DATA FRAME MANIPULATION METHODS

SATHYANARAYANAN RV

CONTENT

DATASET

- A **data set** (or **dataset**) is a collection of data. In the case of tabular data, a data set corresponds to one or more database tables, where every column of a table represents a particular variable, and each row corresponds to a given record of the data set in question. The data set lists values for each of the variables, such as height and weight of an object, for each member of the data set. Each value is known as a datum. Data sets can also consist of a collection of documents or files.

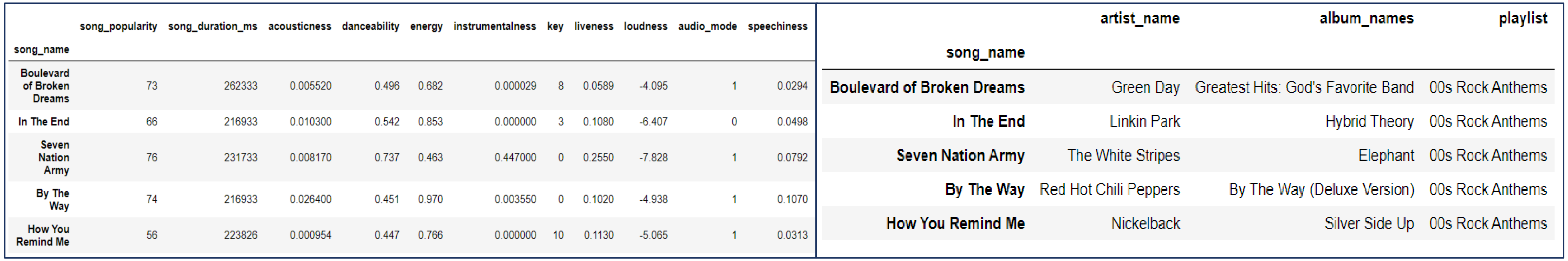
DATASET TAKEN HERE:

- Spotify songs with song data and its information has been separately given in two csv files.
- ***Song_data*** dataset has 18835 rows and 15 columns; some of the columns are ***song_name, song_popularity, song_duration_ms etc.,***
- ***Song_info*** dataset has 18835 rows and 4 columns; some of the columns are ***song_name, artist_name, etc.,***
- *So it is understood that **song_name** been acting as the common column for both the datasets*

LOADING THE DATASET

- Since the file is in the format of csv, we are loading the dataset using the following codes
  - song_d = pd.read_csv('song_data.csv', header=0, index_col=0)
  - song_i = pd.read_csv('song_info.csv', header=0, index_col=0)

As the data loaded, the head function returns here the top 5 rows of the Data Frames.



song_name	song_popularity	song_duration_ms	acousticness	danceability	energy	instrumentalness	key	liveness	loudness	audio_mode	speechiness
Boulevard of Broken Dreams	73	262333	0.005520	0.496	0.682	0.000029	8	0.0589	-4.095	1	0.0294
In The End	66	216933	0.010300	0.542	0.853	0.000000	3	0.1080	-6.407	0	0.0498
Seven Nation Army	76	231733	0.008170	0.737	0.463	0.447000	0	0.2550	-7.828	1	0.0792
By The Way	74	216933	0.026400	0.451	0.970	0.003550	0	0.1020	-4.938	1	0.1070
How You Remind Me	56	223826	0.000954	0.447	0.766	0.000000	10	0.1130	-5.065	1	0.0313

song_name	artist_name	album_names	playlist
Boulevard of Broken Dreams	Green Day	Greatest Hits: God's Favorite Band	00s Rock Anthems
In The End	Linkin Park	Hybrid Theory	00s Rock Anthems
Seven Nation Army	The White Stripes	Elephant	00s Rock Anthems
By The Way	Red Hot Chili Peppers	By The Way (Deluxe Version)	00s Rock Anthems
How You Remind Me	Nickelback	Silver Side Up	00s Rock Anthems

