STAT 482 Final Report: Spotify Streaming Data Analysis

Jaden Oca

Spring 2024

1 Introduction

Spotify is the largest music streaming platform in the world with over 574 million users. At the end of each year, Spotify releases it's annual "Wrapped," which is unique to each user and summarizes their listening activity throughout the year, including top songs, artists, and genres listened to.

The purpose of this project is to use my own listening data to extract trends beyond what Spotify's "Wrapped" gives us, including a time-series analysis of my listening trends over time, seeing how listening to certain songs and artists peak and fall over time.

More specifically, the project hopes to observe if any inflection points can be traced back to important events in the user's life, answering the question "can a shift in music taste shift be traced back to life events?"

By creating a more in-depth analysis of my own listening data, I hope to lay the groundwork for this project's formalization into a package that will allow anyone can analyze their Spotify listening trends on a deeper level.

2 Description of Dataset

2.1 Source

The data set used was pulled directly from Spotify. It tracks every single one of my Spotify streams ranging from June 24th, 2022 to January 5th, 2024.

2.2 Data

The data contains four columns. They are as follows:

- endTime - time in UTC that the song finished by the minute.

- artistName - name of artist of each track streamed
- trackName - name of each track streamed
- msPlayed - duration each track was listened to

The image below is an example of the head of the dataframe.

	endTime	artistName	trackName	msPlayed
0	2022-06-25 00:01	No Rome	Narcissist (feat. The 1975)	195501
1	2022-06-25 00:04	The 1975	Me & You Together Song	207223
2	2022-06-25 00:08	Valley	homebody	191086
3	2022-06-25 00:11	One Direction	Steal My Girl	228133
4	2022-06-25 00:15	HONNE	Day 1 ◑	233600

Figure 1: Head of dataframe

Evident from Figure 1 (above), there are 4 columns as mentioned before, endTime, artistName, trackName, and msPlayed.

3 Exploratory Data Analysis and Results

The exploratory data anaylsis for this project involved looking at top songs listened to, top tracks listened to, and observing visual trends on a time series line chart of listening behavior over the recorded period.

3.1 Artist and Track time series

To begin the analysis, observed was the length, top artists, and top tracks,

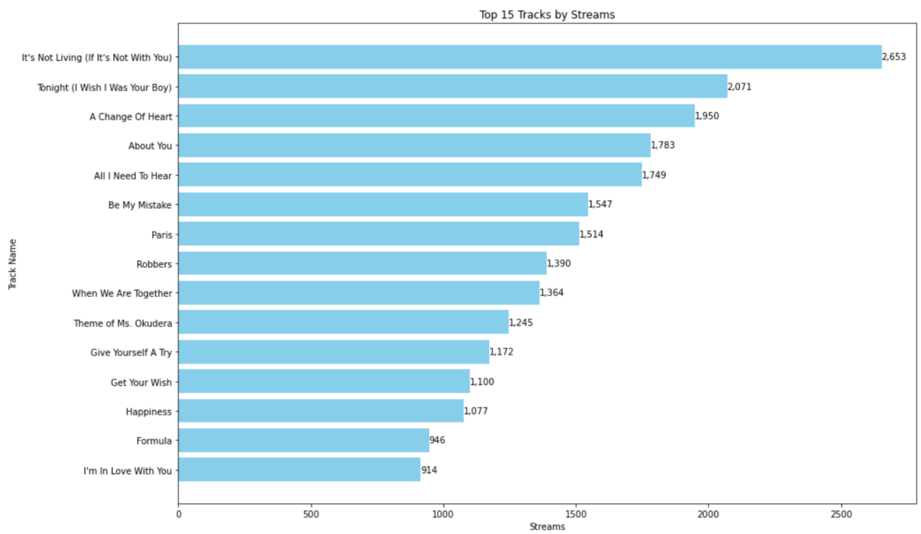

Figure 2: Stream counts of top 15 tracks

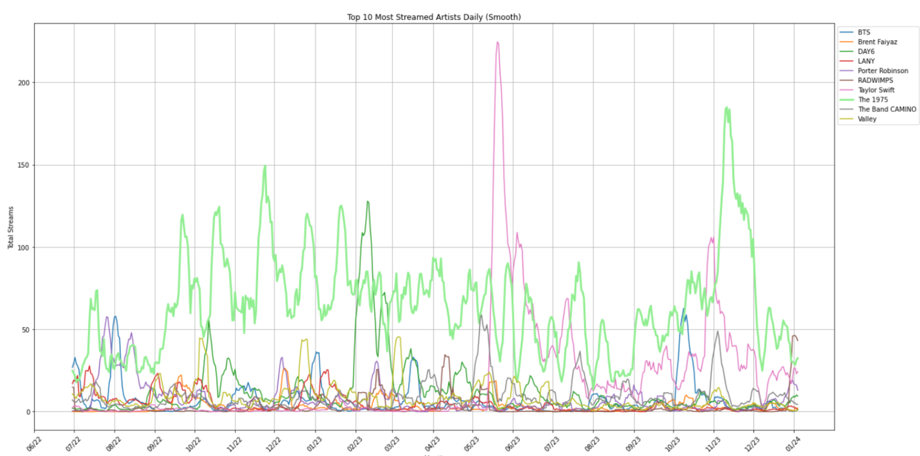

Figure 3: Top artists over time (smooth)

