

Individual Project : Popularity Score

For the individual project assigned in Data Mining 614 I was tasked to develop a predictive model that could help a music streaming company more advantageously determine whether certain songs are more likely to be popular upon future release. If this can be achieved, the company can make better use of its recommender systems and simultaneously increase revenue and lower costs. I took a strategic approach to incorporate three different types of classification in my analysis that included decision trees, logistic regressions, and k-nearest neighbor.

When implementing my decision tree analysis there were certain steps I took beforehand. After uploading the appropriate software packages, I then made sure to clean the data we were using by removing any missing values in the dataset. I then created a subset of all the available data in the dataset so that I could narrow my analysis using better explanatory variables and edited it accordingly to fit into our potential models (true, false, = 1, 0). A training set of 60% of the data and testing set of 40% of the data was then created. Decision tree modeling is a top-down method that uses statistical measures gained to select which test attributes should be used. I was genuinely shocked as to how poorly many of my models performed. None of the models I tried to fit for the data ever received an F1 score above 0.2. I did notice that lower values for sample splitting resulted in marginally better results for a fit, but the model outcomes were still too low for serious consideration.

There also seemed to be a soft cap for tree depth at around 15. Running cross-validation confirmed this. And when visualizing our trees there did not seem to be “too” many nodes that created high Gini levels, and yet, our results were still disappointing. The best fit I could achieve using decision tree modeling was with tree depth equal to 15 and sample splitting equal to 2. It yielded these results, recall: 0.17, precision: 0.17, F1_score: 0.17. I did get slightly higher results when I set the depth equal to 500, but this would have not been useful due to overfitting.

With my logistic regression models, the experience only worsened. I was expecting to have the most success using this type of classification. Logistic regression models are extremely useful when you are dealing with categorical data, especially if your dependent variable is categorical. There were 20 independent variables with which I was working. They ranged from the duration of the song, danceability, loudness, and genre, etc. Before fitting my initial model, I had to be sure to normalize the data since there were different measurements for the quantitative data. I then ran my first logistic regression model using all 20 independent variables and the variable “hot” (popularity above 75) as our dependent variable.

I was dissatisfied with those initial results, but after running a validation I finally began to see a glimmer of hope! The validation results came back with a precision score of 1.00! Now, granted the recall was only 0.01, at least it felt like progress. But after running a cross-validation all those numbers dropped to zero. The recall, precision, and F1 score for my logistic regression model were 0.00. This didn’t make much sense. There actually is a defining reason for these outcomes, more on that later. So, I decided to try building a more accurate model by removing all the variables that had no relevance in terms of correlation.

While two of the variables that I decided to keep in my second model did technically have p-values above 0.05, almost all of the 20 independent variables in the initial logistic regression model had p-values so high that I wanted to see if removing the really high ones would affect or rather lower the p-values of the independent variables that I did decide to keep in my second model. From the initial model I decided to keep the independent variables - explicit, with a p-value of 0.084, loudness, with a p-value of 0.068, and R&B which had a p-value of 0.007. After running the second model, the p-values for explicit and loudness both increased. After sequentially removing them and using the only variable left with any correlation to our dependent variable of popularity over 75, the final results produced another model with recall, precision, and F1 scores of 0.00.

The third type of classification I performed during my analysis was k-nearest neighbor. This is a classification rule assigned to a test sample to identify the category label of its nearest training samples, with k defining the number of neighbors, using some form of distance measurement. The results I got from these models were bad, just “less bad” than the other classification methods I used for my analysis. Since my k-nearest neighbor models seemed to be the lesser of three evils, the final model I chose for this project

would be a k-nearest neighbor model for prediction with $k = 1$. Yes, I understand that this could lead to misleading information because of overfitting. But we were tasked with choosing a final model, and from a financial perspective for the company, this model would produce the least amount of costs. It should be noted that in a real-world scenario, none of these models should be applied in a professional setting, and this is a very poor dataset to use for the purposes outlined by our client.

When initially running my KNN models, I first reloaded the original data since I had amended the dataset while running logistic regression, and then normalized the data. I began with a standard $k = 3$ model. This model produced a recall of 0.13, a precision of 0.25, and an F1 score of 0.17 after cross validation. I then decided to plot and try to find the best k that would fit the model. This wasn't immensely helpful; the plot indicated a high level of accuracy across a wide range of k values. After running a number of different k models, I finally decided that $k = 1$ produced the best results. My final model produced a recall of 0.28, a precision of 0.25, and an F1 score of 0.26. After prolonged and dubious considerations regarding how much this dataset failed to help me produce any meaningful information for my client, I decided that the problem might lie with how the data is constructed. I decided I would take one last approach and try something different.

For my final "final" model I decided to reengineer the data (I did not make any changes to any values in cells, all the reengineered data was taken from the original dataset). The idea to approach the project this way came to me when I noticed there were some songs in the data under the genre columns (pop, rock, hip hop, etc.) that did not have any 1s at all. I noticed this was because there was no column for metal music. So, I made a new column labeled "metal" and then for each row that had a metal tag in the genre column I placed a 1 in the metal column, otherwise I placed a 0. I did this so I could apply conditional formatting to the column to identify metal songs more easily. This could also have been done to add an additional explanatory variable to the dataset.

I then partitioned the data by genre and made a new dataset that only included rows for songs that had a "metal" tag in their genre column. The results produced a dataset with 65 rows that could be used to predict popularity for only the genre of metal music. There were 13 popular songs amongst the 65 records. I decided to change the parameters for popularity to a score of 70 to help prevent any overbalancing. Considering 70 is still a good score, and for the purpose of helping our client achieve a net profit instead of a net loss, this decision was warranted because it produced a result of 29 popular songs among the 65 records.

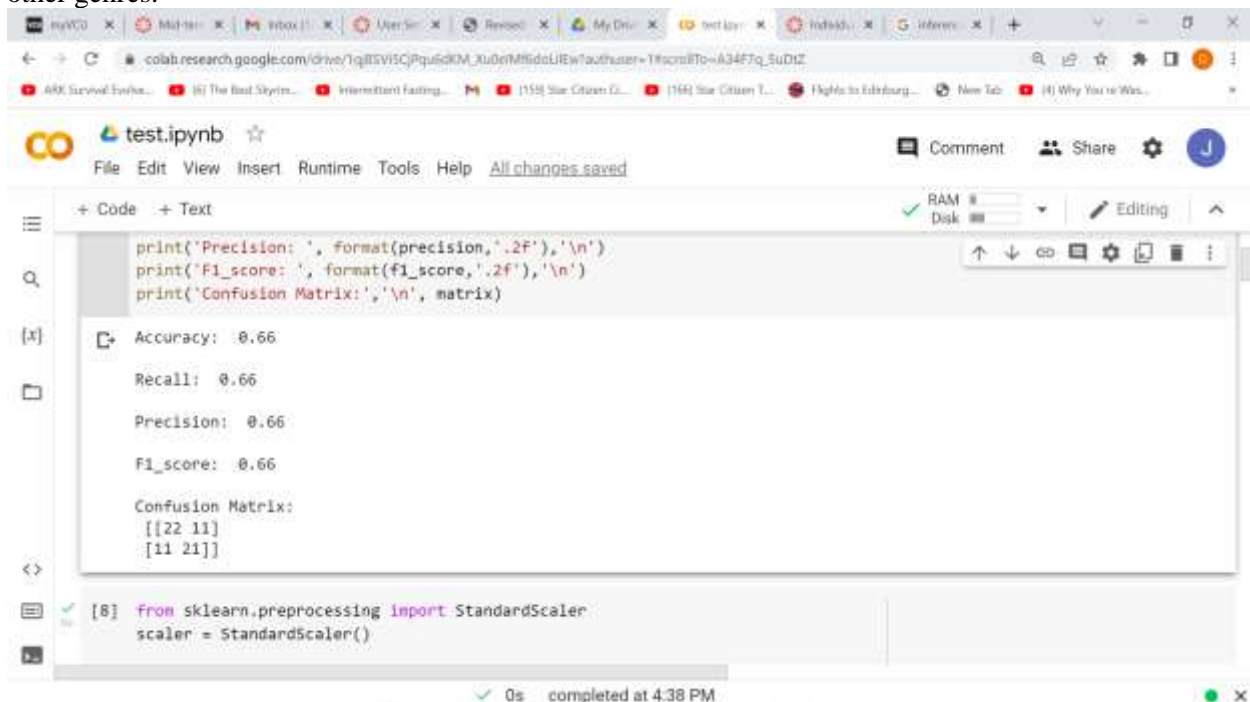
Amongst the explanatory variables indicated that we use, for the purposes of this dataset, I decided not to use hip hop, dance, folk, R&B, and Latin as explanatory variables. All the rows produced zeros for these columns so these variables would not give us very much additional information. I also decided to edit this dataset in excel beforehand, instead of editing it in python. I did this for two reasons. First, it is a smaller dataset, and I can keep track of my progress. Second, I did not want there to be any confusion regarding the first dataset I used which produced poor results. The code for the editing of explicit and removal of missing values only applies to the songs_utf.csv dataset. I changed true/false to 1/0 using the IF function and then copied those values to a new column labeled explicit and manually removed the variables stated above (it would have been easier to do it this way, but I actually decided to leave them in, just in case anyone wanted to view this dataset in its entirety; I excluded them from the subset I produced in python). I saved this dataset and named it "metal."

