Contents

Chapter 1

Data Cleaning and Pre-Processing

- This file deals with cleaning and pruning the dataset.
- It also sets up the necessary data for further exploratory and predictive analysis.

1.1 Importing libraries

```
In [1]: import pandas as pd
        import numpy as np
```

1.2 Raw Data

- Importing dataset - 'daily_ranking.csv
- The above dataset contains Spotify's Top 100 Worldwide rankings for 50 regions for every day from 2017-2018, along with 13 features of each track.
- Obtain dimesions of dataset, attributes such as columns, number of null values etc. The dataset is pruned in size to make it easier for handling #### The original dataset 'daily_ranking.csv' has NOT been uploaded due to it's large size. It can be found at https://www.kaggle.com/davafons/top-100-spotify-songs-with-audio-feat-20172018

```
In [2]: daily_ranks = pd.read_csv ('daily_ranking.csv') #Load dataset
        print(daily_ranks.shape,'\n', daily_ranks.columns,'\n',daily_ranks.isnull().mean()*100)
        print(daily_ranks.Region.unique())
        daily_ranks.head(5) #peek at first five rows
```

```
(3184489, 22)
 Index(['Position', 'Track_Name', 'Artist', 'Streams', 'URL', 'year', 'month',
       'day', 'Region', 'danceability', 'energy', 'key', 'loudness', 'mode',
       'speechiness', 'acousticness', 'instrumentalness', 'liveness',
       'valence', 'tempo', 'duration_ms', 'time_signature'],
      dtype='object')
 Position            0.000000
Track_Name          0.040101
Artist              0.040101
Streams             0.000000
URL                 0.000000
year                0.000000
month               0.000000
day                 0.000000
Region              0.000000
danceability        0.000000
```

```
energy              0.000000
key                 0.000000
loudness            0.000000
mode                0.000000
speechiness         0.000000
acousticness        0.000000
instrumentalness    0.000000
liveness            0.000000
valence             0.000000
tempo               0.000000
duration_ms         0.000000
time_signature      0.000000
dtype: float64
['ee' 'cr' 'pt' 'gr' 'br' 'es' 'lu' 'ca' 'id' 'lt' 'fr' 'my' 'ie' 'ar'
 'gb' 'nz' 'dk' 'jp' 'nl' 'de' 'cz' 'is' 'hn' 'gt' 'at' 'hu' 'hk' 'bg'
 'bo' 'us' 'au' 'lv' 'ec' 'do' 'no' 'py' 'pe' 'ch' 'cl' 'pl' 'pa' 'be'
 'fi' 'it' 'mt' 'ni' 'co' 'mx' 'ph' 'global']
```

Out[2]:

	Position	Track_Name	Artist	\
0	1	Starboy	The Weeknd	
1	2	Tuesday	Burak Yeter	
2	3	Closer	The Chainsmokers	
3	4	Alone	Alan Walker	
4	5	Rockabye (feat. Sean Paul & Anne-Marie)	Clean Bandit	

	Streams	URL	year	month	\
0	1950	https://open.spotify.com/track/5aAx2yezTd8zXrk…	2017	1	
1	1727	https://open.spotify.com/track/7abpmGpF7PGep2r…	2017	1	
2	1692	https://open.spotify.com/track/7BKLCZ1jbUBVqRi…	2017	1	
3	1649	https://open.spotify.com/track/0JiVRyTJcJnmlwC…	2017	1	
4	1638	https://open.spotify.com/track/5knuzwU65gJK7IF…	2017	1	

	day	Region	danceability	…	loudness	mode	speechiness	acousticness	\
0	1	ee	0.681	…	-7.028	1	0.2820	0.1650	
1	1	ee	0.839	…	-6.084	0	0.0810	0.0159	
2	1	ee	0.748	…	-5.599	1	0.0338	0.4140	
3	1	ee	0.676	…	-3.194	1	0.0458	0.1860	
4	1	ee	0.720	…	-4.068	0	0.0523	0.4060	

	instrumentalness	liveness	valence	tempo	duration_ms	time_signature
0	0.000003	0.1340	0.535	186.054	230453	4
1	0.022800	0.0569	0.640	98.972	241875	4
2	0.000000	0.1110	0.661	95.010	244960	4
3	0.000405	0.1210	0.157	97.019	161200	4
4	0.000000	0.1800	0.742	101.965	251088	4

[5 rows x 22 columns]

1.3 Cleaning Data

- Drop unwanted columns - Streams, URL
- Drop rows with ranks > 50 (reducing size of dataset for easier handling and analysis)
- Drop null values

- Convert ‘year’, ‘month’ and ‘day’ columns to single ‘Date’ column
- Pruned dataset written to csv file ‘cleaned_ranks.csv’ for reference

```
In [3]: daily_ranks.drop(['Streams','URL'], axis = 1, inplace = True)
        daily_ranks.drop(daily_ranks[daily_ranks['Position']>50]. index, axis=0, inplace=True)

        daily_ranks.dropna(subset = ["Track_Name"], inplace=True)
        print((daily_ranks.isnull().mean())*100)

        daily_ranks['Date']=pd.to_datetime(daily_ranks[['year','month','day']])
        daily_ranks.drop(['year','month','day'], axis = 1, inplace = True)

        print(daily_ranks.columns)
        daily_ranks.to_csv('cleaned_ranks.csv',index=False)
```

```
Position            0.0
Track_Name          0.0
Artist              0.0
year                0.0
month               0.0
day                 0.0
Region              0.0
danceability        0.0
energy              0.0
key                 0.0
loudness            0.0
mode                0.0
speechiness         0.0
acousticness        0.0
instrumentalness    0.0
liveness            0.0
valence             0.0
tempo               0.0
duration_ms         0.0
time_signature      0.0
dtype: float64
Index(['Position', 'Track_Name', 'Artist', 'Region', 'danceability', 'energy',
       'key', 'loudness', 'mode', 'speechiness', 'acousticness',
       'instrumentalness', 'liveness', 'valence', 'tempo', 'duration_ms',
       'time_signature', 'Date'],
      dtype='object')
```

1.4 Track List

- Get list of unique tracks in datset, along with Artist.
- Written to csv file ‘track_list’ for later reference.

```
In [4]: tracks=daily_ranks[['Track_Name','Artist']].sort_values(by='Track_Name')
        tracks.drop_duplicates(subset ="Track_Name",keep = 'first', inplace = True)
        tracks.to_csv('track_list.csv',index=False)

        tracks.head(5)

        # print(data.tail(5))
```

Out[4]:

	Track_Name	Artist
578143	#DansLeTierquar (Lyon)	RK
578348	#DansLeTierquar (Marseille)	RK
578030	#DansLeTierquar (Nantes)	RK
1996981	#FleKKsinonem	Kreisais Krasts
1127198	#JM	Broederliefde

1.5 Song Features

- Get list of unique songs and their Features.
- Written to a csv 'song_features.csv'.
- This values in this file are later used for clustering to find similarities between songs.

```
In [5]: song_features=(daily_ranks.drop(['Artist','Region','Position','Date'], axis = 1))
        song_features.drop_duplicates(subset ="Track_Name",keep = 'first', inplace = True)
        song_features.sort_values(by='Track_Name',inplace=True)
        song_features.to_csv('song_features.csv',index=False)
        song_features.head(5)
```

Out[5]:

	Track_Name	danceability	energy	key	loudness	\
578143	#DansLeTierquar (Lyon)	0.755	0.600	6	-6.235	
575349	#DansLeTierquar (Marseille)	0.803	0.798	4	-6.483	
576637	#DansLeTierquar (Nantes)	0.805	0.612	2	-8.139	
1995987	#FleKKsinonem	0.774	0.814	2	-4.963	
1126999	#JM	0.841	0.638	2	-7.616	

	mode	speechiness	acousticness	instrumentalness	liveness	valence	\
578143	0	0.170	0.25800	0.000000	0.0848	0.370	
575349	0	0.388	0.42800	0.000000	0.1080	0.188	
576637	1	0.442	0.25800	0.000000	0.1160	0.303	
1995987	1	0.212	0.04630	0.000000	0.1400	0.499	
1126999	1	0.169	0.00612	0.000003	0.0772	0.316	

	tempo	duration_ms	time_signature
578143	160.023	101544	4
575349	135.942	115334	4
576637	155.036	102983	4
1995987	146.001	346118	4
1126999	100.021	211625	4

1.6 Borda Counts

- In order to find the relative ordering between tracks (overall, and also within region) in terms of popularity, the borda count system of voting was used.
- Borda Count - If an item has n items below it in the ranking, it is given a score of n.

```
In [6]: borda_counts=daily_ranks[['Track_Name','Position','Region','Date']]
        borda_counts['borda_count'] = borda_counts.apply(lambda row: 51-row.Position, axis = 1)
        borda_counts.to_csv('borda_counts.csv',index=False)
        borda_counts.head(10)
```

Out[6]:

	Track_Name	Position	Region	\
0	Starboy	1	ee	
1	Tuesday	2	ee	

```
2                                          Closer         3     ee
3                                           Alone         4     ee
4             Rockabye (feat. Sean Paul & Anne-Marie)         5     ee
5  I Don’t Wanna Live Forever (Fifty Shades Darke…         6     ee
6                                    Black Beatles         7     ee
7                              Say You Won't Let Go         8     ee
8                                  I Feel It Coming         9     ee
9                                   Let Me Love You        10     ee
```

```
        Date  borda_count
0 2017-01-01           50
1 2017-01-01           49
2 2017-01-01           48
3 2017-01-01           47
4 2017-01-01           46
5 2017-01-01           45
6 2017-01-01           44
7 2017-01-01           43
8 2017-01-01           42
9 2017-01-01           41
```

Chapter 2

Visualization

- This file deals with some exploratory data analysis and visualization of trends, correlations, etc.

2.1 Importing libraries

```
In [1]: import pandas as pd
        import numpy as np
        import seaborn as sns
        import matplotlib.pyplot as plt
```

2.2 Importing Data

- The cleaned and pruned dataset is imported.

```
In [2]: data=pd.read_csv('cleaned_ranks.csv')
        data.head(5)
```

Out[2]:

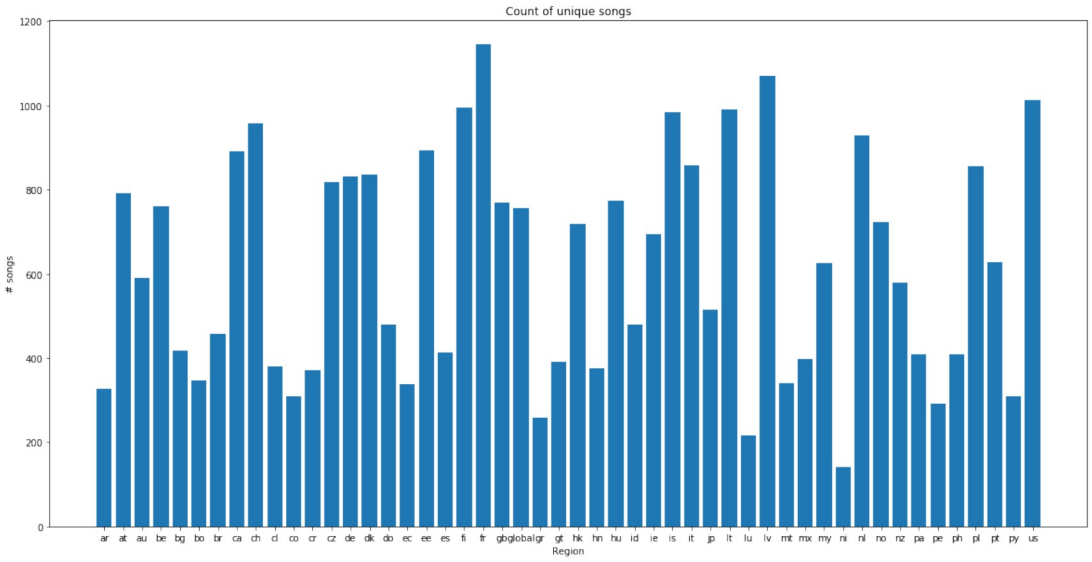
	Position	Track_Name	Artist	Region	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_ms	time_signature	Date
0	1	Starboy	The Weeknd	ee	0.681	0.594	7	-7.028	1	0.2820	0.1650	0.000003	0.1340	0.535	186.054	230453	4	2017-01-01
1	2	Tuesday	Burak Yeter	ee	0.839	0.645	9	-6.084	0	0.0810	0.0159	0.022800	0.0569	0.640	98.972	241875	4	
2	3	Closer	The Chainsmokers	ee	0.748	0.524	8	-5.599	1	0.0338	0.4140	0.000000	0.1110	0.661	95.010	244960	4	
3	4	Alone	Alan Walker	ee	0.676	0.929	10	-3.194	1	0.0458	0.1860	0.000405	0.1210	0.157	97.019	161200	4	
4	5	Rockabye (feat. Sean Paul & Anne-Marie)	Clean Bandit	ee	0.720	0.763	9	-4.068	0	0.0523	0.4060	0.000000	0.1800	0.742	101.965	251088	4	

```
1  2017-01-01
2  2017-01-01
3  2017-01-01
4  2017-01-01
```