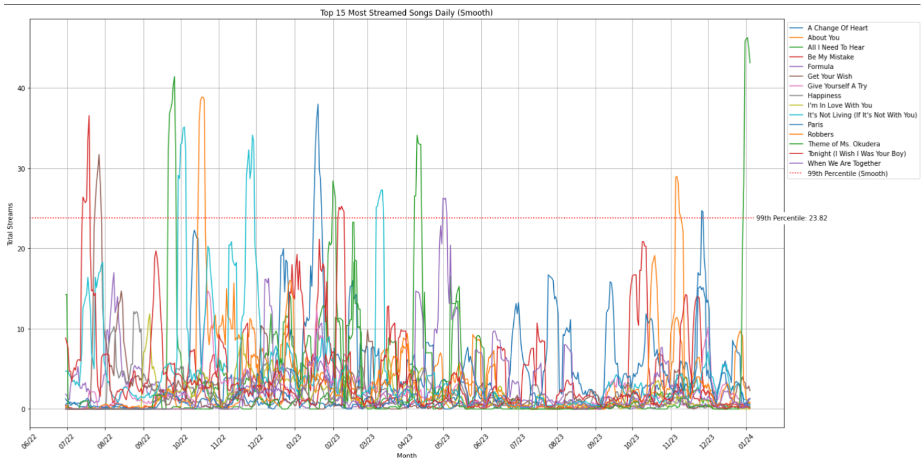

Figure 5: Top tracks over time (raw)

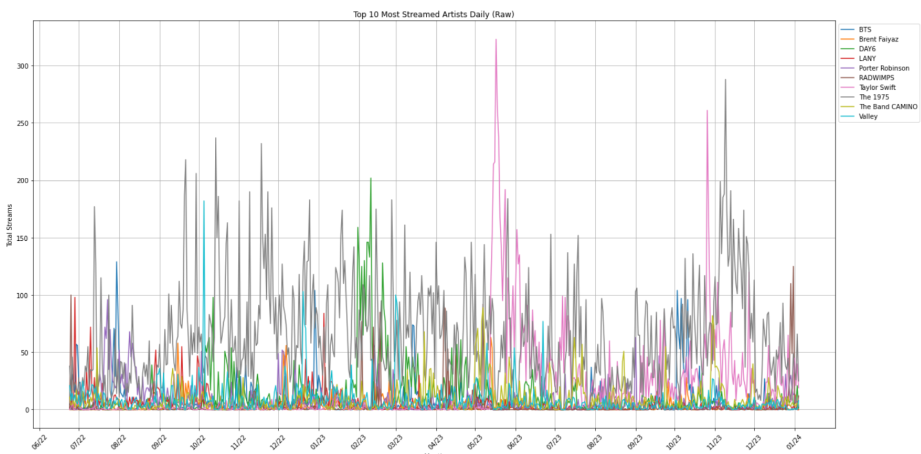

Figure 4: Top artists over time (raw)

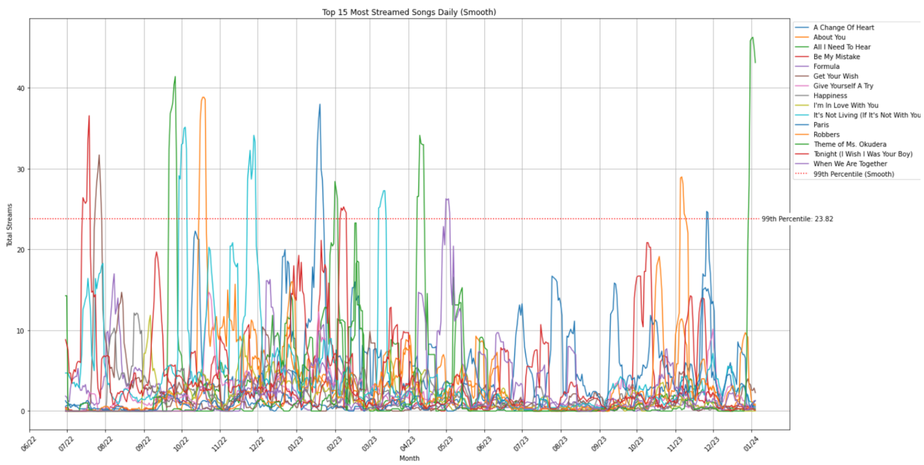

Figure 6: Top tracks over time (smooth)

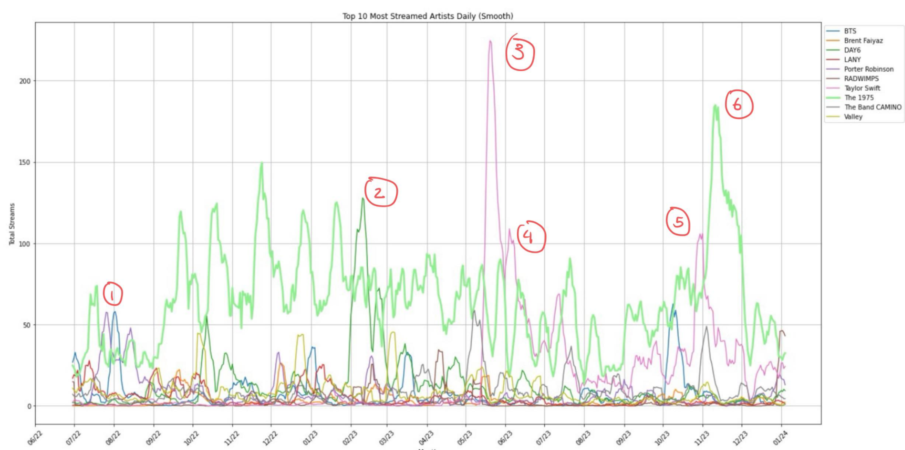

Figure 7: Labeled Top Artists over time (smooth)