Considering how poor the results were with my previous data, I was optimistic for finding some meaningful insight using this new approach. First, I ran a decision tree with maximum depth set to 15 and minimum split set to 2. After cross-validation this model produced the following results: a recall of 0.66, a precision of 0.62, and an F1 score of 0.64. This is a huge improvement from our previous models. And while the accuracy for this model did drop to 0.63, which on the surface sounds bad, we must remember that there is no benefit associated with correctly predicting non-popular songs. We are only concerning ourselves with how many popular songs we can predict. So yes, I had found my best model so far. Increasing the minimum number of splits to 3 improved all our results across the board to 0.66.

After running a logistic regression model with my "metal" dataset, validation produced less favorable results. But our cross-validation produced remarkably similar results to our decision tree model

with an accuracy of 0.66, a recall of 0.59, a precision of 0.68, and an F1 score of 0.63. Now it was time for me to try KNN. First, I plotted and wanted to try and find the best k we could use for the model and used a range from 3 to 39. The plot showed that our highest level of accuracy was around k = 21. But after running models with different k's, I found that k = 5 was the best performing KNN model with an accuracy of 0.62, a recall 0.59, a precision of 0.61, and an F1 score of 0.60.

The final model I chose for my “metal” dataset is a decision tree model with max depth 8 and minimum split 3. Cross-validation confirmed the model produced a recall of 0.66, a precision of 0.66, and an F1 score of 0.66. Evaluation of the confusion matrix illustrated that the model produced 11 false negatives (\$1,200 cost each), 11 false positives (\$500 cost each), and 21 true positives (\$1,000 revenue each). $\$21,000 - \$13,200 - \$5,500 = \$2,300$, so this model does, indeed, produce a profit for this dataset (the inference of the values in the confusion matrix produced in the python code were confirmed by Professor Kim, python lists 0 before 1 which is the opposite of what was on the slides during our class sessions). Scaling this model appropriately with more metal records could produce even better results. This type of reengineering for the metal genre demonstrates that it is feasible to apply these same practices to other genres as well to achieve similar results. I **can not** speculate that the results for the metal genre will **certainly** produce the same favorable results for other genres, only that the success of this approach after dealing with so many failed models should warrant further investigation for the use of this approach for other genres.



```
print('Precision: ', format(precision, '.2f'), '\n')
print('F1_score: ', format(f1_score, '.2f'), '\n')
print('Confusion Matrix: ', '\n', matrix)
```

```
Accuracy: 0.66
Recall: 0.66
Precision: 0.66
F1_score: 0.66
Confusion Matrix:
[[22 11]
 [11 21]]
```

```
[8]: from sklearn.preprocessing import StandardScaler
     scaler = StandardScaler()
```

There were many things I observed about this dataset during my analysis. The first and most obvious thing that popped out at me, and in large part – it might have been the only strong indicator, was that there were many popular hip hop and R&B songs. Pop as a genre gets thrown around a lot and can vary in taste and sound. This did not come as much of a surprise. Most R&B songs are about love and relationships, and anyone in marketing can tell you, sex always sells. And, furthermore, most R&B singers happen to be quite attractive and this only compounds towards their popularity and the potential of the popularity of one of their songs. It would be worth testing to see if R&B should be removed from the dataset because it might be an obvious correlation.

I also noticed that we were instructed to remove “Artist” as one of our explanatory variables. On the surface, this sounds like we might be missing some useful information. There are some bands that develop cult followings, and those die-hard fans might support a particular song just because it was released

by their favorite band. Bands such as Creed and Nickelback had this effect on their fans. You also see this effect take hold in the film industry. That is why Dwayne “The Rock” Johnson gets paid so much for his movie roles. Kids will show up to see the movie if he is in it, regardless of whether it got good reviews or not.

Meanwhile, the most crippling aspect of the dataset was the imbalance of nonpopular songs to popular songs. This is why some of our initial models had very low recall scores but high accuracy scores (our initial logistic regression model exhibited this behavior). There were only 192 popular songs out of the 1500 entries. This left us with a popular to nonpopular ratio of 1 to 7.8 (roughly). The only reason those models had higher accuracy is because they were correctly predicting nonpopular songs, which is not helpful for the purposes of this assignment. There is no associated revenue for predicting nonpopular songs.

Something I could not wrap my head around conceptually is how this dataset might fail to account for culture shifts over time. Some of the things that influence pop culture can change drastically and quickly. Something that made a song popular in 2000 might not necessarily have the same effects on a song in 2022. And these types of influences would affect some of the more “raw” data such as duration or loudness. My point being is that I am not entirely certain that randomly sampling from this data would produce correlations that have fruitful meanings (since the records came out during different years). Just because one correlation is produced from the overall dataset does not necessarily mean that correlation would remain true for any specific given year, again, because of the rapid shifts associated with pop culture.

I do understand that is why the variable “year” was instructed to be excluded from the list of our explanatory variables. But it still does not change the fact that those particular songs still were released in different years under very different cultural circumstances. I also didn’t think that the popularity variable should be included in the dataset (you would still keep “hot” included, but how to measure it needs to be challenged). Yes, I understand we excluded it from our models, but the score itself is misleading in its nature. Some genres of music are naturally more popular than others, and vice versa, some genres are naturally less popular (folk music for example). If the idea for our client is to target certain customers and determine if they will buy a record in the future, then these popularity scores need to reflect how popular a song is amongst the people who actually listen to that genre of music and not how popular a song is amongst the entire population.

