An exploratory analysis of Spotify data

Carlos Medina

10/29/22

Table of contents

Preface **4**
Getting Started . . . 4
Table of Contents . . . 4
Tools . . . 5
About me . . . 5

I Consuming the api **6**

1 An artist's analysis **7**
1.1 API . . . 7
1.2 Libraries . . . 7
1.3 Configuration . . . 8
1.4 Workflow . . . 9
1.5 Artist profile . . . 9
1.6 Top Tracks . . . 11
1.7 Length vs popularity . . . 12
1.8 Release Date and popularity . . . 14
1.9 Conclusion . . . 16

2 Albums, dates and popularity **18**
2.1 API . . . 18
2.2 Libraries . . . 18
2.3 Workflow . . . 20
2.4 Albums . . . 20
2.5 Popularity . . . 22
2.6 Graphs . . . 22
2.7 Conclusion . . . 24

3 Geolocation analysis **25**
3.1 Libraries . . . 25
3.2 Workflow . . . 27
3.3 Get the data . . . 27
3.4 How many markets can each album have? . . . 28
3.5 What is the market with more albums? . . . 29

II Analyzing my listening patterns **32**

4 Getting and cleaning the data **33**
4.1 Requesting the data from Spotify . . . 33
4.2 Cleaning the data . . . 33
4.2.1 JSON Files . . . 33
4.2.2 Importing the data . . . 34

5 Dashboard template creation **35**
5.1 Template creation . . . 35

6 Exploratory analysis **37**
6.1 Overall . . . 37
6.1.1 Listening trend over the years - Line and clustered column chart . . . 37
6.1.2 Shuffle and skipped times - Gauge . . . 38
6.1.3 Most frequent artists - Stacked bar char . . . 38
6.2 Platform and Geo . . . 38
6.2.1 Trend of platform over the years - Ribbon chart . . . 38
6.2.2 Geo Data - ArcGIS maps . . . 39
6.3 Artists . . . 40
6.3.1 Most listened artist over the years - Ribbon chart . . . 40
6.4 Listening time . . . 40
6.4.1 Played time over the years - Scatter chart . . . 40
6.4.2 Average duration by track (m) - Multi-row card . . . 41

7 PowerBI and python **42**
7.0.1 Radar Chart . . . 43

III Music trends in south america **46**

8 Music trends in south america **47**

9 Version history **48**

Preface

A data visualization curriculum of interactive notebooks using *Quarto*, *Plotly*, *PowerBi* and *Google Data Studio*. This document was made to be part of my portfolio as a data analyst. For this book I took one of the subjects that I love the most, music; And using the Spotify API and my listening historic data I will explore the many options we have to display data appropriately.

Getting Started

Before showing code and data, I will explain the following points:

- This document contains multiple Jupyter Notebooks which are available for download in the Github repository.
- Most of the graphs were created with Plotly to take advantage of its visualization advantages with JS.
- The pages were made with markdown and rendered with Quarto.

Table of Contents

This document is divided into three sections, in each one I will use a different tool and analyze different sections in the data.

- **Consuming the api**: In this section I will show how to consume the Spotify API and plot the data with Plotly.
- **Analyzing my listening patterns**: My personal data will be analyzed to see how I listen to music and what are my favorite artists. This seccion will be made with PowerBI.
- **Music trends in south america**: In this section I will get data using the API again, but the tool will be Google Data Studio.

Tools

- Python
- Jupyter Notebook
- Plotly
- PowerBI
- Google Data Studio

About me

I am Carlos Medina, I have 10 years of experience in InfoSec and now I am starting with data analysis, you can see all the information on my *personal page*.

Part I

Consuming the api

1 An artist's analysis

In this first part I'm going to work with the `/artist/{id}` session of the Spotify API. With this functionality general data can be extracted. This general data contains things as genres, popularity and followers. I will also work with the artist's top 10 songs to make a very general summary of the artist's profile, which will give us ideas about their years of popularity, duration of the songs and most well-known albums.

1.1 API

A small summary of the functionalities that I will use:

```
GET /artist/{id}
GET /artist/{id}/albums
GET /artist/{id}/top-tracks
```

Parameter	Type	Description
id (required)	`string`	Get Spotify catalog information for a single artist identified by their unique Spotify ID.The ID of the artist.
albums	`string`	Get Spotify catalog information about an artist's albums.
top-tracks	`string`	Get Spotify catalog information about an artist's top tracks by country.

As I use each parameter, I will explain the configuration and filtering options they have.

1.2 Libraries

I import the libraries I will need:

```
from pandas import json_normalize
import requests
import plotly.express as px
import plotly.io as pio
```

```
pio.renderers.default = "plotly_mimetype+notebook_connected"
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import base64
from IPython.display import Image, display

def mm(graph):
    graphbytes = graph.encode("ascii")
    base64_bytes = base64.b64encode(graphbytes)
    base64_string = base64_bytes.decode("ascii")
    display(Image(url="https://mermaid.ink/img/" + base64_string))
```

1.3 Configuration

Before thinking about making requests to obtain data, you must first create an app in Spotify, this app will give you the possibility to authenticate yourself and make the requests that are needed. To make this app you can go to this link where you will find all the necessary information. Link

After the app has been created, the authentication data is delivered, similar to these:

```
CLIENT_ID = '81e800e81ecf4997b5b9fb12efeb3ff2'
CLIENT_SECRET = '0e4364f440f148779d8a9f17976ecf1b'
```

(These were removed after making this publication)

When the app authenticates, Spotify will return a token, this token is the one used in the header to make all requests. All tokens have a time to live, I'm not sure, but I think it's 3600 seconds, I recommend looking at the documentation if this may be a problem for you.

In the following code blocks I show how to create a function to get this token and then store it in the header variable.

```
# Function to get the token
def get_token():
    url = 'https://accounts.spotify.com/api/token'
    auth_response = requests.post(url, {
        'grant_type': 'client_credentials',
        'client_id': CLIENT_ID,
        'client_secret': CLIENT_SECRET,
    })
```

```
    if auth_response.status_code != 200:
        raise Exception('Error getting token')
    else:
        auth_response_data = auth_response.json()
        return auth_response_data['access_token']
```

```
# Get token
access_token = get_token()
header = {
    'Authorization': 'Bearer {token}'.format(token=access_token),
    'accept':'application/json'
}
```

```
base_url = 'https://api.spotify.com/v1/'
```

1.4 Workflow

The workflow from when the user enters the name of the artist until it is saved in the dataframes is represented in the following diagram

```
mm("""
graph TD
    A[user_input] -->|Search| B(GET item 0)
    B --> C(save artist_id )
    C --> D{GET request}
    D -->|/artist/artist_id/albums| E[df]
    D -->|/artist/artist_id/top-tracks| F[df]
""")
```

```
<IPython.core.display.Image object>
```

1.5 Artist profile

I ask the user to enter the name of the user they want to analyze, and then I use Spotify's search to get the artist ID.

The ID is a unique text string for each artist/song/album or list and this ID is one of the most important parameters to make requests since it allow us to have more precision in our needs.

The output of the search is a list, and I take the first result.

```
#ask for user input for artist name
artist_name = input('Enter artist name: ')
print('the artist name is: ', artist_name)
```

```
the artist name is:  Rush
```

```
#get artist id
artist_id = requests.get(base_url + 'search?q={}&type=artist'.format(artist_name), headers
```

The structure of the json object delivered by `/artist/{id}` can be found at this link. Link

```
#get artist profile
r_artist_profile = requests.get(base_url + 'artists/{}'.format(artist_id), headers=header)
df_artist = json_normalize(r_artist_profile)
df_artist.columns
```

```
Index(['genres', 'href', 'id', 'images', 'name', 'popularity', 'type', 'uri',
       'external_urls.spotify', 'followers.href', 'followers.total'],
      dtype='object')
```

I don't need all this information, so I select only the necessary columns:

```
df_artist.drop(['images', 'uri', 'href', 'followers.href','type'], axis=1, inplace=True)
```

I always change the name of the columns to make it easier to read and understand the data.

```
#change column names
df_artist.columns = ['Genres', 'ID', 'Artist', 'Popularity', 'URL', 'Followers']
```

And finally a have a very brief overview of the artist:

```
df_artist.transpose()
```

	0
Genres	[album rock, art rock, canadian metal, classic...
ID	2Hkut4rAAyrQxRdof7FVJq
Artist	Rush
Popularity	66
URL	https://open.spotify.com/artist/2Hkut4rAAyrQxR...
Followers	1975105

The popularity of the artist. The value will be between 0 and 100, with 100 being the most popular. The artist's popularity is calculated from the popularity of all the artist's tracks. With this we see that Rush is not a very popular band.

