

**Applications of big data**

**-**

**Project documentation**

**Project’s GitHub repository :** [**https://github.com/cesarombredane/Weather-Classification/tree/main**](https://github.com/cesarombredane/Weather-Classification/tree/main)

**(The instructions on how to run the project are provided in the 'README' file)**

**Meryem Kose**

**Arthur Nguyen**

**César Ombredane**

## Table of contents

[**Table of contents 2**](#_gvu5smyi3irg)

[**Introduction : Datascience industrialization 3**](#_a67u863jtj5l)

[**Requirements 3**](#_g5xxtg51gqr9)

[**Project Setup 3**](#_i22jn8bld5gn)

[**Docker Containerization 4**](#_iirl32tpgvtn)

[**Bonus part 5**](#_rr1isl7m6x44)

[**Conclusion 6**](#_6cxbzqoc77g7)

## 

## Introduction : Datascience industrialization

The chosen project (**subject n°2**), involves containerizing a data science model. Containerization is a lightweight virtualization technique that encapsulates an application and its dependencies in a container, ensuring consistent and isolated execution regardless of the host environment. To achieve this, a set of instructions provided context for containerization usage and outlined the project requirements, including the use of volumes. These instructions guided our software choices.

## Requirements

In order to run the project correctly, it is necessary to have Docker and Docker Compose installed.

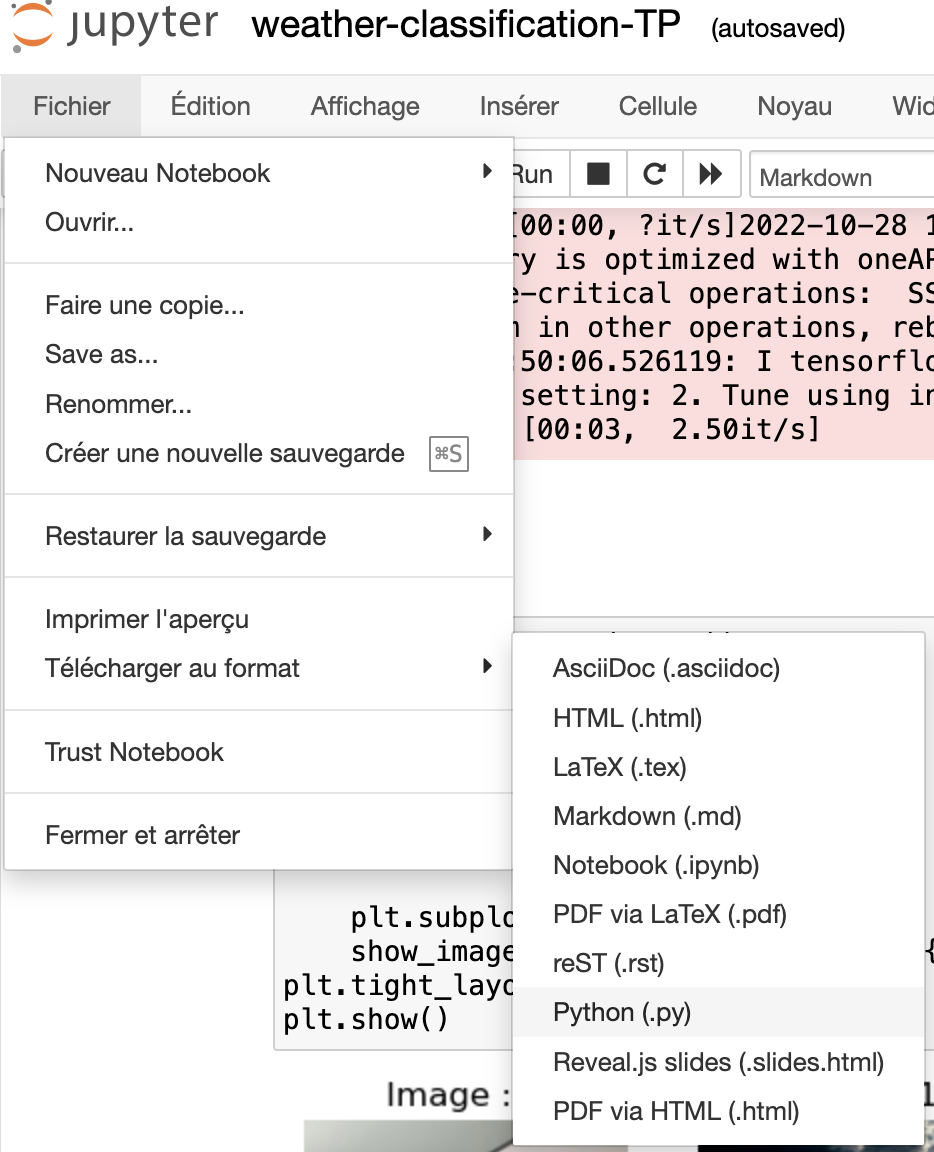
→ Here is a link detailing the installation steps to follow to install Docker : [Overview of Docker Desktop](https://docs.docker.com/desktop/).

