Lab activity: analysis of music networks.

1 Introduction

This lab activity consists of 4 parts, that will be done during the remaining class sessions of the course. The 4 parts will go through the data life cycle, with the implementation of a set of functions defined in the assignments and the analysis of the obtained results.

- Part 1: Acquisition and storage of data
- Part 2: Data preprocessing.
- Part 3: Analysis.
- Part 4: Visualization.

First, we will work on the acquisition of data, showing where we can obtain it, how to do it, and what obstacles we may encounter during the process. Next, we will address the process of storing the acquired data so that the acquisition process only needs to be done once. Subsequently, we will work on the preprocessing of the stored data. Then, we will perform some analysis on the obtained data, that will allow us to characterize it and extract information. Finally, we will generate visualizations that allow us to graphically express the results of the analysis.

The lab activity will be carried out in groups of **two people** and only one submission per group is required (the report must clearly indicate the names and student IDs of each group member). The submission will be done through the virtual campus, in the specific section for practice submission.

There will be **a single deliverable** for the lab activity (covering the four parts). Moreover, before the submission of the deliverable, an **individual oral interview** to assess the progress of the work has to be done. Please take into account that the individual oral interview must be passed in order to be able to submit the deliverable for grading.

The deadline for submitting the activity is **June 7th**. The oral interview can be done during class time, at any time, no latter than **May 30th**. Thus the calendar for the lab activities is the following:

Week	Date	Thursday 15h-17h	Thursday 17h-19h
13	16/05/2024	Lab session 1: data adquisition	Lab session 2: data adquisition
14	23/05/2024	Lab session 3: data preprocessing	Lab session 4: data analysis
15	30/05/2024	Lab session 5: visualization	Lab session 6: final tweaks

Figure 1: Lab activity schedule.

As you will see in the statement of each session, the practice has two aspects: coding and analysing results.

Coding needs to be done in Python (3.7 or higher). A template plain python file (.py) will be provided for each part of the activity. You need to implement all the code of each part in the given template file[1] (indications of where you need to write your code are included as comments in the template). You can define additional global functions (and import any libraries) if needed, but **do not include any other code in the global scope**. The algorithm that implements the work for each session (the calls to the functions) needs to be included inside the *__main__* scope.

The other aspect of the practice requires you to analyze the information and reflect on some issues. These questions should be delivered in an additional PDF document. At the end of the statement of each session you will find a list detailing the files you need to deliver for that session (and their format).

At the beginning of the class, we will provide the statements of each part and, in some cases, present some concepts to be able to carry out the practical work. The rest of the class will be used to perform the work requested in the practice statement.

[1]Do not convert the file to a jupyter notebook. Because of the length of this activity, plain python files, that can be easily included from other files and debugger with IDEs, need to be used.

2 Part 1: Data adquisition and storage

2.1 The Spotify API

Spotify is a music streaming service available since 2006. Spotify primarily offers music and podcasts (currently, it has more than 82 million songs). Spotify users can search for and play music based on the song name, artist, album, or genre; they can create playlists with multiple songs and share them with other users; and they can also receive recommendations based on the music they have listened to so far.

An API (*Application Programming Interface*) is a set of functions and procedures that a certain application offers to be used by another application. Spotify has an API that allows access to the music data it offers and control of a user's music playback, among others. In this practice, we will focus on the first of these functionalities and use the large volume of music-related data (songs, albums, artists, styles, etc.) that Spotify offers us to put into practice the knowledge learned in the course.

Spotify's API has a reference documentation where all available *endpoints* are detailed.

2.1.1 Authentication

To use the API you need to authenticate. Therefore, before continuing with this process, you need to have a Spotify developer account, create an application, and obtain the credentials. To do this, follow these steps:

1. If you don't have a Spotify account, create one. If you already have one, you can use it by activating it for development purposes (as explained in the next step).
2. Connect Spotify Developer with your Spotify account from the developer dashboard.
3. Accept the Terms of Service.
4. Create an application: assign it a name and a description, and accept again the Developer Terms of Service and the Design guidelines. You need to provide a redirect URI (any valid URI is accepted) and select that you are planning to use the *Web API*.

