**Introduction to Data Science - KNN**

**Songs and Their Impact on a Person’s Mood - Will Potter**

Introduction

**Research Objective**

**In this project, I am exploring songs that could potentially alter or change one’s mood if you are listening to the song. The list of songs used describes the songs based on the following criteria: Popularity, Length, Danceability, Acousticness, Energy, Instrumentalness, Liveness, Valence, Loudness, Speechiness, Tempo, Key, Time\_signature. Also the labeling of the song’s assumed influence on a person’ mood - Sad, Calm, Happy, Energetic.**

**The research objective is to use a kNN model trained on the survey dataset, to identify which classification a song potentially belongs to. The kNN model will measure similarity to classify the songs as a feeling of one’s mood. For example, if a person would like to relax and/or mediate, a person would want to load a music playlist that has a calming effect, or if you are creating a wedding playlist you would want to have songs that are either energetic and happy so that it can help enhance the celebration and provide great dance music. Musical preference is of course different for nearly everyone. To make these playlist to suit one’s or a group’s needs without bias, I will build a kNN model. Then, when the situation presents itself based on the four types of moods a person or group can use the model to assess their creation of a playlist that would compliment their mood.**

**Research Questions**

**1) Can we use a limited set of musical attributes: Energy, Danceability, and Loudness to predict the mood a song will elicit**

**2) Would it be possible to optimize predictability of a song’s mood by considering and comparing alternative distance algorithms - Manhattan, Euclidean, and Minkowski.**

**Data Source**

**The dataset lists of the songs contain String-based values detailing the song’s Title, Artist Name, and Date of Release as well as numerical statistics based on musical attributes. The data set was sourced from kaggle.com. The data set was chosen due to my fondness of music, in particular songs that put me in an energetic or happy mood. The dataset was selected for relevance to the topic. Original data source:** [**https://www.kaggle.com/musicblogger/spotify-music-data-to-identify-the-moods.**](https://www.kaggle.com/musicblogger/spotify-music-data-to-identify-the-moods)

**The dataset needed to be altered to be able to be run through the kNN model. In particular the String Values (Title, Artist Name, and Date of Release) were removed from the dataset. The target values - Sad, Calm, Happy, and Energetic - were categorical. Instead of using an OneHotEncoder to handle the categorical values, I altered and established the Targeted “Mood” values to be numerical values [0→3] : Sad(0), Calm(1), Happy(2), and Energetic (4).**

**Altered data source:** [Altered kNN DataMood - WPotter](https://docs.google.com/spreadsheets/d/1FbxoP_XkuUHyWWvOO1KbUy3KLhJDVs5W5GVgJyLfs-Y/edit?usp=sharing)

STEP 1 - Data Descriptive Statistics

In [29]:

*# Import required packages*

*import numpy as np*

*import pandas as pd*

*from sklearn.model\_selection import train\_test\_split*

*from sklearn.naive\_bayes import GaussianNB*

*from sklearn.metrics import accuracy\_score*

*import matplotlib.pyplot as plt*

*import seaborn as sns*

*from sklearn.datasets import load\_iris*

*from sklearn.model\_selection import train\_test\_split*

*from sklearn.naive\_bayes import GaussianNB*

*from sklearn.model\_selection import KFold*

*from sklearn.model\_selection import cross\_validate*

*from sklearn.metrics import classification\_report, confusion\_matrix*

In [30]:

*#read in dataset*

*# Filter warnings*

*import warnings*

*warnings.filterwarnings('ignore')*

*# Read the data, print the shape and first 5 rows.*

*dataMoodsData = pd.read\_csv('data\_moods3.csv')*

*print(dataMoodsData.shape)*

*print(dataMoodsData.head())*

*#Verification of Data*

**The data contained 685 data points with 14 numerical attributes**

*(685, 14)*

*popularity length danceability acousticness energy instrumentalness \*

*43 318800 0.381 0.01890 0.832 0.196000*

*68 379266 0.866 0.13700 0.730 0.000000*

*60 217946 0.346 0.91300 0.139 0.000077*

*2 233000 0.466 0.08900 0.438 0.000006*

*60 268000 0.419 0.00171 0.932 0.000000*

*liveness valence loudness speechiness tempo key time\_signature*

*0.1530 0.166 -5.069 0.0492 120.255 8 4*

*0.0843 0.625 -8.201 0.0767 118.523 5 4*

*0.0934 0.116 -15.326 0.0321 136.168 0 4*

*0.1130 0.587 -12.858 0.0608 193.100 4 4*

*0.1370 0.445 -3.604 0.1060 169.881 1 4*

*mood*

*0*

*2*

*0*

*2*

*3*

*# Print the summary of the data*

*print(dataMoodsData.describe())*

*print(dataMoodsData.describe().T)*

*popularity length danceability acousticness energy \*

*count 685.000000 685.000000 685.000000 685.000000 685.000000*

*mean 41.535766 221843.252555 0.500733 0.448954 0.508412*

*std 23.062538 63430.875272 0.159000 0.410502 0.326186*

*min 0.000000 76773.000000 0.078900 0.000005 0.001290*

*25% 30.000000 178986.000000 0.388000 0.014800 0.202000*

*50% 47.000000 213373.000000 0.506000 0.356000 0.496000*

*75% 57.000000 254613.000000 0.612000 0.905000 0.844000*

*max 88.000000 518373.000000 0.941000 0.996000 0.994000*

*Instrumentalness liveness valence loudness speechiness \*

*count 685.000000 685.000000 685.000000 685.000000 685.000000*

*mean 0.348835 0.166966 0.342820 -11.506343 0.053967*

*std 0.410314 0.142208 0.252289 7.446046 0.041400*

*min 0.000000 0.031800 0.035300 -42.018000 0.023200*

*25% 0.000033 0.092500 0.132000 -15.792000 0.033200*

*50% 0.025800 0.111000 0.283000 -9.343000 0.040700*

*75% 0.856000 0.174000 0.509000 -5.631000 0.057400*

*max 0.966000 0.963000 0.977000 1.342000 0.416000*

*tempo key time\_signature mood*

*count 685.000000 685.000000 685.000000 685.000000*

*mean 119.103958 5.343066 3.870073 1.366423*

*std 28.988679 3.547584 0.500638 1.121968*

*min 50.960000 0.000000 1.000000 0.000000*

*25% 99.008000 2.000000 4.000000 0.000000*

*50% 120.033000 6.000000 4.000000 1.000000*

*75% 132.912000 9.000000 4.000000 2.000000*

*max 217.950000 11.000000 5.000000 3.000000*

*count mean std min \*

*popularity 685.0 41.535766 23.062538 0.000000*

*length 685.0 221843.252555 63430.875272 76773.000000*

*danceability 685.0 0.500733 0.159000 0.078900*

*acousticness 685.0 0.448954 0.410502 0.000005*

*energy 685.0 0.508412 0.326186 0.001290*

*instrumentalness 685.0 0.348835 0.410314 0.000000*

*liveness 685.0 0.166966 0.142208 0.031800*

*valence 685.0 0.342820 0.252289 0.035300*

*loudness 685.0 -11.506343 7.446046 -42.018000*

*speechiness 685.0 0.053967 0.041400 0.023200*

*tempo 685.0 119.103958 28.988679 50.960000*

*key 685.0 5.343066 3.547584 0.000000*

*time\_signature 685.0 3.870073 0.500638 1.000000*

*mood 685.0 1.366423 1.121968 0.000000*

*25% 50% 75% max*

*popularity 30.000000 47.0000 57.0000 88.000*

*length 178986.000000 213373.0000 254613.0000 518373.000*

*danceability 0.388000 0.5060 0.6120 0.941*

*acousticness 0.014800 0.3560 0.9050 0.996*

*energy 0.202000 0.4960 0.8440 0.994*

*instrumentalness 0.000033 0.0258 0.8560 0.966*

*liveness 0.092500 0.1110 0.1740 0.963*

*valence 0.132000 0.2830 0.5090 0.977*

*loudness -15.792000 -9.3430 -5.6310 1.342*

*speechiness 0.033200 0.0407 0.0574 0.416*

*tempo 99.008000 120.0330 132.9120 217.950*

*key 2.000000 6.0000 9.0000 11.000*

*time\_signature 4.000000 4.0000 4.0000 5.000*

*mood 0.000000 1.0000 2.0000 3.000*

**Data Types used:**

*# Print the datatypes which are keys.*

*types = dataMoodsData.dtypes*

*print(types)*

*print("Keys of dataMoodsData dataset:\n", dataMoodsData.keys())*

popularity int64

length int64

danceability float64

acousticness float64

energy float64

instrumentalness float64

liveness float64

valence float64

loudness float64

speechiness float64

tempo float64

key int64

time\_signature int64

mood int64

dtype: object

***PART A - Categorical Variable Frequency Distribution***

*# print the target variable*

*print("Mood:", dataMoodsData['mood'])*

*Mood: 0 0*

*1 2*

*2 0*

*3 2*

*4 3*

*..*

*680 0*

*681 0*

*682 3*

*683 2*

*684 0*

*Value Category Total from Data Set*

0 Sad

1 Calm

2 Happy

3 Energetic

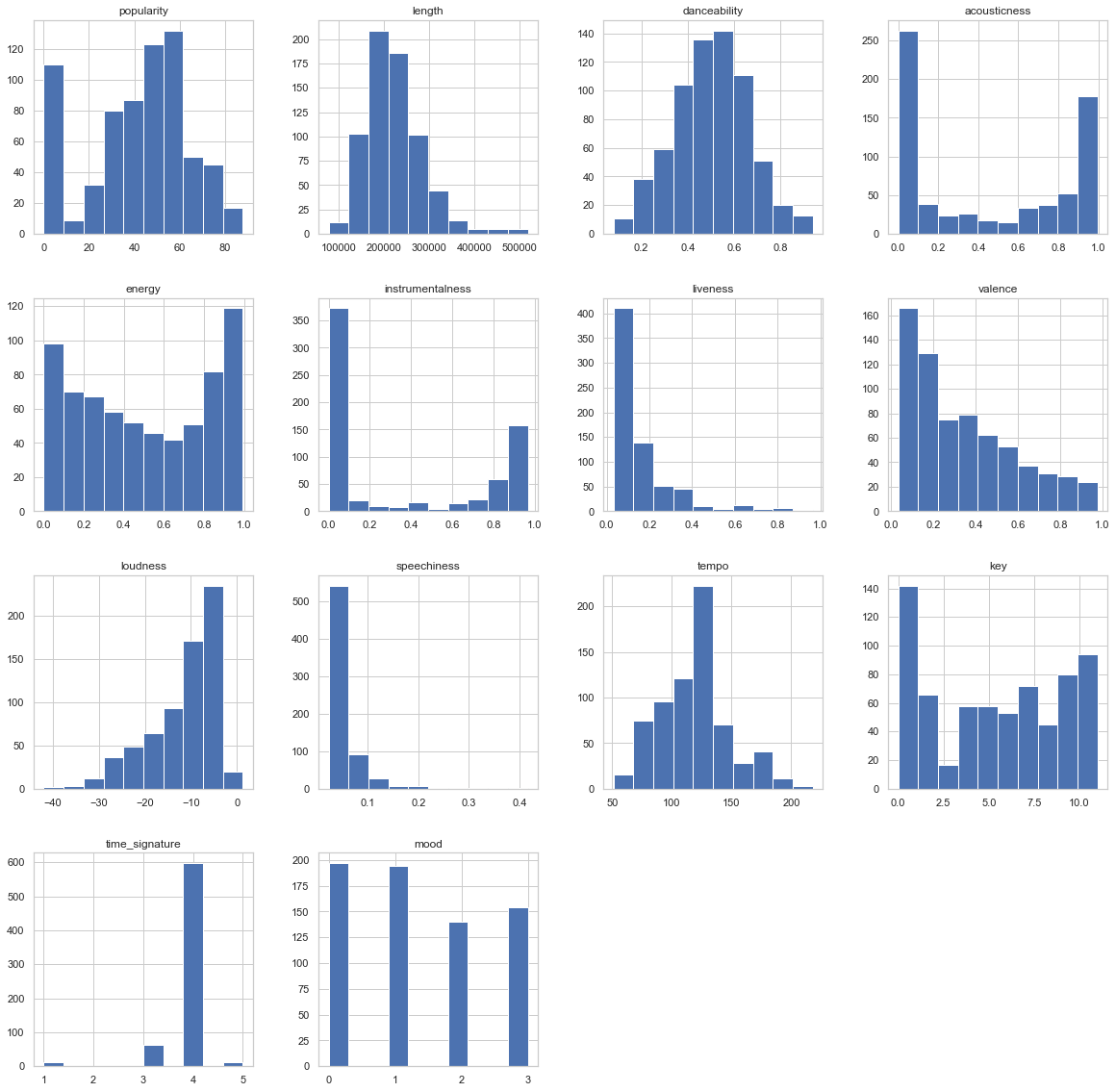
***PART B - Categorical Variable Histogram of all variables***

*# Create the histogram of all the variables.*

*%matplotlib inline*

*dataMoodsData.hist(figsize = (20,20))*

*plt.show()*

**

*In [35]:*

***Three Key Attributes and How they could potentially Interrelate.***

*# create a grid of scatter plot and histogram*

*%matplotlib inline*

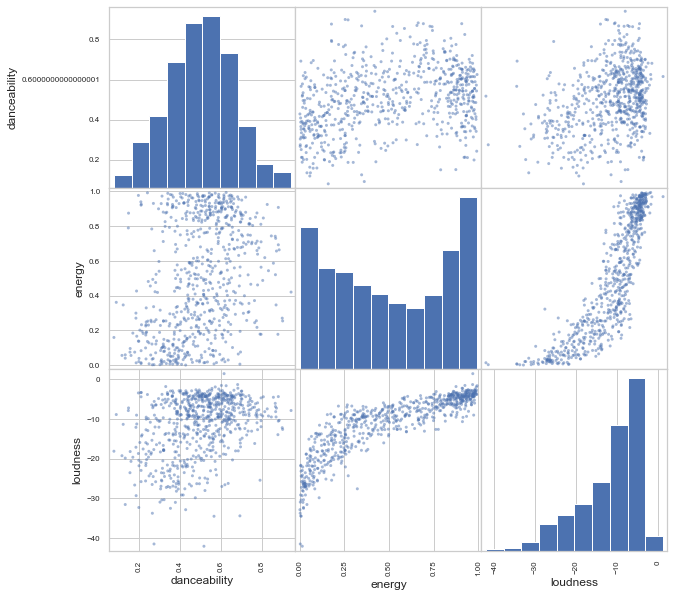
*X = dataMoodsData[['danceability','energy','loudness']]*

*y = dataMoodsData[['mood']]*

*from pandas.plotting import scatter\_matrix*

*scatter\_matrix(X,figsize=(10, 10))*

*plt.show()*

**

***Analysis of Graphical Visualizations - Basic***

***Danceability -***

***General -***

***The reason I chose this attribute was based on the idea that dancing typically involves a celebration and is a mode of expression that is universal to all cultures. Also dancing typically elicits in strong positive emotions which in turn has a major positive effect on a person’s mood. The data for danceability was interesting in that data created a normal curve based visualization. Due to this reason as well I felt it would create a well-balanced representation when cross-referenced with the other two attributes.***

***Vs. Energy -***

***The graph was quite scattered and had no visual correlation. I found it odd as to why this would be, since I had originally believed that Energy would have a strong tie/correlation to Dancing. My original thought was that Energy would parallel Dancing - higher the Energy value the higher the Danceability. My hypothesis is that while energy is needed to dance, dance is not integral to providing energy. Energy can be used as a motivator for other activities such as working out, driving, or other physical activities.***

***Vs. Loudness -***

***The graph comparing these attributes had a bit more correlation to it. The attributes Danceabliity and Loudness visually paralleled each other. The louder the music, the more danceable the music seemed to correlate with each other. This seems to help confirm that a softer sounding musical selection typically does not involve much dancing. It dawned on me that a “slow” or “romantic” dance would typically use a softer musical selection but still be very danceable. I looked over quite a bit of the music data points. I searched for songs typically used for slow dances, and I could not find any that I could recognize. I would not consider these to be outliers necessarily, but rather the dataset just did not contain these types of musical selections. For the dataset chosen however, the graph seemed to correspond to the logical parallel conclusion.***

***Energy -***

***General -***

***The reason I chose Energy as a specific attribute is that music has played a vital part in all aspects of entertainment. Whether that entertainment was for self enjoyment, working out, to rest with, or to enhance visual media, the energy conveyed typically helps to exemplify the general atmosphere that one is trying to achieve or experience. People often listen to calm music when resting or studying. Likewise, people listen to more energetic music when working out. Further it could be said that soundtracks for movies help to enhance the emotional atmosphere of what is transpiring - scary, joyful, sorrowful, etc.***

***Vs. Loudness -***

***The visualization of the two showed a strong positive correlation between each other when referencing a person’s mood. The louder the music, the stronger the energy attribute. My original prediction was that they would have a positive correlation, however I was curious to see that the correlation was not linear but rather logistic/exponential. I am not quite sure as to why this would be. I looked at the dataset and searched for songs that were both strong in both loudness and energy (likewise for weak values). It would seem that the loudness has a lesser impact on energy than energy has on loudness.***

***Loudness -***

***General -***

***The visualization helped to show that my data for this attribute was skewed- right. I believe that since the dataset was done through Spotify, and Spotify is new technology, I have the assumption that users of Spotify are typically younger and younger listeners typically listen to loud music regardless of music genre. The phrase “If it’s too loud, you’re too old” popped in my head when considering loudness and a person's age group or generation. I believe that the dataset would then lead to a right-skewed orientation.***

**Overall the data still provided evidence that these three attributes - Danceability, Energy, and Loudness - would provide a good metric for most music listeners to help enhance or elicit a person’s mood.**

*# Creating a pairplot differentiated by Mood*

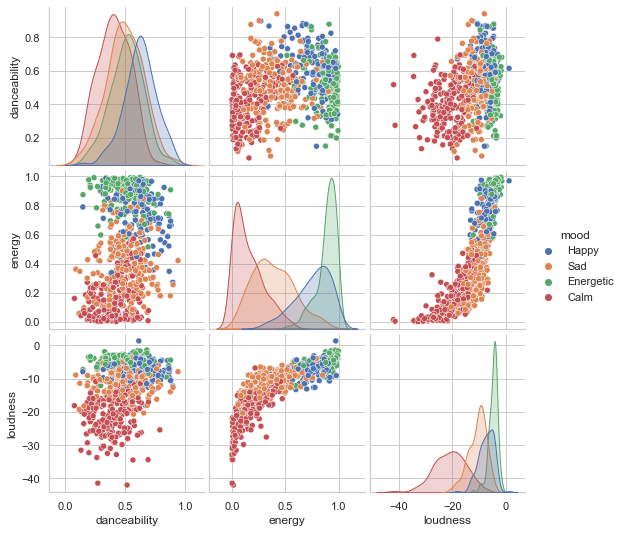
*%matplotlib inline*

*X = dataMoodsData[['danceability','energy','loudness','mood']]*

*from pandas.plotting import scatter\_matrix*

*sns.pairplot(X, hue = 'mood')*

*plt.show()*

**

***In Depth Analysis of Graphical Visualizations using Pair Plot***

***Danceability -***

***General -***

***As with the basic analysis, when the moods are put into perspective the individual moods also were on a bell curve. This helped to exemplify that the data was distributed fairly.***