For example, a folk song using the dataset’s metric might receive a popularity score of 28. But if most people don’t like folk music to begin with, then that number is misleading. It just might be that of the people who do listen to folk music, 98% of those individuals do like the song. So, it seems most appropriate to me that building models to predict whether these songs will be popular or not should be partitioned by genre. And last, but not least, how would you really use the variable “energy” as a predictive variable? Those scores are subjective. They won’t reveal anything concrete when being included with quantitative variables that use more accurate scaling systems. Duration is a count, loudness is a count, tempo is a count, but the “energy” variable is just something that a person is perceiving. There needs to be more clarity on how that variable was recorded.

| | Accuracy | Recall | Precision | F1 Score |
|-------------------------------|----------|--------|-----------|----------|
| DT depth=4, spilt=5 | 0.86 | 0.02 | 0.13 | 0.04 |
| DT depth=5, spilt=2 | 0.85 | 0.03 | 0.14 | 0.04 |
| DT depth=15, spilt=2 | 0.79 | 0.17 | 0.17 | 0.17 |
| DT depth=500, spilt=2 | 0.78 | 0.21 | 0.19 | 0.20 |
| Log Regression | 0.87 | 0.00 | 0.00 | 0.00 |
| Log Reg w/var removed | 0.87 | 0.00 | 0.00 | 0.00 |
| KNN k=3 | 0.84 | 0.13 | 0.25 | 0.17 |
| KNN k=1 | 0.80 | 0.28 | 0.25 | 0.26 |
| DT(metal) depth=8, spilt=3 | 0.66 | 0.66 | 0.66 | 0.66 |
| Log Reg(metal) | 0.66 | 0.59 | 0.68 | 0.63 |
| KNN(metal) k=5 | 0.62 | 0.59 | 0.61 | 0.60 |

https://colab.research.google.com/drive/1FCWEre53uIn7RF5T-S_uEonSZ7ycCVef?usp=sharing

Individual Project

DO NOT LOAD THIS FIRST, THIS IS MY "METAL" DATASET THAT I REENGINEERED FOR THE PURPOSES OF ONLY PREDICTING WHETHER A METAL SONG IS POPULAR. TO FOLLOW ALONG WITH MY INITIAL FINDINGS IN MY REPORT PLEASE EXECUTE THE CODE LISTED UNDER "IMPORT DATA" FIRST. !!!

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
import sklearn.model_selection as ms
from sklearn import tree
from sklearn.metrics import classification_report, confusion_matrix
from IPython.display import Image
import pydotplus
import os
import statsmodels.api as sm
import sklearn.metrics as mt
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.model_selection import cross_val_predict
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
from sklearn.linear_model import LogisticRegression
import sklearn.metrics as mt
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt

file_path = '/content/drive/MyDrive/metal.csv'
df = pd.read_csv(file_path)

df.head()
```

| | song_name_len | duration_ms | explicit | danceability | energy | key | loudness | speechiness | acousticness | instrumentalness | ... | pop | rock |
|---|---------------|-------------|----------|--------------|--------|-----|----------|-------------|--------------|------------------|-----|-----|------|
| 0 | 12 | 224493 | 0 | 0.551 | 0.913 | 0 | -4.063 | 0.0466 | 0.026300 | 0.000013 | ... | 0 | 1 |
| 1 | 10 | 216880 | 0 | 0.556 | 0.864 | 3 | -5.870 | 0.0584 | 0.009580 | 0.000000 | ... | 0 | 1 |
| 2 | 18 | 321040 | 0 | 0.425 | 0.852 | 11 | -5.607 | 0.0460 | 0.017500 | 0.306000 | ... | 0 | 0 |
| 3 | 10 | 233933 | 0 | 0.545 | 0.865 | 11 | -5.708 | 0.0286 | 0.006640 | 0.000011 | ... | 1 | 1 |
| 4 | 24 | 236866 | 0 | 0.313 | 0.831 | 1 | -3.894 | 0.0404 | 0.000127 | 0.000341 | ... | 1 | 1 |

5 rows × 24 columns

```
df = df[["song_name_len", "duration_ms", "explicit", "danceability", "energy", "key", "loudness", "speechiness", "acousticness", "instrumentalness", "pop", "rock"]]
```

```
X = (df.iloc[:, :-1])
y = (df.iloc[:, -1])
y = y.astype('int')
X_train, X_test, y_train, y_test = ms.train_test_split(X, y, test_size = 0.4, random_state = 1)
```

```
dt_clf = DecisionTreeClassifier(criterion='gini', max_depth=8, min_samples_split = 3, random_state=0,)
dt_clf = dt_clf.fit(X_train, y_train)
y_pred = dt_clf.predict(X_test)
```

```
import sklearn.metrics as mt

print('Train_Accuracy: ', dt_clf.score(X_train, y_train),'\n')

accuracy = mt.accuracy_score(y_test, y_pred)
recall = mt.recall_score(y_test, y_pred)
precision = mt.precision_score(y_test, y_pred)
f1_score = mt.f1_score(y_test, y_pred)
matrix = mt.confusion_matrix(y_test, y_pred)

print('Accuracy: ', format(accuracy, '.2f'), '\n')
print('Recall: ', format(recall, '.2f'), '\n')
print('Precision: ', format(precision, '.2f'), '\n')
print('F1_score: ', format(f1_score, '.2f'), '\n')
print('Confusion Matrix:', '\n', matrix)
```

→ Train_Accuracy: 0.9743589743589743

Accuracy: 0.54

Recall: 0.38

Precision: 0.75

F1_score: 0.50

Confusion Matrix:

```
[[ 8  2]
 [10  6]]
```

```
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.model_selection import cross_val_predict
```

```
y_pred_cross = cross_val_predict(dt_clf, X, y, cv=10)
accuracy = mt.accuracy_score(y, y_pred_cross)
recall = mt.recall_score(y, y_pred_cross)
precision = mt.precision_score(y, y_pred_cross)
f1_score = mt.f1_score(y, y_pred_cross)
matrix = mt.confusion_matrix(y, y_pred_cross)
```

```
print('Accuracy: ', format(accuracy, '.2f'), '\n')
print('Recall: ', format(recall, '.2f'), '\n')
print('Precision: ', format(precision, '.2f'), '\n')
print('F1_score: ', format(f1_score, '.2f'), '\n')
print('Confusion Matrix:', '\n', matrix)
```

→ Accuracy: 0.66

Recall: 0.66

Precision: 0.66

F1_score: 0.66

Confusion Matrix:

```
[[22 11]
 [11 21]]
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

```
X = (df.iloc[:, :-1])
y = (df.iloc[:, -1])
y = y.astype('int')
```

```
X = scaler.fit_transform(X)
```

```
X_train, X_test, y_train, y_test = ms.train_test_split(X, y, test_size = 0.4, random_state = 1)
```

```
from sklearn.linear_model import LogisticRegression
import sklearn.metrics as mt
```

```
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

```
print('Train_Accuracy: ', model.score(X_train, y_train), '\n')
```

Train_Accuracy: 0.7948717948717948

```
model = sm.Logit(y_train, X_train)
results = model.fit(method = "newton")
features = list(df.iloc[:, 0:-1].columns)
results.summary(xname=features)
```

Optimization terminated successfully.
Current function value: 0.293300
Iterations 10

Logit Regression Results

| | | | |
|-------------------------|------------------|--------------------------|----------|
| Dep. Variable: | hot | No. Observations: | 39 |
| Model: | Logit | Df Residuals: | 24 |
| Method: | MLE | Df Model: | 14 |
| Date: | Wed, 26 Oct 2022 | Pseudo R-squ.: | 0.5667 |
| Time: | 20:17:42 | Log-Likelihood: | -11.439 |
| converged: | True | LL-Null: | -26.401 |
| Covariance Type: | nonrobust | LLR p-value: | 0.007816 |

| | coef | std err | z | P> z | [0.025 | 0.975] |
|-------------------------|---------|---------|--------|-------|---------|--------|
| song_name_len | -7.4500 | 4.072 | -1.829 | 0.067 | -15.431 | 0.531 |
| duration_ms | -2.1818 | 1.505 | -1.450 | 0.147 | -5.132 | 0.768 |
| explicit | 1.5254 | 1.227 | 1.244 | 0.214 | -0.879 | 3.930 |
| danceability | 4.3271 | 2.349 | 1.842 | 0.065 | -0.277 | 8.931 |
| energy | -3.2715 | 2.269 | -1.442 | 0.149 | -7.719 | 1.176 |
| key | -3.3751 | 2.033 | -1.660 | 0.097 | -7.359 | 0.609 |
| loudness | 3.7622 | 2.087 | 1.803 | 0.071 | -0.328 | 7.852 |
| speechiness | 4.5436 | 2.450 | 1.855 | 0.064 | -0.258 | 9.345 |
| acousticness | 2.5921 | 3.764 | 0.689 | 0.491 | -4.784 | 9.968 |
| instrumentalness | 8.2849 | 9.545 | 0.868 | 0.385 | -10.423 | 26.993 |
| liveness | -5.6904 | 3.132 | -1.817 | 0.069 | -11.829 | 0.448 |
| valence | -0.1513 | 1.144 | -0.132 | 0.895 | -2.394 | 2.091 |
| tempo | 3.5034 | 2.036 | 1.721 | 0.085 | -0.486 | 7.493 |
| pop | -2.2470 | 1.258 | -1.786 | 0.074 | -4.713 | 0.219 |
| rock | -3.8822 | 2.294 | -1.692 | 0.091 | -8.379 | 0.614 |

Possibly complete quasi-separation: A fraction 0.26 of observations can be perfectly predicted. This might indicate that there is complete

