# Songs popularity prediction

By Zakariaa BOHSINE

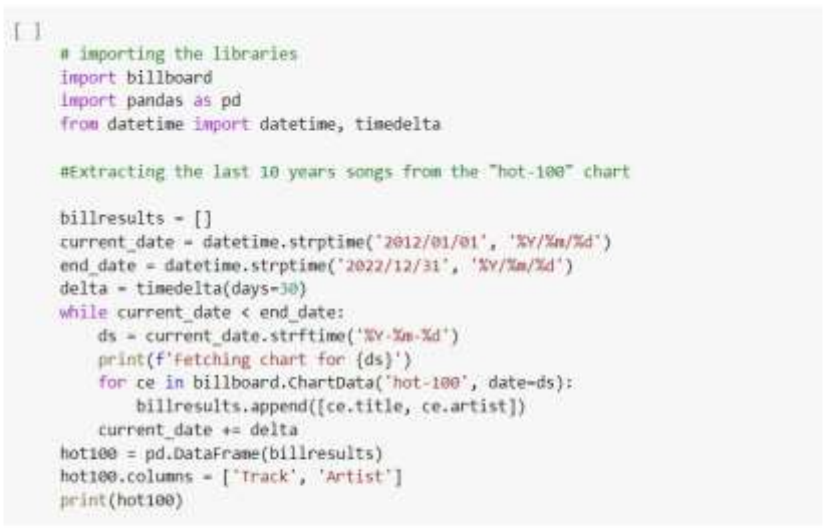
## Introduction

Songs released by popular singers tend to be chart-toppers, but what about the occasional track put out by a lesser-known artist that becomes a smash hit seemingly at random? For this we will be looking at multiple song features in order to determine the ones responsible for the song popularity, and train a machine learning model that will predict the popularity of the song.

## The dataset

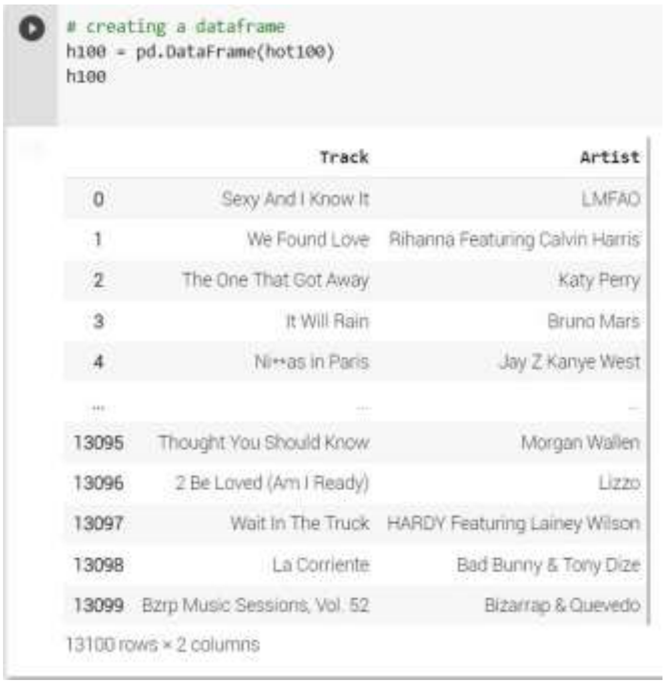
The first step was to scrape the data from the billboard platform using the billboard.py package.

The code below shows the process of scrapping the data from “hot-100” list on the billboard.



```
[ ]
    # importing the libraries
    import billboard
    import pandas as pd
    from datetime import datetime, timedelta

    #Extracting the last 10 years songs from the "hot-100" chart

    billresults = []
    current_date = datetime.strptime('2012/01/01', '%Y/%m/%d')
    end_date = datetime.strptime('2022/12/31', '%Y/%m/%d')
    delta = timedelta(days=30)
    while current_date < end_date:
        ds = current_date.strftime('%Y-%m-%d')
        print(f'fetching chart for {ds}')
        for ce in billboard.ChartData('hot-100', date=ds):
            billresults.append([ce.title, ce.artist])
        current_date += delta
    hot100 = pd.DataFrame(billresults)
    hot100.columns = ['Track', 'Artist']
    print(hot100)
```