2.3 Regional distribution

- Find number of unique songs per region.
- On an average most regions seem to have atleast 500 different songs in their rankings, with very few crossing the 1000 mark.

```
In [3]: region_songs=(data.groupby(['Region'])['Track_Name'].unique())
        count=[]
        for region in region_songs:
            count.append(len(region))
        fig, ax = plt.subplots(figsize=(20, 10))
        points = region_songs.index
        ax.bar(points, count)
        ax.set_title('Count of unique songs')
        ax.set_xlabel('Region')
        ax.set_ylabel('# songs')
        plt.show()
```

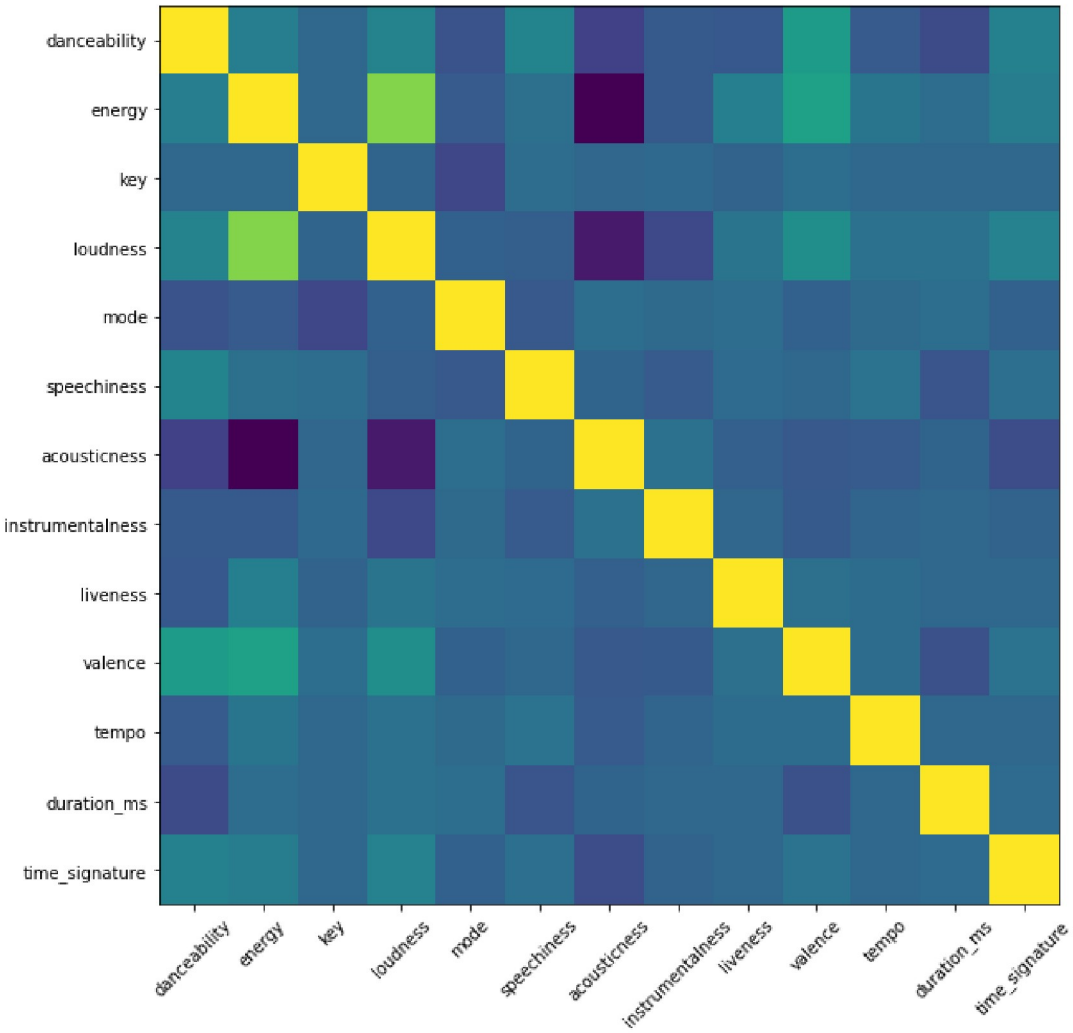


2.4 Song Features

- The file 'song_features.csv' contains a list of all the unique songs which appeared in the global rankings along with the features of each song.
- Analyze the relationship between different features of each songs.
- A heat map, and pairplots are plotted depicting the correlation between the features.

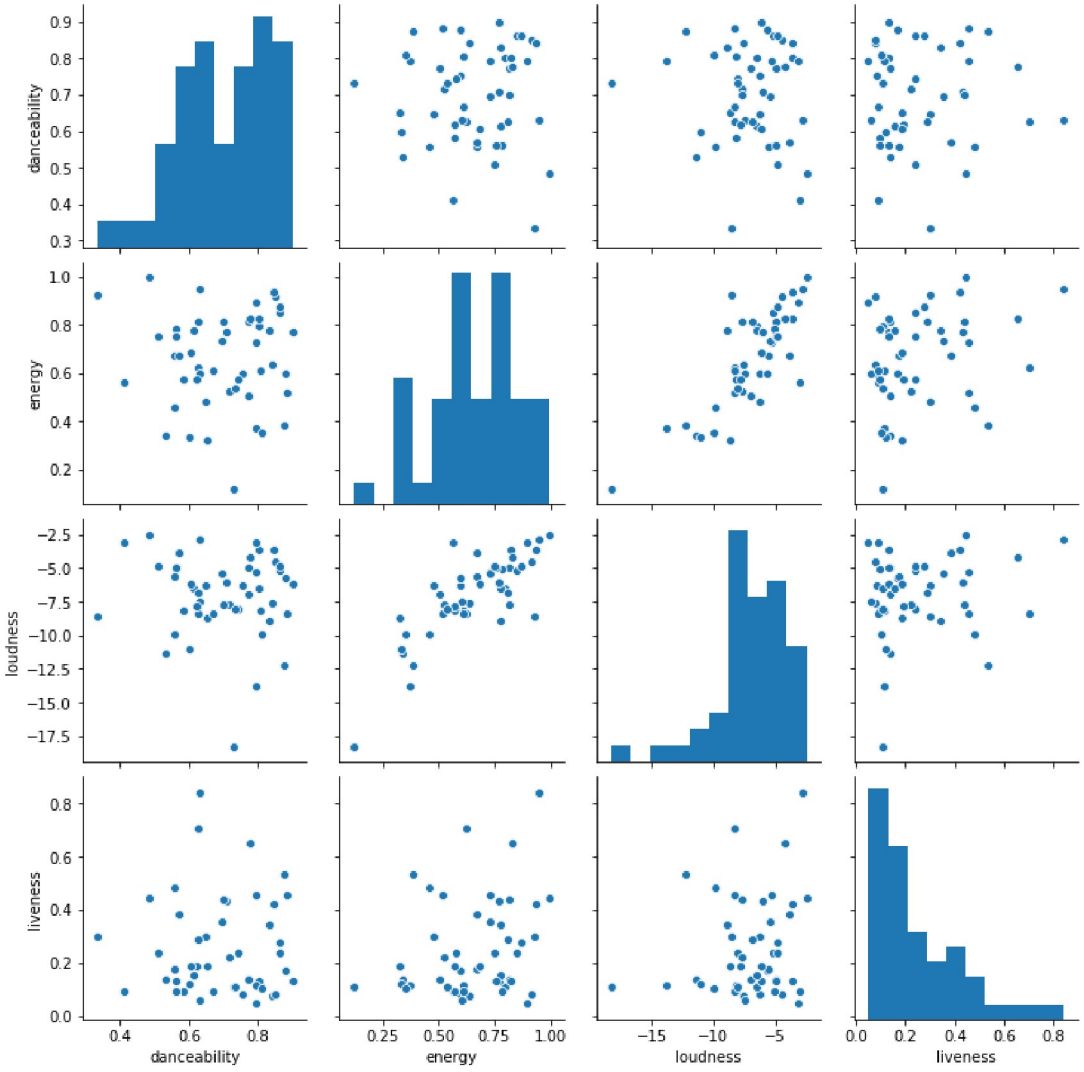
```
In [4]: song_data=pd.read_csv('song_features.csv')
        # print(song_data.columns)
```

```
song_data.set_index('Track_Name',inplace=True)

# get correlation matrix
corr = song_data.corr()

fig, ax = plt.subplots(figsize=(20, 10))
# create heatmap
im = ax.imshow(corr.values)

# set labels
ax.set_xticks(np.arange(len(corr.columns)))
ax.set_yticks(np.arange(len(corr.columns)))
ax.set_xticklabels(corr.columns)
ax.set_yticklabels(corr.columns)
plt.xticks(rotation=45)
plt.show()
```



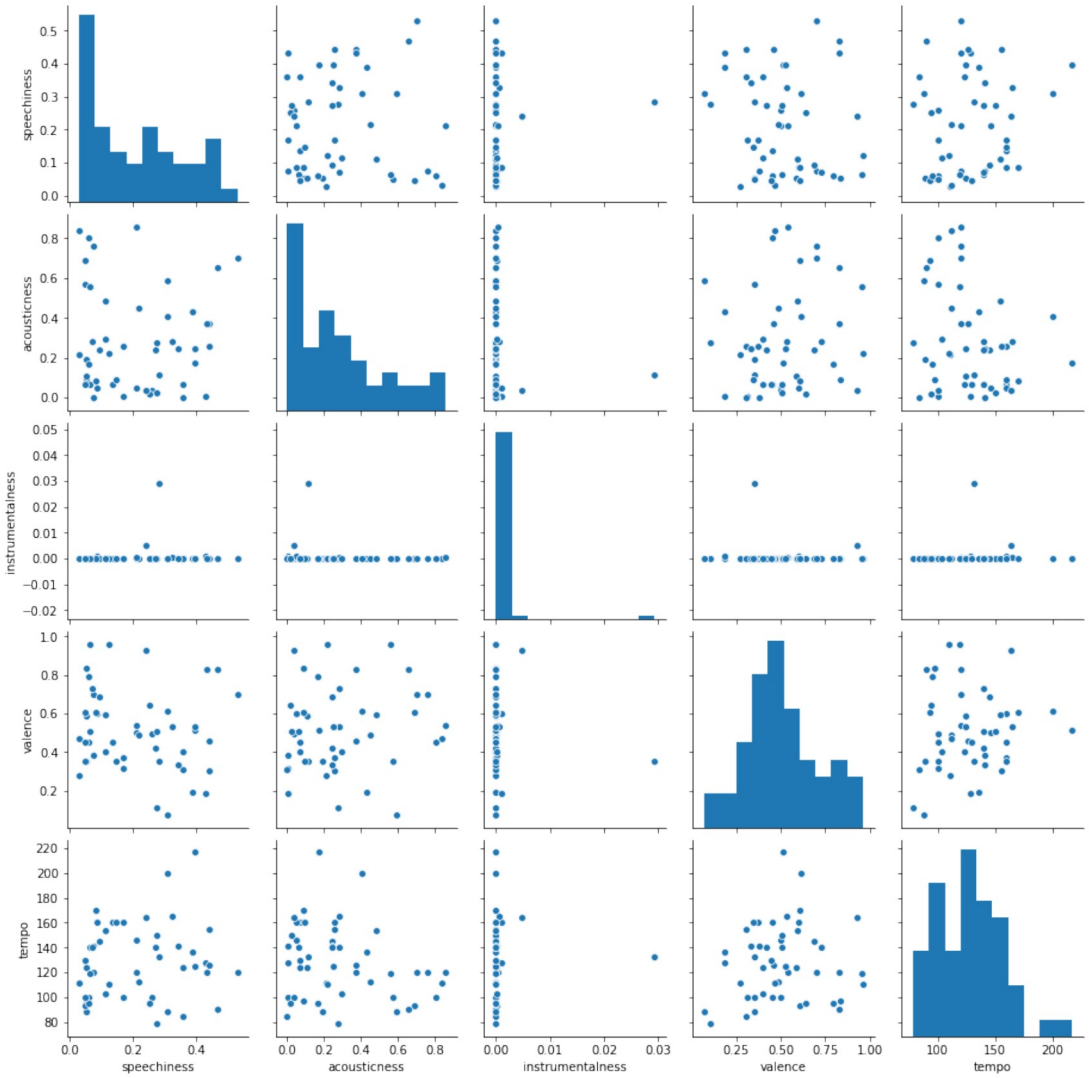
```
In [5]: plot_data=song_data[['danceability','energy','loudness','liveness']].head(50)
        # plot_data.head(10)
        sns.pairplot(plot_data)
        # print(song_data.columns)
```

```
Out[5]: <seaborn.axisgrid.PairGrid at 0x7f5a54435fd0>
```




```
In [6]: plot_data=song_data[['speechiness',        'acousticness',        'instrumentalness', 'valence'
        # plot_data.head(10)
        sns.pairplot(plot_data)
        # print(song_data.columns)
```

