



Project setup













CONTENTS

1. Repo structure
2. Environments
3. Installing libraries in a given environment.
4. Initializing a Git repository
5. Adding a GitHub repository
6. Git directives
7. Yaml file

Repo structure and files



 data	Final project finished	5 months ago
 notebooks	Final version	5 months ago
 resources/pictures	Filled all Nas and basic cleaning, dealing with the high cardinality now	5 months ago
 slides/tableau public	Final version	5 months ago
 src	Project picture added	3 months ago
 .gitignore	Final version	5 months ago
 README.md	Readme updated	3 months ago
 config.yaml	Final version	5 months ago
 requirements-dev.txt	Final version	5 months ago
 requirements.txt	Final version	5 months ago

Repo files



- Add the following files to your repo:
 - README.md -> Introduce the purpose of your project and provide your findings and next steps to go. Also add a link to your data source.
 - requirements.txt -> a list of libraries used in your project alongside with the specific version for a production environment.
 - requirements-dev.txt -> a list of libraries used in your project alongside with the specific version to be used **without including Jupyter**
 - config.yaml -> yaml file with all the settings used in your project (folders, random_seeds, neighbours,...etc)
 - .gitignore -> include all the files/folders that you don't want to commit.

Repo folders (I)

- Add the following folders to your repo:
 - data/
 - raw/data.csv (original data file)
 - cleaned/data_cleaned.csv (cleaned file)
 - slides/
 - Add here any presentation/tableau url

Repo folders (II)



- **Only if you create a machine learning model**, add the following folders to your repo:
 - transformers/
 - Store here any fitted transformer
 - scalers/
 - Store here any scaler
 - encoders/
 - Store here any encoder
 - models/
 - Store here all your models

Repo folders (III)



- Add the following folders to your repo:
 - notebooks/
 - Exploratory notebook -> Here you do all your tests (include Markdown cells with comments about what and why you do what you do)
 - [Clean_notebook]
- If you create a model, you can optionally create this ones:
 - src/
 - main.py -> Python file to load, clean and train, and save your models
 - lib/
 - functions.py

Repo folders (IV)



- functions.py:
 - Add any import needed by the functions OUTSIDE THE FUNCTION
 - Add all the functions
- Add the following line at the end of your “main.py” file:

```
if __name__ == "__main__":
```

```
    function1()
```

```
    function2()
```

- This allows you to run the enclosed code from:
 - jupyter notebook
 - the terminal.

Repo folders (V)



- If you create an app, add the following folders to your repo:
 - app/
 - Any Python file to create a frontend.

Environments

Environment



- An environment is an isolated space in your computer to install specify versions of your libraries.
- By default, all the anaconda installations comes with one default environment called “base” in which all the libraries are installed.
- When we open a new terminal, by default it will be **opened in the default conda “base” enviroment!**
- **Create an environment in your mid-bootcamp project folder!**

Environment tools



- The two main environment tools available in Python are:
 - Conda
 - Anaconda can only “see” “conda” environments.
 - **Every notebook opened with Anaconda is launched in the conda “base” environment.**
 - Venv
 - It can be seen from VSCode, PyCharm, and others but not with Anaconda.
 - Others....

Conda environments

How to create an environment using conda

- `conda create --name environment_name pip python jupyter`
 - This will create a folder for the environment called “conda-env” in your current folder. In addition, it will install “pip”, “python” and “jupyter” in the environment.
 - Afterwards, it’s interesting to install another library called “pip-tools” which is only available on the Python official repositories, not in conda. For this reason, we need to install it using “pip”
 - `pip install -U pip pip-tools`

Activating/deactivating conda environments

- To activate a conda environment, move to the project folder and type
 - `conda activate`
- To deactivate a conda environment, move to the project folder and type:
 - `deactivate`

Installing libraries in conda environments

- To install libraries in conda environments:
 - Move to the project folder
 - Make sure that the environment is activated
 - Type:
 - `conda install library_name`
 - `conda install library_name=version`

Venv environments

How to create an environment using venv

- If you're in:
 - Linux/MacOS, open a Terminal
 - Windows, open an Anaconda Powershell Prompt
- Navigate to the project folder and Type:
 - `python -m venv ./environment_name`
- Replace “environment_name” by any other name you want to give to your mid_project environment. This will create a folder for the environment called “environment_name” in your current project folder

Activating/deactivating venv environments

- Windows:
 - To activate the environment, move to the project folder and type:
 - `environment_name\Scripts\activate`
 - To deactivate the environment type:
 - `deactivate`
- Linux/MacOS:
 - To activate the environment, move to the project folder and type:
 - `source ./environment_name/bin/activate`
 - To deactivate the environment type:
 - `deactivate`

Installing libraries in venv environments

- Move to the project folder
- Make sure to activate the project environment
- Type:
 - `python -m pip install --upgrade pip`
- After this, and in order to install any library type (don't install any library, **only the ones you need**):
 - `pip install library_name`
- If you want to install an specific version of the library use the syntax shown below.
 - `pip install library_name==version`
- Make sure that the new environment has been activated, and install “pip-tools”, “Jupyter”, “ipykernel”, Type:
 - `pip install pip-tools jupyter ipykernel`

Environments and Jupyter

Opening Jupyter notebooks in an environment

- By default, Jupyter is not aware of new environments created externally.
- We need to install a plugin to be able to open Jupyter notebooks in the new environments.
- Therefore, we need to tell jupyter about the existing environments. We can do this by typing in the terminal:
 - `python -m ipykernel install --user --name=environment_name`
- Afterwards we will need to close Jupyter and open it again to be able to select in which environment we want to launch the new notebook.
- To launch a new jupyter notebook, open the terminal, move to the project folder, activate the environment, and type:

- `jupyter notebook`

Requirements files

What is and why we want it?

