Spotify Songs Analysis

Amy Seo
November 2021
Data Analytics & Visualizations Bootcamp Final Project
GitHub Link

Project Topic

Analysis of songs that have charted on the Spotify Top 200s Chart for the United States since January 1, 2021.

- Numerous songs are released every year in multiple countries
- Understanding insights into popularity of songs/artists in **2021**, focusing on **United States**
- Model to predict success of song based on auditory/musical features
    - Can be useful in anticipating what songs will be most popular and influential
    - Can provide insight into what types of songs are appealing to people at a given point in time

Project Questions

- What songs have the most streams?
- What songs have been #1 on the charts?
- Which artists have the most streams?
- Which artists have the most songs on the chart?
- Which artists have the most #1 hits?
- Which songs are most danceable? Most energetic? Most positive-sounding (high valence)? (and so on, for each audio feature)
- Can we predict if a song can break into the top 20 positions, based on the song's audio features?

Data Source

Spotify Top 200 Charts

- Top 200 Charts, United States
- Daily Data, from Jan. 1, 2021 to Nov. 17, 2021
- Source: Spotify Charts Site
- Scraped Data CSV file

Spotify Audio Features Data

- Spotify API Request for Tracks' audio features data
- Source: API Request Info Site
- Saved Audio Features CSV file

Data Extraction Process & Tools

1. Charts data was scraped using:
   a. splinter
   b. BeautifulSoup
   c. ChromeDriverManager

```python
# Define function to run scraping process
def scrape(date_urls):

    data = []

    # Initiate headless driver for deployment
    executable_path = {'executable_path': ChromeDriverManager().install()}
    browser = Browser('chrome', **executable_path, headless=False)

    # Run scraping function
    for url in date_urls:
        # Visit URL
        browser.visit(url)

        # Set up HTML parser
        html = browser.html
        song_soup = soup(html, 'html.parser')
        playlist = song_soup.find('table', class_='chart-table')

        for i in playlist.find('tbody').findAll('tr'):
            song = i.find('td', class_='chart-table-track').find('strong').get_text()

            artist = i.find('td', class_='chart-table-track').find('span').get_text()
            artist = artist.replace("by ", "").strip()

            songurl = i.find('td', class_='chart-table-image').find('a').get('href')
            songid = songurl.split("track/")[1]

            position = i.find('td', class_='chart-table-position').get_text()

            streams = i.find('td', class_='chart-table-streams').get_text()

            date = url.split("daily/")[1]

            data.append([songid, songurl, song, artist, date, position, streams])

    # Stop webdriver
    browser.quit()
```

2. Features data pulled using an API request:
   a. requests

Spotify API

In [7]:
```python
# Import access tokens for Spotify API
from config import client_id, client_secret
```

In [8]:
```python
# Pass the access token
auth_url = 'https://accounts.spotify.com/api/token'
auth_response = requests.post(auth_url, {
    'grant_type': 'client_credentials',
    'client_id': client_id,
    'client_secret': client_secret
})
auth_response_data = auth_response.json()
access_token = auth_response_data['access_token']
```

In [9]:
```python
# Set up access token in header for GET request
headers = {'Authorization': 'Bearer {token}'.format(token=access_token)}
```

In [10]:
```python
# Set up API request
base_url = 'https://api.spotify.com/v1/audio-features/'
```

In [11]:
```python
# GET request for song_id string lists
data = []
for i in song_ids1:
    req = requests.get(base_url + i, headers=headers)
    req = req.json()
    data.append(req)
for i in song_ids2:
    req = requests.get(base_url + i, headers=headers)
    req = req.json()
    data.append(req)
for i in song_ids3:
    req = requests.get(base_url + i, headers=headers)
    req = req.json()
    data.append(req)
for i in song_ids4:
    req = requests.get(base_url + i, headers=headers)
    req = req.json()
    data.append(req)
for i in song_ids5:
    req = requests.get(base_url + i, headers=headers)
    req = req.json()
    data.append(req)
for i in song_ids6:
```

File Links:
"scraping.py" script
Jupyter Notebook for data loading/cleaning/databasing

Data Cleaning Process & Tools

Scraped data table (“scrape_df”) and API data table (“features_df”) both cleaned using Python and Pandas.

