Data Science blueprint

Release 1.0

May 07, 2020

Presentation:

CHAPTER 1

Introduction

Data science blueprint is a nothing more than a data science project structure proposition. Its purpose is very simple :

- help you build a robust project
- help you gain some confidence regarding the reliability of the code
- help you meet the requirements needed to deploy in a production environment
- Bring data scientists and machine learning engineers to work together around a common code base

To reach this goal, the blueprint proposes some interesting features that will be covered in this documentation :

- a personalized backbone for your data science project, thanks to cookiecutter
- a dockerized environment that you can use to work with notebooks
- a code quality focus, with the set of tools that will help you profiling and testing your code
- a set of tools to let you use your project as an app that you can deploy on a production server, or on a remote python

repository like Pypi

The Data Science blueprint works with Python projects, with fully packaged code and also Jupyter notebooks. It is particularly adapted to data science projects.

CHAPTER 2

Blueprint's structure

The blueprint file structure follows the following pattern. Let's consider that after installing the project with cookiecutter, you have decided to call it `awesome_project`

```
├── data                <- The data folder used by default by the dockerized␣
↪environment
│
├── docker              <- docker files injections used to build the environment
│
├── docs                <- Sphinx documentation of the project
│
├── notebooks           <- Jupyter notebooks used by the dockerized environment
│
├── awesome_project     <- Source code location of this project.
│   ├── __init__.py     <- Makes it a Python module
│   │
│   ├── commands        <- Command scripts that will allow you to use your project␣
↪with command lines
│   │   └── cmd.py
│   │
│   ├── features        <- Contains the scripts that will handle feature explorations␣
↪and engineering
│   │
│   ├── models          <- Contains the scripts that will handle the models
│   │
│   ├── operator        <- Contains the scripts that will handle data processing
│   │   ├── data.py
│   │   ├── dataframe.py
│   │   └── generator.py
│   │
│   └── tests           <- Contains the unit tests
│       └── operator
│           ├── data.py
│           ├── dataframe.py
│           └── generator.py
│
```

(continues on next page)

(continued from previous page)

```
│
├── Dockerfile          <- Build file used by Docker to build the project environment
│
├── LICENSE
│
├── Makefile            <- Makefile to run commands over the project
│
├── README.md
│
├── requirements.txt    <- The requirements file that lists all the package␣
↪dependencies to install
│
├── setup.cfg           <- Configuration file used to fill the setup.py file
│
├── setup.py            <- Setup file to install the project as a pip package
│
└── .gitignore          <- Avoid pushing tmp files that should never exist on Github
```

As you can see in the file tree, some python files have been added (particularly in the awesome_project folder). You must keep in mind that those files have been added only for demonstration purposes. Their purpose is to help you understand how the unit tests are organized, how the project code should be organized.

Note: After you have installed the blueprint, and you have got to know how to use it, you will indeed remove the unit tests, the operator files, and write your owns.

CHAPTER 3

Make this your own

The very purpose of this blueprint is that you make it your own. That said, you are allowed to bring any modifications you want to the blueprint. You can fork it, change any part you want, use it for commercial or non commercial use, and even republish it under your own brand if you wish to.

Feel free to give me your feedbacks. With your feedbacks, I can design some new features. I will integrate them into the blueprint, and update the documentation. As I have said before, the project is available on github, here. You can also easily clone it like this:

```
git clone https://github.com/mysketches/data-science-blueprint.git
```

If you wish to contribute to my project, I'd be glad to receive your pull requests on Github. If you see a bug, or something os not clear, you can also create an issue on Github.

CHAPTER 4

Installation

Installation of the blueprint will proceed as follow:

```
$ pip install cookiecutter
$ cookiecutter https://github.com/mysketches/data-science-blueprint
```

This should output the following form:

```
full_name [John Doe]:
email [john.doe@myemail.org]:
project_name [DS-blueprint]:
package_name [Awesome project]:
package_slug [awesome_project]:
project_short_description [No description]:
version [0.1.0]:
Select python_interpreter:
1 - python3
2 - python
Choose from 1, 2 [1]:
application_cli_name [awesome-project-cli]:
Select opensource_licence:
1 - MIT
2 - BSD
3 - ISCL
4 - Apache Software License 2.0
5 - Not open source
Choose from 1, 2, 3, 4, 5 [1]:
docker_base_image [jupyter/base-notebook:python-3.7.6]:
docker_image_name [awesome-project-env]:
docker_container_name [awesome-project-env-instance]:
docker_container_port [8888]: 8082
```

Leave the fields blank if you wish to use the default values.

