Final Portfolio Piece (Project)

Spotify Recommendation and Prediction Analysis

Introduction

Spotify is one of the newest innovations to have come to audio listening and experience with over 125 million subscribers. Though the service has recently begun it dominates Apple Music and Amazon music in the audio streaming market. From music, they have extended the audio service to Podcasts, Audiobooks, and so on. Spotify Trends helps any content creator/musician in order to understand what listeners prefer and how to compete in this immensely growing market.

Overview

1. **Build an ML model** — To Predict the popularity of any song by analyzing metrics . This Prediction helps any content creator/musician to understand what kind of listeners prefer to hear nowadays.

2. It's important to start by doing Exploratory Analysis and achieve a few insights from data. Find out which features are highly correlated with the Popularity attribute. The next step is to test different model algorithms and pick the best model based on key evaluation metric (R2 Score)

2\. **Build a content-based Recommendation system** that can suggest artists for any users. This helps users to listen to songs based on their music preferences.

Data

The dataset is scrapped using the Spotify API.This is basically a computer algorithm that Spotify has that can estimate various aspects of the audio file.

Some of the key attributes present in each event in the data are:

- **key** — The estimated overall key of the track. Integers map to pitches using standard Pitch Class notation.
- **Mode** — Mode indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived. Major is represented by 1 and minor is 0.
- **Acoustiness** — A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic.
- **Danceability** — Danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is the most danceable.
- **Energy** — Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy.
- **Instrumentalness** — Predicts whether a track contains no vocals. The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content.
- **Loudness** — The overall loudness of a track in decibels (dB). Values typical range between -60 and 0 dB.
- **Valence** — A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive.
- **Tempo** — The overall estimated tempo of a track in beats per minute (BPM).
- **Popularity** — The popularity of the track. The value will be between 0 and 100, with 100 being the most popular.

Exploratory Data Analysis

A. Data preprocessing

After extracting the data we will check how the data is and whether it is usable or not

2.Load Datasets

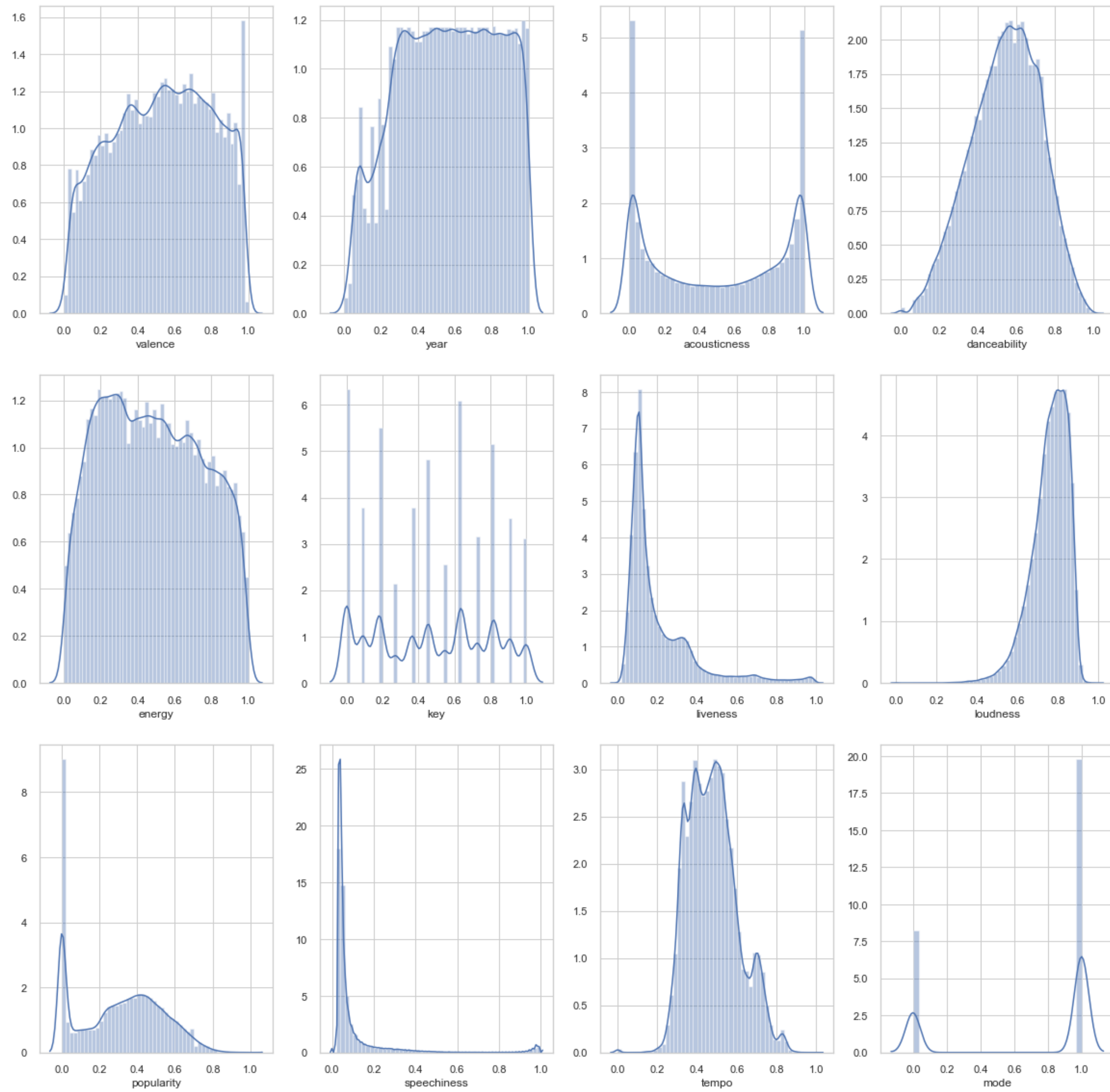
```
#Load all 5 Datasets

df = pd.read_csv('dhttps://raw.githubusercontent.com/Sairaghav1999/Dataset-final/master/data.csv?token=AG3SCJZUVTWCA5LAXB423JTBYZVMG')

df_artist = pd.read_csv('https://raw.githubusercontent.com/Sairaghav1999/Dataset-final/master/data_by_artist.csv?token=AG3SCJZBSBGDN7GBP6ZQUJ3BYZVNK')

df_by_genres = pd.read_csv('https://raw.githubusercontent.com/Sairaghav1999/Dataset-final/master/data_by_genres.csv?token=AG3SCJ5IHBC2V25UN744HOLBYZVOO')

df_year = pd.read_csv('https://raw.githubusercontent.com/Sairaghav1999/Dataset-final/master/data_by_year.csv?token=AG3SCJZYHCVQB7QFO2EZ5LDBYZVPO')

df_w_genres = pd.read_csv('https://raw.githubusercontent.com/Sairaghav1999/Dataset-final/master/data_w_genres.csv?token=AG3SCJ2XKYBS6OZF7DD2HB3BYZVQO')
```

```
[ ] df.head()
```

