In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression, LassoCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, classificatior
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from statsmodels.stats.outliers_influence import variance_inflation_factor
import warnings


Spotify_data = pd.read_csv("/Users/devansh/Downloads/New_Spspotify_chart_sor
kaggle_data = pd.read_csv("/Users/devansh/Downloads/Drake_Spotify_Data.csv")

#Upon looking at the data, we saw that there were a lot of duplicates. So, w
#all the duplicate tracks based on 'uri' and 'artist_names', then count the

artist_per_song_count = Spotify_data.drop_duplicates(subset=['uri', 'artist_
    .groupby('uri') \
    .size() \
    .reset_index(name='count') \
    .sort_values(by='count', ascending=False)

#We now need to filter the Spotify data for tracks by Drake, and again remov
drake_chart_spotify = Spotify_data[Spotify_data['artist_names'] == "Drake"]
    .drop_duplicates(subset='uri') \
    .sort_values(by='WeekDate')

#As the trends are always changing we wanted to make sure that we are focusi
#on the later trends only
#So, we filter the Kaggle data for tracks released in or after 2020 and remo

kaggle_data_2020 = kaggle_data[kaggle_data['album_release_year'] >= 2020] \
    .drop_duplicates(subset='track_name')

#We need to join the filtered Kaggle dataset and Drake's Spotify chart data
#on their respective track URIs. This merges data for tracks found in both c
inner_join_result = pd.merge(kaggle_data_2020, drake_chart_spotify, left_on=

left_join_data = pd.merge(kaggle_data_2020, drake_chart_spotify, left_on='tr


#Now we need to select a subset of columns for analysis, focusing on musical
#This prepares the data for further analysis on the correlation between thes
new_data = left_join_data[['tempo', 'valence', 'liveness', 'instrumentalness
                      'speechiness', 'loudness', 'energy', 'danceability', '
                      'track_name_x']]



with warnings.catch_warnings():
```

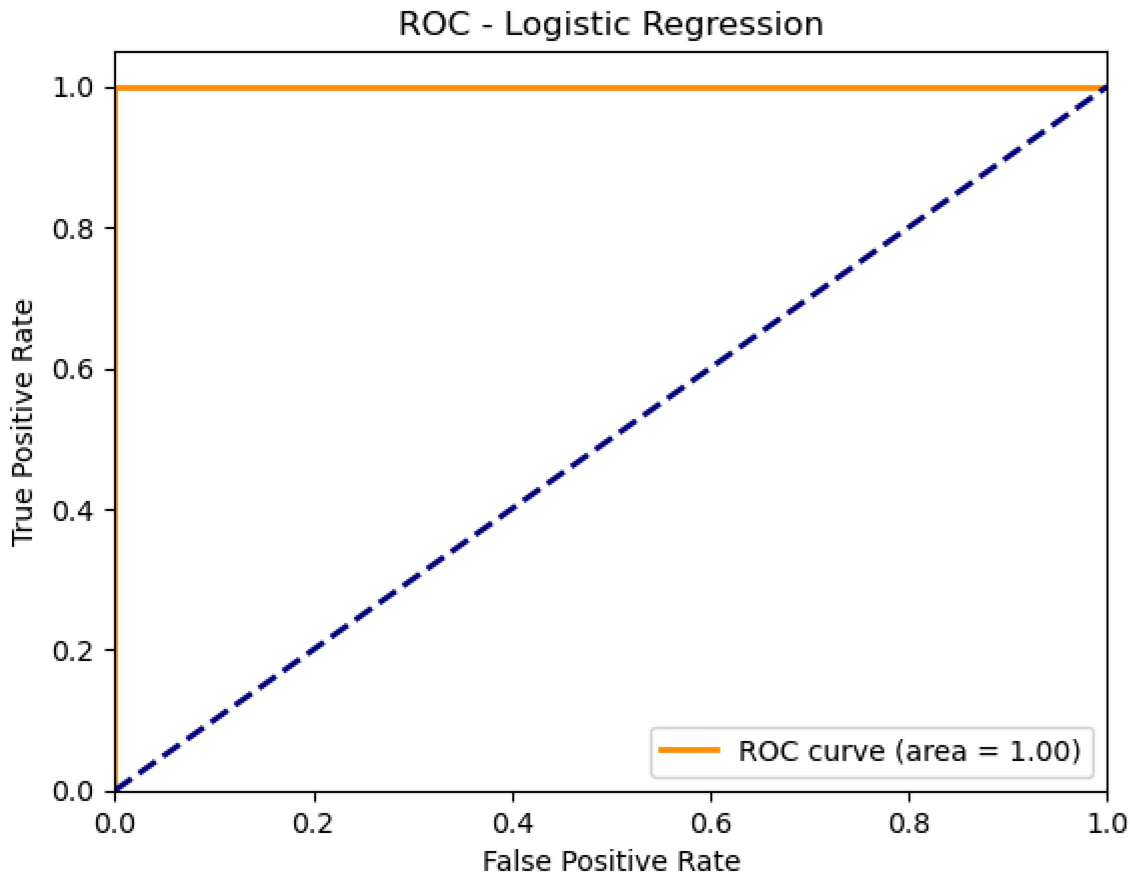
```
    warnings.simplefilter("ignore", category=pd.errors.SettingWithCopyWarnir
    new_data['on_chart'] = np.where(new_data['WeekDate'].isna(), 0, 1)

new_data = new_data.sort_values(by='WeekDate')