```
accuracy = mt.accuracy_score(y_test, y_pred)
recall = mt.recall_score(y_test, y_pred)
precision = mt.precision_score(y_test, y_pred)
f1_score = mt.f1_score(y_test, y_pred)
matrix = mt.confusion_matrix(y_test, y_pred)

print('Accuracy: ', format(accuracy, '.2f'), '\n')
print('Recall: ', format(recall, '.2f'), '\n')
print('Precision: ', format(precision, '.2f'), '\n')
print('F1_score: ', format(f1_score, '.2f'), '\n')
print('Confusion Matrix:', '\n', matrix)
```

Accuracy: 0.42

Recall: 0.38

Precision: 0.55

F1_score: 0.44

Confusion Matrix:

```
[[ 5  5]
 [10  6]]
```



```
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.model_selection import cross_val_predict
```

```
model = LogisticRegression()
```

```
y_pred_cross = cross_val_predict(model, X, y, cv=10)
accuracy = mt.accuracy_score(y, y_pred_cross)
recall = mt.recall_score(y, y_pred_cross)
precision = mt.precision_score(y, y_pred_cross)
f1_score = mt.f1_score(y, y_pred_cross)
matrix = mt.confusion_matrix(y, y_pred_cross)
```

```
print('Accuracy: ', format(accuracy, '.2f'), '\n')
print('Recall: ', format(recall, '.2f'), '\n')
print('Precision: ', format(precision, '.2f'), '\n')
print('F1_score: ', format(f1_score, '.2f'), '\n')
print('Confusion Matrix:', '\n', matrix)
```

```
Accuracy: 0.66
```

```
Recall: 0.59
```

```
Precision: 0.68
```

```
F1_score: 0.63
```

```
Confusion Matrix:
```

```
[[24 9]
```

```
[13 19]]
```

```
import matplotlib.pyplot as plt
```

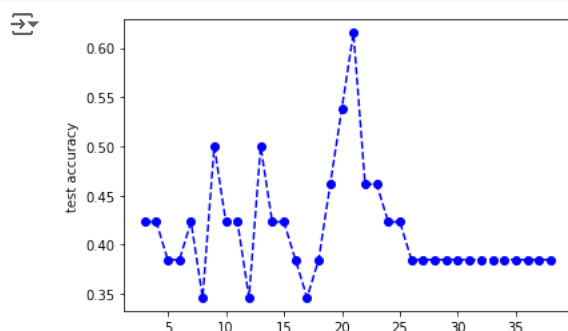
```
k_range = range(3,39)
```

```
accuracy_list = []
```

```
for k in k_range:
    model = KNeighborsClassifier(n_neighbors = k)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy_list.append(mt.accuracy_score(y_test, y_pred))
```

```
# k = sqrt(N)
```

```
plt.plot(k_range, accuracy_list, 'o--', color = 'blue')
plt.xlabel("k")
plt.ylabel("test accuracy")
plt.show()
```



```
model = KNeighborsClassifier(n_neighbors = 5)
model.fit(X_train, y_train)
```

```
y_pred = model.predict(X_test)
print('Train_Accuracy: ', model.score(X_train, y_train), '\n')
```

```
Train_Accuracy: 0.7948717948717948
```

```

accuracy = mt.accuracy_score(y_test, y_pred)
recall = mt.recall_score(y_test, y_pred)
precision = mt.precision_score(y_test, y_pred)
f1_score = mt.f1_score(y_test, y_pred)
matrix = mt.confusion_matrix(y_test, y_pred)

print('Accuracy: ', format(accuracy, '.2f'), '\n')
print('Recall: ', format(recall, '.2f'), '\n')
print('Precision: ', format(precision, '.2f'), '\n')
print('F1_score: ', format(f1_score, '.2f'), '\n')
print('Confusion Matrix:', '\n', matrix)

```

➦ Accuracy: 0.38

Recall: 0.38

Precision: 0.50

F1_score: 0.43

Confusion Matrix:

[[4 6]

[10 6]]

```

model = KNeighborsClassifier(n_neighbors = 5)

y_pred_cross = cross_val_predict(model, X, y, cv=10)
accuracy = mt.accuracy_score(y, y_pred_cross)
recall = mt.recall_score(y, y_pred_cross)
precision = mt.precision_score(y, y_pred_cross)
f1_score = mt.f1_score(y, y_pred_cross)
matrix = mt.confusion_matrix(y, y_pred_cross)

print('Accuracy: ', format(accuracy, '.2f'), '\n')
print('Recall: ', format(recall, '.2f'), '\n')
print('Precision: ', format(precision, '.2f'), '\n')
print('F1_score: ', format(f1_score, '.2f'), '\n')
print('Confusion Matrix:', '\n', matrix)

```

➦ Accuracy: 0.62

Recall: 0.59

Precision: 0.61

F1_score: 0.60

Confusion Matrix:

[[21 12]

[13 19]]

Import Data


```

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
import sklearn.model_selection as ms
from sklearn import tree
from sklearn.metrics import classification_report, confusion_matrix
from IPython.display import Image
import pydotplus
import os
import statsmodels.api as sm

file_path = '/content/drive/MyDrive/songs_utf(backup if other fails).csv'
df = pd.read_csv(file_path)


df.head()

```




| | artist | song | song_name_len | duration_ms | explicit | year | popularity | hot | danceability | energy | ... | valence | tempo | genre | pr |
|---|----------------|------------------------|---------------|-------------|----------|--------|------------|-----|--------------|--------|-----|---------|---------|--------------|----|
| 0 | Britney Spears | Oops!...I Did It Again | 22.0 | 211160.0 | False | 2000.0 | 77.0 | 1.0 | 0.751 | 0.834 | ... | 0.894 | 95.053 | pop | 1 |
| 1 | blink-182 | All The Small Things | 20.0 | 167066.0 | False | 1999.0 | 79.0 | 1.0 | 0.434 | 0.897 | ... | 0.684 | 148.726 | rock, pop | 1 |
| 2 | Faith Hill | Breathe | 7.0 | 250546.0 | False | 1999.0 | 66.0 | 0.0 | 0.529 | 0.496 | ... | 0.278 | 136.859 | pop, country | 1 |
| 3 | Bon Jovi | It's My Life | 12.0 | 224493.0 | False | 2000.0 | 78.0 | 1.0 | 0.551 | 0.913 | ... | 0.544 | 119.992 | rock, metal | 0 |
| 4 | *NSYNC | Bye Bye Bye | 11.0 | 200560.0 | False | 2000.0 | 65.0 | 0.0 | 0.614 | 0.928 | ... | 0.879 | 172.656 | pop | 1 |

5 rows × 27 columns




Remove Missing Values

```
display(df.isna().sum())
df = df.dropna()
```




| | |
|------------------|-----|
| artist | 499 |
| song | 499 |
| song_name_len | 499 |
| duration_ms | 499 |
| explicit | 499 |
| year | 499 |
| popularity | 499 |
| hot | 499 |
| danceability | 499 |
| energy | 499 |
| key | 499 |
| loudness | 499 |
| mode | 499 |
| speechiness | 499 |
| acousticness | 499 |
| instrumentalness | 499 |
| liveness | 499 |
| valence | 499 |
| tempo | 499 |
| genre | 499 |
| pop | 499 |
| rock | 499 |
| hiphop | 499 |
| dance | 499 |
| folk | 499 |
| rnb | 499 |
| latin | 499 |



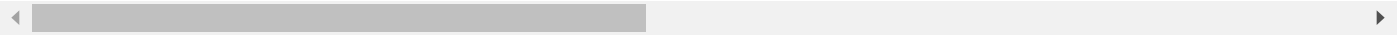
Create Database w/Explanitory Variables

