PREDICTING HOW SUITABLE A SONG IS FOR DANCING

1 Introduction

1.1 Context

Dancing is a universal form of art that can be performed by anyone. Be it ballet, hip hop or even 30 second TikTok dances, majority of us have participated in some form of dancing. Often, most dances are accompanied by music. Hence, we wanted to explore the relationship between dance and music.

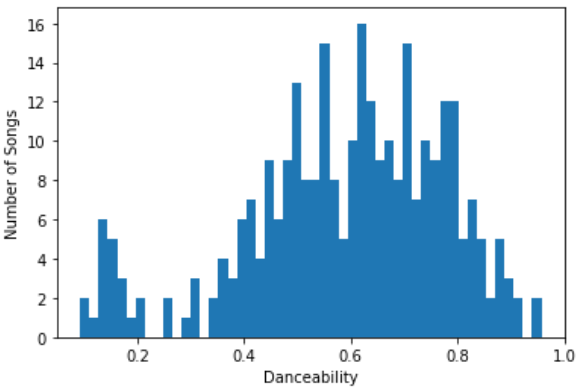
The position of Spotify as the reigning audio streaming service provider has remained undisputed throughout recent years, with over 50 million songs listed on their platform and 271 million total monthly active users[1]. Quoting from Spotify's Chief Marketing Officer Seth Farbman, “There has been some debate about whether big data is muting creativity in marketing, but we have turned that on its head. For us, data inspires and gives an insight into the emotion that people are expressing." [2] Evidently, Spotify’s services are highly data-driven. Therefore, we chose Spotify as our source of data.

1.2 Aims

This report is split into two sections. The first is an exploratory analysis to understand our dataset. The second section aims to build a predictive model to answer our question: What is the most important feature of a song that contributes to its danceability?

1.3 Dataset

Our selected dataset is derived from the “Audio Features of 160k+ Songs Released in Between 1921 and 2020” dataset on Kaggle, which originated from the Spotify Web API. In order to obtain results that accurately reflect the current music industry, our analyses were based on data from 2015 to 2020 only. The data contains 19 attributes of a song, including categorical variables such as the mode and key,as well as continuous variables such as the energy level of a song.



We had initially set out to predict the popularity of a song based on its characteristics. However, we felt that popularity is significantly biased towards the success of the artist’s marketing campaigns, hence we redirected our focus around danceability.

Figure 1: Distribution of songs based on their danceability

2 Exploratory Analysis

2.1 Data Preparation

Firstly, rows with null values were removed as it indicated that no data was collected, leaving us with 294 rows of data remaining. Next, non-meaningful attributes were removed, including song id and release date. This left us with 17 attributes, as shown in Appendix A.

We then binarized the target attribute, danceability. We defined highly danceable songs as the top 50% of data, i.e. those with a danceability score of at least 0.653. We initially used 75% as our benchmark, but it resulted in only a handful of true positive outcomes. We then created a new column, "danceability_Bl" and assigned all data points a Boolean value (1 or 0).

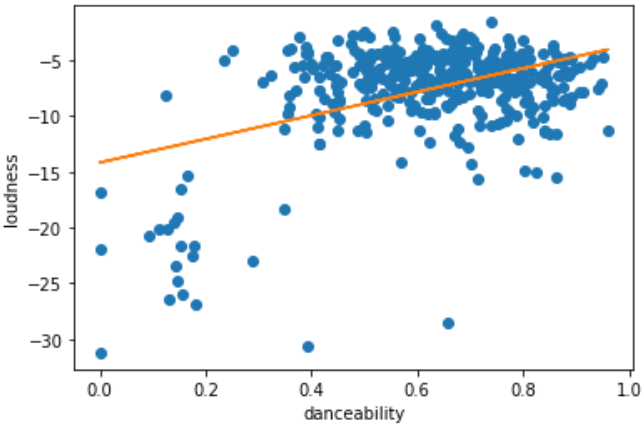
Subsequently, we standardised the data so that our algorithm did not place any emphasis on a particular attribute due to the varying scales of the attribute values.

Finally, the data was split into three sets for training (60%), validation (20%) and testing (20%). Our model was built around the training data, whereas the validation data was used at the end of each development of the model. The testing dataset was only used at the end of the model development in order to assess the true accuracy of the model.

2.2 Data Analysis

Danceability describes how suitable a song is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity [3].

We narrowed down the number of possible variables to be considered based on our intuition. For example, we concluded that the artist and song name are irrelevant to our investigation. We hypothesised that tempo is an important factor affecting the danceability of a song. We also expect to see relationships between the danceability of a song and variables such as its energy level, loudness, instrumentalness, valence and acousticness.