→ Here is a link detailing the installation steps to follow to install Docker Compose : [Overview of installing Docker Compose](https://docs.docker.com/compose/install/).

## Project Setup

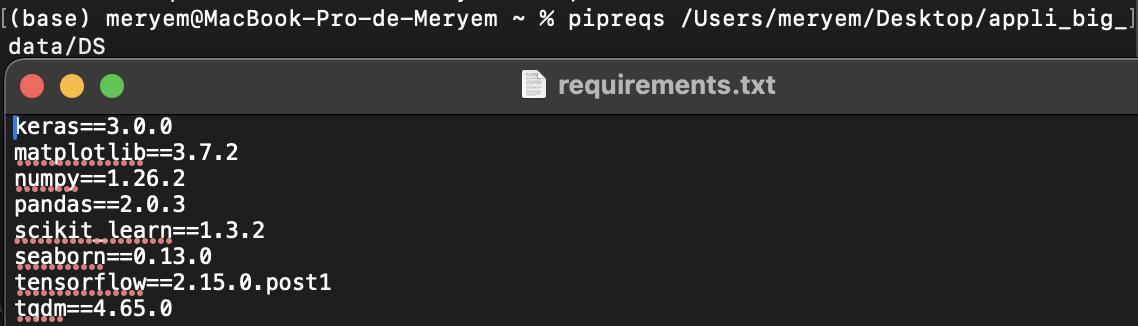
Initially, we downloaded the project folder and loaded it into an environment supporting Jupyter Notebooks, such as Visual Studio Code or Anaconda, based on individual preferences. After importing the file, we gained an understanding of the model and the imported images. It was noted that the images and the model were not in the correct locations. Modifications were made to the folder structure and paths to ensure the script ran correctly.

Once the script was functional in our environments, we exported it to a Python script (.py) from Jupyter Notebook.



Afterward, we automatically generated the required Python packages for our case study script. We used the `pipreqs` command, among other available methods.

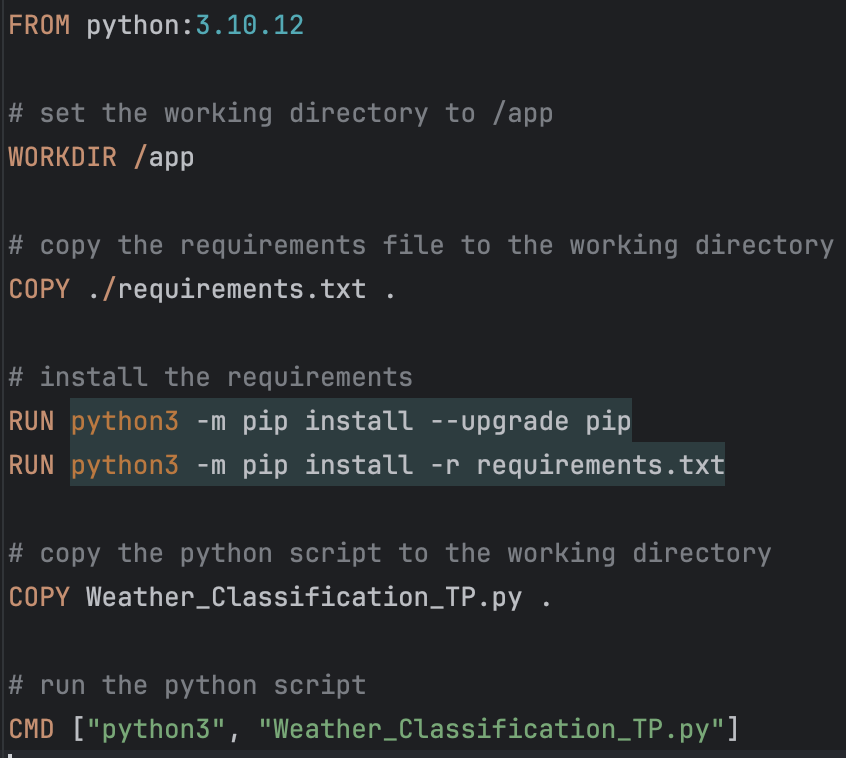
This is the file that was produced :