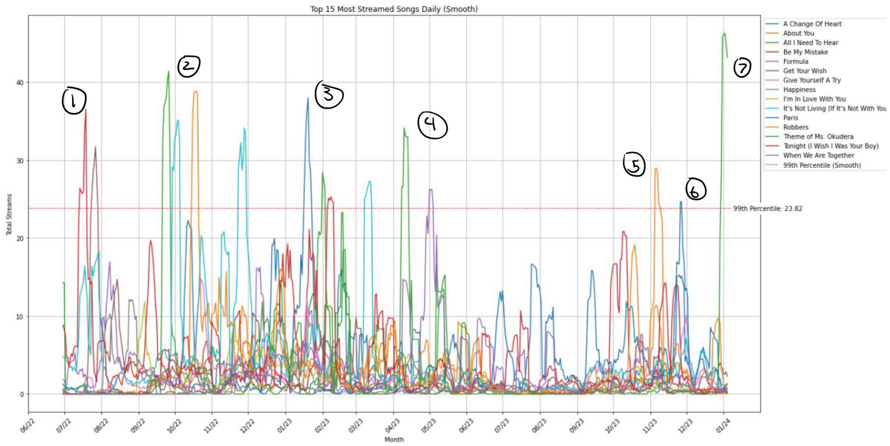

Figure 8: Labeled Top tracks over time (smooth)

Figure 3 shows the top artists listened to over time from June 2022 to January 2024. Evident from the many spikes in the graph, there is much fluctuation over time concerning which artist the user listens to with higher frequency, often changing from day to day. To make the visualization easier to process, a smoothed version is showed in figure 4 using a 7-day moving average.

Figure 4 shows a smoothed version of the top artists listened to over time from June 2022 to January 2024. Highlighted in light green is the moving average of streams per day for The 1975, the top artist listened to by the user. The 1975 is by far the user's most listened artist, so perhaps the time periods where another artist's moving average surpasses that of The 1975 represents an important period of turbulence for the user.

Figure 5 shows the top tracks listened to over time from June 2022 to January 2024. Even more so than the line chart for artists, the visualization is difficult to interpret.

Figure 6 shows a smoothed version of the top tracks listened to over time from June 2022 to January 2024. Furthermore, a line has been drawn at the 99th percentile of entries on the line chart, representing the top 1 percent of song streams per day among the top 15 tracks. One can safely assume that values above that line are very abnormal based on the user's listening history, indicating that these inflection points may be points of interest as the project progresses.

In the following section, major inflection points will be analyzed and traced back to the user's life.

Figure 7 is similar to figure 4, but it also contains labels for inflection points of interest for artists listened to.

1. Turbulent time in Summer with lots of life changes
2. Devastating hand injury
3. Vacation in Europe
4. Moving to New York
5. Release of 1989 Taylor's Version
6. Attending concert for The 1975

Figure 8 is similar to figure 6, but it also contains labels for inflection points of interest for tracks listened to.

1. Summer 2022
2. Fall 2022 Semester
3. New Year's, start of Spring 2023 Semester
4. 2022-2023 School year ending
5. Attending The 1975 concert
6. Thanksgiving break
7. Winter break

Interestingly, many of these behavioral changes occur during periods of change, such as breaks from school or the start of new semesters. This may be an indication that when one's environment changes, their listening behavior may change as well.

Many changes can also be seen in both the track and artist visualizations. Furthermore, a large gap in inflection points can be observed between May 2023 and November 2023 on Figure 8. This may be evident of a calmer lifestyle that includes less big changes.