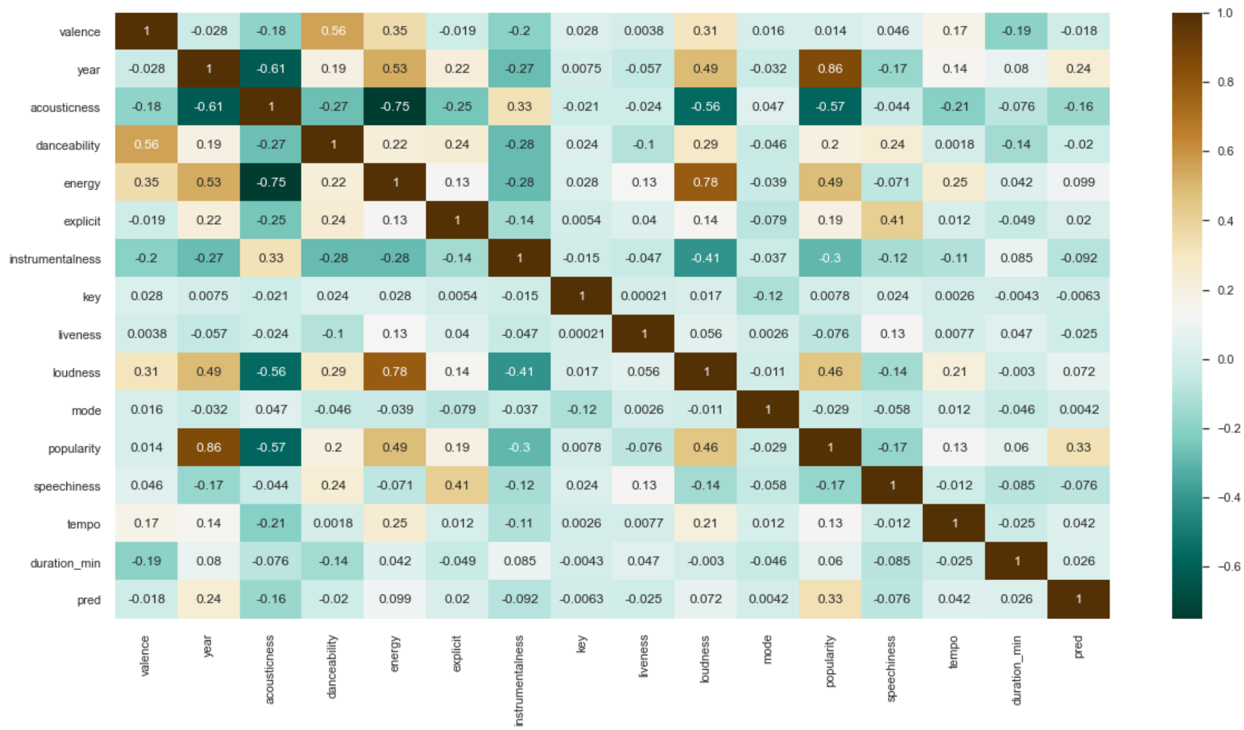
	valence	year	acousticness	artists	danceability	duration_ms	energy	explicit	id	instrumentalness	key	liveness	loudness	mode	name	popularity	release_date	speec
0	0.0594	1921	0.982	['Sergei Rachmaninoff', 'James Levine', 'Berli...	0.279	831667	0.211	0	4BJqT0PrAfrxzMOxytFOIz	0.878000	10	0.665	-20.096	1	Piano Concerto No. 3 in D Minor, Op. 30: III. ...	4	1921	
1	0.9630	1921	0.732	['Dennis Day']	0.819	180533	0.341	0	7xPhfUan2yNtyFG0cUWkt8	0.000000	7	0.160	-12.441	1	Clancy Lowered the Boom	5	1921	
2	0.0394	1921	0.961	['KHP Kridhamardawa Karaton Ngayogyakarta Hadi...	0.328	500062	0.166	0	1o6I8BglA6ylDMrIELygv1	0.913000	3	0.101	-14.850	1	Gati Bali	5	1921	
3	0.1650	1921	0.967	['Frank Parker']	0.275	210000	0.309	0	3ftBPsC5vPBKxYSee08FDH	0.000028	5	0.381	-9.316	1	Danny Boy	3	1921	
4	0.2530	1921	0.957	['Phil Regan']	0.418	166693	0.193	0	4d6HGyGT8e121BsdKmw9v6	0.000002	3	0.229	-10.096	1	When Irish Eyes Are Smiling	2	1921	