***Vs. Energy -***

***The Pair Plot between Danceability and Energy helped to show that there was a type of clustering to the data. But the visualization helped to show there was a clustering based on only Energy. Danceability had little to no effect on any correlation of the data.***

***Vs. Loudness -***

***Like the comparison between Danciability and Energy there was strong evidence of clustering based on Loudness in particular for the Calm and Sad moods. The also was evidence of clustering , though not as strong, for Happy and Energetic. I would hypothesis that these two moods share many of the same emotions.***

***Energy -***

***General -***

***When looking at the Pair Plot of Energy and the particular moods the visualization showed that a song’s Energy level had a strong correlation with each individual mood. Also that the Energy values of Energetic and Happy were more aligned with each other than any other pairing. This helps to provide further evidence that the Happy and Energetic are more entwined with each other than the other two moods.***

***Vs. Loudness -***

***Again the visualization helped show a strong correlation as well as clustering of the data for both Calm and Sad music. Whereas Energetic and Happy were again intertwined.***

***Loudness -***

***General -***

***The visualization helped to show the data most mostly skewed appropriately. With the moods Sad, Happy, and Excited the data was aligned with what I believed would happen - skewed right with a relatively small range. However, the visualization of calm was peculiar due to its bell shaped curve, and large range. Having looked at the dataset I did not see any rational reason as to why this would be.***

**Overall the data still provided evidence that these three attributes - Danceability, Energy, and Loudness - would provide a good metric for most music listeners to help enhance or elicit a person’s mood.**

**STEP 2 - Preparation of the Data for kNN Implementation**

**The preparation of the data was done three separate times so that that data would be handled by different distance calculations.**

Preparation of the Data for kNN Implementation Using All Three Distances

*import numpy as np*

*import matplotlib.pyplot as plt*

*import pandas as pd*

*from sklearn import metrics*

*from sklearn.preprocessing import StandardScaler*

*import matplotlib.pyplot as plt*

*import warnings*

*warnings.filterwarnings('ignore')*

*​*

*​*

*# Divide data into predictor features vector and the label of the target variable 'Risk'*

*​*

*X = dataMoodsData[['danceability','energy','time\_signature','loudness']]*

*y = dataMoodsData[['mood']]*

*​*

*# Check if null values in the columns*

*X.isna().sum()*

​

**danceability 0**

**energy 0**

**time\_signature 0**

**loudness 0**

**dtype: int64**

*PART A - Standardize the Data*

*# Dividing data into two subsets training and test set.Training set trains the model*

*# X train: predictive variables in train set y train: train labels*

*# X test: predictive variables in test set y test test labels*

*from sklearn.model\_selection import train\_test\_split*

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25)*

*# Data normalization*

*scaler = StandardScaler()*

*scaler.fit(X\_train)*

*X\_train = scaler.transform(X\_train)*

*X\_test = scaler.transform(X\_test)*

*print("X\_train shape:", X\_train.shape)*

*print("y\_train shape:", y\_train.shape)*

*print("X\_test shape:", X\_test.shape)*

*print("y\_test shape:", y\_test.shape)*

***X\_train shape: (513, 4)***

***y\_train shape: (513, 1)***

***X\_test shape: (172, 4)***

***y\_test shape: (172, 1)***

***Using the Minkowski Distance Algorithm***

*# Initiating the scikit learn instance and fit is used to train the model.The predict() function*

*# implements prediction.It takes test data as an argument and returns predicted labels. Fitting*

*#the k-NN classiifer*

*# np.ravel() converts labels from a column format to the expected row format and flattens*

*# the numpy.ndarray.*

*​*

*import numpy as np*

*​*

*from sklearn.neighbors import KNeighborsClassifier*

*# Setting the number of neighbors*

*classifier = KNeighborsClassifier(n\_neighbors=10, metric = "minkowski")*

*# Loading the training set*

*classifier.fit(X\_train, np.ravel(y\_train,order='C'))*

*​*

*# Predicting the test labels*

*y\_pred = classifier.predict(X\_test)*

*y\_pred*

***array([0, 3, 2, 3, 2, 2, 1, 2, 2, 1, 0, 3, 1, 1, 1, 3, 0, 1, 3, 3, 3, 2,***

***0, 3, 2, 3, 0, 1, 0, 2, 2, 2, 0, 1, 3, 3, 1, 1, 1, 1, 0, 1, 0, 1,***

***1, 3, 1, 1, 3, 1, 0, 0, 0, 1, 1, 1, 2, 1, 1, 1, 2, 1, 3, 1, 0, 1,***

***1, 0, 2, 1, 0, 0, 0, 2, 3, 2, 0, 1, 1, 1, 0, 1, 0, 3, 1, 3, 2, 0,***

***0, 0, 3, 3, 1, 3, 2, 3, 2, 1, 0, 0, 1, 3, 1, 0, 3, 0, 3, 0, 0, 3,***

***3, 3, 1, 0, 1, 1, 0, 2, 1, 3, 3, 3, 0, 1, 1, 2, 1, 3, 1, 0, 1, 0,***

***3, 1, 0, 0, 3, 1, 0, 3, 1, 0, 0, 1, 1, 3, 0, 2, 0, 1, 0, 2, 3, 1,***

***3, 2, 3, 2, 1, 1, 3, 2, 3, 0, 0, 1, 3, 1, 1, 2, 3, 3], dtype=int64)***

***Evaluate the Minkowski algorithm and its model performance.***

*from sklearn.metrics import classification\_report, confusion\_matrix*

*# creating confusion matrix and printing the classification report*

*print(confusion\_matrix(y\_test, y\_pred))*

*print(classification\_report(y\_test, y\_pred))*

*accuracy = accuracy\_score(y\_test,y\_pred)\*100*

*print(accuracy)*

*​*

***[[29 11 5 1]***

***[ 7 46 0 0]***

***[ 4 2 15 10]***

***[ 4 0 6 32]]***

***precision recall f1-score support***

***0 0.66 0.63 0.64 46***

***1 0.78 0.87 0.82 53***

***2 0.58 0.48 0.53 31***

***3 0.74 0.76 0.75 42***

***accuracy 0.71 172***

***macro avg 0.69 0.69 0.69 172***

***weighted avg 0.70 0.71 0.70 172***

***Minkowski Accuracy: 70.93023255813954***

*Performance Improvement techniques:k values*

*k\_range = range(1, 40)*

*​*

*# Creating a Python dictionary by [] and then appending the accuracy scores*

*​*

*scores = []*

*# looping through the k range 1 to 40*

*​*

*for k in k\_range:*

*knn = KNeighborsClassifier(n\_neighbors=k)*

*knn.fit(X\_train,np.ravel(y\_train,order='C'))*

*y\_pred = knn.predict(X\_test)*

*# appending the accuracy scores in the dictionary named scores.*

*scores.append(metrics.accuracy\_score(y\_test, y\_pred))*

*​*

*print(scores)*

*# Printing the K number of neighbors and Testing Accuracy.*

*import matplotlib.pyplot as plt*

*​*

*# This command allow plots to appear within the notebook*

*%matplotlib inline*

*plt.plot(k\_range, scores)*

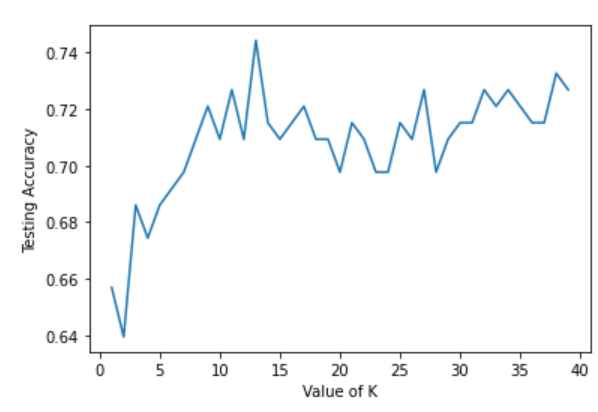
*plt.xlabel('Value of K')*

*plt.ylabel('Testing Accuracy')*

*​*

***[0.6569767441860465, 0.6395348837209303, 0.686046511627907, 0.6744186046511628, 0.686046511627907, 0.6918604651162791, 0.6976744186046512, 0.7093023255813954, 0.7209302325581395, 0.7093023255813954, 0.7267441860465116, 0.7093023255813954, 0.7441860465116279, 0.7151162790697675, 0.7093023255813954, 0.7151162790697675, 0.7209302325581395, 0.7093023255813954, 0.7093023255813954, 0.6976744186046512, 0.7151162790697675, 0.7093023255813954, 0.6976744186046512, 0.6976744186046512, 0.7151162790697675, 0.7093023255813954, 0.7267441860465116, 0.6976744186046512, 0.7093023255813954, 0.7151162790697675, 0.7151162790697675, 0.7267441860465116, 0.7209302325581395, 0.7267441860465116, 0.7209302325581395, 0.7151162790697675, 0.7151162790697675, 0.7325581395348837, 0.7267441860465116]***

