Project Rules

Naming Conventions

folders : **snake_case**

Classes : **PascalCase,** if more than one word, or just simply capitalize the first letter of the class name

functions: **snake_case**

Module : **snake_case,** tells what the module does, e.g read_data which reads any filesource and returns a specific datatype

e.g. binary_to_dec

Folder Structures

General structure: Group related files together.

Packages: Should only contain Modules and other necessary file like files like requirement.txt, setup.txt, init.py, etc.

Packages:

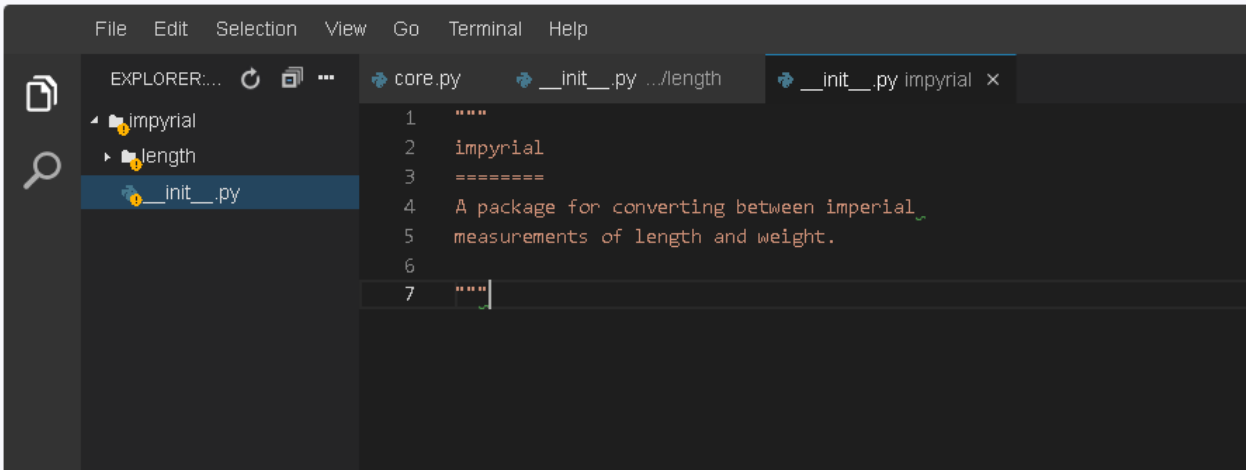
Documentation: Document each Function, Class, Class Method

