Spotify Song Popularity Prediction

Presented by

Malika Patel

<u>Proposal</u>

The goal of the Spotify Prediction Project is to accurately predict the popularity of songs by using a myriad of musical features including danceability, popularity, tempo, tone, artist genres, etc. In order to accurately make these predictions the project will utilize the linear regression machine learning algorithm.

This project will leverage the "6k Spotify Playlist" Dataset from Kaggle. This set of data contains information that was generated by the company, Spotify. The information that is included in this dataset pertains to music- specifically details about artists, song tracks, release dates, and more. For this project, I will be using the audio features: energy, danceability, acousticness, instrumentalness, liveliness, speechiness, and, loudness. From artists.csv I will be using artist_popularity, artist_genres, and artist_follow. From final_playlists.csv: playlist_followers and n_tracks. From final_tracks: popularity, album_type, is_playable, release_date, artists_uris, playlist_uris. All of these attributes will be used to predict our target variable- track_popularity using Linear Regression.

<u>The data attributes of this dataset include 4 csv files and audio features:</u>

<u>artists.csv:</u>
artist_uri,,
artist_popularity,
artist_genres,
artist_follow.

<u>final_playlists.csv:</u>
uri,
name,
description,
query,
author,
playlist_followers,
n_tracks.

final_tracks.csv
name,
artists_names,
track_uri,
popularity,
album_type,
is_playable,
release_date,
artists_uris,
playlist_uris.

main_dataset.csv
track_uri,
name,
artists_names,
popularity,
album_type,
is_playable,
release_date,
artists_uris,
playlist_uris,
danceability.

The audio features:
acousticness,
analysis_url,
danceability,
duration_ms,
energy,
id,
instrumentalness,
key,
liveliness,
loudness,
speechiness,
mode,
tempo,
time_signature,
track_href,
type,
uri,
valence.

Dataset link: 6k Spotify Lists

Data Acquisition:

(Refer to appendix A for more detailed steps and code)

1. Navigated to GCP > Compute Engine > Virtual Machine Instance > Created an Instance4
   - Name: spotify-dataset-instance
   - Region: us-central1 (Iowa)
   - Machine type: e2-medium (2 vCPU, 1 core, 4 GB memory)
   - Boot disk: Changed size to 175 GB to accommodate for data to be downloaded

2. Upload the Kaggle.json file
   - Check if it was successfully uploaded (ls -l)

3. Return to Kaggle website for Spotify dataset and copy API command
   - kaggle datasets download -d viktoriiashkurenko/278k-spotify-songs

4. Check file directory again with ls -l and notice there is now a spotify zip file that needs to be unzipped.

5. To unzip:
   - Install Zip utilities: sudo apt install zip
   - Use the Unzip cmnd: unzip 278k-spotify-songs.zip

6. Rename directories with an underscore
   - mv 'Cleaned Analyses'/'Cleaned Analyses' 'Cleaned Analyses'/Cleaned_Analyses
   - mv 'Cleaned Analyses' Cleaned_Analyses

7. Create a bucket in cloud storage
   - gcloud storage buckets create gs://my-bucket-mpat --project=spotifyproject-415120 --default-storage-class=STANDARD --location=us-central1 --uniform-bucket-level-access

8. Copy local files on the virtual machine into the bucket

```
gcloud storage cp artists.csv gs://my-bucket-mpat/landing/
gcloud storage cp final_playlists.csv gs://my-bucket-mpat/landing/
gcloud storage cp im_getting_these_vibes_uknow.txt gs://my-bucket-mpat/landing/
gcloud storage cp main_dataset.csv gs://my-bucket-mpat/landing/
gcloud storage cp Cleaned_Analyses gs://my-bucket-mpat/landing/
```

9. Download pickle files from virtual machine to local computer

10. Perform Python commands to create a data frame with pickle files

Exploratory Data Analysis and Data Cleaning:

Jupyter Notebook I was able to compile all the necessary information from artists.csv, final_playlists.csv, final_tracks.csv, main_dataset.csv, and the audio features. Reading all the files in data frames and examining them. I dropped columns 2 unnamed columns and then used pandas to 'explode' the artists and playlists URI so each URI would have an individual row. I then merged the data frames on their matching respective URIs. Then I renamed certain column names to best represent the information they were showing.

Observations before combining merged_df with pickle_df:

Column names:

```
merged_df.columns.tolist()
```

```
['track_name',
 'artists_names',
 'track_uri',
 'track_popularity',
 'album_type',
 'is_playable',
 'release_date',
 'artists_uris',
 'playlist_uris',
 'artist_popularity',
 'artist_genres',
 'artist_followers',
 'playlist_name',
 'playlist_description',
 'query',
 'author',
 'n_tracks',
 'playlist_followers',
 'danceability',
 'instrumentalness',
 'liveness',
 'valence',
 'tempo',
 'duration_ms',
 'time_signature']
```

<u>Null Values:</u>

```
merged_df.isnull().sum()
```

```
track_name                18
artists_names              0
track_uri                  0
track_popularity           0
album_type                 0
is_playable                0
release_date               0
artists_uris               0
playlist_uris              0
artist_popularity          0
artist_genres              0
artist_followers           0
playlist_name             17
playlist_description    215453
query                     17
author                    17
n_tracks                  17
playlist_followers        17
danceability               0
instrumentalness           0
liveness                   0
valence                    0
tempo                      0
duration_ms                0
time_signature             0
dtype: int64
```

Observations after combining cleaned merged_df with pickle_df:

<u>Column names:</u>

```
final_df.columns.tolist()
```

```
['track_name',
 'artists_names',
 'track_uri',
 'track_popularity',
 'artists_uris',
 'playlist_uris',
 'artist_popularity',
 'artist_genres',
 'artist_followers',
 'playlist_name',
 'query',
 'author',
 'n_tracks',
 'playlist_followers',
 'danceability',
 'instrumentalness',
 'liveness',
 'valence',
 'tempo_x',
 'duration_ms',
 'time_signature_x',
 'key',
 'num_samples',
 'duration',
 'loudness',
 'tempo_y',
 'time_signature_y']
```

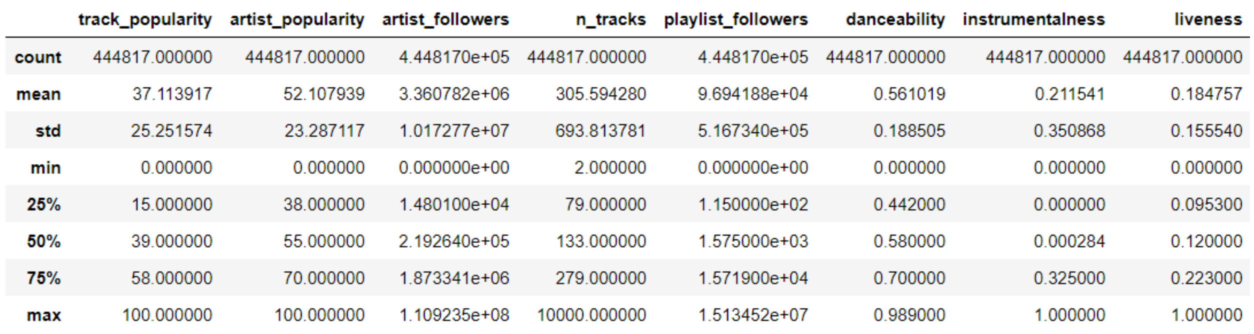
Null Values:

```
final_df.isnull().sum()
```

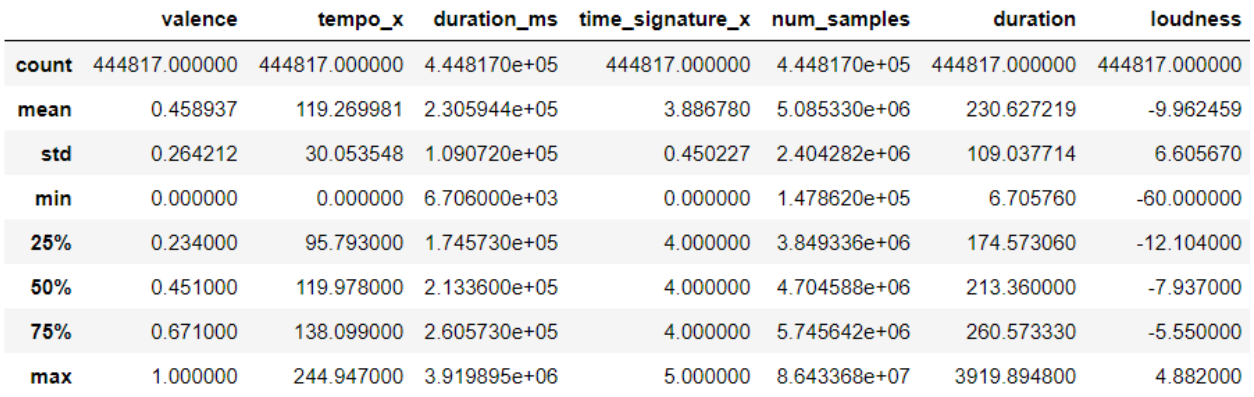
```
track_name                0
artists_names             0
track_uri                 0
track_popularity          0
artists_uris              0
playlist_uris             0
artist_popularity         0
artist_genres             0
artist_followers          0
playlist_name             0
query                     0
author                    0
n_tracks                  0
playlist_followers        0
danceability              0
instrumentalness          0
liveness                  0
valence                   0
tempo_x                   0
duration_ms               0
time_signature_x          0
num_samples           81774
duration              81774
loudness              81774
```

Descriptive Statistics:

```
final_df.describe()
```



	track_popularity	artist_popularity	artist_followers	n_tracks	playlist_followers	danceability	instrumentalness	liveness
count	444817.000000	444817.000000	4.448170e+05	444817.000000	4.448170e+05	444817.000000	444817.000000	444817.000000
mean	37.113917	52.107939	3.360782e+06	305.594280	9.694188e+04	0.561019	0.211541	0.184757
std	25.251574	23.287117	1.017277e+07	693.813781	5.167340e+05	0.188505	0.350868	0.155540
min	0.000000	0.000000	0.000000e+00	2.000000	0.000000e+00	0.000000	0.000000	0.000000
25%	15.000000	38.000000	1.480100e+04	79.000000	1.150000e+02	0.442000	0.000000	0.095300
50%	39.000000	55.000000	2.192640e+05	133.000000	1.575000e+03	0.580000	0.000284	0.120000
75%	58.000000	70.000000	1.873341e+06	279.000000	1.571900e+04	0.700000	0.325000	0.223000
max	100.000000	100.000000	1.109235e+08	10000.000000	1.513452e+07	0.989000	1.000000	1.000000

