An Applied Data Science MITA Capstone Project
on

Predicting Song Success

Music Recommender System

Abstract

Predicting the success of the songs and building recommendation model using data available from Spotify website.

Authored By:
Aishwary Gawande

Under Guidance of:
Prof. Michail Xyntarakis

Rutgers Business School
Masters In Information Technology & Analytics

ABSTRACT

Predicting song popularity based on song features and listening behavior

is of growing importance for streaming services and for music

producers. In this study, we will apply and compare many supervised

learning methods, including logistic regression, ridge and lasso

regression, KNN, SVM, and Naïve Bayes to predict music popularity.

High dimensional visualization techniques will be used to visualize the

data. A thorough literature review will be conducted at the beginning of

the study to identify and document the state-of-the-art in music

recommendation. One of the datasets we will explore is made available

by Spotifylabs contains 19,000 songs and 15 features that past research

has used in similar studies. The features that we will explore include

audio mode, time signature, acousticness, danceability, energy,

instrumentalness, loudness, speechiness, audio valence, tempo, liveness,

and song popularity.

ACKNOWLEDGEMENTS

This Capstone Project has been successfully completed thanks to the guidance and supervision of Professor Michail Xyntarakis. I really appreciate the assistance and mentor ship he has provided.

I would also like to thank the Master of Information Technology and Analytics Program, Management Science and Information Systems Department and all of my previous course professors and colleagues at Rutgers Business School.

My academic, professional, and personal experiences during my time in the Master's program has been enriching and delightful. I hope to apply the technical, theoretical, social and practical components acquired during my graduate study towards my future professional or academic endeavors.

INTRODUCTION

Every year, number of individual of group of artists launch there music in the form of new song, album or remixed version on various platforms such as Spotify. Success of the song depends on various attributes, understanding & find the mechanism which helps to determine the factors which are responsible for the popularity of song is a good information to have both for artist and the record labels.

Currently, being able to predict that something might be popular beforehand is an important research subject for every industry. It also has recently became a very important subject for the growing and competitive music industry as well. Since wide use of digital music platforms (Spotify, Billboard, Lastfm), data can be easily reached and the listening behaviors of the listeners can be easily observed. This provides convenience in forecasting techniques and it is also frequently used in recommendation systems.

Every song has number of attributes, by studying & understanding this attributes with the help of previously obtained data helps us to predict the behavior of that particular data and able to make predictions on that.

By using data science techniques & algorithms we will build recommender system which will help us to find the popularity of the music and answer some of the questions from the dataset.

DATASET

The Dataset is obtained from the https://research.atspotify.com/datasets/ which is futher split into 19000 songs file, this file contains multiple attributes. The dataset is ideal as it contains songs which are more popular and consisting songs from multiple genres thus dataset is not biased.

It contains 18,835 rows and 15 collumns such as song_name, song_popularity, song_duration_ms, acousticness, danceability, energy, instrumentalness, key, loudness, audio_mode, speechiness, tempo, time_signature, audio_valence, liveness.

FEATURE EXPLANATION

Features	**Feature explanation**
duration_ms	The duration of the track in milliseconds.
key	The estimated overall key of the track. Integers map to pitches using standard Pitch Class notation . E.g. 0 = C, 1 = C♯/D♭, 2 = D, and so on. If no key was detected, the value is -1.
audio_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.
time_signature	An estimated overall time signature of a track. The time signature (meter) is a notational convention to specify how many beats are in each bar (or measure).
acousticness	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 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. For example, death metal has high energy, while a Bach prelude scores low on the scale.
instrumentalness	Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live. A value above 0.8 provides strong likelihood that the track is live.
loudness	The overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing relative loudness of tracks. Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude). Values typical range between -60 and 0 db.
speechiness	Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music.Values below 0.33 most likely represent music and other non-speech-like tracks.
audio_valence	A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).
tempo	The overall estimated tempo of a track in beats per minute (BPM). In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration.
song_popularity	Song ratings of spotify audience.
liveness	Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live.

Methedology

For any data science or machine learning project we need to follow sequence of steps which start with data cleaning or preparing the data for exploration. Data Cleaning involves cleaning of the duplicate values, null and invalid values.

Data Preprocessing and Cleaning

In this case data is already cleaned as it was provided by the Spotify. There are no null or missing values, hence we can use the data for further analysis.

Cleaning Data

```
In [8]: song_data.columns[song_data.isnull().any()]

Out[8]: Index([], dtype='object')

In [9]: song_data.isnull().sum()

Out[9]: song_name           0
        song_popularity     0
        song_duration_ms    0
        acousticness        0
        danceability        0
        energy              0
        instrumentalness    0
        key                 0
        liveness            0
        loudness            0
        audio_mode          0
        speechiness         0
        tempo               0
        time_signature      0
        audio_valence       0
        dtype: int64
```

Exploratory Data Analysis

Out of 12 attributes in our dataset, there are 5 integer attributes, 9 float values & 1 object value.

