Python and R for Data Science.

Task 1. Data collection task

For the first part of the assignment we have been asked to create a 1000 rows database using an API. We chose the OMDB API. To create the database (output_db.json and output_db.csv), we used the following functions:

Functions

- `generate_1000_ids()`: this function takes no positional arguments and returns a list of 1000 film ids. Thanks to the OMDB documentation we knew that the ids are composed as follows: every id starts with 'tt' and followed by 7 digits (e.g. the first film as the id tt0000001). To create those ids we first populated an array containing only the digital part, `suffix_list`. The first element is 0000001 and the last one is 0001000. Then we populated a second array, `ids_list` containing the prefix (i.e. tt) followed by the suffix created before.

- `populate_dataframe()`: this function takes one positional argument: a list of ids and returns a pandas dataframe. First, we initialize an empty pandas dataframe called `df`. Second, we loop through the list of ids and for each id we send a request to the server (the OMDB API). We store the response in a variable `response` and transform it into a json object with the `.json()` function. Then we add the response into our dataframe. Finally, we return this dataframe.

- `create_database()`: this function takes no positional arguments and output a *json file*. First, thanks to the two previous functions, we create a `data` variable containing our 1000 rows dataframe. Then, we export it as a json and a csv file. Finally, we print 'Completed!' to show the user that everythings succeeded.

We wrapped the `created_database()` function in a `if __name__ == "__main__":` in order to be able to export each individual function, if needed, without creating a database everytime.

Variable

For this project we needed to instanciate 3 variables. Variables name are written with capital letter to respect pep 8 coding convention (constant variables must be name with capital letters).

- `API_KEY`: this is the personnal API key received from the OMDB website.
- `API_ADDRESS`: this variable store the url needed to connect to the API endpoint.
- `INITIAL_IMDB_ID_PREFIX`: this variable store ids' prefix (i.e. 'tt').

Imports

- `import requests`: we imported the requests library to communicate with the OMDB API. We used mainly the `.get()` method.
- `import pandas as pd`: we imported the pandas library to handle the data retrieved from the API. Pandas allowed us to store the data in a DataFrame which made it easy to query. It also allowed us to export the database as json and csv files.

Virtual Environment

To make the collaboration between group member easier, we decided to create a python virtual environment, `venv`. This allowed us to work with the same environment on each machine (same packages, same version, etc.).

Task 2. Data Analysis

Dataset-1

Structure

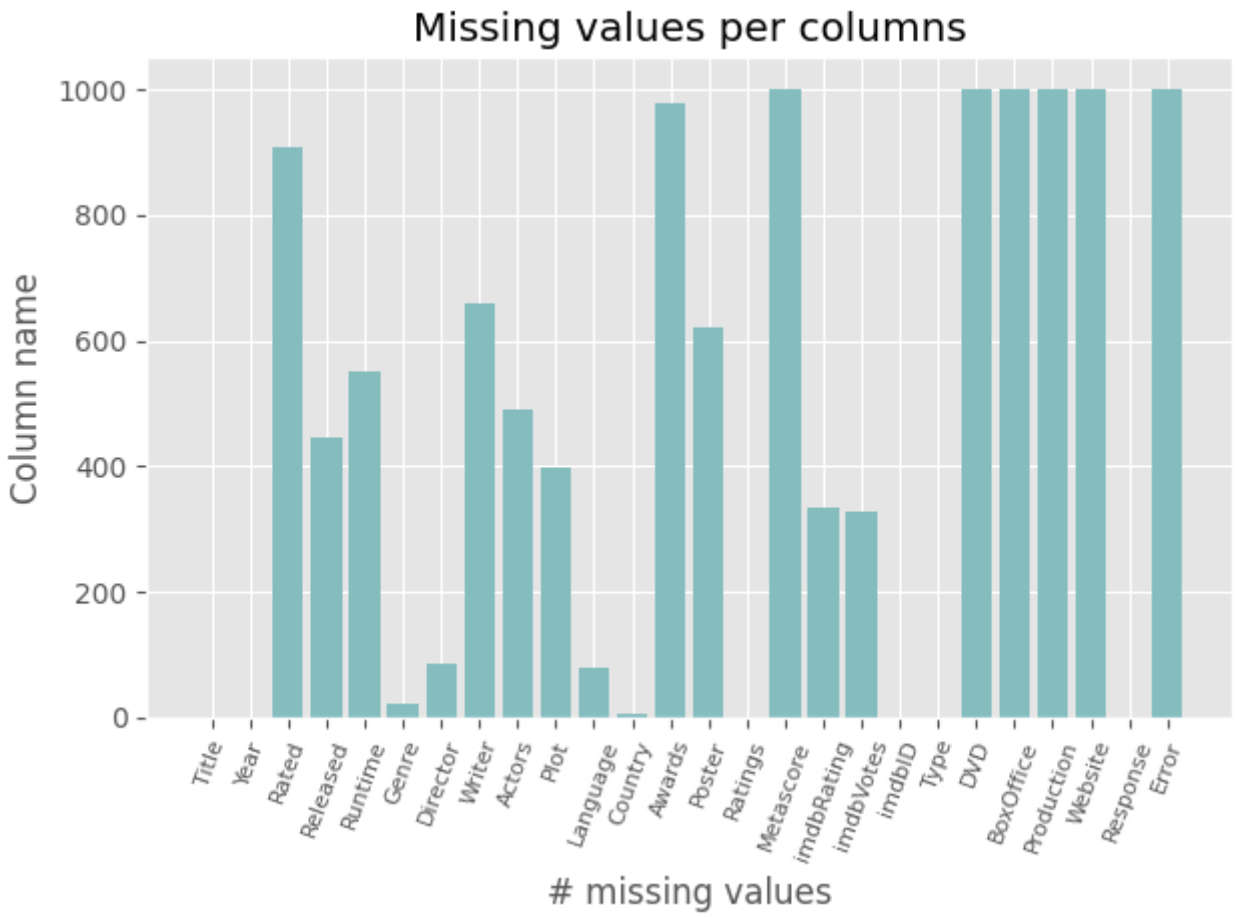
To analyse the data previously collected (see: Task 1), we used two differents files: a python file and a jupyter notebook respectively named graphs.py and data_explorer.ipynb.

Imports

To manipulate the data, we used the well-known `pandas` and `numpy` python library. To plot those data, we used, as asked, the `matplotlib` python library.