1.6 Top Tracks

After obtaining the general profile of the artist, the first thing I will do is analyze the most popular songs. For this I must use the `artists/{id}/top-tracks` section and specify the country since these trends change from country to country.

For this example I will use the United States since for the band I chose it was (and still is) its biggest market.

```
#get top tracks of the artist
r_artist_top_tracks = requests.get(base_url + 'artists/{}/top-tracks?market=US'.format(art
df_artist_top_tracks = json_normalize(r_artist_top_tracks['tracks'])
df_artist_top_tracks['Artist'] = df_artist_top_tracks['artists'].apply(lambda x: x[0]['nam
df_artist_top_tracks. columns
```

```
Index(['artists', 'disc_number', 'duration_ms', 'explicit', 'href', 'id',
       'is_local', 'is_playable', 'name', 'popularity', 'preview_url',
       'track_number', 'type', 'uri', 'album.album_type', 'album.artists',
       'album.external_urls.spotify', 'album.href', 'album.id', 'album.images',
       'album.name', 'album.release_date', 'album.release_date_precision',
       'album.total_tracks', 'album.type', 'album.uri', 'external_ids.isrc',
       'external_urls.spotify', 'Artist'],
      dtype='object')
```

Again I don't need all this information, so I select only the necessary columns:

```
df_artist_top_tracks=df_artist_top_tracks.drop(['artists',
                                                'href',
                                                'is_local',
                                                'is_playable',
                                                'preview_url',
                                                'type',
                                                'uri',
                                                'album.album_type',
                                                'album.artists',
                                                'album.external_urls.spotify',
                                                'album.href',
                                                'album.release_date_precision',
                                                'album.type',
                                                'album.uri',
                                                'external_ids.isrc',
                                                'external_urls.spotify'], axis=1)
```

The information of the duration is in milliseconds, I convert it to minutes that is easier to read and understand.

```
#duration_ms to minutes
df_artist_top_tracks['Duration'] = df_artist_top_tracks['duration_ms'].apply(lambda x: x/6
```

The release date is in the format YYYY-MM-DD, but I only need the year, so I convert it to a year.

```
#change format of release date
df_artist_top_tracks['Release Date'] = df_artist_top_tracks['album.release_date'].apply(la
#sort by release date
df_artist_top_tracks = df_artist_top_tracks.sort_values(by='Release Date')
df_artist_top_tracks.head(2)
```

	disc_number	duration_ms	explicit	id	name	popularity	trac
5	1	429973	False	1gkn90ExKRNAOlhDs4RoW0	Working Man	60	8
4	1	202200	False	54TaGh2JKs1pO9daXNXI5q	Fly By Night	61	5

1.7 Length vs popularity

For visualization my two favorite libraries are Seaborn and Plotly, for this example I will use Plotly to take advantage of its attributes with javascript and that the visualization can be a

little more comfortable.

```
#plot top tracks
fig = px.scatter(df_artist_top_tracks,
                 x='popularity',
                 y='Duration',
                 color='name',
                 title='Top Tracks of {}'.format(artist_name),
                 #marginal_y="violin",
                 #marginal_x="violin",
                 trendline="ols",
                 template="ggplot2",
                 width=800,
                 height=600,
                 labels={'popularity':'Popularity', 'Duration':'Duration (min)'},
                 )
fig.update_layout(width=700, height=600)
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

```
Unable to display output for mime type(s): application/vnd.plotly.v1+json, text/html
```

In this graph I can see that the popularity is inversely proportional to the duration of the song.

In the first part of the graph, there are three songs that are longer than 5 minutes, but are the lowest in popularity, those songs are:

- Subdivisions
- Red Barchetta
- Working man

(A song that talks about not being popular is the least popular xD)

On the other hand, at the other extreme, you see a single song, it lasts less than 5 minutes, and it is the most popular, Tom Sawyer.

Analyzing the previous graph, doubts arise such as: - Is Rush's popularity related to the year? - The other popular songs be on the same album as Tom Sawyer?

1.8 Release Date and popularity

I will use the Spotify API to get the release date of the album that Tom Sawyer is on.

```
#group df by album and count and order by count
df_artist_top_tracks_grouped = df_artist_top_tracks.groupby(['album.name', 'Release Date']
#change column name
df_artist_top_tracks_grouped.columns = ['Album', 'Release Date', 'Count']
#sort by count
df_artist_top_tracks_grouped = df_artist_top_tracks_grouped.sort_values(by='Release Date',
#set release date as int
df_artist_top_tracks_grouped['Release Date'] = df_artist_top_tracks_grouped['Release Date'
df_artist_top_tracks_grouped
```

	Album	Release Date	Count
4	Rush	1974	1
1	Fly By Night	1975	1
0	A Farewell To Kings	1977	1
3	Permanent Waves	1980	2
2	Moving Pictures (2011 Remaster)	1981	4
5	Signals	1982	1

Since this is a fairly small dataframe, the answer can be seen quite obviously in the table above, but with larger dataframes, it won't be, so I still decided to plot it:

```
fig=px.line(df_artist_top_tracks_grouped,
            x='Release Date',
            y='Count',
            text='Count',
            #color='Album',
            title='Top Tracks of {}'.format(artist_name),
            template="ggplot2",
            width=800,
            height=500,
            labels={'Release Date':'Release Date', 'Count':'Count'},
            )
fig.update_traces(textposition="bottom right")
fig.update_layout(width=740, height=600)
fig.show()
```

```
Unable to display output for mime type(s): application/vnd.plotly.v1+json, text/html
```

The graph shows that the most popular songs are on the same album as Tom Sawyer, and the release date is related to the popularity.

Could it be that 1981 was the year Rush released more albums? To find out this, I have to get all the albums I do a simple timeline. In the next part I use the API to get the info and a simple while loop for pagination.

```
#ger artist albums limit 50
r_albums = requests.get(base_url + 'artists/' + artist_id + '/albums', headers=header, par
r_albums=r_albums.json()
df_albums=json_normalize(r_albums['items'])
#get next page
while r_albums['next']:
    r_albums = requests.get(r_albums['next'], headers=header)
    r_albums=r_albums.json()
    df_albums=df_albums.append(json_normalize(r_albums['items']))
df_albums=df_albums.drop(['album_type',
                          'artists',
                          'href',
                          'images',
                          'release_date_precision',
                          'external_urls.spotify',
                          'uri',
                          'type'],axis=1)
df_albums['Release Date'] = df_albums['release_date'].apply(lambda x: x[:4])
```

```
df_albums.head(2)
```

	album_group	available_markets	id
0	album	[CA, JP]	5nZ5I0gA3x6KEkIpHQWw4l
1	album	[AD, AE, AG, AL, AM, AO, AR, AT, AU, AZ, BA, B...	2PBaIv21OWCmecNenZionV

After having all the albums, I'm going to plot both

```
df_artist_top_tracks_grouped.drop(['Album'], axis=1, inplace=True)
```

```
#group by release date and sort by count
df_albums_grouped = df_albums.groupby(['Release Date']).size().reset_index(name='Count')
```

```
df_albums_grouped.columns = ['Release Date', 'Count']
df_albums_grouped = df_albums_grouped.sort_values(by='Release Date', ascending=True)
df_albums_grouped['Release Date'] = df_albums_grouped['Release Date'].astype(int)
```

```
#graph df_artist_top_tracks_grouped and df_albums_grouped
fig = make_subplots(specs=[[{"secondary_y": True}]])
fig.add_trace(go.Scatter(x=df_artist_top_tracks_grouped['Release Date'],
                         y=df_artist_top_tracks_grouped['Count'],
                         name='Top Tracks'),
              secondary_y=False)
fig.add_trace(go.Scatter(x=df_albums_grouped['Release Date'],
                         y=df_albums_grouped['Count'],
                         name='Released Albums'),
              secondary_y=True)
fig.update_layout(title='Top Tracks and Albums of {}'.format(artist_name),
                  template="ggplot2",
                  width=740,
                  height=600,
                  yaxis_title="Count",
                  yaxis2_title="Count")
fig.show()
```