This will give us the following result: a data frame with the songs and the artists

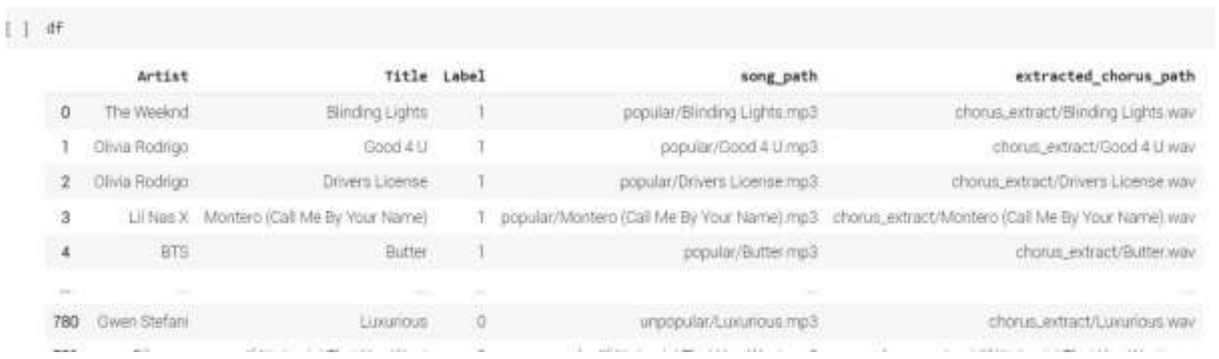


```python
# creating a dataframe
h100 = pd.DataFrame(hot100)
h100
```

```
                          Track                            Artist
0           Sexy And I Know It                              LMFAO
1               We Found Love    Rihanna Featuring Calvin Harris
2       The One That Got Away                          Katy Perry
3                 It Will Rain                         Bruno Mars
4               Ni**as in Paris                   Jay Z Kanye West
...                       ...                                ...
13095  Thought You Should Know                     Morgan Wallen
13096     2 Be Loved (Am I Ready)                             Lizzo
13097        Wait In The Truck   HARDY Featuring Lainey Wilson
13098              La Corriente             Bad Bunny & Tony Dize
13099 Bzrp Music Sessions, Vol. 52          Bizarrap & Quevedo
13100 rows × 2 columns
```

With the names of popular artists and their popular and unpopular songs, we fetch the full audio files from Youtube using youtube-dl. For songs with multiple versions, such as remakes or remixes, we choose the first published studio album version. We then use the pychorus to extract 15 seconds of repeated chorus from each song track. The basic idea of pychorus is to extract similar structures from the frequency spectrum. Essentially, it is a form of unsupervised learning and the performance of this extraction is subject to interpretation. But in this work, we assume they are the ground truth. Empirically, we inspected a few familiar songs and the chorus extracted did match our intuition.

We extracted several audio features from the chorus using librosa. We selected 11 major spectral features for the analysis. The table below shows the songs and their extracted chorus.



```
[ ] df
```

| | Artist | Title | Label | song_path | extracted_chorus_path |
|---|---|---|---|---|---|
| 0 | The Weeknd | Blinding Lights | 1 | popular/Blinding Lights.mp3 | chorus_extract/Blinding Lights.wav |
| 1 | Olivia Rodrigo | Good 4 U | 1 | popular/Good 4 U.mp3 | chorus_extract/Good 4 U.wav |
| 2 | Olivia Rodrigo | Drivers License | 1 | popular/Drivers License.mp3 | chorus_extract/Drivers License.wav |
| 3 | Lil Nas X | Montero (Call Me By Your Name) | 1 | popular/Montero (Call Me By Your Name).mp3 | chorus_extract/Montero (Call Me By Your Name).wav |
| 4 | BTS | Butter | 1 | popular/Butter.mp3 | chorus_extract/Butter.wav |
| ... | ... | ... | ... | ... | ... |
| 780 | Gwen Stefani | Luxurious | 0 | unpopular/Luxurious.mp3 | chorus_extract/Luxurious.wav |

## Data Cleaning

Data in the real world is frequently incomplete, noisy, and inconsistent. Many bits of the data may be irrelevant or missing. Data cleaning is carried out to handle this aspect. Data cleaning methods aim to fill in missing values, smooth out noise while identifying outliers, and fix data discrepancies. Unclean data can confuse data and the model. Therefore, running the data through various Data Cleaning/Cleansing methods is an important Data Preprocessing step.

