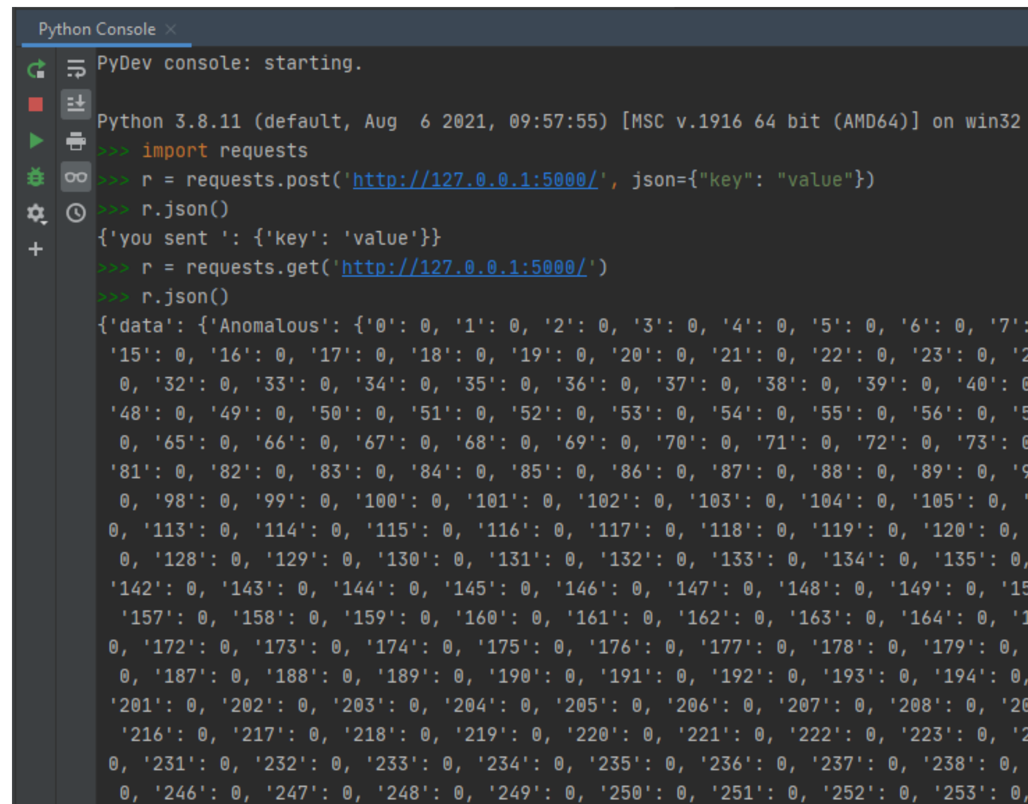
In Python you can use Flask and Flask-Restful (install them via pip) to build and use a simple REST server. For instance you can define the end-point «/» as it follows

```python
from pandas import read_csv
from flask import Flask, request
from flask_restful import Resource, Api

app = Flask(__name__)
api = Api(app)

class HelloWorld(Resource):
   def get(self):
      data = read_csv('data/preprocessedDataset.csv')
      data = data.to_dict()
      return {'data': data}, 200

   def post(self):
      some_json = request.get_json()
      return {'you sent ': some_json}, 201

api.add_resource(HelloWorld, '/')

if __name__ == '__main__':
   app.run(debug=True)
```

```
Run:  main
 * Serving Flask app 'main' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 977-993-967
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [13/Oct/2021 16:37:27] "GET / HTTP/1.1" 200 -
```

TRY IT YOURSELF USING REQUEST

1. Install requests package (e.g. via pip) from the Terminal
2. Send a request via Python Console



REST API AS INTERFACES 3\3

You can also use an **argument** with your request as it follows

```
from flask import Flask, request
from flask_restful import Resource, Api

app = Flask(__name__)
api = Api(app)

class Multi(Resource):
    def get(self, num):
        return {'result ': num * 10}

api.add_resource(Multi, '/multi/<int:num>')

if __name__ == '__main__':
    app.run(debug=True)
```

TRY IT YOURSELF USING REQUEST

Send a request via Python Console

```
>>> r = requests.get('http://127.0.0.1:5000/multi/4')
>>> r.json()
{'result ': 40}
```

SENDING FILES

```
from flask import Flask, send_file

app = Flask(__name__)  # always use this


# the file in "file_path" will be sent due to a GET request on the endpoint "/get_file"
@app.route("/get_file", methods=['GET'])
def test():
    file_path = 'file.sav'
    return send_file(file_path, as_attachment=True)


app.run()
```

EXERCISE

Can you send the trained model you saved (via joblib) in the last laboratory to your closest colleague?

QUESTIONS?