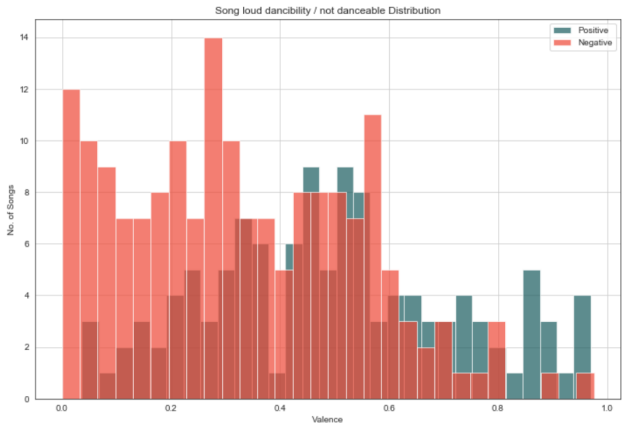


Linear regression was conducted for all attributes and the plots can be found in Appendix B.

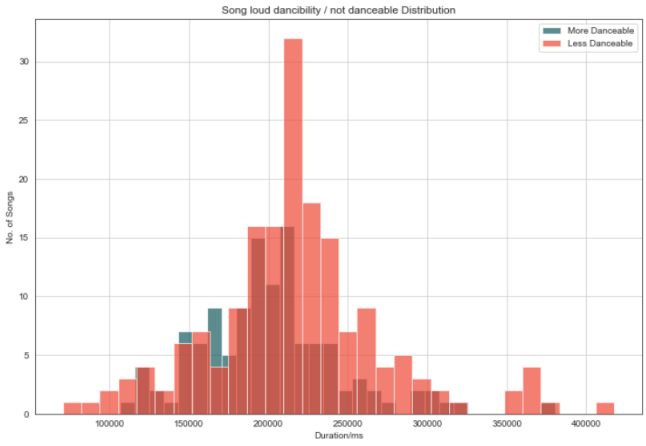
From Figure 2, we hypothesised that loudness would have a significant relationship with danceability.

Figure 2: Linear Regression Plot of Loudness Against Danceability

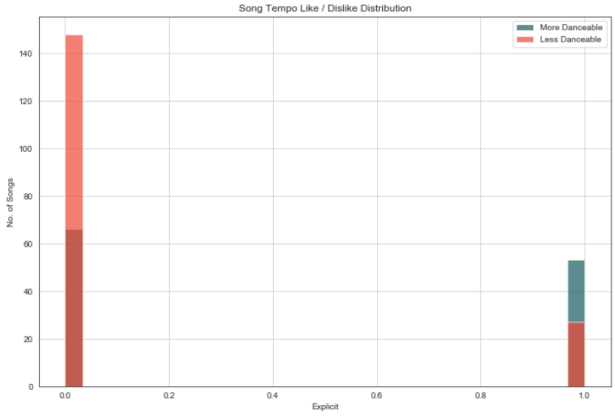
We plotted histograms of the variables for both cases where the variables were danceable and not danceable. Here, the blue bars correspond to more danceable for each instance while the red bars are less danceable.



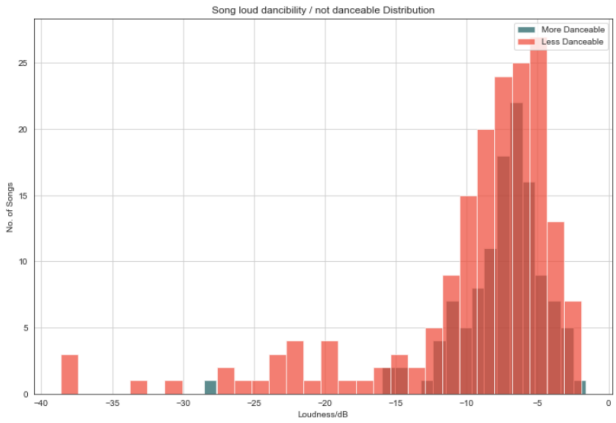
Looking at the valence of the songs, it can be seen that danceable songs are normally distributed around the mean. Meanwhile for the less danceable songs, the plot indicates that a lower valence is correlated with lower danceability.



Next, looking at duration in milliseconds, the plot shows that the more danceable songs have a lower mean while songs in general have a normal distribution when it comes to duration.



In terms of tempo which is a binary variable, it can be seen that songs with tempo are more likely to be danceable.



With regards to loudness, most songs whether danceable or not danceable reside in the same loudness range where they are normally distributed.

2.4 Performance metrics in context

- Accuracy: The proportion of songs which have been correctly predicted to be danceable to or non-danceable to.
- Precision: The proportion of songs predicted to be danceable to which turned out to be danceable.
- Recall: The proportion of songs predicted to be non-danceable to which turn out to be danceable to.

As our dataset was found to be well balanced, accuracy was identified as the most suitable metric to consider as the proportion of correctly predicted danceable and non-danceable songs are relatively equally weighted. Additionally, as new songs are released on a daily basis, being able to predict danceable songs as quickly as possible is vital for us in order to keep up with the latest and trendiest songs. Therefore, we optimised our model for accuracy.