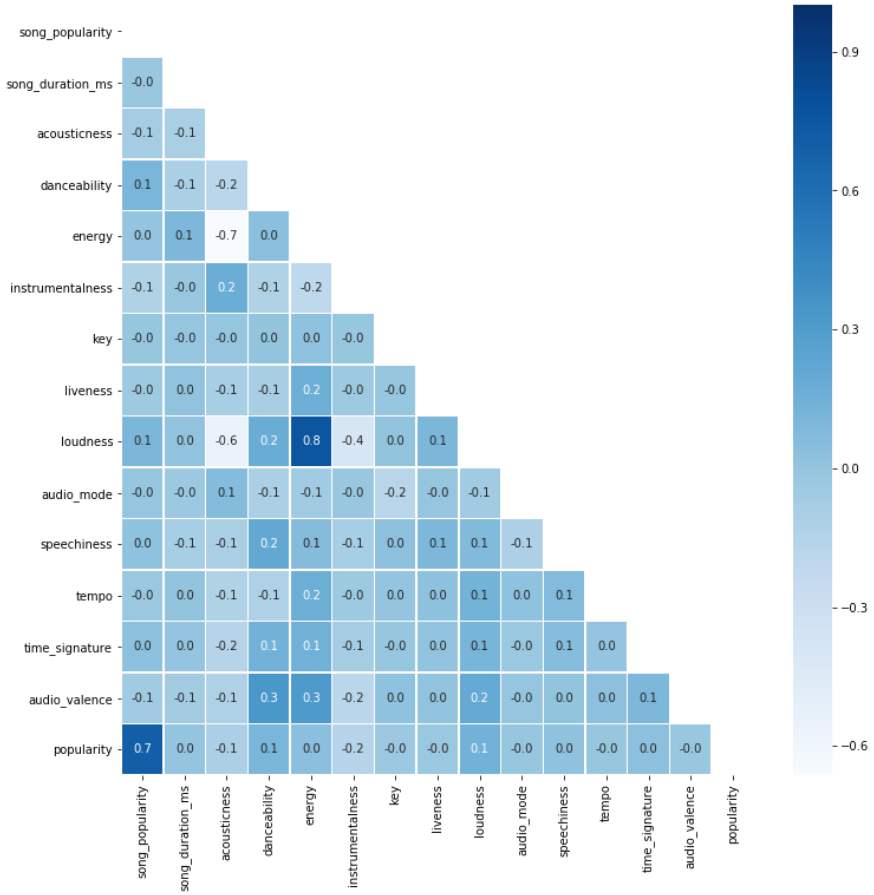
After looking at the whole dataset we got the following result,

	song_popularity	song_duration_ms	acousticness	danceability	energy	instrumentalness	key	liveness	loudness	audio_mode	speechiness	tempo	time_signature	audio_valence	popularity
count	5449.000000	5449.000000	5449.000000	5449.000000	5449.000000	5449.000000	5449.000000	5449.000000	5449.000000	5449.000000	5449.000000	5449.000000	5449.000000	5449.000000	5449.0
mean	76.992292	218539.555515	0.210354	0.659758	0.658601	0.022390	5.11782	0.174400	-6.624852	0.618829	0.106247	120.753878	3.974307	0.520436	1.0
std	8.068717	48620.048311	0.246079	0.147652	0.187495	0.115572	3.65752	0.137557	3.131934	0.485719	0.102181	27.737971	0.255771	0.230911	0.0
min	67.000000	67000.000000	0.000009	0.072200	0.000289	0.000000	0.00000	0.021500	-34.255000	0.000000	0.022400	57.178000	1.000000	0.035200	1.0
25%	71.000000	190185.000000	0.026300	0.562000	0.541000	0.000000	1.00000	0.092000	-7.906000	0.000000	0.039100	98.883000	4.000000	0.336000	1.0
50%	75.000000	212429.000000	0.106000	0.668000	0.680000	0.000000	5.00000	0.121000	-5.985000	1.000000	0.059100	119.886000	4.000000	0.511000	1.0
75%	82.000000	240533.000000	0.300000	0.765000	0.802000	0.000118	8.00000	0.203000	-4.626000	1.000000	0.131000	139.910000	4.000000	0.701000	1.0

	song_popularity	song_duration_ms	acousticness	danceability	energy	instrumentalness	key	liveness	loudness	audio_mode	speechiness	tempo	time_signature	audio_valence	popularity
max	100.000000	547733.000000	0.996000	0.978000	0.997000	0.968000	11.000000	0.978000	-0.739000	1.000000	0.733000	212.058000	5.000000	0.980000	1.0

Checked popularity rating of songs that have been popular in the last 10 years in Spotify and took the mean value of them (66.5) . According to this value, the songs has above this rating could remain on the top lists for a long time. If song_popularity is higher than 66.5 (this is about 30% percent of data) we labeled it "1" and if is not we labeled it "0". So we have "1" for the popular songs and "0" for the unpopular ones.