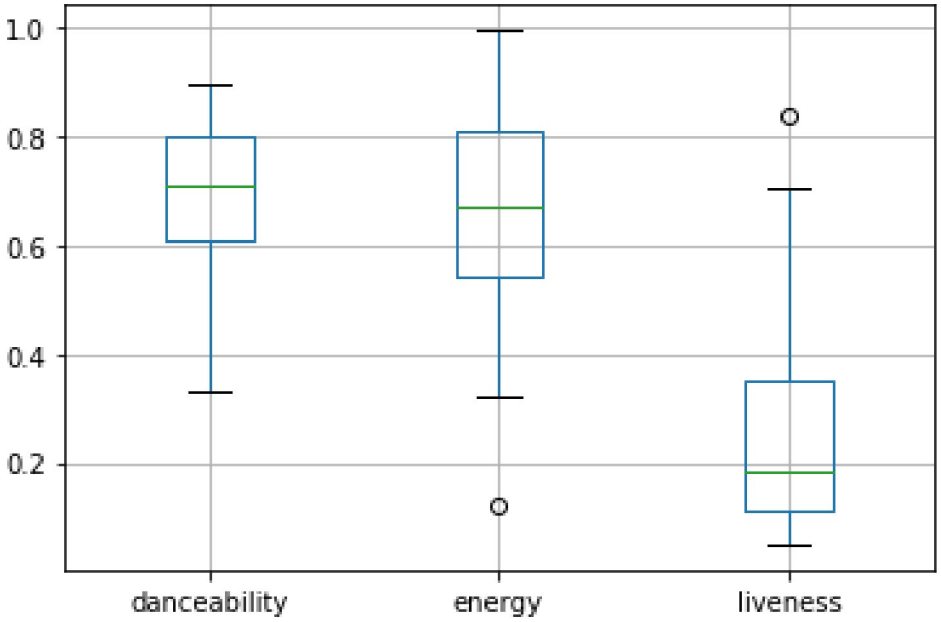
```
Out[6]: <seaborn.axisgrid.PairGrid at 0x7f5a524c9e10>
```



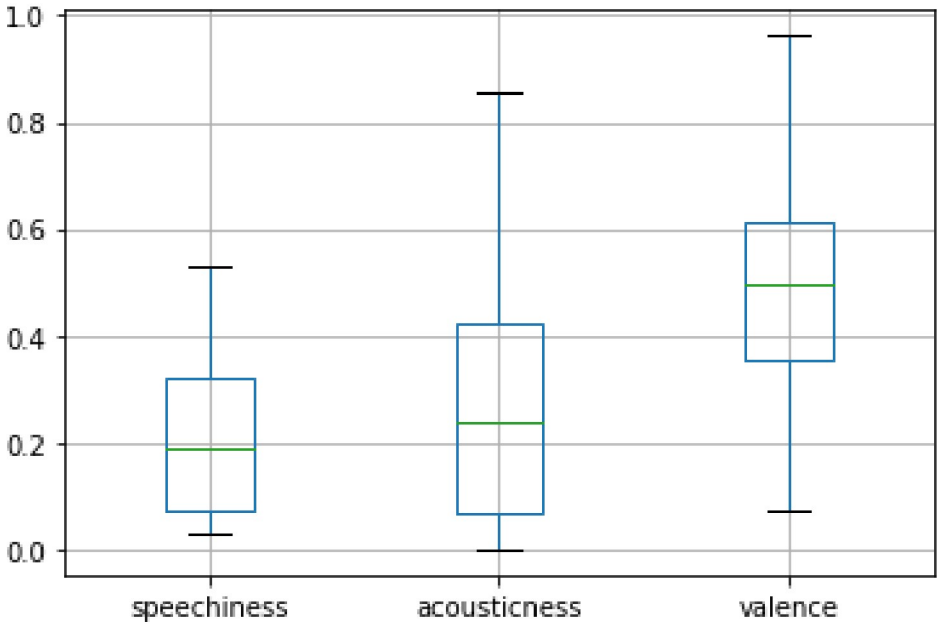
2.5 Box Plots

- Box plots are plotted for the features of each song to understand their distributions.

```
In [7]: plot_data=song_data[['danceability','energy','loudness','liveness','tempo','speechiness','acous
        boxplot = plot_data.boxplot(column=['danceability','energy','liveness'])
```



In [8]:
```
# plot_data=song_data[['speechiness',        'acousticness',        'instrumentalness', 'valenc
boxplot = plot_data.boxplot(column=['speechiness',        'acousticness', 'valence'])
```

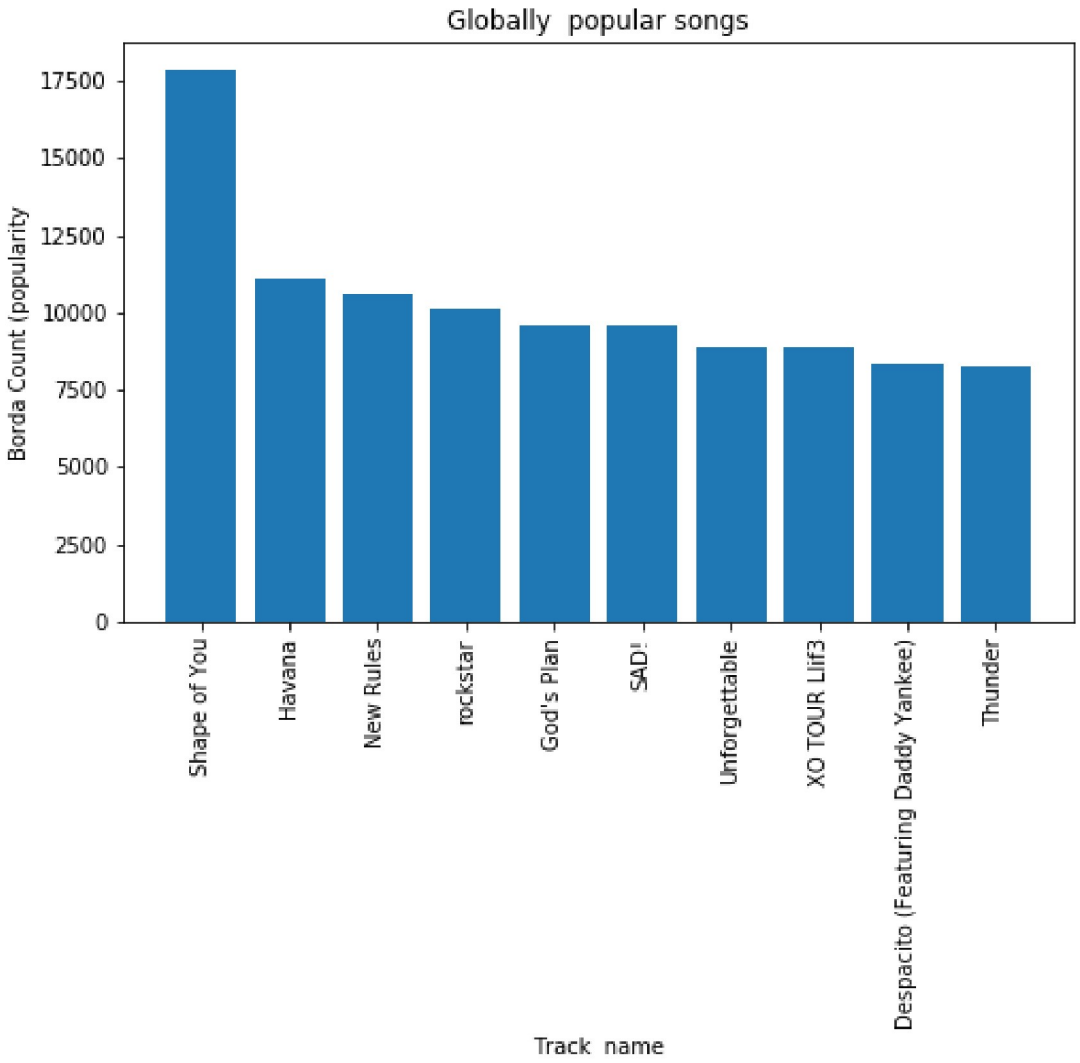


2.6 Globally popular

- The 'borda_counts.csv' file contains data about the borda_counts (popularity score) of each song for each day in each region.
- From this information about the 'global' rankings is obtained, and plotted to find the most popular songs from 2017-2018 globally.

```
In [9]: borda_data=pd.read_csv('borda_counts.csv')
        global_data=pd.DataFrame(borda_data.loc[borda_data['Region']=='global'])

        global_data.head(10)

        global_data_sum=pd.DataFrame(global_data.groupby(['Track_Name'])['borda_count'].sum())
        global_data_sum.sort_values(by=['borda_count'],ascending=False,inplace=True)
```

```
In [10]: temp=global_data_sum.head(10)

         global_popular=pd.DataFrame(columns=['Track_Name','Borda_Count'])
         global_popular['Track_Name']=temp.index
         global_popular['Borda_Count']=temp.values


         fig, ax = plt.subplots(figsize=(8, 5))
         points = global_popular['Track_Name']
         count = global_popular['Borda_Count']
         ax.bar(points, count)
         ax.set_title('Globally  popular songs')
         ax.set_xlabel('Track  name')
         ax.set_ylabel('Borda Count (popularity')
         plt.xticks(rotation=90)
         plt.show()
```

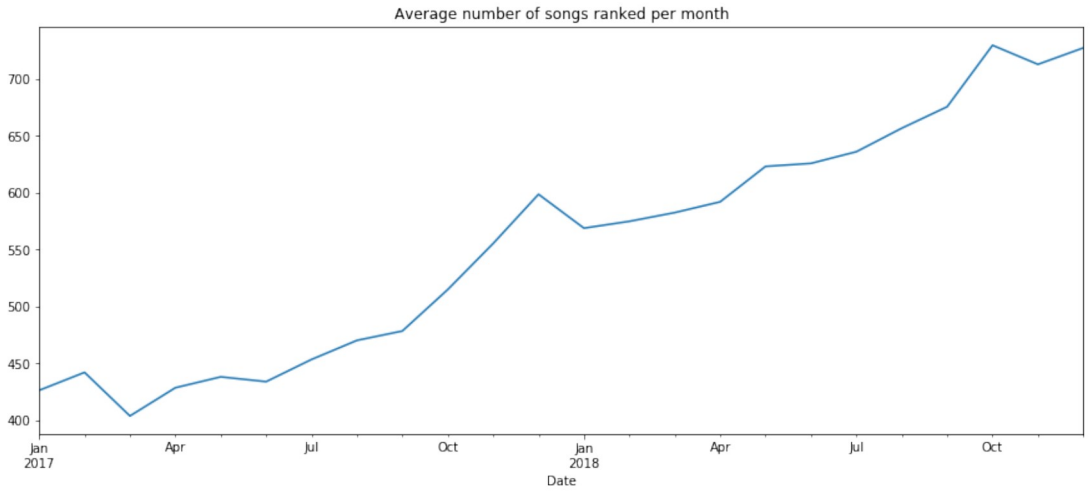


2.7 Song Data - Average number of songs ranked

- The total number of unique songs that find a place in the rankings globally is obtained.
- This is plotted to find the trend in average number of songs that get ranked every month.

```
In [11]: datewise_data=(data.groupby(['Date'])['Track_Name'].unique())
         count=[]
         for date in datewise_data:
             count.append(len(date))

         songs_per_day=pd.DataFrame(columns=['Date','No_of_songs'])
         songs_per_day['Date']=datewise_data.index
         songs_per_day['No_of_songs']=count

         songs_per_day['Date'] = pd.to_datetime(songs_per_day['Date'])
```

```
songs_per_day=songs_per_day.set_index('Date')
# songs_per_day
songs_per_day = songs_per_day['No_of_songs'].resample('MS').mean().fillna(0)
songs_per_day.plot(figsize=(15, 6))

plt.title('Average number of songs ranked per month')
plt.show()

# print(len(region_songs[1]))
```



Chapter 3

Clustering

- This file focuses on clustering the songs on the basis of their features to identify similar tracks.
- The similarity between tracks is later used to recommend songs on the basis of what is already in the list.