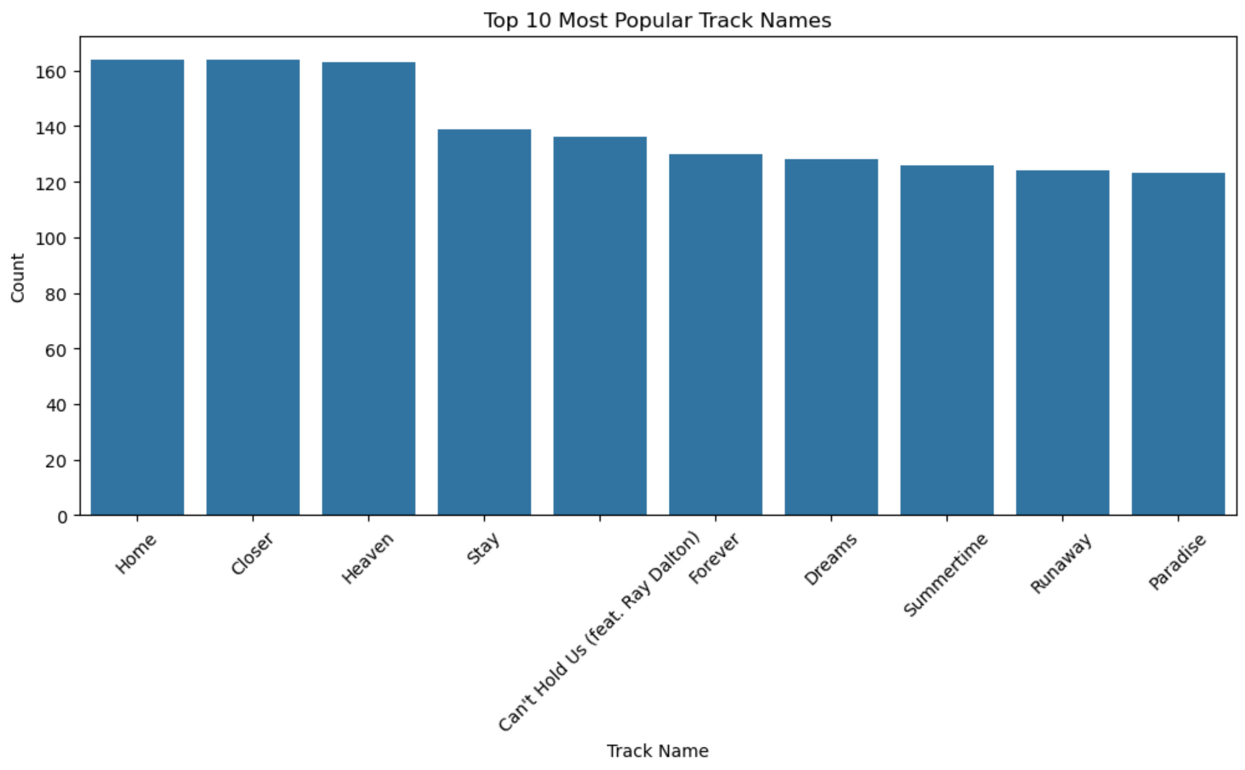


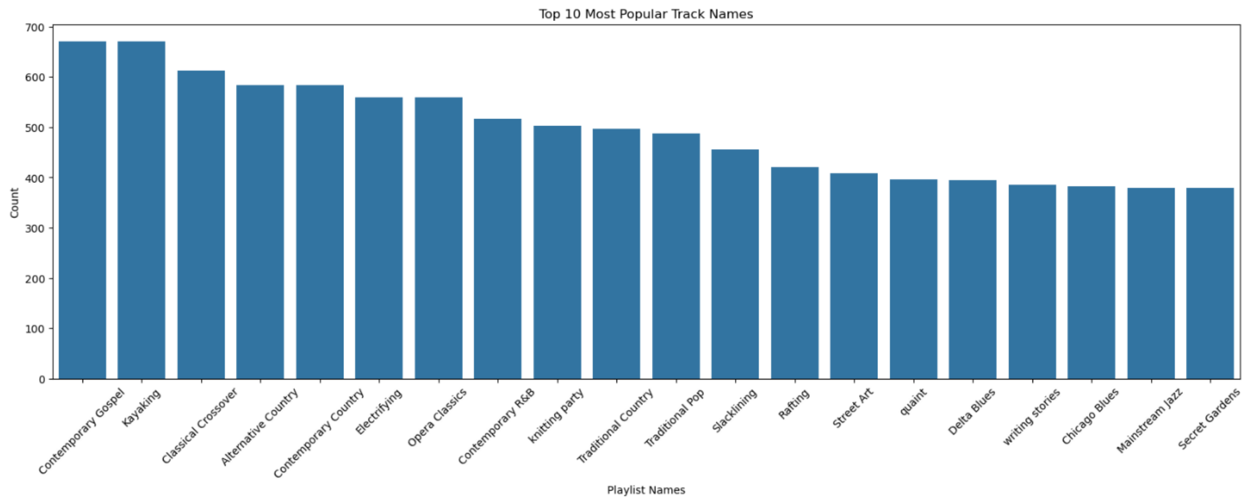
	valence	tempo_x	duration_ms	time_signature_x	num_samples	duration	loudness
count	444817.000000	444817.000000	4.448170e+05	444817.000000	4.448170e+05	444817.000000	444817.000000
mean	0.458937	119.269981	2.305944e+05	3.886780	5.085330e+06	230.627219	-9.962459
std	0.264212	30.053548	1.090720e+05	0.450227	2.404282e+06	109.037714	6.605670
min	0.000000	0.000000	6.706000e+03	0.000000	1.478620e+05	6.705760	-60.000000
25%	0.234000	95.793000	1.745730e+05	4.000000	3.849336e+06	174.573060	-12.104000
50%	0.451000	119.978000	2.133600e+05	4.000000	4.704588e+06	213.360000	-7.937000
75%	0.671000	138.099000	2.605730e+05	4.000000	5.745642e+06	260.573330	-5.550000
max	1.000000	244.947000	3.919895e+06	5.000000	8.643368e+07	3919.894800	4.882000

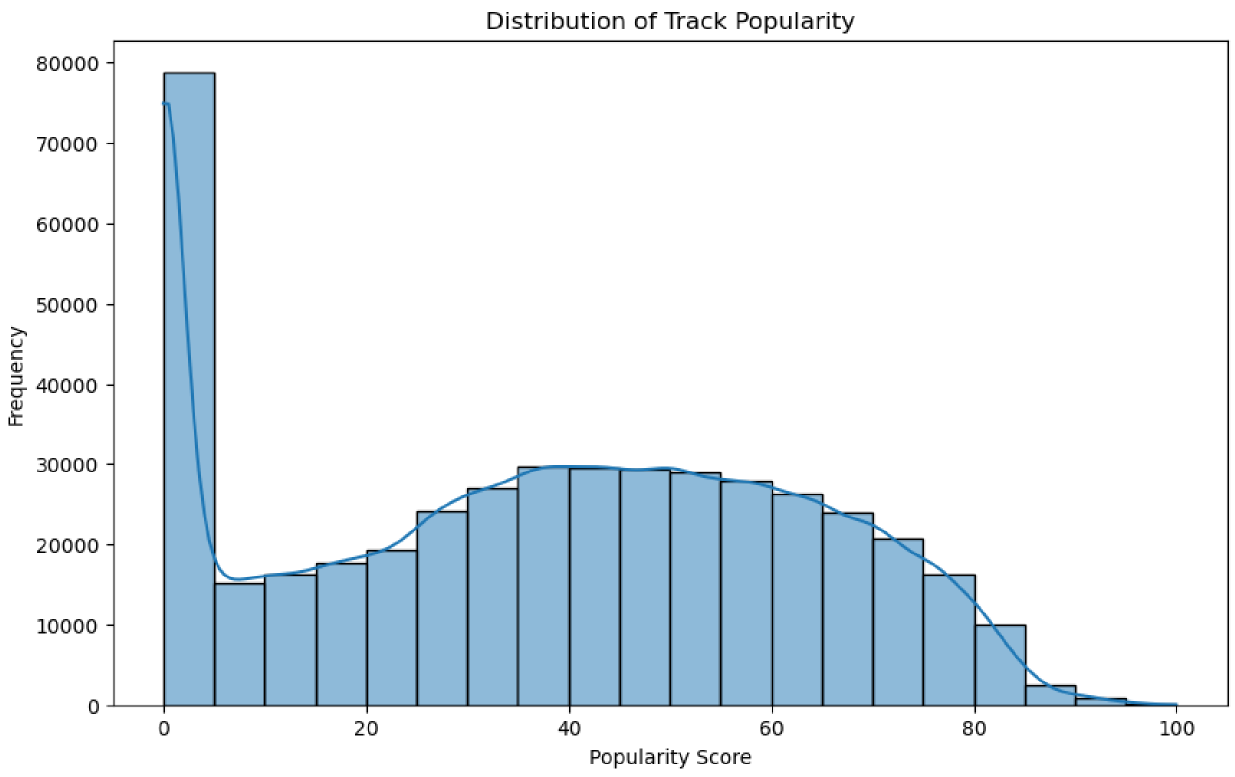
Data Cleaning

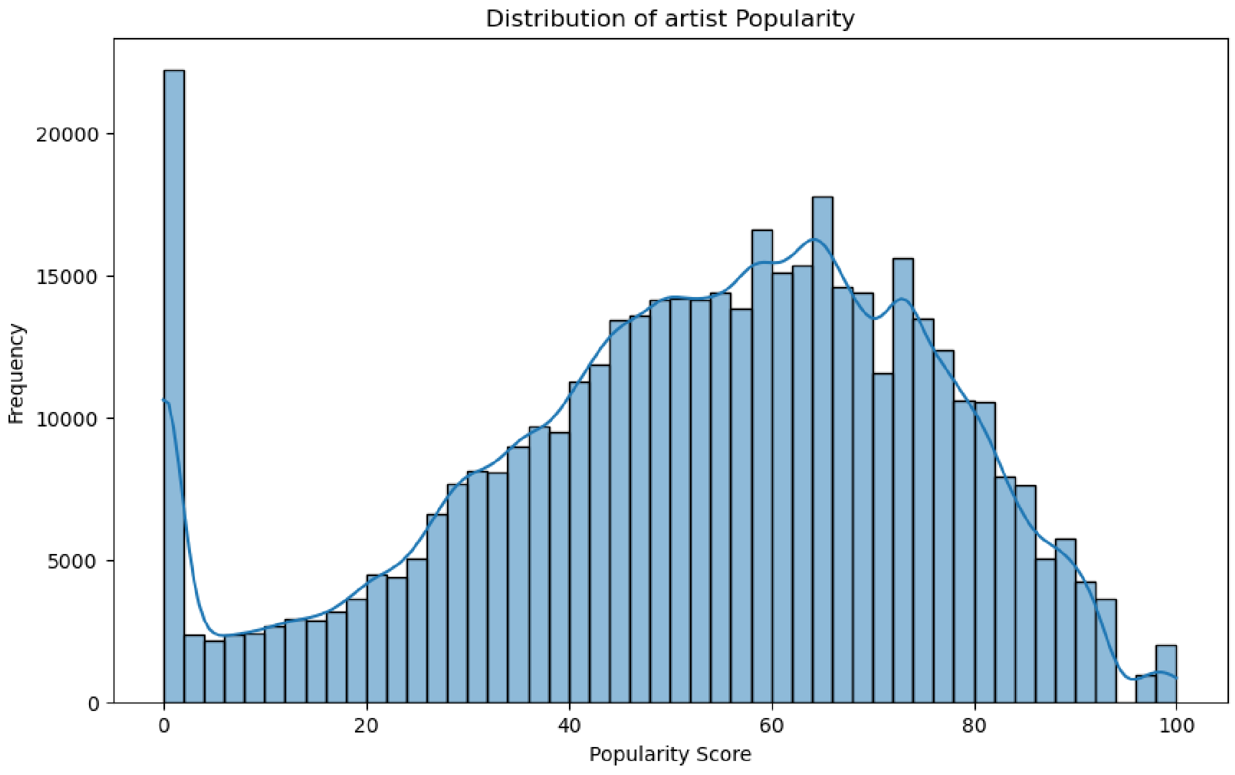
I dropped the following columns: album_type, is_playable, release_date, and playlist_description. I then proceeded to drop rows with null values

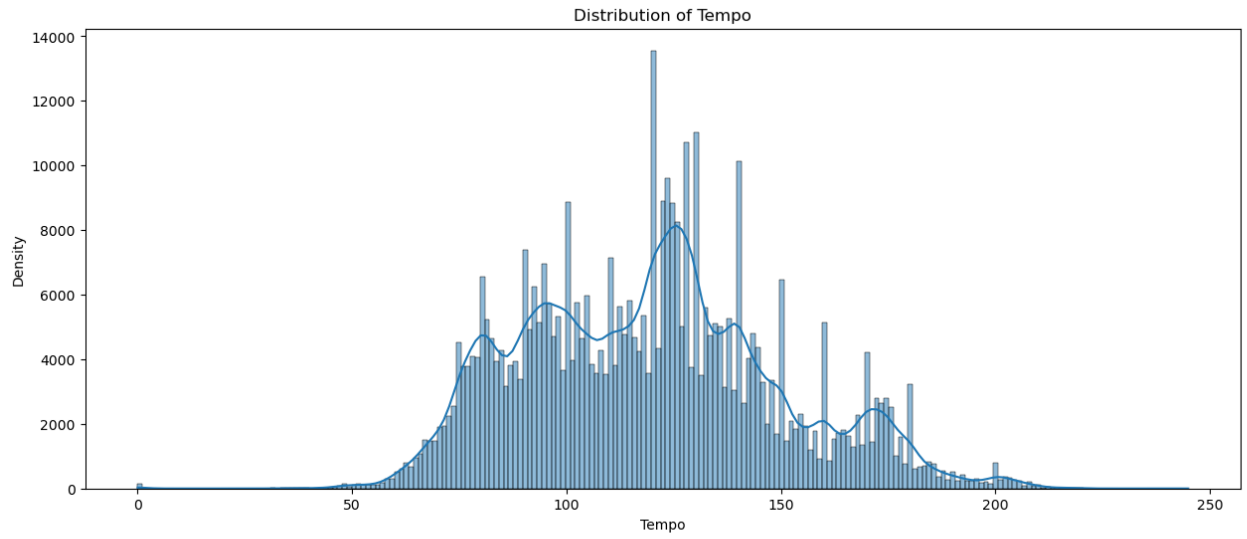
Graphs


Top 10 Most Popular Track Names
Count
Track Name
Home
Closer
Heaven
Stay
Can't Hold Us (feat. Ray Dalton)
Forever
Dreams
Summertime
Runaway
Paradise
0
20
40
60
80
100
120
140
160