That's it, your blueprint is installed, and ready to work !

CHAPTER 5

Environment

5.1 Build the environment

After the blueprint is installed, most of what you will do with your environment will work through the `make` command. You can build your environment like this:

```
$ make environment-build
```

You should see the following output:

```
Successfully built c70c2ec13421
Successfully tagged awesome-project-env:latest

Environment is built! A Docker image was created: awesome-project-env
```

Let's add some sample data to our project:

```
$ make environment-data
```

You can now start your environment:

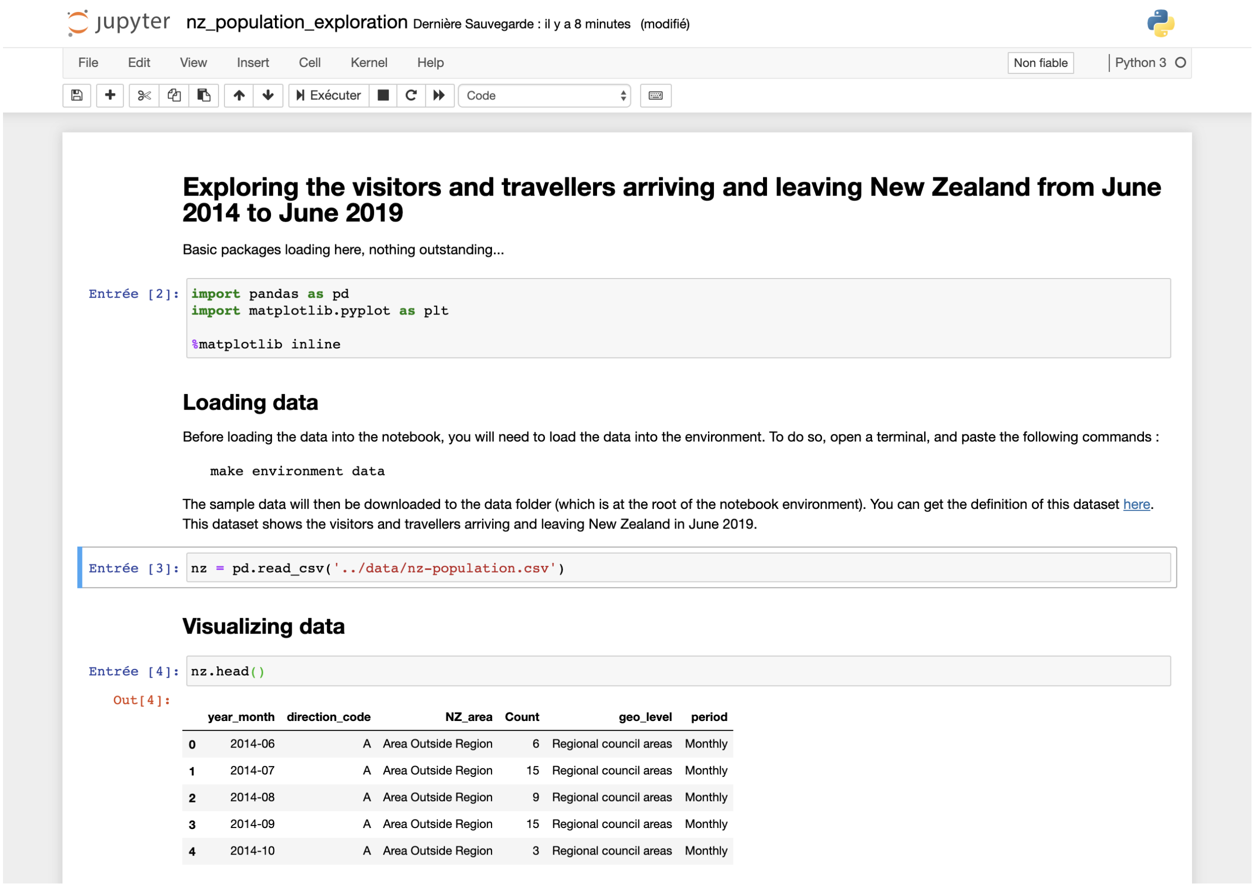
```
$ make environment-start
```

You should see the following output:

```
05f32052868110f38e8233a2ac70ebff076272c7997f5409a8d70a780fc33ec7
Environment is running!
Notebooks interface is available at http://localhost:8082
```