***Text(0, 0.5, 'Testing Accuracy')***



*Performance Improvement techniques cross validation*

*# Optimizing the k-nn by using Cross validation*

*​*

*from sklearn.model\_selection import cross\_val\_score*

*import numpy as np*

*#create a new KNN model*

*knn\_cv = KNeighborsClassifier(n\_neighbors=15)*

*​*

*from sklearn.preprocessing import StandardScaler*

*scaler = StandardScaler()*

*scaler.fit(X)*

*​*

*X = scaler.transform(X)*

*​*

*#train model with cv of 10*

*cv\_scores = cross\_val\_score(knn\_cv, X, np.ravel(y,order='C'), cv=10)*

*#print each cv score (accuracy) and average them*

*print(cv\_scores)*

*print(np.mean(cv\_scores))*

***[0.73913043 0.73913043 0.73913043 0.7826087 0.7826087 0.75***

***0.67647059 0.75 0.73529412 0.72058824]***

***0.7414961636828645***

*Performance Improvement techniques cross validation: k values and cross validation*

*# Using cross validation with all possible k values.*

*​*

*from sklearn.model\_selection import cross\_val\_score*

*import numpy as np*

*#create a new KNN model*

*​*

*from sklearn.preprocessing import StandardScaler*

*scaler = StandardScaler()*

*scaler.fit(X)*

*​*

*# Train with 10 fold cross validation by an outer k value ranges and nested cross validation scores.*