```
df = df[["song_name_len", "duration_ms", "explicit", "danceability", "energy", "key", "loudness", "speechiness", "acousticness", "instrumentalness", "valence"]]
df.head()
```



| | song_name_len | duration_ms | explicit | danceability | energy | key | loudness | speechiness | acousticness | instrumentalness | ... | valence |
|---|---------------|-------------|----------|--------------|--------|-----|----------|-------------|--------------|------------------|-----|---------|
| 0 | 22.0 | 211160.0 | False | 0.751 | 0.834 | 1.0 | -5.444 | 0.0437 | 0.3000 | 0.000018 | ... | 0.894 |
| 1 | 20.0 | 167066.0 | False | 0.434 | 0.897 | 0.0 | -4.918 | 0.0488 | 0.0103 | 0.000000 | ... | 0.684 |
| 2 | 7.0 | 250546.0 | False | 0.529 | 0.496 | 7.0 | -9.007 | 0.0290 | 0.1730 | 0.000000 | ... | 0.278 |
| 3 | 12.0 | 224493.0 | False | 0.551 | 0.913 | 0.0 | -4.063 | 0.0466 | 0.0263 | 0.000013 | ... | 0.544 |
| 4 | 11.0 | 200560.0 | False | 0.614 | 0.928 | 8.0 | -4.806 | 0.0516 | 0.0408 | 0.001040 | ... | 0.879 |

5 rows × 21 columns



Edit Explicit

```
df.explicit = df.explicit.replace('False',0)
df.explicit = df.explicit.replace('True',1)

display(df.head() )
display(df['hot'].value_counts())
df.hot = df.hot.astype(int)
```

| | song_name_len | duration_ms | explicit | danceability | energy | key | loudness | speechiness | acousticness | instrumentalness | ... | valence |
|---|---------------|-------------|----------|--------------|--------|-----|----------|-------------|--------------|------------------|-----|---------|
| 0 | 22.0 | 211160.0 | False | 0.751 | 0.834 | 1.0 | -5.444 | 0.0437 | 0.3000 | 0.000018 | ... | 0.894 |
| 1 | 20.0 | 167066.0 | False | 0.434 | 0.897 | 0.0 | -4.918 | 0.0488 | 0.0103 | 0.000000 | ... | 0.684 |
| 2 | 7.0 | 250546.0 | False | 0.529 | 0.496 | 7.0 | -9.007 | 0.0290 | 0.1730 | 0.000000 | ... | 0.278 |
| 3 | 12.0 | 224493.0 | False | 0.551 | 0.913 | 0.0 | -4.063 | 0.0466 | 0.0263 | 0.000013 | ... | 0.544 |
| 4 | 11.0 | 200560.0 | False | 0.614 | 0.928 | 8.0 | -4.806 | 0.0516 | 0.0408 | 0.001040 | ... | 0.879 |

5 rows × 21 columns

0.0 1308
1.0 192
Name: hot, dtype: int64

```
df.corrwith(df["hot"])
```

| | |
|------------------|-----------|
| song_name_len | -0.055019 |
| duration_ms | -0.008572 |
| danceability | -0.034115 |
| energy | 0.027434 |
| key | -0.071302 |
| loudness | 0.051536 |
| speechiness | -0.037132 |
| acousticness | -0.019968 |
| instrumentalness | -0.029992 |
| liveness | -0.006801 |
| valence | -0.025767 |
| tempo | -0.013699 |
| pop | -0.094597 |
| rock | 0.130465 |
| hiphop | -0.016335 |
| dance | -0.056548 |
| folk | -0.000932 |
| rnb | -0.134866 |
| latin | 0.039824 |
| hot | 1.000000 |

dtype: float64

Partition the Data 60% - 40%

```
X = (df.iloc[:, :-1])
y = (df.iloc[:, -1])
y = y.astype('int')
X_train, X_test, y_train, y_test = ms.train_test_split(X, y, test_size = 0.4, random_state = 1)
```

Fit a Decision Tree

```
dt_clf = DecisionTreeClassifier(criterion='gini', max_depth=4, min_samples_split = 5, random_state=0,)

dt_clf = dt_clf.fit(X_train, y_train)

y_pred = dt_clf.predict(X_test)
```

Visualize

```

import io
import pydot
from IPython.core.display import Image
from sklearn.tree import export_graphviz
import matplotlib as mpl

def draw_decision_tree(model):
    dot_buf = io.StringIO()
    export_graphviz(model, out_file=dot_buf, feature_names=feature_names)
    graph = pydot.graph_from_dot_data(dot_buf.getvalue())[0]
    image = graph.create_png()
    return Image(image)

def plot_decision_regions(X, y, model, title):
    resolution = 0.01
    markers = ('s', '^', 'o')
    colors = ('red', 'blue', 'lightgreen')
    cmap = mpl.colors.ListedColormap(colors)

    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                           np.arange(x2_min, x2_max, resolution))

    Z = model.predict(
        np.array([xx1.ravel(), xx2.ravel()]).T).reshape(xx1.shape)

    plt.contour(xx1, xx2, Z, cmap=mpl.colors.ListedColormap(['k']))
    plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1], alpha=0.8,
                    c=[cmap(idx)], marker=markers[idx], s=80, label=cl)

    plt.xlabel(data.feature_names[2])
    plt.ylabel(data.feature_names[3])
    plt.legend(loc='upper left')
    plt.title(title)