Top 10 Most Popular Track Names
Count
Playlist Names
Contemporary Gospel
Kayaking
Classical Crossover
Alternative Country
Contemporary Country
Electrifying
Opera Classics
Contemporary R&B
knitting party
Traditional Country
Traditional Pop
Slacklining
Rafting
Street Art
quaint
Delta Blues
writing stories
Chicago Blues
Mainstream Jazz
Secret Gardens
0
100
200
300
400
500
600
700

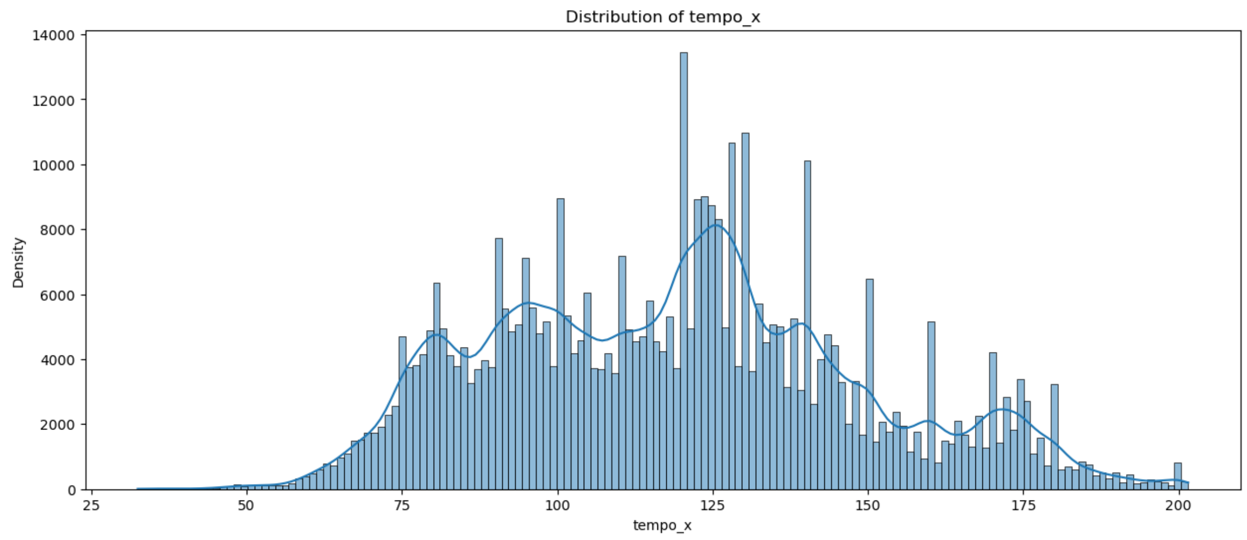

Distribution of Track Popularity
Frequency
Popularity Score
0
10000
20000
30000
40000
50000
60000
70000
80000
0
20
40
60
80
100

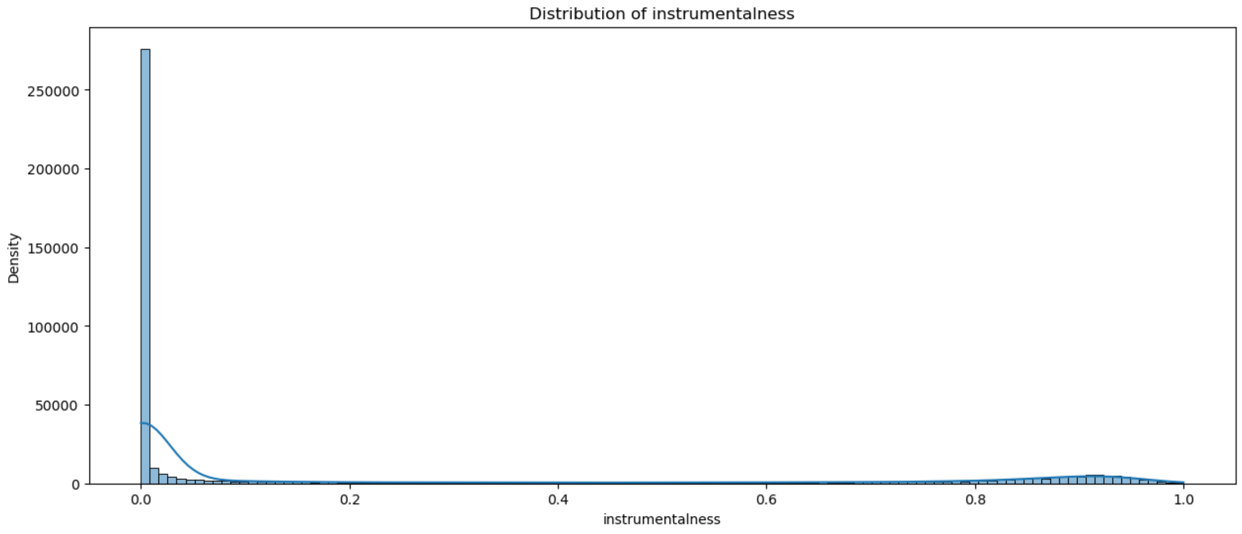

Distribution of artist Popularity
Frequency
Popularity Score
0
5000
10000
15000
20000
0
20
40
60
80
100

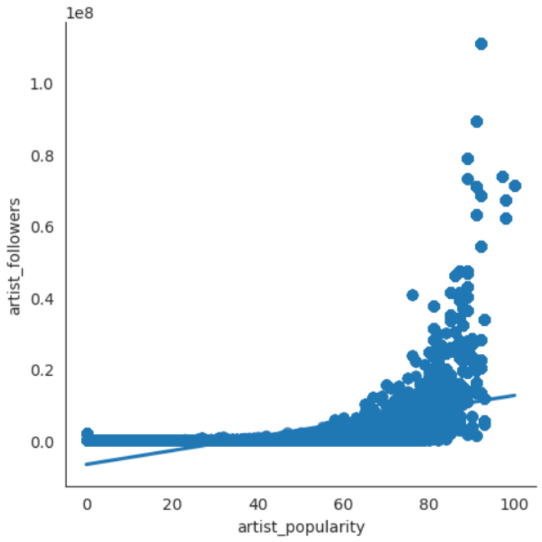


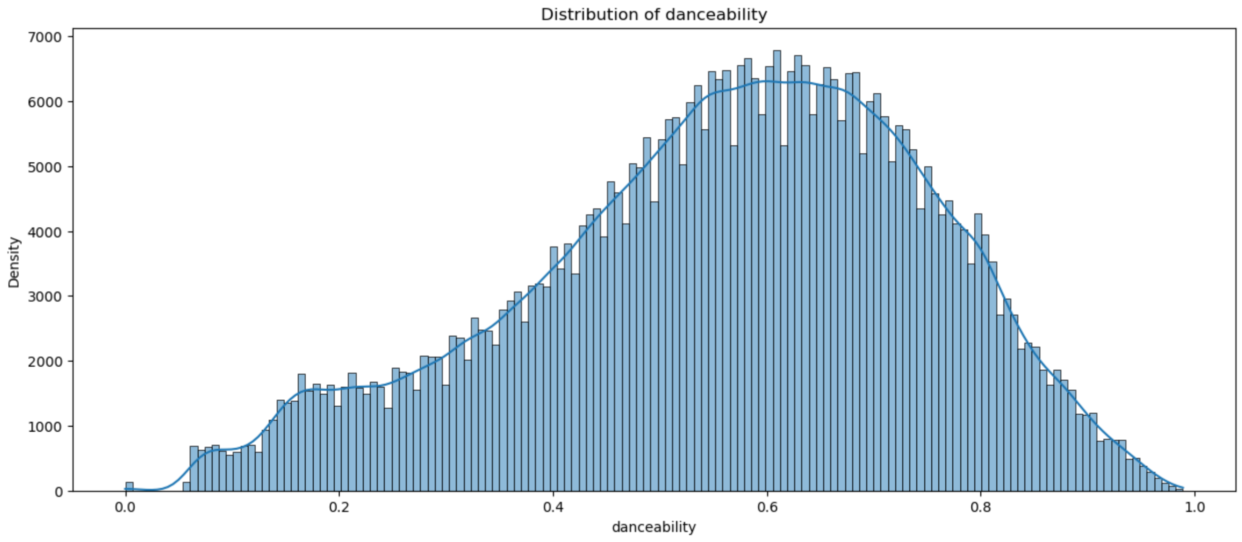
```
Outliers based on IQR:
1537      207.553
1538      207.553
1539      207.553
1540      207.553
1541      207.553
            ...
525433    209.648
525511      0.000
525847    205.805
525874    203.925
526082    206.825
Name: tempo_x, Length: 2071, dtype: float64
```

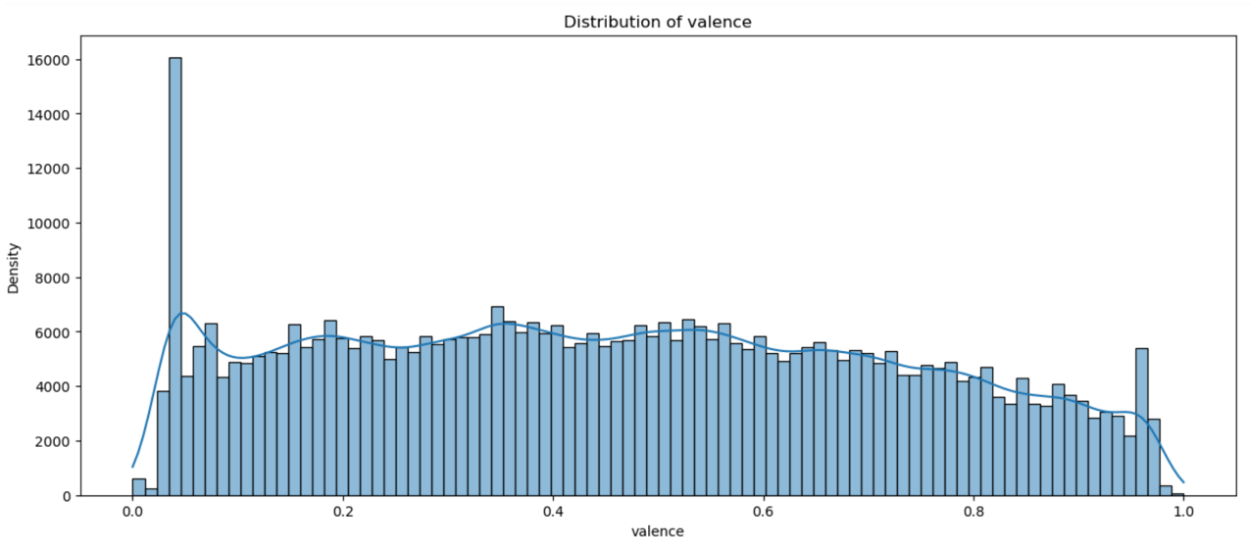
After removing outliers:

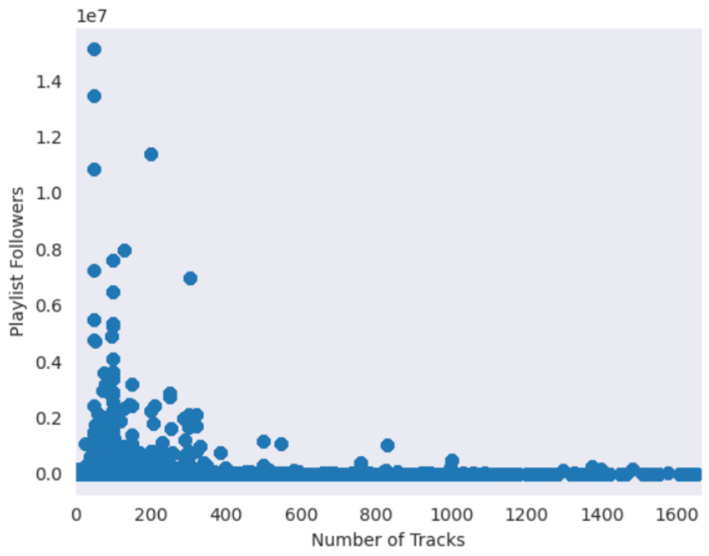


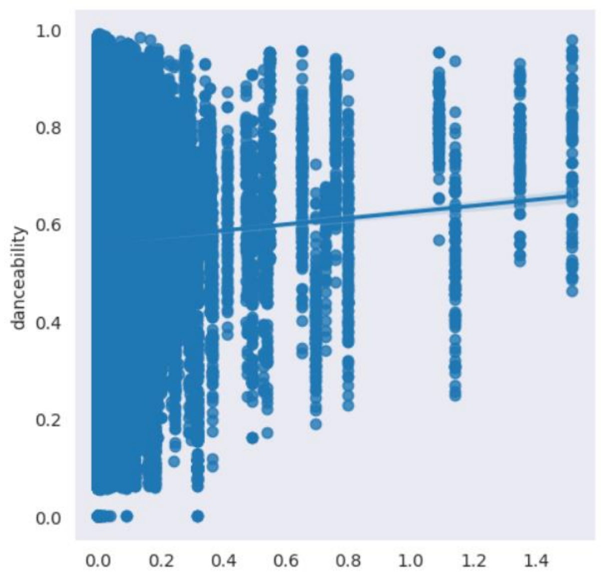

Distribution of instrumentalness
Density
250000
200000
150000
100000
50000
0
0.0
0.2
0.4
0.6
0.8
1.0
instrumentalness


1e8
1.0
0.8
0.6
0.4
0.2
0.0
artist_followers
0
20
40
60
80
100
artist_popularity


Distribution of danceability
Density
7000
6000
5000
4000
3000
2000
1000
0
0.0
0.2
0.4
0.6
0.8
1.0
danceability


Distribution of valence
Density
valence
16000
14000
12000
10000
8000
6000
4000
2000
0
0.0
0.2
0.4
0.6
0.8
1.0


1e7
Playlist Followers
Number of Tracks
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
200
400
600
800
1000
1200
1400
1600



Challenges for feature engineering:

I believe the challenges for feature engineering will be first taking aspects like genre which are nonnumeric and encoding them in order for them to be more useful. Another will be scaling the data. The rang of the data is extremely large and it will be hard to get a precise reading or prediction because of it. I think another challenge will be continuing to figure out what features will be most useful. Based on some of the graphs I have tried (not necessarily included in this document) It was hard to find any trends or patterns in the numeric data.

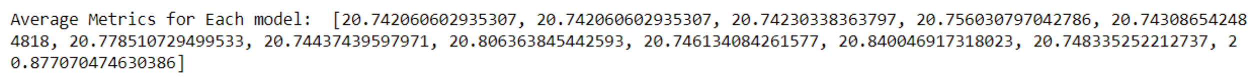
Milestone 4

track_popularity	artist_popularity	danceability	instrumentalness	valence	energy	key
long	long	double	double	double	double	double
						One Hot encoding

speechiness	acousticness	duration	loudness	tempo	time_signature
double	double	double	double	double	double
		Min Max Scaling	Min Max Scaling		

Milestone summary:

For this step in the milestone I selected certain features that I believed needed to be scaled and applied the appropriate feature engineering techniques to aid in the overall accuracy of the model. The first step I took was to apply the linear regression algorithm to the all of the features without feature engineering, This allowed me to have a baseline with how the model would perform in its current state. Then I applied Min Max scaling to the duration column. Following I retested the with the linear regression model and used the duration scaled instead of duration. Next I applied Min Max scaling to the loudness column and then retested the model with loudness scaled. Lastly, having discovered that the key column can be considered a categorical column despite being numerical, I applied the string indexer to it then used One Hot Encoding. I then applied the linear regression model one more time and was able to see the impact of the feature engineering on the mode. Some of the issues I found through applying the feature engineering and model testing was that I saw little improvement in the accuracy of the model. Many of the features using in the model were already nicely scaled, however, through each feature engineering process it made little difference to the overall outcome of the model. The resulting outcome from the model showed: RMSE: 20.78781940875206 and Rsquared: 0.3202617529768369.

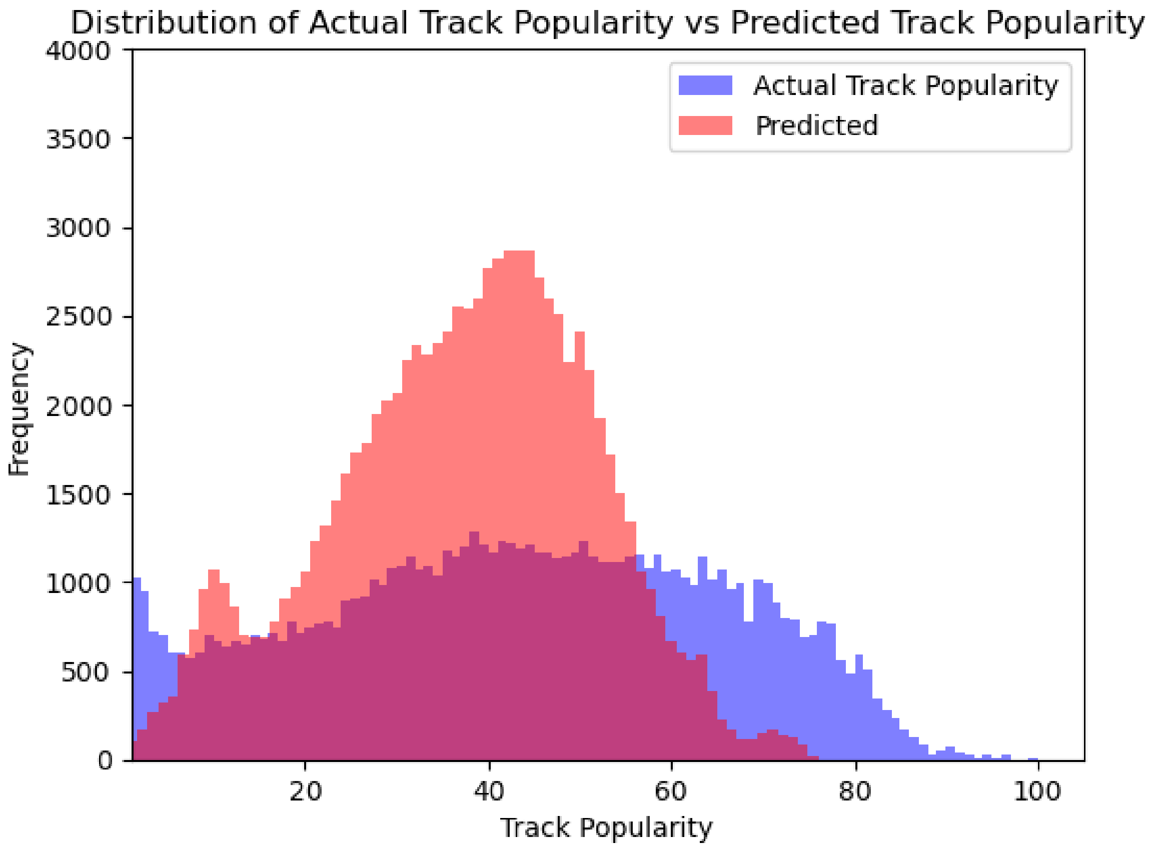


```
Average Metrics for Each model:  [20.742060602935307, 20.742060602935307, 20.74230338363797, 20.756030797042786, 20.74308654248
4818, 20.778510729499533, 20.74437439597971, 20.806363845442593, 20.746134084261577, 20.840046917318023, 20.748335252212737, 2
0.877070474630386]
```

```
Intercept: 9.426188764432252
Coefficient for artist_followers: 1.1258761566672822e-07
Coefficient for danceability: 8.001562344660753
Coefficient for instrumentalness: -4.839850073133193
Coefficient for liveness: -5.660768597265518
Coefficient for valence: -2.657148116154767
Coefficient for energy: -0.4001126738104274
Coefficient for KeyVector: -0.5743494121465644
Coefficient for speechiness: -0.8211715970350232
Coefficient for acousticness: -0.7185976535062186
Coefficient for DurScaled: 0.33634080805933814
Coefficient for loudnessScaled: 0.006746080839926616
Coefficient for tempo: -0.1616891564250245
Coefficient for time_signature: 0.708947477173173
Coefficient for artist_popularity: 0.5613293519339584
```