*X = scaler.transform(X)*

*scores = []*

*k\_range = range(1, 40)*

*for k in k\_range:*

*#train model with cv of 10*

*knn\_cv = KNeighborsClassifier(n\_neighbors=k)*

*cv\_scores = cross\_val\_score(knn\_cv, X, np.ravel(y,order='C'), cv=10)*

*#print each cv score (accuracy) and average them*

*print(k)*

*print(cv\_scores)*

*print(np.mean(cv\_scores))*

*# Prediction*

*knn = KNeighborsClassifier(n\_neighbors=14)*

*knn.fit(X\_train,np.ravel(y\_train,order='C'))*

*y\_pred = knn.predict(X\_test)*

*accuracy\_scores = metrics.accuracy\_score(y\_test, y\_pred)*

*print(accuracy\_scores)*

*​*

*​*

***1***

***[0.71014493 0.5942029 0.66666667 0.60869565 0.69565217 0.57352941***

***0.60294118 0.66176471 0.63235294 0.69117647]***

***0.6437127024722933***

***2***

***[0.69565217 0.55072464 0.71014493 0.66666667 0.69565217 0.51470588***

***0.63235294 0.67647059 0.67647059 0.64705882]***

***0.6465899403239558***

***3***

***[0.66666667 0.66666667 0.75362319 0.71014493 0.69565217 0.64705882***

***0.60294118 0.67647059 0.66176471 0.67647059]***

***0.6757459505541348***

***4***

***[0.68115942 0.71014493 0.68115942 0.73913043 0.73913043 0.63235294***

***0.63235294 0.67647059 0.69117647 0.69117647]***

***0.6874254049445866***

***5***

***[0.71014493 0.71014493 0.72463768 0.72463768 0.76811594 0.64705882***

***0.64705882 0.64705882 0.72058824 0.69117647]***

***0.6990622335890879***

***6***

***[0.68115942 0.69565217 0.71014493 0.72463768 0.76811594 0.67647059***

***0.66176471 0.66176471 0.70588235 0.73529412]***

***0.7020886615515771***

***7***

***[0.69565217 0.73913043 0.76811594 0.76811594 0.76811594 0.70588235***

***0.66176471 0.72058824 0.75 0.76470588]***

***0.7342071611253197***

***8***

***[0.68115942 0.66666667 0.72463768 0.76811594 0.76811594 0.70588235***

***0.63235294 0.70588235 0.72058824 0.75 ]***

***0.7123401534526854***

***9***

***[0.68115942 0.71014493 0.73913043 0.76811594 0.76811594 0.72058824***

***0.63235294 0.73529412 0.73529412 0.73529412]***

***0.7225490196078431***

***10***

***[0.71014493 0.73913043 0.73913043 0.75362319 0.7826087 0.72058824***

***0.67647059 0.75 0.73529412 0.79411765]***

***0.7401108269394714***

***11***

***[0.68115942 0.73913043 0.7826087 0.75362319 0.76811594 0.73529412***

***0.67647059 0.76470588 0.75 0.76470588]***

***0.7415814151747656***

***12***

***[0.71014493 0.69565217 0.71014493 0.76811594 0.76811594 0.73529412***

***0.69117647 0.73529412 0.70588235 0.76470588]***

***0.7284526854219949***

***13***

***[0.71014493 0.71014493 0.75362319 0.75362319 0.76811594 0.76470588***

***0.67647059 0.77941176 0.72058824 0.76470588]***

***0.740153452685422***

***14***

***[0.72463768 0.71014493 0.72463768 0.76811594 0.7826087 0.76470588***

***0.67647059 0.76470588 0.70588235 0.75 ]***

***0.7371909633418585***

***15***

***[0.73913043 0.73913043 0.73913043 0.7826087 0.7826087 0.75***

***0.67647059 0.75 0.73529412 0.72058824]***

***0.7414961636828645***

***16***

***[0.73913043 0.71014493 0.72463768 0.75362319 0.79710145 0.73529412***

***0.70588235 0.75 0.69117647 0.73529412]***

***0.7342284739982949***

***17***

***[0.73913043 0.69565217 0.76811594 0.73913043 0.76811594 0.73529412***

***0.70588235 0.77941176 0.70588235 0.72058824]***

***0.7357203751065644***

***18***

***[0.73913043 0.69565217 0.71014493 0.75362319 0.79710145 0.77941176***

***0.72058824 0.76470588 0.67647059 0.72058824]***

***0.7357416879795398***

***19***

***[0.73913043 0.72463768 0.75362319 0.79710145 0.75362319 0.73529412***

***0.73529412 0.75 0.69117647 0.70588235]***

***0.7385763000852515***

***20***

***[0.73913043 0.71014493 0.69565217 0.76811594 0.76811594 0.72058824***

***0.73529412 0.75 0.67647059 0.72058824]***

***0.7284100596760443***

***21***

***[0.72463768 0.72463768 0.72463768 0.76811594 0.7826087 0.72058824***

***0.73529412 0.76470588 0.69117647 0.70588235]***

***0.7342284739982949***

***22***

***[0.73913043 0.72463768 0.71014493 0.75362319 0.75362319 0.75***

***0.73529412 0.76470588 0.64705882 0.69117647]***

***0.7269394714407502***

***23***

***[0.75362319 0.73913043 0.72463768 0.75362319 0.76811594 0.73529412***

***0.72058824 0.76470588 0.67647059 0.69117647]***

***0.7327365728900256***

***24***

***[0.72463768 0.72463768 0.71014493 0.76811594 0.76811594 0.72058824***

***0.73529412 0.77941176 0.64705882 0.69117647]***

***0.726918158567775***

***25***

***[0.73913043 0.72463768 0.72463768 0.76811594 0.76811594 0.72058824***

***0.73529412 0.77941176 0.69117647 0.69117647]***

***0.7342284739982949***

***26***

***[0.75362319 0.73913043 0.71014493 0.76811594 0.76811594 0.76470588***

***0.73529412 0.77941176 0.66176471 0.69117647]***

***0.7371483375959079***

***27***

***[0.76811594 0.71014493 0.75362319 0.76811594 0.76811594 0.76470588***

***0.73529412 0.77941176 0.72058824 0.67647059]***

***0.744458653026428***

***28***

***[0.75362319 0.72463768 0.69565217 0.76811594 0.75362319 0.77941176***

***0.73529412 0.77941176 0.69117647 0.67647059]***

***0.7357416879795398***

***29***

***[0.73913043 0.71014493 0.75362319 0.75362319 0.75362319 0.79411765***

***0.72058824 0.77941176 0.72058824 0.66176471]***

***0.7386615515771526***

***30***

***[0.73913043 0.69565217 0.72463768 0.76811594 0.7826087 0.75***

***0.75 0.76470588 0.70588235 0.67647059]***

***0.7357203751065644***

***31***

***[0.73913043 0.69565217 0.72463768 0.76811594 0.73913043 0.73529412***

***0.69117647 0.75 0.72058824 0.67647059]***

***0.7240196078431372***

***32***

***[0.76811594 0.69565217 0.72463768 0.76811594 0.76811594 0.75***

***0.69117647 0.75 0.75 0.70588235]***

***0.7371696504688832***

***33***

***[0.73913043 0.71014493 0.76811594 0.7826087 0.73913043 0.75***

***0.72058824 0.75 0.73529412 0.72058824]***

***0.7415601023017903***

***34***

***[0.73913043 0.71014493 0.72463768 0.7826087 0.73913043 0.72058824***

***0.70588235 0.75 0.70588235 0.72058824]***

***0.7298593350383632***

***35***

***[0.76811594 0.72463768 0.73913043 0.7826087 0.75362319 0.73529412***

***0.70588235 0.75 0.75 0.70588235]***

***0.7415174765558398***

***36***

***[0.76811594 0.73913043 0.72463768 0.7826087 0.73913043 0.75***

***0.69117647 0.73529412 0.72058824 0.72058824]***

***0.7371270247229327***

***37***

***[0.73913043 0.73913043 0.72463768 0.7826087 0.75362319 0.72058824***

***0.69117647 0.75 0.72058824 0.70588235]***

***0.7327365728900256***

***38***

***[0.73913043 0.73913043 0.75362319 0.7826087 0.76811594 0.75***

***0.69117647 0.75 0.70588235 0.73529412]***

***0.7414961636828645***

***39***

***[0.72463768 0.73913043 0.75362319 0.76811594 0.75362319 0.73529412***

***0.67647059 0.75 0.72058824 0.72058824]***

***0.7342071611253197***

***0.7151162790697675***

*Performance Improvement techniques: Feature Importance*

*import numpy as np*

*import pandas as pd*

*from sklearn.model\_selection import train\_test\_split*

*from sklearn.ensemble import RandomForestRegressor*

*import seaborn as sns*

*import matplotlib.pyplot as plt*

*%matplotlib inline*

*​*

*sns.set(style="whitegrid")*

*from matplotlib import pyplot as plt*

*​*

*​*

*# overriding the default figure size and font size.*

*# plt.rcParams.update({'figure.figsize': (12.0, 8.0)})*

*# plt.rcParams.update({'font.size': 14})*

*​*

*​*

*dataMoodsData = pd.read\_csv('data\_moods3.csv')*

*X = pd.DataFrame(dataMoodsData)*

*X = X.iloc[:,0:13]*

*print(X)*

*​*

*​*

*y = dataMoodsData['mood']*

*print(y)*

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=12)*

*​*

*# Generating the importance of predictive variables using Random Forest*

*rf = RandomForestRegressor(n\_estimators=100)*

*rf.fit(X\_train, y\_train)*

*rf.feature\_importances\_*

*print(rf.feature\_importances\_)*

*# Sorting them in descending order*

*sorted\_idx = rf.feature\_importances\_.argsort()*

*# Printing feature scores*

*plt.barh(X\_train.columns[sorted\_idx], rf.feature\_importances\_[sorted\_idx])*

*plt.xlabel("Random Forest Feature Importance")*

*​*

*print(sorted\_idx)*

*​*

*# alternate way of viewing the feature scores in descending order*

*​*

*feature\_scores = pd.Series(rf.feature\_importances\_, index=X\_train.columns).sort\_values(ascending=False)*

*​*

*print(feature\_scores)*

*# f, ax represents figure f and the axes.*

*f, ax = plt.subplots(figsize=(30, 24))*

*ax = sns.barplot(x=feature\_scores, y=feature\_scores.index, data=dataMoodsData.T)*

*ax.set\_title("Visualize feature scores of the features")*

*ax.set\_yticklabels(feature\_scores.index)*

*ax.set\_xlabel("Feature importance score")*

*ax.set\_ylabel("Features")*

*plt.show()*

*​*

*​*

popularity length danceability acousticness energy instrumentalness \

0 43 318800 0.381 0.01890 0.8320 0.196000

1 68 379266 0.866 0.13700 0.7300 0.000000

2 60 217946 0.346 0.91300 0.1390 0.000077

3 2 233000 0.466 0.08900 0.4380 0.000006

4 60 268000 0.419 0.00171 0.9320 0.000000

.. ... ... ... ... ... ...

680 76 219146 0.561 0.91300 0.0848 0.000026

681 49 239500 0.634 0.81900 0.1930 0.000000

682 0 178333 0.547 0.00485 0.8770 0.000000

683 57 170800 0.518 0.15800 0.7610 0.007650

684 0 251626 0.588 0.61000 0.4770 0.001880

liveness valence loudness speechiness tempo key time\_signature

0 0.1530 0.166 -5.069 0.0492 120.255 8 4

1 0.0843 0.625 -8.201 0.0767 118.523 5 4

2 0.0934 0.116 -15.326 0.0321 136.168 0 4

3 0.1130 0.587 -12.858 0.0608 193.100 4 4

4 0.1370 0.445 -3.604 0.1060 169.881 1 4

.. ... ... ... ... ... ... ...

680 0.1120 0.206 -15.099 0.0404 102.128 2 4

681 0.1130 0.159 -9.503 0.0277 95.004 7 4

682 0.2470 0.611 -5.135 0.0658 126.083 9 4

683 0.0715 0.576 -7.025 0.0452 118.738 0 4

684 0.1220 0.488 -12.710 0.0492 150.396 9 4

[685 rows x 13 columns]

0 0

1 2

2 0

3 2

4 3

..

680 0

681 0

682 3

683 2

684 0

Name: mood, Length: 685, dtype: int64

[0.01890469 0.02738925 0.02159725 0.04646389 0.64212032 0.07128779

0.0147028 0.05045591 0.03678045 0.03655287 0.02121519 0.0095334

0.00299618]

[12 11 6 0 10 2 1 9 8 3 7 5 4]

energy 0.642120

instrumentalness 0.071288

valence 0.050456

acousticness 0.046464

loudness 0.036780

speechiness 0.036553

length 0.027389

danceability 0.021597

tempo 0.021215

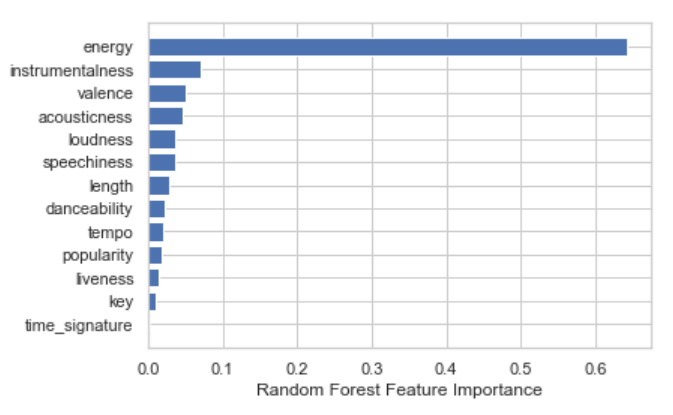
popularity 0.018905

liveness 0.014703

key 0.009533

time\_signature 0.002996

dtype: float64

****

*# Using a subset of predictor feature variables for the classification:*

*​*

*from sklearn.model\_selection import cross\_val\_score*

*import numpy as np*

*from sklearn.neighbors import KNeighborsClassifier*

*#create a new KNN model*

*​*

*from sklearn.preprocessing import StandardScaler*

*from sklearn import metrics*

*​*

*dataMoodsData\_Optimize = pd.read\_csv('data\_moods3.csv')*

*# Remove spaces in the column names*

*dataMoodsData\_Optimize.columns = dataMoodsData\_Optimize.columns.to\_series().apply(lambda x: x.strip())*

*X = dataMoodsData\_Optimize[['popularity','length','tempo','time\_signature','danceability','energy','loudness']]*

