Recommendation Systems: AI Modelling and Data Analysis

Aditya Aeri

DESIDOC, DRDO, MOD- Metcalfe House

Dr. Mohd Yousuf Ansari

19th June 2024

INTRODUCTION

This project aims to model a recommendation system for songs, along with identifying and understanding trends followed by specified features in the chosen dataset.

Recommendation systems are tools that are used to utilise vast amounts of datasets available to an organisation to provide personalised suggestions for items of possible interest to their users. They are commonly used by e- commerce platforms and streaming services, and vary in the approach they employ, based on the requirements of the organisation and the data available, broadly being classified into collaborative filtering, content- based filtering and hybrid models. By predicting user preferences, they are able to improve satisfaction and also boost business success through increased sales and revenue.

For the purpose of the project, a content- based filtering approach has been chosen. The model analyses song metadata available to give suggestions based on individual items. It does not use any collaborative features, and hence does not necessitate a large database of user history and preferences, hence preventing it from facing a cold starting problem.

Additionally, the model is not a plot- based system, unlike that of movie or book recommendation systems, due to the nature of the items present and of the metadata available, with the former having most features with numerical entries, and latter having large amounts of text- based metadata.

As a part of the project, several procedures have been employed to attain the goals highlighted above, including data preprocessing, Exploratory Data Analysis (EDA), and model designing. The same will be discussed in detail in the upcoming sections. The programming language used is Python 3, further supplemented by libraries and toolkits to enhance data visualisation and analysis. Jupyter Notebook was used as the platform to run the code.

The code written to for the project is also given sequentially, in the upcoming sections, categorised under relevant headings, along with brief explanations regarding the decisions made that led to the development of the code in its current form. It has also been uploaded to GitHub, and a link to access the same is provided below:

https://github.com/AdityaAeri/Song-Recommendation-System-/blob/main/Song_Recc.ipynb

DATASET

The dataset used for this project was obtained from Kaggle.com. A link to the same has been provided below:

https://www.kaggle.com/datasets/joebeachcapital/30000-spotify-songs?select=spotify_songs.csv

The dataset consists of 32833 rows and 23 columns. The columns contain the following features:

track_id
track_name
track_artist
track_popularity
track_album_id
track_album_name
track_album_release_date
playlist_name
playlist_id
playlist_genre
playlist_subgenre
danceability
energy
key
loudness
mode
speechiness
acousticness
instrumentalness
liveness
valence
tempo
duration_ms

While items such as track name, artist, playlist genre etc. are textual information regarding each song, other features such as valence, instrumentalness, acousticness etc. provide an objective measurement based on a metric, generally ranging from 0 to 1, hence providing a numerical value to the featured columns.

We hence have a dataset which contains both textual and numerical information in its given columns.

DATA PREPROCESSING

Before starting to preprocess data for analysis, a number of toolkits and libraries are imported, to ensure the code runs smoothly.

```
import pandas as pd
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
import seaborn as sb
import plotly.express as px
from yellowbrick.target import FeatureCorrelation
```

We then load the dataset into Jupyter Notebook, titled as df.

```
df= pd.read_csv("spotify_songs.csv")
df
```