- Unnecessary columns removed
- Column values relabeled as needed
- Data types fixed
- Data checked for null values

Additional data tables created for aggregates and merged for final table to be used for analysis:

- Total streams aggregated for each song
- Highest position in charts recorded for each song
- List of unique track names and artists
- The above three listed tables and “features_df” merged for final dataframe for analysis

```sql
SELECT DISTINCT ON (ta.song_id) ta.song_id,
    ta.song,
    ta.artist,
    ts.streams,
    hp.position,
    f.danceability,
    f.energy,
    f.key,
    f.loudness,
    f.mode,
    f.speechiness,
    f.acousticness,
    f.instrumentalness,
    f.liveness,
    f.valence,
    f.tempo,
    f.duration_ms,
    f.time_signature
INTO songs
FROM track_artist AS ta
JOIN total_streams AS ts
ON (ta.song_id = ts.song_id)
JOIN highest_position AS hp
ON (ta.song_id = hp.song_id)
JOIN features AS f
ON (ta.song_id = f.song_id);
```

File Links:
Jupyter Notebook for data loading/cleaning/databasing
SQL script for merging tables

Database

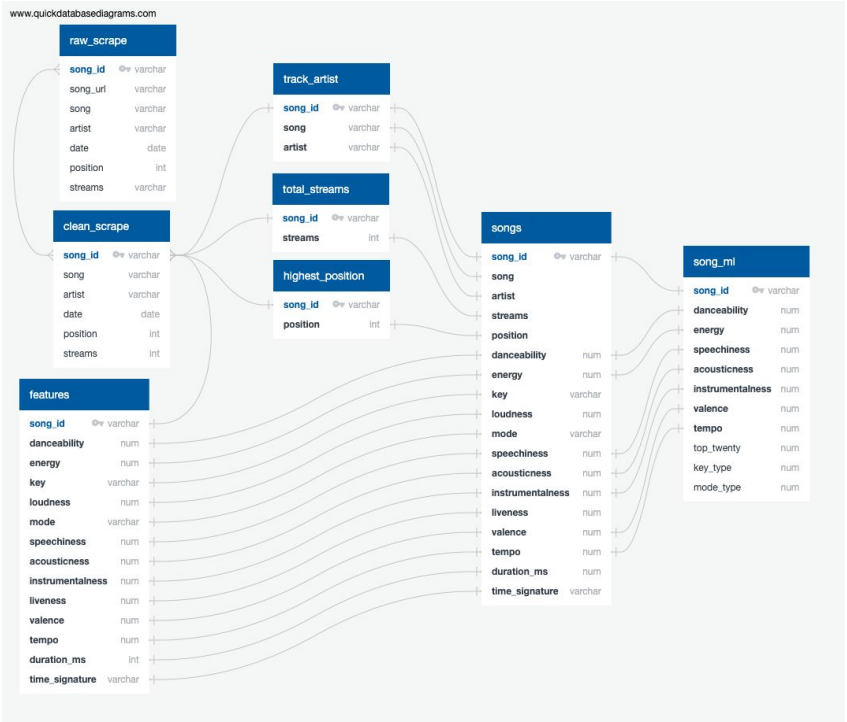
Database Type: **PostgreSQL**

Link: sqlalchemy

Number of Tables: 8

- *Includes raw and cleaned tables*
- *“songs” table = final merged data used for analysis*
- *“song_ml” table = final data used for machine learning inputs*

Primary Key: “song_id”



File Links:
Jupyter Notebook for data loading/cleaning/databasing
SQL script for creating tables

Data Exploration Process & Tools

Tools Used: Python, Pandas, Plotly

Questions Answered

What songs have the most streams?