3 Predictive Model

3.1 Logistic Regression

We considered 15 variables in the building of our predictive model. Among other variables, the tempo and valence were expected to improve the accuracy of the predictive model significantly.

Using the forward selection method, we found the optimum model, which is a four-factor model. Below is a table displaying the results of the model on our validation set.

Table 1: Results of Model on Validation Set

Model	**danceability ~ explicit + valence + acousticness + instrumentalness**
Confusion Matrix	[[15 9] [7 28]]
Accuracy	0.7288135593220338

Precision	0.6818181818181818
Recall	0.625

Some of the attributes in the model were within our expectations, such as valence, as it is natural for people to feel like dancing along to happier-sounding songs. However, 'explicit' was an unexpected attribute that improved the model accuracy as it showed little correlation to danceability in a linear regression plot done during our exploratory analysis. It was also shocking to see that tempo and loudness did not improve the model as we had expected as they showed potential, as seen in Figure 2 for loudness.

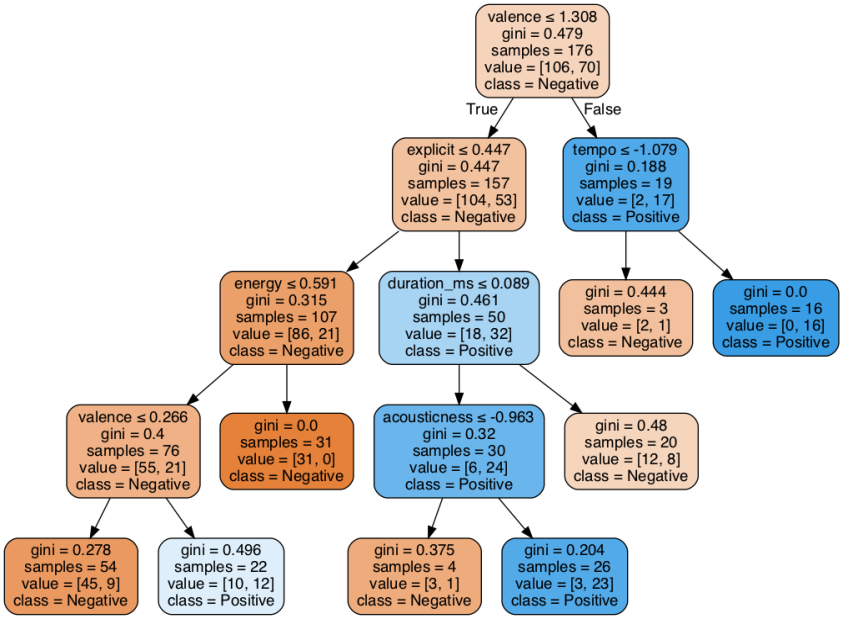
3.2 Non-linear Model

3.2.1 Decision Tree

Gini Impurity is used to build our decision tree model, it measures the probability whether a randomly assigned datapoint is incorrect. The model is greedy, at each step it will make a decision that is locally optimal and hoping to end up close to a global optima.

Similar to logistic regression, it is essential to prevent the data from overfitting. After ensuring the dataset is balanced, we standardized the data to have a mean of zero and standard deviation of 1. We then train the model on the training set and check its accuracy on the validation set. The validation accuracy reaches to peak at max_depth = 4 and min_inpurity_decrease = 0.01. Giving an accuracy of 0.695, precision of 0.636 and recall of 0.583. The resulting accuracy is good but the logistic regression model has a better performance in comparison.