After exploring the dataset we can see that there are no duplicates but again it is due to the unique id. What we can do next is drop this cclumn to have a better picture about the dataset then we observe that there are 565 duplicates which are later dropped.

B. Dataset exploration



After all the features are extracted we observe the variability of each in the dataset.



We can have a couple of takeaways from the Pearsons correlation matrix we have.

1) Evident from the data popularity shows high correlation with the year its released. It shows evidence to the popularity metric not just going by no of streams but also how latest they were played.

2) If we see Energy it seems to affect the popularity. Majority of the songs are energetic but again we cannot conclude that they affect dance songs. The correlation isn't high enough.

3) Acousticness shows very low correlation with popularity. As going by recent trends it is dominated by electronic dance music and related genres. Especially in pop unless its a classic song its very rare that when an acoustic band releases a song chances of that going viral are low

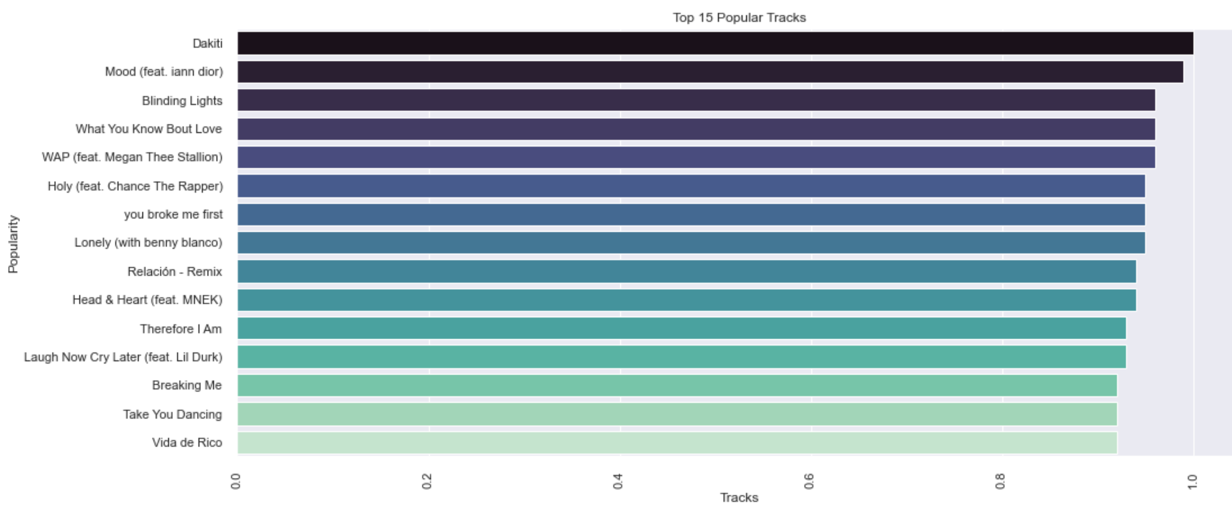
4) Loudnesws and energy show very high correlation. As the obvious energy and correlation should definelty show correlation.