What songs have been #1 on the charts?

Which artists have the most streams?

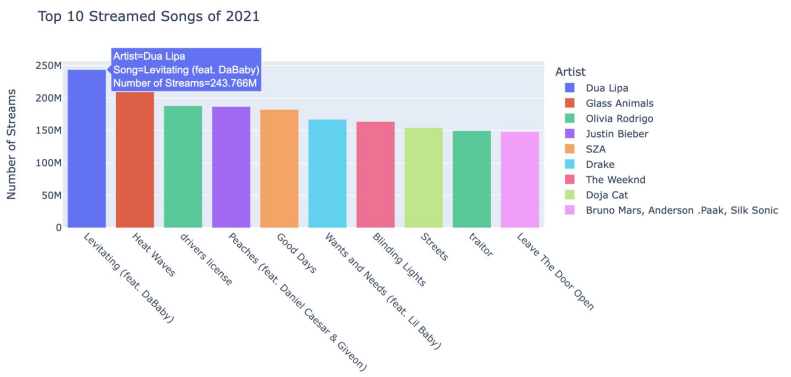
Which artists have the most songs on the chart?

Which artists have the most #1 hits?

Which songs are most danceable? Most energetic? Most positive-sounding (high valence)? (and so on, for each audio feature)

```
In [8]: top_ten_streams = song_streams_df[:10]

# Create visualization
fig = px.bar(
    top_ten_streams,
    x='song',
    y='streams',
    color='artist',
    title='Top 10 Streamed Songs of 2021',
    labels={'song':'Song', 'streams':'Number of Streams', 'artist':'Artist'}
).update_xaxes(categoryorder='total descending', tickangle=45)
# Save an image of chart
fig.write_image("./Visualizations/Top10Streams.png")
# Show chart
fig.show()
```



File Links:
Jupyter Notebook for data exploration
Plotly Chart Visualization Image Files

Data Exploration

Walk-through of Exploration Analysis & Visualizations

(Flask Dashboard)

Flask App/Dashboard Files for Compiled Chart Visualizations:
index.html
app.py

Data Analysis/Model Process & Tools

Machine Learning Model Question

Can we predict if a song can break into the top 20 positions, based on the song's audio features?

Tools Used:

- Pandas → to clean "songs" table for model ("song_ml" table)
- scikit-learn library
- StandardScaler
- LabelEncoder
- SMOTEENN
- RandomForestClassifier

File Links:
Jupyter Notebook for machine learning model
Flask/Dashboard Links: index.html, app.py, run_ml.py, results.html

Data Analysis/Model Process & Tools

1) Additional column (“top_twenty”) added to codify whether a song reached a rank within top 20 ranks or not.

```
In [4]: # Add column for Yes/No encoding of whether song hit top 20 positions
        # Save as new df for machine learning
        song_ml_df = song_df.copy()

        bins = [0, 20, 200]
        group_names = [1, 0]

        song_ml_df['top_twenty'] = pd.cut(song_ml_df['position'], bins, labels=group_names)
        song_ml_df.sample(10)
```

Out[4]:

song_id	song	artist	streams	position	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalnes
1k7Sn04nBNcLKbSwORsFgJ	Session 33	Summer Walker	729869	50	0.618	0.137	G♯/A♭	-12.774	major	0.0609	0.8710	0.0000(
1i6N76fftMZhijOzFQ5ZtL	Psycho Killer - 2005 Remaster	Talking Heads	233001	139	0.721	0.521	D	-8.123	major	0.0613	0.0708	0.0054(
7LzpbYrsoaJFlFDwZbibN9	4th Baby Mama - Prelude	Summer Walker	314716	94	0.453	0.457	B	-10.676	major	0.3480	0.3810	0.0000(
2RsGjBGFCB2LygfIqEEpsz	Aegean	Solkatt	265193	105	0.792	0.256	C	-13.660	minor	0.0727	0.0195	0.9480(
2lePEfMAtqWS6rLXXFZIgI	His & Hers (feat. Don Toliver, Lil Uzi Vert & ...	Internet Money	33133955	33	0.792	0.707	G	-5.218	minor	0.0623	0.1160	0.0000(
03NqHNdG3Ur6kciQWK6e8V	Oh Na Na	Camila Cabello, Myke Towers, Tainy	286760	104	0.744	0.862	C	-3.446	major	0.0622	0.0715	0.0000(
30QR0ndUdiiMQMA9g1PGCm	...And to Those I Love, Thanks for Sticking Ar...	*uicideboy*	30404964	35	0.792	0.511	D	-6.876	major	0.0409	0.1240	0.0000(