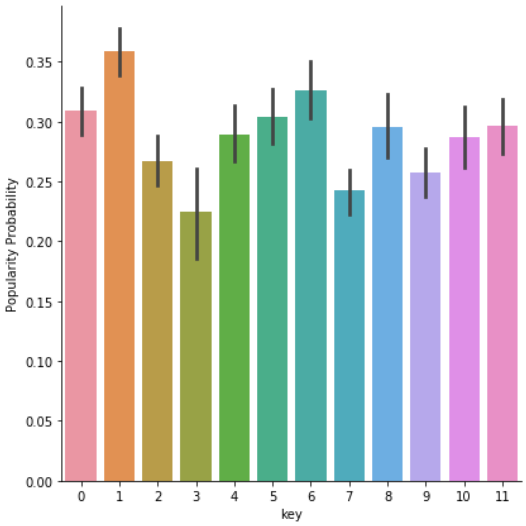
Correlation helps us to know, how closely the twovariables are connected to each other and there impact on each other. The graph below shows the correlation between all the attributes for this dataset.



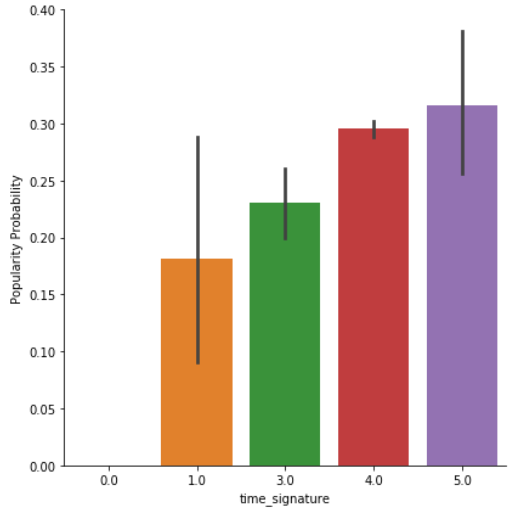
Correlation between loudness and energy is 0.8 which is strong and correlation between loudness and accusticness is 0.6 which is moderate. Except two of them all the correlations are quite low. When compare the correlation between song_popularity and all other features, we don't see a strong correlation (a linear relationship) that gives us a clear information about popularity. Accusticness,danceability and loudness seems to have correlation with popularity feature(0.10) and istrumentalness has 0.20.

Comparision of the variables with the independent variable will help us to understand the relation between them even in more depth.

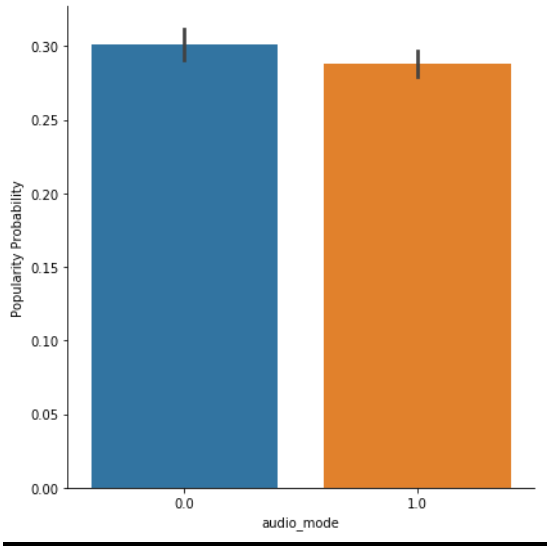
- **Popular Probablity vs Key**



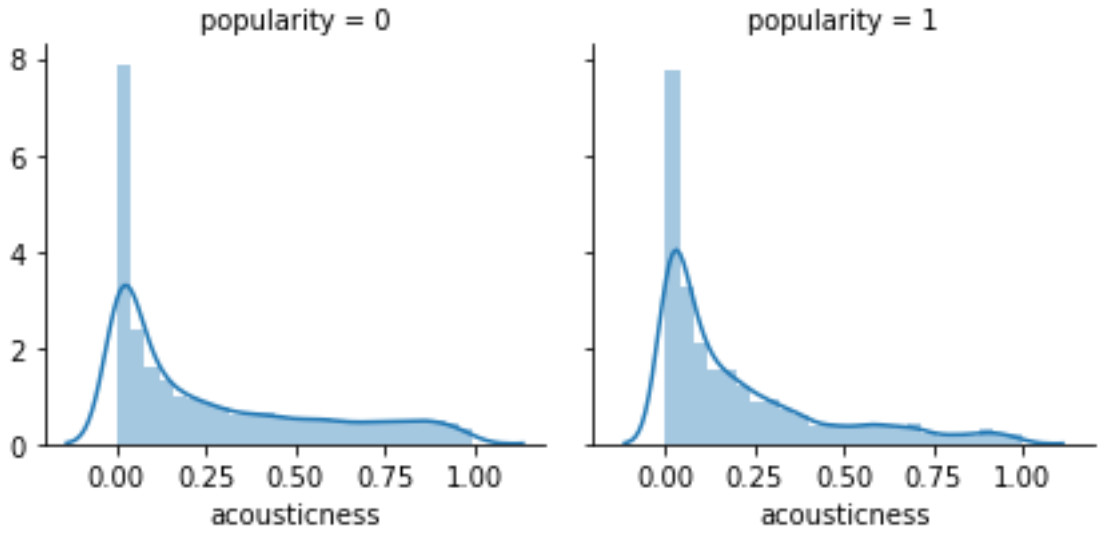
- **Popular Probablity vs Time_signature**