SAVING THE DATA FRAME

- Saving a Data Frame to a CSV file
  - We could able to save the data frame into a Comma Separated Values file and the following code will do that for us.
  - Song_d.to_csv('song_d.csv', encoding='utf-8')
- Saving a Data Frame to a Python dictionary
  - We could able to save the data frame into a dictionary and the following code will do that for us.
  - dictionary = Song_d.to_dict()
- Saving a Data Frame to a Python string
  - We could able to save the data frame into a string and the following code will do that for us.
  - string = Song_d.to_string()

WORKING WITH DATA FRAMES

- Song_d.info()
  - It will give the information of all the columns in the DF with total number of non-null values with datatypes
- Song_d.describe()
  - It will give us the statistical counts of the whole data frame
- Song_d.head(n)
  - It will provide the top ‘n’ number of rows with all the column values in the DF
- Song_d.tail(n)
  - It will provide the bottom ‘n’ number of rows with all the column values in the DF
- Song_d.shape
  - It will show the number of rows and column values present merely in the DF

WORKING WITH DATA FRAMES(CONTD.)

Working with rows:

- *song_d1 = song_d[song_d['song_popularity']>=90]* will return the new data frame with the song popularity greater or equal to 90
- *song_d = song_d[pd.notnull(song_d['song_name'])]* will return the data frame with the non-null values of the column 'song_name' throughout the DF
- *sd_new.sort_index()* will give the result of the data frame with the indices sorted
- *sd_new.sort_values(by='song_duration_ms',ascending=False)* will sort the data frames values based on the column supplied here and in descending order
- *song_d.iloc[:100,:]* will retreive the rows index values of 0 to 100 with all the column values in it

Working with Cells:

- *song_d.loc[0:100,'song_name':'song_duration_ms']* Will do the slice and show the output between 0 and 100 rows with the column values provided
- *Song_d.loc[1]* and *song_d.song_name* will slice the row and column according to the name parsed

WORKING WITH DATA FRAMES(CONTD.)

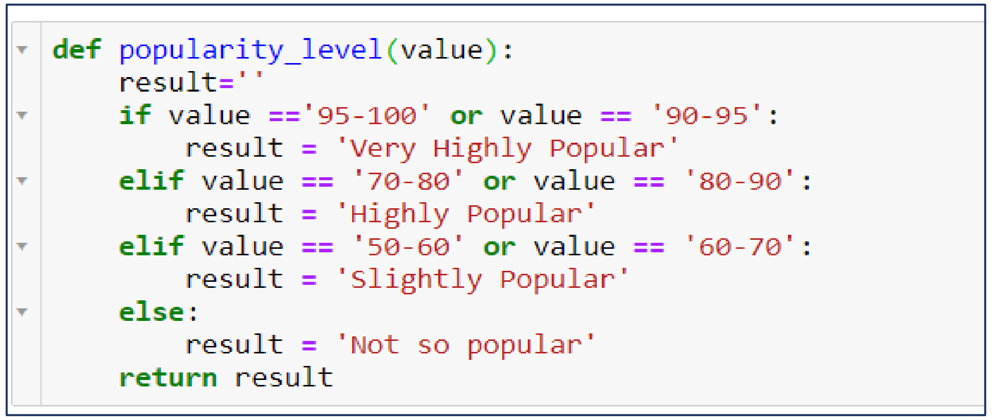
- **Working with Columns:**
  - *song_d1 = song_d1.drop(['audio_valence'], axis=1)* will return the data frame with the specified column removed from the whole data frame
  - *song_d1['popularity_range'] = pd.cut(song_d1.song_popularity,bins=bins,labels=range_v)* will do add the column named 'popularity_range' with the data frame by taking the column 'song_popularity' and binning has been done
  - *song_d1[['song_name','popularity_range','danceability']]* will have the view of the three columns mentioned in a data frame
  - **Find index label for min/max values in column**
    - *song_d['key'].idxmin()* and *song_d['key'].idxmax()* will show us the min and max values of the label column we code
  - **Maths on the whole DataFrame**
    - *song_d.count()* and *song_d.mean()* will retrieve the total count and average value of the DF, likewise it can be achieved to specified columns also.
    - *song_d.song_popularity.cumsum()* and *song_d.song_popularity.fillna(0)* will do the cumulative sum of each row value of the column and the second one will fill NaN values with 0