2) Unnecessary columns removed from dataframe.

```
In [6]: # Drop the non-beneficial ID columns(song', 'artist') & 'position' (redundant with 'top_twenty' column)
        song_ml_df = song_ml_df.drop(['song', 'artist', 'position'], 1)

        # Drop 'streams', 'duration_ms', 'time_signature', 'loudness', 'liveness' columns
        song_ml_df = song_ml_df.drop(['streams', 'duration_ms', 'time_signature', 'loudness', 'liveness'], 1)

        song_ml_df.head()
```

Out[6]:

song_id	danceability	energy	key	mode	speechiness	acousticness	instrumentalness	valence	tempo	top_twenty
003vvx7Niy0yvhvHt4a68B	0.352	0.911	C♯/D♭	major	0.0747	0.00121	0.000000	0.236	148.033	0
00Blm7zeNqgYLPtW6zg8cj	0.687	0.781	C♯/D♭	major	0.0530	0.03610	0.000000	0.688	97.014	1
00selpxxljfn9n5Pf4K3VR	0.583	0.405	C	major	0.0534	0.64300	0.003910	0.549	84.997	0
01FvQEvHETjWqcDpQDJdTb	0.555	0.771	E	major	0.0282	0.24700	0.000073	0.442	139.971	0
01K4zKU104LyJ8gMb7227B	0.606	0.377	C	major	0.0275	0.81700	0.000000	0.446	101.960	1

File Links:

Jupyter Notebook for machine learning model

Flask/Dashboard Links: index.html, app.py, run_ml.py, results.html

Data Analysis/Model Process & Tools

3) Categorical variables encoded to numerical data.

```
In [7]: # Generate categorical variable list
        song_cat = song_ml_df.dtypes[song_ml_df.dtypes == "object"].index.tolist()

        # Check number of unique values in each column
        song_ml_df[song_cat].nunique()

Out[7]: key     12
        mode     2
        dtype: int64
```

Encode Dataframe's non-numerical data

```
In [8]: # Create instance for label encoder
        labelencoder = LabelEncoder()

In [9]: # Encode categorical columns and store as another column
        song_ml_rf_df = song_ml_df.copy()
        song_ml_rf_df['key_type'] = labelencoder.fit_transform(song_ml_rf_df['key'])
        song_ml_rf_df['mode_type'] = labelencoder.fit_transform(song_ml_rf_df['mode'])

        # Drop the original columns
        song_ml_rf_df = song_ml_rf_df.drop(["key", "mode"], 1)

        song_ml_rf_df.head()
```

Out[9]:

song_id	danceability	energy	speechiness	acousticness	instrumentalness	valence	tempo	top_twenty	key_type	mode_type
003vvx7Niy0yvhvHt4a68B	0.352	0.911	0.0747	0.00121	0.000000	0.236	148.033	0	4	0
00Blm7zeNqgYLPtW6zg8cj	0.687	0.781	0.0530	0.03610	0.000000	0.688	97.014	1	4	0
00selpxxIjfn9n5Pf4K3VR	0.583	0.405	0.0534	0.64300	0.003910	0.549	84.997	0	3	0
01FvQEvHETjWqcDpQDJdTb	0.555	0.771	0.0282	0.24700	0.000073	0.442	139.971	0	7	0
01K4zKU104LyJ8gMb7227B	0.606	0.377	0.0275	0.81700	0.000000	0.446	101.960	1	3	0

4) Target (newly created “top_twenty” column) and features assigned.