First, we check to see if there are any null values in the dataset

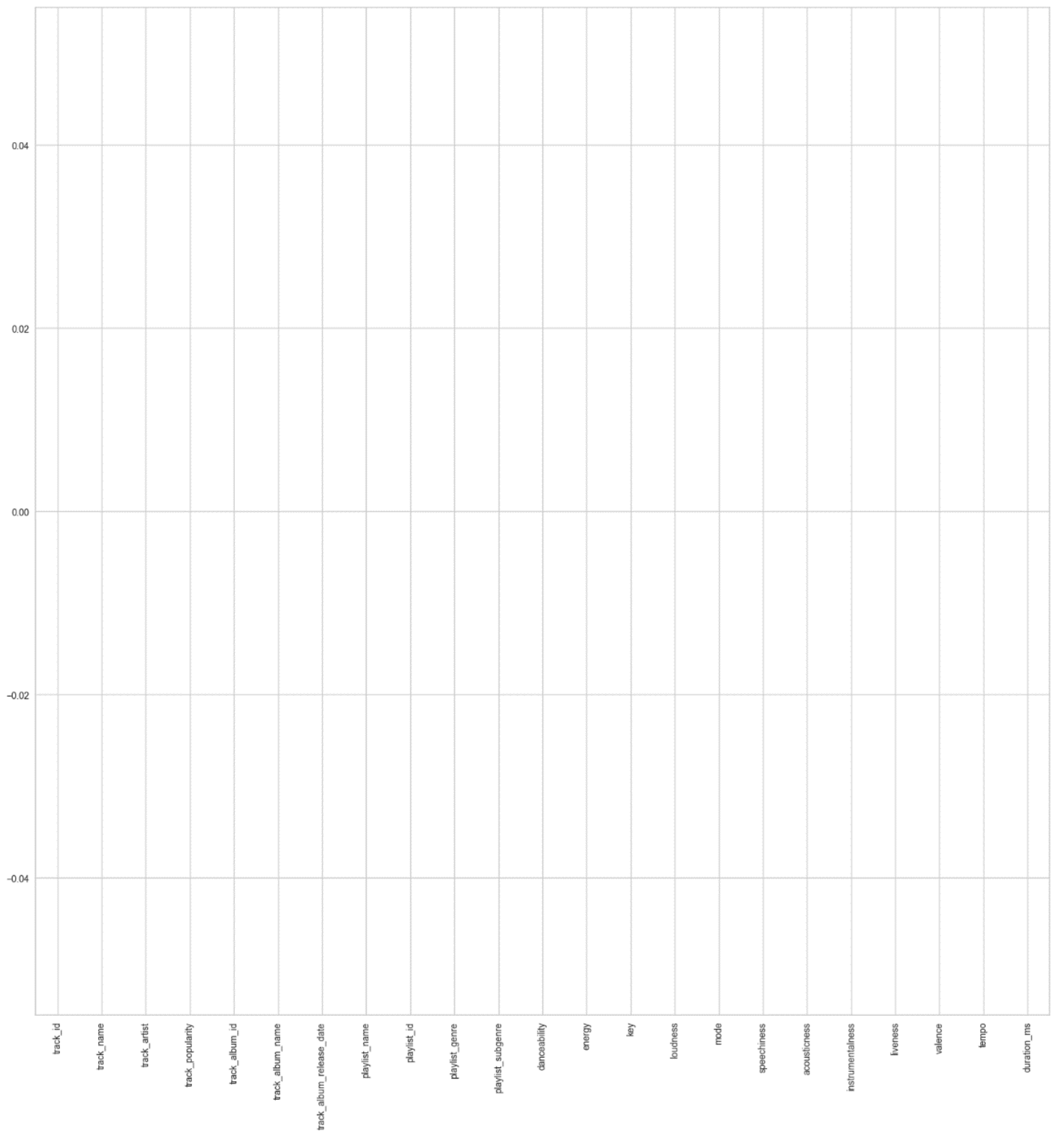
```
df.isnull().sum()
```

As the output, we get:

```
track_id                    0
track_name                  5
track_artist                5
track_popularity            0
track_album_id              0
track_album_name            5
track_album_release_date    0
playlist_name               0
playlist_id                 0
playlist_genre              0
playlist_subgenre           0
danceability                0
energy                      0
key                         0
loudness                    0
mode                        0
speechiness                 0
acousticness                0
instrumentalness            0
liveness                    0
valence                     0
tempo                       0
duration_ms                 0
dtype: int64
```

There exists 5 rows with null values. The dataset itself has over 30,000 rows, so removing 5 of them would not have a huge impact on it.

```
df.dropna(inplace = True)
df.isnull().sum().plot.bar()
plt.show()
```



We have hence removed all null values from all the columns

Next, we sort the values by track popularity, with the most popular tracks being on top. This is done so that we keep only the top 10,000 most popular songs, since the original dataset is too large. We also drop duplicate rows, based on track name.

```
df = df.sort_values(by=['track_popularity'], ascending=False)
df.drop_duplicates(subset=['track_name'], keep='first', inplace=True)
df = df.sort_values(by=['track_popularity'], ascending=False).head(1000)
df
```

We hence obtain a dataset having 10,000 rows and 23 columns.