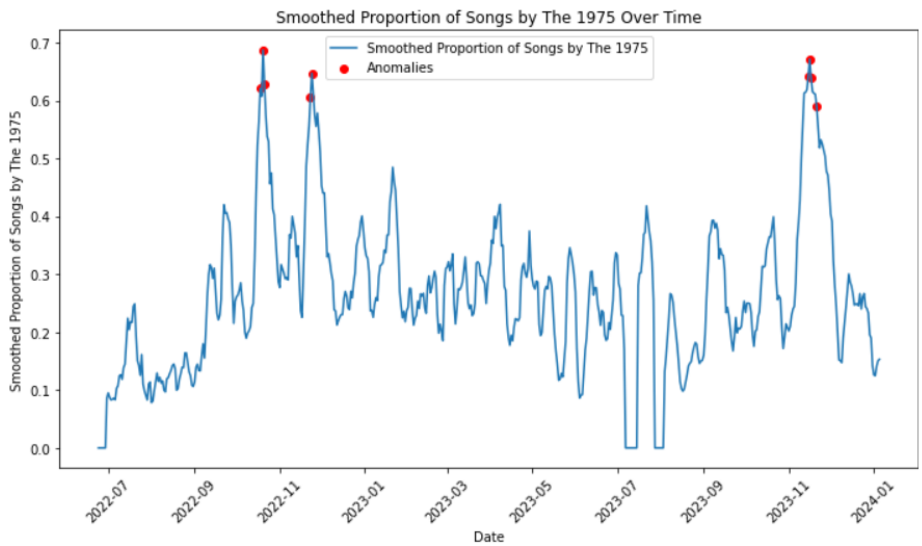

Figure 9: Abnormality Detection High

4 Isolation Forest High Anomaly Detection

This next section discusses the fit of an Isolation Forest to the data to detect anomalies in listening behavior. An isolation forest works by isolating anomalies (outliers) in a dataset by randomly partitioning the data into subsets, making it more efficient in detecting anomalies compared to traditional methods. It identifies anomalies as points that are easier to isolate due to their uniqueness, making them stand out from the majority of normal data points.

Figure 9 illustrates the output from using Isolation Forest anomaly detection to find abnormalities in our smoothed averages. The y-axis represents the proportion of The 1975 tracks of total streams over a 7-day period. The 'red' abnormal points represent statistical anomalies as detected by the random forest using a contamination rate of 0.05.

As we can see, the three main anomaly areas are October 2022, November 2022, and November 2023. This means that during these periods, I was listening to an abnormally high amount of The 1975 as a proportion of my total listens. Because the October 2022 and November 2022 period are adjacent, I believe that it's safe to group them as one "event."

I believe that these line up with life events that I was experiencing because in both November 2022 and November 2023 I watched The 1975 live in concert. This is evidence to suggest that when I see an artist live in concert, I have a tendency to increase how much I listen to them in the near future. For a more extensive analysis, I would like to observe how each concert I goes to affects my listening to each artist, beyond just my top artist.

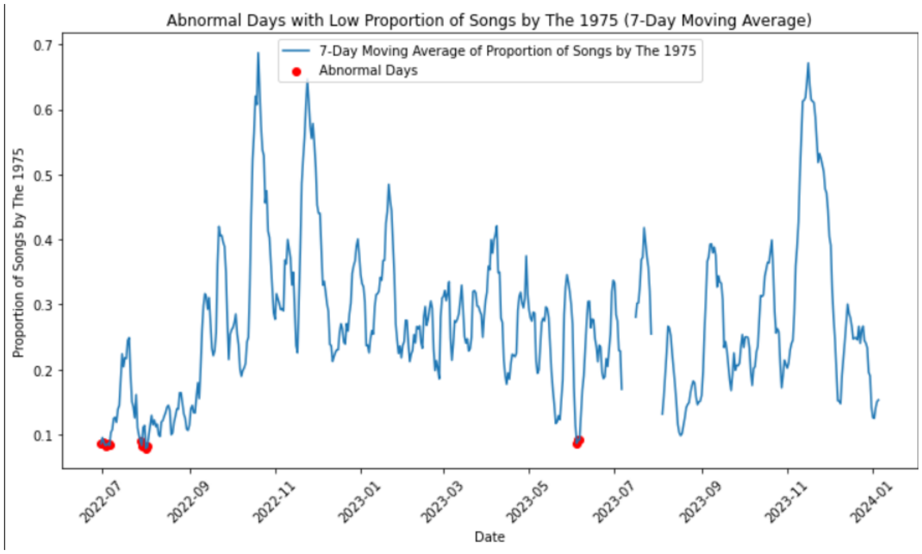

Figure 10: Caption

5 Z-Score Low Anomaly Detection

Figure 10 (featured above) expresses the output of z-score anomaly detection in Python using our time series data. The threshold for anomaly was set using a z-score of -1.5. This means that proportion values whose z-score falls under -1.5 were marked as anomalies and denoted with a red dot in our visualization.

As you can see, the abnormal lows in our data are in July 2022, August 2022, and May 2023. Since July 2022 and August 2022 are adjacent, I also believe that it suffices to group them as the same "event" in my timeline.

This means that the two outlier instances in my listening are late summer 2022 and early summer of 2023. This might suggest that summer is one of the most regularly turbulent periods of my life where I experience the most change. In summer 2022, I was doing a lot of new things and sorting out how I wanted to approach the upcoming school year. During the May 2023 dip, I was traveling to London, Paris, Italy, and eventually settling in New York following the conclusion of the May 2023 semester.

This data suggests that I experience the most change in listening when there is a lot of change in my life, whether it be mentally (Summer 2022) or physically/geographically (Summer 2023). I would love to spend more time further exploring whether or not the same would be true in the data of other users.

6 Time Series Decomposition

Time series decomposition is a statistical method used to break down a time series into its constituent components, typically trend, seasonality, and noise, enabling the identification and analysis of underlying patterns and trends within the data.

By applying such decomposition to my listening data of The 1975, we hope to find trends of seasonality in the listening data.

Figure 11: Time Series Decomposition

Based on Figure 11 shown above, there is no evident trend in seasonality. It ranges from -5 to 5, which is a too small when compared to the bulk of daily listens to be considered relevant. Interestingly, the trend graph roughly follows the overall graph, further expressing the lack of seasonality evident from the decomposition. Lastly, the residual graph shows heightened variance in the first half of the dataset, with "spikes" in residuals occurring every few months. These high variance periods could be indicative of times where listening behavior is shifting.