### (a) Missing Data :

It's fairly common for your dataset to contain missing values. It could have happened during data collection or as a result of a data validation rule, but missing values must be considered anyway.

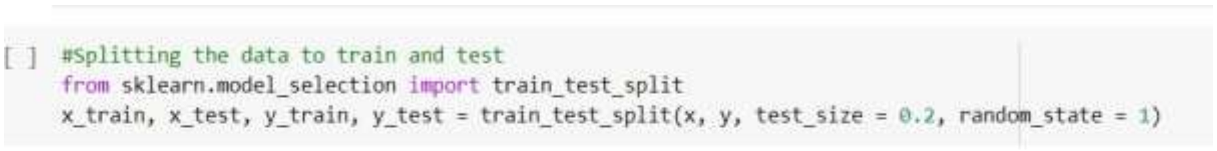
Dropping rows/columns: If the complete row is having NaN values then it doesn't make any value out of it. So such rows/columns are to be dropped immediately. Or if the % of row/column is mostly missing say about more than 65% then also one can choose to drop.

Checking for duplicates: If the same row or column is repeated then also you can drop it by keeping the first instance. So that while running machine learning algorithms, so as not to offer that particular data object an advantage or bias.

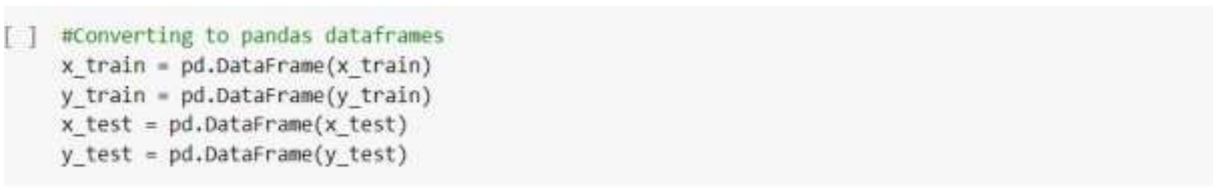
Estimate missing values: If only a small percentage of the values are missing, basic interpolation methods can be used to fill in the gaps. However, the most typical approach of dealing with missing data is to fill them in with the feature's mean, median, or mode value.

## Building the model

We are now ready to split our data into training and test set for ultimately running the Random Forest algorithm. The following lines of code will separate the y-labels from the data frame, and perform the required train-test split. We have kept 30% of the data for testing purposes.

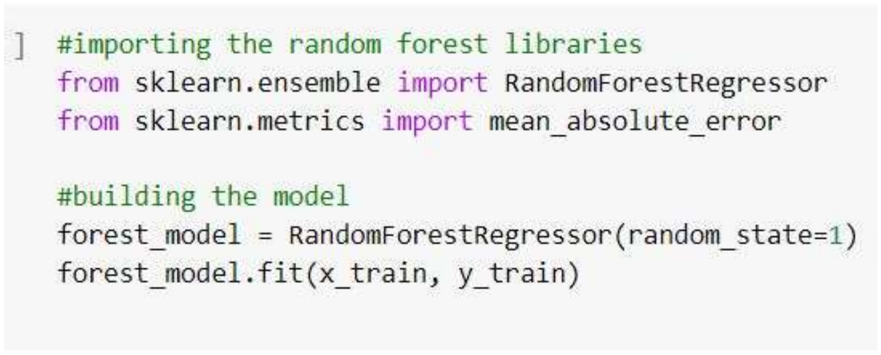


```
[ ] #Splitting the data to train and test
    from sklearn.model_selection import train_test_split
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 1)
```