3.1 Importing libraries

```
In [1]: import numpy as np
        import pandas as pd
        from sklearn.decomposition import PCA
        from sklearn.preprocessing import StandardScaler
        from sklearn.cluster import KMeans
        import matplotlib.pyplot as plt
        from sklearn.metrics import silhouette_score
```

3.2 Song Features

- The file 'song_features.csv' contains a list of all the unique songs which appeared in the global rankings along with the features of each song.

```
In [2]: song_features=pd.read_csv('song_features.csv')
        print(song_features.shape)
        song_features.head(10)
```

```
(8524, 14)
```

Out[2]:

	Track_Name	danceability	energy	key	loudness	mode	\
0	#DansLeTierquar (Lyon)	0.755	0.600	6	-6.235	0	
1	#DansLeTierquar (Marseille)	0.803	0.798	4	-6.483	0	
2	#DansLeTierquar (Nantes)	0.805	0.612	2	-8.139	1	
3	#FleKKsinonem	0.774	0.814	2	-4.963	1	
4	#JM	0.841	0.638	2	-7.616	1	
5	#JesuispasséchezSo EP 11	0.850	0.918	10	-4.483	0	
6	#Natural	0.803	0.825	0	-3.594	0	
7	#Rohffback	0.794	0.730	7	-5.290	1	
8	#SNTL	0.695	0.733	11	-5.429	0	
9	#VIKINGCLAP	0.483	0.996	2	-2.489	1	

	speechiness	acousticness	instrumentalness	liveness	valence	tempo	\

```
   0         0.170       0.25800          0.000000    0.0848    0.370  160.023
   1         0.388       0.42800          0.000000    0.1080    0.188  135.942
   2         0.442       0.25800          0.000000    0.1160    0.303  155.036
   3         0.212       0.04630          0.000000    0.1400    0.499  146.001
   4         0.169       0.00612          0.000003    0.0772    0.316  100.021
   5         0.217       0.44900          0.000000    0.0800    0.490  111.982
   6         0.053       0.09030          0.000000    0.1330    0.833   97.015
   7         0.260       0.03330          0.000000    0.4580    0.497   99.924
   8         0.274       0.02360          0.000000    0.3530    0.509  150.053
   9         0.432       0.00441          0.000930    0.4450    0.187  127.970

      duration_ms  time_signature
   0       101544               4
   1       115334               4
   2       102983               4
   3       346118               4
   4       211625               4
   5       160467               4
   6       227013               4
   7       199089               4
   8       209116               4
   9       191947               4
```

3.3 Dimensionality Reduction

- As the number of features of each song is large (13), Principal Component Analysis had been applied to reduce the number of dimensions to 2.

```
In [3]: pca = PCA(n_components=2)

        X = song_features.iloc[:, 1:].values
        sc = StandardScaler()
        X_scaled= sc.fit_transform(X)
        X_pca = pca.fit_transform(X_scaled)

        print(X_pca)
```

```
[[-0.08696403 -1.65634894]
 [-0.26866842 -2.03944821]
 [ 0.27564549 -1.77757463]
 …
 [-1.87647979  0.19559217]
 [-0.8931586   0.6615406 ]
 [-2.27696268  0.05154483]]
```

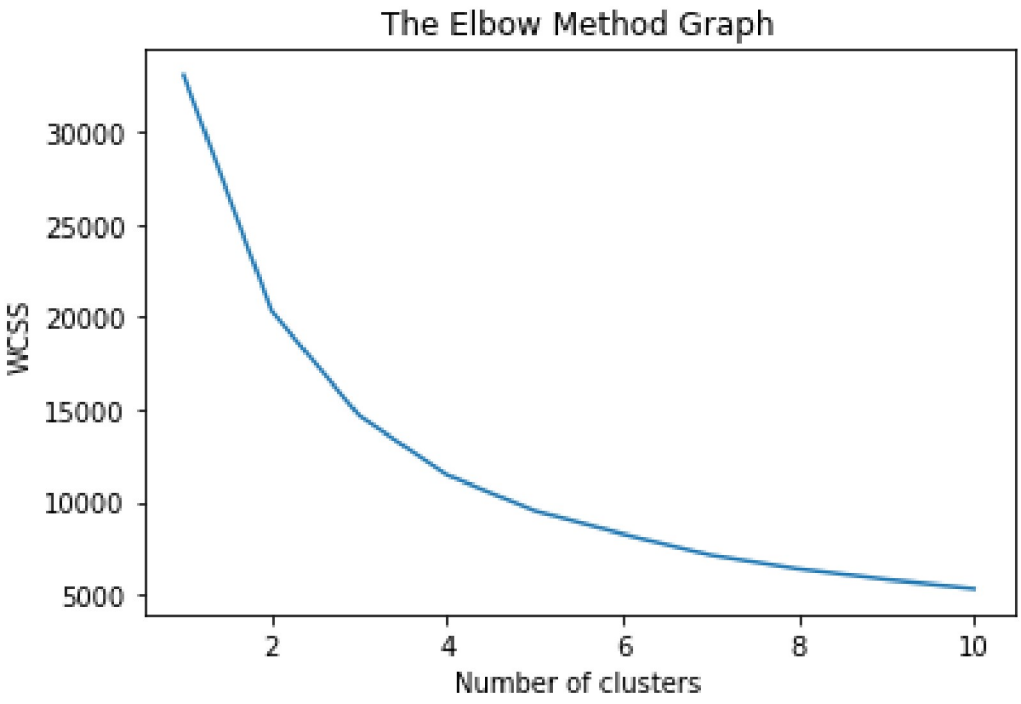
3.4 Clustering

- To find the ideal number of clusters for the data, the following techniques have been implemented:
  - The elbow method
  - Silhouette Score

3.4.1 Elbow Method

- The results of the elbow method show that the ideal number of clusters for the songs is 4

```
In [4]: wcss=[]
        for i in range(1,11):
             kmeans = KMeans(n_clusters=i, init ='k-means++', max_iter=300,  n_init=10,random_state=0 )
             kmeans.fit(X_pca)
             wcss.append(kmeans.inertia_)
        # print(wcss)
        plt.plot(range(1,11),wcss)
        plt.title('The Elbow Method Graph')
        plt.xlabel('Number of clusters')
        plt.ylabel('WCSS')
        plt.show()
```

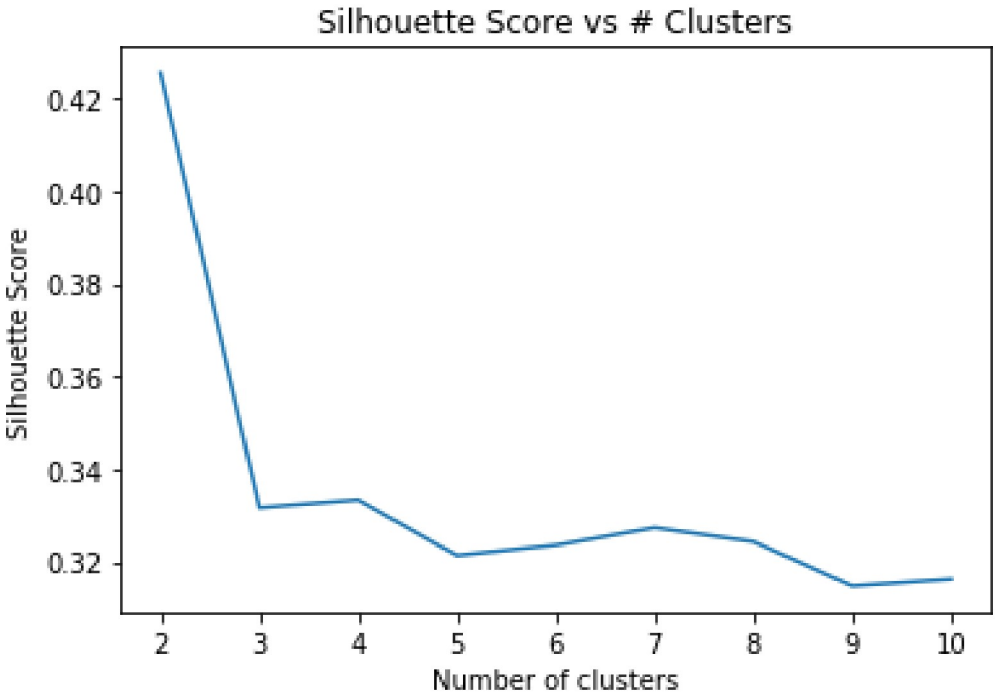


3.4.2 Silhouette Score

- The results obtained through the elbow method, are validated by the Silhouette Score. The larger the solhouette score the better.
- The Silhouette score for '4' is acceptable, and hence the data is clustered into 4 groups.

```
In [5]: silhouette = []
        kmax = 10

        # dissimilarity would not be defined for a single cluster, thus, minimum number of clusters sho
        for k in range(2, kmax+1):
          kmeans = KMeans(n_clusters = k).fit(X_pca)
```

```
    labels = kmeans.labels_
    silhouette.append(silhouette_score(X_pca, labels, metric = 'euclidean'))

# print(wcss)
plt.plot(range(2,11),silhouette)
plt.title('Silhouette Score vs # Clusters')
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette Score')
plt.show()
```



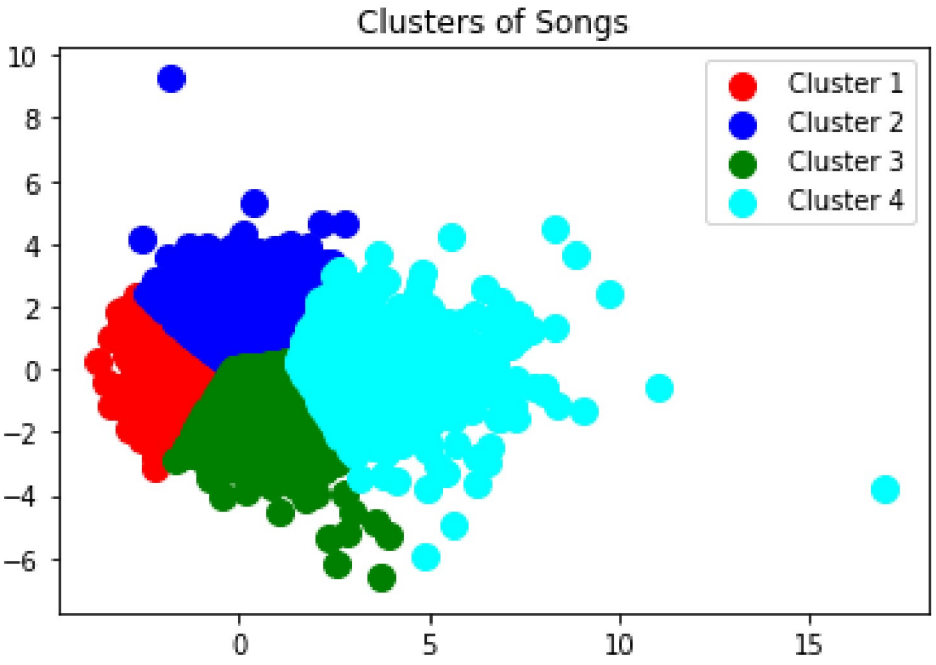
3.4.3 Clustering using K means

In [6]:
```
# fit kmeans object to data
kmeans = KMeans(n_clusters=4)
kmeans.fit(X_pca)

print(kmeans.cluster_centers_)
clustered_values= kmeans.fit_predict(X_pca)

print(clustered_values)
```

```
[[ 3.26954245  0.26036302]
 [-1.29975796 -0.17046193]
 [-0.00348613  1.2428108 ]
 [ 0.37909062 -1.07129648]]
[2 2 2 … 0 0 0]
```