Documentation Style: Google documentation Style or **NumPy Style**

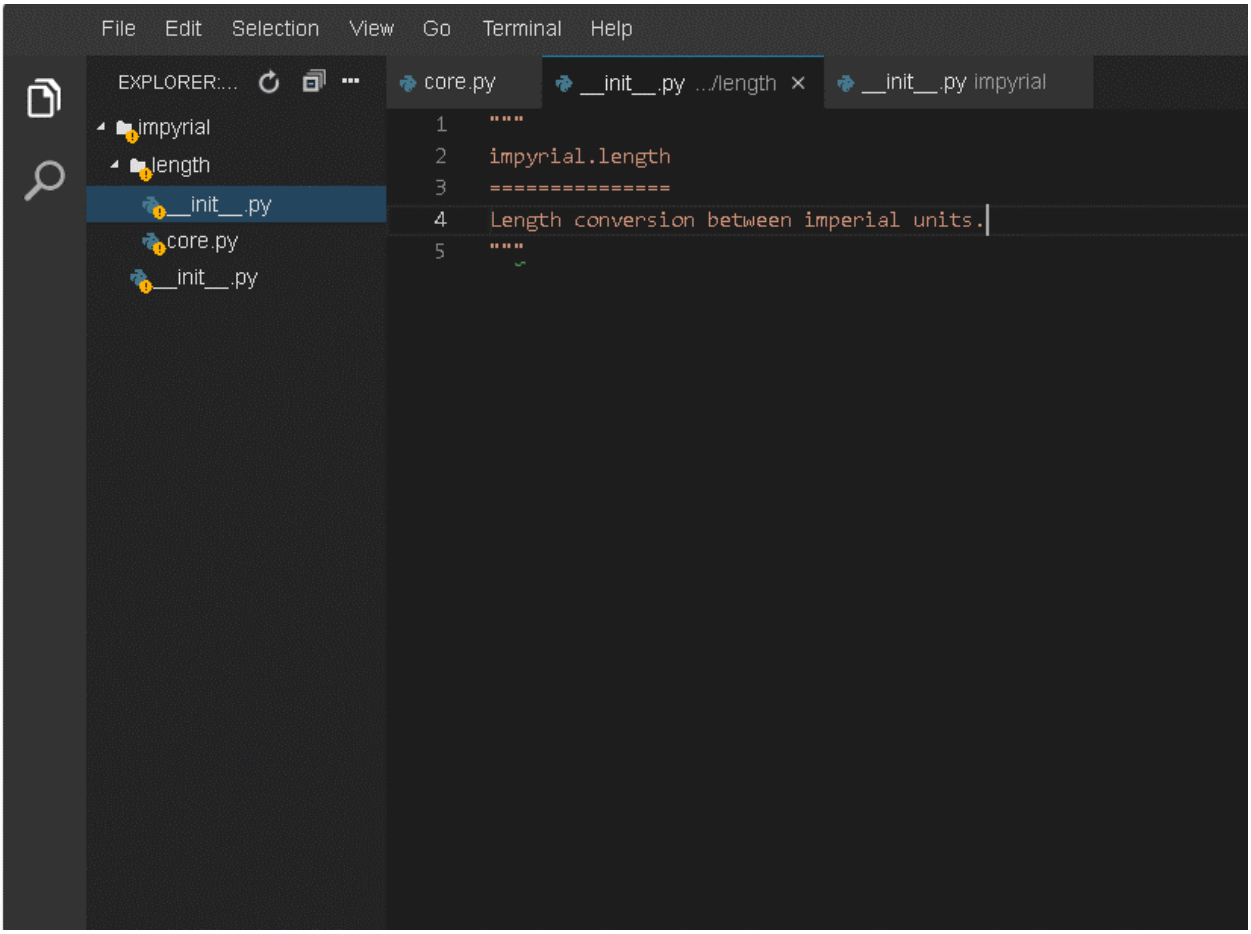
NumPy Style Documentation

1. a Docstring at the top of each function
2. It begins and ends with three quotation marks
3. First sentence of the documentaion is a summary, and should be read as a command, like youre telling the function what to do.
4. After the summary, next are the sections that outline the input parameters and return values