5.2 Use the Jupyter interface

You can now reach out to your environment interface, and access the Jupyter notebook instance. A notebook has been created with the blueprint, to give you a sample exploration. It is located in the notebooks folder. After displaying it, you can run the cells.


jupyter
nz_population_exploration Dernière Sauvegarde : il y a 8 minutes (modifié)
File Edit View Insert Cell Kernel Help
Non fiable
Python 3
Exécuter
Code

Exploring the visitors and travellers arriving and leaving New Zealand from June 2014 to June 2019

Basic packages loading here, nothing outstanding...

Entrée [2]:
```
import pandas as pd
import matplotlib.pyplot as plt

%matplotlib inline
```

Loading data

Before loading the data into the notebook, you will need to load the data into the environment. To do so, open a terminal, and paste the following commands :

```
make environment data
```

The sample data will then be downloaded to the data folder (which is at the root of the notebook environment). You can get the definition of this dataset here. This dataset shows the visitors and travellers arriving and leaving New Zealand in June 2019.

Entrée [3]:
```
nz = pd.read_csv('../data/nz-population.csv')
```

Visualizing data

Entrée [4]:
```
nz.head()
```

Out[4]:

	year_month	direction_code	NZ_area	Count	geo_level	period
0	2014-06	A	Area Outside Region	6	Regional council areas	Monthly
1	2014-07	A	Area Outside Region	15	Regional council areas	Monthly
2	2014-08	A	Area Outside Region	9	Regional council areas	Monthly
3	2014-09	A	Area Outside Region	15	Regional council areas	Monthly
4	2014-10	A	Area Outside Region	3	Regional council areas	Monthly

5.3 Work with the shell

Back to your terminal, you can now launch a shell:

```
$ make environment-shell
```

You are now in the shell, and you will notice that the prompt has changed:

```
jovyan@awesome_project:~$
```

Move to the scripts folder, and launch a python script that tests your package:

```
jovyan@awesome_project:~$ cd scripts
jovyan@awesome_project:~/scripts$ python3 nz-mapping.py
```

This is the output you should expect:

```
             NZ_area  Count             geo_level
0  Area Outside Region      6  Regional council areas
1  Area Outside Region     15  Regional council areas
2  Area Outside Region      9  Regional council areas
3  Area Outside Region     15  Regional council areas
4  Area Outside Region      3  Regional council areas


             NZ_area  Count             geo_level
0  area outside region      6  REGIONAL COUNCIL AREAS
1  area outside region     15  REGIONAL COUNCIL AREAS
2  area outside region      9  REGIONAL COUNCIL AREAS
3  area outside region     15  REGIONAL COUNCIL AREAS
4  area outside region      3  REGIONAL COUNCIL AREAS
```

CHAPTER 6

Quality testing

You can check that the your project meets the Python coding conventions:

```
$ make lint
```

If you try this command with the fresh installation of the blueprint, nothing should output this command as the code written in the blueprint respects the conventions.

Let's run unit tests now:

```
$ make test
```

This is the output you should get:

```
=================================== test session starts ==============================
platform linux -- Python 3.7.6, pytest-5.4.1, py-1.8.1, pluggy-0.13.1
rootdir: /data
collected 4 items

awesome_project/tests/operator/test_data.py .                                 [ 25%]
awesome_project/tests/operator/test_dataframe.py ..                           [ 75%]
awesome_project/tests/operator/test_generator.py .                            [100%]

=================================== 4 passed in 0.84s ================================
```

Finally, you can launch a coverage test over your project:

```
$ make coverage
```

This is the output you should get:

```
Name                                                    Stmts   Miss  Cover   Missing
--------------------------------------------------------------------------------
awesome_project/__init__.py                                 4      0   100%
awesome_project/operator/data.py                            6      0   100%
```

(continues on next page)

(continued from previous page)

```
awesome_project/operator/dataframe.py                  6      0   100%
awesome_project/operator/generator.py                  4      0   100%
awesome_project/tests/operator/test_data.py            6      0   100%
awesome_project/tests/operator/test_dataframe.py      11      0   100%
awesome_project/tests/operator/test_generator.py       3      0   100%
--------------------------------------------------------------------------------
TOTAL                                                 40      0   100%
```