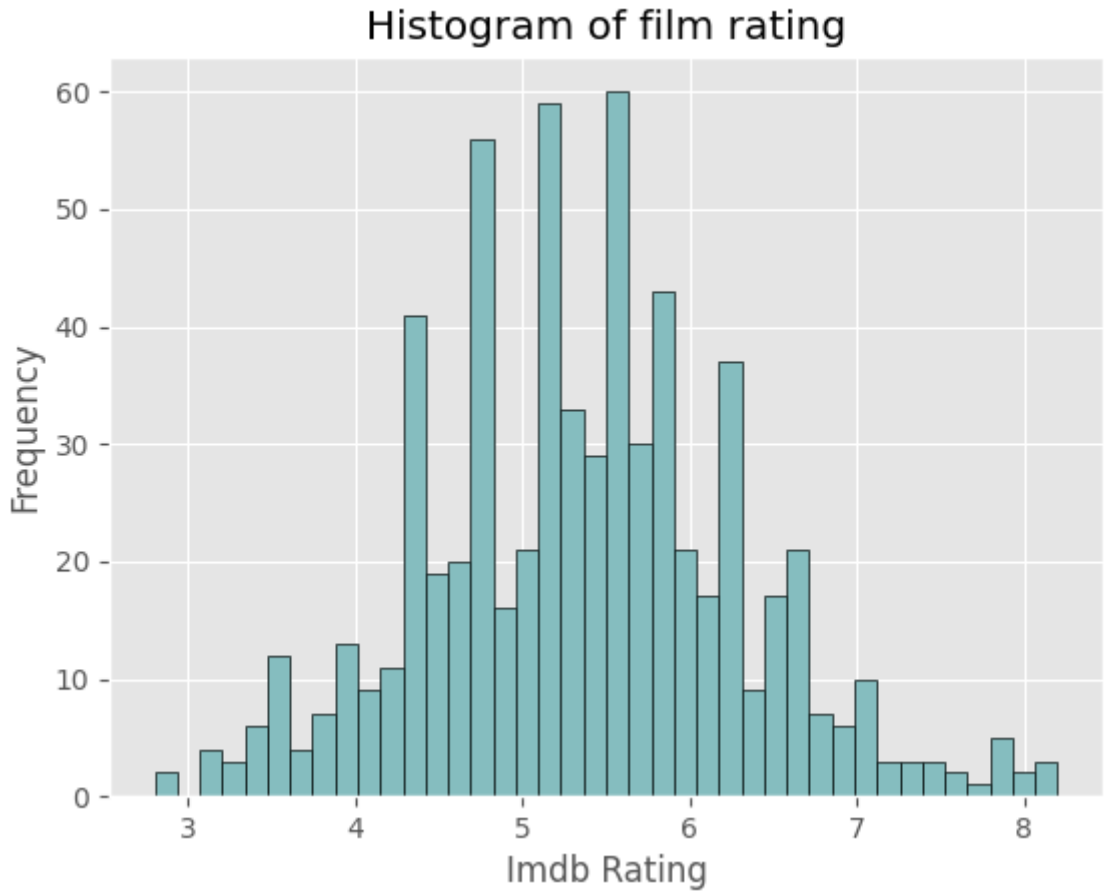
Data Cleaning and Visualisation

Thanks to the pandas' `describe()` method, we quickly realised that we add a lot of missing data in the dataset. We decided to drop the column having missing values and then plot the interesting features one by one.



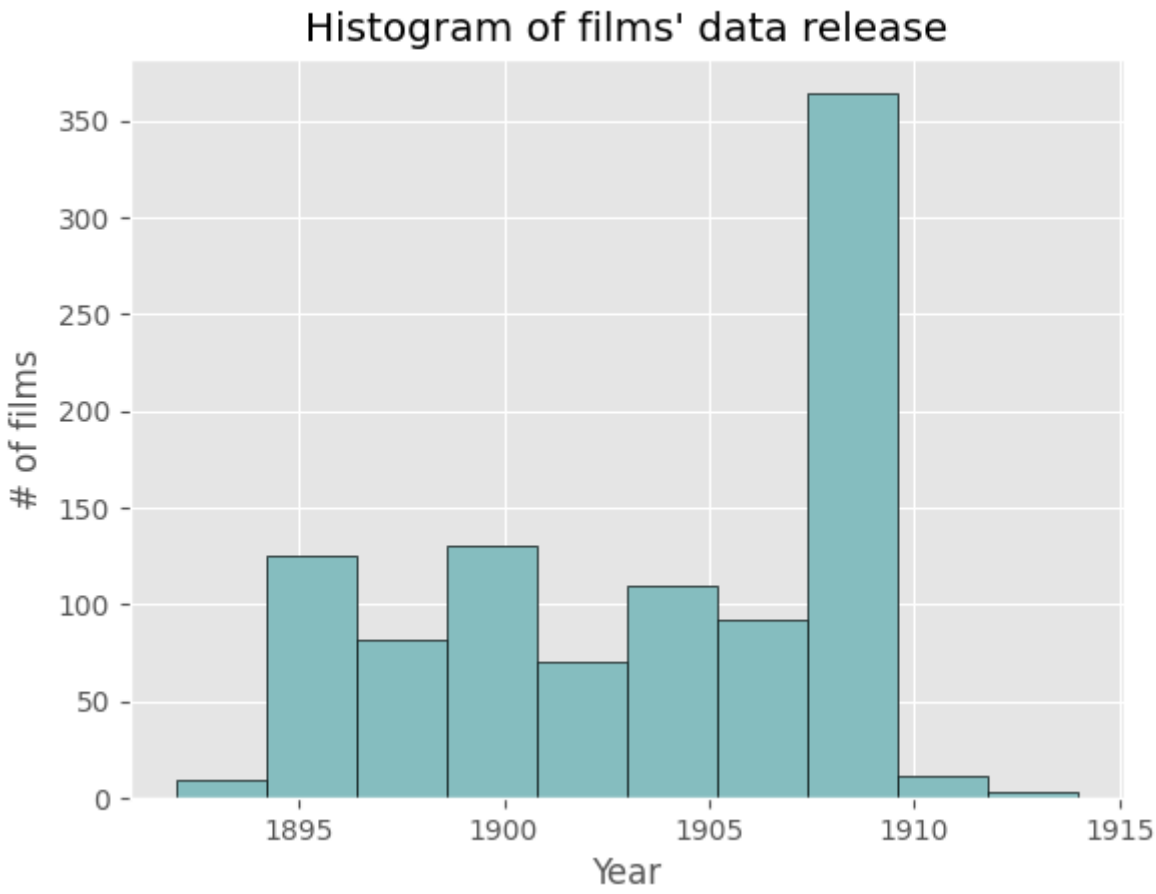
As

you can see in the graph above, for 5 features we had no data at all. That is why we decided to not include them in our analysis. The rating and year feature could be used directly. The only cleaning necessary was to remove the potential missing values.