*y = dataMoodsData\_Optimize[['mood']]*

*​*

*​*

*scaler = StandardScaler()*

*scaler.fit(X)*

*​*

*X = scaler.transform(X)*

*scores = []*

*k\_range = range(1, 40)*

*for k in k\_range:*

*#train model with cv of 10*

*knn\_cv = KNeighborsClassifier(n\_neighbors=k)*

*cv\_scores = cross\_val\_score(knn\_cv, X, np.ravel(y,order='C'), cv=10)*

*#print each cv score (accuracy) and average them*

*print(k)*

*print(cv\_scores)*

*print(np.mean(cv\_scores))*

*knn = KNeighborsClassifier(n\_neighbors=15)*

*knn.fit(X\_train,np.ravel(y\_train,order='C'))*

*y\_pred = knn.predict(X\_test)*

*accuracy\_scores = metrics.accuracy\_score(y\_test, y\_pred)*

*print(accuracy\_scores)*

***1***

***[0.66666667 0.66666667 0.63768116 0.73913043 0.71014493 0.66176471***

***0.58823529 0.70588235 0.66176471 0.66176471]***

***0.6699701619778347***

***2***

***[0.60869565 0.66666667 0.71014493 0.7826087 0.71014493 0.64705882***

***0.61764706 0.75 0.67647059 0.66176471]***

***0.6831202046035807***

***3***

***[0.73913043 0.71014493 0.68115942 0.76811594 0.72463768 0.66176471***

***0.70588235 0.73529412 0.69117647 0.63235294]***

***0.7049658994032397***

***4***

***[0.68115942 0.71014493 0.65217391 0.76811594 0.73913043 0.64705882***

***0.73529412 0.79411765 0.70588235 0.72058824]***

***0.7153665814151748***

***5***

***[0.73913043 0.68115942 0.7826087 0.73913043 0.76811594 0.73529412***

***0.77941176 0.73529412 0.75 0.67647059]***

***0.7386615515771526***

***6***

***[0.69565217 0.68115942 0.75362319 0.72463768 0.69565217 0.73529412***

***0.82352941 0.72058824 0.72058824 0.66176471]***

***0.7212489343563513***

***7***

***[0.73913043 0.66666667 0.75362319 0.75362319 0.72463768 0.79411765***

***0.79411765 0.73529412 0.77941176 0.63235294]***

***0.7372975277067348***

***8***

***[0.69565217 0.68115942 0.72463768 0.76811594 0.71014493 0.80882353***

***0.79411765 0.72058824 0.80882353 0.73529412]***

***0.7447357203751065***

***9***

***[0.75362319 0.68115942 0.73913043 0.76811594 0.68115942 0.77941176***

***0.80882353 0.73529412 0.76470588 0.70588235]***

***0.7417306052855925***

***10***

***[0.72463768 0.68115942 0.72463768 0.76811594 0.71014493 0.80882353***

***0.80882353 0.77941176 0.77941176 0.70588235]***

***0.7491048593350385***

***11***

***[0.76811594 0.69565217 0.72463768 0.7826087 0.72463768 0.75***

***0.79411765 0.79411765 0.76470588 0.75 ]***

***0.7548593350383632***

***12***

***[0.72463768 0.72463768 0.71014493 0.75362319 0.71014493 0.75***

***0.77941176 0.76470588 0.75 0.75 ]***

***0.7417306052855925***

***13***

***[0.69565217 0.75362319 0.75362319 0.76811594 0.73913043 0.76470588***

***0.79411765 0.73529412 0.76470588 0.70588235]***

***0.7474850809889173***

***14***

***[0.69565217 0.72463768 0.73913043 0.76811594 0.71014493 0.75***

***0.79411765 0.73529412 0.77941176 0.72058824]***

***0.7417092924126172***

***15***

***[0.75362319 0.73913043 0.75362319 0.7826087 0.72463768 0.76470588***

***0.79411765 0.72058824 0.79411765 0.72058824]***

***0.754774083546462***

***16***

***[0.73913043 0.72463768 0.71014493 0.7826087 0.72463768 0.75***

***0.82352941 0.76470588 0.79411765 0.72058824]***

***0.7534100596760444***

***17***

***[0.73913043 0.71014493 0.72463768 0.7826087 0.72463768 0.76470588***

***0.77941176 0.73529412 0.80882353 0.73529412]***

***0.7504688832054561***

***18***

***[0.75362319 0.71014493 0.71014493 0.76811594 0.73913043 0.75***

***0.80882353 0.75 0.77941176 0.76470588]***

***0.7534100596760444***

***19***

***[0.73913043 0.75362319 0.73913043 0.7826087 0.73913043 0.75***

***0.80882353 0.75 0.80882353 0.77941176]***

***0.7650682011935208***

***20***

***[0.71014493 0.72463768 0.73913043 0.75362319 0.71014493 0.75***

***0.77941176 0.75 0.79411765 0.77941176]***

***0.7490622335890877***

***21***

***[0.72463768 0.73913043 0.75362319 0.75362319 0.73913043 0.73529412***

***0.79411765 0.73529412 0.79411765 0.77941176]***

***0.7548380221653879***

***22***

***[0.72463768 0.72463768 0.76811594 0.76811594 0.72463768 0.73529412***

***0.80882353 0.75 0.82352941 0.77941176]***

***0.7607203751065643***

***23***

***[0.72463768 0.75362319 0.76811594 0.76811594 0.72463768 0.75***

***0.80882353 0.73529412 0.82352941 0.76470588]***

***0.7621483375959079***

***24***

***[0.72463768 0.73913043 0.75362319 0.76811594 0.72463768 0.73529412***

***0.79411765 0.75 0.82352941 0.76470588]***

***0.7577791986359761***

***25***

***[0.71014493 0.73913043 0.72463768 0.7826087 0.73913043 0.75***

***0.76470588 0.73529412 0.80882353 0.79411765]***

***0.7548593350383631***

***26***

***[0.71014493 0.73913043 0.72463768 0.7826087 0.72463768 0.77941176***

***0.77941176 0.73529412 0.80882353 0.75 ]***

***0.7534100596760442***

***27***

***[0.69565217 0.73913043 0.75362319 0.76811594 0.73913043 0.76470588***

***0.79411765 0.75 0.80882353 0.73529412]***

***0.7548593350383631***

***28***

***[0.71014493 0.73913043 0.75362319 0.7826087 0.71014493 0.76470588***

***0.77941176 0.75 0.79411765 0.75 ]***

***0.7533887468030691***

***29***

***[0.69565217 0.75362319 0.72463768 0.79710145 0.72463768 0.75***

***0.77941176 0.75 0.80882353 0.73529412]***

***0.7519181585677749***

***30***

***[0.66666667 0.75362319 0.71014493 0.7826087 0.72463768 0.75***

***0.79411765 0.75 0.79411765 0.72058824]***

***0.7446504688832054***

***31***

***[0.69565217 0.76811594 0.72463768 0.7826087 0.68115942 0.76470588***

***0.77941176 0.76470588 0.77941176 0.73529412]***

***0.7475703324808184***

***32***

***[0.69565217 0.75362319 0.76811594 0.76811594 0.71014493 0.75***

***0.79411765 0.76470588 0.77941176 0.75 ]***

***0.753388746803069***

***33***

***[0.71014493 0.73913043 0.73913043 0.7826087 0.71014493 0.76470588***

***0.75 0.76470588 0.79411765 0.75 ]***

***0.7504688832054561***

***34***

***[0.71014493 0.75362319 0.73913043 0.79710145 0.71014493 0.75***

***0.76470588 0.76470588 0.79411765 0.76470588]***

***0.7548380221653879***

***35***

***[0.71014493 0.75362319 0.73913043 0.79710145 0.71014493 0.75***

***0.76470588 0.76470588 0.77941176 0.75 ]***

***0.7518968456947996***

***36***

***[0.71014493 0.76811594 0.73913043 0.7826087 0.72463768 0.76470588***

***0.76470588 0.79411765 0.77941176 0.72058824]***

***0.7548167092924126***

***37***

***[0.72463768 0.75362319 0.72463768 0.7826087 0.71014493 0.75***

***0.76470588 0.75 0.77941176 0.70588235]***

***0.7445652173913044***

***38***

***[0.72463768 0.72463768 0.75362319 0.76811594 0.73913043 0.76470588***

***0.77941176 0.77941176 0.76470588 0.69117647]***

***0.7489556692242114***

***39***

***[0.69565217 0.72463768 0.73913043 0.7826087 0.73913043 0.76470588***

***0.76470588 0.77941176 0.76470588 0.69117647]***

***0.7445865302642797***

***0.3430232558139535***

*Evaluating Algorithms: Train versus Test, Error Rates*

*from sklearn.neighbors import KNeighborsClassifier*

*​test\_scores = []*

*train\_scores = []*

*​*

*for i in range(1,15):*

*​*

*knn = KNeighborsClassifier(i)*

*knn.fit(X\_train,y\_train)*

*train\_scores.append(knn.score(X\_train,y\_train))*

*test\_scores.append(knn.score(X\_test,y\_test))*

*## Training Evaluation*

*max\_train\_score = max(train\_scores)*

*​*

*# # Store the max train test score index by enumerating through all the scores.*

*​*

*train\_scores\_ind = [i for i, v in enumerate(train\_scores) if v == max\_train\_score]*

*​*

*# Store the max score in the first curly parenthesis and the indices in the second.*

*# The lambda function takes the index starting at zero therefore one is added to get the k value.*

*​*

*print('Max train score {} % and k = {}'.format(max\_train\_score\*100,list(map(lambda x: x+1, train\_scores\_ind))))*

*​*

*## Testing Evaluation*

*max\_test\_score = max(test\_scores)*

*​*

*test\_scores\_ind = [i for i, v in enumerate(test\_scores) if v == max\_test\_score]*

*print('Max test score {} % and k = {}'.format(max\_test\_score\*100,list(map(lambda x: x+1, test\_scores\_ind))))*

*​*

*## Train Test Evaluation by comparative graph.*

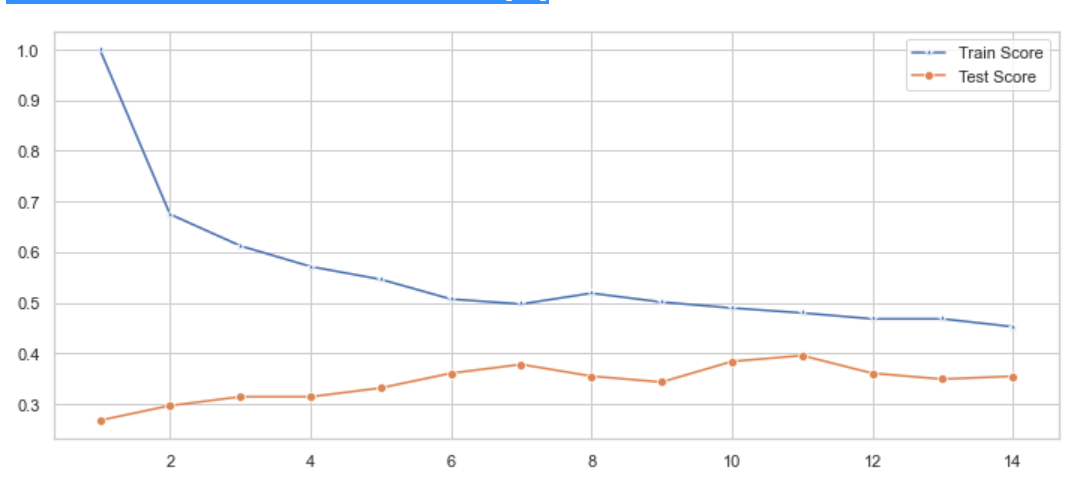
*plt.figure(figsize=(12,5))*

*p = sns.lineplot(range(1,15),train\_scores,marker='\*',label='Train Score')*

*p = sns.lineplot(range(1,15),test\_scores,marker='o',label='Test Score')*

***Max train score 100.0 % and k = [1]***

***Max test score 39.53488372093023 % and k = [11]***



*## Error Rate Graph*

*# Create an empty dictionary to collect errors across the different k-values*

*error = []*

*​# Iterate through k=1 to 40 and run the classifier.Predict and append the error for each iteration.*