```
Unable to display output for mime type(s): application/vnd.plotly.v1+json, text/html
```

1.9 Conclusion

After the above analysis, you can see a simple use of the Spotify API.

As conclusions:

- The most popular year for Rush was 1981 and this popularity is related to the fact that it was the time when they released the most albums.
- Most Rush songs exceed 5 minutes in length and are not as popular as the shorter ones.
- As of 2020 Rush continued to release albums, but none of these had the popularity of Moving Pictures in 1981.

But...

- Why these following albums weren't so popular?
- Were there important changes in the structure of the songs?

- Why is less activity seen in the 90s?

To answer these questions, a more complete analysis must be made, which is shown in the second part.

2 Albums, dates and popularity

In this part I'm going to work with the `/albums/{id}` API section which return all the information Spotify has of an album. I use this data to answer questions like:

- Does the era affect the amount of tracks that an album had?
- The popularity of an album is it affected by the era?
- The number of tracks it's related to the popularity of the album?

Remember that the analysis of the first part was done with a small sample of artist data, so far we have the following partial conclusions:

- The most popular year for Rush was 1981 and this popularity is related to the fact that it was the time when they released the most albums.
- Most Rush tracks exceed 5 minutes in length and are not as popular as the shorter ones.
- As of 2020 Rush continued to release albums, but none of these had the popularity of Moving Pictures in 1981.

2.1 API

A small summary of the functionalities that I will use:

```
GET /albums/{id}/albums?market=US
```

Parameter	Type	Description
id (required)	`string`	The ID of the album

2.2 Libraries

Import the libraries I will need:

```
from pandas import json_normalize
import requests
import plotly.express as px
import plotly.io as pio
pio.renderers.default = "plotly_mimetype+notebook_connected"
import base64
from IPython.display import Image, display

def mm(graph):
    graphbytes = graph.encode("ascii")
    base64_bytes = base64.b64encode(graphbytes)
    base64_string = base64_bytes.decode("ascii")
    display(Image(url="https://mermaid.ink/img/" + base64_string))
```

This is the section of the authentication and generation of the token that was already explained in the first section

```
CLIENT_ID = '81e800e81ecf4997b5b9fb12efeb3ff2'
CLIENT_SECRET = '0e4364f440f148779d8a9f17976ecf1b'
def get_token():
    url = 'https://accounts.spotify.com/api/token'
    auth_response = requests.post(url, {
        'grant_type': 'client_credentials',
        'client_id': CLIENT_ID,
        'client_secret': CLIENT_SECRET,
    })
    if auth_response.status_code != 200:
        raise Exception('Error getting token')
    else:
        auth_response_data = auth_response.json()
        return auth_response_data['access_token']
access_token = get_token()
header = {
    'Authorization': 'Bearer {token}'.format(token=access_token),
    'accept':'application/json'
}
base_url = 'https://api.spotify.com/v1/'
```

To obtain a dataframe with the albums of an artist, a few previous steps must first be carried out:

1- Ask the user for the artists. 2- Get the artist ID 3- Extract all the albums of the artist.

Steps one and two were already explained in the first section:

```
#ask for user input for artist name
artist_name = input('Enter artist name: ')
print('the artist name is: ', artist_name)
#get artist id
artist_id = requests.get(base_url + 'search?q={}&type=artist'.format(artist_name), headers
```

```
the artist name is:  Rush
```

2.3 Workflow

The workflow from when the user enters the name of the artist, the request to get all de details and then saved in the dataframes is represented in the following diagram:

```
mm("""
graph TD
    A[user_input] -->|Search| B(GET item 0)
    B --> |save artist_id| D{GET request}
    D -->|/artist/artist_id/albums| E[df album list]
    D --> F[df albums details]
    F --> E
    E --> |/artist/album_id/| F
""")
```

```
<IPython.core.display.Image object>
```

2.4 Albums

In order to obtain the albums of an artist, I will use the `/artists/{id}/albums` API section. In the next part I use the API to get the info and a simple while loop for pagination. I will use the `limit` parameter to limit the number of albums per request to 50 and the `offset` parameter to move through the pages. I will also use the `include_groups` parameter to only get the albums and not the singles or compilations.

The structure of the json object delivered by `/artists/{id}/albums` can be found at this link. Link

```
#extract all albums of artist
r_albums = requests.get(base_url + 'artists/' + artist_id + '/albums?market=US', headers=h
r_albums=r_albums.json()
df_albums=json_normalize(r_albums['items'])
#get next page
while r_albums['next']:
    r_albums = requests.get(r_albums['next'], headers=header)
    r_albums=r_albums.json()
    df_albums=df_albums.append(json_normalize(r_albums['items']))
df_albums=df_albums.drop(['album_type',
                          'artists',
                          'href',
                          'images',
                          'release_date_precision',
                          'external_urls.spotify',
                          'uri',
                          'type'],axis=1)
df_albums['Release Date'] = df_albums['release_date'].apply(lambda x: x[:4])
df_albums.head(5)
```

	album_group	id	name	rel
0	album	2PBaIv21OWCmecNenZionV	Moving Pictures (40th Anniversary Super Deluxe)	202
1	album	06hsxtm7Y1gDM5sNliCD5d	Permanent Waves (40th Anniversary)	202
2	album	5G0G9TLLWr8n1abpY4ihmy	Hemispheres (40th Anniversary)	201
3	album	3bMJQ8LQWi42IAhVPP0M9O	A Farewell To Kings (40th Anniversary Deluxe E...	201
4	album	6q4SMJ8ggxBVrCzPSnDl7c	2112 (40 Anniversary)	201

Despite having used `include_groups`, Spotify's response contains some live albums, anniversary editions and others that generate noise in the data. Since my goal is to get only the studio albums, the first step is to delete the albums I don't need.

```
#delete live albums
df_albums=df_albums[~df_albums['name'].str.contains('Live|Anniversary|Remix|Tour|Stage|Ret
#loower case all names
df_albums['name']=df_albums['name'].str.lower()
df_albums.reset_index(drop=True, inplace=True)
df_albums.head(5)
```

	album_group	id	name	release_date	total_track
0	album	744i0LypfMwHHrKhzsqAx0	clockwork angels	2012-06-11	12
1	album	7hgcHQbB7xYr75qPPulfro	snakes & arrows	2007-04-27	13
2	album	0mT6ezOOTIUucAF9csghFE	feedback	2004-06-29	8
3	album	5fwkYtHrckROAs4ALRJ2Cz	test for echo (2004 remaster)	1996-09-06	11
4	album	6JNHWbFco4bnRP5ybKGriN	counterparts (2004 remaster)	1993-09-28	11

2.5 Popularity

In order to get the popularity of every album, I use the `/albums/{id}` API section and to concatenate in the dataframe.

```
#get the popularity of every album
df_albums['popularity'] = df_albums['id'].apply(lambda x: requests.get(base_url + 'albums/
df_albums.head(5)
```

	album_group	id	name	release_date	total_track
0	album	744i0LypfMwHHrKhzsqAx0	clockwork angels	2012-06-11	12
1	album	7hgcHQbB7xYr75qPPulfro	snakes & arrows	2007-04-27	13
2	album	0mT6ezOOTIUucAF9csghFE	feedback	2004-06-29	8
3	album	5fwkYtHrckROAs4ALRJ2Cz	test for echo (2004 remaster)	1996-09-06	11
4	album	6JNHWbFco4bnRP5ybKGriN	counterparts (2004 remaster)	1993-09-28	11

2.6 Graphs

The first graph shows the number of albums per year and the number of tracks per album.

```
#group by name and release date
df_tracks_and_year=df_albums.groupby(['name']).agg({'total_tracks':'first','Release Date':
#sort by release date
df_tracks_and_year=df_tracks_and_year.sort_values(by=['Release Date'])
#reset index
df_tracks_and_year.reset_index(drop=True, inplace=True)
#number of albums per year
fig = px.scatter(df_tracks_and_year,
                 x='Release Date',
                 y='total_tracks',
```

```
                    color='Release Date',
                    color_continuous_scale='Viridis',
                    marginal_x="histogram",
                    labels={'x':'Release year', 'y':'Number of Tracks'},
                    template="ggplot2" )
fig.update_layout(title='Number of Tracks per Album',
                     width=700,
                     height=600)
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

```
Unable to display output for mime type(s): application/vnd.plotly.v1+json, text/html
```

- Does the era affect the amount of tracks that an album had?

R:// Yes! The number of tracks per album has increased over time. The first album had 8 tracks and the last one 12 tracks.

```
#sotring by release date
df_albums=df_albums.sort_values(by=['Release Date'], ascending=True)
#heatmap of popularity, release date and number of tracks
fig = px.density_heatmap(df_albums,
                         x="Release Date",
                         y="popularity",
                         z="total_tracks",
                         #histfunc="max",
                         color_continuous_scale="Blues",
                         labels={'popularity':'Popularity', 'Release Date':'Release Date',
                         template="ggplot2")
fig.update_layout(title='Heatmap: Release date, popularity and number of tracks per album'
fig.layout['coloraxis']['colorbar']['title'] = 'Number of Tracks'
fig.update_layout(width=700, height=600)
fig.show()
```

```
Unable to display output for mime type(s): application/vnd.plotly.v1+json, text/html
```

- The popularity of an album is it affected by the era? R:// Yes! The popularity of an album is affected by the era. The most popular albums were released in the 70s.
- The number of tracks it's related to the popularity of the album? R:// Yes! The more tracks an album has, the less popular it is.