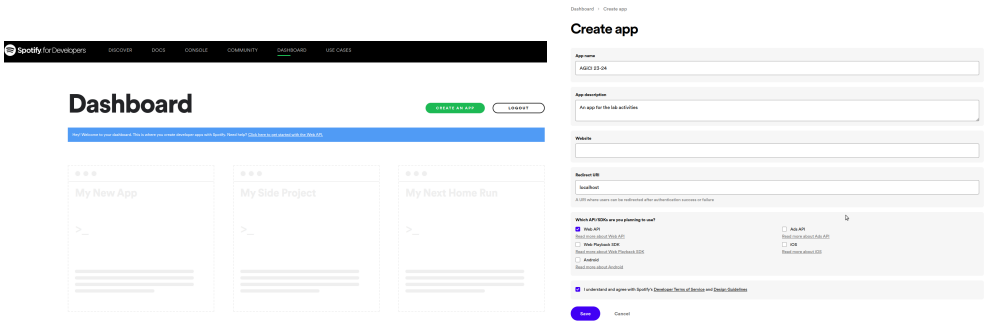

Figure 2: Creating the application in the Spotify developer dashboard.

Once these steps are completed, you will obtain the access credentials that you can use to access the API (the values `Client ID` and `Client Secret`).

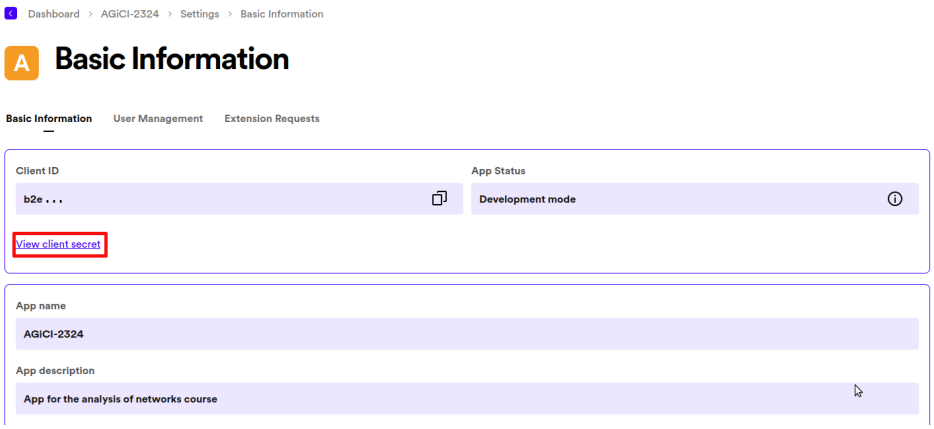

Figure 3: Application credentials (`Client ID` and `Client Secret`).

It is very important to keep in mind that the Spotify API, like the vast majority of APIs, has usage limits. Although the specific limits are not described in the documentation (to be able to adjust them), the error code returned by the API when reaching this limit and some strategies to manage it are detailed.

Please take a look at the API calls available before starting to code. Note that there may be more than one call that provides you with the information you want, but keep in mind that the number of calls is restricted, so try to **use the calls that optimize the information you want to obtain as much as possible**.

All the activities proposed in this practice can be done within the API's rate limits, so if you are getting banned for surpassing the limits, **please review your code and/or contact the teachers of the course to help you identify the problem**. Do not execute the same code again without having first identified and solved the problem.

Note that your dashboard has counters that inform you of the number of calls you have already made (although these are not updated in real time).

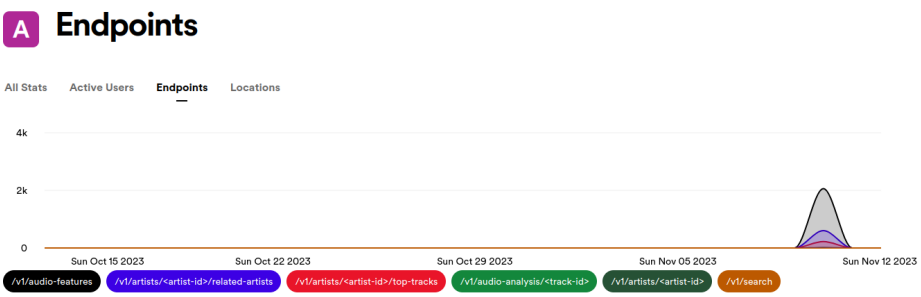

Figure 4: The dashboard provides information about the number of calls made to the API.