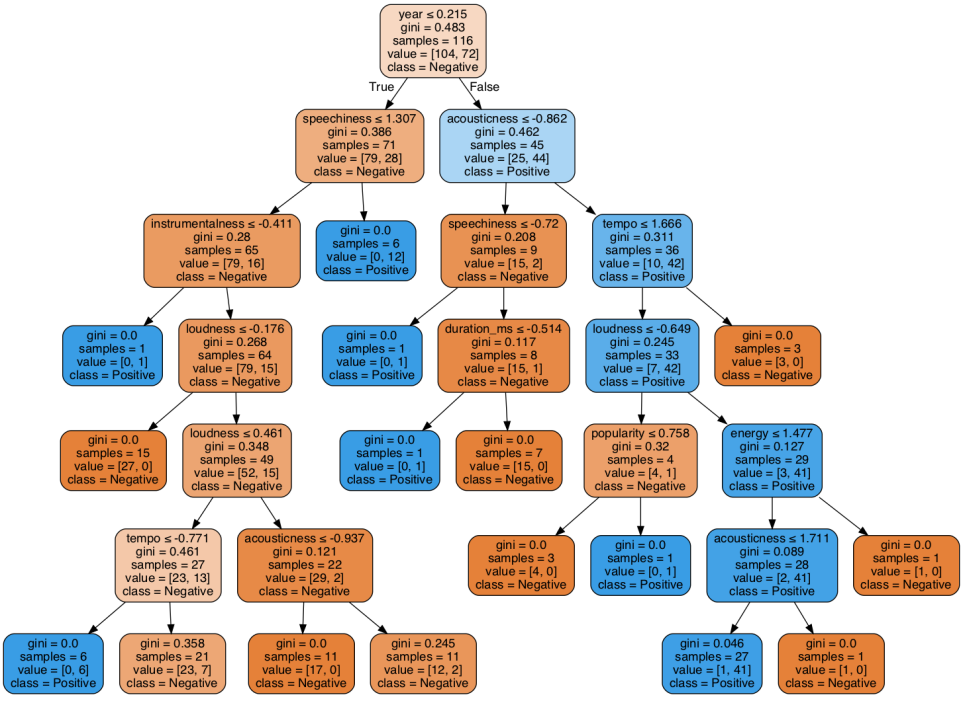
```
music = tree.DecisionTreeClassifier(max_depth = 4, min_impurity_decrease = 0.01)
music = music.fit(X_train, y_train)


print('\nFor the validation set:')
print('Accuracy: \t{}'.format(accuracy_score(y_val, music.predict(X_val))))
print('Precision: \t{}'.format(precision_score(y_val, music.predict(X_val))))
print('Recall: \t{}'.format(recall_score(y_val, music.predict(X_val))))
```



Final Decision Tree Graph

3.2.2 Random Forest

Random forest is an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes or average prediction of the individual trees. [4]In our cases, the peak accuracy of the model occurs at max_depth is 6 and when min_samples_split is 4. This gives an average (out of five trial) accuracy of 0.661, precision of 0.583 and 0.521 for recall. Even though the features are different, the model has similar Gini value when splitting and it's accuracy is lower than the previous two models.



Example Tree from Our Random Forest

3.2.3 Support Vector Machine

By definition, a support-vector machine constructs a hyperplane or set of hyperplanes in a high-dimensional space, which can be used for classification, regression, or other tasks. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (so-called functional margin), since in general the larger the margin, the lower the generalization error of the classifier. [5]In short, it used to find the best separation between classes. In our situation, after standardizing the data, we have set the C value to be 1 (no penalty), kernel to be "radial bias function", and gamma to be "auto". Resulting in an accuracy of 0.678, precision of 0.647 and recall of 0.458.

4 Comparison of Results

Table 2: Performance Metrics on Validation Set

	Accuracy	Precision	Recall
Logistic Regression	0.712	0.733	0.458
Decision Tree	0.695	0.636	0.583
Random Forest	0.661	0.583	0.521
Support Vector Machine	0.678	0.647	0.458

The accuracy scores for all of the models were found to be over 0.600, which is satisfactory, with the result from the logistic regression model performing the best at an accuracy level of 0.712.

5 Conclusion

Adhering to the performance metric that we had set from the beginning, we selected the logistic regression model due to the following reasons:

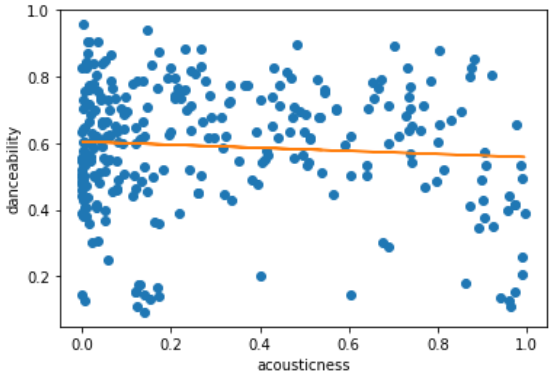
1. Highest accuracy
2. Highest precision
3. Intuitive - it is easy to understand and the attributes used are logical