3.4.4 Plotting the clusters

```
In [7]: plt.scatter(X_pca[clustered_values==0, 0], X_pca[clustered_values==0, 1], s=100, c='red', label
        plt.scatter(X_pca[clustered_values==1, 0], X_pca[clustered_values==1, 1], s=100, c='blue', label
        plt.scatter(X_pca[clustered_values==2, 0], X_pca[clustered_values==2, 1], s=100, c='green', lab
        plt.scatter(X_pca[clustered_values==3, 0], X_pca[clustered_values==3, 1], s=100, c='cyan', label
        plt.legend()
        plt.title('Clusters of Songs')
        plt.show()
```



3.4.5 Saving the clusters

- The file 'track_list.csv' contains a list of unique songs that have been found in the ranking globally.
- A column 'Cluster' is added with data about which cluster the song belongs to.

```
In [8]: track_list=pd.read_csv('track_list.csv')

        track_list['Cluster']=clustered_values
        track_list.to_csv('track_list.csv',index=False)
        track_list.head(5)
```

Out[8]:

	Track_Name	Artist	Cluster
0	#DansLeTierquar (Lyon)	RK	2
1	#DansLeTierquar (Marseille)	RK	2
2	#DansLeTierquar (Nantes)	RK	2
3	#FleKKsinonem	Kreisais Krasts	1
4	#JM	Broederliefde	2

Chapter 4

Recommender-System

- This file deals with obtaining recommendations for each region based on the songs already present in it's rankings.
- In real-world applications of recommender systems 9Spotify, Amazon, etc) there are two kinds of recommendations made:
  - Item-based collaborative filtering - Items similar to the ones bought by the user are recommended
  - User-based collaborative filtering - Items bought by users who have a similar purchase paatern to the target user recommended.
- The system below attempts to make recommendations based on both the item as well as the user.
- In this case, each region is considered to be a user and the songs are the items.

4.1 Importing libraries

```
In [1]: import pandas as pd
        import numpy as np
        import random
```

4.2 Importing data

- The 'borda_counts.csv' file contains a list of songs with their borda counts (Popularity scores) along with the region and the dates.

```
In [2]: data= pd.read_csv ('borda_counts.csv') #Load dataset

        print(data.shape)
        (data.columns)
```

```
(1644778, 5)
```

```
Out[2]: Index(['Track_Name', 'Position', 'Region', 'Date', 'borda_count'], dtype='object')
```

- Get list of unique songs grouped by region. .

```
In [3]: regions=data.Region.unique()
        region_songs=(data.groupby(['Region'])['Track_Name'].unique())
```

4.3 Aggregating the values

- The popularity scores for each track are summed up over all the dates to get the total score for each track in a region.
- The higher the score, the more popular the song in the region.
- The aggregated values are sorted to get the most popular songs per region.

```
In [4]: borda_counts_sum=pd.DataFrame(data.groupby(['Region','Track_Name'])['borda_count'].sum())
        borda_counts_sum.sort_values(by=['Region','borda_count'],ascending=False,inplace=True)
        borda_counts_sum.head(10)
```

Out[4]:

Region	Track_Name	borda_count
us	XO TOUR Llif3	14596
	Congratulations	13560
	HUMBLE.	12093
	SAD!	11613
	Lucid Dreams	9942
	rockstar	9822
	God's Plan	9762
	1-800-273-8255	9733
	I Fall Apart	9554
	Bank Account	9054

4.4 Correlation between regions

- The aggregated scores are used to find the Correlation between regions based on the rankings they give to the same/similar tracks.
- A correlation matrix is obtained which is later used to identify similar regions.

```
In [5]: borda_counts_agg=borda_counts_sum.unstack(0).fillna(0)
        (borda_counts_agg.head(5))
        # print(borda_counts_agg.shape())
```

Out[5]:

	borda_count								\
Region	ar	at	au	be	bg	bo	br	ca	
Track_Name									
#DansLeTierquar (Lyon)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
#DansLeTierquar (Marseille)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
#DansLeTierquar (Nantes)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
#FleKKsinonem	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
#JM	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

	…						\
Region	ch	cl	…	nl	no	nz	pa	pe	ph
Track_Name			…						
#DansLeTierquar (Lyon)	0.0	0.0	…	0.0	0.0	0.0	0.0	0.0	0.0
#DansLeTierquar (Marseille)	0.0	0.0	…	0.0	0.0	0.0	0.0	0.0	0.0
#DansLeTierquar (Nantes)	0.0	0.0	…	0.0	0.0	0.0	0.0	0.0	0.0
#FleKKsinonem	0.0	0.0	…	0.0	0.0	0.0	0.0	0.0	0.0
#JM	0.0	0.0	…	1710.0	0.0	0.0	0.0	0.0	0.0

Region	pl	pt	py	us
Track_Name				

```
#DansLeTierquar (Lyon)       0.0  0.0  0.0  0.0
#DansLeTierquar (Marseille)  0.0  0.0  0.0  0.0
#DansLeTierquar (Nantes)     0.0  0.0  0.0  0.0
#FleKKsinonem                0.0  0.0  0.0  0.0
#JM                          0.0  0.0  0.0  0.0

[5 rows x 50 columns]
```

In [6]:
```
correlation_df = borda_counts_agg.corr()
(correlation_df.head(10))
```

Out[6]:

		borda_count					\
Region		ar	at	au	be	bg	
	Region						
borda_count	ar	1.000000	0.095831	0.062599	0.121061	0.166404	
	at	0.095831	1.000000	0.675116	0.762622	0.503718	
	au	0.062599	0.675116	1.000000	0.794188	0.496251	
	be	0.121061	0.762622	0.794188	1.000000	0.541927	
	bg	0.166404	0.503718	0.496251	0.541927	1.000000	
	bo	0.892117	0.124531	0.094975	0.152705	0.195295	
	br	0.078602	0.138145	0.129008	0.170617	0.179886	
	ca	0.072878	0.592766	0.831562	0.724063	0.530299	
	ch	0.179035	0.876781	0.756086	0.854019	0.603136	
	cl	0.893863	0.074900	0.048237	0.102267	0.135811	

							…	\
Region		bo	br	ca	ch	cl	…	
	Region						…	
borda_count	ar	0.892117	0.078602	0.072878	0.179035	0.893863	…	
	at	0.124531	0.138145	0.592766	0.876781	0.074900	…	
	au	0.094975	0.129008	0.831562	0.756086	0.048237	…	
	be	0.152705	0.170617	0.724063	0.854019	0.102267	…	
	bg	0.195295	0.179886	0.530299	0.603136	0.135811	…	
	bo	1.000000	0.080720	0.106024	0.209890	0.908773	…	
	br	0.080720	1.000000	0.131013	0.169036	0.065909	…	
	ca	0.106024	0.131013	1.000000	0.722091	0.058293	…	
	ch	0.209890	0.169036	0.722091	1.000000	0.156883	…	
	cl	0.908773	0.065909	0.058293	0.156883	1.000000	…	

							\
Region		nl	no	nz	pa	pe	
	Region						
borda_count	ar	0.101912	0.078232	0.070010	0.832563	0.902296	
	at	0.600935	0.689751	0.654965	0.164182	0.086164	
	au	0.598602	0.696008	0.946770	0.140736	0.059822	
	be	0.753263	0.724012	0.783277	0.200044	0.113924	
	bg	0.437422	0.424242	0.503097	0.210954	0.148993	
	bo	0.125629	0.103387	0.104409	0.867072	0.952428	
	br	0.126763	0.120255	0.136360	0.093546	0.071595	
	ca	0.534528	0.613792	0.856070	0.161847	0.071663	
	ch	0.669149	0.734759	0.747439	0.255093	0.169324	
	cl	0.084650	0.060284	0.055528	0.885371	0.924411	

Region	ph	pl	pt	py	us

```
                Region
borda_count ar      0.054657  0.106816  0.216821  0.873692  0.047197
            at      0.454402  0.686050  0.618263  0.092228  0.394716
            au      0.562324  0.620396  0.729835  0.065705  0.688095
            be      0.525030  0.714664  0.736669  0.116149  0.527188
            bg      0.370037  0.566093  0.570828  0.141358  0.399004
            bo      0.084596  0.139274  0.247895  0.897894  0.071385
            br      0.110323  0.163049  0.231878  0.100009  0.089276
            ca      0.461632  0.573498  0.759162  0.076813  0.900300
            ch      0.489731  0.739038  0.761265  0.165049  0.523359
            cl      0.038788  0.088632  0.194301  0.918058  0.037653

[10 rows x 50 columns]
```

4.5 Similar regions

The correlation matrix is used to identify the top 5 similar regions for each region in the list.

```
In [7]: similar_regions = pd.DataFrame(columns=['Region','Top 5'])
        for region in regions:
            top=[]
            sim_region=[]
            top=list(correlation_df['borda_count'][region].sort_values(ascending=False).head(6))
            for item in top:
                temp=list((correlation_df == item).idxmax(axis=1)['borda_count'][region])
                # print(temp[1])
                sim_region.append(temp[1])
            sim_region.pop(0) # Removing the target  region as it is similar to itself
            row=[region,sim_region]
        #     # print(row)
            similar_regions = similar_regions.append(pd.Series(row,index=similar_regions.columns),ignor

        similar_regions.set_index('Region',inplace=True)
        similar_regions.head(5)
```

```
Out[7]:                          Top 5
        Region
        ee      [lt, lv, global, hu, cz]
        cr          [hn, gt, pa, bo, py]
        pt      [global, ch, ca, lv, nz]
        gr          [mt, bg, lu, ee, lt]
        br      [pt, hk, global, bg, lu]
```

4.6 Recommendations

The Recommendations for a particular region are made as follows:

4.6.1 Item-Based Collaborative Filtering (Track Based)

- For every target region:
  - Get a list of songs already ranked by that region
  - Find the top 5 most popular songs.
  - For every song in top 5:

    * Find other songs which belong to the same cluster and add to the recommended list.
  - From the recommended list, remove songs that already exist in the target region's rankings (analogy: Have already been bought by user)
  - Choose 5 songs in random from the cleaned recommended list for the final Recommendations.

4.6.2 User-Based Collaborative Filtering (Region Based)

- For every target region:
  - Find top 5 similar regions to target region, based on correlation values.
  - For every region in top 5:
    * Find the most popular songs of that region and add to recommended list.
  - From the recommended list, remove songs that already exist in the target region's rankings (analogy: Have already been bought by user)
  - Choose 5 songs in random from the cleaned recommended list for the final Recommendations.
- The Recommendations from the Item-based and User-Based approaches are combined and the final list is recommended to the user.
- The are are written to a csv file.

```
In [8]: recommendations = pd.DataFrame(columns=['Region','Songs'])
        track_list=pd.read_csv('track_list.csv')

        for region in regions:
            recommend_songs=[]
            temp=[]
            songs=region_songs[region]

            top_5=borda_counts_sum.loc[region].head(5)
            for track in top_5:
                cluster=track_list[track_list['Track_Name'] =='Shape of You'].Cluster
                sim_track_list=track_list.loc[track_list['Cluster'] == cluster.values[0]]['Track_Name']
                temp=set(temp)|set(sim_track_list)


            for sim_region in similar_regions['Top 5'][region]:
                sim_region_songs=borda_counts_sum.loc[sim_region].head(5)
                temp=set(temp)|set(sim_region_songs)

            temp=list(set(temp)-set(songs))
            recommend_songs=random.choices(temp, k=10)
            row=[region,recommend_songs]
            recommendations = recommendations.append(pd.Series(row,index=recommendations.columns),ignor
        recommendations.to_csv('Recommendations.csv')

        recommendations.head(10)
```

Out[8]:

	Region	Songs
0	ee	[Devuelveme, Nei, nei, ekki um jólin, Sieben I…
1	cr	[Rica, Bien Duro, Silhouettes - Original Radio…
2	pt	[Ritmo Mexicano, Limonada Coco - Remix, Slecht…
3	gr	[Fear for Nobody, Yhen elämän juttu, Ai Ai Ai,…
4	br	[Geen Seconde Rust, Qui dit mieux (feat. Orels…
5	es	[Kontrollieren, What You Want, Migraine, Llama…

```
6     lu  [Juna, LA CRIMINAL, Che ne sai, Livet Er For K…
7     ca  [Kan Niet Kiezen, Kruunu tikittää (feat.TIPPA)…
8     id  [Never Let Me Go, Range, Destiny, Oh La La - T…
9     lt  [Mi No Lob, Brr Brr, One Life, Le Encanta, Vuo…
```

Chapter 5

Forecasting

- This file deals with time series analysis and forecasting.
- The trends in popularity of a song over timme are analyzed and visualised.
- The change in popularity of the song is also forecasted.