2.7 Conclusion

The most popular years of Rush were the 70s, in the beginning of the band. As time increased, the songs became shorter and the albums did not enjoy as much popularity.

3 Geolocation analysis

By now we know that Rush was a very popular band in the 70s and 80s mainly for their short songs, however, their focus was more towards long songs especially in their first albums. As time passed the band dedicated themselves to making not so long songs, but the albums had more and more.

Now, what can Spotify tell us about geolocation? In this part I will use Spotify's "markets" field to determine if Rush albums are in all countries.

```
GET /artist/{id}
GET /artist/{id}/albums
```

Parameter	Type	Description
id (required)	`string`	Get Spotify catalog information for a single artist identified by their unique Spotify ID.The ID of the artist.
albums	`string`	Get Spotify catalog information about an artist's albums.

3.1 Libraries

Import the libraries I will need:

```
from pandas import json_normalize
import requests
import plotly.express as px
import plotly.io as pio
pio.renderers.default = "plotly_mimetype+notebook_connected"
import pandas as pd
from pycountry_convert import country_alpha2_to_country_name, country_name_to_country_alph

import base64
from IPython.display import Image, display

def mm(graph):
```

```
    graphbytes = graph.encode("ascii")
    base64_bytes = base64.b64encode(graphbytes)
    base64_string = base64_bytes.decode("ascii")
    display(Image(url="https://mermaid.ink/img/" + base64_string))
```

This is the section of the authentication and generation of the token that was already explained in the first section

```
CLIENT_ID = '81e800e81ecf4997b5b9fb12efeb3ff2'
CLIENT_SECRET = '0e4364f440f148779d8a9f17976ecf1b'
def get_token():
    url = 'https://accounts.spotify.com/api/token'
    auth_response = requests.post(url, {
        'grant_type': 'client_credentials',
        'client_id': CLIENT_ID,
        'client_secret': CLIENT_SECRET,
    })
    if auth_response.status_code != 200:
        raise Exception('Error getting token')
    else:
        auth_response_data = auth_response.json()
        return auth_response_data['access_token']
access_token = get_token()
header = {
    'Authorization': 'Bearer {token}'.format(token=access_token),
    'accept':'application/json'
}
base_url = 'https://api.spotify.com/v1/'
```

```
#ask for user input for artist name
artist_name = input('Enter artist name: ')
print('the artist name is: ', artist_name)
#get artist id
artist_id = requests.get(base_url + 'search?q={}&type=artist'.format(artist_name), headers
```

```
the artist name is:  Rush
```

3.2 Workflow

The workflow from when the user enters the name of the artist, the request to get all de details and then transform the two-letters country code to the full name of the country is the following:

```
mm("""
graph TD
    A[user_input] -->|Search| B(GET item 0)
    B --> |save artist_id| D{GET request}
    D -->|/artist/artist_id/albums| E[df album list and available_markets]
    E --> |Transform | F[df available_markets in three-letter standard standard]
""")
```

```
<IPython.core.display.Image object>
```

3.3 Get the data

To get all the IDs and markets available for albums under the Rush name, I have to make a request to /albums.

As in the previous sections I will remove some columns that I don't need and I will extract the year of publication in a separate column.

```
#extract all albums of artist
r_albums = requests.get(base_url + 'artists/' + artist_id + '/albums', headers=header, par
r_albums=r_albums.json()
df_albums=json_normalize(r_albums['items'])
#get next page
while r_albums['next']:
    r_albums = requests.get(r_albums['next'], headers=header)
    r_albums=r_albums.json()
    df_albums=df_albums.append(json_normalize(r_albums['items']))
df_albums=df_albums.drop(['album_type',
                          'artists',
                          'href',
                          'images',
                          'release_date_precision',
                          'external_urls.spotify',
                          'uri',
```

```
                            'type'],axis=1)
df_albums['Release Date'] = df_albums['release_date'].apply(lambda x: x[:4])
df_albums.head(5)
```

	album_group	available_markets	id
0	album	[CA, JP]	5nZ5I0gA3x6KEkIpHQWw4l
1	album	[AD, AE, AG, AL, AM, AO, AR, AT, AU, AZ, BA, B...	2PBaIv21OWCmecNenZionV
2	album	[CA, JP]	3EUhoI6JRdxYzml9gHWzJI
3	album	[AD, AE, AG, AL, AM, AO, AR, AT, AU, AZ, BA, B...	06hsxtm7Y1gDM5sNliCD5d
4	album	[CA]	0UQOn626iDanxtIZlnQyUK

3.4 How many markets can each album have?

```
#COUNT NUMBER OF ALBUMS AVAILABLE per MARKET
df_albums['count_available_markets'] = df_albums['available_markets'].apply(lambda x: len(
#sort count_available_markets
df_albums = df_albums.sort_values(by=['Release Date'], ascending=True)
df_albums = df_albums.reset_index(drop=True)
df_albums.head(5)
```

	album_group	available_markets	id
0	album	[AD, AE, AL, AM, AO, AR, AT, AU, AZ, BA, BD, B...	43OvqHDAEOUKfHNFPCgsvf
1	album	[JP]	5foiAR3bxvhZ2660J8Nntg
2	album	[CA]	0lZRCf7prVEVVYjH5Im0TS
3	album	[AD, AE, AG, AL, AM, AO, AR, AT, AU, AZ, BA, B...	57ystaP7WpAOxvCxKFxByS
4	album	[JP]	4FZkxs4KHgwYJQ94cWMFqz

The following graph shows:

- How many releases are there for each album
- In how many markets is it available

They are organized by year of release

```
#density plot between release date and count_available_markets
fig = px.density_heatmap(df_albums,
                         x="name",
                         y="count_available_markets",
```

```
                          marginal_x="histogram",
                          template="ggplot2")
fig.update_layout(title='Top Tracks and Albums of {}'.format(artist_name),
                  width=1500,
                  height=800,
                  xaxis_title='Albums',
                  yaxis_title='Number of Available Markets')
fig.update_layout(width=700, height=600)
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

```
Unable to display output for mime type(s): application/vnd.plotly.v1+json, text/html
```

3.5 What is the market with more albums?

There are several points here: I must take into account that Spotify returns me a list of countries per album, I must first extract that list and convert it into a dataframe. In the end I will have a line for each album and market available.

```
df_available_markets=pd.DataFrame(df_albums['available_markets'].apply(lambda x: pd.Series
#add the album name to the dataframe
df_available_markets['album_name'] = df_albums['name']
df_available_markets=df_available_markets.reset_index()
df_available_markets.head(5)
```

	index	available_markets	album_name
0	0	AD	The First Us Tours
1	0	AE	The First Us Tours
2	0	AL	The First Us Tours
3	0	AM	The First Us Tours
4	0	AO	The First Us Tours

Second point, to be able to work by market/country, I must group the data using this parameter.

```
#count number of albums per country
df_country = df_available_markets.groupby(['available_markets']).count()
```

```
df_country = df_country.reset_index()
df_country = df_country.rename(columns={'index':'count'})
df_country = df_country.sort_values(by=['count'], ascending=False)
df_country = df_country.reset_index(drop=True)
#drop album_name column
df_country = df_country.drop(['album_name'], axis=1)
df_country.head(5)
```

	available_markets	count
0	CA	43
1	DM	40
2	GT	40
3	TT	40
4	NI	40

Third point, Kosovo is a country recognized by Spotify, but not by the international standard which Plotly uses, so to avoid errors, I remove it.

```
#drop the XK country
df_country = df_country.drop(df_country[df_country['available_markets'] == 'XK'].index)
df_country = df_country.reset_index(drop=True)
```

And finally, Spotify delivers the countries in a two-letter standard, but Plotly works with a three-letter standard, to solve this problem I use the pycountry-convert library

```
#add country name to dataframe
df_country['country_name'] = df_country['available_markets'].apply(lambda x: country_alpha
df_country.head(5)
```

	available_markets	count	country_name
0	CA	43	Canada
1	DM	40	Dominica
2	GT	40	Guatemala
3	TT	40	Trinidad and Tobago
4	NI	40	Nicaragua

```
#graph
fig = px.choropleth(df_country, locations="country_code",
```

```
                    color="count", # lifeExp is a column of gapminder
                    hover_name="country_name", # column to add to hover information
                    color_continuous_scale=px.colors.sequential.Plasma,
                    template="ggplot2")
fig.update_layout(title='Country distribution by {}'.format(artist_name))
fig.update_layout(width=700, height=600)
fig.show()
```

```
Unable to display output for mime type(s): application/vnd.plotly.v1+json, text/html
```

The graph above shows several things:

- The country with the most albums available is Canada, it's not surprising given that Rush is a Canadian band.
- Most countries have the same number of albums.
- And apparently Spotify is not present in certain countries where the account is 0.

Part II

Analyzing my listening patterns

4 Getting and cleaning the data

4.1 Requesting the data from Spotify

In this part, I will analyze my personal data that Spotify has collected in the last 9 years. In order to do this you first need the data, this must be requested from Spotify in the privacy section of your account, here you will have three options:

- Account data
- Streaming history
- Technical information

For this post I chose to do it with my streaming history. Spotify took about a week to deliver the data.

4.2 Cleaning the data

4.2.1 JSON Files

Spotify will deliver several json files which can be directly imported into PowerBI for analysis, however this is something I am not comfortable with and I decided to make a small script to transform them into a single csv file.