USER DEFINED FUNCTIONS

- A function is a reusable block of programming statements designed to perform a certain task. To define a function, Python provides the *def* keyword. The following is the syntax of defining a function.
- **Syntax**

```
def function_name(parameters):
    "function docstring"
    statement1
    statement2
    ...
    return [expr]
```

- **Example**



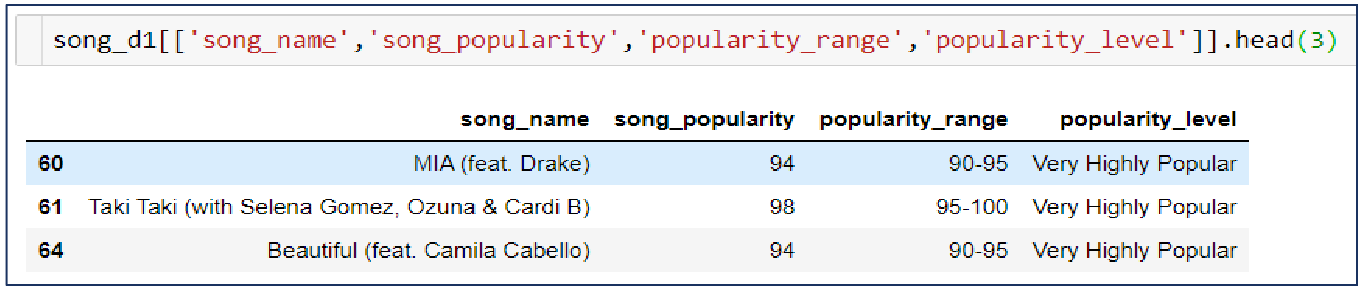
```
def popularity_level(value):
    result=''
    if value =='95-100' or value == '90-95':
        result = 'Very Highly Popular'
    elif value == '70-80' or value == '80-90':
        result = 'Highly Popular'
    elif value == '50-60' or value == '60-70':
        result = 'Slightly Popular'
    else:
        result = 'Not so popular'
    return result
```

USER DEFINED FUNCTIONS(CONTD.)

Function Explanation

- If the popularity range lies in 95-100 and 90-95 ranges, then the function return it as 'Very Highly Popular'
- If the popularity range lies in 80-90 and 70-80 ranges, then the function return it as 'Highly Popular'
- If the popularity range lies in 50-60 and 60-70 ranges, then the function return it as 'Slightly Popular'
- If all the above condition fails, it goes to the category 'Not so popular'

- If we apply our function using the following code, we could able to achieve the function working and creating a new column variable 'popularity_level' in the DF.
  - *song_d1['popularity_level'] = song_d1.popularity_range.apply(popularity_level)*



```
song_d1[['song_name','song_popularity','popularity_range','popularity_level']].head(3)
```

	song_name	song_popularity	popularity_range	popularity_level
60	MIA (feat. Drake)	94	90-95	Very Highly Popular
61	Taki Taki (with Selena Gomez, Ozuna & Cardi B)	98	95-100	Very Highly Popular
64	Beautiful (feat. Camila Cabello)	94	90-95	Very Highly Popular

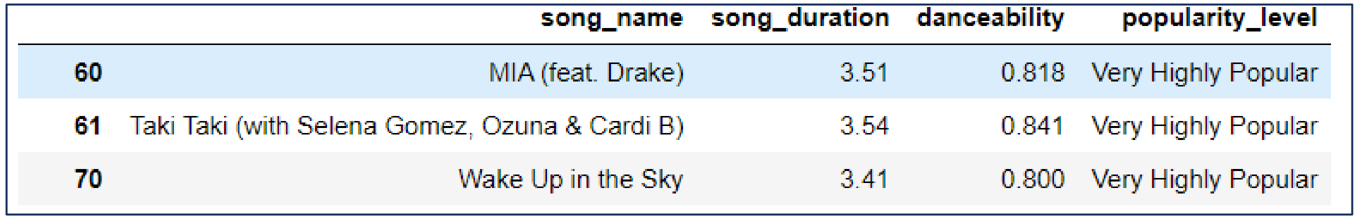
CLEANING DATASET

Creating two data frames

- With the conditional value, the data frame is getting separated into two as follows
- *song_d1_0 = song_d1[(song_d1.audio_mode==0)]*
  - DF song_d1_0 with audio_mode of 0 only
- *song_d1_1 = song_d1[(song_d1.audio_mode==1)]*
  - DF song_d1_1 with audio_mode of 1 only