In order to help in EDA and in designing the model, we change the feature type of the album release date to a datetime feature. We also add a new column, containing only the year of release of album, as a float datatype.

```
df['track_album_release_date']=
pd.to_datetime(df['track_album_release_date'], errors= 'coerce')
df['year'] = pd.DatetimeIndex(df['track_album_release_date']).year
df.info()
```

The output confirms that the desired result is obtained:

```
<class 'pandas.core.frame.DataFrame'>
Index: 1000 entries, 20091 to 19322
Data columns (total 24 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   track_id                  1000 non-null   object
 1   track_name                1000 non-null   object
 2   track_artist              1000 non-null   object
 3   track_popularity          1000 non-null   int64
 4   track_album_id            1000 non-null   object
 5   track_album_name          1000 non-null   object
 6   track_album_release_date  978 non-null    datetime64[ns]
 7   playlist_name             1000 non-null   object
 8   playlist_id               1000 non-null   object
 9   playlist_genre            1000 non-null   object
 10  playlist_subgenre         1000 non-null   object
 11  danceability              1000 non-null   float64
 12  energy                    1000 non-null   float64
 13  key                       1000 non-null   int64
 14  loudness                  1000 non-null   float64
 15  mode                      1000 non-null   int64
 16  speechiness               1000 non-null   float64
 17  acousticness              1000 non-null   float64
 18  instrumentalness          1000 non-null   float64
 19  liveness                  1000 non-null   float64
 20  valence                   1000 non-null   float64
 21  tempo                     1000 non-null   float64
 22  duration_ms               1000 non-null   int64
 23  year                      978 non-null    float64
dtypes: datetime64[ns](1), float64(10), int64(4), object(9)
memory usage: 195.3+ KB
```

We also drop the columns having track ID, album ID, playlist name and playlist ID, since these are not important for the purpose of our project:

```
df = df.drop(columns=['track_id', 'track_album_id', 'playlist_name', 
'playlist_id'])
df
```

The dataset is now of 10,000 rows and 20 columns.

With the desired changes being brought about in the dataset, the data is now primed for analysis.

EDA

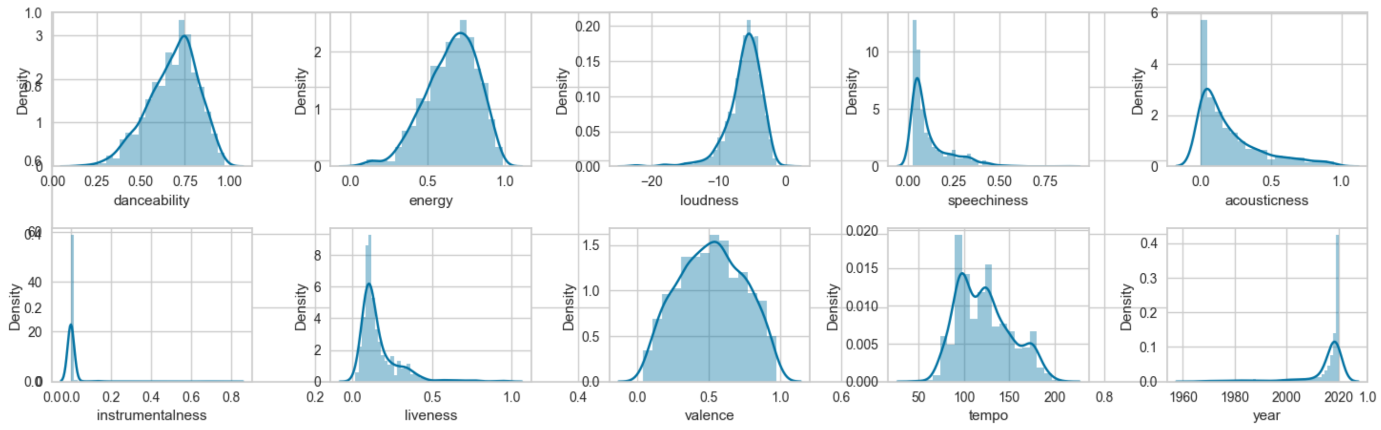
EDA is a fundamental element of any data science project. It involves examining and visualising the given dataset to understand its main characteristics and identify trends. It is one of the first things to be done, after data has been processed for modelling.