```
#convert json to csv
import json
import csv
#open json files
for i in range(0, 10):
    with open('endsong_{}.json'.format(i)) as f:
        data = json.load(f)
#json to dataframe
import pandas as pd
df = pd.DataFrame(data)
df.size
#convert dataframe to csv
```

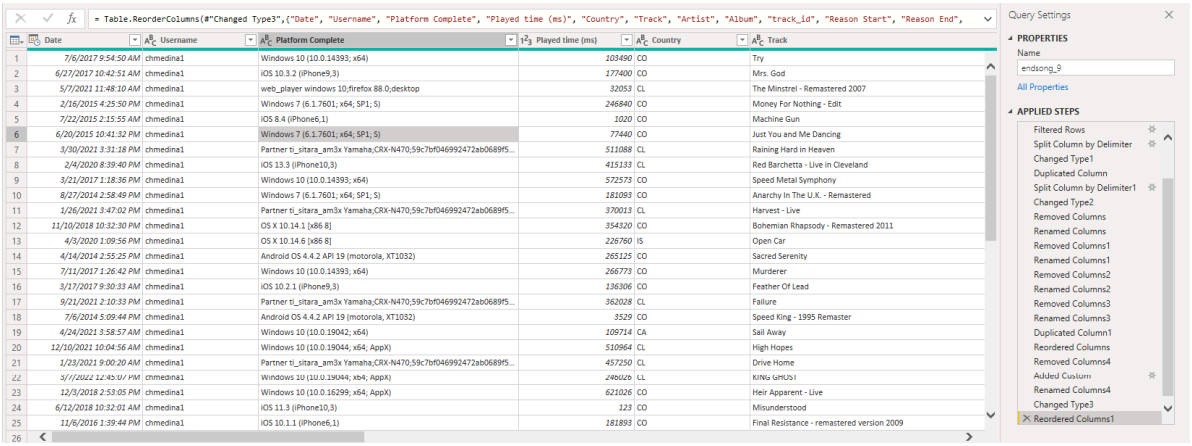
```
df.to_csv('endsong_{}.csv'.format(i), index=False)
```

4.2.2 Importing the data

After having the csv file, it is imported into Power BI and the steps I followed for this were:

1. Delete podcast related data as it is not relevant to me.
2. Rename columns appropriately
3. The "platform" column is divided by space to obtain the platform without its version, for example, from having "Windows 7 (6.1.7601; x64; SP1; S)" it becomes only "Windows"
4. Create a new column where I transformed the duration of the songs from milliseconds to minutes.

In the end my table looks like this:



	Date	Username	Platform Complete	Played time (ms)	Country	Track
1	7/6/2017 9:54:50 AM	chmedina1	Windows 10 (10.0.14393; x64)	103490	CO	Try
2	6/27/2017 10:42:51 AM	chmedina1	iOS 10.3.2 (iPhone9,3)	177400	CO	Mrs. God
3	5/7/2021 11:48:10 AM	chmedina1	web_player windows 10;firefox 88.0;desktop	32053	CL	The Minstrel - Remastered 2007
4	2/16/2015 4:25:50 PM	chmedina1	Windows 7 (6.1.7601; x64; SP1; S)	246840	CO	Money For Nothing - Edit
5	7/22/2015 2:15:55 AM	chmedina1	iOS 8.4 (iPhone6,1)	1020	CO	Machine Gun
6	6/20/2015 10:41:32 PM	chmedina1	Windows 7 (6.1.7601; x64; SP1; S)	77440	CO	Just You and Me Dancing
7	3/30/2021 3:31:18 PM	chmedina1	Partner ti_sitara_am3x Yamaha;CRX-N470;59c7bf046992472ab06895...	511088	CL	Raining Hard in Heaven
8	2/4/2020 8:39:40 PM	chmedina1	iOS 13.3 (iPhone10,3)	415133	CL	Red Barchetta - Live in Cleveland
9	3/21/2017 1:18:36 PM	chmedina1	Windows 10 (10.0.14393; x64)	572573	CO	Speed Metal Symphony
10	8/27/2014 2:58:49 PM	chmedina1	Windows 7 (6.1.7601; x64; SP1; S)	181093	CO	Anarchy In The U.K. - Remastered
11	1/26/2021 3:47:02 PM	chmedina1	Partner ti_sitara_am3x Yamaha;CRX-N470;59c7bf046992472ab06895...	370013	CL	Harvest - Live
12	11/10/2018 10:32:30 PM	chmedina1	OS X 10.14.1 [x86 8]	354320	CO	Bohemian Rhapsody - Remastered 2011
13	4/3/2020 1:09:56 PM	chmedina1	OS X 10.14.6 [x86 8]	226760	IS	Open Car
14	4/14/2014 2:55:25 PM	chmedina1	Android OS 4.4.2 API 19 (motorola, XT1032)	265125	CO	Sacred Serenity
15	7/11/2017 1:26:42 PM	chmedina1	Windows 10 (10.0.14393; x64)	266773	CO	Murderer
16	3/17/2017 9:30:33 AM	chmedina1	iOS 10.2.1 (iPhone9,3)	136306	CO	Feather Of Lead
17	9/21/2021 2:10:33 PM	chmedina1	Partner ti_sitara_am3x Yamaha;CRX-N470;59c7bf046992472ab06895...	362028	CL	Failure
18	7/6/2014 5:09:44 PM	chmedina1	Android OS 4.4.2 API 19 (motorola, XT1032)	3529	CO	Speed King - 1995 Remaster
19	4/24/2021 3:58:57 AM	chmedina1	Windows 10 (10.0.19042; x64)	109714	CA	Sail Away
20	12/10/2021 10:04:56 AM	chmedina1	Windows 10 (10.0.19044; x64; AppX)	510964	CL	High Hopes
