

# Visualization of Massive Data

# **Analysis and prediction on Spotify Music data**

---

Antoine Cauquil & Diesen Nwoumga

2 feb. 2020



## Introduction

1. The Spotify dataset
2. Libraries used
3. Data analysis
4. Popularity prediction based on musical features

---

## The Spotify dataset

For this project, we wanted to apply what we learnt in data analysis to music. We chose the largest Spotify dataset available on Kaggle: Spotify Tracks DB ([here](#)).

This dataset is made of 232k tracks gathered from the Spotify API and provides us with some typical fields in music databases (*genre, artist\_name, track\_name, track\_id, duration\_ms, popularity*) coupled with a lot of interesting fields (*acousticness, danceability, energy, instrumentalness, key, liveness, loudness, mode, speechiness, tempo, time\_signature, valence*). You can find more information about those sound properties (called “Features”) on the [Spotify API Reference](#).

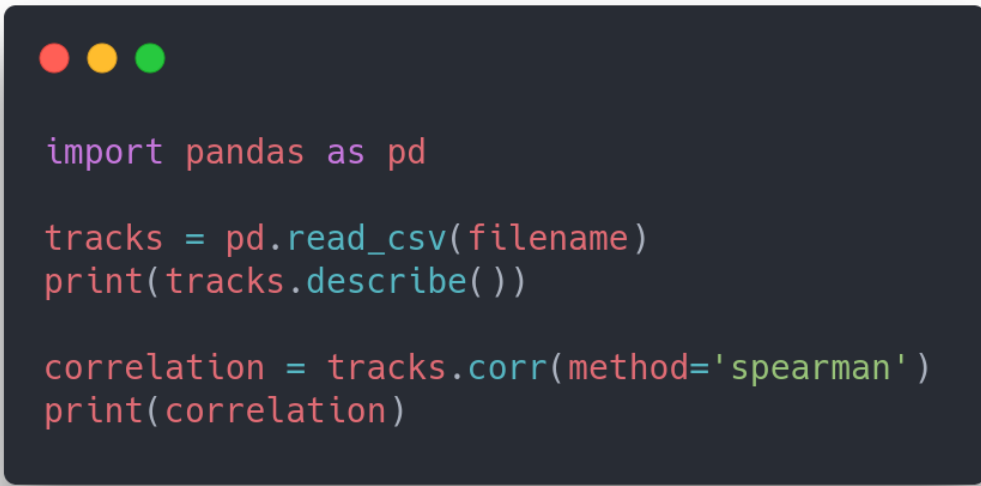
## Libraries used

This project is written in Python 3.8 and includes the following libraries:

- [pandas](#): Data structures and analysis
- [NumPy](#): Matrix operations
- [Matplotlib](#): Data visualization
- [seaborn](#): Data visualization
- [scikit-learn \(sklearn\)](#): Machine learning tools

## Data analysis

The first thing we do is opening the dataset into a pandas DataFrame and use the methods *describe* to get an overview of the dataset and *corr* to generate the correlation matrix of the columns. For the last one, we use the Spearman method because it is best suited for monotonic (linear and non-linear) relationship.