5) Acoustincess has very low in fact negative correlation with loudness, energy and year.

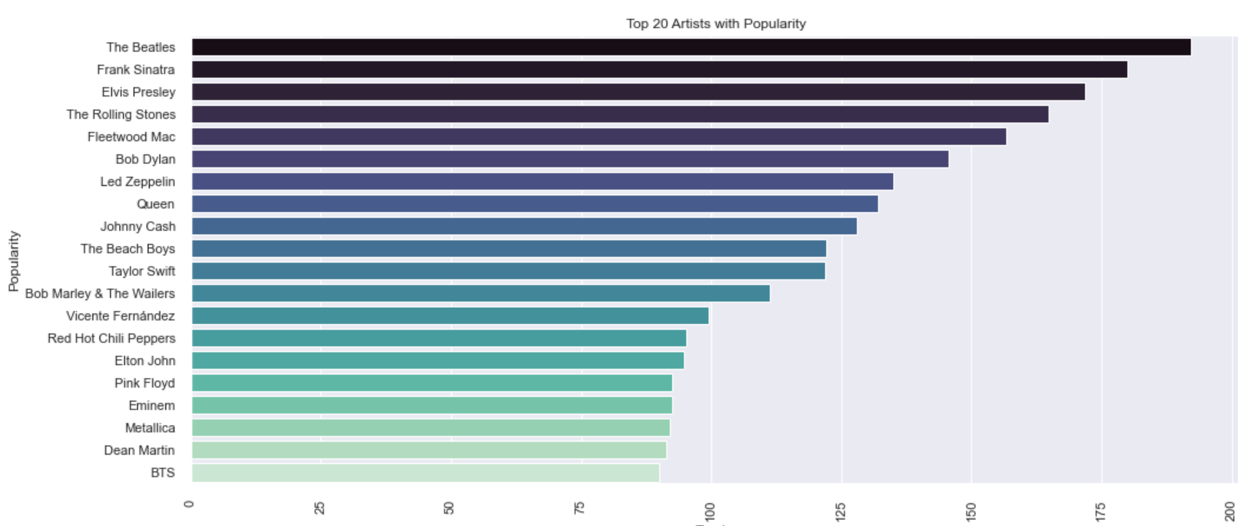
6) Danceability and valence are very highly correlated as dance songs are more on the happy note.

Hence after seeing this data we can conclude that an artist should create songs which has high energy involving EDM to go more viral.