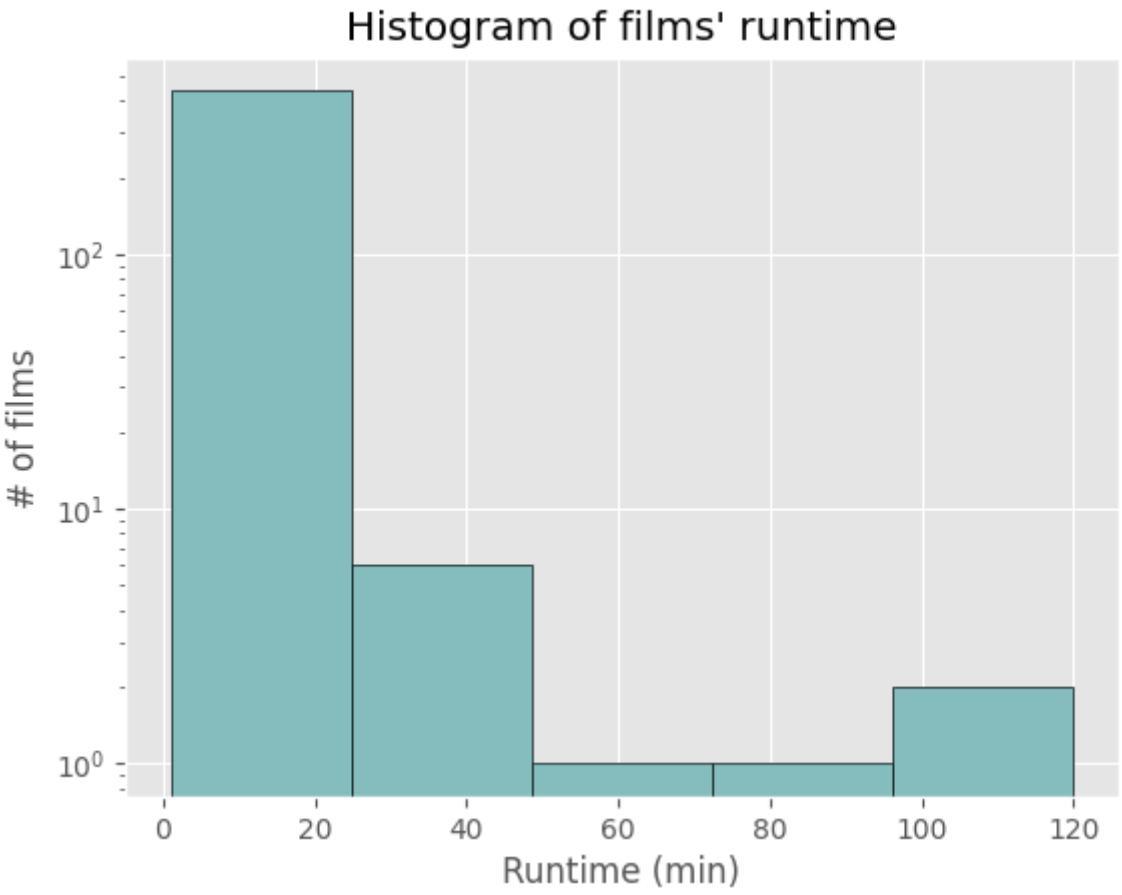


In the graph bellow, we can see that our database contains only films from 1890 until 1915. This is

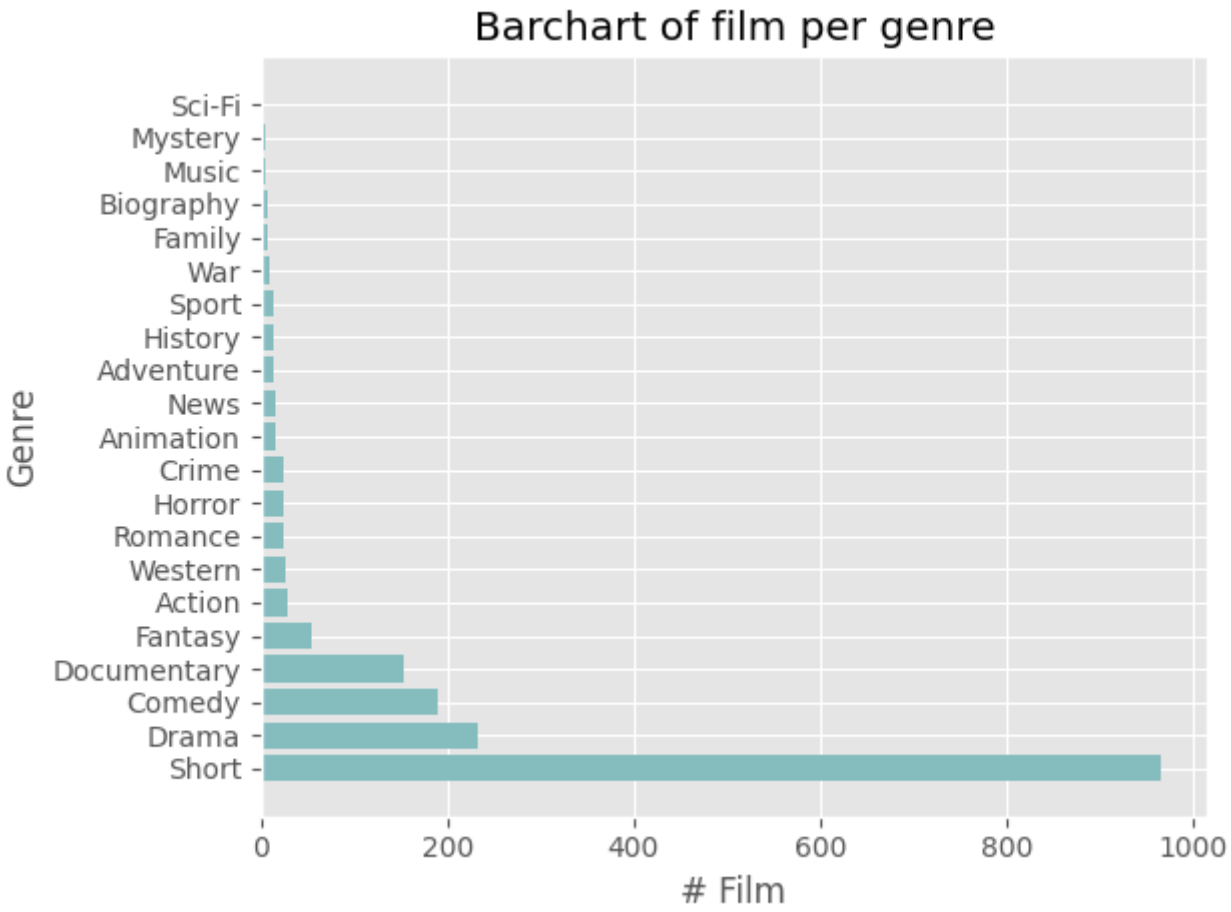
important to keep in mind for the rest of the analysis.