CHAPTER 7

Using the cli interface

As the project is packaged in the environment, it is possible to use to run your project command lines. Open the environment shell:

```
$make environment-shell
```

We will use a sample command written for the sake of the blueprint. It just generates a sequence of text. You can call it like this:

```
jovyan@awesome_project:~$ awesome-project-cli
```

As you didn't write any action to perform, here is the obvious output you'll get:

```
Usage:
    awesome-project-cli lorem <iterations> [--text-size=<text_size>]
    awesome-project-cli data <data-url> <data-location>
    awesome-project-cli (-h | --help)
```

Let's use the lorem action. It generates a list of *lorem ipsum* text:

```
jovyan@awesome_project:~$ awesome-project-cli lorem 10 --text-size=50
```

Here is the final output:

```
['Lorem ipsum dolor sit amet, consectetur adipiscing',
'Lorem ipsum dolor sit amet, consectetur adipiscing',
'Lorem ipsum dolor sit amet, consectetur adipiscing',
'Lorem ipsum dolor sit amet, consectetur adipiscing',
'Lorem ipsum dolor sit amet, consectetur adipiscing',
'Lorem ipsum dolor sit amet, consectetur adipiscing',
'Lorem ipsum dolor sit amet, consectetur adipiscing',
'Lorem ipsum dolor sit amet, consectetur adipiscing',
'Lorem ipsum dolor sit amet, consectetur adipiscing',
'Lorem ipsum dolor sit amet, consectetur adipiscing']
```

CHAPTER 8

Requirements

To install the Data Science blueprint and benefit from all its features, a few requirements will have to be checked

- python
- docker
- pip
- git

And that's it. To check your python version, you can type in this instruction in your favorite terminal. Knowing which executable is installed on your machine will help you decide on the python interpreter to choose

```
python --version
python3 --version
```

Regarding Docker, you must be able to run docker commands without being sudo. Therefore, you must be able to check this command on your terminal:

```
docker --version
```

If it appears that you need to be sudo to run docker, then you must add your user account to the docker group that was created after installing docker on your machine. This can be done thanks to this command:

```
sudo usermod -aG docker $USER
```

To interact avec the blueprint, you will need to run the `make` command many times. This command can run definitions that are specified in the `Makefile` file. Depending on the operating system you are installing the blueprint on, you will find much documentation about how to install `make`.

CHAPTER 9

Using cookiecutter

The blueprint will be installed using a great tool called cookiecutter. When launching Cookiecutter, the program will ask for some variables, whose values will configure the blueprint in order to make it **your** project.

Here is the list of the variables that will be set by Cookiecutter

Variable	Default value	Definition
full_name	John Doe	Name of the author / maintainer
email	john.doe@myemail.org	Email of the author / maintainer
project_name	DS project	Name of the folder where the blueprint will be installed
package_name	Awesome project	Name of the project
package_slug	awesome.project	Formatted name of the project that will be used with packages
project_short_description	No description	Description of your project
version	1.0.0	Version of the project
application-cli-name	{project_slug}-cli	Unix command to use your packaged project as an app
opensource_licence	"MIT", "BSD", "ISCL", "Apache"…	Licence for your project
docker_base_image	jupyter/base-notebook:python-3.7.6	Docker image used to build the environment
docker_image_name	{package_slug}-env	Name of the built Docker image that will be used as your environment
docker_container_name	{docker_image_name}-instance	Name of the Docker container that will be instanciated
docker_container_port	8888	Port exposed to access Jupyter notebooks

CHAPTER 10

Installation

Installing the blueprint will be quite easy.

First, you need to install cookiecutter, with this command:

```
pip install cookiecutter
```

Now Cookiecutter is installed, you can install the Data Science blueprint:

```
cookiecutter https://github.com/mysketches/data-science-blueprint
```

The blueprint should now be installed in the folder matching the value you set for the cookiecutter variable `project_name`.

After testing the blueprint, I found situations where the cookiecutter would not be recognized. If this happens, you can call it with this alternate command:

```
python -m cookiecutter https://github.com/mysketches/data-science-blueprint
```

CHAPTER 11

Project operations

The Makefile will help you work on the structure of your project. Data science projects are about coding of course, but it is also about all that is going around, such as working on a Jupyter notebooks interface, running unit tests, and even deploying your project.

