DOCUMENTATION ABOUT PYHTON AND R WORKGROUP – Task 1

API: STUDIO GHIBLI

C. Di Razza, M. Gioia, V. J. Striano

Our work was divided into two main tasks: data collection and data designing, and data analysis. This documentation relates to the first tasks.

API, or *Application Programming Interfaces*, are the interfaces provided by servers that you can use to retrieve and send third-party data using code.

Studio Ghibli is a Japanese animation film studio. Their APIs provides information about the film they've produced until 2016. There are 5 different APIs, one for **films**, which provides general information about films, **people**, which provides information about characters, **species**, which provides information about the living species in the film (like mammals, cats, or fantasy ones), **vehicles** and **locations**.

The documentation related to the APIs are provided at the following link: https://ghibliapi.herokuapp.com/#section/Studio-Ghibli-API

To access each APIs we used the following link: https://ghibliapi.herokuapp.com followed by 5 endpoints, one for each API: /films, /people, /species, /vehicles and /location. eg. https://ghibliapi.herokuapp.com/films. No other parameters and endpoints are available.

Each APIs has different variables, so we couldn't merge them (as long as each APIs shows observation of different nature)

Our purpose was to make the code simple, readable, and the leanest possible, mainly using for cycles. To achieve the same result for different objects (like drawing graphs) we provided different approaches.

Since .json files embedded into the APIs aren't natively manageable in Python, we need to transform them into a dataframe. Dataframe is a 2-dimensional labeled data structure with columns of potentially different types.

To achieve that we used *Pandas* library we used for the main task. Pandas is a library mainly used for data analysis. Pandas allows importing data from various file formats such as comma-separated values, JSON, SQL, and Microsoft Excel. Pandas allows various data manipulation operations such as merging, reshaping, selecting, as well as data cleaning, and data wrangling features. Its documentation is available here: https://pandas.pydata.org/docs/reference/index.html

import is a statement to import non-default-installed packages. We import pandas and abbreviate that with pd, thanks to the command as.

```
[3] import pandas as pd
```

For brevity sake we create a list named api, each element of it is a string with the endpoints.

```
[9] api = ["films","people","locations", "species","vehicles"]
```

We create an empty dictionary d.

```
[10] d = {}
```

Thanks to a for loop, we associate each i key (which relates to each string into the api list) to the json content of the name-related API. With this method we also save 1 line of code.

Pandas' read_json convert the file in the argument into a dataframe. The argument can be URLS, files, compressed files and anything that is in json format. The +i concatenate the main url with

```
[11] for i in api:

[12]    d[i] = pd.read_json(('https://ghibliapi.herokuapp.com/'+i))
```

Once the dataframes had been imported, we noticed that many of the data are not useful for our analysis -like images and urls- so we decided to delete them with the command drop() function from Pandas library. Finally we reorder the columns with the iloc() function from Pandas library in the order that we preferred.

The Pandas drop() function is used to drop specified labels from rows or columns. Remove rows or columns by specifying label names and corresponding axis, or by specifying directly index or column names. When using a multi-index, labels on different levels can be removed by specifying the level.
The loc() function helps us to retrieve data values from a dataset based on the index value passed to the function.

```
[15] d["films"] = d["films"].drop(columns=d["films"].columns[[4,5,-1,-2,-3,-4,-5]])

[16] d["films"] = d["films"].iloc[:, [1,2,3,4,5,6,7,8,9,0]]

[19] d["people"] = d["people"].drop(columns=d["people"].columns[[-1,-2,-3,-4]])

[20] d["people"] = d["people"].iloc[:, [1,2,3,4,5,0]]
```

Since the last 3 dataframes need the same changes, we use a for loop for each of the last three entry in the dictionary thanks to the if statement. In addition to automating the process, we also gain in the number of lines used, saving a line of code.

```
[23] for i in d:

[24]     if i in api[2:]:

[25]         d[i].drop(columns=d[i].columns[[-1,-2,-3]])
```

```
[26]        d[i] = d[i].iloc[:,[1,2,3,4,0]]
```

Finally, we save a .csv file for each dictionary. To name each file, we concatenate each key name and the “dataframe.csv” string for each file. Pandas DataFrame to_csv() function exports the dataframe to CSV format. If a file argument is provided, the output will be the CSV file. CSV is a data storage format that stands for Comma Separated Values with the extension .csv. CSV files store data values (plain text) in a list format separated by commas. Notably, CSV files tend to be smaller in size and can be opened in text editors. Converting JSON to CSV files can be a more convenient way to save and view data.

```
[29] for i in d:

[30]     d[i].to_csv((i+"dataframe.csv”))
```