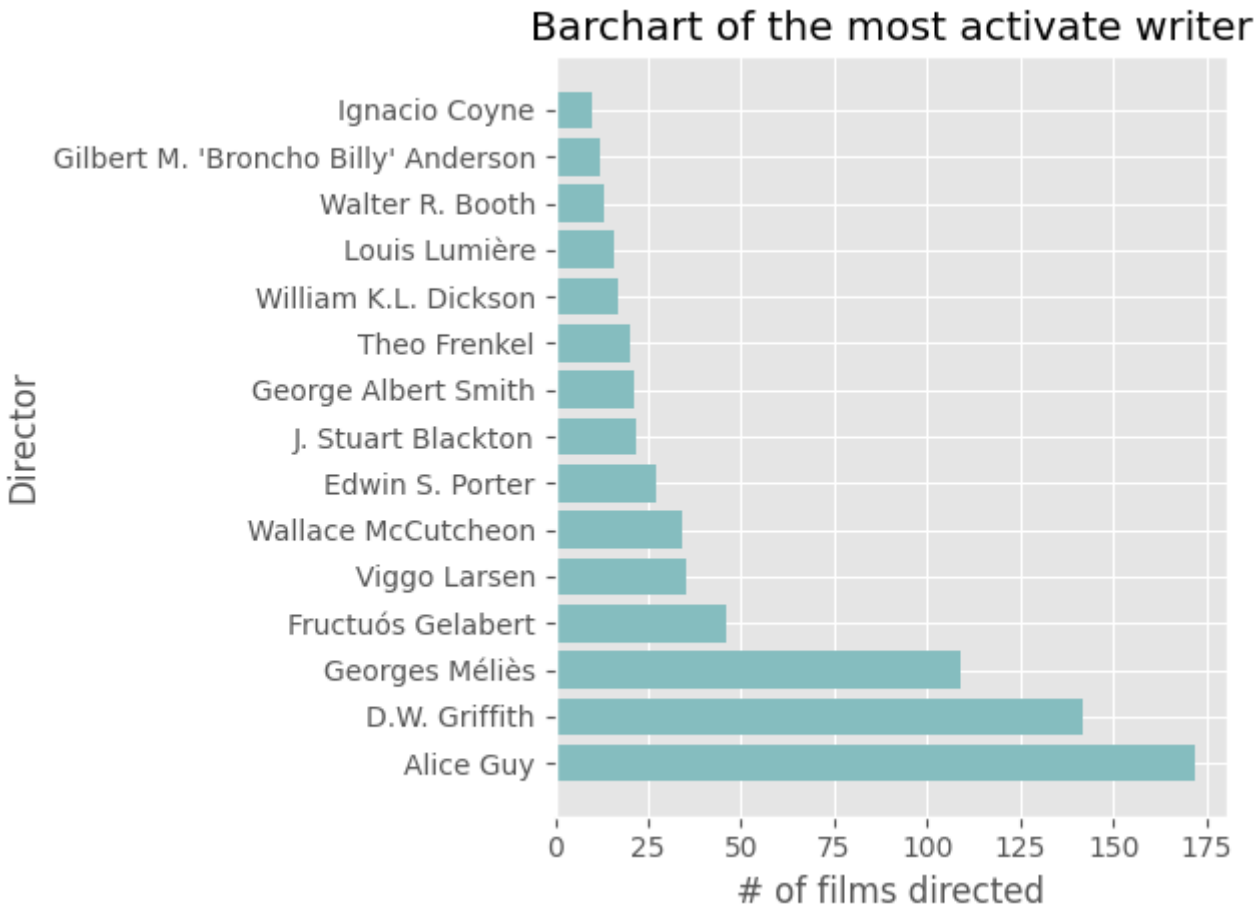
We had a problem when plotting the *runtime* feature. Indeed, the values where strings following this format "x min". Matplotlib does not know how to handle such values. For that reason we used the `split(' ')` method that allows us to split the string in different component after each space. Then we kept only the first part (i.e. the integer). Finally, we converted the previous with the `int()` function and exported the dataframe to a .csv file.



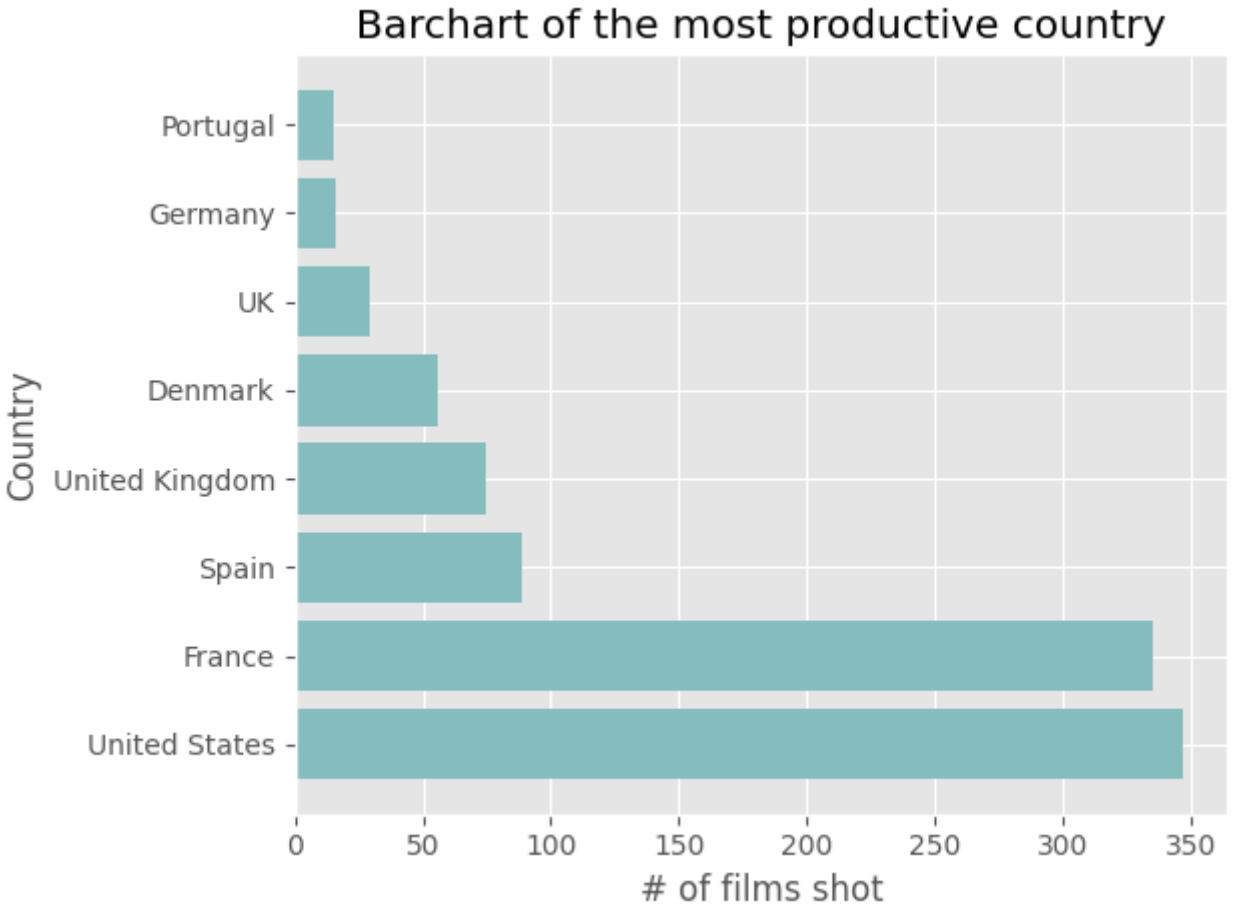
We have to note that the y-scale in the graph above is logaritmical, thanks to that we can more appreciate the fact that back in the days there were mostly short films. For the *genre* feature we add a similar problem. Some films had mutliple genre and were written as follows: "genre1 genre2 ... genreN". We used the same `split(' ')` function. Then we append every genre to a more general list and exported it to a .csv file.



The plot above, confirme our finding about the runtime feature. Indeed, we can see that almost every film is *short*. We can also see that the we have moslty drama, comedy and documentary. For the *director* feature, we used a python *default dictionary* to count the number of occurrences of each director. We kept only the top directors (i.e. the one with more than 10 films directed) then we exported the dictionary in the *graph.py* file. We did the exact same thing for the *country* feature.



From the above plot, we can see that 3 directors stand out. Indeed G. Méliès, D.W. Griffith, and Alice Guy directed more than 100 films in a 25 years period. That seems pretty impressive.



As

expected, the majority of films were shot in the US. More surprisingly, we can see that a lot of films (almost as much as the one shot in the US) were shot in France.

Visualisation Techniques

We knew from the data exploration phase that we would use the same kind of plot multiple times. To make our life easier, we decided to create two functions that could render different matplotlib chart in one line. Those functions can be found in the *graph.py* file.