A dark-themed terminal window with three colored window control buttons (red, yellow, green) in the top-left corner. It contains Python code for data analysis using pandas.

```
import pandas as pd

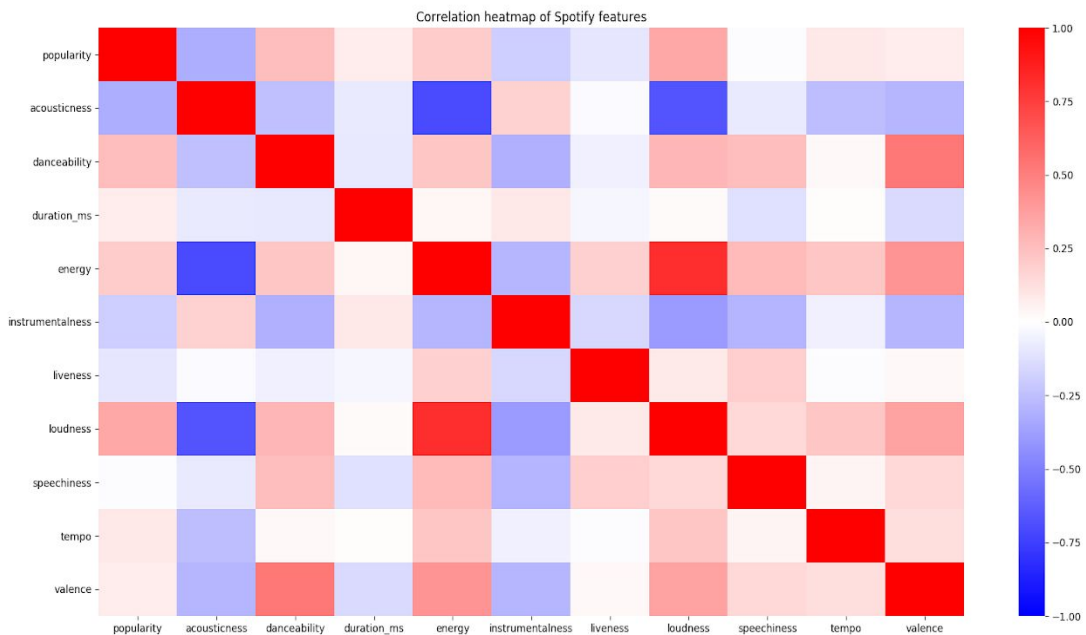
tracks = pd.read_csv(filename)
print(tracks.describe())

correlation = tracks.corr(method='spearman')
print(correlation)
```

We then use matplotlib to generate a visual representation of the correlation matrix. Because the Spearman correlation coefficient is 0-centered, we set *vmin* and *vmax* to 1 and a diverging colormap.

```
import matplotlib.pyplot as plt

plt.figure(figsize=(20, 10))
plt.title('Correlation heatmap of Spotify features')
sns.heatmap(correlation, vmin=-1, vmax=1, cmap="bwr")
# plt.show()
plt.savefig("output/correlation")
plt.close()
```



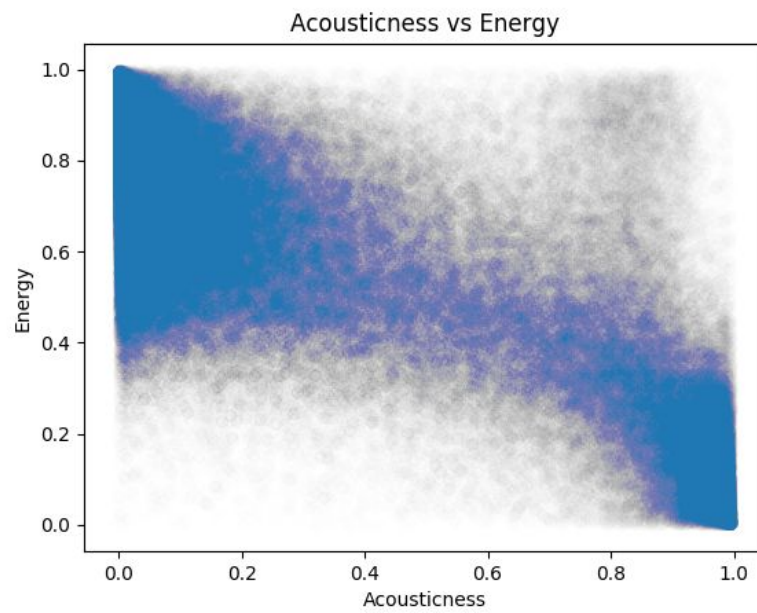
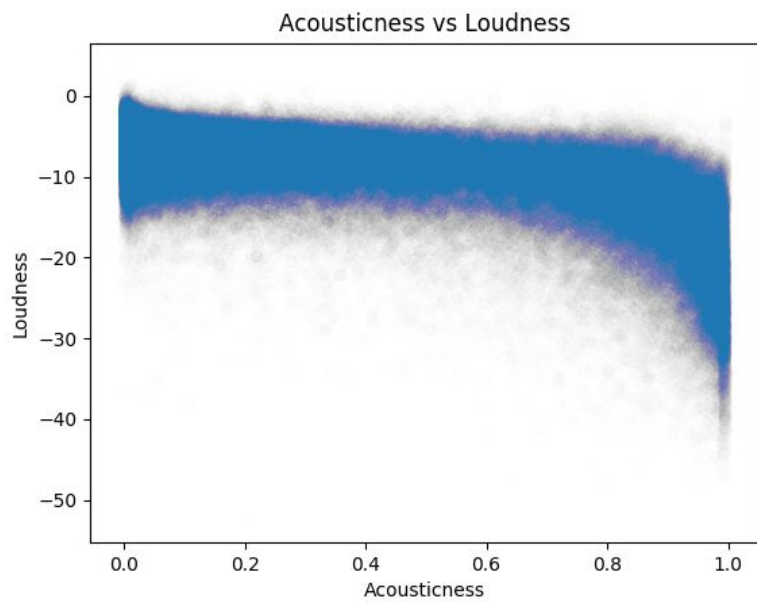
---