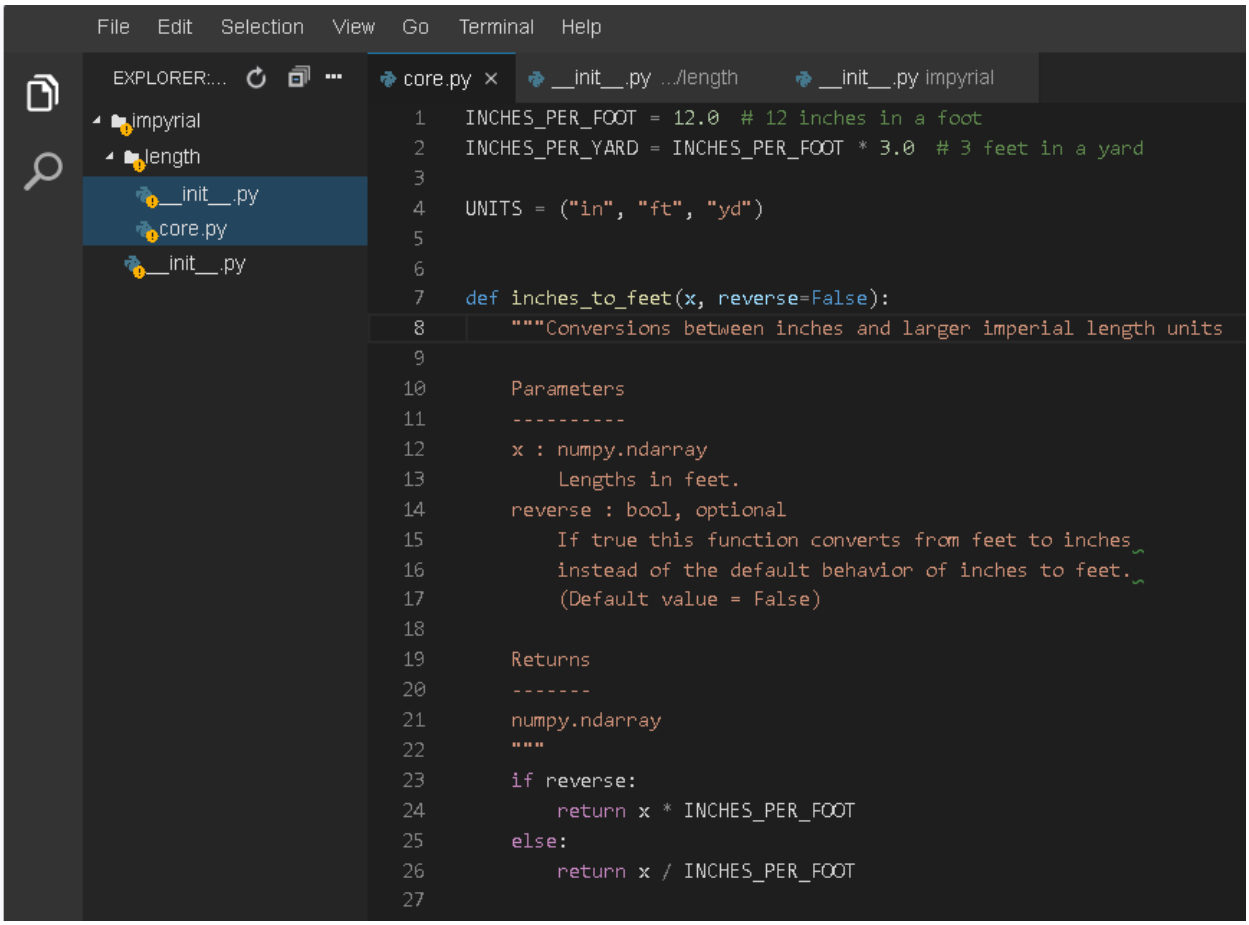
Package level documentation



Sub package documentation



Module Documentation



Generating boilerpate for documentation

Documentation Tool: **Pyment**

- Go inside the folder of the module, run this command:
- pyment -w -o numpydoc <module_name>.py

Structuring Imports

Relative imports

- Use of the full path to the module starting from the project's root directory.

Absolute imports (what we will use)

- Use of the relative path to the module from the current module's location.

Importing modules

`mysklearn/preprocessing/__init__.py`

Absolute import

```
from mysklearn.preprocessing import normalize
```

Relative import

```
from . import normalize
```

Directory tree for package with subpackages

```
mysklearn/
|-- __init__.py
|-- preprocessing
|   |-- __init__.py    <--
|   |-- normalize.py
|   |-- standardize.py
|-- regression
|   |-- __init__.py
|   |-- regression.py
|-- utils.py
```

Now when we import the package, we can access all the functions in this module.

Installation of Package

- It allows easy import of the package anywhere in the project

Why should you install your own package?