    return Z

```

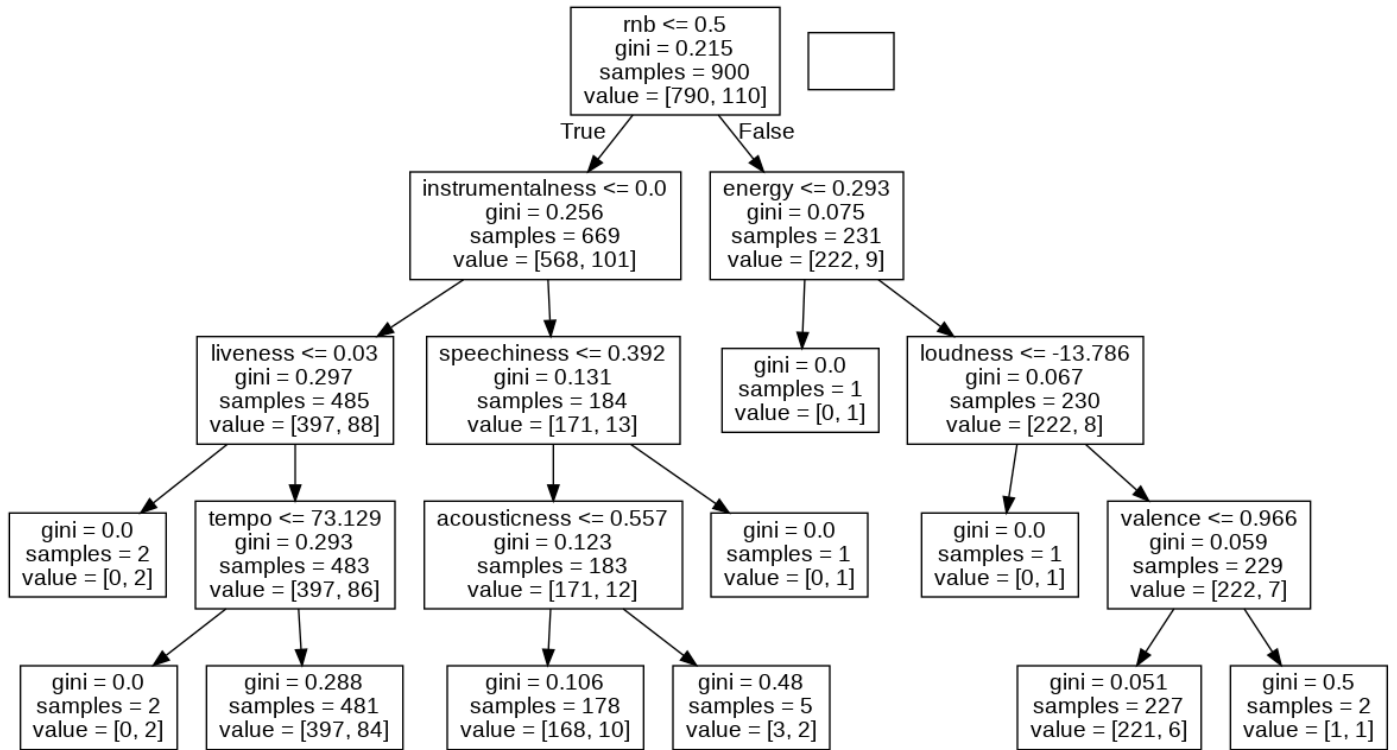
```

feature_names = df.columns.tolist()
feature_names = feature_names[0:20]
target_name = np.array(['1', '0'])

dt_dot_data = tree.export_graphviz(dt_clf, out_file = None,
                                   feature_names = feature_names,
                                   class_names = target_name,
                                   filled = True, rounded = True,
                                   special_characters = True)

dt_graph = pydotplus.graph_from_dot_data
draw_decision_tree(dt_clf)

```



```

import sklearn.metrics as mt

print('Train_Accuracy: ', dt_clf.score(X_train, y_train), '\n')

accuracy = mt.accuracy_score(y_test, y_pred)
recall = mt.recall_score(y_test, y_pred)
precision = mt.precision_score(y_test, y_pred)
f1_score = mt.f1_score(y_test, y_pred)
matrix = mt.confusion_matrix(y_test, y_pred)

print('Accuracy: ', format(accuracy, '.2f'), '\n')
print('Recall: ', format(recall, '.2f'), '\n')
print('Precision: ', format(precision, '.2f'), '\n')
print('F1_score: ', format(f1_score, '.2f'), '\n')
print('Confusion Matrix:', '\n', matrix)

```

Train_Accuracy: 0.8855555555555555

Accuracy: 0.85

Recall: 0.00

Precision: 0.00

F1_score: 0.00

Confusion Matrix:
[[510 8]
[82 0]]


```

from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.model_selection import cross_val_predict

y_pred_cross = cross_val_predict(dt_clf, X, y, cv=10)
accuracy = mt.accuracy_score(y, y_pred_cross)
recall = mt.recall_score(y, y_pred_cross)
precision = mt.precision_score(y, y_pred_cross)
f1_score = mt.f1_score(y, y_pred_cross)
matrix = mt.confusion_matrix(y, y_pred_cross)

print('Accuracy: ', format(accuracy, '.2f'), '\n')
print('Recall: ', format(recall, '.2f'), '\n')
print('Precision: ', format(precision, '.2f'), '\n')
print('F1_score: ', format(f1_score, '.2f'), '\n')
print('Confusion Matrix:', '\n', matrix)

```

 Accuracy: 0.86
 Recall: 0.02
 Precision: 0.13
 F1_score: 0.04
 Confusion Matrix:
 [[1282 26]
 [188 4]]

New Tree

```

dt_clf = DecisionTreeClassifier(criterion='gini', max_depth=5, min_samples_split = 2, random_state=0)

dt_clf = dt_clf.fit(X_train, y_train)

y_pred = dt_clf.predict(X_test)

```

```


import sklearn.metrics as mt

print('Train_Accuracy: ', dt_clf.score(X_train, y_train), '\n')

accuracy = mt.accuracy_score(y_test, y_pred)
recall = mt.recall_score(y_test, y_pred)
precision = mt.precision_score(y_test, y_pred)
f1_score = mt.f1_score(y_test, y_pred)
matrix = mt.confusion_matrix(y_test, y_pred)

print('Accuracy: ', format(accuracy, '.2f'), '\n')
print('Recall: ', format(recall, '.2f'), '\n')
print('Precision: ', format(precision, '.2f'), '\n')
print('F1_score: ', format(f1_score, '.2f'), '\n')
print('Confusion Matrix:', '\n', matrix)

```

 Train_Accuracy: 0.8888888888888888
 Accuracy: 0.85
 Recall: 0.01
 Precision: 0.10
 F1_score: 0.02
 Confusion Matrix:
 [[509 9]
 [81 1]]

```
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.model_selection import cross_val_predict

y_pred_cross = cross_val_predict(dt_clf, X, y, cv=10)
accuracy = mt.accuracy_score(y, y_pred_cross)
recall = mt.recall_score(y, y_pred_cross)
precision = mt.precision_score(y, y_pred_cross)
f1_score = mt.f1_score(y, y_pred_cross)
matrix = mt.confusion_matrix(y, y_pred_cross)

print('Accuracy: ', format(accuracy, '.2f'), '\n')
print('Recall: ', format(recall, '.2f'), '\n')
print('Precision: ', format(precision, '.2f'), '\n')
print('F1_score: ', format(f1_score, '.2f'), '\n')
print('Confusion Matrix:', '\n', matrix)
```

```
➦ Accuracy:  0.85

Recall:  0.03

Precision:  0.14

F1_score:  0.04

Confusion Matrix:
[[1276  32]
 [ 187   5]]
```

We are getting horrible results, go deeper

```
dt_clf = DecisionTreeClassifier(criterion='gini', max_depth=15, min_samples_split = 2, random_state=0)

dt_clf = dt_clf.fit(X_train, y_train)

y_pred = dt_clf.predict(X_test)
```

```
import sklearn.metrics as mt

print('Train_Accuracy: ', dt_clf.score(X_train, y_train), '\n')

accuracy = mt.accuracy_score(y_test, y_pred)
recall = mt.recall_score(y_test, y_pred)
precision = mt.precision_score(y_test, y_pred)
f1_score = mt.f1_score(y_test, y_pred)
matrix = mt.confusion_matrix(y_test, y_pred)

print('Accuracy: ', format(accuracy, '.2f'), '\n')
print('Recall: ', format(recall, '.2f'), '\n')
print('Precision: ', format(precision, '.2f'), '\n')
print('F1_score: ', format(f1_score, '.2f'), '\n')
print('Confusion Matrix:', '\n', matrix)
```

```
➦ Train_Accuracy:  0.9822222222222222

Accuracy:  0.77

Recall:  0.13

Precision:  0.14

F1_score:  0.14

Confusion Matrix:
[[453  65]
 [ 71  11]]
```



```
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.model_selection import cross_val_predict

y_pred_cross = cross_val_predict(dt_clf, X, y, cv=10)
accuracy = mt.accuracy_score(y, y_pred_cross)
recall = mt.recall_score(y, y_pred_cross)
precision = mt.precision_score(y, y_pred_cross)
f1_score = mt.f1_score(y, y_pred_cross)
matrix = mt.confusion_matrix(y, y_pred_cross)

print('Accuracy: ', format(accuracy, '.2f'), '\n')
print('Recall: ', format(recall, '.2f'), '\n')
print('Precision: ', format(precision, '.2f'), '\n')
print('F1_score: ', format(f1_score, '.2f'), '\n')
print('Confusion Matrix:', '\n', matrix)
```

➦ Accuracy: 0.79

Recall: 0.17

Precision: 0.17

F1_score: 0.17

Confusion Matrix:
[[1146 162]
[159 33]]