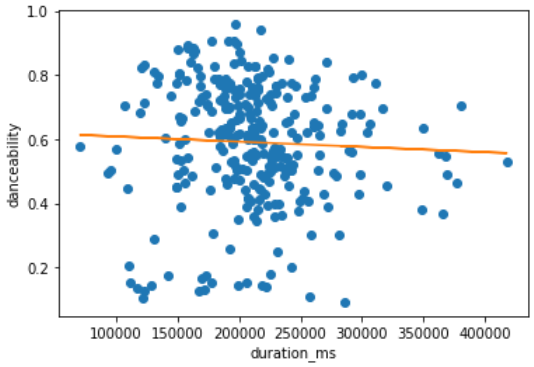
Appendix A: Attributes Table

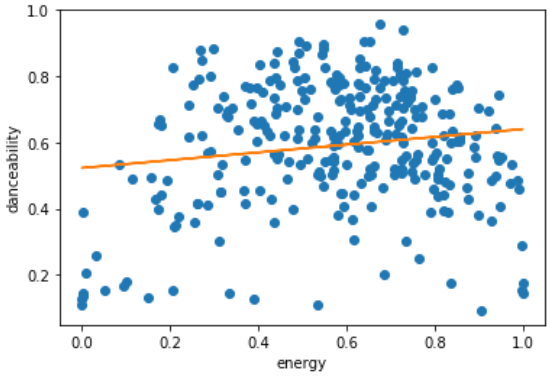
Attribute	**Meaning**	**Value Range**	**Value Type**
acousticness	A measure of whether the song is acoustic	0 to 1	Float
danceability	How suitable a song is for dancing based on a combination of musical elements including tempo, rhythm, stability, beat strength and overall regularity	0 to 1	Float
energy	A perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud and noisy	0 to 1	Float
duration_ms	Duration of song in milliseconds	200k to 300k	Integer
instrumentalness	Whether a track contains no vocals. "Ooh" and "aah" sounds are treated as instrumental. (1 means track contains no vocal content). Instrumentalness above 0.5 represents instrumental tracks.	0 to 1	Float
valence	Musical positiveness conveyed by the song (high means happy, cheerful and euphoric)	0 to 1	Float
popularity	Popularity of a song	0 to 100	Integer
tempo	The speed or pace of a given song in beats per minute (BPM).	50 to 150	Float
liveness	The presence of an audience in the song recording. A value above 0.8 provides strong likelihood that the song is live.	0 to 1	Float
loudness	Loudness values (in dB) are averaged across the entire song	-60 to 0	Float
speechiness	Presence of spoken words in a song. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value.	0 to 1	Float
year	Year of release	1921 to 2020	Integer
mode	Musical modality of the song - the type of scale from which its melodic content is derived	0=Minor, 1=Major	Integer
explicit	Presence of explicit content	0=No, 1=Yes	Integer
key	All keys on octave (C as 0, C#/D♭ as 1, D as 2 ...) If no key was detected, the value is -1	0 to 11	Integer
artists*	List of artists mentioned	-	String
name*	Name of song	-	String

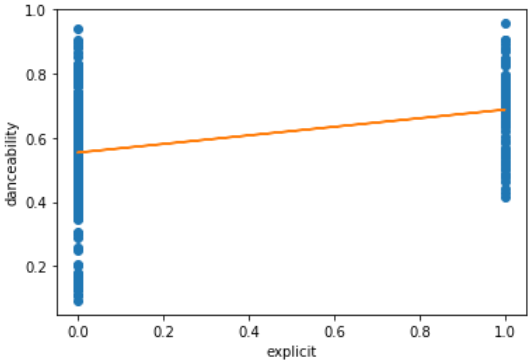
** Variables were not used for the predictive model development*.