Inside `example_script.py`

```
import mysklearn
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'mysklearn
```

Directory tree

```
home/
|-- mypackages
|   |-- mysklearn     <---
|       |-- __init__.py
|       |-- preprocessing
|       |   |-- __init__.py
|       |   |-- normalize.py
|       |   |-- standardize.py
|       |-- regression
|           |-- __init__.py
|           |-- regression.py
`-- myscripts
    `-- example_script.py     <---
```

Package directory structure

Directory tree for package with subpackages

```
mysklearn/      <-- outer directory
|-- mysklearn   <--- inner source code director
|   |-- __init__.py
|   |-- preprocessing
|   |   |-- __init__.py
|   |   |-- normalize.py
|   |   |-- standardize.py
|   |-- regression
|   |   |-- __init__.py
|   |   |-- regression.py
|   |-- utils.py
|-- setup.py   <-- setup script in outer
```

Inside setup.py

```python
# Import required functions
from setuptools import setup, find_packages

# Call setup function
setup(
    author="James Fulton",
    description="A complete package for linear regression.",
    name="mysklearn",
    version="0.1.0",
    packages=find_packages(include=["mysklearn", "mysklearn.*"]),
)
```

my-sklearn-dot-star tells the function to include all the subpackages inside my-sklearn as well.

Editable installation

```
pip install -e .
```

- `.` = package in current directory
- `-e` = editable

Directory tree for package with subpackages

```
mysklearn/  <-- navigate to here
|-- mysklearn
|   |-- __init__.py
|   |-- preprocessing
|   |   |-- __init__.py
|   |   |-- normalize.py
|   |   |-- standardize.py
|   |-- regression
|   |   |-- __init__.py
|   |   |-- regression.py
|   |-- utils.py
|-- setup.py
```

Inside the setup.py in the terminal, type “pip install -e .”

Remember to include a ‘ . ‘ at the end.

Adding dependencies to setup.py

```
from setuptools import setup, find_packages

setup(
    ...
    install_requires=['pandas', 'scipy', 'matplotlib'],
)
```

Include the modules used in the package or module so that the other users won't have to install them one by one.

Controlling dependency version

```
from setuptools import setup, find_packages

setup(
    ...
    install_requires=[
        'pandas>=1.0',          # good
        'scipy==1.1',           # bad
        'matplotlib>=2.2.1,<3' # good
    ],
)
```

- Allow as many package versions as possible

Making an environment for developers

Save package requirements to a file

```
pip freeze > requirements.txt
```

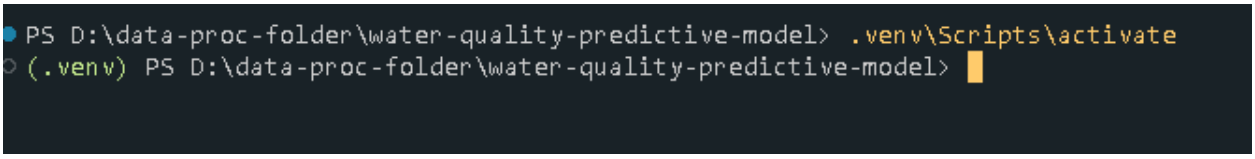
```
mysklearn/
|-- mysklearn
|   |-- __init__.py
|   |-- preprocessing
|   |   |-- __init__.py
|   |   |-- normalize.py
|   |   |-- standardize.py
|   |-- regression
|   |   |-- __init__.py
|   |   |-- regression.py
|   |-- utils.py
|-- setup.py
|-- requirements.txt   <-- developer environmen
```

If you think your package or modules are ready for usage, you may type “ pip freeze > requirements.txt” in the terminal. Make sure you are inside of the root folder.

Virtual Environment Setup

In the terminal make sure you are inside of the inside the root folder. Then type this command in the terminal “py -3 -m venv .venv”