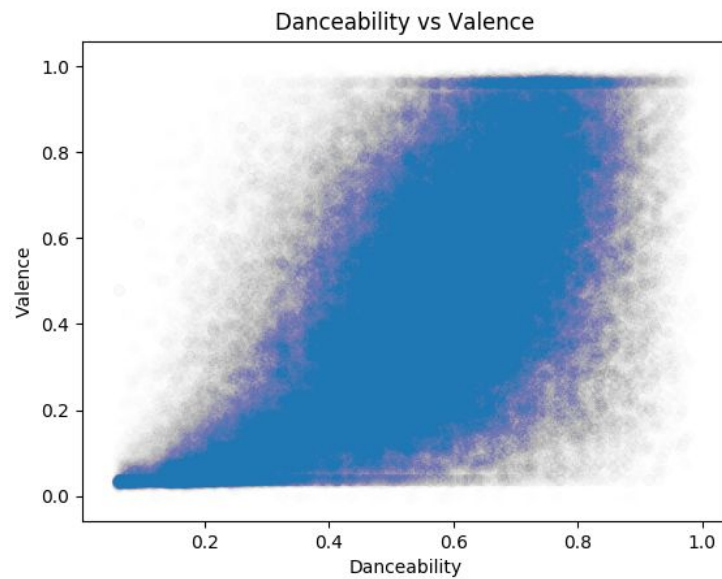
Thanks to this graph, we can see which features are related and focus on them during the next phase of the analysis:

- Acousticness vs. Energy
- Acousticness vs. Loudness
- Danceability vs. Valence

For each pair of feature, we plot the first one on the x axis and the second one on the y axis on a scatter plot. Since we have a lot of data, we use a low alpha value to be able to see the concentration better.

```
plt.title("Acousticness vs Loudness")
plt.scatter(tracks["acousticness"], tracks["loudness"], alpha=0.005)
plt.xlabel("Acousticness")
plt.ylabel("Loudness")
# plt.show()
plt.savefig("output/acousticness_vs_loudness")
plt.close()
```





These plots reflect pretty interesting properties of music: the more acoustic a track, the quieter it tends to be and the less energy the track will convey. The last graph tells us that a danceable track will feel more positive (e.g. happy, cheerful, euphoric) to the listener.