2.1.2 The spotipy library

To simplify data collection, we will access the Spotify API using the spotipy Python library. The documentation for the `spotipy` library can be found in the API reference document. You can install the library with `pip`:

```
pip install spotipy --upgrade
```

To authenticate yourself with your developer credentials and start using the Spotify API with the `spotipy` library, you can use the basic example code provided by the library in its official documentation:

```
import spotipy
from spotipy.oauth2 import SpotifyClientCredentials

auth_manager = SpotifyClientCredentials()
sp = spotipy.Spotify(auth_manager=auth_manager)

playlists = sp.user_playlists('spotify')
while playlists:
    for i, playlist in enumerate(playlists['items']):
        print("%4d %s %s" % (i + 1 + playlists['offset'], playlist['uri'],
    playlist['name']))
    if playlists['next']:
        playlists = sp.next(playlists)
    else:
        playlists = None
```

The code imports the library (lines 1 and 2), constructs the `sp` object with user authentication (lines 4 and 5), and finally retrieves information about the playlists of the `spotify` user. Note that in order to authenticate correctly, you will need to define the environment variables `SPOTIPY_CLIENT_ID` and `SPOTIPY_CLIENT_SECRET` with the values of your application[2]. Alternatively, you can specify these values in your code when creating the `SpotifyClientCredentials` object:

```
CLIENT_ID = "..."
CLIENT_SECRET = "..."
auth_manager = SpotifyClientCredentials(
         client_id=CLIENT_ID,
         client_secret=CLIENT_SECRET)
sp = spotipy.Spotify(auth_manager=auth_manager)
```

2.2 Data acquisition and storage

In this first session of the practice, we will obtain two artist datasets (two networks) and a song dataset (tabular data). Together with the statement of this first session, a skeleton file where the functions to be implemented in each activity are explicitly stated is attached (`Lab_AGX_202324_P1_skeleton.py`).

To obtain the Spotify artists datasets, a crawler will be implemented.

[2]If you are using the pycharm IDE, you can configure them by right-clicking on the code file name > *More Run/Debug* > *Modify Run Configuration* > *Environment variables*

1. (**1 point**) Implement the function `search_artist` with the following header:

```
def search_artist(sp: spotipy.client.Spotify, artist_name: str) -> str:
    """
    Search for an artist in Spotify.

    :param sp: spotipy client object
    :param artist_name: name to search for.
    :return: spotify artist id.
    """
```

   The function will receive the name of the artist to search for and will return a string with the Spotify ID (Spotify ID) of this artist.

2. (**3 points**) Implement the function `crawler` with the following header:

```
def crawler(sp: spotipy.client.Spotify, seed: str, max_nodes_to_crawl: int,
    strategy: str = "BFS", out_filename: str = "g.graphml") -> nx.DiGraph:
    """
    Crawl the Spotify artist graph, following related artists.

    :param sp: spotipy client object
    :param seed: starting artist id.
    :param max_nodes_to_crawl: maximum number of nodes to crawl.
    :param strategy: BFS or DFS.
    :param out_filename: name of the graphml output file.
    :return: networkx directed graph.
    """
```

   The *crawler* will take as seed the identifier of an artist and obtain its related artists, which will be used to continue the data capture process.

   Two scheduling algorithms have to be implemented: breadth-first search (BFS) and depth-first search (DFS). For both algorithms, in case of having to decide which node to explore among a set of nodes at the same level (that is, in case of having to break a tie), the first artist obtained from the API will always be selected.

   The *crawler* will stop when it has explored the maximum number of artists indicated as a parameter or when there are no more known artists pending exploration. Remember that exploring an artist means retrieving all of its related artists.