5.1 Importing libraries

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        import itertools
        from pylab import rcParams
        import statsmodels.tsa.seasonal as smts
        import statsmodels.tsa as smt
        from statsmodels.tsa.seasonal import seasonal_decompose
        import statsmodels.tsa.statespace.sarimax as sari
        from fbprophet import Prophet
```

```
Importing plotly failed. Interactive plots will not work.
```

5.2 Importing data

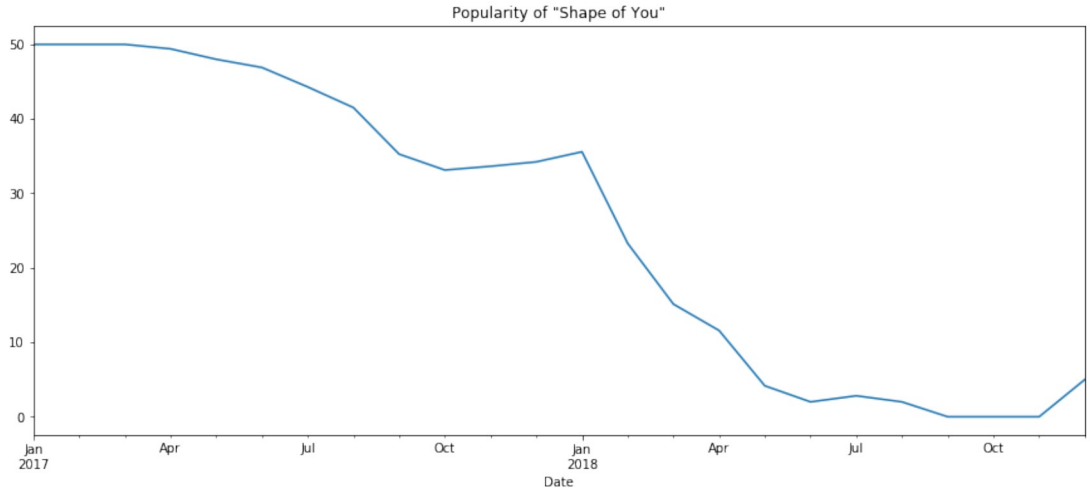
- The data from 'borda_counts.csv' is imported in order to get the rankings and popularity score of the songs over time.
- From the visualizations and basic EDA it was clear that the two most popular songs from 2017-2018 were 'Shape of You' and 'Havana'.
- Hence the data of these two songs are considered with respect to the 'global' region.
- Data about 'Shape of You' is stored in 'Shape_Data' and 'Havana' in 'Hav_Data'

```
In [2]: data=pd.read_csv('borda_counts.csv')

        global_data=pd.DataFrame(data.loc[data['Region']=='global'])

        Shape_Data = global_data[global_data['Track_Name'] == 'Shape of You'][['Date','borda_count']]
        Hav_Data= global_data[global_data['Track_Name'] == 'Havana'][['Date','borda_count']]
```

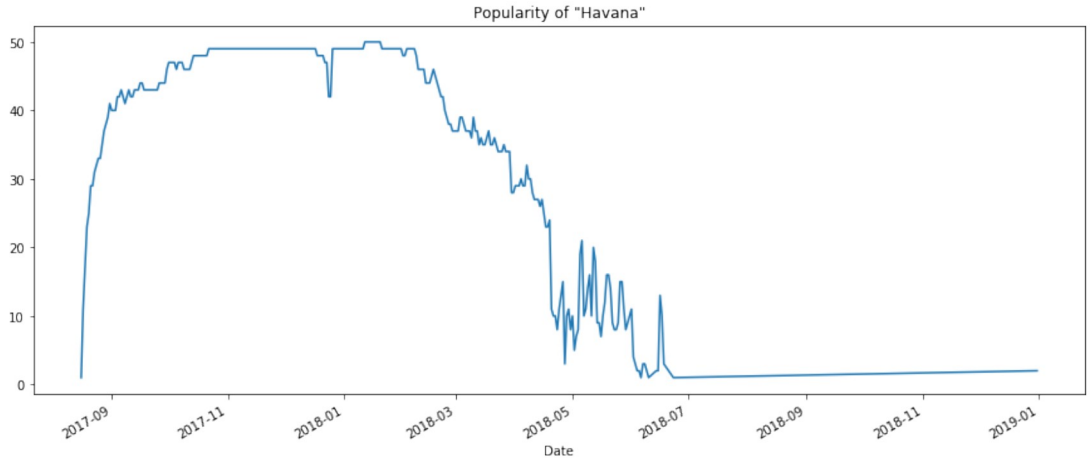
5.3 Visualizing popularity time series data

- For easier visualization the average monthly borda_count (popularity score) is used with the start of each month as the timestamp.
- Distinguishable patterns are observed in the plot.
- The popularity seems to have peeked around the beginning of 2018

In [3]:
```
# Convert data in 'Date' column to DateTime format and set as index
Shape_Data['Date'] = pd.to_datetime(Shape_Data['Date'])
Shape_Data=Shape_Data.set_index('Date')

 # resample to get monthly average
trend_shape = Shape_Data['borda_count'].resample('MS').mean().fillna(0)

trend_shape.plot(figsize=(15, 6))
plt.title('Popularity of "Shape of You"')
plt.show()
```



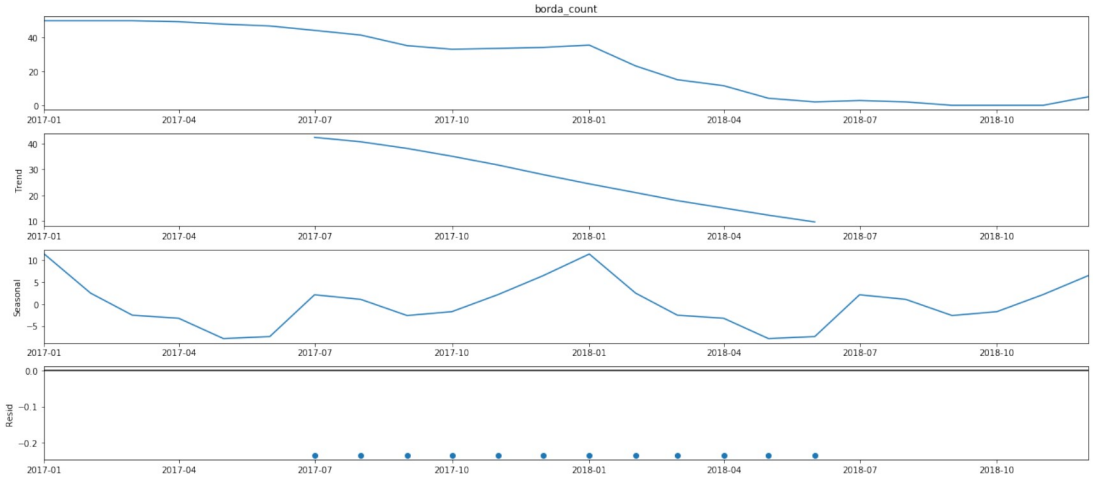
In [4]:
```
# Convert data in 'Date' column to DateTime format and set as index
Hav_Data['Date'] = pd.to_datetime(Hav_Data['Date'])
Hav_Data=Hav_Data.set_index('Date')

# As the Havana data is relatively sparse with respect frequency, it is not sampled but used as
trend_Havana = Hav_Data['borda_count']

trend_Havana.plot(figsize=(15, 6))
plt.title('Popularity of "Havana"')
plt.show()
```



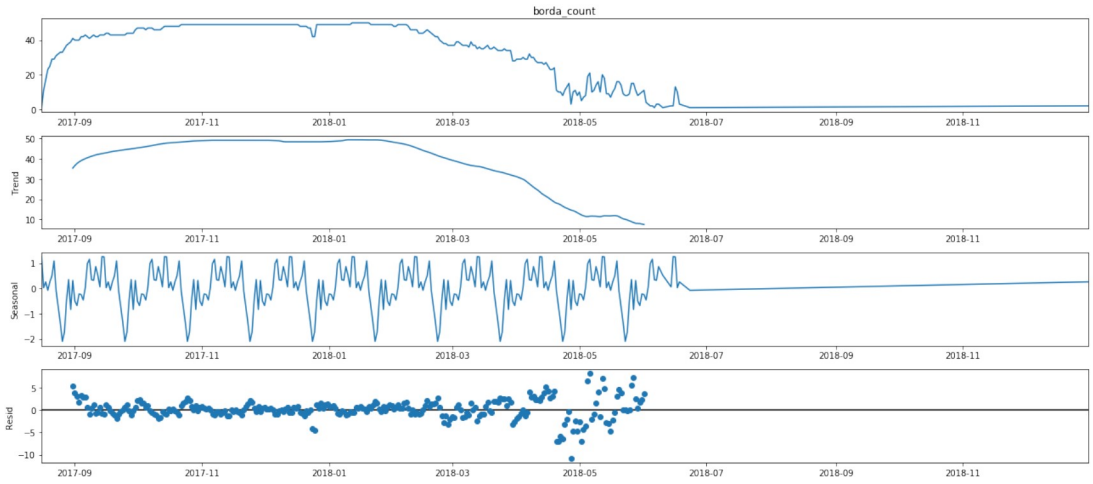
5.4 Time Series Decomposition

- Decompose time series into three distinct components: trend, seasonality, and noise to better understand the changes.

```
In [5]: rcParams['figure.figsize'] = 18, 8
        decomposition = smts.seasonal_decompose(trend_shape, model='additive')
        fig = decomposition.plot()
        plt.show()
```




```
In [6]: rcParams['figure.figsize'] = 18, 8
        decomposition = smts.seasonal_decompose(trend_Havana, model='additive',period=30)
        fig = decomposition.plot()
        plt.show()
```

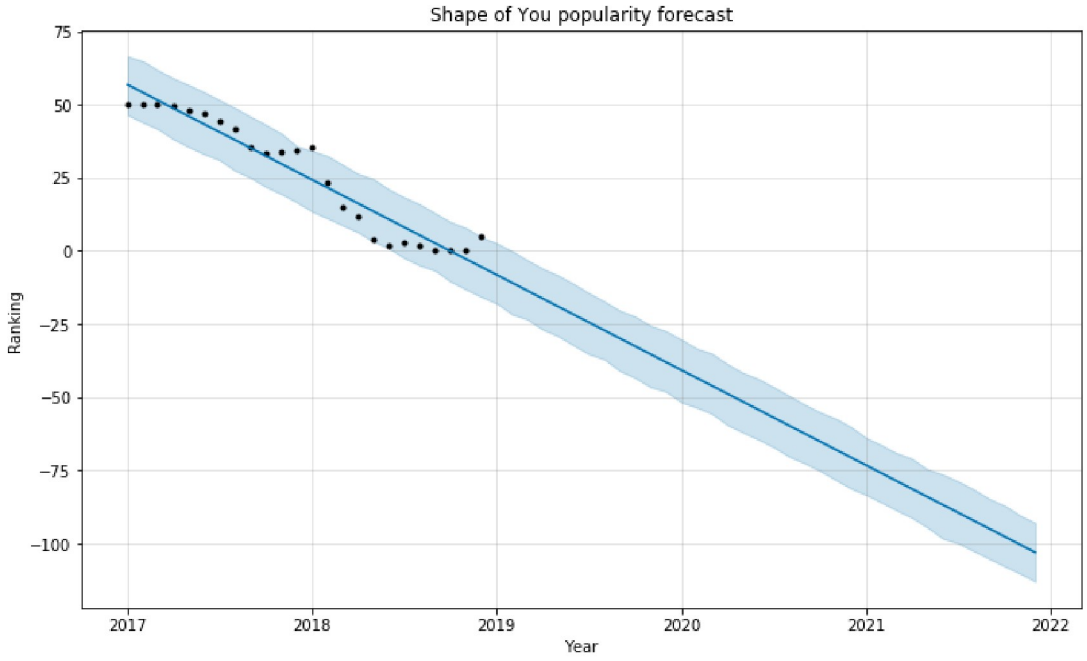


5.5 Time series modelling with Prohpet

- Using Prophet to analyze time-series and find patterns on different time scales such as yearly, weekly and daily.
- There is a clear decline in the popularity of the song after a perios of two years.
- The forecast/prediction of the popularity is also plotted, along with the confidence intervals of the predictions.