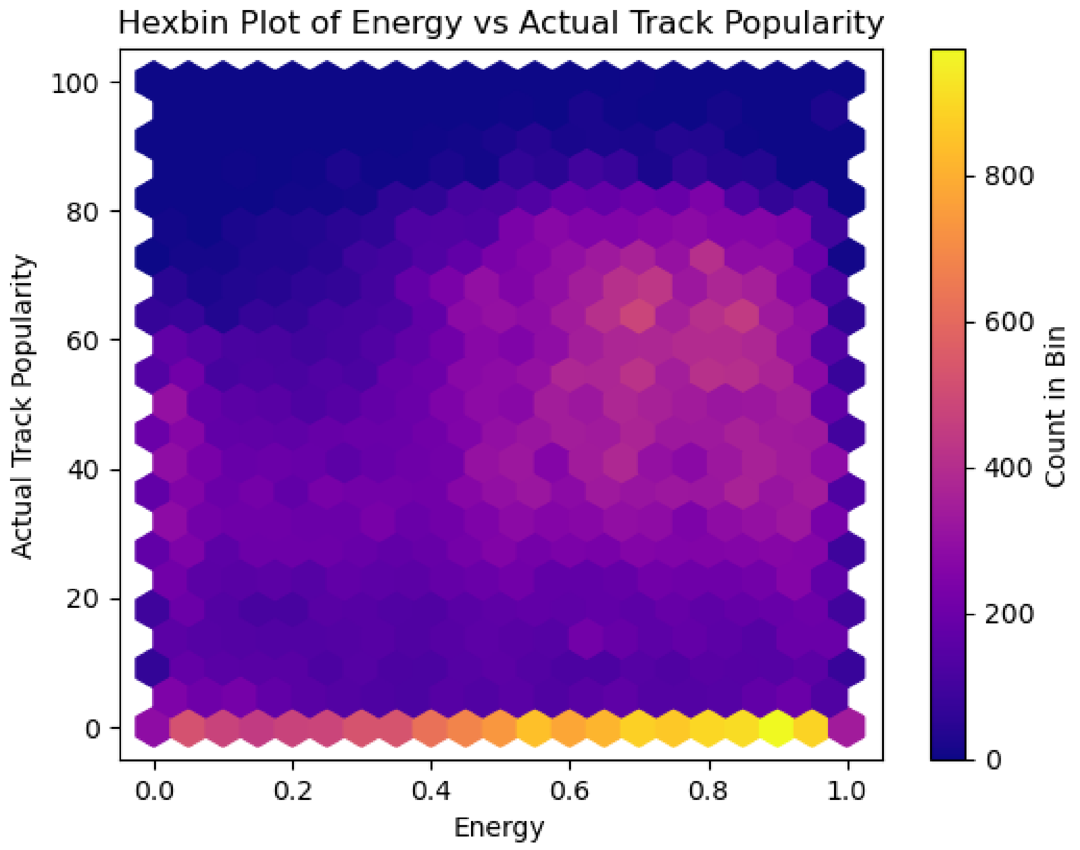
Milestone 5

Visualization 1



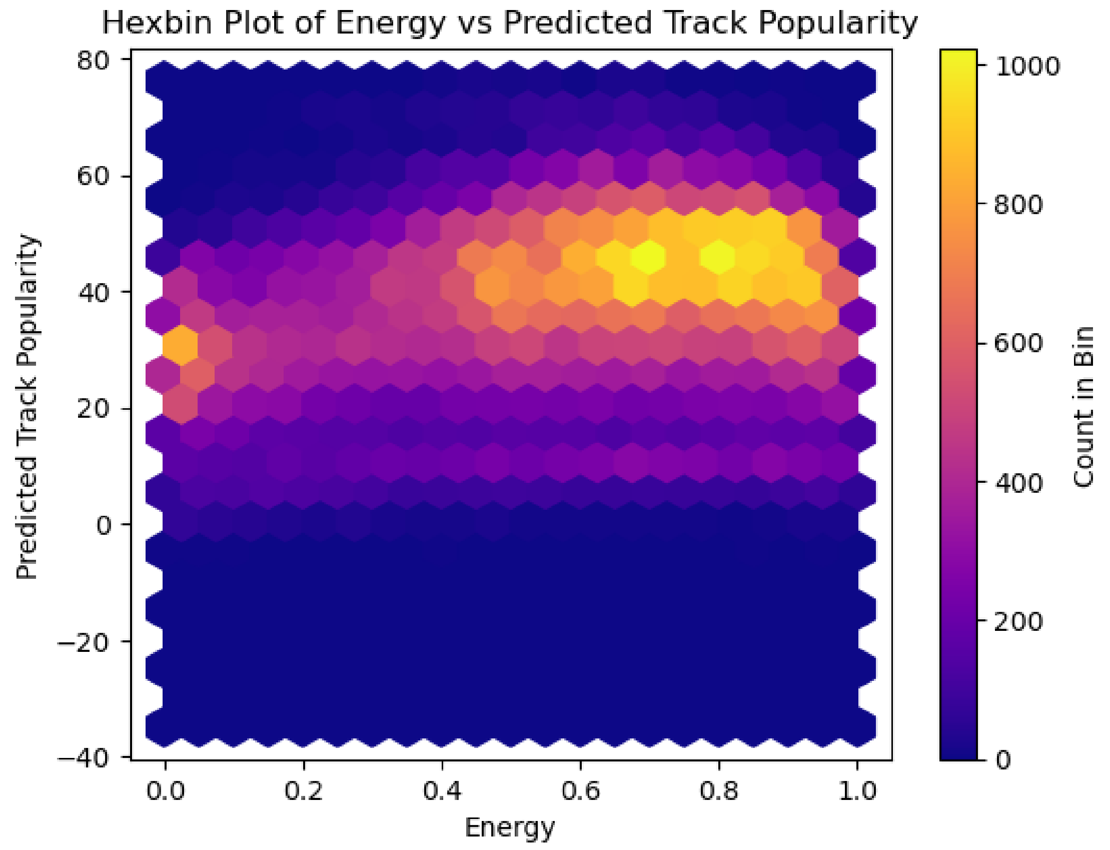
This plot shows how the prediction preformed in comparison to the actual track popularity.

Visualization 2)



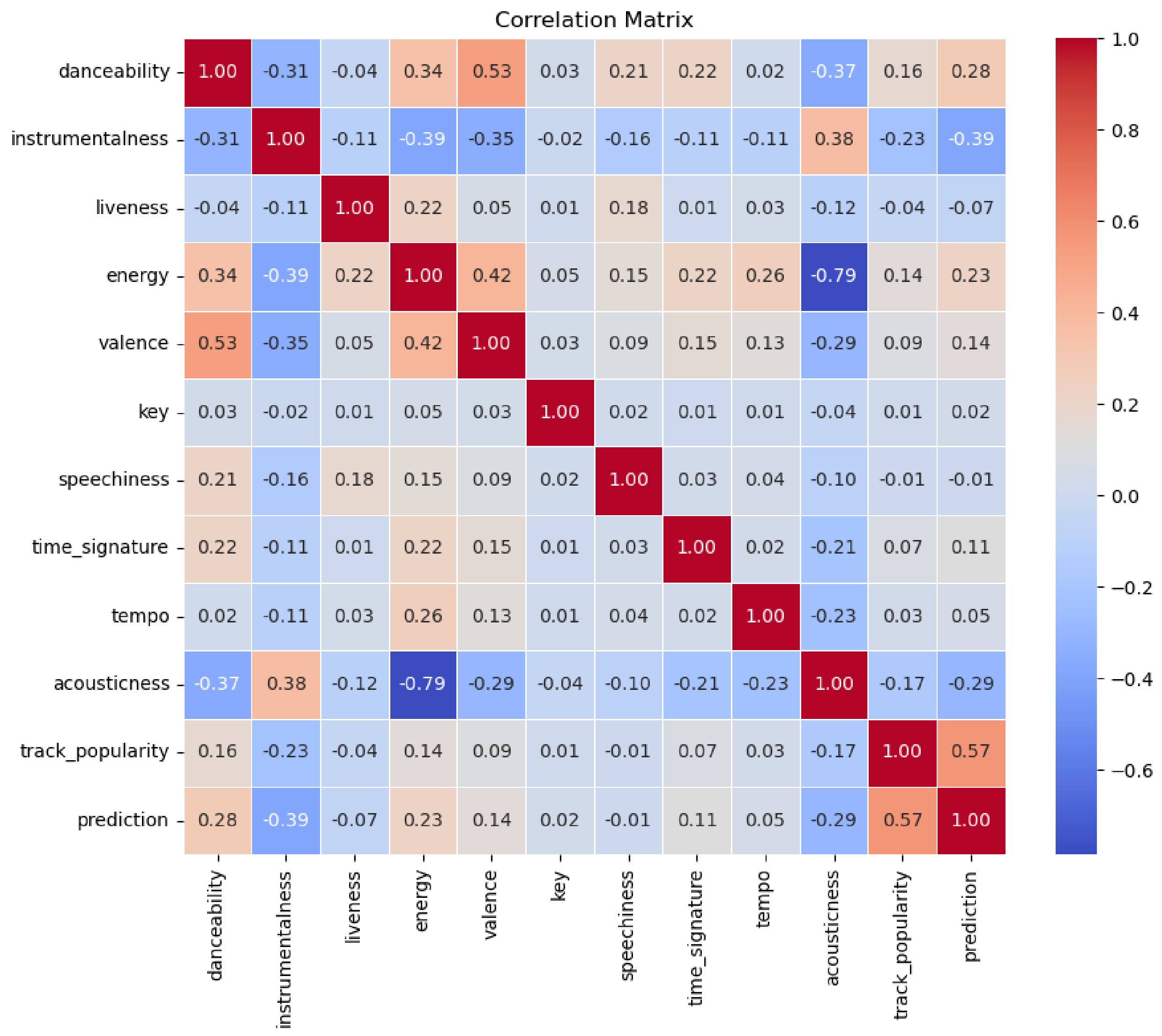
This graph shows the impact of energy on actual track popularity. And where most values fall in terms of energy and track popularity. Visually we can see that a large number of the values have a greater actual track popularity when the energy is higher.

Visualization 3)



Based on the previous plot we could see that the level of energy has an impact on the popularity of the track. Using this information the predicted track popularity from the linear regression model reflects this notion.

Visualization 4)



	danceability	instrumentalness	liveness	energy	valence	key	speechiness	time_signature	tempo	acousticness	track_popularity	prediction
danceability	1.00	-0.31	-0.04	0.34	0.53	0.03	0.21	0.22	0.02	-0.37	0.16	0.28
instrumentalness	-0.31	1.00	-0.11	-0.39	-0.35	-0.02	-0.16	-0.11	-0.11	0.38	-0.23	-0.39
liveness	-0.04	-0.11	1.00	0.22	0.05	0.01	0.18	0.01	0.03	-0.12	-0.04	-0.07
energy	0.34	-0.39	0.22	1.00	0.42	0.05	0.15	0.22	0.26	-0.79	0.14	0.23
valence	0.53	-0.35	0.05	0.42	1.00	0.03	0.09	0.15	0.13	-0.29	0.09	0.14
key	0.03	-0.02	0.01	0.05	0.03	1.00	0.02	0.01	0.01	-0.04	0.01	0.02
speechiness	0.21	-0.16	0.18	0.15	0.09	0.02	1.00	0.03	0.04	-0.10	-0.01	-0.01
time_signature	0.22	-0.11	0.01	0.22	0.15	0.01	0.03	1.00	0.02	-0.21	0.07	0.11
tempo	0.02	-0.11	0.03	0.26	0.13	0.01	0.04	0.02	1.00	-0.23	0.03	0.05
acousticness	-0.37	0.38	-0.12	-0.79	-0.29	-0.04	-0.10	-0.21	-0.23	1.00	-0.17	-0.29
track_popularity	0.16	-0.23	-0.04	0.14	0.09	0.01	-0.01	0.07	0.03	-0.17	1.00	0.57
prediction	0.28	-0.39	-0.07	0.23	0.14	0.02	-0.01	0.11	0.05	-0.29	0.57	1.00

This plot shows the level of correlation between all of the features, however, by looking the track popularity and the prediction we can say how the other features impact the correlation. These rows show that that the correlation between the features are nearly doubled for the prediction in comparison to the actual.

Milestone 6

In summary, the project to a myriad of steps to complete. We began with downloading the data via a Kaggle api and a virtual machine, after that we were able to place the data into buckets on the Google cloud platform. However, there were difficulties with certain data files specifically the Spotify pickle files because of its vast size. As a result I had to use Python to merge the large number of pickle files into a merged file and then I was able to place the files into my bucket. After I ran a cluster with data proc and I was able to run a Python code against the merged files which in essence iterated through each of the files contained within it and selected certain aspects then appended itself to a pandas data frame. After I was able to use the other data files and combine them into a singular data frame. I then performed data cleaning techniques in order to make the data more robust. After that I began to implement some visualization techniques to get a better understanding of the data that I would be working with. After I ran some linear regression algorithms against the data but realise the accuracy wasn't as promising as I had hoped it to be so I began to implement scaling and the coating to certain features that were being used in the algorithm then I reran the linear regression algorithm. Lastly I began to visualize the results and I was able to see how my predicted results did against the actual results.

In completing this project there were several conclusions that could be drawn. The main being that these instrumental aspects don’t have as much influence as we initially perceived. In general there is a large population of people with varying tastes. So its difficult to quantify whether or not something is going to be popular if everyone enjoys something different. Although we are able to visually that certain aspects such as high energy generally contain more popular tracks, the distribution of it so very wide and thus may not act as a accurate predictor. Furthermore, It demonstrates that the inclusion of other features may be beneficial in increasing the accuracy of the overall predictions made.