## Docker Containerization

To containerize the application, a "Dockerfile" was defined at the location of our .py file and requirements.txt. This file includes instructions for configuring the Python environment within the container, such as copying packages from requirements.txt and the model.

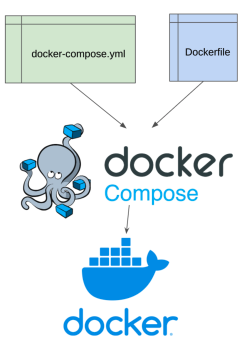
Here is an excerpt from our Dockerfile:

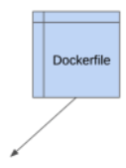


Volumes were not defined in this file. Instead, they are specified in the docker-compose.yml file.

*What is a docker-compose.yml file, and why include input and output volumes in it instead of the Dockerfile?*

The docker-compose.yml file provides a more flexible and service-centric approach, facilitating the coordination of volumes across multiple services. This proves useful in complex scenarios with interdependent containers. The illustration below clarifies this concept:





As seen in the illustration, a single docker-compose.yml file simplifies the management of different Docker containers, especially when dealing with multiple Dockerfiles. It allows direct control over various general configurations, such as input and output volumes, for each Docker container.

*How does it install and work?*

As per the instructions, the Dockerfile should be executed using the "docker run" command. Now, this command becomes "docker compose up." Upon execution, the environment gradually comes together.

## 

*What’s the place of generative AI in our work?*

We resorted to utilizing generative AI when we were confronted with bugs, particularly in the GitHub Actions section, where we encountered significant compatibility issues between TensorFlow and the Python version within the container.

## 

## Bonus part

**1 - Make the Python app avoid predicting already predicted images, but only if explicitly requested.**

Firstly, since we are using Docker Compose, it is not feasible to prompt the user for input immediately upon the 'docker-compose up' command. At that moment, the user doesn’t have visibility into the various images being processed.

Therefore, it seems more reasonable to allow input once Docker has been launched, and the user runs the Python program directly locally. Otherwise, the user would lack sufficient information to determine whether to restart the process for all images or only those that have not been processed

To achieve this, data is categorized into two groups: processed and unprocessed. The selection criterion is based on the presence of one or more files in the CSV file within the "result" folder. If certain image names are missing in the CSV file compared to the "data" folder, unprocessed images are considered. The user is prompted during the program's relaunch to decide whether to re-execute files that have already been processed, in addition to those that haven't. If the user chooses 'yes,' the initially recorded files in the result file will undergo prediction again. Otherwise, only the unprocessed files will be considered and recorded in the new CSV file.

**2 - Add automatic packaging using DevOps Tool: GitHub Action**

We attempted to automate the packaging process using the GitHub Actions tool. However, despite numerous attempts, we consistently encountered an error indicating an inconsistency between the Python version and the TensorFlow version. This is why we were unable to complete this bonus section.

## Conclusion

In conclusion, this project provided practical insights into Docker Compose and GitHub Actions. We believe that studying the model through concrete cases, as demonstrated here, is more engaging than purely theoretical courses.