To work your project beyond coding, all you will need is the `make` command installed on your machine. Depending on your operating system, there are many ways to install `make`.

11.1 Install dependencies

The blueprint comes with a `requirements.txt` file, whose purpose is to list the packages that will be required to work your project.

Here is the list of the requirements that come with the blueprint:

```
coverage
flake8
numpy
pandas
scikit-learn
matplotlib
docopt
configparser
pytest
```

You are indeed encouraged to update this file and make it fit your needs.

To install the dependencies on your machine, just open a terminal, move to the folder where the blueprint is installed, and type in the following command:

```
$ make install
```

11.2 Update dependencies

In order to update the version of your dependencies (because you have specified a specific version of a dependency in the `requirements.txt` file, or because there is a newer latest version of the dependency on pypi), you can open a terminal, move to the folder where the blueprint is installed, and type in the following command:

```
$ make upgrade
```

11.3 Install your project as a package on your machine

You can package your project, and install it as a python package on your machine. Once packaged, your project is part of the Python path, and can be called in another project or anywhere else on your machine.

In particular, you could also imagine packaging your project in a Docker image, and deploy it in the Cloud. You could then spawn resources and call entry points to run your data science project.

To package your project on your machine, just open a terminal, move to the folder where the blueprint is installed, and type in the following command:

```
$ make package
```

After this command is complete, you can import your project in any python script that would be running on your machine.

11.4 Build your environment

Your work environment will be built with Docker. Therefore, in order to create this environment, you will have to build a Docker image.

To build your environment, just open a terminal, move to the folder where the blueprint is installed, and type in the following command:

```
$ make environment-build
```

11.5 Start your environment

Starting your environment will allow you to access your shell, and Jupyter notebooks. Once the environment is started, you can immediately use your browser to connect to Jupyter notebooks on your localhost.

To start your environment, just open a terminal, move to the folder where the blueprint is installed, and type in the following command:

```
$ make environment-start
```

11.6 Stop your environment

To stop your environment, just open a terminal, move to the folder where the blueprint is installed, and type in the following command:

```
$ make environment-stop
```

11.7 Restart your environment

To restart your environment, just open a terminal, move to the folder where the blueprint is installed, and type in the following command:

```
$ make environment-restart
```

11.8 Get your environment status

To check whether your environment is running, and where you can access your Notebooks interface, just open a terminal, move to the folder where the blueprint is installed, and type in the following command:

```
$ make environment-status
```

11.9 Access your environment shell

You can load a shell in your environment. This will allow you call your project package through command lines, and also test your code. To enter your shell, just open a terminal, move to the folder where the blueprint is installed, and type in the following command:

```
$ make environment-shell
```

11.10 Load the sample data

The blueprint comes with some sample data to load into your project. This command will only be useful for the sake of the tutorial. You may update this command according to your needs.

The data that will be loaded into your project is a data sample of the visitors and travellers arriving in New Zealand from June 2014 to June 2019. To load this data, just open a terminal, move to the folder where the blueprint is installed, and type in the following command:

```
$ make environment-data
```

11.11 Clean your project

This command will remove from your project all the temporary files, in particular those that shall never be pushed on Github. If you identify other temp files to add, feel free to update the Makefile.

To clean your project, just open a terminal, move to the folder where the blueprint is installed, and type in the following command:

```
$ make clean
```

11.12 Test your project

To run unit tests to your project, just open a terminal, move to the folder where the blueprint is installed, and type in the following command:

```
$ make test
```

11.13 Visualize the tests coverage for your project

After running your unit tests, you can use visualization of the tests coverage. To do so, just open a terminal, move to the folder where the blueprint is installed, and type in the following command:

```
$ make coverage
```

11.14 Check the code quality of your project

The Data Science blueprint uses Flake8 to test the code quality of the project. To output the coding styles tests that wouldn't pass, just open a terminal, move to the folder where the blueprint is installed, and type in the following command:

```
$ make lint
```

CHAPTER 12

Make the installation your own

12.1 Updating cookiecutter template

The cookiecutter template is written in the file `cookiecutter.json`. You are free to update the file and add (or remove) the parameters that you need. Keep in mind that if you update this file, you will need to crawl the blueprint code in order to integrate the new parameters.