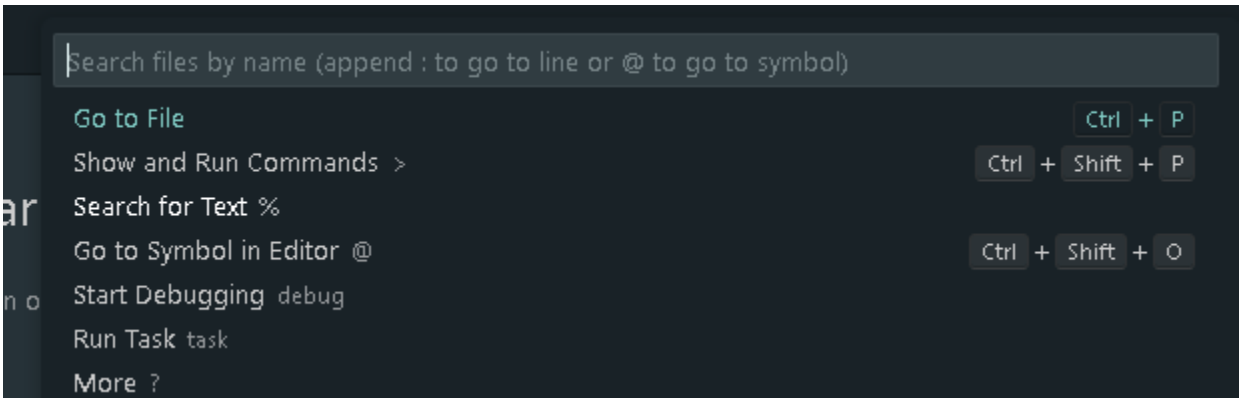
A file named “.venv” will be generated.



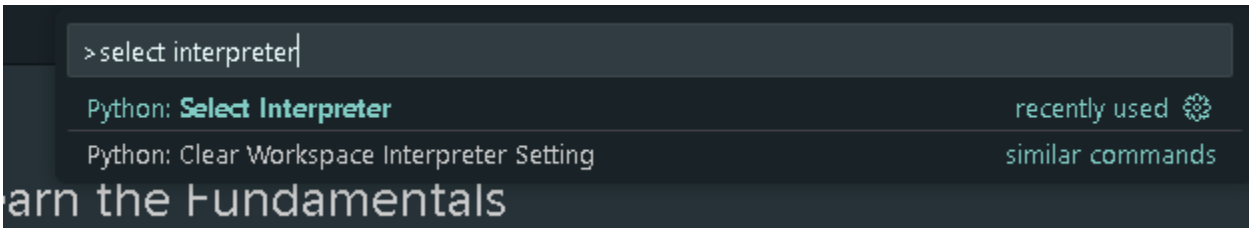
Type another command in the terminal “.venv\Scripts\activate”. This will activate the virtual environment in your machine.

After activating the virtual environment, you may start creating “.gitignore” file.

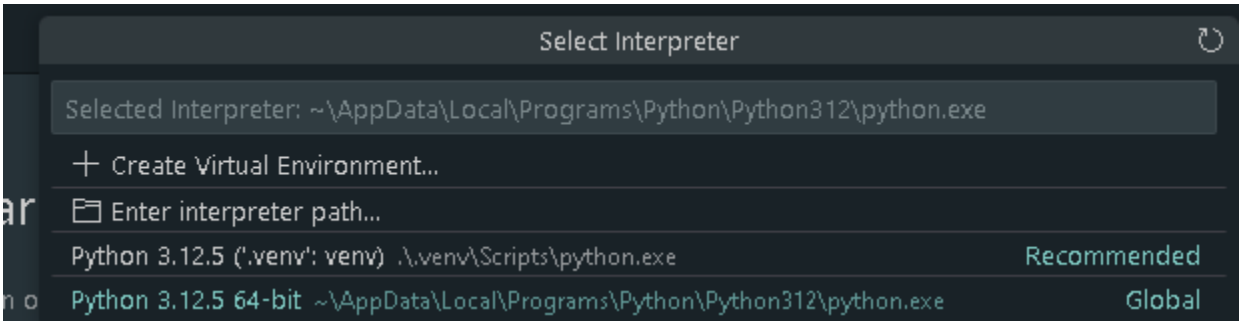
Go inside of .gitignore file and type the name of the virtual environment file you have created, in my case .venv



Click Ctrl + Shift + p, to run commands.



Type “select interpreter”, then click the first one.



Select the Python that has ‘.venv’ ;venv. This is the virtual environment you have created earlier.

```
(.venv)
MAHID DANDAMUN@MahidDandamun MINGW64 E:/Microsoft VS Code (data-processing)
$
```

After selecting the virtual environment, you will notice a (.venv) in your terminal. This indicates that your virtual environment is active.