Dataset-2

This dataset was retrieved on the GitHub repository available at this link: `https://github.com/enricoromano/Python-and-R-project-`. It encompasses weather data accessed through the API OpenMeteo. In total, there are initially 1008 instances or observations of 25 variables.

Imports

Various libraries have been used:

- `library(dplyr)`: this library was used to manipulate the dataset (i.e. select particular variables, create new dataframe subsets,etc.)
- `library(ggplot2)`: ggplot2 is one of the most used tool for visualisation purposes in R.
- `library(ggmap)`: it is a library to visualize spatial data on static maps. It was used to visualise on a real map to which cities the latitude and longitude coordinates matched.
- `library(sf)`: this library allows to transform an object into a simple features which have emphasis on spatial geometry and coordinates.
- `library(mapview)`: it provides functions to very quickly and conveniently create interactive visualisations of spatial data.
- `library(corrplot)`: corrplot provides a visual exploratory tool on correlation matrix that helps detect hidden patterns among variables.
- `library(plotly)`: it supports the functions to create interactive graphs. Thanks to this library a 3D-plot was displayed for the apparent temperature, the actual one and the cities.
- `library(lubridate)`: it eases the handling of datatimes.

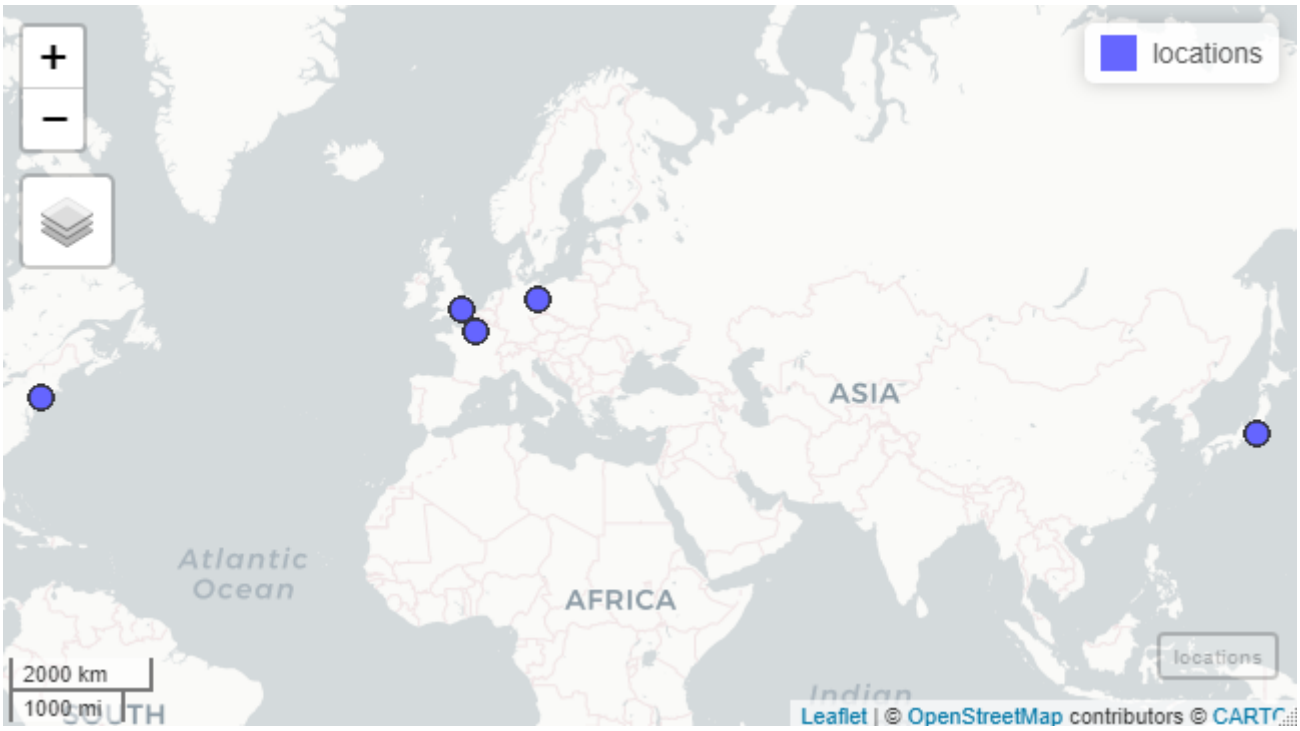
Data Cleaning

Initially, the dataset was made of 1008 observations and 25 variables. However, the columns containing the unit values were **removed** as they are not useful for visualisation purposes. Columns were **renamed** to avoid unnecessary words and clarify the dataframe.

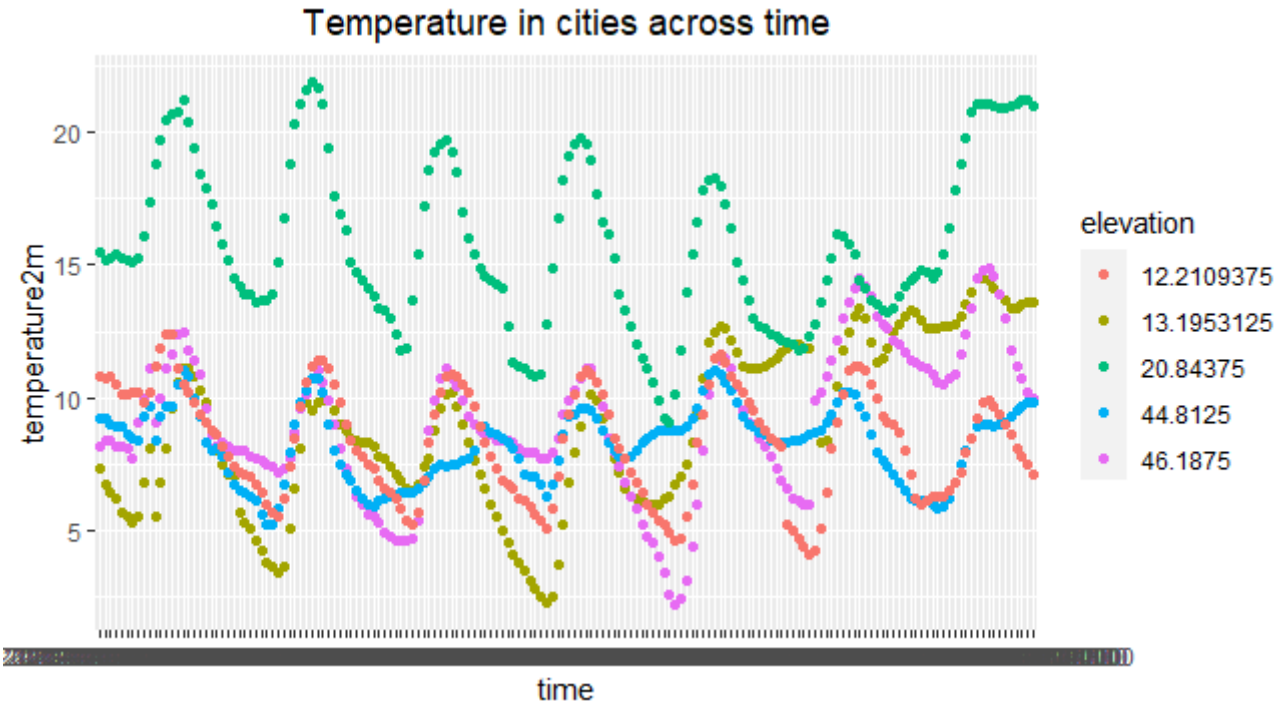
After primary analyses, it standed out that data about only 5 cities were uploaded instead of the presumed 6 (see GitHub repo where the data were retrieved from).