If you want more information about how to update the `cookiecutter.json` file, you can take a look at the cookiecutter documentation

12.2 Updating the Makefile

If you want to add new commands, or update existing commands, you are free to open the file `Makefile`, and update it according to your needs.

CHAPTER 13

Commands

Your work environment will be built with Docker. Therefore, in order to create this environment, you will have to build a Docker image.

To build your environment, just open a terminal, move to the folder where the blueprint is installed, and type in the following command:

```
$ make environment-build
```

13.1 Start your environment

Starting your environment will allow you to access your shell, and Jupyter notebooks. Once the environment is started, you can immediately use your browser to connect to Jupyter notebooks on your localhost.

To start your environment, just open a terminal, move to the folder where the blueprint is installed, and type in the following command:

```
$ make environment-start
```

13.2 Stop your environment

To stop your environment, just open a terminal, move to the folder where the blueprint is installed, and type in the following command:

```
$ make environment-stop
```

13.3 Restart your environment

To restart your environment, just open a terminal, move to the folder where the blueprint is installed, and type in the following command:

```
$ make environment-restart
```

13.4 Get your environment status

To check whether your environment is running, and where you can access your Notebooks interface, just open a terminal, move to the folder where the blueprint is installed, and type in the following command:

```
$ make environment-status
```

13.5 Access your environment shell

You can load a shell in your environment. This will allow you call your project package through command lines, and also test your code. To enter your shell, just open a terminal, move to the folder where the blueprint is installed, and type in the following command:

```
$ make environment-shell
```

13.6 Load the sample data

The blueprint comes with some sample data to load into your project. This command will only be useful for the sake of the tutorial. You may update this command according to your needs.

The data that will be loaded into your project is a data sample of the visitors and travellers arriving in New Zealand from June 2014 to June 2019. To load this data, just open a terminal, move to the folder where the blueprint is installed, and type in the following command:

```
$ make environment-data
```

13.7 Clean your project

This command will remove from your project all the temporary files, in particular those that shall never be pushed on Github. If you identify other temp files to add, feel free to update the Makefile.

To clean your project, just open a terminal, move to the folder where the blueprint is installed, and type in the following command:

CHAPTER 14

Build & Start

The blueprint's environment runs with Docker. Therefore, before using it, you must build it. All is done thanks to the `Dockerfile` file that is located at the root of the project.

The content of the Dockerfile is pretty straightforward:

```
FROM jupyter/base-notebook:python-3.7.6

COPY docker/docker-entrypoint.sh /home/
COPY requirements.txt /home/requirements.txt
RUN rm -rf /home/jovyan/work

RUN pip install -r /home/requirements.txt

ENTRYPOINT ["sh", "/home/docker-entrypoint.sh"]
```

The base image used (base-notebook here) can be configured in the `cookiecutter.json`. In this file, you can set the default value for the `docker_base_image` parameter, and you will also be prompted the parameter when installing the blueprint with the `cookiecutter` command.

When the blueprint's environment is built, you can start it with this command:

```
$ make environment-start
```

The output of this command should be:

```
8c512b6e09c1b5551075fb8921b24f08044dad876e102cd0e7f1813943c1816a
Environment is running!
Notebooks interface is available at http://localhost:8888
```

The Jupyter notebook interface is then accessible through port 8888 (by default). It is also possible to configure the default port value in the `cookiecutter.json` file.

All the dependencies listed in the file `requirements.txt` will be loaded in the environment during its build. This means that if you update the file requirements.txt and you want to add those new dependencies to the environment, you will need to build it again.

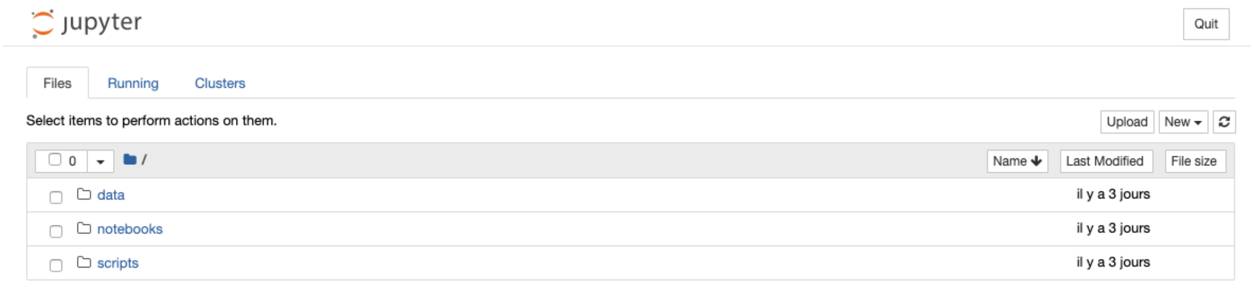
To do so, just launch a new build thanks to the following command:

```
$ make environment-build
```