Keeping Required Columns

- We are just taking some column alone in the data frames as follows
- *song_d1_0 = song_d1_0[['song_name','song_duration','danceability','popularity_level’]]*
- *song_d1_1 = song_d1_1[['song_name','song_duration','danceability','popularity_level']]*



	song_name	song_duration	danceability	popularity_level
60	MIA (feat. Drake)	3.51	0.818	Very Highly Popular
61	Taki Taki (with Selena Gomez, Ozuna & Cardi B)	3.54	0.841	Very Highly Popular
70	Wake Up in the Sky	3.41	0.800	Very Highly Popular

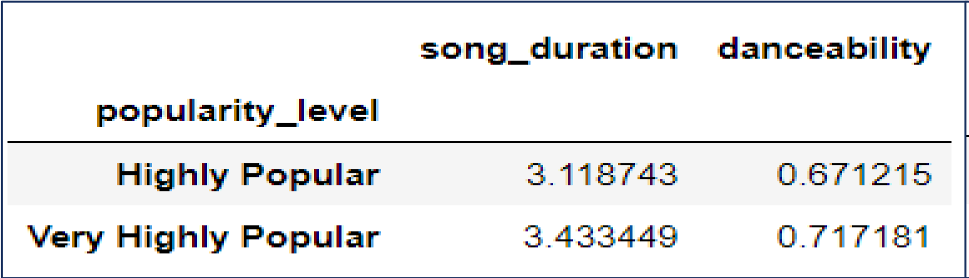
JOINING / GROUP BY DATA FRAMES

Joining Data Frames

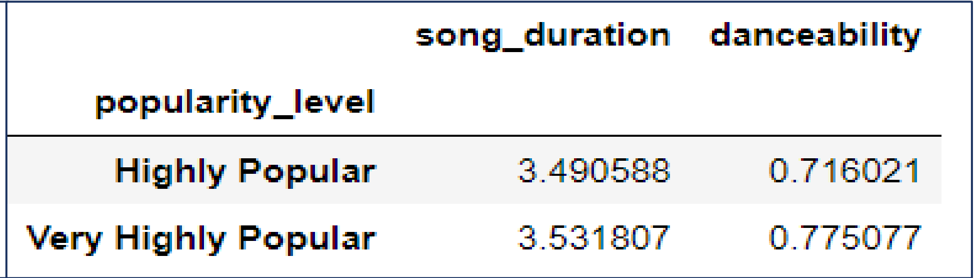
- The following code will merge the two dataset we have with the newly created data frames
  - *song_d1_0_new = pd.merge(left=song_d1_0, right=song_i, how='left',left_on='song_name',right_on='song_name’)*
  - *song_d1_1_new = pd.merge(left=song_d1_1, right=song_i, how='left',left_on='song_name',right_on='song_name’)*
- The above code created two new data frames with merged values from both the datasets

Grouping

- We are grouping based on the popularity level of the songs and the results are
  - *grouping_1 = song_d1_1_new.groupby(['popularity_level'])['song_duration','danceability'].agg(np.mean)*
  - *grouping_0 = song_d1_0_new.groupby(['popularity_level'])['song_duration','danceability'].agg(np.mean)*



	song_duration	danceability
popularity_level		
Highly Popular	3.118743	0.671215
Very Highly Popular	3.433449	0.717181