Then we tried to see if we could find some interesting statistics about the [mode](#) of a track. We splitted our dataset between *Major* and *Minor* and for each feature, we plotted both in a histogram with the number of tracks as y and the feature as x.

```
import numpy as np

metas = ["popularity", "duration_ms", "tempo", "key",
        "mode", "time_signature", "tempo", "loudness"]

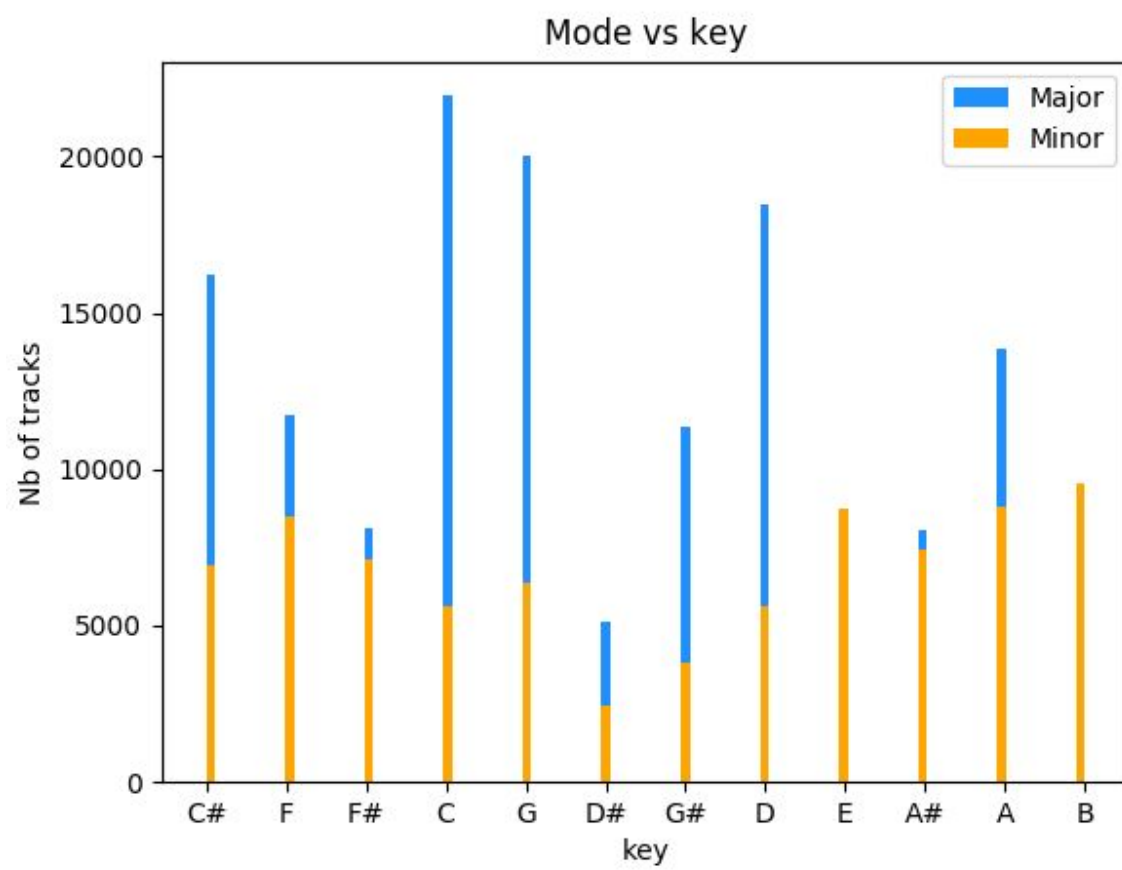
features = ["acousticness", "danceability", "energy",
            "instrumentalness", "liveness", "speechiness", "valence"]

x1 = tracks.loc[np.where(tracks["mode"] == "Major")]
x2 = tracks.loc[np.where(tracks["mode"] == "Minor")]

kwargs = dict(alpha=1, bins=100)

for f in features + metas:
    plt.hist(x1[f], **kwargs, color='dodgerblue', label='Major')
    plt.hist(x2[f], **kwargs, color='orange', label='Minor')
    plt.gca().set(title='Mode vs ' + f, xlabel=f, ylabel="Nb of tracks")
    plt.legend()
    # plt.show()
    plt.savefig("output/mode_vs/" + f)
    plt.close()
```

Using this method we found that even if Antonio Vivaldi used the E major scale for the "Spring" concerto from The Four Seasons, it is very rarely used. We found the same pattern for the B major scale as well as the F# and A# major to a lesser extent.