Assign features and target variables

```
In [11]: # Assign preprocessed data into features and target arrays
         y = song_ml_rf_df["top_twenty"].ravel()
         X = song_ml_rf_df.drop(["top_twenty"], 1)
```

File Links:

Jupyter Notebook for machine learning model

Flask/Dashboard Links: index.html, app.py, run_ml.py, results.html

Data Analysis/Model Process & Tools

5) Data is split into training and test sets. Then, data is resampled using SMOTEENN technique (to help with class imbalance). Resampled data is then scaled given different scales of features.

Split, resample, and scale data for model

Using SMOTEENN combination sampling to address disparate class sizes

```
In [13]: # Split preprocessed data into training and testing dataset
         X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)

In [14]: # Check dataset split of training set
         Counter(y_train)

Out[14]: Counter({0: 1087, 1: 220})

In [15]: # Data resampled with SMOTEENN
         smote_enn = SMOTEENN(random_state=1)
         X_resampled, y_resampled = smote_enn.fit_resample(X_train, y_train)
         Counter(y_resampled)

Out[15]: Counter({0: 375, 1: 699})

In [16]: # Create StandardScaler instances
         scaler = StandardScaler()

In [17]: # Fit the StandardScaler
         X_scaler = scaler.fit(X_resampled)

In [18]: # Scale data
         X_train_scaled = X_scaler.transform(X_resampled)
         X_test_scaled = X_scaler.transform(X_test)
```

File Links:

Jupyter Notebook for machine learning model

Flask/Dashboard Links: index.html, app.py, run_ml.py, results.html

Data Analysis/Model Process & Tools

6) Random Forest Model is run on the data. Model is trained using 100 estimators.

Random Forest Model

```
In [19]: # Create a random forest classifier
         rf_model = RandomForestClassifier(n_estimators=100, random_state=1)

In [20]: # Fitting the model
         rf_model = rf_model.fit(X_train_scaled, y_resampled)
```

File Links:
Jupyter Notebook for machine learning model
Flask/Dashboard Links: index.html, app.py, run_ml.py, results.html

Data Analysis/Model Results

In [23]:
```
# Training score & Testing score
print(f'Training Data Score: {rf_model.score(X_train_scaled, y_resampled)}')
print(f'Testing Data Score: {rf_model.score(X_test_scaled, y_test)}')
```

```
Training Data Score: 1.0
Testing Data Score: 0.5711009174311926
```

In [24]:
```
# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
cm_df = pd.DataFrame(cm, index=["Actual 0", "Actual 1"], columns = ["Predicted 0", "Predicted 1"])
cm_df
```

Out[24]:

	Predicted 0	Predicted 1
Actual 0	216	142
Actual 1	45	33

In [25]:
```
# Classification report
print(f'Classification Report:\n{classification_report(y_test, y_pred)}')
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.60      0.70       358
           1       0.19      0.42      0.26        78

    accuracy                           0.57       436
   macro avg       0.51      0.51      0.48       436
weighted avg       0.71      0.57      0.62       436
```

File Links:

Jupyter Notebook for machine learning model

Flask/Dashboard Links: index.html, app.py, run_ml.py, results.html

Data Analysis/Model

Walk-through of Interactive Machine Learning Model

(Flask Dashboard)

File Links:
Jupyter Notebook for machine learning model
Flask/Dashboard Links: index.html, app.py, run_ml.py, results.html

Recommendations for Future Analyses

Recommendations

- Testing if audio features' have statistically different values between songs that do reach top 20 vs. don't.
- Testing other new features, such as…
  - Artist popularity/followers
  - Time of release (year/season)

Anything Done Different?

- Conduct in-depth, statistical analyses on each of audio features first to determine usability in model
- Test with wider dataset across multiple years and across different countries

Thank you!