For this project, we start the EDA process by looking at all identifying all features that contain float values:

```
floats = []
for col in df.columns:
    if df[col].dtype == 'float':
        floats.append(col)

len(floats)
```

We see that there exist 10 features having float values. We plot them all as separate graphs, to understand the density of songs with respect to values of each of the features:

```
plt.subplots(figsize = (15, 5))
for i, col in enumerate(floats):
    plt.subplot(2, 5, i + 1)
    sb.distplot(df[col])
plt.tight_layout()
plt.show()
```



The graphs for most features have a normal distribution, except that of tempo. An interesting observation one can make from the graph with the feature column ‘year’ is that we can see how most of the songs in our dataset are recent ones, almost all of them belonging to the time frame around 2020. It is important to note that our dataset contains only the top 10,000 most popular songs, so it is clear that recent songs are the most popular ones.

Before continuing, we fill empty strings in the track name:

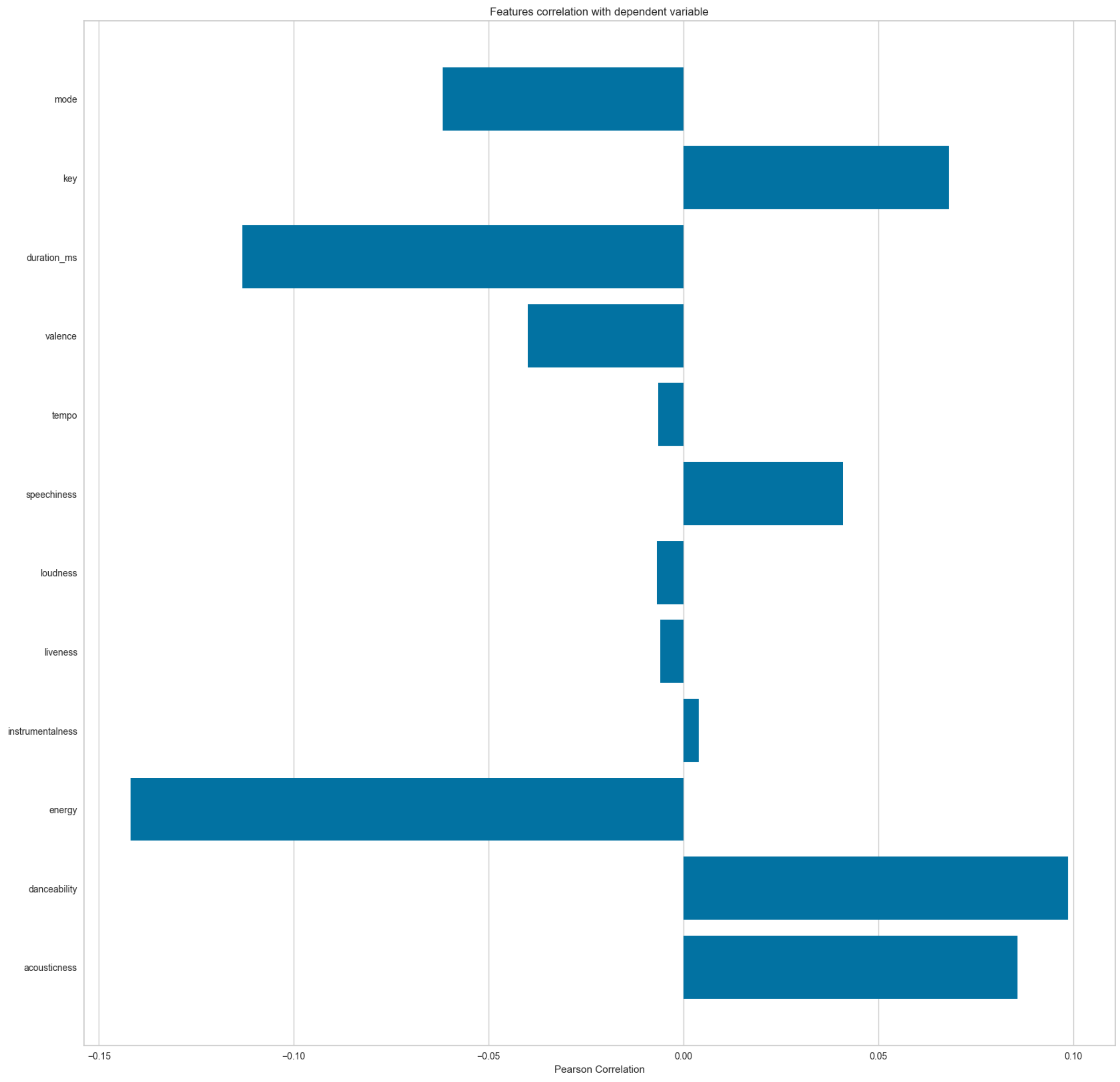
```
df['track_name'] = df['track_name'].fillna('')
df['year'] = df['year'].fillna('')
```