7 Conclusion

The most important findings in this data indicate that my listening trends shift the most during summer, likely to both physical and non-physical aspects of my life changing drastically during these times. Examples of these changes include the start of new semesters and traveling to new cities in the world. In other words, shifts in my music taste reflect major changes in my own life.

Despite these findings, this project has even more potential to grow in the future. Suggestions for future analysis include:

1. Formalizing code into a package or application which will allow anyone to easily visualize their exact listening trends.

2. Using sentiment analysis to analyze song titles or lyrics. Lyrics can be scraped using Genius.com's API. Looking at common words or phrases could be more indicative of "emotion-based" listening trends at different times of day or eras.

3. Collecting more data over time. Since there has only been two summers, it's difficult to generalize trends occurring annually each summer. Collecting more data over more years would allow a more seasoned analysis of how listening trends vary throughout the year.

In conclusion, this project reveals a potential relationship between Spotify listening behavior and life events while the proposed future analyses, including formalizing code, sentiment analysis on lyrics, and longitudinal data collection, promise further insights into this dynamic relationship. These avenues offer valuable opportunities for furthering our understanding of listener behavior alongside possible standardization into something that anyone can use.

8 References

1. GeeksforGeeks. (n.d.). What is Isolation Forest? Retrieved from `https://www.geeksforgeeks.org/what-is-isolation-forest/`

2. ChatGPT. (n.d.). Retrieved from `https://chatgpt.com/`

3. Spotify. (n.d.). Data Rights and Privacy Settings. Retrieved from `https://support.spotify.com/us/article/data-rights-and-privacy-settings/`