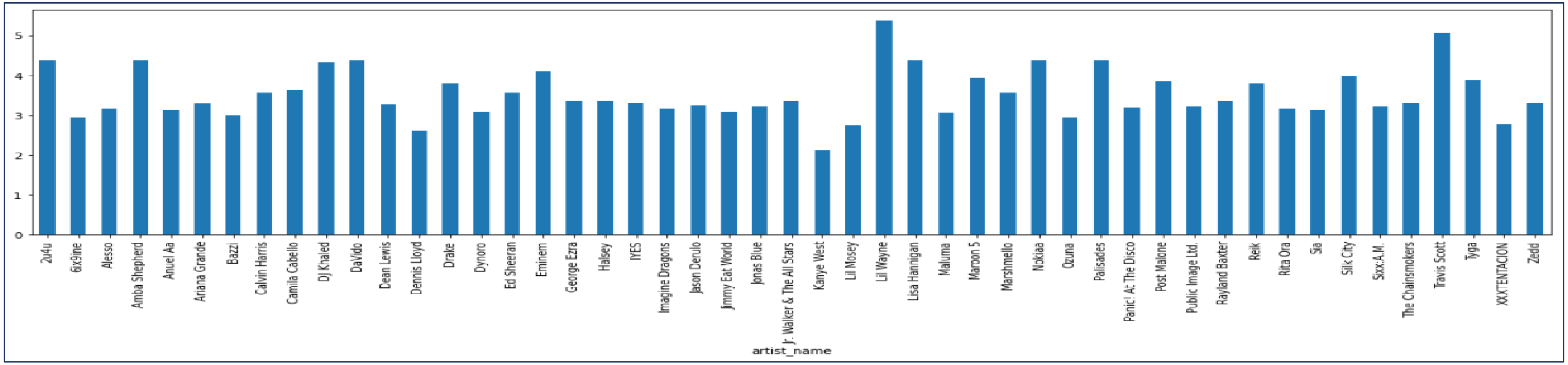


	song_duration	danceability
popularity_level		
Highly Popular	3.490588	0.716021
Very Highly Popular	3.531807	0.775077