CHAPTER 15

Notebook

After starting the blueprint's environment, you will have an access to Jupyter notebooks. The default port used for the interface is 8888. You can configure the default port value thanks to the file `cookiecutter.json`.



Select items to perform actions on them.

```
0   /
data        il y a 3 jours
notebooks   il y a 3 jours
scripts     il y a 3 jours
```

15.1 Mapping folders

When the environment is started, it is automatically mapped with your local folders so that in the Jupyter interface, you you will see the following folders:

- `notebooks`, that is mapped with the `notebooks` folder of your project
- `scripts`, that is mapped with the `scripts` folder of your project
- `data`, that is mapped with the `data` folder of your project

That said, you will be able to launche the Juypter interface, create your notebooks and scripts, and theyr will automatically be saved in your project. Therefore, there is no need to export any notebook or Python code. You can safely save your workspace to Github.

15.2 Working with passwords

It is often recommended to secure the Jupyter interface with a password or a token. The purpose of theis blueprint is to provide you with a coding environment. As I expect that you will be the single user of this environment, I have suggested that you wouldn't require any password. Though, it is possible to activate a password to your Jupyter interface.

To activate the password, you can update the default password value (currently set to null) in the file `cookiecutter.json`. When installing the blueprint, you will also be prompted whether or not you wish to use a password for your notebooks. At this point of the installation, you can also set a password.

15.3 Working with your packaged project

Every time you start your environment, your project will be packaged and deployed in the environment. It is then made possible to import your project in your notebooks, and work with as if you had installed a new dependency.

You can then add some code to your project, and live test it in your notebooks. As you update your code, its packaged version in your notebooks will automatically be updated. The only limitation is that Jupyter takes a snapshot of your Python environment status when it loads its workspace. Therefore, if you update your project code, you will need to reload your notebook's workspace, and rerun the cells.

CHAPTER 16

Shell

It is possible to open a shell to your environment. This can be useful if you want to manually install dependencies to your environment, or test a python script. Please note though that the operations that you will apply to your environment will not be persistent. This means that if you manually install dependencies and restart your Docker environment, you will have to install your dependencies again. To make them persistent, you may update the `requirements.txt` file and persist by rebuilding the environment.

To access your environment shell, just use this command:

```
$ make environment-shell
```

This will behave a bit the same as if you were using SSH. Once in the shell, you can simply exit it with a `exit` command.

16.1 Working with your packaged project

Every time you start your environment, your project will be packaged and deployed in the environment. It is then made possible to import your project in your notebooks, and work with as if you had installed a new dependency.

You can then add some code to your project, and live test it with Python scripts. As you update your code, its packaged version in PYTHON_PATH will automatically be updated.

This can be very useful as for instance, it allows you to create some test scripts (you may store them in the script folder in the environment, and they will be mapped with the scripts folder in your project), and test/challenge your code.

If you decide to test your code with Python command in interactive mode, you will need to reload the python cache in your environment, to take into account the code updates. You can also exit the python interactive mode, and reload it again.

CHAPTER 17

Make the environment your own

The blueprint's environment is entirely managed with Docker. Therefore, if you wish to customize the environment, you may update the following files

17.1 Updating Dockerfile

The Dockerfile's content is pretty straightforward:

```
FROM jupyter/base-notebook:python-3.7.6

COPY docker/docker-entrypoint.sh /home/
COPY requirements.txt /home/requirements.txt
RUN rm -rf /home/jovyan/work

RUN pip install -r /home/requirements.txt

ENTRYPOINT ["sh", "/home/docker-entrypoint.sh"]
```

The installation process is explained as follow:

- Load the base docker image for jupyter notebooks
- Copy the starting script
- Copy the python dependencies list
- Install the dependencies
- Map the image starting script with the one we have just added

17.2 Updating docker-entrypoint

The purpose of the docker-entrypoint script is just to prepend the jupyter workspace launch with a package build of your project. Thanks to this docker-entrypoint script, you can use your project as a python package, in your working environment.