C. Popular Tracks.

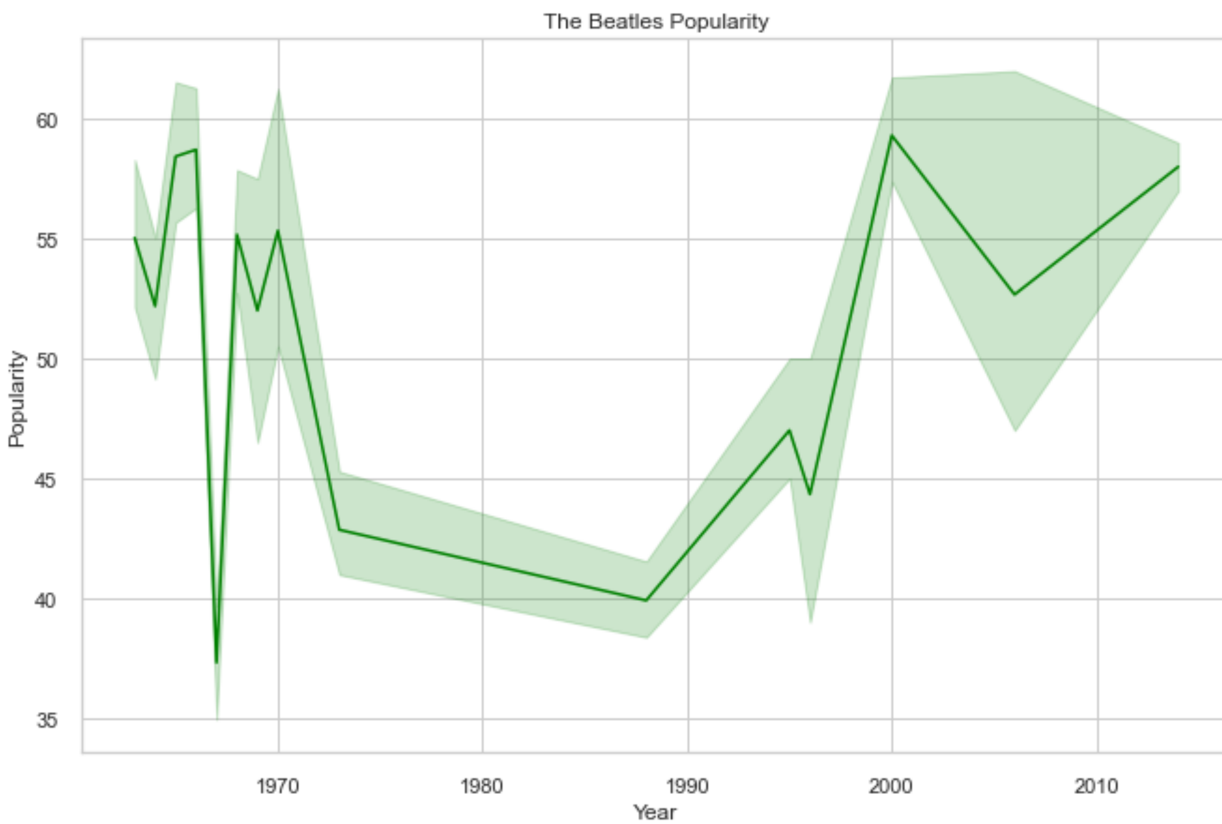


The highest rating was received by Dakiti by this visualization.