*for i in range(1, 40):*

*knn = KNeighborsClassifier(n\_neighbors=i)*

*knn.fit(X\_train, y\_train)*

*pred\_i = knn.predict(X\_test)*

*error.append(np.mean(pred\_i != y\_test))*

*​*

*# Create a plot of Mean error versus kvalue.*

*plt.figure(figsize=(12, 6))*

*plt.plot(range(1, 40), error, color='red', linestyle='dashed', marker='o',*

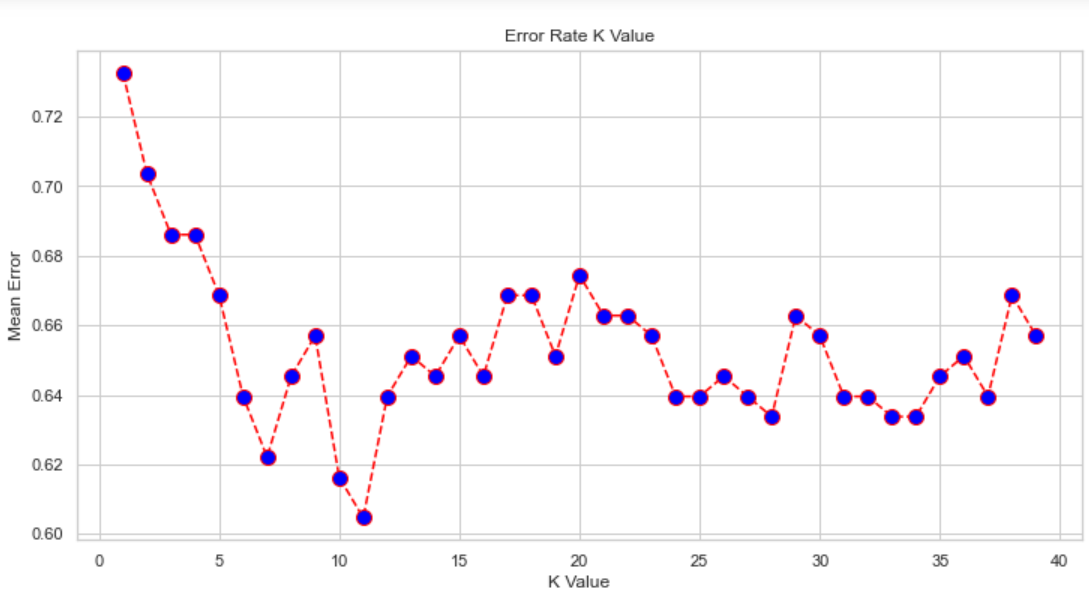
*markerfacecolor='blue', markersize=10)*

*plt.title('Error Rate K Value')*

*plt.xlabel('K Value')*

*plt.ylabel('Mean Error')*

***Text(0, 0.5, 'Mean Error')***

*​*******

***Using the other two Distance Algorithms - Manhattan and Euclidean - the code was quite similar with the obvious difference when using the KNN Classifier Metric parameter.***

***knn = KNeighborsClassifier(n\_neighbors=i, metric = "minkowski") vs***

***knn = KNeighborsClassifier(n\_neighbors=i, metric = "euclidean") vs***

***knn = KNeighborsClassifier(n\_neighbors=i, metric = "manhattan")***

***Looking at a side by side comparison of key results would provide a better understanding of the differences between the three algorithms***

***Minkowski***

***[[29 11 5 1]***

***[ 7 46 0 0]***

***[ 4 2 15 10]***

***[ 4 0 6 32]]***

***precision recall f1-score support***

***0 0.66 0.63 0.64 46***

***1 0.78 0.87 0.82 53***

***2 0.58 0.48 0.53 31***

***3 0.74 0.76 0.75 42***

***accuracy 0.71 172***

***macro avg 0.69 0.69 0.69 172***

***weighted avg 0.70 0.71 0.70 172***

***Minkowski Accuracy: 70.93023255813954***

***Manhattan***

***[[38 5 5 3]***

***[ 6 37 0 0]***

***[ 6 1 23 5]***

***[ 1 0 11 31]]***

***precision recall f1-score support***

***0 0.75 0.75 0.75 51***

***1 0.86 0.86 0.86 43***

***2 0.59 0.66 0.62 35***

***3 0.79 0.72 0.76 43***

***accuracy 0.75 172***

***macro avg 0.75 0.75 0.75 172***

***weighted avg 0.75 0.75 0.75 172***

***Manhattan Accuracy***

***75.0***

***Euclidean***

***[[44 3 3 2]***

***[11 43 0 0]***

***[10 0 13 12]***

***[ 1 0 7 23]]***

***precision recall f1-score support***

***0 0.67 0.85 0.75 52***

***1 0.93 0.80 0.86 54***

***2 0.57 0.37 0.45 35***

***3 0.62 0.74 0.68 31***

***accuracy 0.72 172***

***macro avg 0.70 0.69 0.68 172***

***weighted avg 0.72 0.72 0.71 172***

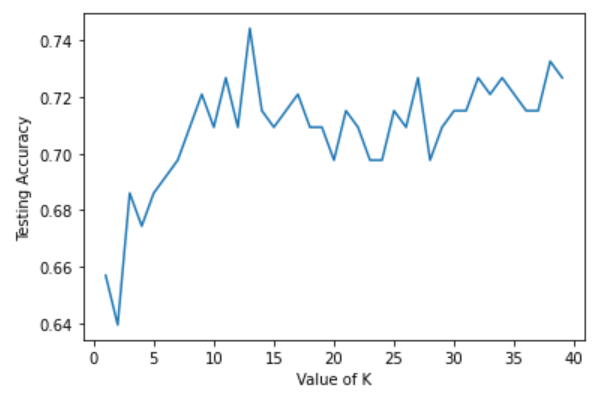
***Euclidean Accuracy 71.51162790697676***

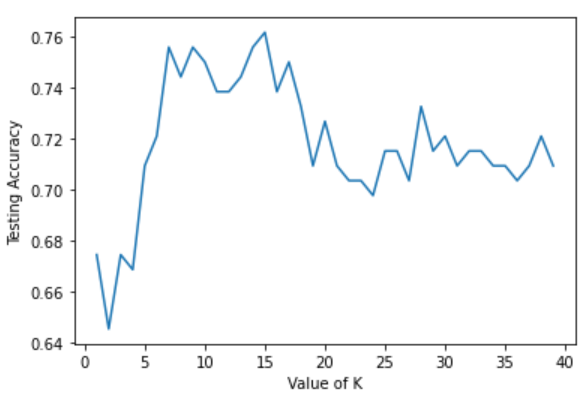
***Analysis of the Accuracy -***

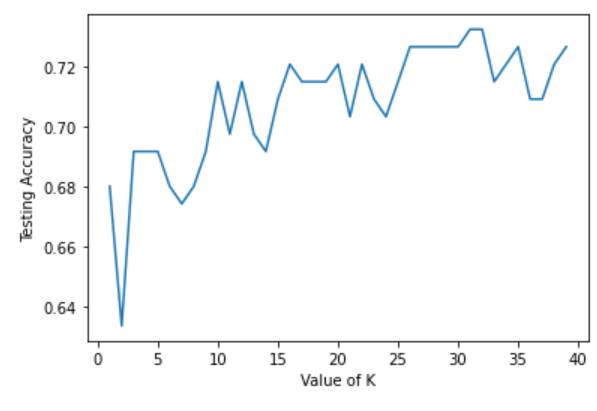
***Oddly the most accurate turned out to be the least complicated algorithm. It is my assumption then that the dataset had few outliers that would have affected the more complicated algorithms.***

***Looking Multiple K-values***

***Minkowski Manhattan Euclidean***

******

****

****

**When looking at the three graphs all three do quickly elevate their accuracy levels. However, Minkowski and Manhattan’s K-values both peak at similarly at a much lower value than Euclidean representation. The Euclidean representation reaches is most optimal level at k -values exceeding k >20. In terms of long range accuracy levels of Manhattan drops considerably, whereas both Minkowski and Euclidean have a tighter or more cohesive range of predictive values.**