<u>Appendix A</u>

Data Acquisition:

1. Logged into Kaggle and Google Cloud Platform
2. Navigated to Kaggle> Settings> API> Create new token (downloaded to computer: "Kaggle.json")
3. Navigated to GCP > Compute Engine > Virtual Machine Instance > Created an Instance4
    - Name: spotify-dataset-instance
    - Region: us-central1 (Iowa)
    - Machine type: e2-medium (2 vCPU, 1 core, 4 GB memory)
    - Boot disk: Changed size to 175 GB to accommodate for data to be downloaded
4. Connect SSH > open in browser
5. Create Kaggle folder with the command line
    - Mkdir .kaggle
    - Ls -la (to check if the folder was created)
6. Upload the Kaggle.json file
    - Check if it was successfully uploaded (ls -l)
7. Move Kaggle.json into the Kaggle folder we created
    - mv kaggle.json .kaggle/
    - mv merge_pickle.pl Cleaned_Analyses/
8. Change permission to give us sole control, making it more secure
    - chmod 600 .kaggle/kaggle.json
9. Install python3-pip and virtual environment
    - sudo apt -y install python3-pip python3.11-venv

10. Give virtual environment a name

- python3 -m venv pydev

11. Change to the new directory

- cd pydev

12. Activate pydev environment

- source bin/activate

13. Install Kaggle Comand Line tools

- pip3 install Kaggle

14. Return to Kaggle website for Spotify dataset and copy API command

- kaggle datasets download -d viktoriiashkurenko/278k-spotify-songs

15. Check file directory again with ls -l and notice there is now a spotify zip file that needs to be unzipped.

16. To unzip:

- Install Zip utilities: sudo apt install zip
- Use the Unzip cmnd: unzip 278k-spotify-songs.zip

17. Rename directories with an underscore

- mv 'Cleaned Analyses'/'Cleaned Analyses' 'Cleaned Analyses'/Cleaned_Analyses
- mv 'Cleaned Analyses' Cleaned_Analyses

18. Authenticate virtual machine

- gcloud auth login

19. Create a bucket in cloud storage

- gcloud storage buckets create gs://my-bucket-mpat --project=spotifyproject-415120 --default-storage-class=STANDARD --location=us-central1 --uniform-bucket-level-access

20. Copy local files on the virtual machine into the bucket

```
gcloud storage cp artists.csv gs://my-bucket-mpat/landing/
gcloud storage cp final_playlists.csv gs://my-bucket-mpat/landing/
gcloud storage cp im_getting_these_vibes_uknow.txt gs://my-bucket-mpat/landing/
gcloud storage cp main_dataset.csv gs://my-bucket-mpat/landing/
gcloud storage cp Cleaned_Analyses gs://my-bucket-mpat/landing/
```

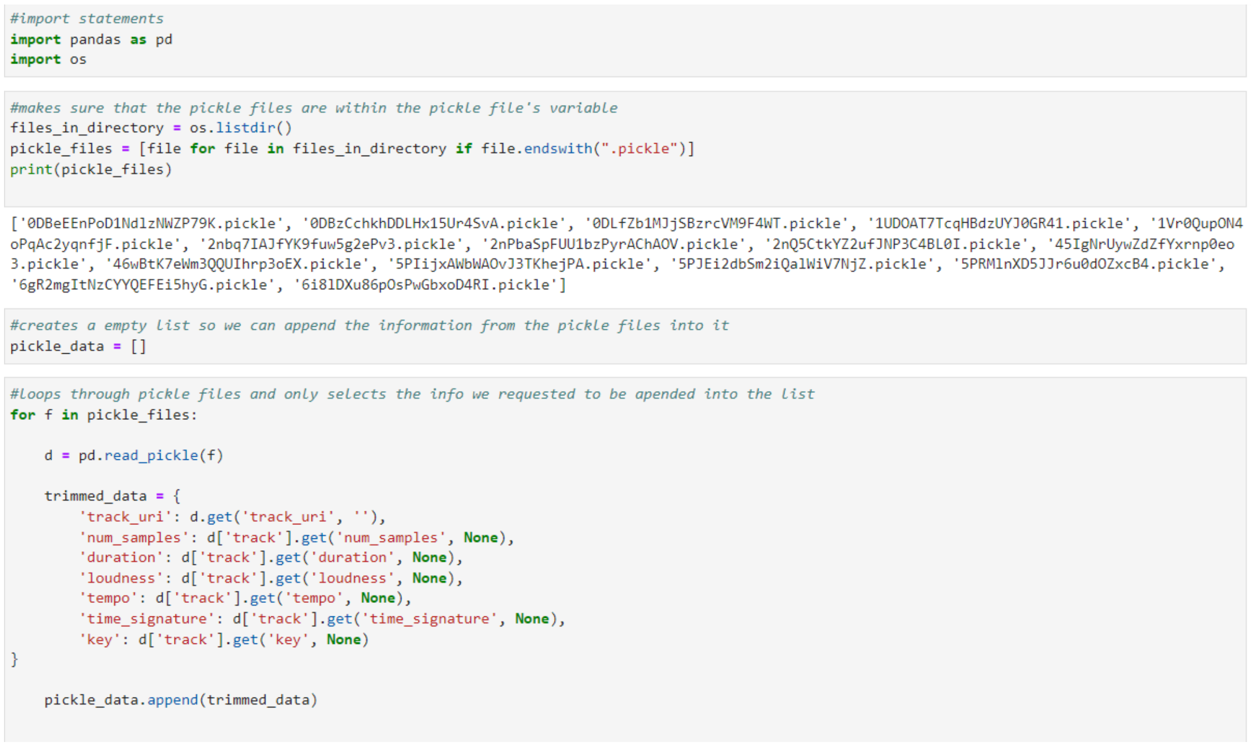
- Copy spotify merged from Google cloud bucket into mine

  ```
  gsutil -m cp gs://my-project-bucket-spotify/landing/* gs://my-bucket-mpat/landing/
  ```

21. Download pickle files from virtual machine to local computer

- /home/malika_patel1091/pydev/Cleaned_Analyses/Cleaned_Analyses/5PJEi2dbSm2iQalWiV7NjZ.pickle

10. Perform Python commands to create a data frame with pickle files



```python
#import statements
import pandas as pd
import os
```

```python
#makes sure that the pickle files are within the pickle file's variable
files_in_directory = os.listdir()
pickle_files = [file for file in files_in_directory if file.endswith(".pickle")]
print(pickle_files)
```

```
['0DBeEEnPoD1NdlzNWZP79K.pickle', '0DBzCchkhDDLHx15Ur4SvA.pickle', '0DLfZb1MJjSBzrcVM9F4WT.pickle', '1UDOAT7TcqHBdzUYJ0GR41.pickle', '1Vr0QupON4
oPqAc2yqnfjF.pickle', '2nbq7IAJfYK9fuw5g2ePv3.pickle', '2nPbaSpFUU1bzPyrAChAOV.pickle', '2nQ5CtkYZ2ufJNP3C4BL0I.pickle', '45IgNrUywZdZfYxrnp0eo
3.pickle', '46wBtK7eWm3QQUIhrp3oEX.pickle', '5PIijxAWbWAOvJ3TKhejPA.pickle', '5PJEi2dbSm2iQalWiV7NjZ.pickle', '5PRMlnXD5JJr6u0dOZxcB4.pickle',
'6gR2mgItNzCYYQEFEi5hyG.pickle', '6i8lDXu86pOsPwGbxoD4RI.pickle']
```

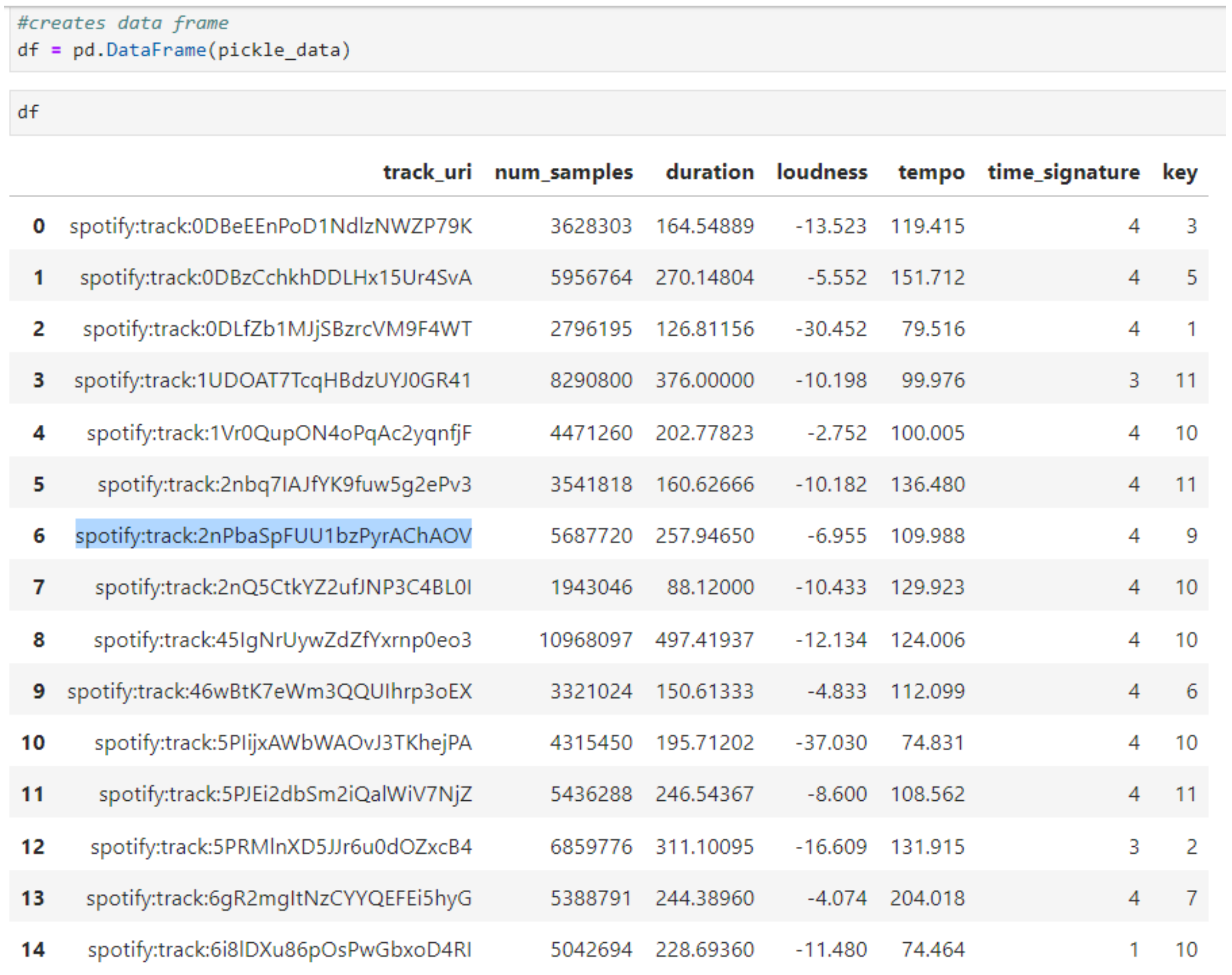
```python
#creates a empty list so we can append the information from the pickle files into it
pickle_data = []
```

```python
#loops through pickle files and only selects the info we requested to be apended into the list
for f in pickle_files:

    d = pd.read_pickle(f)

    trimmed_data = {
        'track_uri': d.get('track_uri', ''),
        'num_samples': d['track'].get('num_samples', None),
        'duration': d['track'].get('duration', None),
        'loudness': d['track'].get('loudness', None),
        'tempo': d['track'].get('tempo', None),
        'time_signature': d['track'].get('time_signature', None),
        'key': d['track'].get('key', None)
}

    pickle_data.append(trimmed_data)
```



```python
#creates data frame
df = pd.DataFrame(pickle_data)
```

```python
df
```

	track_uri	num_samples	duration	loudness	tempo	time_signature	key
0	spotify:track:0DBeEEnPoD1NdlzNWZP79K	3628303	164.54889	-13.523	119.415	4	3
1	spotify:track:0DBzCchkhDDLHx15Ur4SvA	5956764	270.14804	-5.552	151.712	4	5
2	spotify:track:0DLfZb1MJjSBzrcVM9F4WT	2796195	126.81156	-30.452	79.516	4	1
3	spotify:track:1UDOAT7TcqHBdzUYJ0GR41	8290800	376.00000	-10.198	99.976	3	11
4	spotify:track:1Vr0QupON4oPqAc2yqnfjF	4471260	202.77823	-2.752	100.005	4	10
5	spotify:track:2nbq7IAJfYK9fuw5g2ePv3	3541818	160.62666	-10.182	136.480	4	11
6	spotify:track:2nPbaSpFUU1bzPyrAChAOV	5687720	257.94650	-6.955	109.988	4	9
7	spotify:track:2nQ5CtkYZ2ufJNP3C4BL0I	1943046	88.12000	-10.433	129.923	4	10
8	spotify:track:45IgNrUywZdZfYxrnp0eo3	10968097	497.41937	-12.134	124.006	4	10
9	spotify:track:46wBtK7eWm3QQUIhrp3oEX	3321024	150.61333	-4.833	112.099	4	6
10	spotify:track:5PIijxAWbWAOvJ3TKhejPA	4315450	195.71202	-37.030	74.831	4	10
11	spotify:track:5PJEi2dbSm2iQalWiV7NjZ	5436288	246.54367	-8.600	108.562	4	11
12	spotify:track:5PRMlnXD5JJr6u0dOZxcB4	6859776	311.10095	-16.609	131.915	3	2
13	spotify:track:6gR2mgItNzCYYQEFEi5hyG	5388791	244.38960	-4.074	204.018	4	7
14	spotify:track:6i8lDXu86pOsPwGbxoD4RI	5042694	228.69360	-11.480	74.464	1	10