9 Appendix

```
# %%
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")

# %%
df_2022 = pd.read_csv("Streams_0722_0723.csv")
```

```python

# %%
df = pd.read_json("StreamingHistory0.json")
df1 = pd.read_json("StreamingHistory1.json")
df2 = pd.read_json("StreamingHistory2.json")
df3 = pd.read_json("StreamingHistory3.json")
df4 = pd.read_json("StreamingHistory4.json")
df5 = pd.read_json("StreamingHistory5.json")
df6 = pd.read_json("StreamingHistory6.json")
df7 = pd.read_json("StreamingHistory7.json")
df8 = pd.read_json("StreamingHistory8.json")

# %%
dflist = [df_2022, df, df1, df2, df3, df4, df5, df6, df7, df8]
master_stream_df = pd.concat(dflist)
if 'Unnamed: 0' in master_stream_df.columns:
    master_stream_df.drop(columns=['Unnamed: 0'], inplace=True)
master_stream_df.head()


# %% [markdown]
# Next, we will check the length of the df

# %%
print("Length of combined df including duplicates: {}".format(len(master_stream_df)))
master_stream_df.drop_duplicates(inplace=True)
print("Length of combined df without duplicates: {}".format(len(master_stream_df)))


# %%
master_stream_df.head()

# %%
print(master_stream_df.iloc[0,0])
print(master_stream_df.iloc[len(master_stream_df)-1],0)

# %%
from pytz import timezone

# Assuming master_stream_df contains your dataframe
master_stream_df['endTime'] = pd.to_datetime(master_stream_df['endTime'])

# Define timezones
utc_timezone = timezone('UTC')
central_timezone = timezone('US/Central')

# Localize the endTime column to UTC timezone
master_stream_df['endTime'] = master_stream_df['endTime'].dt.tz_localize(utc_timezone)

# Convert times from UTC to Central Time and format to retain seconds count
master_stream_df['endTime'] = master_stream_df['endTime'].dt.tz_convert(central_timezone).dt.strftime('%Y-%m-%d %H:%M:%S')
```

```python

# Filter the dataframe to keep rows with endTime in 2023 and reset the index
#filtered_df = master_stream_df[master_stream_df['endTime'].str.startswith('2023')].reset_index(drop=True)

filtered_df = master_stream_df

# %%
len(filtered_df)

# %%
def getTopTracks(df):
    playCounts = df['trackName'].value_counts()
    playCounts_df = pd.DataFrame({'Track Name': playCounts.index, 'Streams': playCounts.values})
    return playCounts_df
def getTopArtists(df):
    artistCounts = df['artistName'].value_counts()
    artistCounts_df = pd.DataFrame({'Artist Name' : artistCounts.index, 'Streams': artistCounts.values})
    return artistCounts_df

# %%
top_filtered = getTopTracks(filtered_df)
top_filtered.head(15)

# %%
unknownFilter = filtered_df['trackName'] != 'Unknown Track'
no_unknown = filtered_df[unknownFilter]
print(len(no_unknown))

# %%
filtered_unknown = getTopTracks(no_unknown)
filtered_unknown.head(15)

# %%
filtered_unknown = filtered_unknown.head(15)
plt.figure(figsize = (15,10))
bars = plt.barh(filtered_unknown['Track Name'], filtered_unknown['Streams'],color = 'skyblue')
for bar in bars:
    plt.text(bar.get_width(), bar.get_y() + bar.get_height() / 2, f'{bar.get_width():,.0f}', va='center')
plt.gca().invert_yaxis()
plt.xlabel('Streams')
plt.ylabel('Track Name')
plt.title('Top 15 Tracks by Streams')
plt.show()

# %%
artist_counts = getTopArtists(no_unknown)
artist_counts.head(20)

# %%
top_artists = artist_counts.head(10)
plt.figure(figsize = (10,10))
```

```python
bars = plt.barh(top_artists['Artist Name'], top_artists['Streams'], color = 'skyblue')
for bar in bars:
    plt.text(bar.get_width(), bar.get_y() + bar.get_height() / 2, f'{bar.get_width():,.0f}', va='center')
plt.gca().invert_yaxis()
plt.xlabel('Streams')
plt.ylabel('Artist Name')
plt.title('Top 10 Artists by Streams')
plt.show()

# %%
no_unknown.head(10)

# %%
import matplotlib.dates as mdates

def plot_top_artists_smooth(df, period, n_artists, plot_type, display_agg=True):
    # Convert 'endTime' to datetime format
    df.loc[:, 'endTime'] = pd.to_datetime(df['endTime'])

    if period == 'Day':
        df.loc[:, 'DaysElapsed'] = (df['endTime'] - df['endTime'].min()).dt.days
        grouped_df = df.groupby(['artistName', 'DaysElapsed']).size().reset_index(name='Counts')
    elif period == 'Week':
        df.loc[:, 'Week'] = df['endTime'].dt.isocalendar().week
        df.loc[:, 'YearWeek'] = df['endTime'].dt.strftime('%Y-%W')
        grouped_df = df.groupby(['artistName', 'YearWeek']).size().reset_index(name='Counts')
    elif period == 'Month':
        df.loc[:, 'Month'] = df['endTime'].dt.to_period('M')
        grouped_df = df.groupby(['artistName', 'Month']).size().reset_index(name='Counts')

    top_artists = grouped_df.groupby('artistName')['Counts'].sum().nlargest(n_artists).index
    top_artists_df = grouped_df[grouped_df['artistName'].isin(top_artists)]

    if plot_type == 'smooth':
        pivot_df = top_artists_df.pivot_table(index=['DaysElapsed'], columns='artistName', values='Counts', fill_value=0)