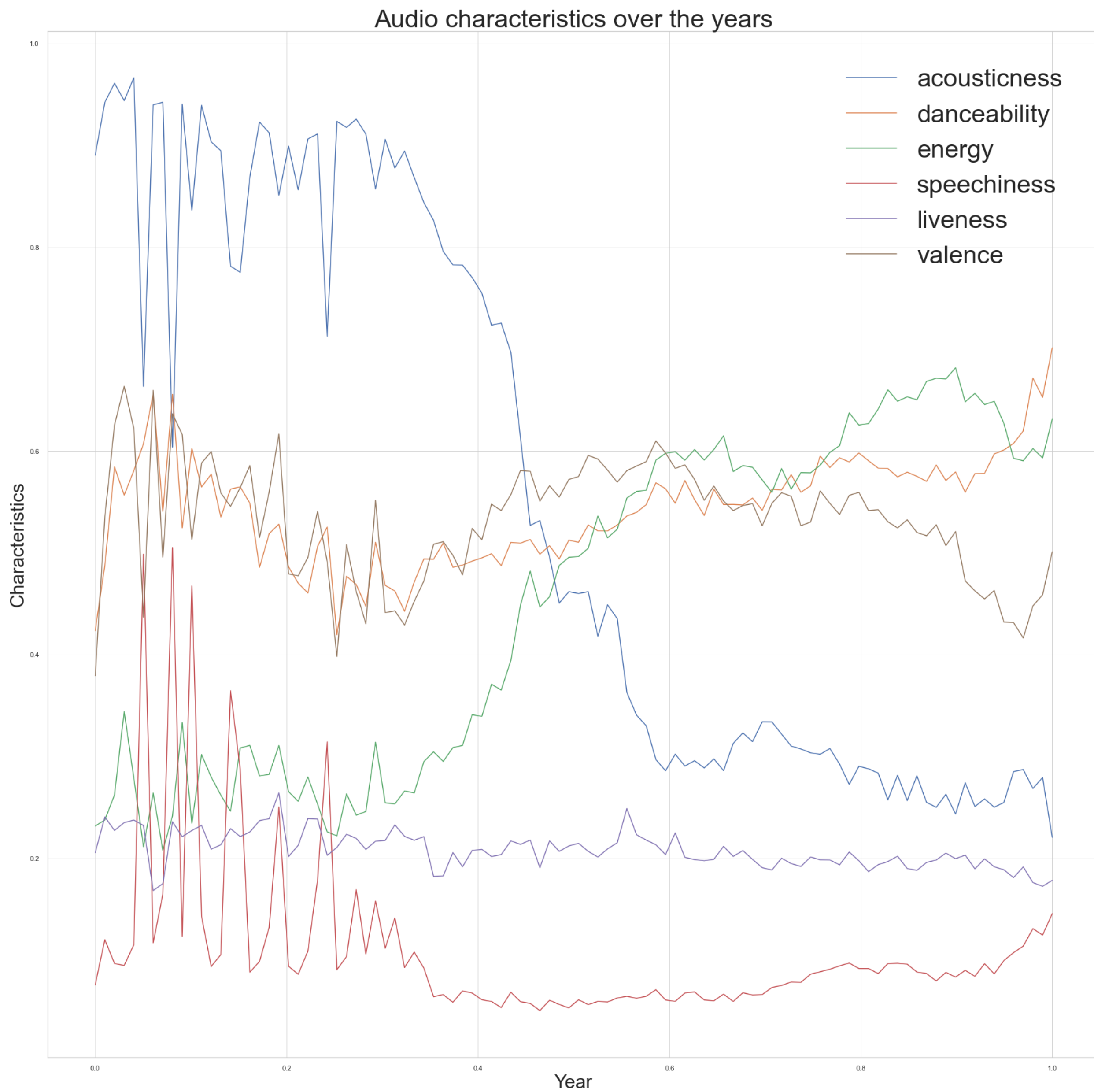


D. Most popular artists.

The most popular artist is Beatles from 1921-2020



E. Audio Features across all years



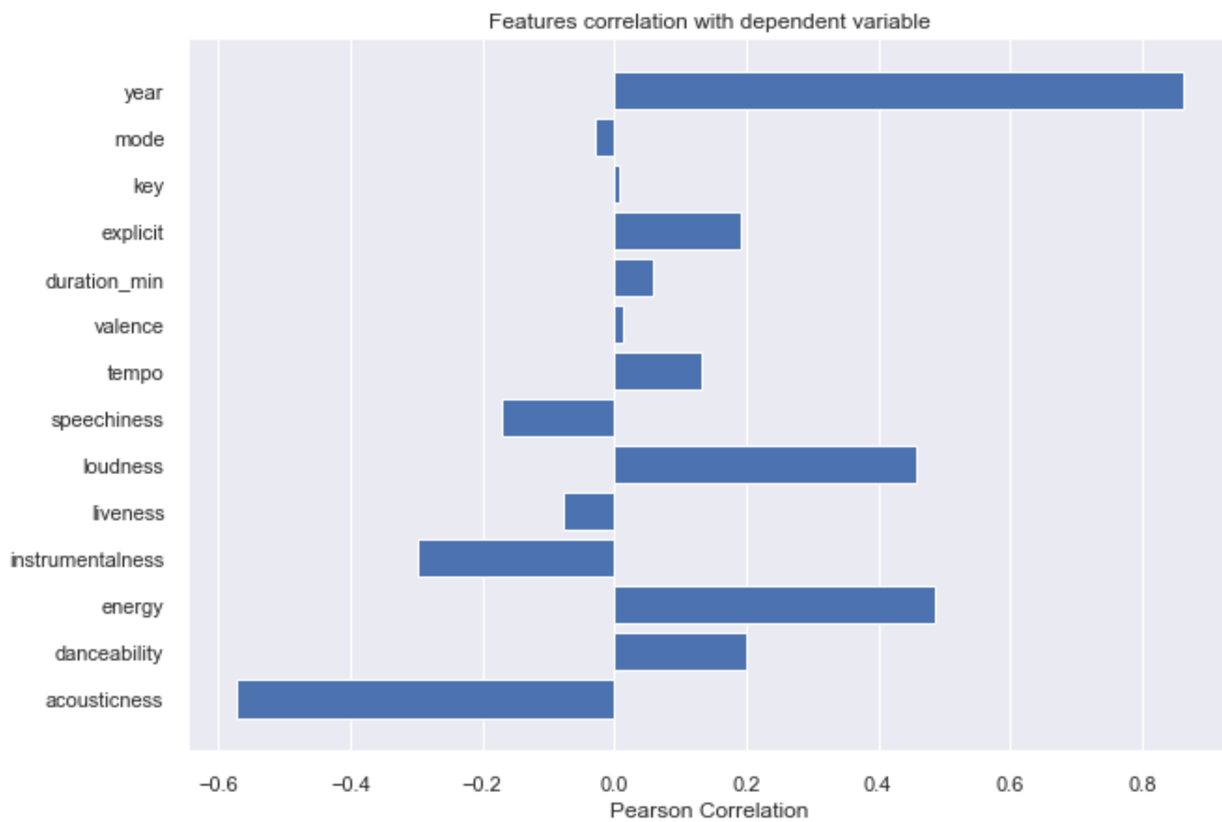
1) Pointed out earlier acousticness has come down drastically. As we move past 1960 electric has been used and especially post 1980 electronic sounds. The latest songs which go viral have either of those elements.

2) Danceability has been more and less the same since 1980

3) The interesting metric we can observe is how inversely proportional energy and acoustincess are at every decade it kept going up as acousticness kept going down.

Modelling

A) Feature Selection



The graph shows there are 9 features show positive correlation and the other 5 show negative correlation.

1) Id won't be of any help in our model so we would be dropping that accordingly

2) We can drop the release_date as dealing with year seems more appropriate given we have analyzed according to year.

3) Given we have 1 lakh unique names it will get complex we can drop name.

B) Feature Transformation

1) Eliminate null values and replace
2) Standardize instrumental data with numeric values
3) Use OneHotEncoder
4) Target scaling for popularity
5) MinMax scaling

c) Model Building

We split the data by 80:20 ratio for training and test dataset. We make use of Random forest, decision tree regressor with Grid Search CV and Decision TreeRegressor.