Appendix B

Source code for EAD

```
#!/usr/bin/env python

# coding: utf-8


# In[1]:

import pandas as pd



# In[2]:

filepath= "gs://my-bucket-mpat"

filename= "artists.csv"



# In[3]:

def perform_EDA(df : pd.DataFrame, filename : str):

    """

    perform_EDA(df : pd.DataFrame, filename : str)

    Accepts a dataframe and a text filename as inputs.

    Runs some basic statistics on the data and outputs to console.


    :param df: The Pandas dataframe to explore

    :param filename: The name of the data file

    :return:

    """

    print(f"{filename} Number of records:")

    print(df.count())

    print(f"{filename} Number of duplicate records: { len(df)-
len(df.drop_duplicates())}" )

    print(f"{filename} Info")

    print(df.info())
    print(f"{filename} Describe")

    print(df.describe())

    print(f"{filename} Columns with null values")

    print(df.columns[df.isnull().any()].tolist())

    rows_with_null_values = df.isnull().any(axis=1).sum()

    print(f"{filename} Number of Rows with null values:
{rows_with_null_values}" )

    integer_column_list = df.select_dtypes(include='int64').columns

    print(f"{filename} Integer data type columns: {integer_column_list}")

    float_column_list = df.select_dtypes(include='float64').columns

    print(f"{filename} Float data type columns: {float_column_list}")

    # Add other codes here to explore and visualize specific columns




# In[4]:

filepath = "gs://my-bucket-mpat/landing"

filename_list = ['artists.csv', 'final_playlists.csv',
'final_tracks.csv','main_dataset.csv', ]


for filename in filename_list:

    # Read in amazon reviews. Reminder: Tab-separated values files

    print(f"Working on file: {filename}")

    reviews_df = pd.read_csv(f"{filepath}/{filename}", sep=',',
on_bad_lines='skip')

    perform_EDA(reviews_df,filename)


# In[5]:

# Iterate through the filenames

for filename in filename_list:

    # Read the file into a DataFrame

    print(f"Working on file: {filename}")

    globals()[filename.split('.')[0] + '_df'] =
pd.read_csv(f"{filepath}/{filename}", sep=',', on_bad_lines='skip')
```

```
# DataFrames named  based on the filenames

# e.g., artists_df, final_playlists_df, final_tracks_df, main_dataset_df


# In[6]:

#View current data frames



# In[7]:

artists_df.head(5)



# In[8]:

final_playlists_df.head(2)



# In[9]:

#Note: multiple arists included in artists_uris and mulitple playlists in
playlists_uri

final_tracks_df.head(2)



# In[10]:

main_dataset_df.head(2)



# In[11]:

#Drop unamed col in final_playlists_df

final_playlists_df = final_playlists_df.drop(columns=['Unnamed: 0'])

final_playlists_df.head(2)



# In[12]:
```

```
#Drop unamed col in final_tracks_df

final_tracks_df = final_tracks_df.drop(columns=['Unnamed: 0'])

final_tracks_df.head(2)




# In[13]:

#expand artists_uri in final_tracks_df

Exp_final_tracks_df = final_tracks_df.copy()

Exp_final_tracks_df['artists_uris'] =
Exp_final_tracks_df['artists_uris'].apply(eval)  # Convert string
representation of list to list

Exp_final_tracks_df = Exp_final_tracks_df.explode('artists_uris')

Exp_final_tracks_df.head(2)




# In[14]:

#expand artists_uri in final_tracks_df

Exp_final_tracks_df['playlist_uris'] =
Exp_final_tracks_df['playlist_uris'].apply(eval)  # Convert string
representation of list to list

Exp_final_tracks_df = Exp_final_tracks_df.explode('playlist_uris')

Exp_final_tracks_df.head(2)




# In[15]:

# Merging Exp_final_tracks_df with artists_df

merged_df = Exp_final_tracks_df.merge(artists_df, left_on='artists_uris',
right_on='artist_uri', how='left')

merged_d=merged_df.drop(columns=['artist_uri'], inplace=True)




# In[16]:
```

```
merged_df.head(2)



# In[17]:

# Merging Exp_final_tracks_df with final_playlists_df

merged_df = merged_df.merge(final_playlists_df, left_on='playlist_uris',
right_on='uri', how='left')

merged_d=merged_df.drop(columns=['uri'], inplace=True)



# In[18]:

merged_df.head(200)



# In[19]:

# Define the columns to keep from main_dataset_df

columns_to_keep = ['track_uri', 'danceability', 'instrumentalness',
'liveness', 'valence', 'tempo', 'duration_ms', 'time_signature']

merged_df = merged_df.merge(main_dataset_df[columns_to_keep], on='track_uri',
how='inner')

merged_df



# In[22]:

merged_df = merged_df.rename(columns={

    'name_x': 'track_name',

    'popularity': 'track_popularity',

    'name_y': 'playlist_name',

    'description': 'playlist_description'

})



# In[24]:
```

```
merged_df



# In[30]:

merged_df.columns.tolist()



# In[29]:

merged_df.isnull().sum()



# In[32]:

merged_df.describe()

#!/usr/bin/env python
# coding: utf-8
```

Pickle-merged-visualizations

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:



import pandas as pd



# In[2]:
```

```
filepath= 'gs://my-bucket-mpat/landing/outputSPM2.csv'
filepath2= 'gs://my-bucket-mpat/cleaned/Clean_data.parquet'


# In[3]:


pickle_df=pd.read_csv(filepath)


# In[4]:


pickle_df


# In[5]:


pickle_df.isnull().sum()


# In[6]:


clean_df=pd.read_parquet(filepath2)


# In[7]:
```

```
clean_df


# In[8]:


final_df=clean_df.merge(pickle_df, on='track_uri',how='left')


# In[9]:


final_df.columns.tolist()


# In[10]:


final_df.drop(columns=['tempo_y','time_signature_y','key'], inplace=True)


# In[11]:


final_df


# In[12]:


final_df.isnull().sum()
```

```
# In[13]:


final_df.dropna(axis=0, inplace=True)


# In[14]:


final_df.isnull().sum()


# In[15]:


#saving to landing as a parquet
import pyarrow.parquet as pq
import pyarrow as pa
import gcsfs


# In[16]:


patable = pa.Table.from_pandas(final_df)

gbucket = 'my-bucket-mpat'
gcs_path = 'gs://{}/cleaned/Final_clean_data.parquet'.format(gbucket)

table = pa.Table.from_pandas(final_df)
```

```
files = gcsfs.GCSFileSystem()


# Write PyArrow Table to Parquet file on GCS
with files.open(gcs_path, 'wb') as f:
    pq.write_table(table, f)



# In[17]:



final_df.describe()



# In[18]:



# Assuming 'df' is your DataFrame with the specified columns
desired_columns = ['valence', 'tempo_x', 'duration_ms', 'time_signature_x',
'num_samples', 'duration', 'loudness']


# Use describe() on the desired subset of columns
final_df[desired_columns].describe()



# In[ ]:





# In[19]:
```

```
import matplotlib.pyplot as plt

import seaborn as sns



# In[20]:



plt.figure(figsize=(12, 5))

sns.countplot(x='track_name', data=final_df,
order=final_df['track_name'].value_counts().index[:10])

plt.title('Top 10 Most Popular Track Names')

plt.xlabel('Track Name')

plt.ylabel('Count')

plt.xticks(rotation=45)

plt.show()



# In[21]:



plt.figure(figsize=(20, 6))

sns.countplot(x='playlist_name', data=final_df,
order=final_df['playlist_name'].value_counts().index[:20])

plt.title('Top 10 Most Popular Track Names')

plt.xlabel('Playlist Names')

plt.ylabel('Count')

plt.xticks(rotation=45)

plt.show()



# In[22]:
```

```
plt.figure(figsize=(10, 6))

sns.histplot(final_df['track_popularity'], bins=20, kde=True)

plt.title('Distribution of Track Popularity')

plt.xlabel('Popularity Score')

plt.ylabel('Frequency')

plt.show()



# In[37]:



plt.figure(figsize=(10, 6))

sns.histplot(final_df['artist_popularity'], bins=50, kde=True)

plt.title('Distribution of artist Popularity')

plt.xlabel('Popularity Score')

plt.ylabel('Frequency')

plt.show()



# In[24]:



plt.figure(figsize=(15, 6))

sns.histplot(final_df['danceability'], kde=True)

plt.title('Distribution of danceability')

plt.xlabel('danceability')

plt.ylabel('Density')

plt.show()



# In[25]:
```

```
import seaborn as sns

import matplotlib.pyplot as plt


# Example: Boxplot of 'tempo_x' to visualize outliers

plt.figure(figsize=(10, 6))

sns.boxplot(x='loudness', data=final_df)

plt.title('Boxplot of Tempo')

plt.xlabel('instrumentalness')

plt.show()



# In[38]:



Q1 = final_df['tempo_x'].quantile(0.25)

Q3 = final_df['tempo_x'].quantile(0.75)

IQR = Q3 - Q1


# Define lower and upper bounds for outlier detection

lower_bound = Q1 - 1.5 * IQR

upper_bound = Q3 + 1.5 * IQR


# Identify outliers

outliers_iqr = final_df['tempo_x'][(final_df['tempo_x'] < lower_bound) |
(final_df['tempo_x'] > upper_bound)]

print("Outliers based on IQR:")

print(outliers_iqr)



# In[39]:
```

```
plt.figure(figsize=(15, 6))

sns.histplot(final_df['tempo_x'], kde=True)

plt.title('Distribution of tempo_x')

plt.xlabel('tempo_x')

plt.ylabel('Density')

plt.show()



# In[40]:



# Filter out outliers from 'final_df'

final_df = final_df[~final_df['tempo_x'].isin(outliers_iqr)]


# Display the shape of the new DataFrame without outliers

print("Shape of DataFrame without outliers:", non_outliers_df.shape)



# In[42]:



#View new distribution

plt.figure(figsize=(15, 6))

sns.histplot(final_df['tempo_x'], kde=True)

plt.title('Distribution of tempo_x')

plt.xlabel('tempo_x')

plt.ylabel('Density')

plt.show()
```

```
# In[46]:


#View new distribution
plt.figure(figsize=(15, 6))
sns.histplot(final_df['instrumentalness'], kde=True)
plt.title('Distribution of instrumentalness')
plt.xlabel('instrumentalness')
plt.ylabel('Density')
plt.show()


# In[49]:


#View new distribution
plt.figure(figsize=(15, 6))
sns.histplot(final_df['valence'], kde=True)
plt.title('Distribution of valence')
plt.xlabel('valence')
plt.ylabel('Density')
plt.show()


# In[ ]:
```

Appendix C

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import pandas as pd

# Load DataFrame from CSV
merged_df = pd.read_csv('merged_data.csv')


# In[2]:


merged_df


# In[3]:


colDrop = ['album_type', 'is_playable', 'release_date', 'playlist_description']

# Drop columns using the list of column names
merged_df.drop(columns=colDrop, inplace=True)


# In[4]:
```

```
# Display the current column names in merged_df
print(merged_df.columns)


# In[5]:


merged_df


# In[6]:


# Drop rows with any missing values
merged_df.dropna(axis=0, inplace=True)
print("Shape after dropping rows with missing values:", merged_df.shape)


# In[7]:


merged_df.isnull().sum()


# In[8]:


Clean_merged_df=merged_df.copy()


# In[9]:
```

```
Clean_merged_df


# In[11]:


Clean_merged_df.to_csv('home/malika_patel1091/Clean_data.csv', index=False)


# In[15]:


#saving to landing as a parquet
import pyarrow.parquet as pq
import pyarrow as pa
import gcsfs


# In[18]:


patable = pa.Table.from_pandas(Clean_merged_df)

gbucket = 'my-bucket-mpat'
gcs_path = 'gs://{}/cleaned/Clean_data.parquet'.format(gbucket)

table = pa.Table.from_pandas(Clean_merged_df)

files = gcsfs.GCSFileSystem()

# Write PyArrow Table to Parquet file on GCS
```

```
with files.open(gcs_path, 'wb') as f:
    pq.write_table(table, f)




# In[ ]:
```