   The *crawler* will create a graph using `networkx` with the obtained data (the nodes will represent artists and the edges will represent the relationships between artists provided by the Spotify related artist call). For each artist, we will store at least their name, Spotify identifier, number of followers, popularity, and associated musical genres.

   Finally, the function will save the graph in `graphml` format to the file indicated as a parameter and return the graph (the `networkx` graph object).

3. (**2 points**) Implement the function `get_track_data` with the following header:

```
def get_track_data(sp: spotipy.client.Spotify, graphs: list, out_filename:
    str) -> pd.DataFrame:
    """
    Get track data for each visited artist in the graph.

    :param sp: spotipy client object
    :param graphs: a list of graphs with artists as nodes.
    :param out_filename: name of the csv output file.
    :return: pandas dataframe with track data.
    """
```

The function will receive a list of `networkx` graphs (where nodes represent artists) and will return a `pandas dataframe` with the top songs in Spain for the artists that appear in each and all of the graphs in the list (each row of the dataframe will represent a song). In addition, the function will save the data from the `dataframe` in `csv` format (comma-separated values) to the file indicated by the parameter.

For each song, we will store at least:

- Basic song data: identifier, duration, name, and popularity.
- Audio feature data for the song: danceability, energy, loudness, speechiness, acousticness, instrumentalness, liveness, valence, and tempo.
- Album data: identifier, name, and release date.
- Artist data: identifier and name. In case more than one artist participates in a song, we will only store the data of the explored artist from which we retrieved the song.

In order to avoid rate limits, it's very important that the number of calls to the API made by this function is minimized. Please read the API documentation to use it as efficiently as possible.

Recall that all the activities proposed in this practice can be done within the API's rate limits, so if you are getting banned for surpassing the limits, **please review your code and/or contact the teachers of the course to help you identify the problem**. Do not execute the same code again without having first identified and solved the problem.

4. (**1 point**) Using the previous functions, obtain:

   (a) A graph of related artists starting with the artist *Taylor Swift* and exploring 100 artists with BFS (we will call this graph g_B).

   (b) A graph of related artists starting with the artist *Taylor Swift* and exploring 100 artists with DFS (we will call this graph g_D).

   (c) A dataset of songs from the artists that appear in both the previous graphs (we will call this dataset $\mathbb{D}$).

   (d) A graph of related artists starting with the artist *Pastel Ghost* and exploring 100 artists with BFS (we will call this graph h_B).

2.3 Questions to answer in the report

1. (**1.5 points**) Provide the order and size of the graphs g_B and g_D.
   (a) Explain why, having explored the same number of nodes, the order of the two graphs (g_B and g_D) differs.
   (b) Justify which of the two graphs should have a higher order.
   (c) Explain what size the two graphs should have.
2. (**1 point**) Indicate the minimum, maximum, and median of the in-degree and out-degree of the two graphs (g_B and g_D). Justify the obtained values.
3. (**0.5 points**) Indicate the number of songs in the dataset $\mathbb{D}$ and the number of different artists and albums that appear in it.
   (a) Justify why the number of songs you obtained is correct, considering the input graphs.
   (b) Justify why the number of retrieved albums is correct.

2.4 Evaluation and expected results of this part of the practice

This first session of the practice will account for 25% of the total grade of the practice.

Remember that it is important that **the Python code includes sufficient comments** to understand its operation and **respect the headers** of the provided functions.

The final submission must be a zip file. For this part of the lab activity, you need to include:

- `Lab_AGX_202324_P1_skeleton.py`: a plain python file where each of the functions defined in this document should be implemented.
- `Lab_AGX_202324_report.pdf`: the final report (a pdf file) must have a section called **Part 1: data acquisition** where all the questions asked in this document will be answered in detail. The answers to the questions should be numbered and in order.
- The data files obtained in exercise 4:
  - `gB.graphml`: a graphml file with the artist graph g_B.
  - `gD.graphml`: a graphml file with the artist graph g_D.
  - `songs.csv`: a csv file with the songs dataset $\mathbb{D}$.
  - `hB.graphml`: a graphml file with the artist graph h_B.