The last analysis that we've done is a genre-based repartition of features. To achieve this goal, we extracted each genre available in the dataset and computed the median value of each feature of the tracks present in the given genre. At the end of the following instructions, we will have two lists: *genre\_names* containing the names of all the genres and indexed in the same order *genre\_meds* contains the median value for each feature of a genre.

```
import numpy as np

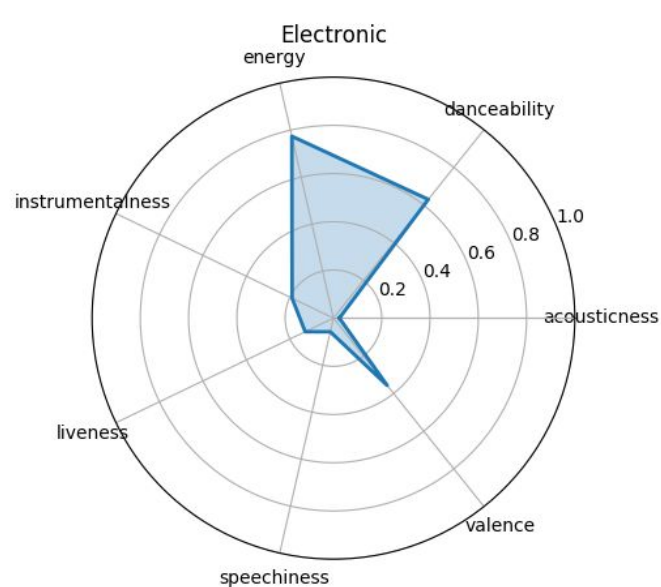
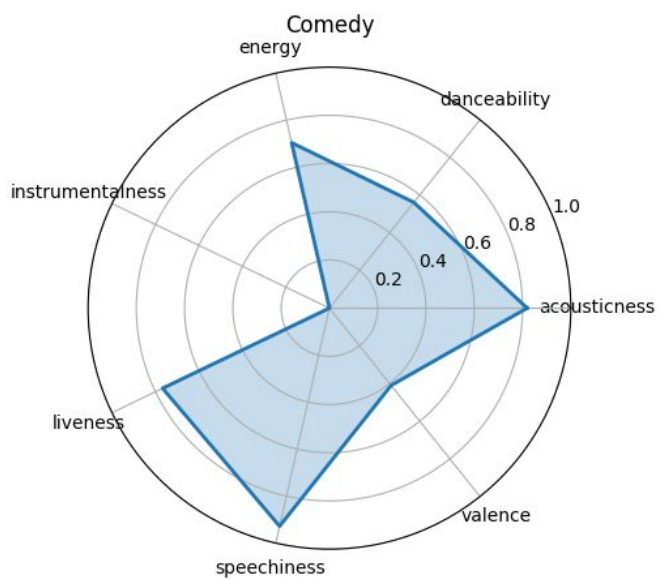
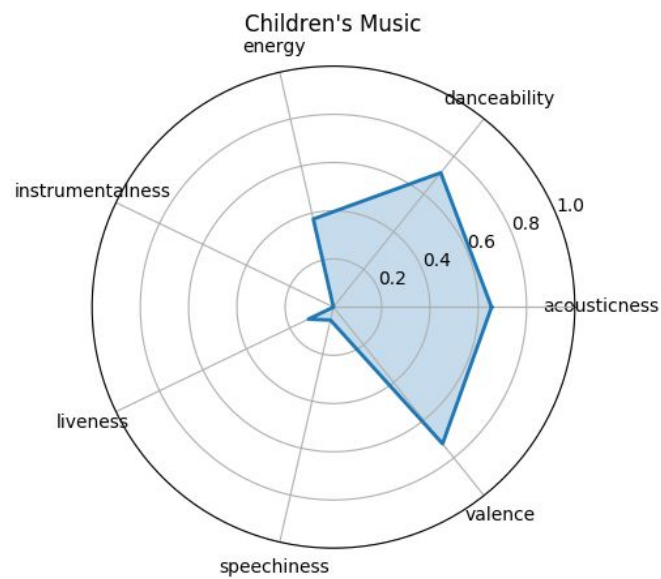
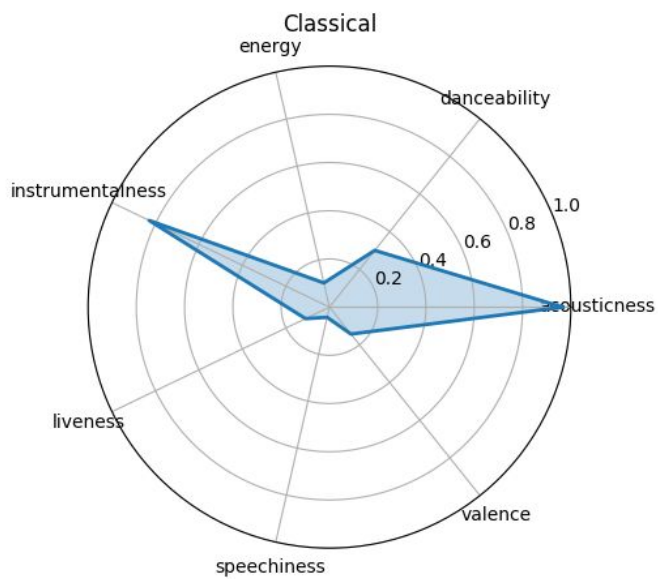
genre_names = tracks.genre.unique()
genre_meds = []


for g in genre_names:
    subtracks = tracks.loc[np.where(tracks["genre"] == g)]
    genre_meds.append([np.median(subtracks[f]) for f in features])
```