GRAPHS AND STATISTICS

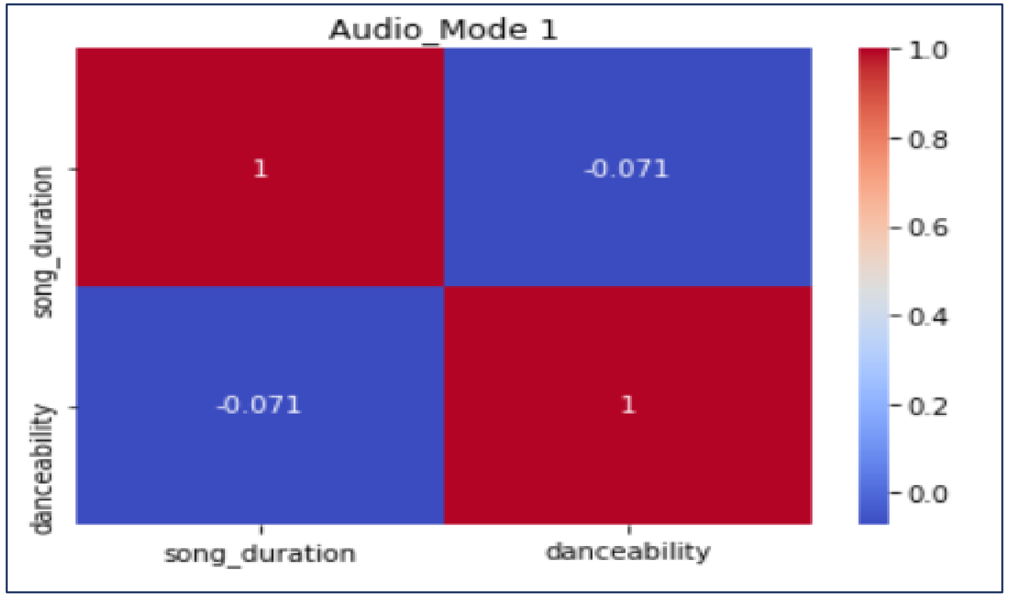
Graphs

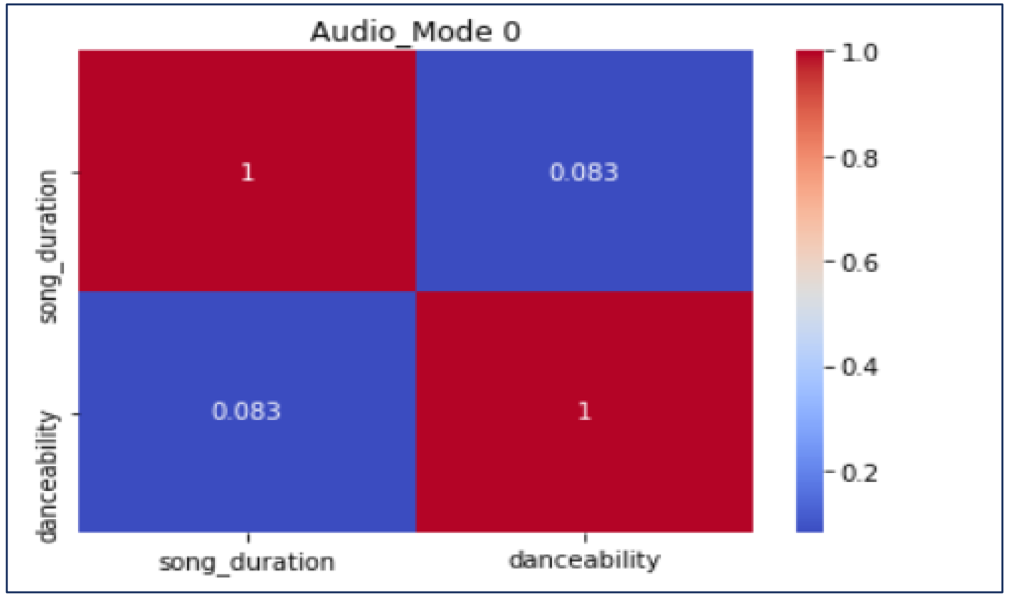
- While analyzing one of the data frame created , and plotted the same in bar plot
  - song_d1_1_new.groupby(by=['artist_name']).mean()['song_duration'].plot.bar()
  - We are taking the various artist with average minutes of a single song composed by.



CORRELATION

- Using the correlation function , we are finding the different insights out of it.
  - *sns.heatmap(song_d1_1_new.corr(),cmap='coolwarm',annot=True)*
  - *sns.heatmap(song_d1_0_new.corr(),cmap='coolwarm',annot=True)*
  - There is an inversional correlation between soong_duration and danceability in audio_mode 1 , while the audio_mode 0 has positively correlated to each other