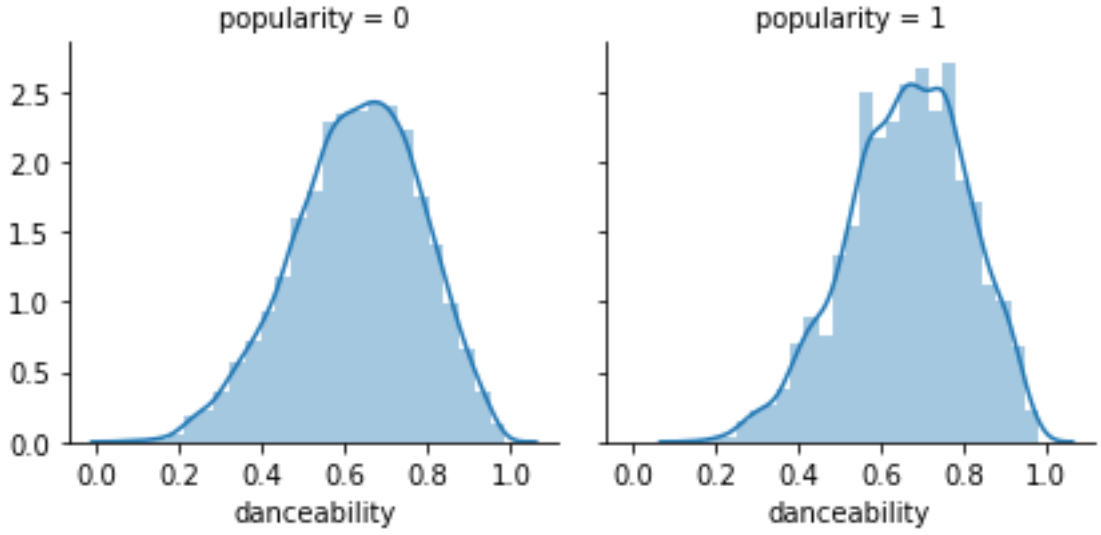
- **Popular Probablity vs Audio mode**



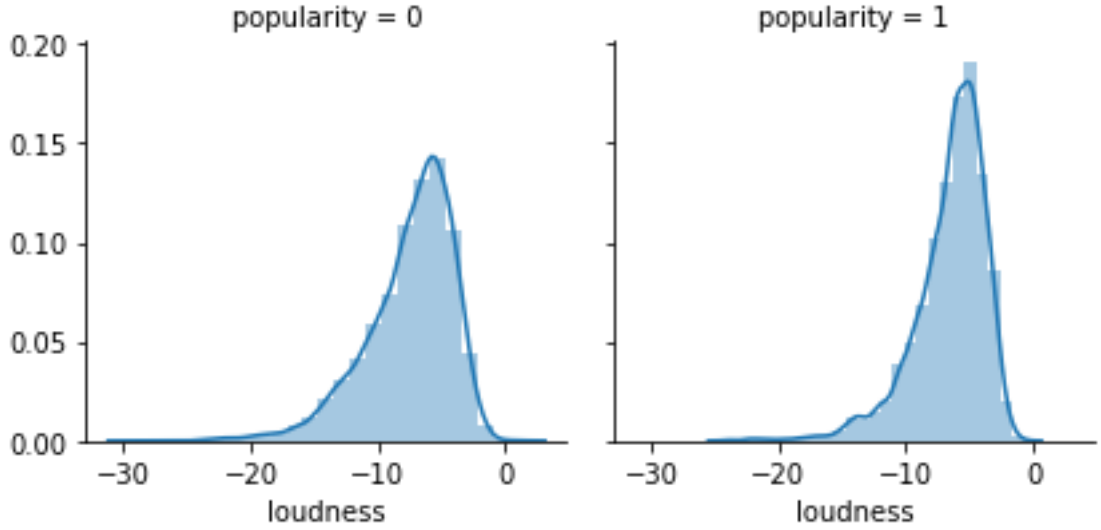
- **Popular Probablity vs acousticness**