To remove duplicates we used the function `distinct()` from dplyr.

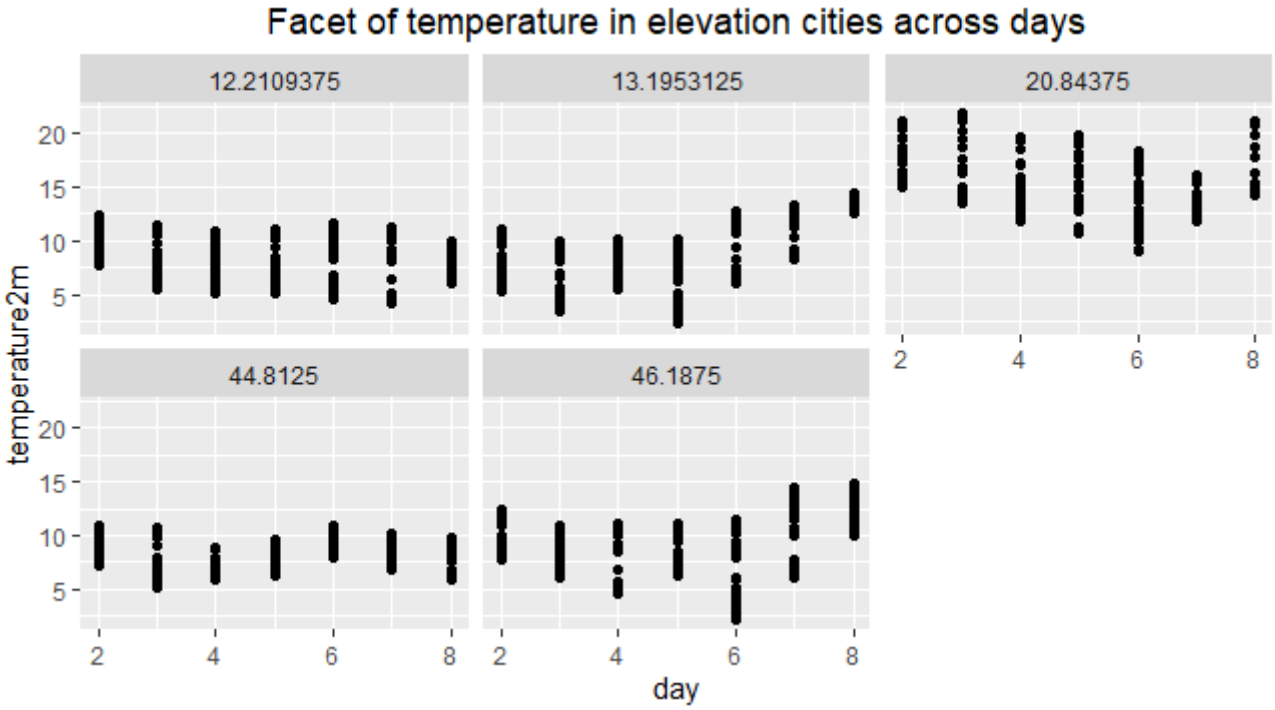
A **world map** was created to visualise the exact location of the cities and have a direct better understanding of next analyses.



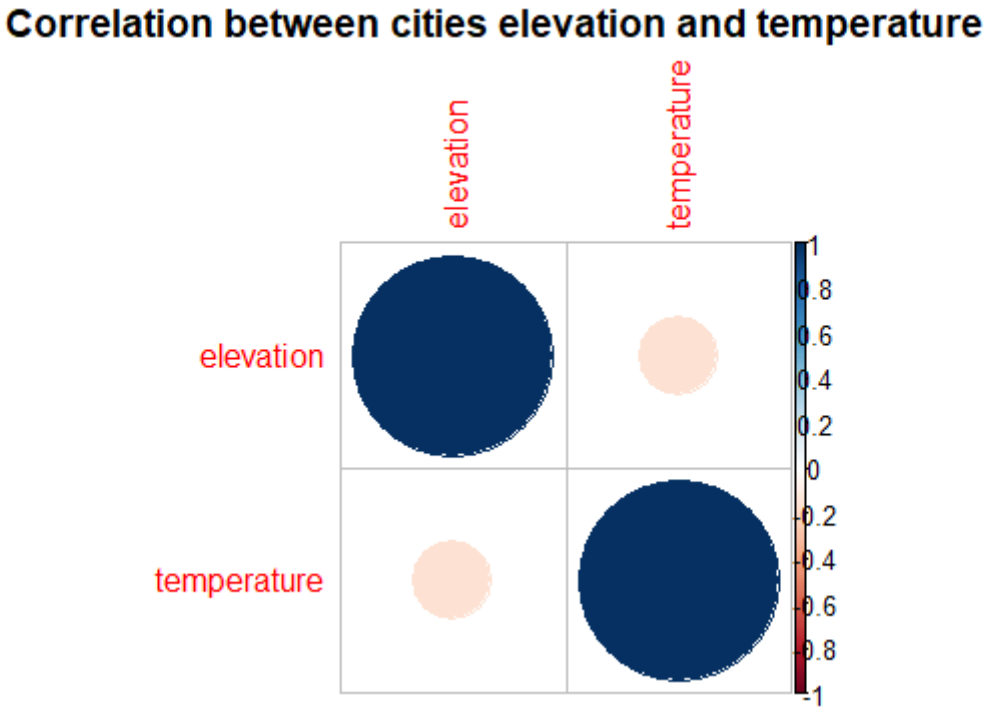
Then, this scatterplot seems to indicate that elevation is not really an indicator of the temperature.