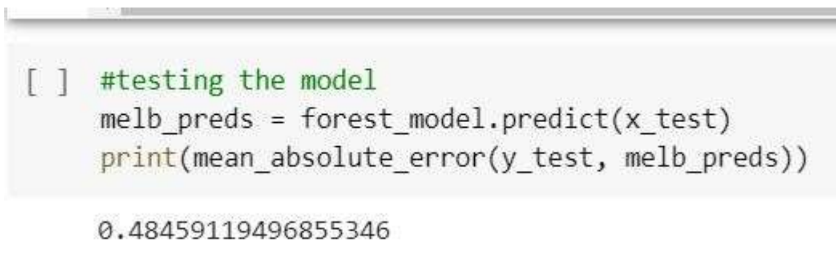
```
[ ] #Converting to pandas dataframes
    x_train = pd.DataFrame(x_train)
    y_train = pd.DataFrame(y_train)
    x_test = pd.DataFrame(x_test)
    y_test = pd.DataFrame(y_test)
```

It is now time to fit our random forest model



```
] #importing the random forest libraries
  from sklearn.ensemble import RandomForestRegressor
  from sklearn.metrics import mean_absolute_error

  #building the model
  forest_model = RandomForestRegressor(random_state=1)
  forest_model.fit(x_train, y_train)
```

We will now predict our test data



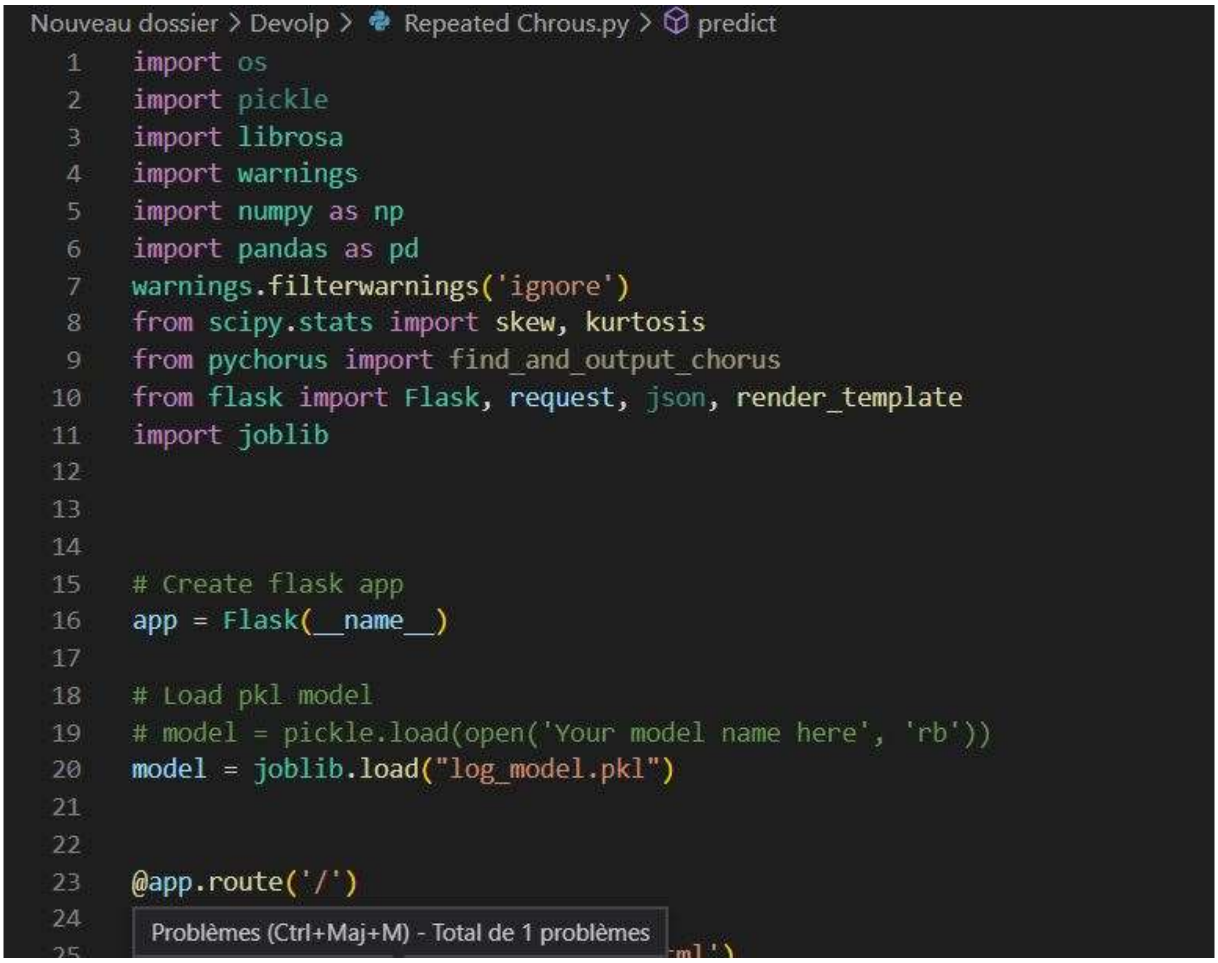
```
[ ] #testing the model
    melb_preds = forest_model.predict(x_test)
    print(mean_absolute_error(y_test, melb_preds))

    0.48459119496855346
```