```
dt_clf = DecisionTreeClassifier(criterion='gini', max_depth=500, min_samples_split = 2, random_state=0,)

dt_clf = dt_clf.fit(X_train, y_train)

y_pred = dt_clf.predict(X_test)
```

```
import sklearn.metrics as mt

print('Train_Accuracy: ', dt_clf.score(X_train, y_train), '\n')

accuracy = mt.accuracy_score(y_test, y_pred)
recall = mt.recall_score(y_test, y_pred)
precision = mt.precision_score(y_test, y_pred)
f1_score = mt.f1_score(y_test, y_pred)
matrix = mt.confusion_matrix(y_test, y_pred)

print('Accuracy: ', format(accuracy, '.2f'), '\n')
print('Recall: ', format(recall, '.2f'), '\n')
print('Precision: ', format(precision, '.2f'), '\n')
print('F1_score: ', format(f1_score, '.2f'), '\n')
print('Confusion Matrix:', '\n', matrix)
```

➦ Train_Accuracy: 1.0

Accuracy: 0.79

Recall: 0.17

Precision: 0.19

F1_score: 0.18

Confusion Matrix:
[[459 59]
[68 14]]

```

from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.model_selection import cross_val_predict

y_pred_cross = cross_val_predict(dt_clf, X, y, cv=10)
accuracy = mt.accuracy_score(y, y_pred_cross)
recall = mt.recall_score(y, y_pred_cross)
precision = mt.precision_score(y, y_pred_cross)
f1_score = mt.f1_score(y, y_pred_cross)
matrix = mt.confusion_matrix(y, y_pred_cross)

print('Accuracy: ', format(accuracy, '.2f'), '\n')
print('Recall: ', format(recall, '.2f'), '\n')
print('Precision: ', format(precision, '.2f'), '\n')
print('F1_score: ', format(f1_score, '.2f'), '\n')
print('Confusion Matrix:', '\n', matrix)

```

```

↗ Accuracy: 0.78

Recall: 0.21

Precision: 0.19

F1_score: 0.20

Confusion Matrix:
[[1134 174]
 [ 152  40]]

```

Lets Build a Logistic Regression Model

Normalize and fit Training and Testing

```

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

X = (df.iloc[:, :-1])
y = (df.iloc[:, -1])
y = y.astype('int')

X = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = ms.train_test_split(X, y, test_size = 0.4, random_state = 1)

```

Fit the Model

```

from sklearn.linear_model import LogisticRegression
import sklearn.metrics as mt

model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print('Train_Accuracy: ', model.score(X_train, y_train), '\n')

```

```

↗ Train_Accuracy: 0.8777777777777778

```

Table

```

model = sm.Logit(y_train, X_train)
results = model.fit(method = "newton")
features = list(df.iloc[:, 0:-1].columns)
results.summary(xname=features)

```

Optimization terminated successfully.
Current function value: 0.679692
Iterations 4

Logit Regression Results

| | | | |
|-------------------------|------------------|--------------------------|---------|
| Dep. Variable: | hot | No. Observations: | 900 |
| Model: | Logit | Df Residuals: | 880 |
| Method: | MLE | Df Model: | 19 |
| Date: | Mon, 24 Oct 2022 | Pseudo R-squ.: | -0.8304 |
| Time: | 00:47:09 | Log-Likelihood: | -611.72 |
| converged: | True | LL-Null: | -334.20 |
| Covariance Type: | nonrobust | LLR p-value: | 1.000 |

| | coef | std err | z | P> z | [0.025 0.975] |
|-------------------------|---------|---------|--------|-------|---------------|
| song_name_len | -0.1012 | 0.070 | -1.444 | 0.149 | -0.239 0.036 |
| duration_ms | 0.0123 | 0.071 | 0.173 | 0.862 | -0.127 0.151 |
| explicit | -0.1426 | 0.082 | -1.730 | 0.084 | -0.304 0.019 |
| danceability | 0.0264 | 0.087 | 0.303 | 0.762 | -0.144 0.197 |
| energy | -0.1099 | 0.103 | -1.062 | 0.288 | -0.313 0.093 |
| key | -0.0699 | 0.069 | -1.016 | 0.309 | -0.205 0.065 |
| loudness | 0.1612 | 0.088 | 1.822 | 0.068 | -0.012 0.335 |
| speechiness | 0.0199 | 0.082 | 0.244 | 0.807 | -0.140 0.180 |
| acousticness | 0.0118 | 0.077 | 0.153 | 0.878 | -0.140 0.163 |
| instrumentalness | -0.0605 | 0.077 | -0.790 | 0.430 | -0.211 0.090 |
| liveness | -0.0026 | 0.070 | -0.036 | 0.971 | -0.140 0.135 |
| valence | -0.0559 | 0.087 | -0.646 | 0.518 | -0.225 0.114 |
| tempo | -0.0698 | 0.071 | -0.979 | 0.327 | -0.209 0.070 |
| pop | -0.0598 | 0.073 | -0.822 | 0.411 | -0.202 0.083 |
| rock | 0.0544 | 0.077 | 0.710 | 0.477 | -0.096 0.204 |
| hiphop | 0.0695 | 0.086 | 0.811 | 0.417 | -0.098 0.237 |
| dance | -0.0643 | 0.075 | -0.858 | 0.391 | -0.211 0.083 |
| folk | -0.0691 | 0.061 | -1.125 | 0.261 | -0.189 0.051 |
| rnb | -0.2094 | 0.077 | -2.709 | 0.007 | -0.361 -0.058 |

```
accuracy = mt.accuracy_score(y_test, y_pred)
recall = mt.recall_score(y_test, y_pred)
precision = mt.precision_score(y_test, y_pred)
f1_score = mt.f1_score(y_test, y_pred)
matrix = mt.confusion_matrix(y_test, y_pred)

print('Accuracy: ', format(accuracy, '.2f'), '\n')
print('Recall: ', format(recall, '.2f'), '\n')
print('Precision: ', format(precision, '.2f'), '\n')
print('F1_score: ', format(f1_score, '.2f'), '\n')
print('Confusion Matrix:', '\n', matrix)
```

Accuracy: 0.86

Recall: 0.01

Precision: 1.00

F1_score: 0.02

Confusion Matrix:

```
[[518  0]
 [ 81  1]]
```

```

from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.model_selection import cross_val_predict

model = LogisticRegression()

y_pred_cross = cross_val_predict(model, X, y, cv=10)
accuracy = mt.accuracy_score(y, y_pred_cross)
recall = mt.recall_score(y, y_pred_cross)
precision = mt.precision_score(y, y_pred_cross)
f1_score = mt.f1_score(y, y_pred_cross)
matrix = mt.confusion_matrix(y, y_pred_cross)

print('Accuracy: ', format(accuracy, '.2f'), '\n')
print('Recall: ', format(recall, '.2f'), '\n')
print('Precision: ', format(precision, '.2f'), '\n')
print('F1_score: ', format(f1_score, '.2f'), '\n')
print('Confusion Matrix:', '\n', matrix)

```

Accuracy: 0.87

Recall: 0.00

Precision: 0.00

F1_score: 0.00

Confusion Matrix:
 [[1307 1]
 [192 0]]

let's build a model with explicit, loudness, and rnb

```

df = df[["explicit", "loudness", "rnb", "hot"]]

X = (df.iloc[:, :-1])
y = (df.iloc[:, -1])
y = y.astype('int')

X = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = ms.train_test_split(X, y, test_size = 0.4, random_state = 1)

model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print('Train_Accuracy: ', model.score(X_train, y_train), '\n')

```

Train_Accuracy: 0.8777777777777778