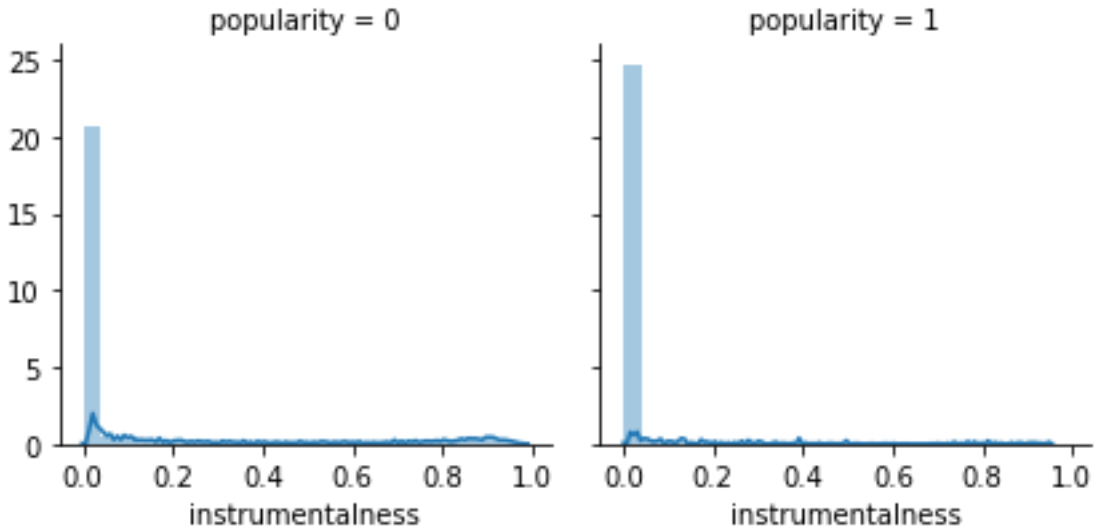
- **Popular Probablity vs danceability**



- **Popular Probablity vs loudness**



- **Popular Probablity vs instrumentalness**



- Data distribution of songs display today's songs features like dancebility, energy, loudness and tempo are quite high. People like fast and loud music.

- According to instrumentalness, liveness and speechness, most of the songs are not live performances and they have lyrics.

- Keys like 0,1,5,6 and 11 seems more effective in songs. And if key== 0 or 1 or 6 song has more chance to be populer.

- Time_signure is mostly 4 and 5 in both populer and general data.

- If danceability>0.6 song has more chance to be popular.

- If loudness > -10 song has more chance to be popular.

Let’s see wether the particular song is Happy or Sad. First we will look in the overall songs then in the list of popular songs and finally into the Top 500 lists.

All Songs

In [36]:
```
song_data3=song_data.copy()
song_data3["song_audio_valence"]= [ "Happy" if i>=0.5 else "Sad" for i in song_data.audio_valence ]
song_data3["song_audio_valence"].value_counts()
```

Out[36]:
```
Happy    10117
Sad       8718
Name: song_audio_valence, dtype: int64
```

Popular Songs

In [37]:
```
song_data1=song_data3[song_data3["song_popularity"]>66.5]
song_data1["song_audio_valence"]= [ "Happy" if i>=0.5 else "Sad" for i in song_data1.audio_valence ]
song_data1["song_audio_valence"].value_counts()
```

```
/Users/ash/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning
-a-view-versus-a-copy
```

Out[37]:
```
Happy    2849
Sad      2600
Name: song_audio_valence, dtype: int64
```

Top 500

In [38]:
```
song_data2_new=song_data1[song_data1["song_popularity"]>90]
song_data2_new["song_audio_valence"]= [ "Happy" if i>=0.5 else "Sad" for i in song_data2_new.audio_valence ]
song_data2_new["song_audio_valence"].value_counts()
```

```
/Users/ash/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning
-a-view-versus-a-copy
```

Out[38]:
```
Sad      332
Happy    167
Name: song_audio_valence, dtype: int64
```

As seen in the feature explanations, audio valance describes the musical positiveness conveyed by a track (like sad or happiness - between 0 to 1). lets say our threshold is 0.5. With this threshold,happy songs more in normal data and the numbers are pretty close in the popular songs, but when looked at top 500, it can be said that negative songs are twice as much as positives.

Splitting the data, Training/Testing

Training data is the data on which we are supposed to train and build the model and testing data is the pre-existing data through which we verify our training data. Data set is split in 80-20 i.e. 80% of it is training data and 20% of it is testing data. As we go forward we tried splitting training data into training and validation (80/20 split) and again took the sub sample of it to train the model. Doing such way we find the mean performance of all the small splits we have make to compute the accuracy of the model.

In [47]:
```
#data preparation
y = song_data["popularity"].values
x_data=song_data.drop(["popularity"],axis=1)
#normalization
x = (x_data - np.min(x_data))/(np.max(x_data)-np.min(x_data)).values#train test split
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.2,random_state=42)
x_train = x_train.T
x_test = x_test.T
y_train = y_train.astype(int).T
y_test = y_test.astype(int).T
print("x_train: ",x_train.shape)
print("x_test: ",x_test.shape)
print("y_train: ",y_train.shape)
print("y_test: ",y_test.shape)
```

```
x_train:  (13, 15068)
x_test:  (13, 3767)
y_train:  (15068,)
y_test:  (3767,)
```

Proposed Algorithms:

Logistic Regression

LR is one of the basic classification method is used prediction of categorical variables.Dataset problem has two possible outputs popular(1) and unpopular(0) which is suitable for binary logistic regression. Since it is a probability value that we want to get from the problem, we obtained a value between [0,1] using the sigmoid function.