Next, we calculate the Pearson Correlation for the features, with respect to track popularity:

```
feature_names = ['acousticness', 'danceability', 'energy',
'instrumentalness',
       'liveness', 'loudness', 'speechiness', 'tempo',
'valence','duration_ms','key','mode']

X, y = df[feature_names], df['track_popularity']

# Create a list of the feature names
features = np.array(feature_names)

# Instantiate the visualizer
visualizer = FeatureCorrelation(labels=features)

plt.rcParams['figure.figsize']=(20,20)
visualizer.fit(X, y)     # Fit the data to the visualizer
visualizer.show()
```

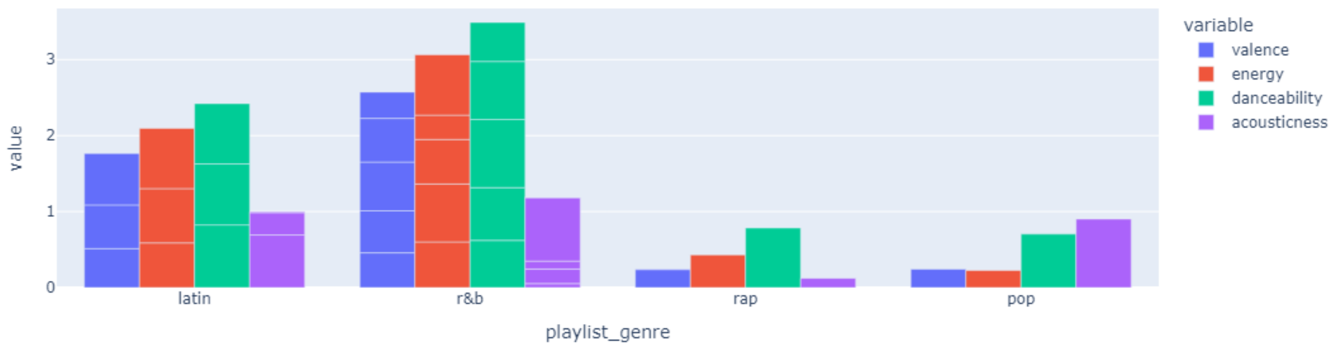
We make use of FeatureCorrelation here, as a part of YellowBrick library to visualise the Pearson Correlation. The Pearson Correlation is a score ranging from -1 to 1, where -1 indicates a total negative correlation, 1 a total positive correlation and 0 implies that the two variables are in no way correlated with each other.