- A requirements file is a file that contains all the libraries needed to execute our programs, alongside with their versions.
- This file is needed because sometimes we're making use of some library functions not available in earlier versions. Therefore, our code will depend on which libraries we're using and the corresponding versions.
- In other words, having the same library installed in a third party system doesn't give you any warranty that third parties can execute the repository code without errors.
- Third parties need to know which libraries we were using in our project.

Generate a requirements using conda

- Conda uses yaml files to store all the information related with a given environment.
- Open the terminal
- Activate the conda environment
- Type:
 - `conda env export --file requirements.yml`

Generate requirements using piptools

- Open the terminal
- Activate the venv environment
- Create a file named: “requirements-dev.in” with Sublime.
 - This file must contain a full list of all the library names used in your **python programs**
- Create a file named: requirements.in with Sublime.
 - Add the line below:
 - `-c requirements-dev.txt`
 - Add any other library name not listed in the previous file like “jupyter”
- To generate the requirements.txt and requirements-dev.txt, go to the project folder, make sure that you’re in the correct environment, and type:
 - `python -m piptools compile requirements-dev.in`
 - `python -m piptools compile requirements.in`
- The previous commands will generate a new set of files (requirements.txt and requirements-dev.txt) in the current folder with the versions of all the libraries listed in the *.in files.

Initializing a Git repository

Initializing a git repository

- Move to your project folder and type:
 - `git init`

Adding a GitHub repository



Initializing a GitHub repository and linking it

- Go to your GitHub page and create a new repository.
- Grab the url of your new GitHub repository
- Open a terminal window into your project folder and type:
 - `git remote add origin remote_url`

Git steps

Mandatory (I)

- **At the end of every day:**

- If in your project folder you have file bigger than 20MB.

- Option A:

- Skip Tableau workbooks and only include a Markdown file with the link to your public repo.
- Compress data files bigger than 20MB
- Add a .gitignore file and place inside any files/folders that you don't want to commit.

- Option B:

- Use <https://git-lfs.github.com/>

Mandatory (I)

- **At the end of every day:**
 - Add the files to the stack
 - Create a commit
 - Push your changes to your repo.

Config.yaml file

Yaml file

- Is a configuration plain text file.
- The purpose of this file is to host all the parameters needed by your notebooks/codes, ie:
 - Folders of data, transformers, scalers, models,...etc
 - Random seed
 - Hyperparameters of each of your models
- It has a hierarchical structure
 - section:
 - Parameter: value

Example Yaml file

```
1  data:
2    info: '../data/clean_data/info.csv'
3    orders: '../data/raw_data/orders.csv'
4    orders_prior: '../data/raw_data/order_products__prior.csv'
5    instacart_sample: '../data/clean_data/instacart_sample.csv'
6    instacart_labels: '../data/clean_data/instacart_labels.csv'
7
8  scalers:
9    minmax: '../scalers/minmaxscaler.pickle'
10
11 encoders:
12   ohe: '../encoders/ohe.pickle'
13
14 models:
15   kmeans_randomstate: 1234
16   kmeans_path: '../models/kmeans_scaled_'
17   kmeans_path_k: '../models/kmeans_scaled_3.pickle'
```

Reading a Yaml file in Python

```
import pandas as pd
import yaml
```

```
try:
    with open ("../params.yaml", 'r') as file:
        config = yaml.safe_load(file)
except Exception as e:
    print('Error reading the config file')
```

Saving models, scalars and transformers

Set up

Mandatory (I)

- Go to your project folder and create three new folders with the following names: *models*, *scalers*, *transformers* (if you haven't done it yet)
- Each of this folder will be used to stored it's corresponding objects.

Saving a model, scaler or transformer

- import pickle:
 - import pickle
 - Pickle is a tool that can be used on any kind of object. Although at this time we are going to use it to save **models, scalers or transformers**.
 - *with open("../folder path/model.pkl", "wb") as file:*

```
pickle.dump(model, file)
```

 - This will save the model, scaler or transformer inside the corresponding folder that you should already have created inside your project folder.

Loading a model, scaler or transformer

- import pickle:

- `pickled_model = pickle.load(open("model.pkl", "rb"))`

- `pickled_model.predict(X_test)`

- We pass the saved object into a pickle ***load()*** function. If you assign it to a variable you will be able to use the ***.predict()*** function again to pass test data and get predictions on it.