## Deployment

In this part, we will see the deployment of our model by building a simple Web Application using Flask. It will run the Machine Learning model in the server as inference.

after training the model, the fastest way for the deployment is to use flask. The flow is as follows:

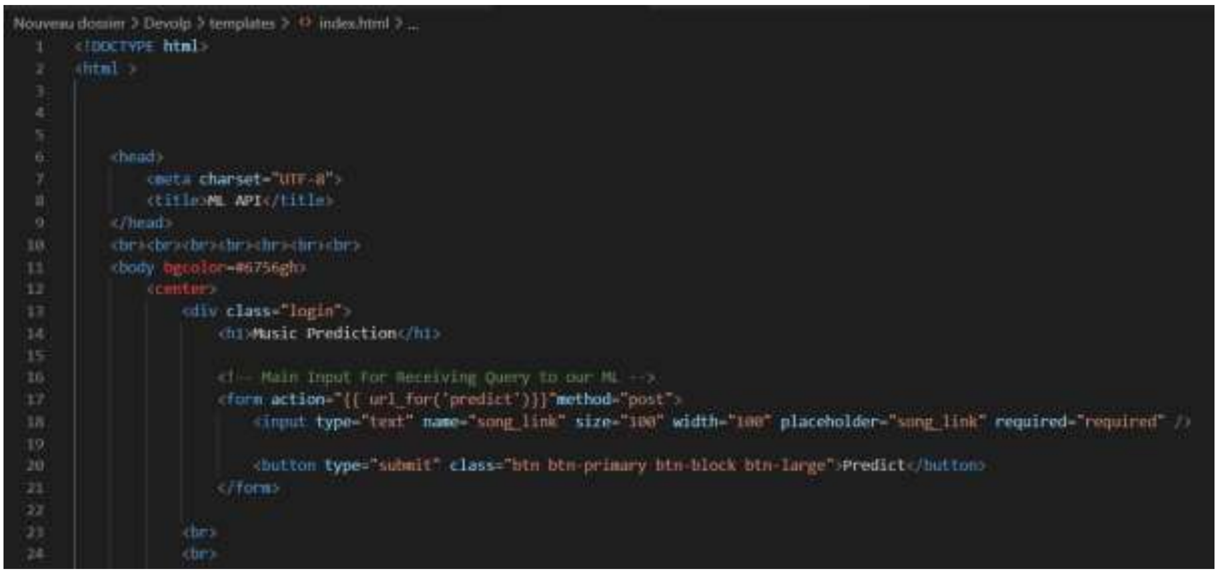
- Save the trained machine learning model
- Develop a web app using Flask
- On a user request, run inference using the saved model in the server and return the results in the UI
- Results are presented to the user on the web page



```python
import os
import pickle
import librosa
import warnings
import numpy as np
import pandas as pd
warnings.filterwarnings('ignore')
from scipy.stats import skew, kurtosis
from pychorus import find_and_output_chorus
from flask import Flask, request, json, render_template
import joblib



# Create flask app
app = Flask(__name__)

# Load pkl model
# model = pickle.load(open('Your model name here', 'rb'))
model = joblib.load("log_model.pkl")


@app.route('/')
```