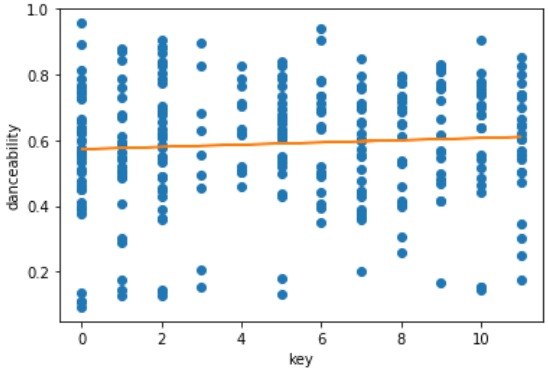
Appendix B: Linear Regression Plots

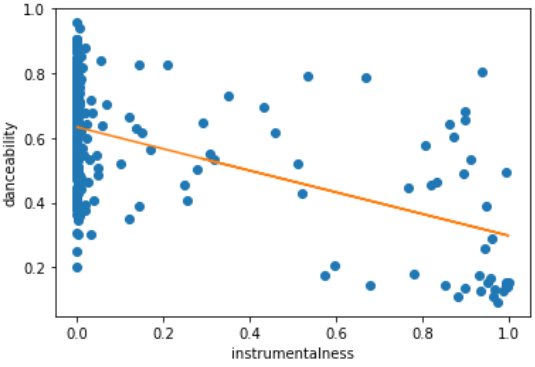


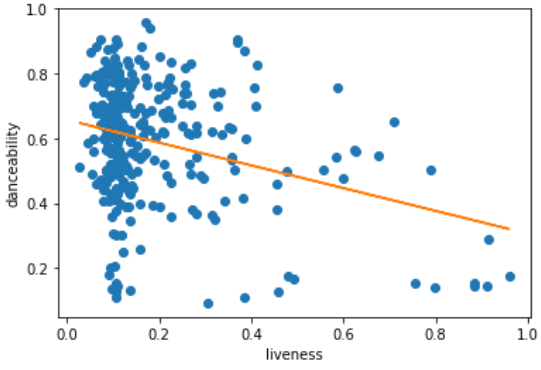


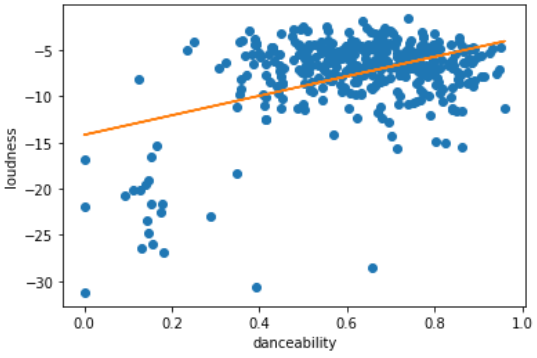


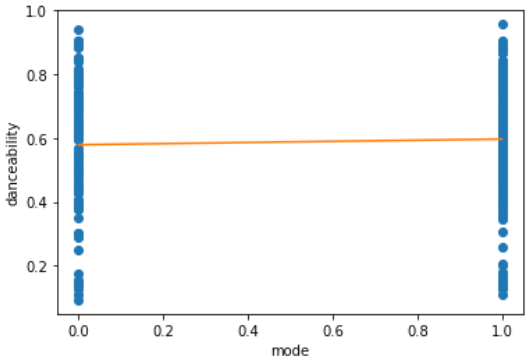


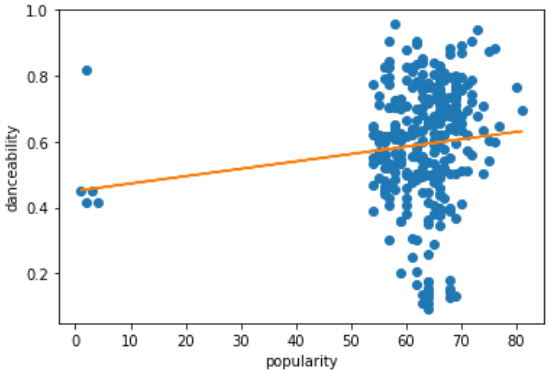


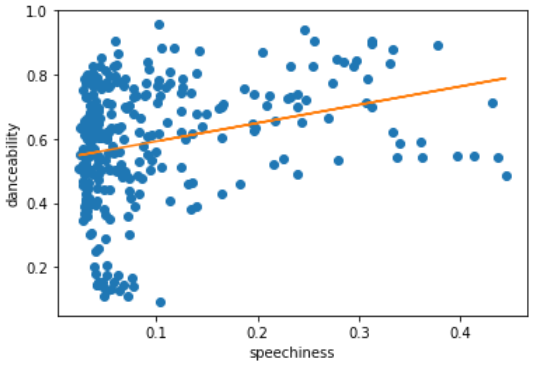


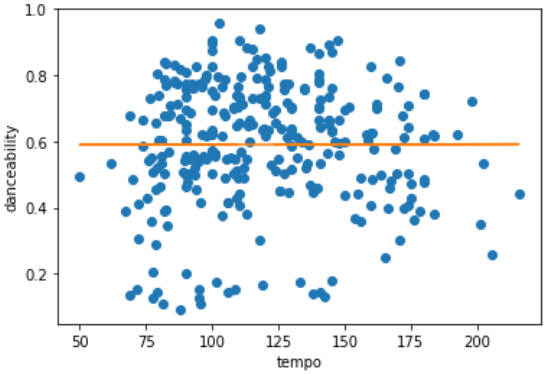


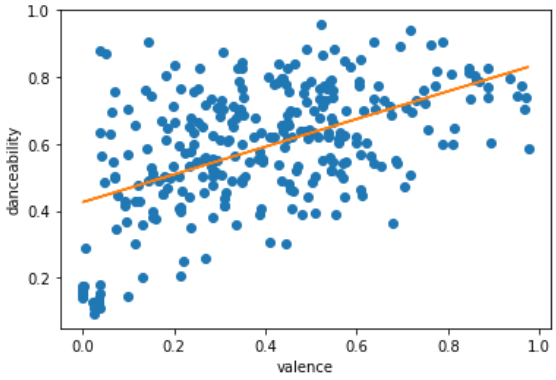



danceability
mode


danceability
popularity


danceability
speechiness


danceability
tempo


danceability
valence

Appendix C: Original Code

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import graphviz
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from scipy import stats
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score
music = pd.read_csv('spotify15-20.csv', engine='python')

class ModelSummary:
    def __init__(self, clf, X, y):
        self.clf = clf
        self.X = X
        self.y = y
        pass

    def get_se(self):
        predProbs = self.clf.predict_proba(self.X)
        X_design = np.hstack([np.ones((self.X.shape[0], 1)), self.X])
        V = np.diagflat(np.product(predProbs, axis=1))
        covLogit = np.linalg.inv(np.dot(np.dot(X_design.T, V), X_design))
        return np.sqrt(np.diag(covLogit))