**Looking at Cross validation and Mean Prediction**

**Minkowski**

**[0.73913043 0.73913043 0.73913043 0.7826087 0.7826087 0.75 0.67647059 0.75 0.73529412 0.72058824]**

**Minkowski Average 0.7414961636828645**

**Manhattan**

**[0.75362319 0.71014493 0.76811594 0.75362319 0.7826087 0.76470588**

**0.64705882 0.75 0.72058824 0.72058824]**

**Manhattan Average 0.7371057118499573**

**Euclidean**

**[0.73913043 0.73913043 0.73913043 0.7826087 0.7826087 0.75**

**0.67647059 0.75**

**0.73529412 0.72058824]**

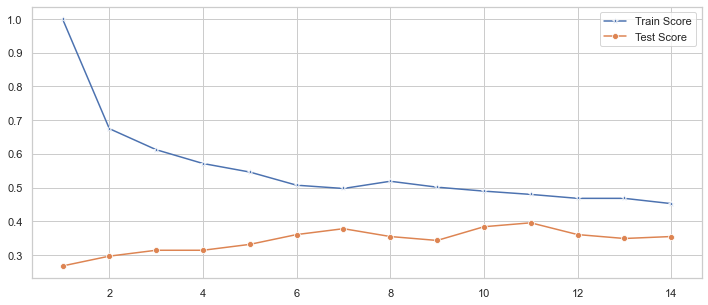
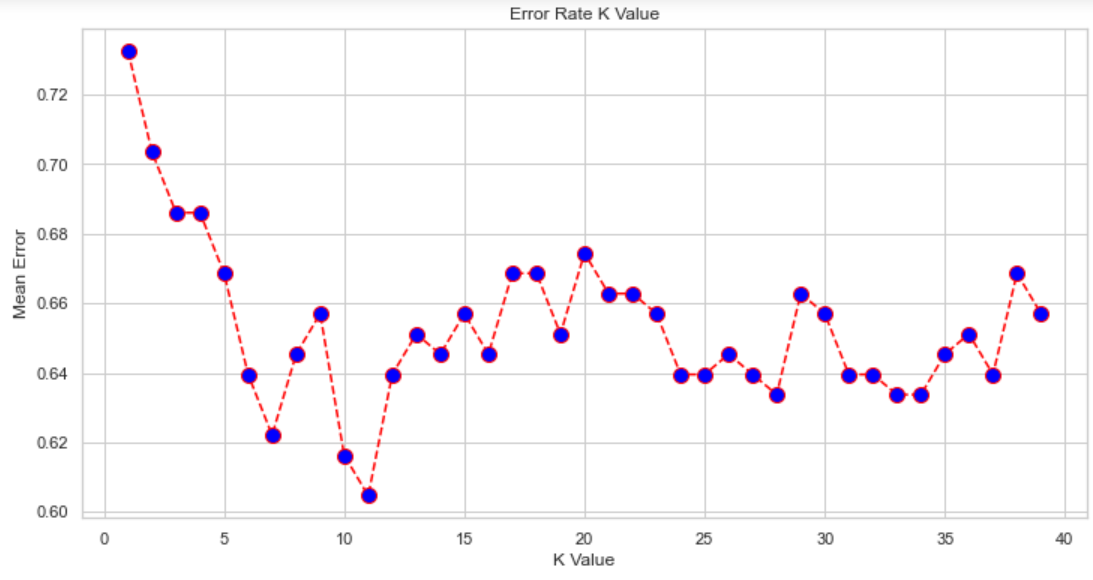
**Euclidean Average 0.7414961636828645**

**When looking at the averages of some of the k-values it would conclude that the original Averages are now in reverse order. This is evident when considering the Manhattan graph and that its accuracy fell dramatically after hitting its peak accuracy. Similarly the Euclidean maintained a tighter range of values which would lead to higher accuracy levels.**

**Minkowski**

**Max train score 100.0 % and k = [1]**

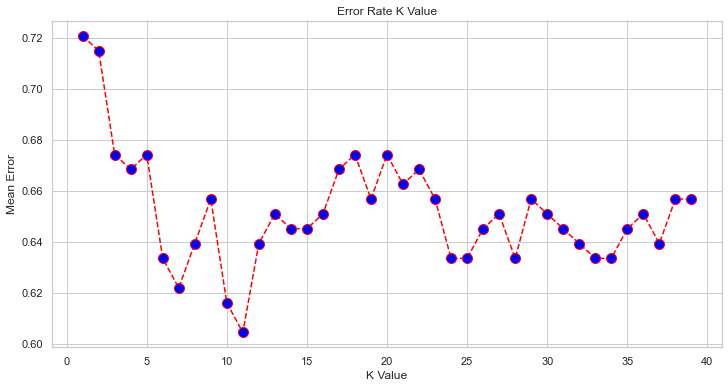
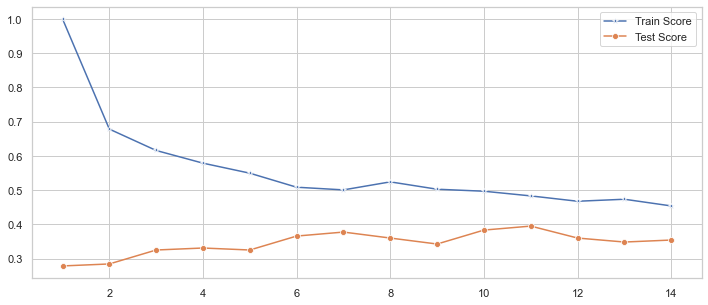
**Max test score 39.53488372093023 % and k = [11]**

** **

**Manhattan**

**Max train score 100.0 % and k = [1]**

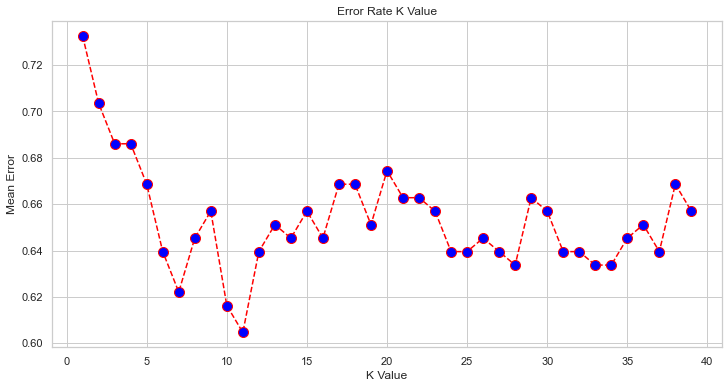
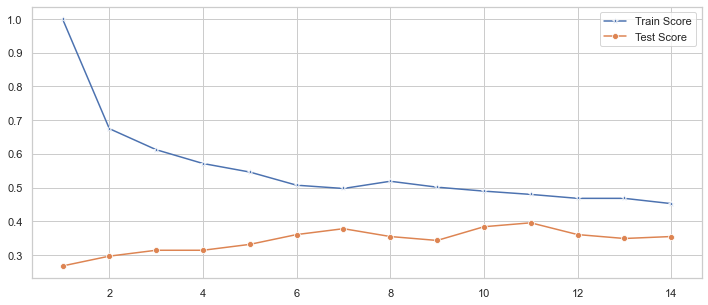
**Max test score 39.53488372093023 % and k = [11]**



**Euclidean**

**Max train score 100.0 % and k = [1]**

**Max test score 39.53488372093023 % and k = [11]**



**Oddly the graphs produced the same exact test score as well as the same k-value of 11.**

**When looking at the error Rate Graphs they are very similar in shape, but are clearly not same with minor positive and and negative spikes that would be considered not to be enough evidence to dissuade either of the three to be used nor cast away.**

**Conclusion**

***In this project we explored the songs and particular attributes and how they could impact or elicit particular moods that are typically used to describe emotions that the artists are trying to convey. I specifically looked at the attributes of Danceability, Energy, and Loudness, while determining the moods Sad, Calm, Happy, and Excited.***

***The research objective is to use a kNN model trained on the survey dataset, to identify which classification a mood can help elicit or enhance a mood from a listener and if there was any correlation of these particular attributes versus the mood.***

***The kNN multiple models will measure similarity to classify the observations. In particular the average music listener would typically not care about a number of attributes recorded in the original dataset such as valence, time\_measure, and musical key. Listeners neither have the patience to listen through an entire song that they did not care for. There was clustering in both Energy and Loudness compared to the song’s Danceability, as well as a strong positive correlation between Loudness and Energy. Unless a listener is specifically looking for a song for dancing, a user should be more focused on a songs’ Loudness and Energy levels. What was interesting to discover is that when comparing their average predictability they were nominally different. So much so that even though Manhattan Algorithm I would recommend a typical listener to use to find the song to suit their mood. It is far easier to calculate quickly without losing much due to error.***

***Research Questions:***

**I sought to answer the following questions:**

**1) Can we use a limited set of musical attributes: Energy, Danceability, and Loudness to predict the mood a song will elicit**

**2) Would it be possible to optimize predictability of a song’s mood by considering and comparing alternative distance algorithms - Manhattan, Euclidean, and Minkowski.**

***To answer the questions, I needed to engineer the Mood variable from a categorical to nominal value. This was used as the targeting variable for the kNN network, for a 80/20 split of training and test data. The model was run iteratively to optimize for the k, finding k=11 was the best accuracy score, without a risk of overfitting.***

***The classification report measured the quality of the prediction from the k-NN classification algorithm, and showed a precision (% of correct predictions) of 70.9% (Minkowski), 75% (Manhattan) , and 71.5% (Euclidean).***

***The cross validation showed a mean accuracy score of 74% (Minkowski), 73% (Manhattan) , and 74% (Euclidean).***

***Research Application:***

***I would implement the findings from this study by creating an intake survey for listeners to input a mood that they are feeling and what songs that would be suggested for the listener.***

***Critique:***

***An improvement could be to further analyze the three attributes and how they were quantified, especially Danceability. Danceability is an attribute that could potentially vary from listener to listener based on their particular genre of music that they prefer. Again it was also surprising to see Danceability had little impact on the other two attributes. If I had isolated the songs from the dataset based on the date of release I would have liked to see if there was a better correlation based on different generations of listeners and their definition of the emotions that songs impact them***

***Appendix:***

***List of variables in the revised dataset. dataset:***

* ***Attribute Min Max***
* ***popularity 0.000000 88.000***
* ***length 76773.000000 518373.000***
* ***danceability 0.078900 0.941***
* ***acousticness 0.000005 0.996***
* ***energy 0.001290 0.964***
* ***instrumentalness 0.000000 0.996***
* ***liveness 0.031800 0.966***
* ***valence 0.035300 0.977***
* ***loudness -42.018000 1.342***
* ***speechiness 0.023200 0.416***
* ***tempo 50.960000 217.950***
* ***key 0.000000 11.000***
* ***time\_signature 1.000000 5.000***
* ***mood\* 0.000000\* 3.000\****

***\* Note: Mood was altered from categorical to numeric values***

***0 = Sad 1 = Calm 2 = Happy 3 = Energetic***