        plt.figure(figsize=(20, 10))
        for column in pivot_df.columns:
            pivot_df[column] = pivot_df[column].rolling(window=7).mean()
            if column == top_artists[0]:  # Assuming the first artist in the top list is the top artist of the year
                plt.plot(pivot_df.index, pivot_df[column], label=column, linewidth=3, color='lightgreen')  # Adjust linewidth and color as needed
            else:
                plt.plot(pivot_df.index, pivot_df[column], label=
```

```
                    column)

        # Calculate aggregate streams across top artists if display_agg is True
        if display_agg:
            aggregate_streams = pivot_df.sum(axis=1)
            if 'smooth' in plot_type:
                aggregate_streams = aggregate_streams.rolling(window=7).mean()
            plt.plot(pivot_df.index, aggregate_streams, label='Aggregate Streams', linestyle='--', color='blue')  # Adjust linestyle and color as needed

    elif plot_type == 'raw':
        pivot_df = top_artists_df.pivot_table(index=['DaysElapsed'], columns='artistName', values='Counts', fill_value=0)
        plt.figure(figsize=(20, 10))
        for column in pivot_df.columns:
            plt.plot(pivot_df.index, pivot_df[column], label=column)

        # Calculate aggregate streams across top artists if display_agg is True
        if display_agg:
            aggregate_streams = pivot_df.sum(axis=1)
            plt.plot(pivot_df.index, aggregate_streams, label='Aggregate Streams', linestyle='--', color='blue')  # Adjust linestyle and color as needed

    plt.xlabel('Month')
    plt.ylabel('Total Streams')
    plt.title('Top {} Most Streamed Artists Daily ({})'.format(n_artists, plot_type.capitalize()))

    # Move the legend outside the plot area
    plt.legend(loc='upper left', bbox_to_anchor=(1, 1))

    # Add month labels
    first_day_of_month = pd.date_range(start='2022-06-01', end='2024-01-01', freq='MS')
    plt.xticks([(month_start - pd.to_datetime('2022-06-24')).days for month_start in first_day_of_month], [month_start.strftime('%m/%y') for month_start in first_day_of_month], rotation=45)

    plt.grid(True)
    plt.tight_layout()
    plt.show()

# %%
def plot_top_songs_smooth(df, period, n_songs, plot_type, display_agg=True, threshold=None):
    # Convert 'endTime' to datetime format
    df['endTime'] = pd.to_datetime(df['endTime'])

    if period == 'Day':
        df['DaysElapsed'] = (df['endTime'] - df['endTime'].min()).
```

```python
dt.days
        grouped_df = df.groupby(['DaysElapsed', 'trackName']).size().reset_index(name='Streams')
    elif period == 'Week':
        df['Week'] = df['endTime'].dt.isocalendar().week
        df['YearWeek'] = df['endTime'].dt.strftime('%Y-%W')
        grouped_df = df.groupby(['Week', 'trackName']).size().reset_index(name='Streams')
    elif period == 'Month':
        df['Month'] = df['endTime'].dt.to_period('M')
        grouped_df = df.groupby(['Month', 'trackName']).size().reset_index(name='Streams')

    top_songs = grouped_df.groupby('trackName')['Streams'].sum().nlargest(n_songs).index
    top_songs_df = grouped_df[grouped_df['trackName'].isin(top_songs)]

    if plot_type == 'smooth':
        pivot_df = top_songs_df.pivot_table(index=['DaysElapsed'], columns='trackName', values='Streams', fill_value=0)
        plt.figure(figsize=(20, 10))
        for column in pivot_df.columns:
            pivot_df[column] = pivot_df[column].rolling(window=7).mean()
            plt.plot(pivot_df.index, pivot_df[column], label=column)

        # Calculate the 99th percentile of song occurrences per day after applying rolling mean
        all_values = np.nan_to_num(pivot_df.values.flatten())
        percentile_99 = np.percentile(all_values, 99)
        plt.axhline(y=percentile_99, color='r', linestyle=':', label='99th Percentile (Smooth)')

        # Add label for the line outside the plot area
        if np.isfinite(pivot_df.index[-1]) and np.isfinite(percentile_99):
            plt.text(pivot_df.index[-1] + 5, percentile_99, f'99th Percentile: {percentile_99:.2f}', ha='left', va='center', backgroundcolor='w')

    elif plot_type == 'raw':
        pivot_df = top_songs_df.pivot_table(index=['DaysElapsed'], columns='trackName', values='Streams', fill_value=0)
        plt.figure(figsize=(20, 10))
        for column in pivot_df.columns:
            plt.plot(pivot_df.index, pivot_df[column], label=column)

        # Calculate the 99th percentile of song occurrences per day
        all_values = np.nan_to_num(top_songs_df.groupby('DaysElapsed')['Streams'].sum().values)
        percentile_99 = np.percentile(all_values, 99)
        plt.axhline(y=percentile_99, color='r', linestyle=':', label='99th Percentile (Raw)')