    def get_ci(self, SE_est):
         p = 0.975
        df = len(self.X) - 2
        crit_t_value = stats.t.ppf(p, df)
        coefs = np.concatenate([self.clf.intercept_, self.clf.coef_[0]])
        upper = coefs + (crit_t_value * SE_est)
        lower = coefs - (crit_t_value * SE_est)
        cis = np.zeros((len(coefs), 2))
        cis[:,0] = lower
        cis[:,1] = upper
        return cis

    def get_pvals(self):
        p = self.clf.predict_proba(self.X)
        n = len(p)
        m = len(self.clf.coef_[0]) + 1
        coefs = np.concatenate([self.clf.intercept_, self.clf.coef_[0]])
        se = self.get_se()
        t =  coefs/se
        p = (1 - stats.norm.cdf(abs(t))) * 2
        return p

    def get_summary(self, names=None):
        ses = self.get_se()
        cis = self.get_ci(ses)
        lower = cis[:, 0]
        upper = cis[:, 1]
        pvals = self.get_pvals()
        coefs = np.concatenate([self.clf.intercept_, self.clf.coef_[0]])
```

```
        data = []
        for i in range(len(coefs)):
            currlist = []
            currlist.append(np.round(coefs[i], 3))
            currlist.append(np.round(ses[i], 3))
            currlist.append(np.round(pvals[i], 3))
            currlist.append(np.round(lower[i], 3))
            currlist.append(np.round(upper[i], 3))
            data.append(currlist)
        cols = ['coefficient', 'std', 'p-value', '[0.025', '0.975]']
        sumdf = pd.DataFrame(columns=cols, data=data)
        if names is not None:
            new_names = ['intercept']*(len(names) + 1)
            new_names[1:] = [i for i in names]
            sumdf.index = new_names
        else:
            try:
                names = list(self.X.columns)
                new_names = ['intercept']*(len(names) + 1)
                new_names[1:] = [i for i in names]
                sumdf.index = new_names
            except:
                pass
        print(sumdf)
        acc = accuracy_score(self.y, self.clf.predict(self.X))
        confmat = confusion_matrix(self.y, self.clf.predict(self.X))
        print('-'*60)
        print('Confusion Matrix (total:{}) \t Accuracy: \t  {}'.format(len(self.X),np.round(acc, 3)))
        print('  TP: {} | FN: {}'.format(confmat[1][1],confmat[1][0]))
        print('  FP: {} | TN: {}'.format(confmat[0][1],confmat[0][0]))

def rss(x, y, a, b):
    x = np.asarray(x) #Cast to numpy array so that we can do arithmatic with arrays
    y = np.asarray(y)

    y_predicted = a*x+b # This is an array. Uncomment next line to see
    residuals = y - y_predicted
    return sum(pow(residuals, 2))

def tss(y):
    ymean = np.mean(y)
    return sum(pow((y - ymean), 2))

def r2metric(x, y, a, b):
    return 1 - rss(x, y, a, b) / tss(y)
```

Binarising danceability values and creating a new bin

```
music["danceability_Bl"] = music["danceability"]
music['danceability_Bl'] = music['danceability_Bl'].apply(lambda x: 1 if x > 0.653 else 0)
music = music.drop(['artists', 'id','name','release_date',"danceability"], axis=1)
music.head()
```

Split the dataset into training, validation and test sets

```
train, other = train_test_split(music, test_size=0.4, random_state=0)
validation, test = train_test_split(other, test_size=0.5, random_state=0)
```

```
X_train = train.drop(columns=['danceability_Bl'])
y_train = train['danceability_Bl']

X_val = validation.drop(columns=['danceability_Bl'])
y_val = validation['danceability_Bl']

X_test = test.drop(columns=['danceability_Bl'])
y_test = test['danceability_Bl']
```

Standardising our data

```
X_means = X_train.mean(axis=0)
X_stds = X_train.std(axis=0)

X_train = ((X_train - X_means) / X_stds)
X_val = ((X_val - X_means) / X_stds)
X_test = ((X_test - X_means) / X_stds)
print(X_train)

from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score, recall_score, accuracy_score
```

Logistic Regression: Forward Selection

```
def my_fwd_selector(X_train, y_train, X_val, y_val):
    print('================ Begining forward selection =================')
    cols = list(X_train.columns)
    best_val_acc = 0
    selected_vars = []
    while len(cols) > 0:
        print('Trying {} var models'.format(len(selected_vars) + 1))
        candidate = None
        for i in range(len(cols)):
            current_vars = selected_vars.copy()
            current_vars.append(cols[i])
            if len(current_vars) == 1:
                new_X_train = X_train[current_vars].values.reshape(-1, 1)
                new_X_val = X_val[current_vars].values.reshape(-1, 1)
            else:
                new_X_train = X_train[current_vars]
                new_X_val = X_val[current_vars]

            mod = LogisticRegression(C=1e9).fit(new_X_train, y_train)
            val_acc = accuracy_score(y_val, mod.predict(new_X_val))
            if val_acc - best_val_acc > 0.005:
                candidate = cols[i]
                best_val_acc = val_acc
        if candidate is not None:
            selected_vars.append(candidate)
            cols.remove(candidate)
            print('------- Adding {} to the model ---------'.format(candidate))
        else:
            break
        print('Columns in current model: {}'.format(', '.join(selected_vars)))
        print('Best validation accuracy is {}'.format(np.round(best_val_acc, 3)))
    return selected_vars
```