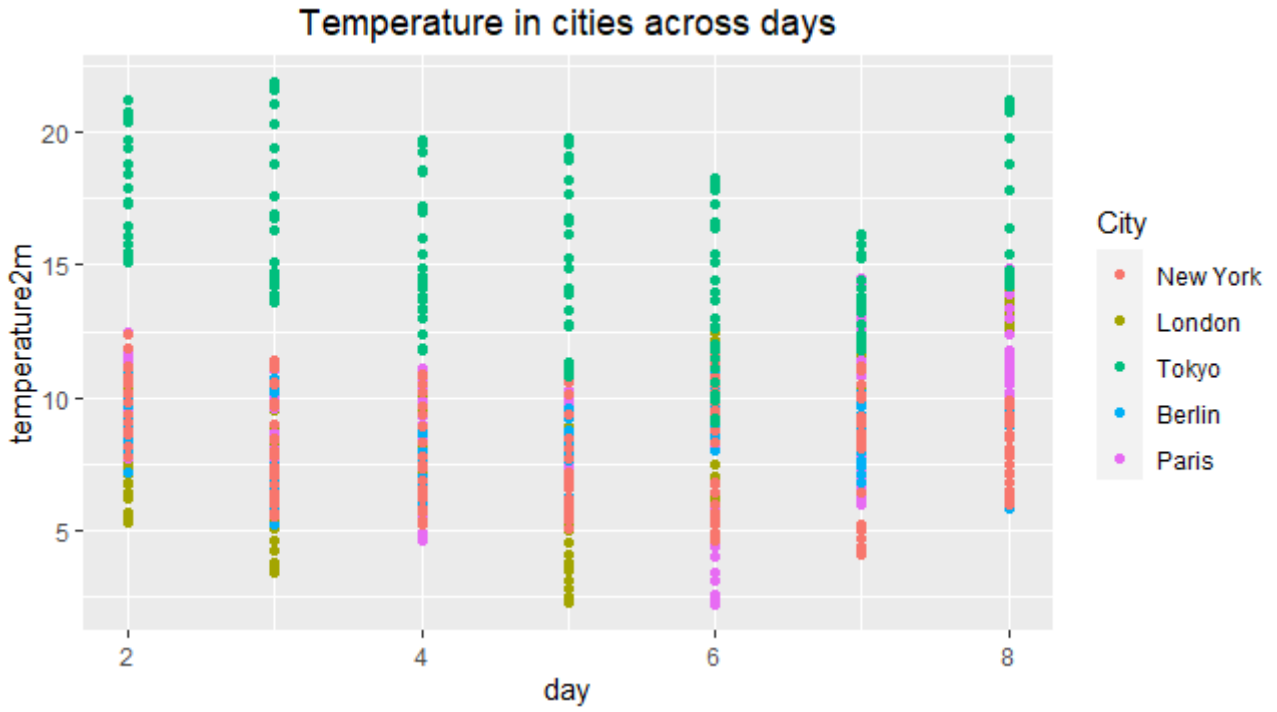
It is further confirmed thanks to the next **facet visualisation**.



Unsurprisingly given the plots, the **correlation** between elevation and temperature is close to 0.

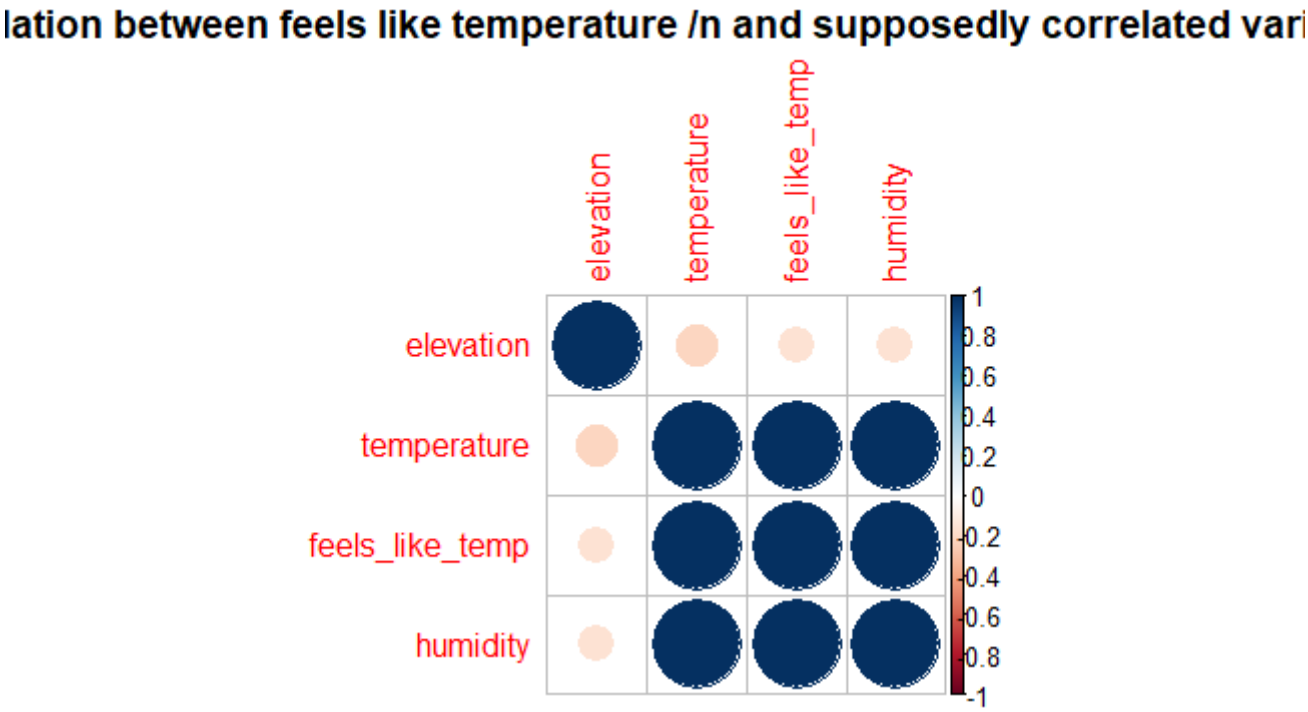


To have a better visualization of which city had the highest temperature on each day, some manipulations have been made. Clearly, **Tokyo** has the highest temperature and **London** the coolest ones.



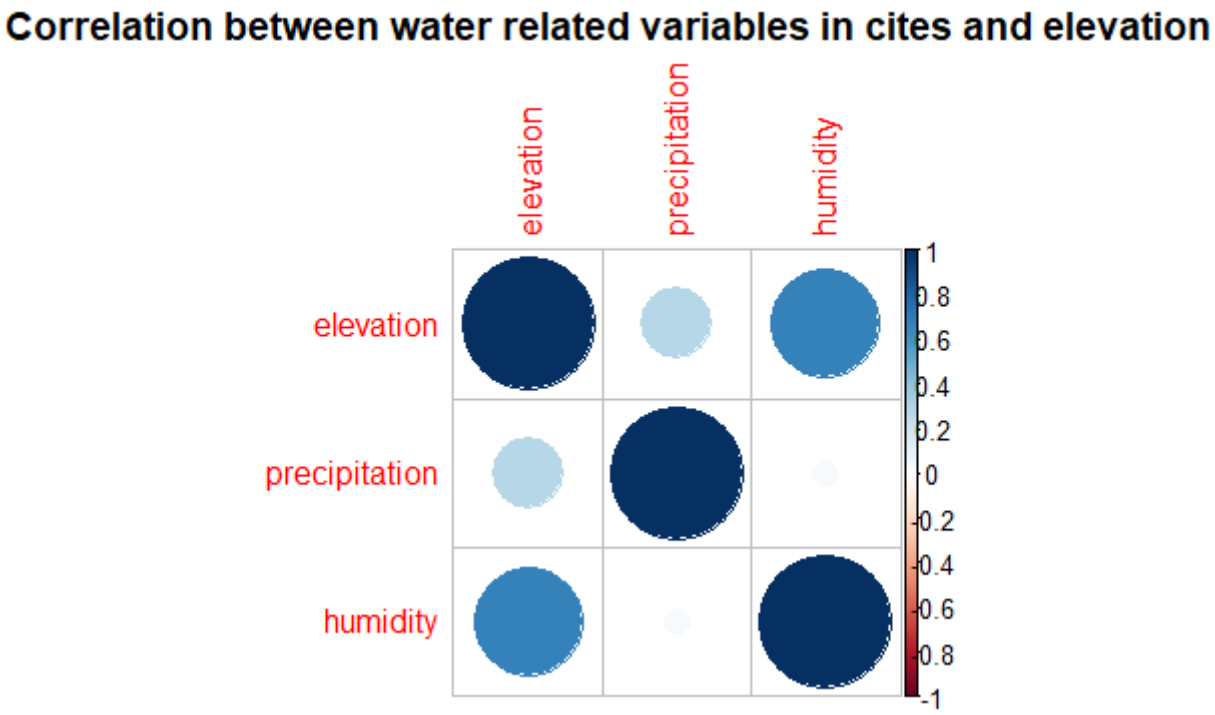
In the database, there were 2 kind of temperatures, namely the **air temperature 2 meters above the ground** and the **perceived feel-like temperature**. As can be seen from the next 3D plot it seems that a higher apparent temperature is as expected related with a higher real measured temperature.