From the graph (given on the next page), we can conclude that features such as energy, danceability, acousticness and duration have a very high correlation with track popularity. Other features such as liveness, intrumentalness, loudness and tempo are not that highly correlated. We can choose to ignore such features while modelling our recommender system, but in this project, they have been included as well, to provide a more accurate set of predictions.



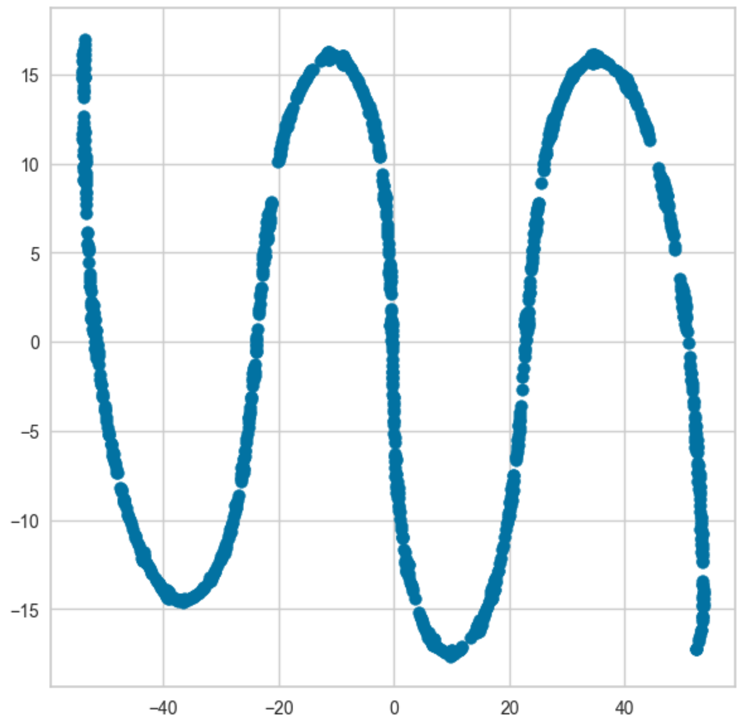
The dataset contains songs having different genre. We can study trends followed by each genre separately, with respect to a certain feature. We choose valence, energy, danceability and acousticness as the features we want to analyse:

```
top10_genres = df.nlargest(10, 'track_popularity')

fig = px.bar(top10_genres, x='playlist_genre', y=['valence', 'energy',
'danceability', 'acousticness'], barmode='group')
fig.show()
```