```

```
        # Add label for the line outside the plot area
        if np.isfinite(pivot_df.index[-1]) and np.isfinite(percentile_99):
            plt.text(pivot_df.index[-1] + 5, percentile_99, f'99th Percentile: {percentile_99:.2f}', ha='left', va='center', backgroundcolor='w')

    plt.xlabel('Month')
    plt.ylabel('Total Streams')
    plt.title('Top {} Most Streamed Songs Daily ({})'.format(n_songs, plot_type.capitalize()))

    # Move the legend outside the plot area
    plt.legend(loc='upper left', bbox_to_anchor=(1, 1))

    # Add month labels
    first_day_of_month = pd.date_range(start='2022-06-01', end='2024-01-01', freq='MS')
    plt.xticks([(month_start - pd.to_datetime('2022-06-24')).days for month_start in first_day_of_month], [month_start.strftime('%m/%y') for month_start in first_day_of_month], rotation=45)

    plt.grid(True)

    # Draw a horizontal line at the specified threshold if provided
    if threshold is not None:
        plt.axhline(y=threshold, color='g', linestyle='--', label='Threshold')

    plt.tight_layout()
    plt.show()

# %%
plot_top_artists_smooth(no_unknown, 'Day', 10, 'smooth', False)

# %%
plot_top_artists_smooth(no_unknown, 'Day', 10, 'smooth', True)

# %%
plot_top_songs_smooth(no_unknown, 'Day', 15, 'smooth', True)

# %%
plot_top_songs_smooth(no_unknown, 'Day', 15, 'smooth', True)

# %%
plot_top_songs_smooth(no_unknown, 'Day', 15, 'raw', False)

# %%
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import IsolationForest
# Convert 'endTime' to datetime format
no_unknown['endTime'] = pd.to_datetime(no_unknown['endTime'])
# Encode 'artistName' using Label Encoding
label_encoder = LabelEncoder()
no_unknown['artistCode'] = label_encoder.fit_transform(no_unknown['artistName'])
```

```python
# Calculate the daily counts of all songs and "The 1975" songs
daily_counts = no_unknown.groupby(no_unknown['endTime'].dt.date)['trackName'].count()
the_1975_songs = no_unknown[no_unknown['artistName'] == 'The 1975']
the_1975_daily_counts = the_1975_songs.groupby(the_1975_songs['endTime'].dt.date)['trackName'].count()
# Calculate the proportion of songs by "The 1975" each day
proportion_by_day = the_1975_daily_counts / daily_counts
# Initialize Isolation Forest model
model = IsolationForest(contamination=0.05)  # Adjust contamination based on expected anomaly rate
# Fit the model to your data (using the raw proportion)
proportion_by_day = proportion_by_day.fillna(0)  # Replace NaN values with 0 for the initial period
model.fit(proportion_by_day.values.reshape(-1, 1))
# Predict anomalies (-1 indicates anomalies, 1 indicates normal data points)
anomaly_predictions = model.predict(proportion_by_day.values.reshape(-1, 1))
# Filter the dataframe to keep only anomalies
anomalies = proportion_by_day[anomaly_predictions == -1]
# Visualize anomalies with raw proportion of songs by "The 1975" as y-axis
plt.figure(figsize=(10, 6))
plt.plot(proportion_by_day.index, proportion_by_day.values, label='Proportion of Songs by The 1975')
plt.scatter(anomalies.index, anomalies.values, color='red', label='Anomalies')
plt.xlabel('Date')
plt.ylabel('Proportion of Songs by The 1975')
plt.title('Proportion of Songs by The 1975 Over Time')
plt.legend()
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.tight_layout()
plt.show()
# Print or further analyze the anomalies dataframe
print("Number of anomalies detected:", len(anomalies))
from sklearn.preprocessing import StandardScaler
# Calculate daily counts of all songs and "The 1975" songs
daily_counts = no_unknown.groupby(no_unknown['endTime'].dt.date)['trackName'].count()
the_1975_songs = no_unknown[no_unknown['artistName'] == 'The 1975']
the_1975_daily_counts = the_1975_songs.groupby(the_1975_songs['endTime'].dt.date)['trackName'].count()
# Calculate the proportion of songs by "The 1975" for each day
proportion_by_day = the_1975_daily_counts / daily_counts
# Calculate the 7-day moving average of the proportion
moving_average = proportion_by_day.rolling(window=7).mean()
# Calculate z-scores for each day's proportion
scaler = StandardScaler()
z_scores = scaler.fit_transform(moving_average.values.reshape(-1, 1))
# Set thresholds for abnormality detection (e.g., z-score below -1.5 and above 1.5)
lower_threshold = -1.5
upper_threshold = 1.5
```

```
# Find abnormal days based on thresholds
abnormal_low_days = moving_average[z_scores.flatten() <
    lower_threshold]
abnormal_high_days = moving_average[z_scores.flatten() >
    upper_threshold]
plt.figure(figsize=(10, 6))
plt.plot(moving_average.index, moving_average.values, label='7-Day
    Moving Average of Proportion of Songs by The 1975')
plt.scatter(abnormal_low_days.index, abnormal_low_days.values,
    color='red', label='Abnormal Low Days')
plt.scatter(abnormal_high_days.index, abnormal_high_days.values,
    color='red', label='Abnormal High Days')
plt.xlabel('Date')
plt.ylabel('Proportion of Songs by The 1975')
plt.title('Abnormal Days with Low and High Proportion of Songs by
    The 1975 (7-Day Moving Average)')
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
from statsmodels.tsa.seasonal import seasonal_decompose
no_unknown['endTime'] = pd.to_datetime(no_unknown['endTime'])
# Filter rows where 'artistName' is 'The 1975'
the_1975_data = no_unknown[no_unknown['artistName'] == 'The 1975']
# Group by date and count occurrences, setting the frequency to 'D'
     (daily)
the_1975_daily_counts = the_1975_data.groupby(pd.Grouper(key='
    endTime', freq='D')).size()
# Perform seasonal decomposition
decomposition = seasonal_decompose(the_1975_daily_counts, model='
    additive')
# Plotting the decomposition with a wider figure size
plt.figure(figsize=(14, 8))  # Adjust the width and height as neede
# Original time series
plt.subplot(4, 1, 1)
plt.plot(the_1975_daily_counts, label='Original')
plt.legend()
plt.ylabel('The 1975 Listens')
# Trend component
plt.subplot(4, 1, 2)
plt.plot(decomposition.trend, label='Trend', color='orange')
plt.legend()
plt.ylabel('Trend')
# Seasonal component
plt.subplot(4, 1, 3)
plt.plot(decomposition.seasonal, label='Seasonal', color='green')
plt.legend()
plt.ylabel('Seasonal')
# Residual component
plt.subplot(4, 1, 4)
plt.plot(decomposition.resid, label='Residual', color='red')
plt.legend()
plt.ylabel('Residual')
plt.xlabel('Date')
plt.tight_layout()
plt.show()
no_unknown['endTime'] = pd.to_datetime(no_unknown['endTime'])
```

```python
# Encode 'artistName' using Label Encoding
label_encoder = LabelEncoder()
no_unknown['artistCode'] = label_encoder.fit_transform(no_unknown['artistName'])
# Calculate the daily counts of all songs and "The 1975" songs
daily_counts = no_unknown.groupby(no_unknown['endTime'].dt.date)['trackName'].count()
the_1975_songs = no_unknown[no_unknown['artistName'] == 'The 1975']
the_1975_daily_counts = the_1975_songs.groupby(the_1975_songs['endTime'].dt.date)['trackName'].count()
# Calculate the proportion of songs by "The 1975" each day
proportion_by_day = the_1975_daily_counts / daily_counts
# Smooth the proportion using a 3-day moving average
smoothed_proportion = proportion_by_day.rolling(window=5).mean()
# Initialize Isolation Forest model
model = IsolationForest(contamination=0.05)  # Adjust contamination based on expected anomaly rate
# Fit the model to your data (using the smoothed proportion)
smoothed_proportion = smoothed_proportion.fillna(0)  # Replace NaN values with 0 for the initial period
model.fit(smoothed_proportion.values.reshape(-1, 1))
# Predict anomalies (-1 indicates anomalies, 1 indicates normal data points)
anomaly_predictions = model.predict(smoothed_proportion.values.reshape(-1, 1))
# Filter the dataframe to keep only anomalies
anomalies = smoothed_proportion[anomaly_predictions == -1]
# Visualize anomalies with smoothed proportion of songs by "The 1975" as y-axis
plt.figure(figsize=(10, 6))
plt.plot(smoothed_proportion.index, smoothed_proportion.values, label='Smoothed Proportion of Songs by The 1975')
plt.scatter(anomalies.index, anomalies.values, color='red', label='Anomalies')
plt.xlabel('Date')
plt.ylabel('Smoothed Proportion of Songs by The 1975')
plt.title('Smoothed Proportion of Songs by The 1975 Over Time')
plt.legend()
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.tight_layout()
plt.show()
# Print or further analyze the anomalies dataframe
print("Number of anomalies detected:", len(anomalies))
```

482FinalCode.py