```
selected_columns = my_fwd_selector(X_train, y_train, X_val, y_val)
model = LogisticRegression(C=1e9).fit(X_train[selected_columns], y_train)

y_train_predicted = model.predict(X_train[selected_columns])
y_val_predicted = model.predict(X_val[selected_columns])

print('======= Accuracy  table =======')
print('Training accuracy is:    {}'.format(accuracy_score(y_train, y_train_predicted)))
print('Validation accuracy is:  {}'.format(accuracy_score(y_val, y_val_predicted)))
```

Getting LogReg Results on Validation and Test set

```
columns=['tempo','speechiness','instrumentalness','explicit','valence','key']

mod = LogisticRegression(C=1e9).fit(X_train[columns], y_train)

print('Validation results:')
y_val_predicted = mod.predict(X_val[columns])
cm = confusion_matrix(y_val, y_val_predicted, labels=[1, 0])
print(cm)
print('Accuracy validation: {}'.format(accuracy_score(y_val, y_val_predicted)))
print('Precision validation: {}'.format(precision_score(y_val, y_val_predicted)))
print('Recall validation: {}'.format(recall_score(y_val, y_val_predicted)))

print('\n Test results:')
y_test_predicted = mod.predict(X_test[columns])
cm = confusion_matrix(y_test, y_test_predicted, labels=[1, 0])
print(cm)
print('Accuracy test: {}'.format(accuracy_score(y_test, y_test_predicted)))
print('Precision test: {}'.format(precision_score(y_test, y_test_predicted)))
print('Recall test: {}'.format(recall_score(y_test, y_test_predicted)))
```

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
music = tree.DecisionTreeClassifier(max_depth = 4, min_impurity_decrease = 0.01)
music = music.fit(X_train, y_train)

print('\nFor the validation set:')
print('Accuracy: \t{}'.format(accuracy_score(y_val, music.predict(X_val))))
print('Precision: \t{}'.format(precision_score(y_val, music.predict(X_val))))
print('Recall: \t{}'.format(recall_score(y_val, music.predict(X_val))))

dot_data = tree.export_graphviz(music, out_file=None)
graph = graphviz.Source(dot_data)

predictors = X_train.columns
dot_data = tree.export_graphviz(music, out_file=None,
                                feature_names = predictors,
                                class_names = ('Negative', 'Positive'),
                                filled = True, rounded = True,
                                special_characters = True)
graph = graphviz.Source(dot_data)
graph
```

Random Forest

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(max_depth=6, min_samples_split=4)
model = model.fit(X_train, y_train)
ypred = model.predict(X_val)

acc = accuracy_score(y_val, ypred)
prec = precision_score(y_val, ypred)
rec = recall_score(y_val, ypred)

print('Precision:{}'.format(prec))
print('Recall: {}'.format(rec))
print('Accuracy: {}'.format(acc))

est = model.estimators_[5]
dot_data = tree.export_graphviz(est, out_file=None)
graph = graphviz.Source(dot_data)

predictors = X_train.columns
dot_data = tree.export_graphviz(est, out_file=None,
                                feature_names = predictors,
                                class_names = ('Negative', 'Positive'),
                                filled = True, rounded = True,
                                special_characters = True)
graph = graphviz.Source(dot_data)
graph
```

Support Vector Machine

```
from sklearn.svm import SVC
model = SVC(C=1, kernel='rbf', gamma="auto")   # what gamma and kernel value we should use
model = model.fit(X_train, y_train)

ypred = model.predict(X_val)

acc = accuracy_score(y_val, ypred)
prec = precision_score(y_val, ypred)
rec = recall_score(y_val, ypred)

print('Precision:{}'.format(prec))
print('Recall:{}'.format(rec))
print('Accuracy:{}'.format(acc))
```

References

[1] https://www.businessofapps.com/data/spotify-statistics/#:~:text=Spotify%20currently%20lists%20over%2050,the%20largest%20music%20library%20available.
[2] https://en.wikipedia.org/wiki/Spotify
[3] https://developer.spotify.com/documentation/web-api/reference/tracks/get-audio-features/
[4] https://en.wikipedia.org/wiki/Random_forest
[5] https://en.wikipedia.org/wiki/Support_vector_machine