Finally, we plot each genre analysis in a separate radar plot.

```
angles = np.linspace(0, 2*np.pi, len(features),  
                    endpoint=False) # Set the radar angles  
angles = np.concatenate((angles, [angles[0]]))  
  
for i in range(len(genre_names)):  
    name = genre_names[i]  
    med = genre_meds[i]  
    med = np.concatenate((med, [med[0]])) # Closed  
    fig = plt.figure()  
    ax = fig.add_subplot(111, polar=True) # Set polar axis  
    ax.plot(angles, med, '-', linewidth=2)  
    ax.fill(angles, med, alpha=0.25) # Fulfill the area  
    # Set the label for each axis  
    ax.set_thetagrids(angles * 180/np.pi, features)  
    ax.set_title(name)  
    ax.set_rlim(0, 1)  
    ax.grid(True)  
    plt.savefig("output/radar_genres/" + name)  
    # plt.show()  
    plt.close()
```

Here are some examples of the graphs produced.





Here we can see that classical music has high acoustic and instrumentalness values whereas comedy has a speechiness predominance. Electronic music has a high level of energy and children's music feel happier.

## Popularity prediction based on musical features

In the last part, we decided to predict the popularity of a track based only on its musical features. We first created two arrays:  $x$  for the input values (features) and  $y$  for the output of the model (track's popularity). We then split the model between a training and validation subset using 20% of the entire set as the validation.

```
from sklearn.model_selection import train_test_split

x = tracks.loc[:, features].values
y = tracks.loc[:, 'popularity'].values

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.20)
```

To be able to achieve the prediction, we tried (and included) several regression-based algorithms provided in the *scikit-learn* library. Isotonic Regression and Random Forest Classifier didn't give us exploitable results, even when tweaking the hyperparameters and using some preprocessing. The two that gave us the best results are the Linear Regressor and Linear Support Vector Classification. The Linear SVC was our first choice at first due to its known efficiency with multi-label models like ours, but the processing time was orders of magnitude higher than the simple Linear Regressor for comparable results. Thus we chose to use of the Linear Regressor.

We trained the model on our training subset, then used the validation set to make predictions and analyze the results.

```

from sklearn.linear_model import LinearRegression
from sklearn import metrics

regressor = LinearRegression()
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)

df_output = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print(df_output)
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(
    metrics.mean_squared_error(y_test, y_pred)))

```


```

      Actual Predicted
0         78  42.112938
1         17  32.327481
2         55  45.476063
3         32  22.967836
4         30  40.416586
...      ...      ...
46540      50  46.173310
46541      47  47.478845
46542      53  37.633668
46543      30  30.505116
46544      27  17.766196

[46545 rows x 2 columns]
Mean Absolute Error: 12.838605510346062
Mean Squared Error: 259.35177179122496
Root Mean Squared Error: 16.104402248802188

```





With a root mean square error of 16 we cannot say that our model is highly accurate, but we believe it is still a great score considering that we are only using musical features and not properties such as genre, artist or country.