```

model = sm.Logit(y_train, X_train)
results = model.fit(method = "newton")
features = list(df.iloc[:, 0:-1].columns)
results.summary(xname=features)

```

Optimization terminated successfully.
 Current function value: 0.687862
 Iterations 4

Logit Regression Results

| | | | |
|-------------------------|------------------|--------------------------|---------|
| Dep. Variable: | hot | No. Observations: | 900 |
| Model: | Logit | Df Residuals: | 897 |
| Method: | MLE | Df Model: | 2 |
| Date: | Sun, 23 Oct 2022 | Pseudo R-squ.: | -0.8524 |
| Time: | 22:37:00 | Log-Likelihood: | -619.08 |
| converged: | True | LL-Null: | -334.20 |
| Covariance Type: | nonrobust | LLR p-value: | 1.000 |

| | coef | std err | z | P> z | [0.025 0.975] |
|----------|---------|---------|--------|-------|---------------|
| explicit | -0.0758 | 0.067 | -1.132 | 0.258 | -0.207 0.055 |
| loudness | 0.0869 | 0.067 | 1.298 | 0.194 | -0.044 0.218 |
| rnb | 0.0415 | 0.067 | 0.607 | 0.540 | -0.085 0.168 |

remove explicit

```
df = df[["loudness", "rnb", "hot"]]

X = (df.iloc[:, :-1])
y = (df.iloc[:, -1])
y = y.astype('int')

X = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = ms.train_test_split(X, y, test_size = 0.4, random_state = 1)

model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print('Train_Accuracy: ', model.score(X_train, y_train), '\n')
```

↗ Train_Accuracy: 0.8777777777777778

```
model = sm.Logit(y_train, X_train)
results = model.fit(method = "newton")
features = list(df.iloc[:, 0:-1].columns)
results.summary(xname=features)
```

↗ Optimization terminated successfully.
Current function value: 0.688575
Iterations 4

Logit Regression Results

| | | | |
|-------------------------|------------------|--------------------------|---------|
| Dep. Variable: | hot | No. Observations: | 900 |
| Model: | Logit | Df Residuals: | 898 |
| Method: | MLE | Df Model: | 1 |
| Date: | Sun, 23 Oct 2022 | Pseudo R-squ.: | -0.8544 |
| Time: | 22:38:53 | Log-Likelihood: | -619.72 |
| converged: | True | LL-Null: | -334.20 |
| Covariance Type: | nonrobust | LLR p-value: | 1.000 |

| | coef | std err | z | P> z | [0.025 | 0.975] |
|----------|--------|---------|-------|-------|--------|--------|
| loudness | 0.0909 | 0.067 | 1.359 | 0.174 | -0.040 | 0.222 |
| rnb | 0.1001 | 0.068 | 1.473 | 0.141 | -0.033 | 0.233 |
| hot | 0.1001 | 0.068 | 1.473 | 0.141 | -0.033 | 0.233 |

remove loudness

```
df = df[["rnb", "hot"]]

X = (df.iloc[:, :-1])
y = (df.iloc[:, -1])
y = y.astype('int')

X = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = ms.train_test_split(X, y, test_size = 0.4, random_state = 1)

model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print('Train_Accuracy: ', model.score(X_train, y_train), '\n')
```

↗ Train_Accuracy: 0.8777777777777778

```
model = sm.Logit(y_train, X_train)
results = model.fit(method = "newton")
features = list(df.iloc[:, 0:-1].columns)
results.summary(xname=features)
```

Optimization terminated successfully.
Current function value: 0.689610
Iterations 4

Logit Regression Results

| | | | |
|-------------------------|------------------|--------------------------|---------|
| Dep. Variable: | hot | No. Observations: | 900 |
| Model: | Logit | Df Residuals: | 899 |
| Method: | MLE | Df Model: | 0 |
| Date: | Sun, 23 Oct 2022 | Pseudo R-squ.: | -0.8571 |
| Time: | 22:40:18 | Log-Likelihood: | -620.65 |
| converged: | True | LL-Null: | -334.20 |
| Covariance Type: | nonrobust | LLR p-value: | nan |

| coef | std err | z | P> z | [0.025 | 0.975] |
|--------|---------|-------|-------|--------|--------|
| 0.4705 | 0.000 | 0.544 | 0.010 | 0.000 | 0.000 |

```
accuracy = mt.accuracy_score(y_test, y_pred)
recall = mt.recall_score(y_test, y_pred)
precision = mt.precision_score(y_test, y_pred)
f1_score = mt.f1_score(y_test, y_pred)
matrix = mt.confusion_matrix(y_test, y_pred)

print('Accuracy: ', format(accuracy, '.2f'), '\n')
print('Recall: ', format(recall, '.2f'), '\n')
print('Precision: ', format(precision, '.2f'), '\n')
print('F1_score: ', format(f1_score, '.2f'), '\n')
print('Confusion Matrix:', '\n', matrix)
```

Accuracy: 0.86

Recall: 0.00

Precision: 0.00

F1_score: 0.00

Confusion Matrix:

```
[[518  0]
 [ 82  0]]
```

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and bei
_warn_prf(average, modifier, msg_start, len(result))

```
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.model_selection import cross_val_predict
```

```
model = LogisticRegression()
```

```
y_pred_cross = cross_val_predict(model, X, y, cv=10)
accuracy = mt.accuracy_score(y, y_pred_cross)
recall = mt.recall_score(y, y_pred_cross)
precision = mt.precision_score(y, y_pred_cross)
f1_score = mt.f1_score(y, y_pred_cross)
matrix = mt.confusion_matrix(y, y_pred_cross)
```

```
print('Accuracy: ', format(accuracy, '.2f'), '\n')
print('Recall: ', format(recall, '.2f'), '\n')
print('Precision: ', format(precision, '.2f'), '\n')
print('F1_score: ', format(f1_score, '.2f'), '\n')
print('Confusion Matrix:', '\n', matrix)
```

Accuracy: 0.87

Recall: 0.00

Precision: 0.00

F1_score: 0.00

Confusion Matrix:

```
[[1308  0]
 [ 192  0]]
```

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and bei
_warn_prf(average, modifier, msg_start, len(result))

We are still getting horrible results

KNN make sure to rerun the df table from the beginning!!!!

Normalize and partition

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

X = (df.iloc[:, :-1])
y = (df.iloc[:, -1])
y = y.astype('int')

X = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = ms.train_test_split(X, y, test_size = 0.4, random_state = 1)
```

```
from sklearn.neighbors import KNeighborsClassifier

model = KNeighborsClassifier(n_neighbors = 3)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
print('Train_Accuracy: ', model.score(X_train, y_train), '\n')
```

Train_Accuracy: 0.9066666666666666

```
import sklearn.metrics as mt

accuracy = mt.accuracy_score(y_test, y_pred)
recall = mt.recall_score(y_test, y_pred)
precision = mt.precision_score(y_test, y_pred)
f1_score = mt.f1_score(y_test, y_pred)
matrix = mt.confusion_matrix(y_test, y_pred)

print('Accuracy: ', format(accuracy, '.2f'), '\n')
print('Recall: ', format(recall, '.2f'), '\n')
print('Precision: ', format(precision, '.2f'), '\n')
print('F1_score: ', format(f1_score, '.2f'), '\n')
print('Confusion Matrix:', '\n', matrix)
```

Accuracy: 0.82

Recall: 0.11

Precision: 0.21

F1_score: 0.15

Confusion Matrix:

```
[[485 33]
 [ 73  9]]
```

```
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.model_selection import cross_val_predict

model = KNeighborsClassifier(n_neighbors = 3)

y_pred_cross = cross_val_predict(model, X, y, cv=10)
accuracy = mt.accuracy_score(y, y_pred_cross)
recall = mt.recall_score(y, y_pred_cross)
precision = mt.precision_score(y, y_pred_cross)
f1_score = mt.f1_score(y, y_pred_cross)
matrix = mt.confusion_matrix(y, y_pred_cross)

print('Accuracy: ', format(accuracy, '.2f'), '\n')
print('Recall: ', format(recall, '.2f'), '\n')
print('Precision: ', format(precision, '.2f'), '\n')
print('F1_score: ', format(f1_score, '.2f'), '\n')
print('Confusion Matrix:', '\n', matrix)
```

Accuracy: 0.84

Recall: 0.13

Precision: 0.25

F1_score: 0.17

Confusion Matrix:
[[1232 76]