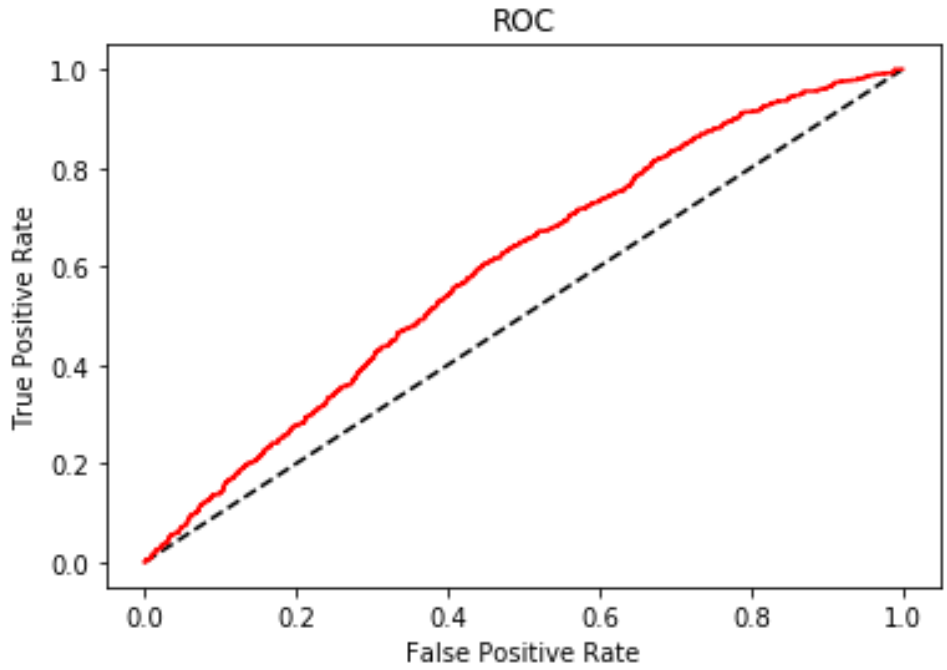
```
Optimization terminated successfully.
         Current function value: 0.571328
         Iterations 7
                          Results: Logit
================================================================
Model:              Logit            Pseudo R-squared: 0.050
Dependent Variable: y                AIC:              21547.9388
Date:               2020-11-30 15:46 BIC:              21649.9039
No. Observations:   18835            Log-Likelihood:   -10761.
Df Model:           12               LL-Null:          -11330.
Df Residuals:       18822            LLR p-value:      4.8928e-236
Converged:          1.0000           Scale:            1.0000
No. Iterations:     7.0000
----------------------------------------------------------------
                   Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
----------------------------------------------------------------
song_duration_ms  -0.6369   0.5345  -1.1915 0.2335 -1.6846  0.4108
acousticness      -1.0382   0.0800 -12.9784 0.0000 -1.1950 -0.8814
danceability       0.9986   0.1257   7.9475 0.0000  0.7524  1.2449
energy            -1.1926   0.1492  -7.9954 0.0000 -1.4850 -0.9003
instrumentalness  -2.4742   0.1398 -17.6932 0.0000 -2.7482 -2.2001
key               -0.2635   0.0507  -5.1923 0.0000 -0.3630 -0.1640
liveness          -0.3720   0.1178  -3.1571 0.0016 -0.6029 -0.1410
loudness           2.4513   0.2697   9.0894 0.0000  1.9228  2.9799
audio_mode        -0.0653   0.0347  -1.8800 0.0601 -0.1333  0.0028
speechiness       -0.2562   0.1570  -1.6314 0.1028 -0.5639  0.0516
tempo             -0.4719   0.1429  -3.3017 0.0010 -0.7520 -0.1918
time_signature    -1.7957   0.2205  -8.1431 0.0000 -2.2279 -1.3635
audio_valence     -0.5981   0.0791  -7.5639 0.0000 -0.7531 -0.4432
================================================================
```

AIC and BIC values which are known as Additive and multiplicative Information criteria and Bayesian criterion are used to get the score and shows which particular attributes give us a good fit for the model. The Lower the AIC & BIC score is the better fit for the model. The

above function graph shows that after the 7 iterations it has given us the lowest score and best possible fit.

The p-value gives us the probability that we falsely rejects the null hypothesis and z-value indicates the standard deviation of each parameter. Instrumentalness and acousticness is widely spreaded.