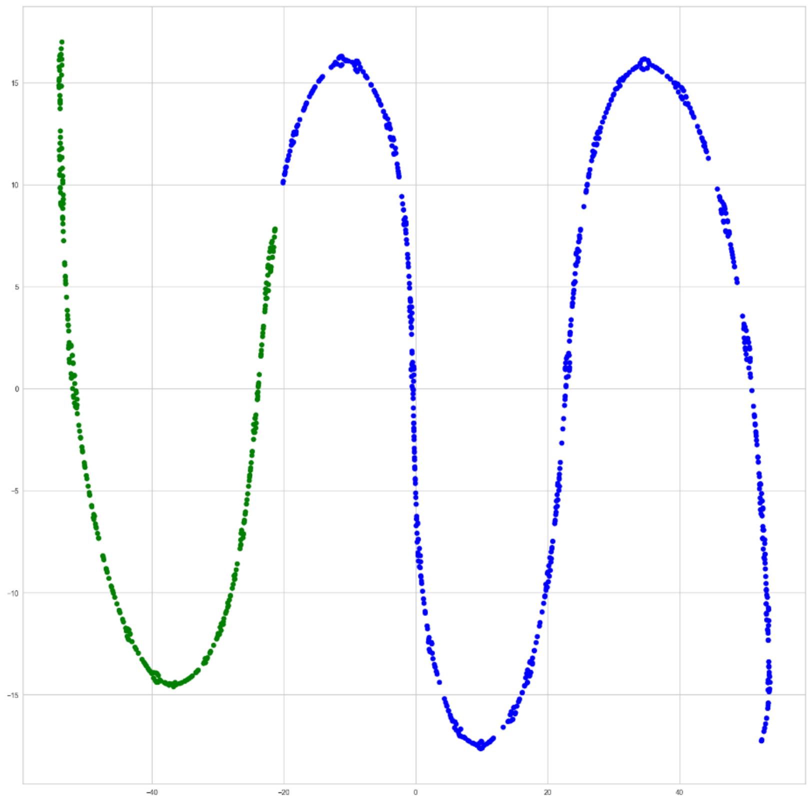
We can further visualise the available data by plotting data points on a graph. But the problem here is that there are 14 numerical columns in the dataset, hence it is of a very high dimension and cannot be plotted in its given state. We hence use the t-SNE algorithm. This algorithm converts high dimension data into lower dimension, using non- linear transformations:

```
# Select only numerical columns for t-SNE
numerical_df = df.select_dtypes(include=[np.number])

# Now apply t-SNE on the numerical data
model = TSNE(n_components=2, random_state=0)
tsne_data = model.fit_transform(numerical_df.head(1000))

# Plotting the result
plt.figure(figsize=(7, 7))
plt.scatter(tsne_data[:, 0], tsne_data[:, 1])
plt.show()
```



We hence obtain a graph representing the dataset with a reduced dimensionality value. Here, we can use spectral clustering to try to group certain data points together:

```python
# Import Spectral Clustering from scikit-learn
from sklearn.cluster import SpectralClustering

# Assuming you already have 'tsne_data' (t-SNE data) available
# Define the Spectral Clustering Model
model = SpectralClustering(n_clusters=2, affinity='nearest_neighbors')

# Fit and predict the labels
y_m_sc = model.fit_predict(numerical_df)

# Define custom colors for clusters
colors = ['blue', 'green']
cluster_colors = [colors[label] for label in y_m_sc]

# Plot the colored clusters as identified by Spectral Clustering
plt.scatter(tsne_data[:, 0], tsne_data[:, 1], c=cluster_colors, s=50)
plt.show()
```



We use spectral clustering, since given the shape of the plot, simple k- means clustering would not have been ideal to identify clusters accurately.

Note, we have specified the colour as black and green for clustering. This is because the default colours being used were black and white, rendering the white data points as being invisible.

MODEL DESIGNING

With the EDA complete, we have been able to identify certain trends and understand the nature of the dataset. We can now move to creating the recommendation system model.

Before crafting the function to recommend songs, we need to find the similarity ratings of the songs with respect to each other, based on their respective cosine similarity scores. We hence define a function to calculate these scores:

```
def get_similarities(df, song_name):
    # Instantiate CountVectorizer
    count_vectorizer = CountVectorizer()

    # Getting vector for the input song.
    text_array1 = count_vectorizer.fit_transform(df['track_name'] + " " +
df['playlist_genre']).toarray()
    num_array1 =
df[df["track_name"]==song_name].select_dtypes(include=np.number).to_numpy()

    # We will store similarity for each row of the dataset.
    sim = []
    for idx, row in df.iterrows():
        name = row["track_name"]

        # Getting vector for current song.
        text_array2 = count_vectorizer.transform([name + " " +
row['playlist_genre']]).toarray()
        num_array2 =
df[df['track_name']==name].select_dtypes(include=np.number).to_numpy()

        # Calculating similarities for text as well as numeric features
        text_sim = cosine_similarity(text_array1, text_array2)[0][0]
        num_sim = cosine_similarity(num_array1, num_array2)[0][0]
        sim.append(text_sim + num_sim)

    return sim
```