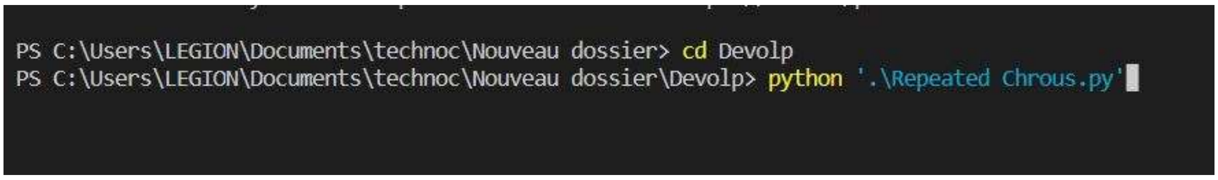
In the preceding code block, you first import the Flask object from the flask package. You then use it to create your Flask application instance with the name app. You pass the special variable __name__ that holds the name of the current Python module. It’s used to tell the instance where it’s located—you need this because Flask sets up some paths behind the scenes. Under the templates folder, there exists just one simple HTML file which expects an user to input the values.

This is the home page with the input form. It is located at: templates -> index.html.

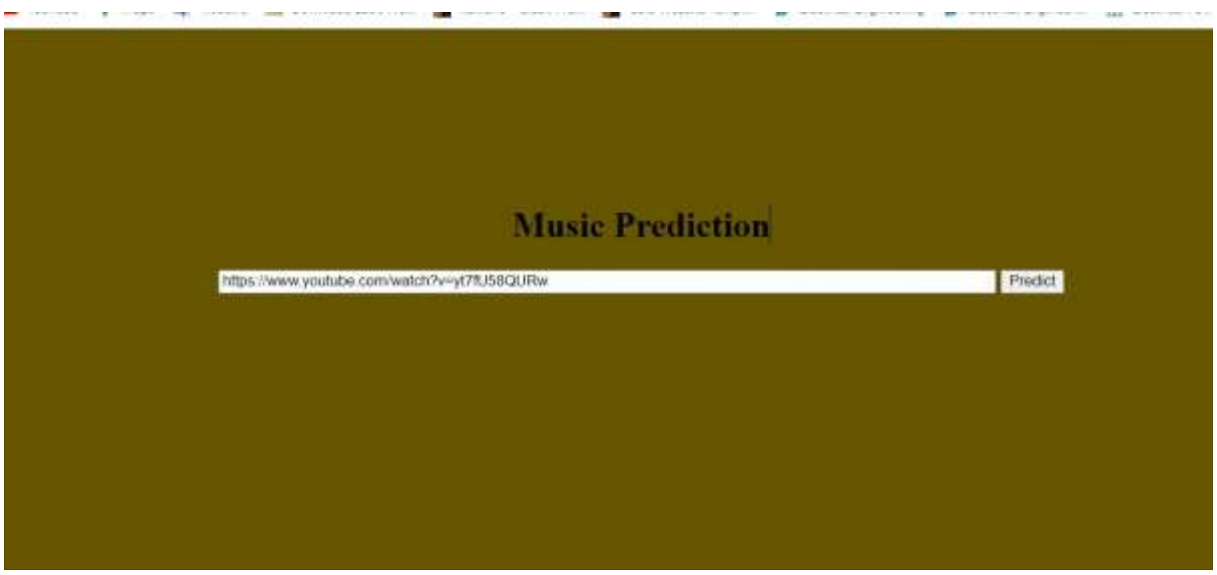


```
Nouveau dossier > Devolp > templates > index.html > ...
<!DOCTYPE html>
<html >



    <head>
        <meta charset="UTF-8">
        <title>ML API</title>
    </head>
    <br><br><br><br><br><br><br>
    <body bgcolor=#6756gh>
        <center>
            <div class="login">
                <h1>Music Prediction</h1>

                <!-- Main Input For Receiving Query to our ML -->
                <form action="{{ url_for('predict')}}"method="post">
                    <input type="text" name="song_link" size="100" width="100" placeholder="song_link" required="required" />

                    <button type="submit" class="btn btn-primary btn-block btn-large">Predict</button>
                </form>

            <br>
            <br>
```

Now to run the Flask Server, open cmd (command prompt) and run:



```
PS C:\Users\LEGION\Documents\technoc\Nouveau dossier> cd Devolp
PS C:\Users\LEGION\Documents\technoc\Nouveau dossier\Devolp> python '.\Repeated Chrous.py'
```



Music Prediction

https://www.youtube.com/watch?v=yt7fU58QURw