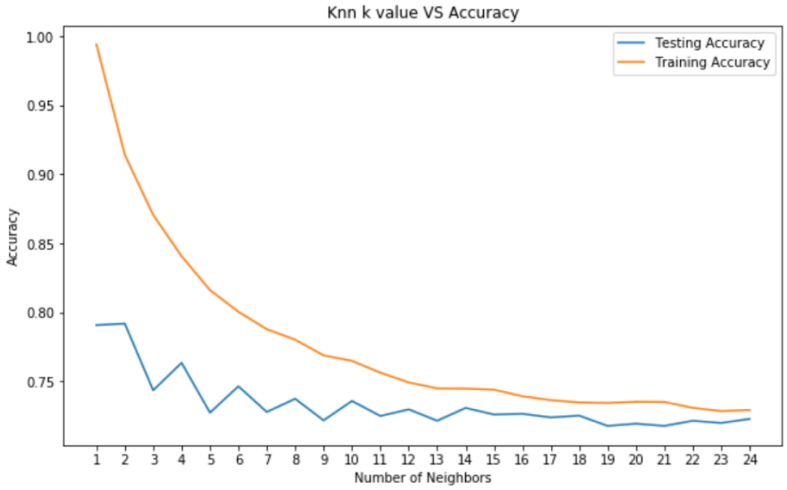
ROC with Logistic Curve



ROC i.e. receiver operator characteristic in logistic regression is used to determine cut off value to identify wether the new value is good or not. We have taken cut-off 0.5. Depending upon our need of specificity and sensitivity we will adjust the threshold.

K-nearest neighbours Algorithm

KNN algorithm works on the idea that things/points exists in close proximity. It hinges on this concept to make the algorithm come true. It works by calculating the distance between two or more points. We set the k (i.e. no of neighbours) value. As, we decrease the value of the k our predictions become less stable and as we increase the value of the k our assumption become more stronger. Disadvantage with KNN is that as the no of independent variables increases the algorithm gets slower. As, we can see from the above graph that we have got the best accuracy with k = 2 i.e. 79.18%.




```
Best accuracy is 0.7918768250597292 with K = 2
```

Naïve Bayes

Gaussian Naïve Bayes Algorithm in machine learning is based on Baye's theorem with strong independent connection between the parameters. They are the family of probabilistic classifiers. We have got a training accuracy of 70% and testing accuracy of 71%.

Naive Bayes

In [62]:
```
from sklearn.naive_bayes import GaussianNB
nb= GaussianNB()
nb.fit(x_train,y_train)
print("Train accuracy of naive bayes:",nb.score(x_train,y_train))
print("Test accuracy of naive bayes:",nb.score(x_test,y_test))
```

```
Train accuracy of naive bayes: 0.7090522962569684
Test accuracy of naive bayes: 0.7172816564905761
```

In [63]:
```
Naive_bayes_score=nb.score(x_test,y_test)
```

Decision Tree Classifier

Decision Tree algorithm which uses tree like graph or various decision models with there possible outcomes including utility outcomes and resources costs. It is one of the way which displays that the following algorithm contains conditional statement. We have got a training accuracy of 99.5% and testing accuracy of 80.14%. After performing k-fold cross validation considering k=10 we got an average of 79.33% which is almost close enough to the testing accuracy.

Decision Tree Classifier

In [64]:
```
from sklearn.metrics import accuracy_score,recall_score,precision_score,confusion_matrix,f1_score
from sklearn.tree import DecisionTreeClassifier

dt= DecisionTreeClassifier()
dt.fit(x_train,y_train)
y_pred=dt.predict(x_test)
DecisionTree_score=dt.score(x_test,y_test)
print("Train ccuracy of decision tree:",dt.score(x_train,y_train))
print("Test accuracy of decision tree:",dt.score(x_test,y_test))
```

```
Train ccuracy of decision tree: 0.9951552959915052
Test accuracy of decision tree: 0.8014335014600478
```

In [65]:
```
from sklearn.model_selection import cross_val_score
k =10
cv_result = cross_val_score(dt,x_train,y_train,cv=k) # uses R^2 as score
print('Cross_val Scores: ',cv_result)
print('Cross_val scores average: ',np.sum(cv_result)/k)
```

```
Cross_val Scores:  [0.78832117 0.800929   0.80690113 0.79230259 0.78898474 0.78500332
 0.78699403 0.78964831 0.80278884 0.79150066]
Cross_val scores average:  0.7933373782022981
```

Random Forest Classifier

Random forest algorithm operates by constructing multiples of decision trees and outputting the mean predicition of each individual trees. It is correct for the decision tree way of overfitting the training set.

In [68]:
```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier(n_estimators=150,random_state = 3)
rf.fit(x_train,y_train)
print("Train ccuracy of random forest",rf.score(x_train,y_train))
print("Test accuracy of random forest",rf.score(x_test,y_test))
RandomForestClassifier_score=rf.score(x_test,y_test)
y_pred=rf.predict(x_test)
t_true=y_test
```

```
Train ccuracy of random forest 0.9951552959915052
Test accuracy of random forest 0.8863817361295461
```

In [69]:
```
from sklearn.model_selection import cross_val_score
k = 10
cv_result = cross_val_score(rf,x_train,y_train,cv=k) # uses R^2 as score
cv_result_randomforest=np.sum(cv_result)/k
print('Cross_val Scores: ',cv_result)
print('Cross_val scores average: ',np.sum(cv_result)/k)
```

```
Cross_val Scores:  [0.88984738 0.8739217  0.88254811 0.86794957 0.86529529 0.86861314
 0.87193099 0.8805574  0.88446215 0.88247012]
Cross_val scores average:  0.876759584092297
```

RF is one of the most popular ensemble learning method in machine learning not only gives good results even without hyperparameter optimization but also can use both classification and regression problems. But the common problem of the traditional decision trees is over-fitting. In order to avoid overfitting, random forest models select and train hundreds of different sub-samples (multiple deep decision trees) randomly and reduce the variance. We used 100 estimators and random state 3 gave the about 89% accuracy.