We will aim to fit these models and train our data and once test the accuracy of the fit.

i) Decision Tree Regressor

We

Decision Tree Regressor Model

```
def Decision_tree(X_train,y_train,X_test,y_test,min_samples_split,max_leaf_nodes):
    tree = DecisionTreeRegressor(max_leaf_nodes =max_leaf_nodes , min_samples_split =min_samples_split )
    tree.fit(X_train, y_train)
    y_train_pred = tree.predict(X_train)
    train_rmse = np.sqrt(mse(y_train, y_train_pred))

    y_test_pred = tree.predict(X_test)
    test_rmse = np.sqrt(mse(y_test, y_test_pred))

    r2_train = r2_score(y_train, y_train_pred)

    r2_test= r2_score(y_test, y_test_pred)

    mae = (abs(y_test - y_test_pred)).mean()

    return train_rmse,test_rmse,r2_train,r2_test,y_train_pred,y_test_pred,mae

train_rmse, test_rmse, r2_train, r2_test,y_train_pred,y_test_pred,mae= Decision_tree(X_train,y_train,X_test,y_test,min_samples_split = 200,max_leaf_nodes=167)

print("Root Mean Squared Error for Train dataset is {}".format(train_rmse))
print("Root Mean Squared Error for Test  dataset is {}".format(test_rmse))
print("r2-score for Train Dataset is {}".format(r2_train))
print("r2-score for Test Dataset is {}".format(r2_test))
print("Mean Absolute Error for Test dataset is {}".format(mae))
```

```
Root Mean Squared Error for Train dataset is 0.08769988801533651
Root Mean Squared Error for Test  dataset is 0.1088464644237765
r2-score for Train Dataset is 0.8381548180085765
r2-score for Test Dataset is 0.7489620916837287
Mean Absolute Error for Test dataset is 0.07926579723971713
```

hyperparameter tune the model using GridSearchCV to predict and improve accuracy.

```
n_features = df.shape[1]
n_samples = df.shape[0]

grid = GridSearchCV(DecisionTreeRegressor(random_state=0), cv=3, n_jobs=-1, verbose=5,
                    param_grid ={
                    'max_depth': [None,5,6,7,8,9,10,11],
                    'max_features': [None, 'sqrt', 'auto', 'log2', 0.3,0.5,0.7, n_features//2, n_features//3, ],
                    'min_samples_split': [2,0.3,0.5, n_samples//2, n_samples//3, n_samples//5],
                    'min_samples_leaf':[1, 0.3,0.5, n_samples//2, n_samples//3, n_samples//5]},
                    )

grid.fit(X_train, y_train)
print('Train R^2 Score : %.3f'%grid.best_estimator_.score(X_train, y_train))
print('Test R^2 Score : %.3f'%grid.best_estimator_.score(X_test,y_test))
print('Best R^2 Score Through Grid Search : %.3f'%grid.best_score_)
print('Best Parameters : ',grid.best_params_)
```

```
Fitting 3 folds for each of 2592 candidates, totalling 7776 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  48 tasks      | elapsed:    4.1s
[Parallel(n_jobs=-1)]: Done 138 tasks      | elapsed:    6.1s
[Parallel(n_jobs=-1)]: Done 264 tasks      | elapsed:    9.1s
[Parallel(n_jobs=-1)]: Done 426 tasks      | elapsed:   12.2s
[Parallel(n_jobs=-1)]: Done 624 tasks      | elapsed:   16.1s
[Parallel(n_jobs=-1)]: Done 858 tasks      | elapsed:   21.5s
[Parallel(n_jobs=-1)]: Done 1128 tasks      | elapsed:   27.4s
[Parallel(n_jobs=-1)]: Done 1434 tasks      | elapsed:   34.0s
[Parallel(n_jobs=-1)]: Done 1776 tasks      | elapsed:   41.3s
[Parallel(n_jobs=-1)]: Done 2154 tasks      | elapsed:   49.2s
[Parallel(n_jobs=-1)]: Done 2568 tasks      | elapsed:   58.0s
[Parallel(n_jobs=-1)]: Done 3018 tasks      | elapsed:  1.2min
[Parallel(n_jobs=-1)]: Done 3504 tasks      | elapsed:  1.4min
[Parallel(n_jobs=-1)]: Done 4026 tasks      | elapsed:  1.6min
[Parallel(n_jobs=-1)]: Done 4584 tasks      | elapsed:  1.8min
[Parallel(n_jobs=-1)]: Done 5178 tasks      | elapsed:  2.0min
[Parallel(n_jobs=-1)]: Done 5808 tasks      | elapsed:  2.3min
[Parallel(n_jobs=-1)]: Done 6474 tasks      | elapsed:  2.6min
[Parallel(n_jobs=-1)]: Done 7176 tasks      | elapsed:  2.9min
[Parallel(n_jobs=-1)]: Done 7776 out of 7776 | elapsed:  3.1min finished
Train R^2 Score : 0.840
Test R^2 Score : 0.766
Best R^2 Score Through Grid Search : 0.829
Best Parameters :  {'max_depth': 9, 'max_features': None, 'min_samples_leaf': 1, 'min_samples_split': 2}
```

D) Evaluation

When we analyze fitting of models there are two factors which afffect that are r2 score and Mean Absolute Error. MAE measures errors between observations. The lower the model goes the better model will perform.