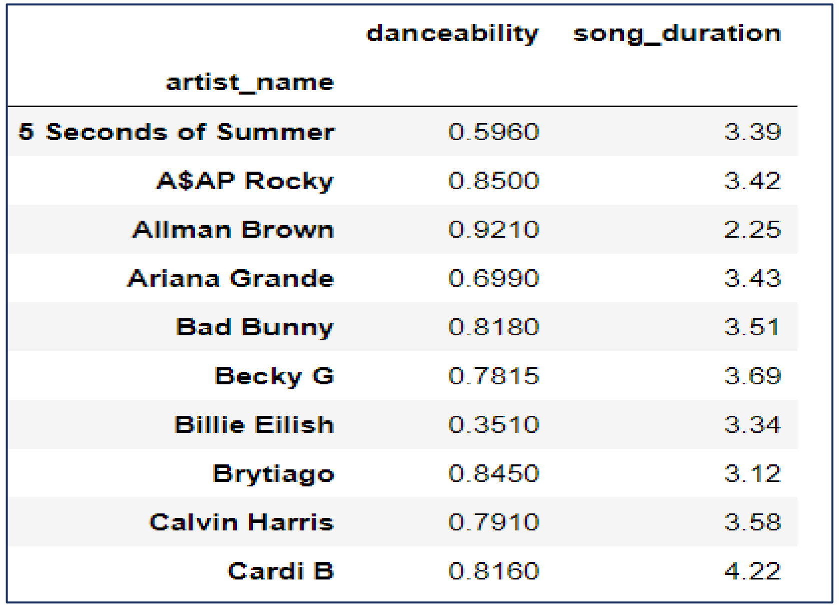


MISSING VALUES

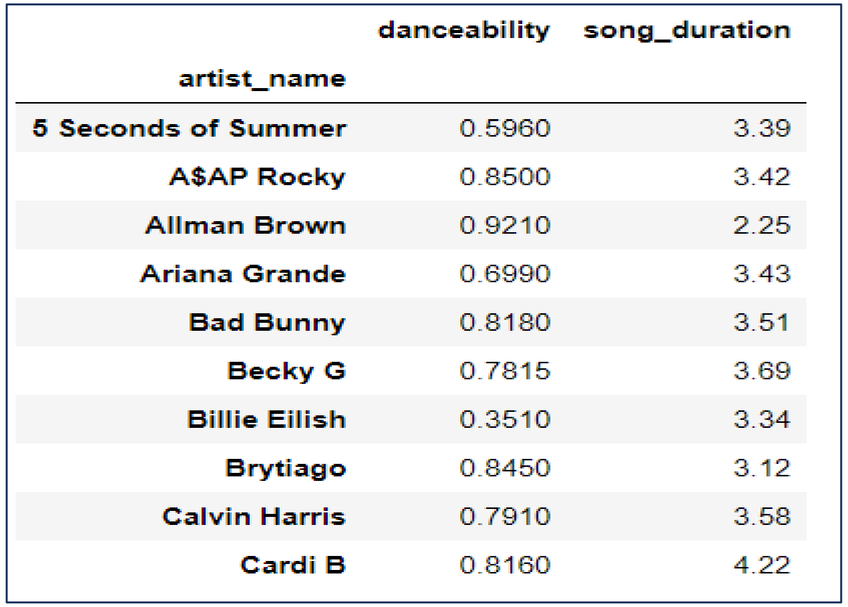
- **Drop all rows with NaN**
  - *Song_d1 = Song_d1.dropna()* will drop all the rows with NaN values
- **Drop all columns with NaN**
  - *Song_d1 = Song_d1.dropna(axis=1)* will drop all the columns with NaN values
- **Drop all rows where NaN appear more than twice**
  - *Song_d1 = Song_d1.dropna(thresh=2)* will drop all the rows with NaN values appears more than 2 times as given in the parameter ***threshold***
- **Drop all rows where NaN appear in a special column**
  - *Song_d1 = Song_d1.dropna(Song_d1 ['song_name'].notnull())* will drop all the rows with the condition parsed
- **Recoding all missing data**
  - *Song_d1.fillna(0, inplace=True)* will fill all the NaN with 0 and inplace parameter will ensure no new copy is produced
- **String Operation**
  - *Song_d1['song_name'].str.lower()* , *Song_d1 [song_name''].str.upper(), Song_d1 [song_name''].str.len()* and *Song_d1 [song_name''].str.replace('old', 'new')* operations are used to change all the string letters to lower case, upper case, finding its length and replacing the old word with the new one

PIVOT TABLES

- The following Pivot Table function will return the data frame object with index as ‘artist_name’ with the numeric columns average values
  - *song_d1_pt = pd.pivot_table(song_d1_0_new,index='artist_name’,,values=['song_duration','danceability’])*
  - *song_d0_pt = pd.pivot_table(song_d1_0_new,index='artist_name',values=['song_duration','danceability'])*



	danceability	song_duration
artist_name		
5 Seconds of Summer	0.5960	3.39
A$AP Rocky	0.8500	3.42
Allman Brown	0.9210	2.25
Ariana Grande	0.6990	3.43
Bad Bunny	0.8180	3.51
Becky G	0.7815	3.69
Billie Eilish	0.3510	3.34
Brytiago	0.8450	3.12
Calvin Harris	0.7910	3.58
Cardi B	0.8160	4.22



	danceability	song_duration
artist_name		
5 Seconds of Summer	0.5960	3.39
A$AP Rocky	0.8500	3.42
Allman Brown	0.9210	2.25
Ariana Grande	0.6990	3.43
Bad Bunny	0.8180	3.51
Becky G	0.7815	3.69
Billie Eilish	0.3510	3.34
Brytiago	0.8450	3.12
Calvin Harris	0.7910	3.58
Cardi B	0.8160	4.22

THANK YOU