21	1/23/2021 9:00:20 AM	chmedina1	Partner ti_sitara_am3x Yamaha;CRX-N470;59c7bf046992472ab06895...	457250	CL	Drive Home
22	3/7/2022 12:45:07 PM	chmedina1	Windows 10 (10.0.19044; x64; AppX)	246026	CL	KING GHOST
23	12/3/2018 2:53:05 PM	chmedina1	Windows 10 (10.0.16299; x64; AppX)	621026	CO	Heir Apparent - Live
24	6/12/2018 10:32:01 AM	chmedina1	iOS 11.3 (iPhone10,3)	123	CO	Misunderstood
25	11/6/2016 1:39:44 PM	chmedina1	iOS 10.1.1 (iPhone6,1)	181893	CO	Final Resistance - remastered version 2009
26						

Figure 4.1: Spotify Dataset 1 Preview

For now I don't need more, so I can start creating the dashboard with PowerBi.

5 Dashboard template creation

To create a dashboard in PowerBi there are many steps and how to execute them can vary depending on the analyst, I personally like to create the aesthetic part of the dashboard first and then create the graphs.

5.1 Template creation

In this I chose to create a basic template which contains:

- Logo
- Background image
- Shapes to add transparencies in the colors
- Basic KPI of my musical tendencies.

The template is created in a separate page and then I can use it in the other pages. It finally looks like this:

(I am a colorblind person, please bear with me.)

The KPIs shown were created with the distinct count, for example, if an artist was listened to many times, it would only be counted once.

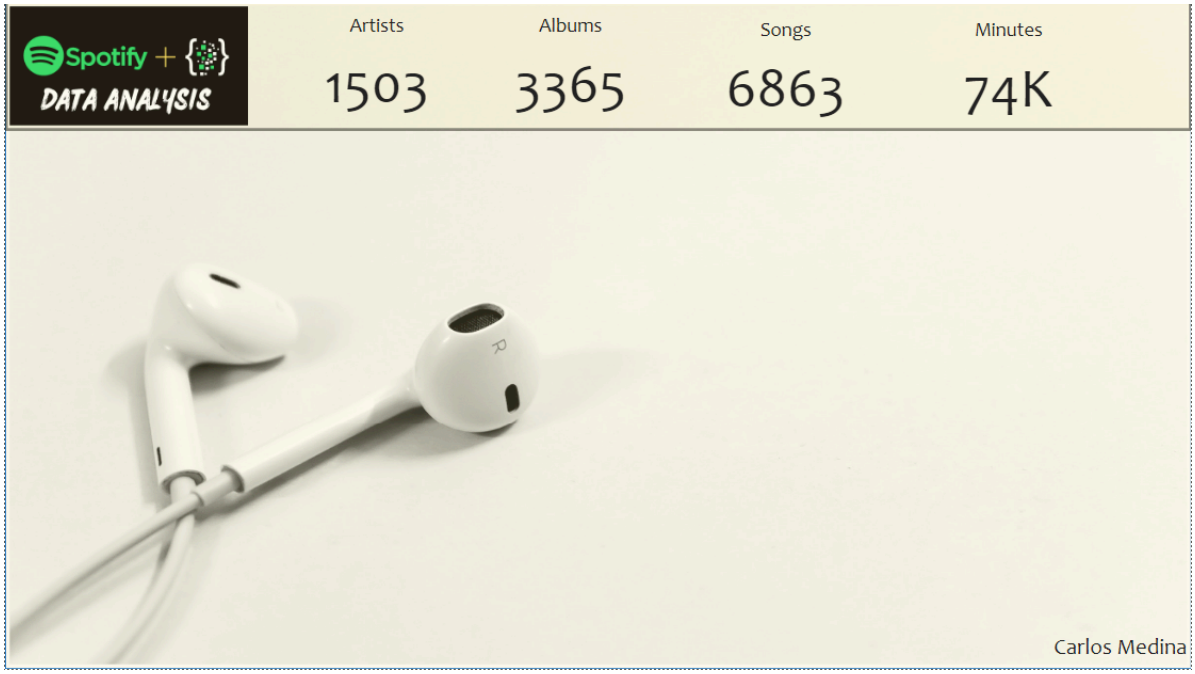


Figure 5.1: PowerBi Template

6 Exploratory analysis

Now that I have the template created, I start to explore and analyze the data to answer certain questions like:

- What is the platform that I have used the most to listen?
- How my edge tendencies have changed over the years.
- Who are my favorite artists?
- It will be possible to analyze my state of mind through the years. ?

I have decided to split the data as follows:

- **Overall** : General data of my listening patterns.
- **Platform and Geo** : Analysis of the devices that I have used to listen to music and the geolocations where I have done it.
- **Artists** : Analysis of artists over time.
- **Listening time** : My favorite artist will be the same one that I listen to the most?

6.1 Overall

6.1.1 Listening trend over the years - Line and clustered column chart

In this graph I can see relevant points for me:

- I started paying for Spotify at the end of 2013, I think this date coincides with the arrival date of the service in Colombia, which was where I lived at the time.
- Trends were always on the rise, every year there were more artists and therefore more albums to listen to.
- In the years 2020 and 2021, the years of the pandemic were where I listened music the most, this is likely due to the long periods of quarantine to which the city where I live was subjected.

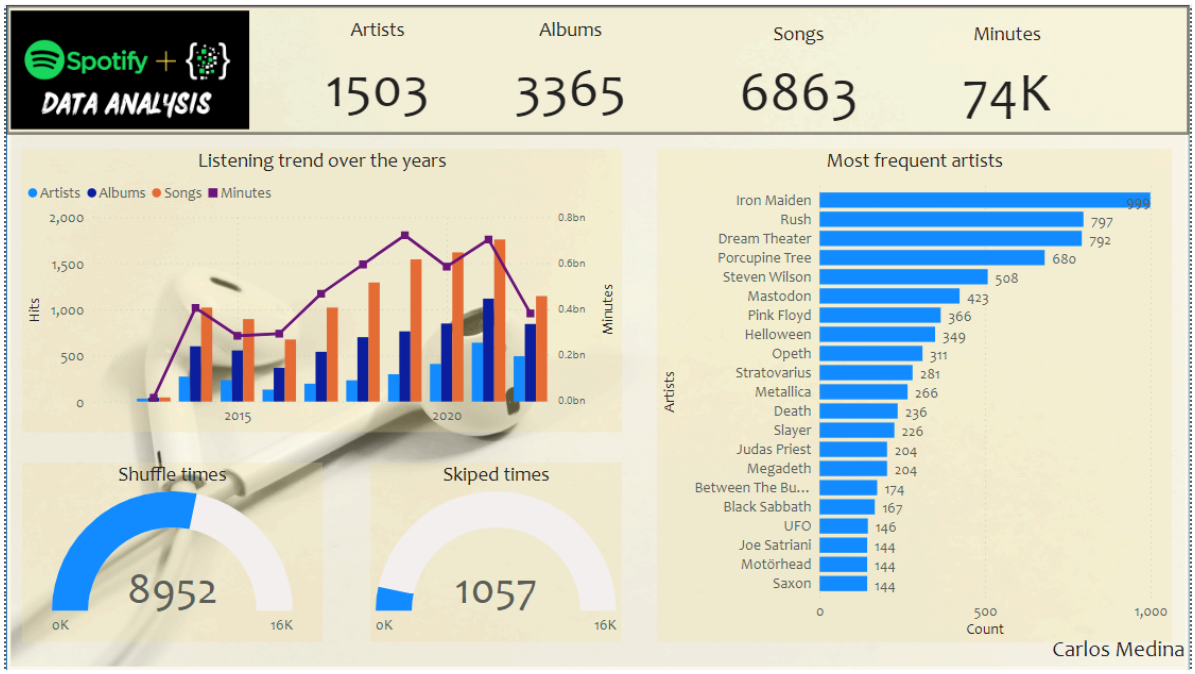


Figure 6.1: Page Overall

6.1.2 Shuffle and skipped times - Gauge