```
In [7]: shape_model_data=pd.DataFrame(columns=['ds','y'])
        shape_model_data['ds']=trend_shape.index
        shape_model_data['y']=trend_shape.values

        shape_model=Prophet(interval_width=0.95)
        shape_model.fit(shape_model_data)
        shape_future=shape_model.make_future_dataframe(periods=36, freq='MS')
        shape_forecast=shape_model.predict(shape_future)



        plt.figure(figsize=(18, 6))
        shape_model.plot(shape_forecast, xlabel = 'Year', ylabel = 'Ranking')
        plt.title('Shape of You popularity forecast');
        plt.show()

        shape_forecast.head(10)
```

```
INFO:fbprophet:Disabling yearly seasonality. Run prophet with yearly_seasonality=True to override this.
INFO:fbprophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=True to override this.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
INFO:fbprophet:n_changepoints greater than number of observations. Using 18.


<Figure size 1296x432 with 0 Axes>
```



Out[7]:

	ds	trend	yhat_lower	yhat_upper	trend_lower	trend_upper
0	2017-01-01	56.784122	46.454021	66.503081	56.784122	56.784122
1	2017-02-01	54.025231	43.825231	64.803807	54.025231	54.025231
2	2017-03-01	51.533328	41.729657	61.837485	51.533328	51.533328
3	2017-04-01	48.774437	38.188213	59.044798	48.774437	48.774437
4	2017-05-01	46.104542	35.577219	56.780450	46.104542	46.104542
5	2017-06-01	43.345650	33.071595	54.308411	43.345650	43.345650
6	2017-07-01	40.675755	31.100483	51.642242	40.675755	40.675755
7	2017-08-01	37.916863	27.417992	48.928999	37.916863	37.916863
8	2017-09-01	35.157971	24.996496	45.934442	35.157971	35.157971
9	2017-10-01	32.488076	21.889659	43.181675	32.488076	32.488076

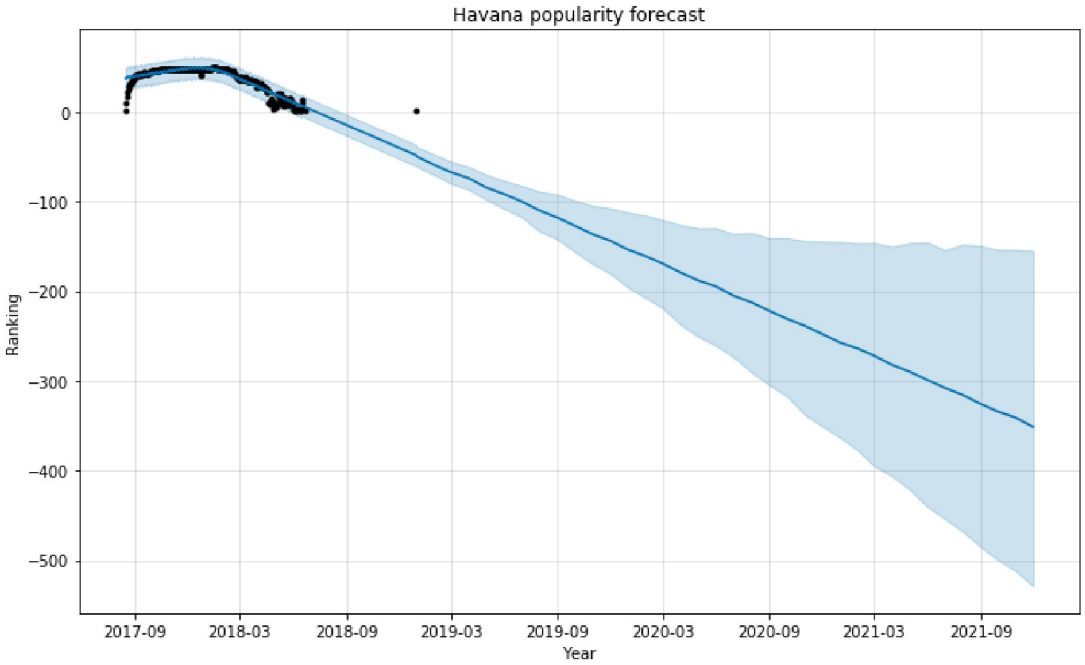
	additive_terms	additive_terms_lower	additive_terms_upper
0	0.0	0.0	0.0
1	0.0	0.0	0.0
2	0.0	0.0	0.0
3	0.0	0.0	0.0
4	0.0	0.0	0.0
5	0.0	0.0	0.0
6	0.0	0.0	0.0
7	0.0	0.0	0.0
8	0.0	0.0	0.0
9	0.0	0.0	0.0

	multiplicative_terms	multiplicative_terms_lower
0	0.0	0.0
1	0.0	0.0
2	0.0	0.0
3	0.0	0.0

```
4                    0.0                         0.0
5                    0.0                         0.0
6                    0.0                         0.0
7                    0.0                         0.0
8                    0.0                         0.0
9                    0.0                         0.0

   multiplicative_terms_upper       yhat
0                         0.0  56.784122
1                         0.0  54.025231
2                         0.0  51.533328
3                         0.0  48.774437
4                         0.0  46.104542
5                         0.0  43.345650
6                         0.0  40.675755
7                         0.0  37.916863
8                         0.0  35.157971
9                         0.0  32.488076
```

In [8]:
```
model_data=pd.DataFrame(columns=['ds','y'])
model_data['ds']=trend_Havana.index
model_data['y']=trend_Havana.values

Hav_model=Prophet(interval_width=0.95)
Hav_model.fit(model_data)
Hav_future=Hav_model.make_future_dataframe(periods=36, freq='MS')

Hav_forecast=Hav_model.predict(Hav_future)

plt.figure(figsize=(18, 6))
Hav_model.plot(Hav_forecast, xlabel = 'Year', ylabel = 'Ranking')
plt.title('Havana popularity forecast');
plt.show()

Hav_forecast.head(10)
```

```
INFO:fbprophet:Disabling yearly seasonality. Run prophet with yearly_seasonality=True to override this.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.


<Figure size 1296x432 with 0 Axes>
```



Out[8]:

	ds	trend	yhat_lower	yhat_upper	trend_lower	trend_upper
0	2017-08-16	38.419924	25.361127	49.685667	38.419924	38.419924
1	2017-08-17	38.520259	26.013374	50.163491	38.520259	38.520259
2	2017-08-18	38.620595	26.639676	50.080907	38.620595	38.620595
3	2017-08-19	38.720931	27.469153	49.353380	38.720931	38.720931
4	2017-08-20	38.821267	28.015598	51.360489	38.821267	38.821267
5	2017-08-21	38.921603	28.262513	52.505100	38.921603	38.921603
6	2017-08-22	39.021938	27.883632	50.067228	39.021938	39.021938
7	2017-08-23	39.122274	26.471702	51.335480	39.122274	39.122274
8	2017-08-24	39.222610	27.422654	50.609528	39.222610	39.222610
9	2017-08-25	39.322946	26.831177	48.837637	39.322946	39.322946

	additive_terms	additive_terms_lower	additive_terms_upper	weekly
0	-0.780210	-0.780210	-0.780210	-0.780210
1	-0.355638	-0.355638	-0.355638	-0.355638
2	-0.931794	-0.931794	-0.931794	-0.931794
3	0.601990	0.601990	0.601990	0.601990
4	0.505561	0.505561	0.505561	0.505561
5	1.039412	1.039412	1.039412	1.039412
6	-0.079321	-0.079321	-0.079321	-0.079321
7	-0.780210	-0.780210	-0.780210	-0.780210
8	-0.355638	-0.355638	-0.355638	-0.355638
9	-0.931794	-0.931794	-0.931794	-0.931794

	weekly_lower	weekly_upper	multiplicative_terms
0	-0.780210	-0.780210	0.0
1	-0.355638	-0.355638	0.0
2	-0.931794	-0.931794	0.0
3	0.601990	0.601990	0.0

```
4      0.505561      0.505561                  0.0
5      1.039412      1.039412                  0.0
6     -0.079321     -0.079321                  0.0
7     -0.780210     -0.780210                  0.0
8     -0.355638     -0.355638                  0.0
9     -0.931794     -0.931794                  0.0

   multiplicative_terms_lower  multiplicative_terms_upper       yhat
0                         0.0                         0.0  37.639714
1                         0.0                         0.0  38.164621
2                         0.0                         0.0  37.688801
3                         0.0                         0.0  39.322921
4                         0.0                         0.0  39.326828
5                         0.0                         0.0  39.961015
6                         0.0                         0.0  38.942618
7                         0.0                         0.0  38.342064
8                         0.0                         0.0  38.866972
9                         0.0                         0.0  38.391151
```

5.6 Merging Forecast to compare trends

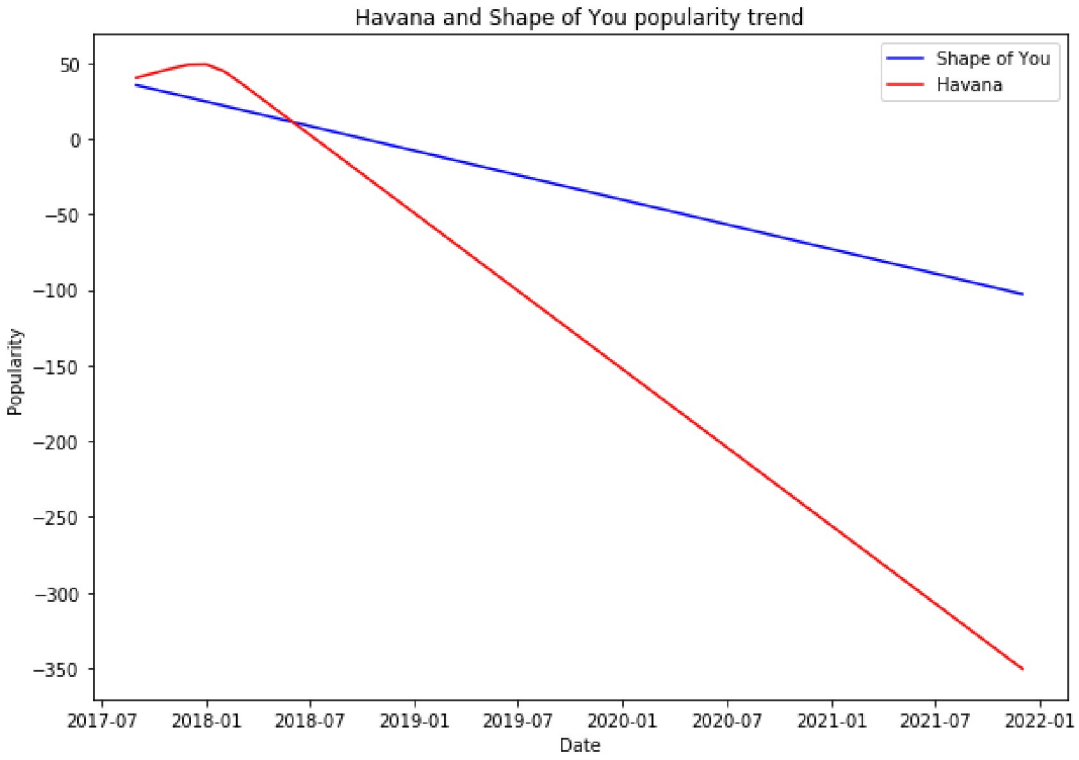
- The Forecasts of both songs are merged and the trends are plotted in a single graph.
- Both songs seem to have the same trend. There is a decrease in popularity over time.
- Howeve for a brief period, "Havana" seems to have been more popular than 'Shape of you' although in general the latter is more popular.