But in our case we will chose R2score as we are working with regressor model than classifier model.

The r2-score function computes R^2. It provides a measure of how well future samples are likely to be predicted by the model. The best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of y, disregarding the input features, would get an R^2 score of 0.0.

Result Parameters	Decision Tree Regressor	Decision Tree with Grid Search CV	Random Forest Regressor
R2-score	74.896	76.6	74.6873
Mean Absolute Error	0.0792	0.073	0.0758

From the above analysis Decision, Tree Regression Model along with Grid Search CV proved to have the most reliable results. In Comparison of Random Forest and Decision Tree Regressor models, the Random Forest model resulted in less accuracy

6) Song Recommendation System

We use neighborhood collaborative filtering with similarity metrics method as this is the most used in Spotify recommendation algorithm.

```
#Load the dataset
df = pd.read_csv('data/data.csv')

#Remove the Square Brackets from the artists

df["artists"]=df["artists"].str.replace("[", "")
df["artists"]=df["artists"].str.replace("]", "")
df["artists"]=df["artists"].str.replace("'", "")

def normalize_column(col):
    """
    col - column in the dataframe which needs to be normalized
    """
    max_d = df[col].max()
    min_d = df[col].min()
    df[col] = (df[col] - min_d)/(max_d - min_d)

#Normalize allnumerical columns so that min value is 0 and max value is 1
num_types = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
num = df.select_dtypes(include=num_types)

for col in num.columns:
    normalize_column(col)

#perform Kmeans CLustering
km = KMeans(n_clusters=25)
pred = km.fit_predict(num)
df['pred'] = pred
normalize_column('pred')

#Song Recommender
class Song_Recommender():

    def __init__(self, data):
        self.data_ = data

    #function which returns recommendations, we can also choose the amount of songs to be recommended
    def get_recommendations(self, song_name, n_top):
        distances = []
        #choosing the given song_name and dropping it from the data
        song = self.data_[(self.data_.name.str.lower() == song_name.lower())].head(1).values[0]
        rem_data = self.data_[self.data_.name.str.lower() != song_name.lower()]
        for r_song in tqdm(rem_data.values):
            dist = 0
            for col in np.arange(len(rem_data.columns)):
                #indeces of non-numerical columns(id,Release date,name,artists)
                if not col in [3,8,14,16]:
                    #calculating the manhettan distances for each numerical feature
                    dist = dist + np.absolute(float(song[col]) - float(r_song[col]))
            distances.append(dist)
        rem_data['distance'] = distances
        #sorting our data to be ascending by 'distance' feature
        rem_data = rem_data.sort_values('distance')
        columns = ['artists', 'name']
        return rem_data[columns][:n_top]


#Instantiate recommender class
recommender = Song_Recommender(df)

#Get recommendations 'Red Roses (feat. Landon Cube)' song
recommender.get_recommendations('Red Roses (feat. Landon Cube)', 5)
```

```
recommender.get_recommendations('dynamite', 10)
```

100%|██████████| 170646/170646 [00:12<00:00, 13666.21it/s]

	artists	name
36728	Bowling For Soup	Stacy's Mom
18017	One Direction	What Makes You Beautiful
54538	Billy Talent	Fallen Leaves
107011	Glee Cast	Loser Like Me (Glee Cast Version)
17818	Hot Chelle Rae	Tonight Tonight
91698	Paramore	Rose-Colored Boy
18192	Katy Perry	Part Of Me
37542	5 Seconds of Summer	She's Kinda Hot
54360	RBD, Anahí, Dulce María, Maite Perroni, Christ...	Aún Hay Algo
124309	Easton Corbin	A Girl Like You

Conclusion

1) We have successfully predicted the popularity and after analysis built a recommendation system

2) Around 2000 popular songs are generated in Spotify every year

3) Most famous artist was The Beatles

4) Using GridSearchCV we have achieved an accuracy of 76.6%

References

1) https://developer.spotify.com/documentation/web-api/reference/#/operations/search

2) https://www.univ.ai/post/spotify-recommendations

3) https://towardsdatascience.com/clustering-music-to-create-your-personal-playlists-on-spotify-using-python-and-k-means-a39c4158589a

4) https://medium.com/s/story/spotifys-discover-weekly-how-machine-learning-finds-your-new-music-19a41ab76efe

5) https://scikit-learn.org/stable/

6) https://medium.com/systems-ai/spotifys-machine-learning-algorithms-and-your-daily-mix-f49d97db4b16

7) https://blogs.cornell.edu/info2040/2019/09/15/strong-ties-and-spotifys-recommendation-algorithms/