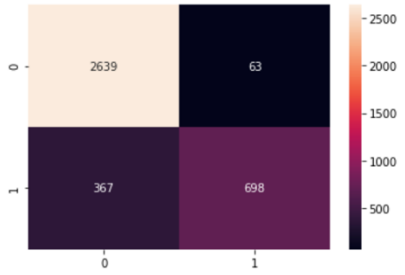
Confusion Matrix

Confusion Matrix with Random Forest

In [70]:
```
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(random_state = 4)
rf.fit(x_train,y_train)
y_pred = rf.predict(x_test)
cm = confusion_matrix(y_test,y_pred)
print('Confusion matrix: \n',cm)
print('Classification report: \n',classification_report(y_test,y_pred))
```

```
Confusion matrix:
 [[2639   63]
 [ 367  698]]
Classification report:
              precision    recall  f1-score   support

           0       0.88      0.98      0.92      2702
           1       0.92      0.66      0.76      1065

    accuracy                           0.89      3767
   macro avg       0.90      0.82      0.84      3767
weighted avg       0.89      0.89      0.88      3767
```

In [71]:
```
sns.heatmap(cm,annot=True,fmt="d")
plt.show()
```



Since we set our threshold such that 70% of the songs in our dataset are named as not popular and 30% are named as popular, 70% accuracy can be achieved by predicting all 0s. In order to capture this, we should consider the precision and recall values. As we see in confusion matrix, model predicted “695 popular” songs and “2596 unpopular” songs correct.

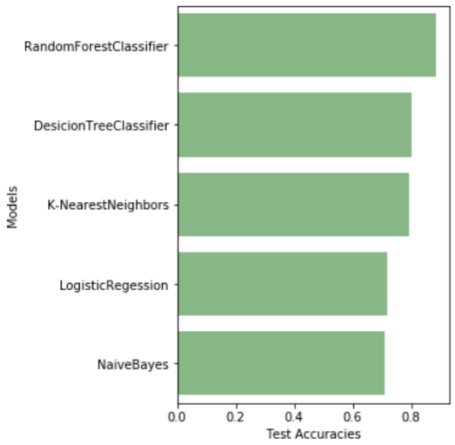
Comparision of Perfomance

Out of all the models that we applied, Random forest classifier gave us the highest accuracy with 0.88 second is the decision tree classifier with 0.80. K-nearest neighbors gave the accuracy of 0.79. Naives bayes and logistic regression with 0.71 and 0.70.

Out[74]:

	Model	Accuracy
0	RandomForestClassifier	0.886382
1	DesicionTreeClassifier	0.801434
2	K-NearestNeighbors	0.791877
4	NaiveBayes	0.717282
3	LogisticRegession	0.709052

In [75]:
```
model_list= list(model_performances['Model'].unique())
accuracy_list= list(model_performances['Accuracy'].sort_values(ascending=False))
f,ax = plt.subplots(figsize = (4,6))
sns.barplot(x=accuracy_list,y=model_list,color='green',alpha = 0.5)
ax.set(xlabel='Test Accuracies', ylabel='Models')
plt.show()
```



Conclusion

First I tried to predict popular songs using audio features then I added song name texts' polarity to it and tried to improve our model. We also fitted the model using each importance as a threshold. Although accousticness is the most important of these features did not lead us to a strong result.

I had 18835 songs available. Decision Tree algorithms which mainly given better results when we don't have so much data got the best result with RF. There were no strong linear correlations in the data, so linear methods did not fit well. In LR, I built the model using the gradient decent and this gave the best result in 200 iterations which was a moderate result. Adding Polarity to features value has almost not changed the result at all . For the future work, the following questions can be asked. Nowadays, the industrialization of popular music has became more common all over the world and most musicians are using computers when they create their songs. So are songs started to be more similar and less unique? Does these affect our predictions in a positive way? With industrialization, prediction of the popular and trendy items are getting easier? Could the prediction increase if we had data with all the songs are created by a computer?

References

- https://stackabuse.com/random-forest-algorithm-with-python-and-scikit-learn/

- https://www.datacamp.com/community/tutorials/random-forests-classifier-python

- www.spotify.com

- www.opendatascience.com/a-machine-learning-deep-dive-into-my-spotify-data/

- www.researchgate.net/post/Is_there_any_documentation_about_Spotifys_au
- dio_features

- https://www.reddit.com/r/dataisbeautiful/comments/88q0d1/songs_have_gotten_louder_over_time_oc/