Appendix D

```
#!/usr/bin/env python
# coding: utf-8

# In[2]:



from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression
from pyspark.ml import Pipeline
from pyspark.ml.evaluation import RegressionEvaluator



# In[3]:



spotify_sdf=spark.read.parquet('gs://my-bucket-mpat/cleaned/Final_clean_data2.parquet')



# # TEST LINEAR REG

# In[4]:



features = ['artist_followers','danceability', 'instrumentalness', 'liveness', 'valence', 'energy', 'key',
            'speechiness', 'acousticness', 'duration', 'loudness', 'tempo', 'time_signature', 'artist_popularity']

label='track_popularity'
```

```
# In[5]:



# Assemble features into a single feature vector column
assembler = VectorAssembler(inputCols=features, outputCol="features")

# Initialize Linear Regression model
lr = LinearRegression(labelCol="track_popularity", featuresCol="features")

# Split the data into training and testing sets (80% training, 20% testing)
(train_data, test_data) = spotify_sdf.randomSplit([0.8, 0.2], seed=123)

# Define Pipeline
pipeline = Pipeline(stages=[assembler, lr])




# In[6]:



from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.evaluation import BinaryClassificationEvaluator



evaluator = RegressionEvaluator(labelCol='track_popularity')



# Create a grid to hold hyperparameters
```

```
grid = ParamGridBuilder()

# Build the parameter grid
grid = grid.build()

# Create the CrossValidator using the hyperparameter grid
cv = CrossValidator(estimator=pipeline, estimatorParamMaps=grid, evaluator=evaluator, numFolds=3)

# Train the models
all_models = cv.fit(train_data)

# Get the best model from all of the models trained
bestModel = all_models.bestModel

# Use the model 'bestModel' to predict the test set
test_results = bestModel.transform(test_data)

# Show the predicted tip
test_results.select('prediction','track_popularity','artist_followers','danceability', 'instrumentalness',
'liveness', 'valence', 'energy', 'key',
            'speechiness').show(truncate=False)

# Calculate RMSE and R2
rmse = evaluator.evaluate(test_results, {evaluator.metricName:'rmse'})
r2 =evaluator.evaluate(test_results,{evaluator.metricName:'r2'})
print(f"RMSE: {rmse} R-squared:{r2}")


# # DURATION SCALING
# In[7]:


train_data.printSchema()


# In[8]:


train_data.select(features).show(truncate=False)


# In[9]:


spotify_sdf.groupby('time_signature').count().show()


# In[10]:


spotify_sdf.groupby('key').count().show()


# In[11]:


from pyspark.ml.feature import MinMaxScaler
```

```
dur_assembler = VectorAssembler(inputCols=['duration'], outputCol='durationVector')
spotify_sdf = dur_assembler.transform(spotify_sdf)


dur_scaler = MinMaxScaler(inputCol="durationVector", outputCol="DurScaled")
spotify_sdf = dur_scaler.fit(spotify_sdf).transform(spotify_sdf)




# In[12]:


spotify_sdf.select('DurScaled','durationVector','duration').show()


# # SECON TEST

# In[13]:


features = ['artist_followers','danceability', 'instrumentalness', 'liveness', 'valence', 'energy', 'key',
            'speechiness', 'acousticness', 'DurScaled', 'loudness', 'tempo', 'time_signature','artist_popularity']

# Assemble features into a single feature vector column
assembler = VectorAssembler(inputCols=features, outputCol="features")

# Initialize Linear Regression model
lr = LinearRegression(labelCol="track_popularity", featuresCol="features")
```

```
# Split the data into training and testing sets (80% training, 20% testing)
(train_data, test_data) = spotify_sdf.randomSplit([0.8, 0.2], seed=123)

# Define Pipeline
pipeline = Pipeline(stages=[assembler, lr])



# In[14]:



evaluator = RegressionEvaluator(labelCol='track_popularity')



# Create a grid to hold hyperparameters
grid = ParamGridBuilder()

# Build the parameter grid
grid = grid.build()

# Create the CrossValidator using the hyperparameter grid
cv = CrossValidator(estimator=pipeline, estimatorParamMaps=grid, evaluator=evaluator, numFolds=3)

# Train the models
all_models = cv.fit(train_data)

# Get the best model from all of the models trained
bestModel = all_models.bestModel

# Use the model 'bestModel' to predict the test set
test_results = bestModel.transform(test_data)
```

```
# Show the predicted tip
test_results.select('prediction','track_popularity','artist_followers','danceability', 'instrumentalness',
'liveness', 'valence', 'energy', 'key',
                'speechiness').show(truncate=False)


# Calculate RMSE and R2
rmse = evaluator.evaluate(test_results, {evaluator.metricName:'rmse'})
r2 =evaluator.evaluate(test_results,{evaluator.metricName:'r2'})
print(f"RMSE: {rmse} R-squared:{r2}")



# # LOUDNESS SCALING


# In[15]:



dur_assembler = VectorAssembler(inputCols=['loudness'], outputCol='loudnessVector')
spotify_sdf = dur_assembler.transform(spotify_sdf)



dur_scaler = MinMaxScaler(inputCol="loudnessVector", outputCol="loudnessScaled")
spotify_sdf = dur_scaler.fit(spotify_sdf).transform(spotify_sdf)




# In[16]:



spotify_sdf.select('loudnessScaled','loudnessVector','loudness').show()
```

```
# # THIRD TEST


# In[17]:



features = ['artist_followers','danceability', 'instrumentalness', 'liveness', 'valence', 'energy', 'key',
            'speechiness', 'acousticness', 'DurScaled', 'loudnessScaled', 'tempo',
'time_signature','artist_popularity']

# Assemble features into a single feature vector column
assembler = VectorAssembler(inputCols=features, outputCol="features")

# Initialize Linear Regression model
lr = LinearRegression(labelCol="track_popularity", featuresCol="features")

# Split the data into training and testing sets (80% training, 20% testing)
(train_data, test_data) = spotify_sdf.randomSplit([0.8, 0.2], seed=123)

# Define Pipeline
pipeline = Pipeline(stages=[assembler, lr])



# In[18]:



evaluator = RegressionEvaluator(labelCol='track_popularity')
```

```
# Create a grid to hold hyperparameters
grid = ParamGridBuilder()

# Build the parameter grid
grid = grid.build()

# Create the CrossValidator using the hyperparameter grid
cv = CrossValidator(estimator=pipeline, estimatorParamMaps=grid, evaluator=evaluator, numFolds=3)

# Train the models
all_models = cv.fit(train_data)

# Get the best model from all of the models trained
bestModel = all_models.bestModel

# Use the model 'bestModel' to predict the test set
test_results = bestModel.transform(test_data)

# Show the predicted tip
test_results.select('prediction','track_popularity','artist_followers','danceability', 'instrumentalness', 
'liveness', 'valence', 'energy', 'key',
            'speechiness').show(truncate=False)

# Calculate RMSE and R2
rmse = evaluator.evaluate(test_results, {evaluator.metricName:'rmse'})
r2 =evaluator.evaluate(test_results,{evaluator.metricName:'r2'})
print(f"RMSE: {rmse} R-squared:{r2}")


# # ENCODING KEY
# In[20]:


from pyspark.ml.feature import OneHotEncoder, StringIndexer

indexer = StringIndexer(inputCols=['key'], outputCols=['KeyIndex'])
spotify_sdf = indexer.fit(spotify_sdf).transform(spotify_sdf)


# In[21]:


encoder = OneHotEncoder(inputCols=['KeyIndex'], outputCols=['KeyVector'], dropLast=False)
spotify_sdf = encoder.fit(spotify_sdf).transform(spotify_sdf)


# # FOURTH TEST

# In[22]:


features = ['artist_followers','danceability', 'instrumentalness', 'liveness', 'valence', 'energy', 'KeyVector',
            'speechiness', 'acousticness', 'DurScaled', 'loudnessScaled', 'tempo',
'time_signature','artist_popularity']

# Assemble features into a single feature vector column
assembler = VectorAssembler(inputCols=features, outputCol="features")

# Initialize Linear Regression model
```

```
lr = LinearRegression(labelCol="track_popularity", featuresCol="features")

# Split the data into training and testing sets (80% training, 20% testing)
(train_data, test_data) = spotify_sdf.randomSplit([0.8, 0.2], seed=123)

# Define Pipeline
pipeline = Pipeline(stages=[assembler, lr])



# In[23]:



evaluator = RegressionEvaluator(labelCol='track_popularity')



# Create a grid to hold hyperparameters
grid = ParamGridBuilder()

# Build the parameter grid
grid = grid.build()

# Create the CrossValidator using the hyperparameter grid
cv = CrossValidator(estimator=pipeline, estimatorParamMaps=grid, evaluator=evaluator, numFolds=3)

# Train the models
all_models = cv.fit(train_data)

# Get the best model from all of the models trained
bestModel = all_models.bestModel
```

```
# Use the model 'bestModel' to predict the test set
test_results = bestModel.transform(test_data)

# Show the predicted tip
test_results.select('prediction','track_popularity','artist_followers','danceability', 'instrumentalness',
'liveness', 'valence', 'energy', 'key',
            'speechiness').show(truncate=False)

# Calculate RMSE and R2
rmse = evaluator.evaluate(test_results, {evaluator.metricName:'rmse'})
r2 =evaluator.evaluate(test_results,{evaluator.metricName:'r2'})
print(f"RMSE: {rmse} R-squared:{r2}")

# Create a grid to hold hyperparameters

grid = ParamGridBuilder()

# Add hyperparameters to the grid
grid = grid.addGrid(lr.regParam, [0.0, 0.2, 0.4, 0.6, 0.8, 1.0])
grid = grid.addGrid(lr.elasticNetParam, [0, 1])

# Build the grid
grid = grid.build()

print('Number of models to be tested: ', len(grid))

# Create the CrossValidator using the pipeline and the new hyperparameter grid
cv = CrossValidator(estimator=pipeline, estimatorParamMaps=grid, evaluator=evaluator, numFolds=3)

# Call cv.fit() to create models with all of the combinations of parameters in the grid
```

```
all_models = cv.fit(train_data)


# Print average metrics for each model
print("Average Metrics for Each model: ", all_models.avgMetrics)


bestModel = cv_model.bestModel



lr_model = bestModel.stages[-1]



coefficients = lr_model.coefficients
intercept = lr_model.intercept


print("Intercept:", intercept)
for i, feature in enumerate(features):
    print(f"Coefficient for {feature}: {coefficients[i]}")


# In[26]:



spotify_sdf.write.mode("overwrite").format("parquet").save("gs://my-bucket-
mpat/trusted/spotifyDF_features.parquet")


# In[ ]:


all_models.save("gs://my-bucket-mpat/Model/all_LR_models")
```

Appendix E

```
#!/usr/bin/env python
# coding: utf-8

# In[91]:


import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt


# In[7]:


df= pd.read_parquet('gs://my-bucket-mpat/Model/Test_results.parquet/part-00000-674fbb85-8126-45c5-
afbe-11de24982637-c000.snappy.parquet')


# In[34]:


#VIS 1
plt.hist(df.track_popularity, bins=100, alpha=0.5, label='Actual Track Popularity', color='blue')
plt.hist(df.prediction, bins=100, alpha=0.5, label='Predicted', color='red')
plt.xlabel('Track Popularity')
plt.ylabel('Frequency')
plt.title('Distribution of Actual Track Popularity vs Predicted Petal Width')
plt.legend()
plt.show()


# In[85]:


plt.hexbin(x=df.energy, y=df.track_popularity, gridsize=20, cmap='plasma')
plt.colorbar(label='Count in Bin')
plt.xlabel('Energy')
plt.ylabel('Actual Track Popularity')
plt.title('Hexbin Plot of Energy vs Actual Track Popularity')
plt.show()


# In[76]:


plt.hexbin(x=df.energy, y=df.prediction, gridsize=20, cmap='plasma')
plt.colorbar(label='Count in Bin')
plt.xlabel('Energy')
plt.ylabel('Predicted Track Popularity')
plt.title('Hexbin Plot of Energy vs Predicted Track Popularity')
plt.show()
```

```
# In[95]:


features = ['danceability', 'instrumentalness', 'liveness', 'energy', 'valence',
            'key', 'speechiness', 'time_signature', 'tempo', 'acousticness',
            'track_popularity', 'prediction', ]

# Assuming you are using pandas DataFrame
correlation_matrix = df[features].corr()

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=.5)
plt.title('Correlation Matrix')
plt.show()


# In[ ]:
```