Here I show how of all the records I have, I have listened to more than half in random mode, but very rarely do I skip a song. I think this is directly correlated with the upward trends, since my favorite Spotify lists are the personalized mixes that the platform creates for you, and these mixes give you the opportunity to listen more and more artists according to your tastes.

6.1.3 Most frequent artists - Stacked bar char

The information in this graph is quite clear, the artists that I have listened to the most in these 9 years.

6.2 Platform and Geo

6.2.1 Trend of platform over the years - Ribbon chart

- The type of device that I use the most from 2013 to 2019 was a Windows PC. This is related to the fact that the company where I worked in that period provided me with

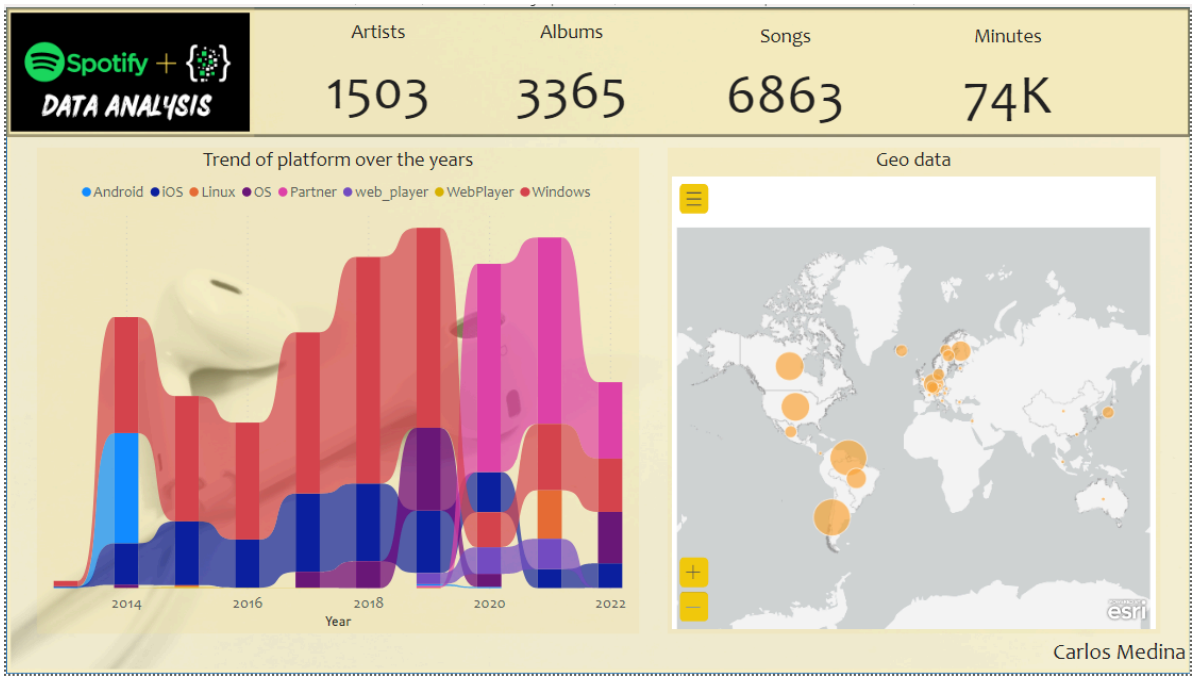


Figure 6.2: Page Platform and Geo

this OS.

- For 2020 there are two important changes, the first change is that I buy a sound system which connects directly to Spotify and I hardly use it on the PC anymore.
- The second important change in 2020 is that Mac OS (my PC) begins to have more relevance.
- 2020 and 2021 my sound system had more use since those were the years of the pandemic and I spent almost all my time at home.
- The last time I used an Android device was in 2014.

6.2.2 Geo Data - ArcGIS maps

A self explanatory graph. My main listening places.

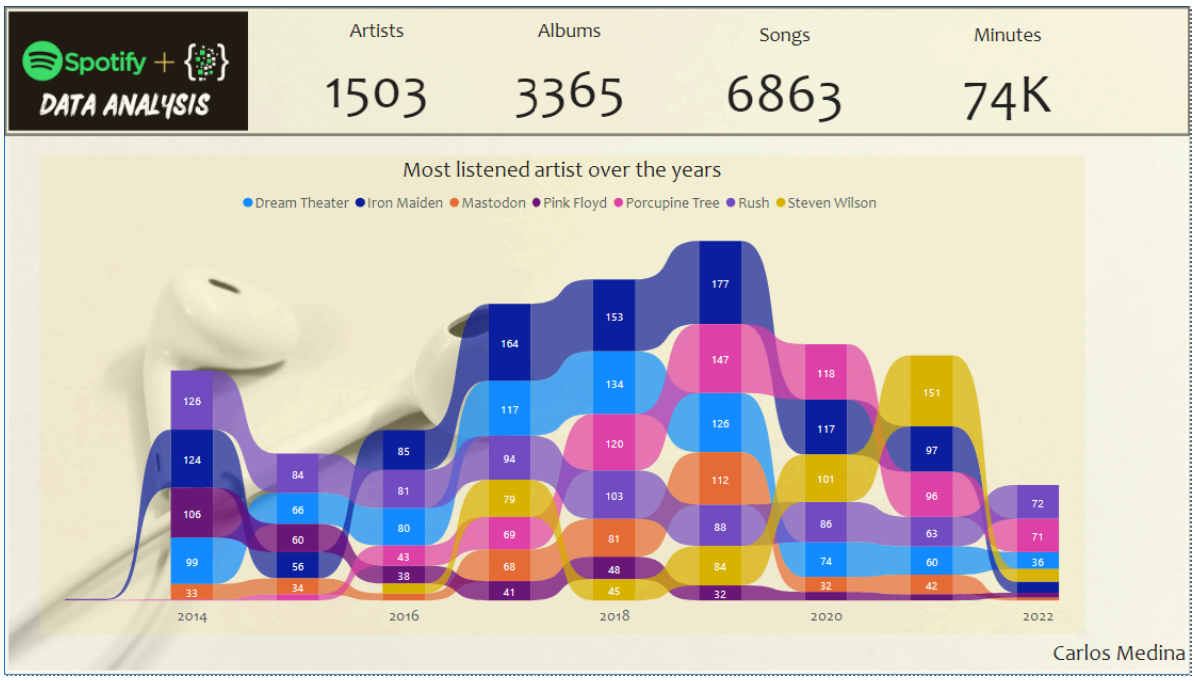


Figure 6.3: Artists

6.3 Artists

6.3.1 Most listened artist over the years - Ribbon chart

In the Overall section I showed the artists that I listen to the most, so I decided to see how they behaved over time, the graph is easy to read and Iron Maiden is always first, except for the last two years where Steven Wilson and his mutation in Porcupine Tree has taken the lead.

However, the most listened to artist is the one that I have dedicated the most time to?

6.4 Listening time

6.4.1 Played time over the years - Scatter chart

Iron Maiden is the artist I've heard the most songs from, however, it's nowhere near as long as I've heard Dream Theater. The difference between the most listened to artist and the longest listening artist should be understood very well.

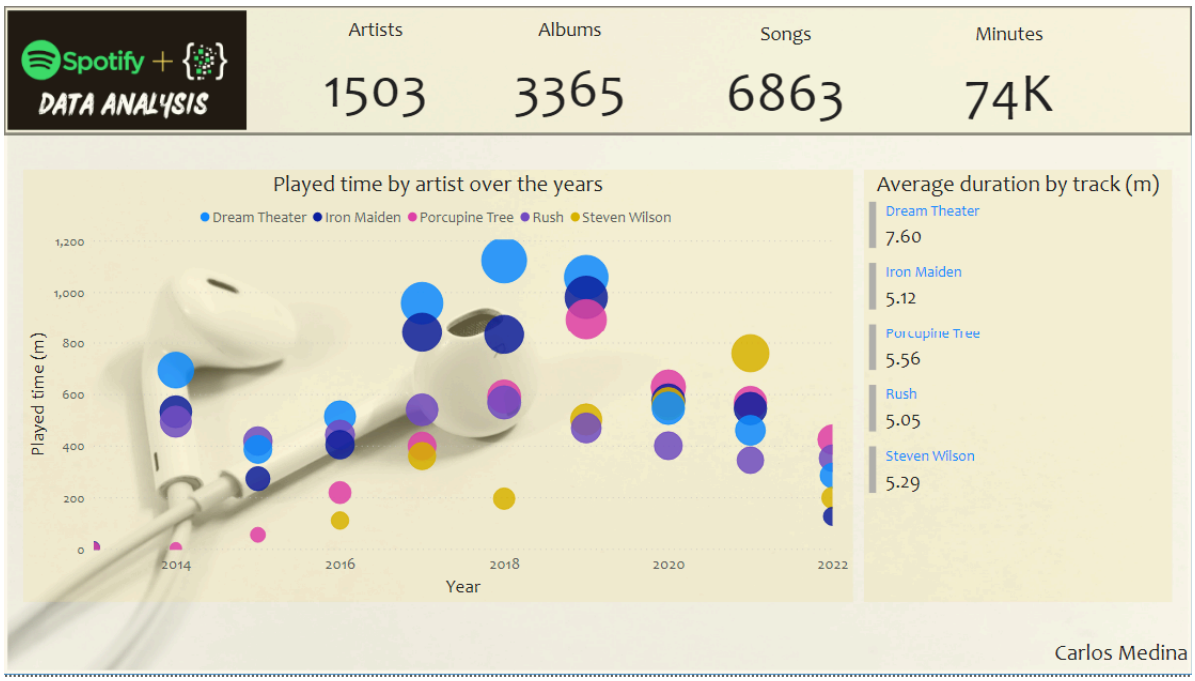


Figure 6.4: Listening time

6.4.2 Average duration by track (m) - Multi-row card

To analyze this phenomenon a little more, here I show the average length of a song by these artists, and Dream Theater wins with an average length of seven and a half minutes.

7 PowerBI and python

I think that despite having all the data, there is still a lack of it that Spotify can give me about my listening habits. For this I will use the API to extract the audio features of the songs, the extracted data will be the following:

- Instrumentalness
- Acousticness
- Danceability
- Energy
- Liveness
- Speechiness

Documentation of these factors can be found here

To extract this data I use the following python script.