The docker-entrypoint file is located here : `docker/docker-entrypoint.sh`

CHAPTER 18

Flake8

I have chosen flake8 package to test coding conventions. Indeed, you are free to use your own, but flake8 is installed by default, and it is working just fine.

18.1 Installation and usage

To work it, you don't have to integrate anything in your code.

Here are the few steps to follow to use flake8 :

Installation (once):

```
$ make install
```

Use:

```
$ make lint
```

18.2 Configure flake8

Flake8 configuration is stored in the file `setup.cfg`. In the `[flake8]` section, you can update a few parameters:

```
[flake8]
max-line-length = 79
max-complexity = 10
filename = ./awesome_project/*.py
```

If you need to get some information about flake8 parameters, you can check these links:

- flake8 options
- flake8 error codes

CHAPTER 19

Unit tests

I ‘ve decided to use pytest package to run unit tests.

To run unit tests over your project, you can use the command:

```
$ make test
```

The unit tests are located in the package’s test folder. I recommend you follow the folder structure of the files that are tested. For instance in the blueprint, the python code that is covered with unit tests is located in the `operators` folder. This is why unit tests are located in the folder `tests/operators`.

The output of the unit test will look like this:

```
platform darwin -- Python 3.7.2, pytest-5.4.1, py-1.8.0, pluggy-0.13.1
collected 4 items

awesome_project/tests/operator/test_data.py .                 [ 25%]
awesome_project/tests/operator/test_dataframe.py ..           [ 75%]
awesome_project/tests/operator/test_generator.py .            [100%]

======================== 4 passed in 0.82s =============================
```

CHAPTER 20

Code coverage

After running unit tests, you can display the code coverage of your project. To show the code coverage, simply use the command:

```
$ make coverage
```

The output of the coverage report should look like this:

```
Name                                              Stmts   Miss  Cover   Missing
-------------------------------------------------------------------------------
awesome_project/__init__.py                           4      0   100%
awesome_project/operator/data.py                      6      0   100%
awesome_project/operator/dataframe.py                 6      0   100%
awesome_project/operator/generator.py                 4      0   100%
awesome_project/tests/operator/test_data.py           6      0   100%
awesome_project/tests/operator/test_dataframe.py     11      0   100%
awesome_project/tests/operator/test_generator.py      3      0   100%
-------------------------------------------------------------------------------
TOTAL                                                40      0   100%
```

CHAPTER 21

Package your project

Packaging your project should be the final step before deploying in your production environment. The purpose of this step is to convert your project to a Python package, that will be installed on your machine.

Once your project is packaged, it can be imported by any Python code that would be running on your machine. This makes your project portable, and exportable in a Docker image, on Pypi repository, or on a remote machine. At this stage of the blueprint, the strategy to deploy on production is yours.

To package your project, just use this command:

```
$ make package
```

It is also the very same command that is used in the blueprint's environment, and allows you to use your packaged project with your notebooks.

CHAPTER 22

Create your app

After your project is deployed, the question of how you will interact with it remains. You can certainly open a Python shell or call a Python command and call a module, but this is will require you to strictly interact with Python, which is not very comfortable, in particular if you wish to deploy in production, and maybe interact with other programs.

This is why the blueprint also onboards command lines, so that you can call some specific modules with a dedicated command line on your machine.

I guess this would be much more convenient to train a model by simply calling it in a shell:

```
mymodel-cli train xgboost datafile.csv output
```

This is just an example to show you that working like will make your deployments and in-production executions much more accessible.

22.1 Create you cli interfaces

You can declare your cli interfaces in the file `setup.py`:

```
entry_points={
        'console_scripts': [
            'awesome-project-cli={}.commands.cmd:main'.format(config['application'][
↪'package'])
        ]
    }
```

For the sake of the blueprint, an example of entry point was written : `awesome-project-cli`. You can access the source code of this command line in the file `commands/cmd.py`. You are free to update the entry points and add your owns as you wish.

In the blueprint's example, one sample call is defined like this:

```
awesome-project-cli lorem 10 --text-size=50
```

CHAPTER 23

Indices and tables

- genindex
- modindex
- search