We make use of the Count Vectorizer here, since we have features containing textual information. It is one of the simplest vectorizers that exist, simply creating a vector indicating the number of times a word is used in a particular feature, for a specified row. We choose the track's name and the genre of the playlist as the core features to be considered for this.

For the numerical features, we simply convert them into an array. For calculating the overall cosine similarity value for a pair of songs, the function would add the individual cosine similarity values it has computed for the textual vector (formed using Count Vectorizer), and that of the numerical array.

Next, we write down a function to recommend songs, based on the cosine similarity index calculated by the previous function. This function would arrange most similar songs sequentially, based on their similarity scores and their popularity. In case the input song does not belong to the dataset, the function returns a statement saying 'This song is not so popular or you have entered an invalid name'

Note, we have asked the function to return a list of the highest songs, starting from 2 to 7, leaving out the first song. This is because this will be the song itself, as it will have the highest cosine similarity score with itself and will hence score very high.

```
def recommend_songs(song_name, df):
    # Base case
    if df[df['track_name'] == song_name].shape[0] == 0:
        print('This song is either not so popular or you have entered an
invalid name.\nSome songs you may like:\n')
        for song in df.sample(n=5)['track_name'].values:
            print(song)
        return

    df['similarity_factor'] = get_similarities(df, song_name)

    df.sort_values(by=['similarity_factor', 'track_popularity'],
                   ascending=[False, False],
                   inplace=True)

    # First song will be the input song itself as the similarity will be
highest.
    # Replace 'display' with 'print' if not using Jupyter notebooks.
    print(df[['track_name', 'track_artist']][2:7])
```

Finally, we test the system by asking the function to recommend songs similar to the song 'Circles':

```
recommend_songs ('Circles', df)
```

We get the following recommendations:

```
      track_name               track_artist
18888      Fresa                       TINI
17592       Tabú              Pablo Alborán
20792  Instagram  Dimitri Vegas & Like Mike
19200         HP                     Maluma
20252       Tusa                    KAROL G
```

We have hence completed designing the recommendation system for the given dataset.

CONCLUSION

Through the means of this project, we have successfully been able to understand and study various types of recommendation systems, while creating an original one. We also studied methods of EDA and how to process data correctly, to ensure smooth running of the AI model and the code in general.

The designed system was successfully able to provide recommendations for songs that are present in the chosen dataset, and is able to do so under 40 seconds, which is a decent time to process a dataset of such a size.

One of the main shortcomings of this model however is the fact that in order to make the dataset smaller and reduce the time it takes for the function to return songs that are suggested, we have had to leave out certain important criterion from being considered. This included subgenres and artist name.

The model did, though, incorporate conclusions drawn from an intense EDA process, with a clear understanding of the nature of the data.

Overall, learning outcomes include an understanding of AI modelling, tools used for EDA and data cleaning, including data visualisation techniques, how to utilise tools from several libraries in addition to Python 3.

REFERENCES

Banik, B. (2018) *Hands on Recommendation Systems with Python* (Accessed: 18th June 2024)
Hands-On Recommendation Systems with Python.pdf

Music Recommendation System Using Machine Learning (Accessed: 19th June 2024)
https://www.geeksforgeeks.org/music-recommendation-system-using-machine-learning/

What is Exploratory Data Analysis? (Accessed: 18th June 2024)
https://www.geeksforgeeks.org/what-is-exploratory-data-analysis/

Music Recommendation System using Spotify Dataset (Accessed: 19th June 2024)
https://www.kaggle.com/code/vatsalmavani/music-recommendation-system-using-spotify-dataset