In ordrer to confirm these visual insights, let us compute the correlation. The **correlation matrix** (on the new subset of the dataframe *feels_temp_by_cities*) correlations are really high between temperature and feels like temperature, as well as between temperatures and humidity.

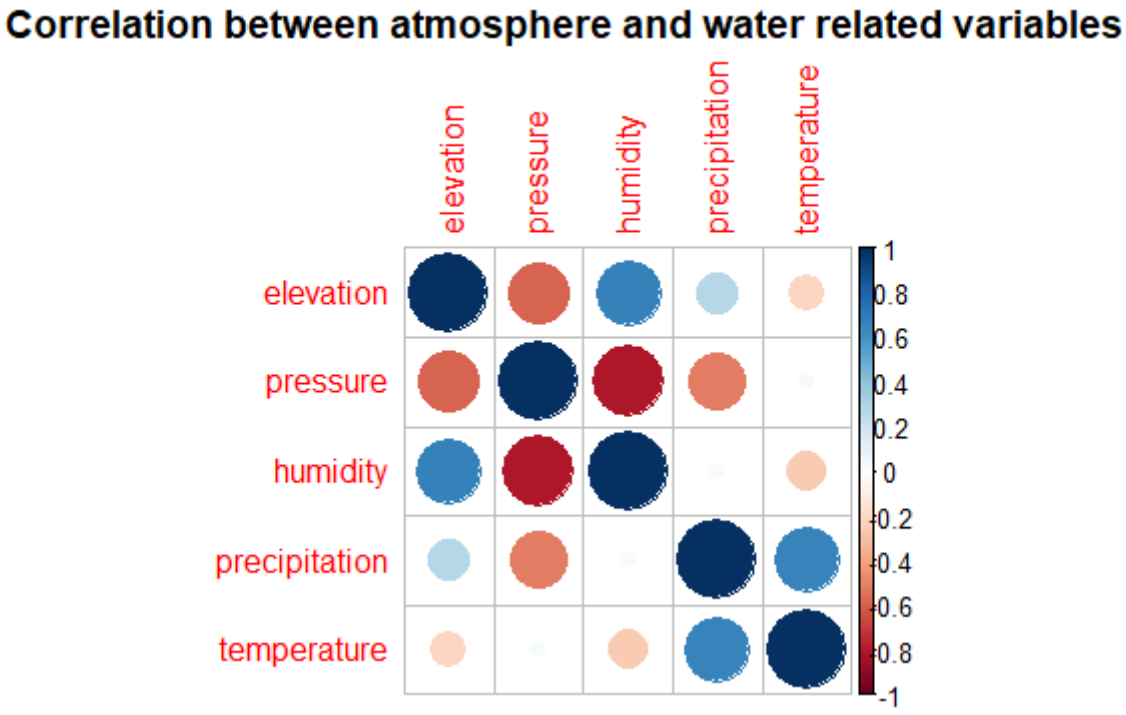


Other variables were the **humidity** and the **precipitation**. It seems that the **elevation** is slightly highly correlated with the humidity. Surprisingly, **precipitation** that happened the hour before the weather report

does not seem correlated to **humidity** at all.

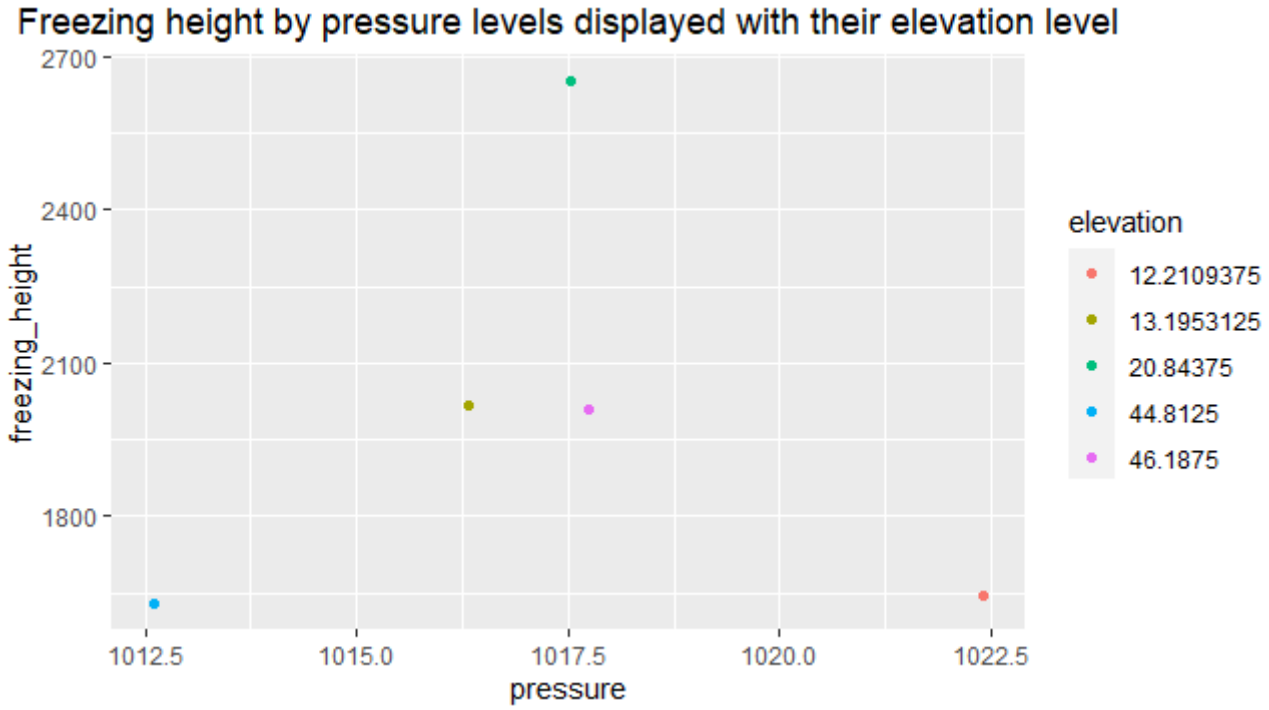


Atmospheric pressure is an indicator of weather. As such, according to National Geographic, when a low-pressure system moves into an area, it usually leads to cloudiness, wind, and precipitation. High-pressure systems usually lead to fair, calm weather. Let's try to verifiy this stated fact.



Contrarily to what expected from national geographic, a higher **pressure** does not seem to indicate higher **precipitation**.

In the end, the relation between the **freezing level height** and the **atmospheric pressure** was considered.



From the scatterplot, it seems that freezing height is not really related to pressure.