```
import pandas as pd
import requests
# Function to get the token
def get_token():
    url = 'https://accounts.spotify.com/api/token'
    auth_response = requests.post(url, {
        'grant_type': 'client_credentials',
        'client_id': 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
        'client_secret': 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    })
    if auth_response.status_code != 200:
        raise Exception('Error getting token')
    else:
        auth_response_data = auth_response.json()
        return auth_response_data['access_token']
# Get token
access_token = get_token()
header = {
    'Authorization': 'Bearer {token}'.format(token=access_token),
    'accept':'application/json'
```

```
}
base_url = 'https://api.spotify.com/v1/audio-features/'
dataframe=pd.read_csv('endsong_9 copy.csv')
index_startpoint=0
#get the audio features for each song strating from the index_startpoint
for index, row in dataframe.iterrows():
    if index >= index_startpoint:
        url = base_url + row['track_id']
        response = requests.get(url, headers=header)
        if response.status_code != 200:
            print('Error getting data')
        else:
            response_data = response.json()
            dataframe.loc[index, 'acousticness'] = response_data['acousticness']
            dataframe.loc[index, 'danceability'] = response_data['danceability']
            dataframe.loc[index, 'energy'] = response_data['energy']
            dataframe.loc[index, 'instrumentalness'] = response_data['instrumentalness']
            dataframe.loc[index, 'liveness'] = response_data['liveness']
            dataframe.loc[index, 'loudness'] = response_data['loudness']
            dataframe.loc[index, 'speechiness'] = response_data['speechiness']
            dataframe.loc[index, 'tempo'] = response_data['tempo']
            dataframe.loc[index, 'valence'] = response_data['valence']
            dataframe.loc[index, 'duration_ms'] = response_data['duration_ms']
            dataframe.loc[index, 'time_signature'] = response_data['time_signature']
            dataframe.loc[index, 'key'] = response_data['key']
            dataframe.loc[index, 'mode'] = response_data['mode']
            #save log in file
            with open('log.log', 'a') as f:
                f.write('Song: ' + str(index) + ' - ' + dataframe['track_id'][index])
#save the dataframe in a csv file
dataframe.to_csv('Audio_Features.csv', index=False)
```

This resulting dataframe must be imported into PowerBi and the relationships with the previous dataset must be created:

7.0.1 Radar Chart

With the help of some radar charts, I can show the variance of these values over the years. The final result is the following:

If you want to download the PowerBI file (pbix) you can do it from this link

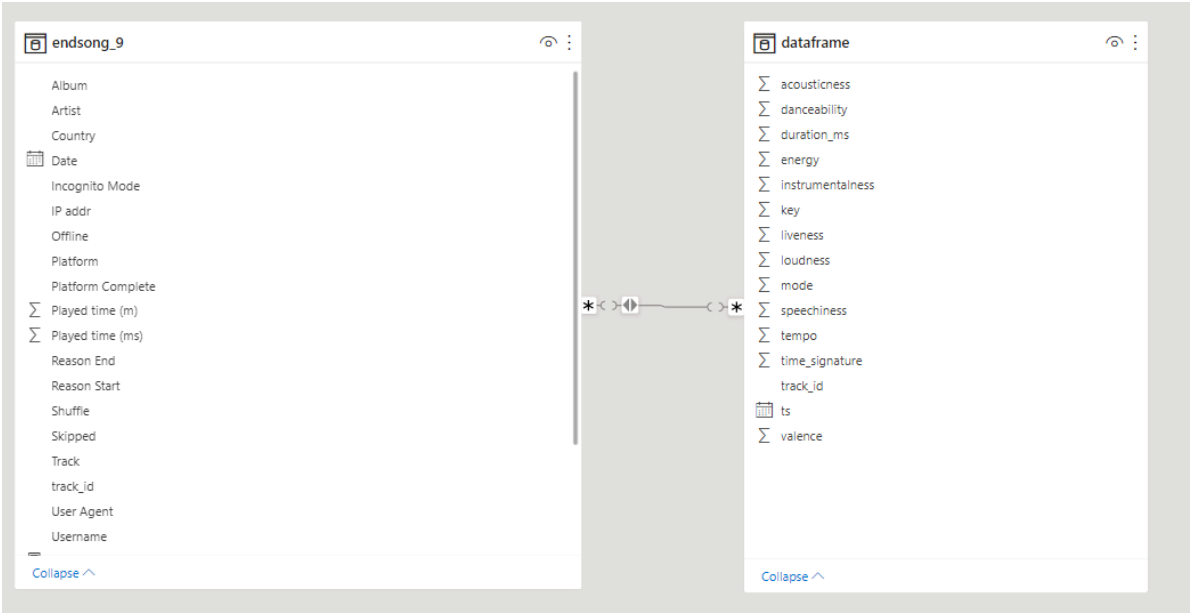


Figure 7.1: Second DF

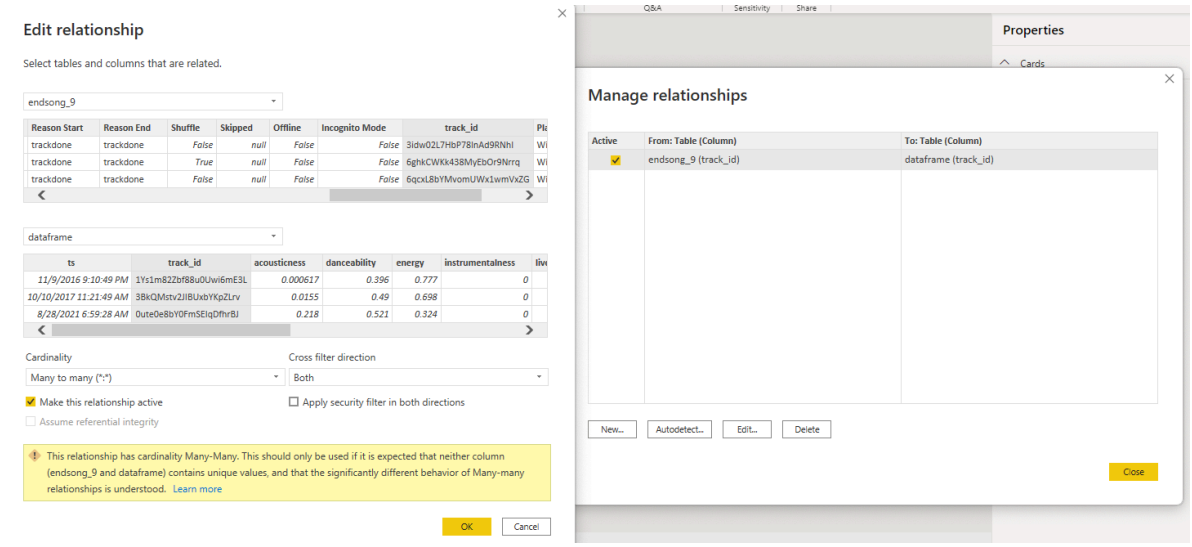


Figure 7.2: Relationships

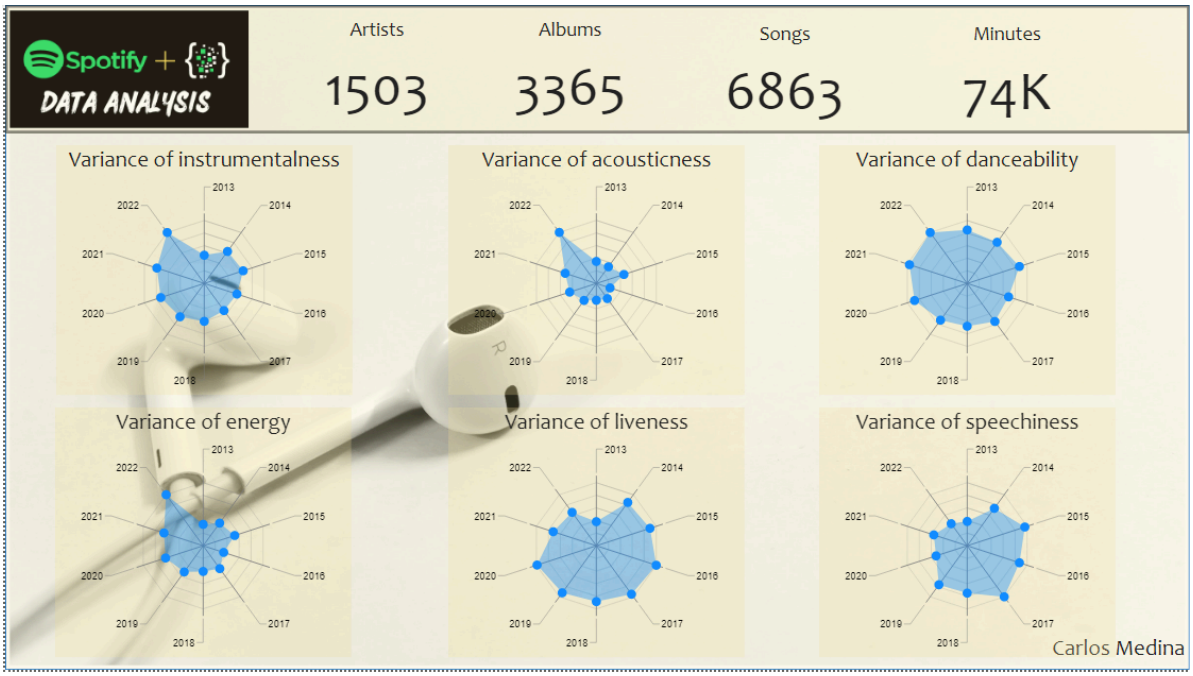


Figure 7.3: Song audio features

Part III

Music trends in south america

8 Music trends in south america

coming soon

9 Version history

Version 2:

Date: Nov. 1.

Pages: No Changes

Details: Flow charts on current pages.

Version 1:

Date: Oct. 29.

Pages:

- An artist's analysis
- Albums, dates and popularity
- Geolocation analysis