```
In [9]: shape_names = ['shape_%s' % column for column in shape_forecast.columns]
        havana_names = ['havana_%s' % column for column in Hav_forecast.columns]


        merge_havana_forecast = Hav_forecast.copy()
        merge_shape_forecast = shape_forecast.copy()

        merge_shape_forecast.columns = shape_names
        merge_havana_forecast.columns = havana_names

        forecast = pd.merge(merge_shape_forecast, merge_havana_forecast, how = 'inner', left_on = 'shape

        forecast = forecast.rename(columns={'shape_ds': 'Date'}).drop('havana_ds', axis=1)
        forecast.head()
```

```
Out[9]:         Date  shape_trend  shape_yhat_lower  shape_yhat_upper  \
        0 2017-09-01    35.157971         24.996496         45.934442
        1 2017-10-01    32.488076         21.889659         43.181675
        2 2017-11-01    29.729185         19.436036         40.158778
        3 2017-12-01    27.059289         16.660698         35.827176
        4 2018-01-01    24.300398         13.472911         34.189134

           shape_trend_lower  shape_trend_upper  shape_additive_terms  \
        0          35.157971          35.157971                   0.0
        1          32.488076          32.488076                   0.0
        2          29.729185          29.729185                   0.0
        3          27.059289          27.059289                   0.0
        4          24.300398          24.300398                   0.0
```

```
   shape_additive_terms_lower  shape_additive_terms_upper  \
0                         0.0                         0.0
1                         0.0                         0.0
2                         0.0                         0.0
3                         0.0                         0.0
4                         0.0                         0.0

   shape_multiplicative_terms  …  havana_additive_terms  \
0                         0.0  …              -0.931794
1                         0.0  …               0.505561
2                         0.0  …              -0.780210
3                         0.0  …              -0.931794
4                         0.0  …               1.039412

   havana_additive_terms_lower  havana_additive_terms_upper  havana_weekly  \
0                    -0.931794                    -0.931794      -0.931794
1                     0.505561                     0.505561       0.505561
2                    -0.780210                    -0.780210      -0.780210
3                    -0.931794                    -0.931794      -0.931794
4                     1.039412                     1.039412       1.039412

   havana_weekly_lower  havana_weekly_upper  havana_multiplicative_terms  \
0            -0.931794            -0.931794                          0.0
1             0.505561             0.505561                          0.0
2            -0.780210            -0.780210                          0.0
3            -0.931794            -0.931794                          0.0
4             1.039412             1.039412                          0.0

   havana_multiplicative_terms_lower  havana_multiplicative_terms_upper  \
0                                0.0                                0.0
1                                0.0                                0.0
2                                0.0                                0.0
3                                0.0                                0.0
4                                0.0                                0.0

   havana_yhat
0    39.093502
1    43.540931
2    45.342368
3    47.751932
4    49.936813

[5 rows x 28 columns]
```

In [10]:
```
plt.figure(figsize=(10, 7))
plt.plot(forecast['Date'], forecast['shape_trend'], 'b-',label='Shape of You')
plt.plot(forecast['Date'], forecast['havana_trend'], 'r-',label='Havana')
plt.legend();  plt.xlabel('Date'); plt.ylabel('Popularity')
plt.title('Havana and Shape of You popularity trend');
```

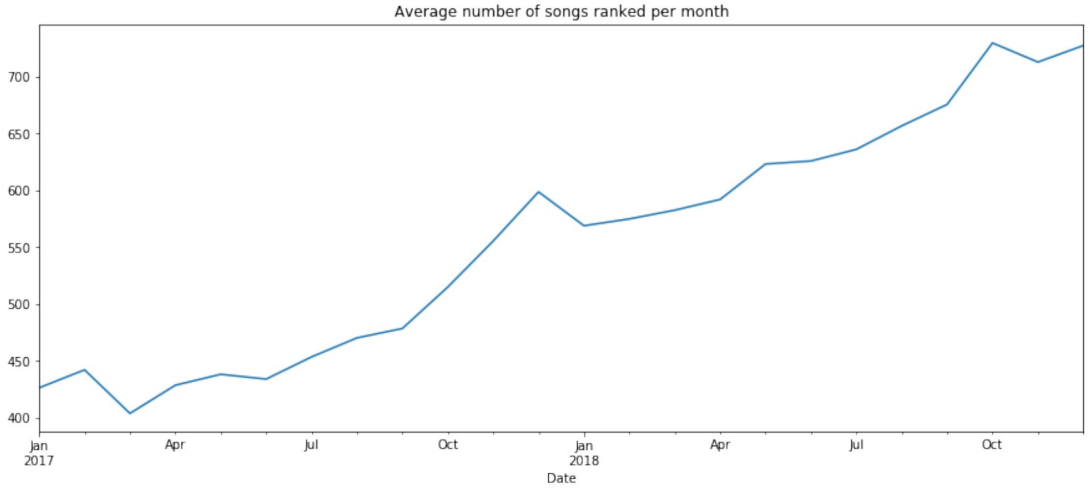


5.7 Song Data - Average number of songs ranked

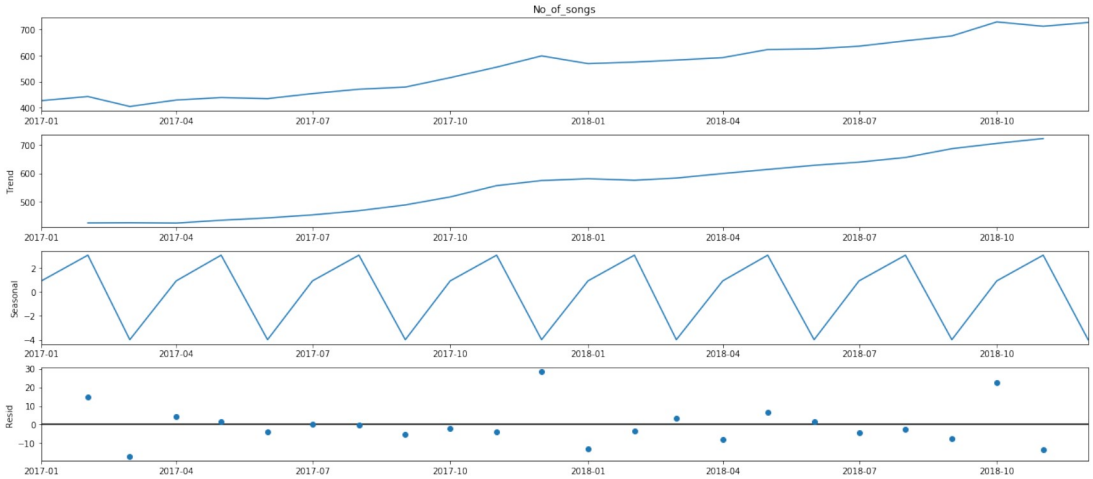
- The total number of unique songs that find a place in the rankings globally is obtained.
- This is plotted to find the trend in average number of songs that get ranked every month.
- The forecast/prediction of the number of songs has also been plotted.
- There is a clear upwards trend. This could be attributed to the following reasons:
  - As new songs come out over time, they get added to the rankings.
  - However, the rate at which older songs disappear from the rankings is less, and hence there is an upwards trend.

```
In [11]: data=pd.read_csv('cleaned_ranks.csv')
         datewise_data=(data.groupby(['Date'])['Track_Name'].unique())
         count=[]
         for date in datewise_data:
             count.append(len(date))

         songs_per_day=pd.DataFrame(columns=['Date','No_of_songs'])
         songs_per_day['Date']=datewise_data.index
         songs_per_day['No_of_songs']=count

         songs_per_day['Date'] = pd.to_datetime(songs_per_day['Date'])
         songs_per_day=songs_per_day.set_index('Date')
         # songs_per_day
         songs_per_day = songs_per_day['No_of_songs'].resample('MS').mean().fillna(0)
         songs_per_day.plot(figsize=(15, 6))
```

```
plt.title('Average number of songs ranked per month')
plt.show()
```



In [12]:
```
rcParams['figure.figsize'] = 18, 8
decomposition = smts.seasonal_decompose(songs_per_day, model='additive',period=3)
fig = decomposition.plot()
plt.show()
```

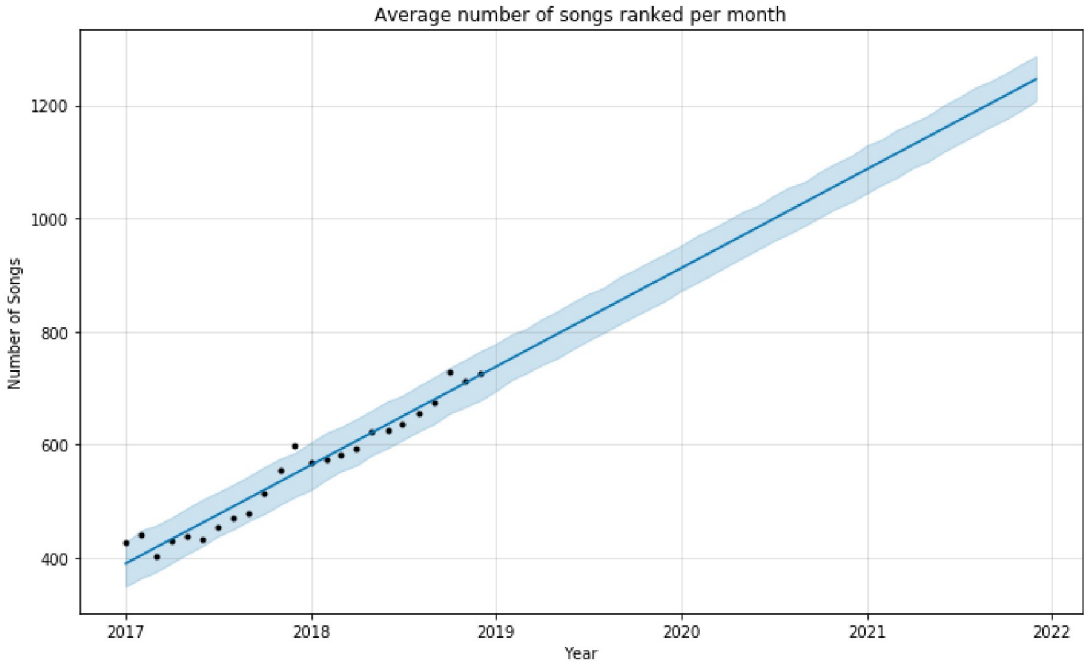


In [13]:
```
model_data=pd.DataFrame(columns=['ds','y'])
model_data['ds']=songs_per_day.index
model_data['y']=songs_per_day.values

songs_model=Prophet(interval_width=0.95)
songs_model.fit(model_data)
songs_future=songs_model.make_future_dataframe(periods=36, freq='MS')
```

```
songs_forecast=songs_model.predict(songs_future)


plt.figure(figsize=(18, 6))
songs_model.plot(songs_forecast, xlabel = 'Year', ylabel = 'Number of Songs')
plt.title('Average number of songs ranked per month');
plt.show()

songs_forecast.head(5)

# plt.figure(figsize=(10, 7))
# plt.plot(songs_forecast.index, songs_forecast['trend'], 'b-')
# plt.xlabel('Date'); plt.ylabel('Number of Songs')
# plt.title('Average number of songs ranked per month');
```

```
INFO:fbprophet:Disabling yearly seasonality. Run prophet with yearly_seasonality=True to override this.
INFO:fbprophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=True to override this.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
INFO:fbprophet:n_changepoints greater than number of observations. Using 18.
```

```
<Figure size 1296x432 with 0 Axes>
```



Out[13]:

	ds	trend	yhat_lower	yhat_upper	trend_lower	trend_upper	\
0	2017-01-01	390.011609	349.678898	427.150664	390.011609	390.011609	
1	2017-02-01	404.752421	364.680542	449.291984	404.752421	404.752421	
2	2017-03-01	418.066703	374.784188	457.063761	418.066703	418.066703	

```
3 2017-04-01  432.807515  390.755761  471.343942   432.807515   432.807515
4 2017-05-01  447.074149  406.431903  487.309853   447.074149   447.074149

   additive_terms  additive_terms_lower  additive_terms_upper  \
0             0.0                   0.0                   0.0
1             0.0                   0.0                   0.0
2             0.0                   0.0                   0.0
3             0.0                   0.0                   0.0
4             0.0                   0.0                   0.0

   multiplicative_terms  multiplicative_terms_lower  \
0                   0.0                         0.0
1                   0.0                         0.0
2                   0.0                         0.0
3                   0.0                         0.0
4                   0.0                         0.0

   multiplicative_terms_upper        yhat
0                         0.0  390.011609
1                         0.0  404.752421
2                         0.0  418.066703
3                         0.0  432.807515
4                         0.0  447.074149
```