new_data.drop(columns='WeekDate', inplace=True)
new_data.rename(columns={'track_name_x': 'track_name'}, inplace=True)
```

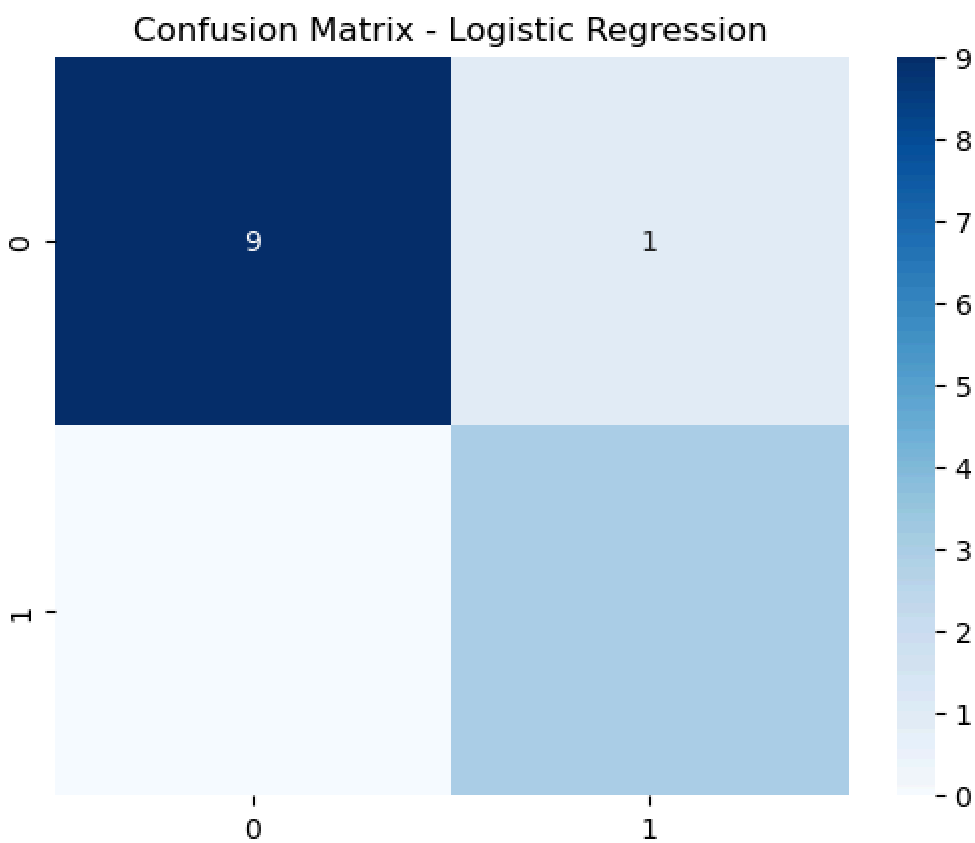
In [ ]:
```
#For an initial EDA we are checking the correlation between each col of the
correlation_matrix_full = new_data.corr(numeric_only=True)
print(correlation_matrix_full)

#We are now focusing on a few features we want to check how they affect the
features_to_compare = ['danceability', 'energy', 'loudness', 'valence', 'tem

X = new_data[features_to_compare]
y = new_data['on_chart'] #Target Binary Variable

#Splitting the data at 80% - 20% ratio
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

#We now scale the data using the StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

#Performing Logistic Regression
logistic = LogisticRegression()
logistic.fit(X_train_scaled, y_train)
predict_test_logistic = logistic.predict(X_test_scaled)
prob_logistic = logistic.predict_proba(X_test_scaled)[:,1]
accuracy = accuracy_score(y_test, predict_test_logistic)
print(f"Logistic Regression Test Accuracy: {accuracy}")
print(classification_report(y_test, predict_test_logistic))

#Performing ROC for Logistic Regression
fpr_lr, tpr_lr, _ = roc_curve(y_test, prob_logistic)
roc_auc_lr = auc(fpr_lr, tpr_lr)
plt.figure()
lw = 2
plt.plot(fpr_lr, tpr_lr, color='darkorange', lw=lw, label='ROC curve (area =
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC - Logistic Regression')
plt.legend(loc="lower right")
plt.show()

#Confusion Matrix for Logistic Regression
confusion = confusion_matrix(y_test, predict_test_logistic)
sns.heatmap(confusion, annot=True, fmt="d", cmap="Blues")
plt.title('Confusion Matrix - Logistic Regression')
plt.show()
```

```
#Random Forest
rf = RandomForestClassifier()
rf.fit(X_train_scaled, y_train)
predict_test_rf = rf.predict(X_test_scaled)
prob_rf = rf.predict_proba(X_test_scaled)[:,1]
accuracy = accuracy_score(y_test, predict_test_rf)
print(f"Random Forest Test Accuracy: {accuracy}")
print(classification_report(y_test, predict_test_rf))

#ROC for Random Forest
fpr_rf, tpr_rf, _ = roc_curve(y_test, prob_rf)
roc_auc_rf = auc(fpr_rf, tpr_rf)
plt.figure()
plt.plot(fpr_rf, tpr_rf, color='darkorange', lw=lw, label='ROC curve (area =
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC - Random Forest')
plt.legend(loc="lower right")
plt.show()

#Confusion Matrix for Random Forest
confusion = confusion_matrix(y_test, predict_test_rf)
sns.heatmap(confusion, annot=True, fmt="d", cmap="Blues")
plt.title('Confusion Matrix - Random Forest')
plt.show()

#Perform Lasso regression to see whic hcef results in 0
lasso = LassoCV(cv=5).fit(X_train_scaled, y_train)
lasso_coefficients = pd.Series(lasso.coef_, index=features_to_compare)
print("Lasso Coefficients:")
print(lasso_coefficients)

#check for a non-linear relationship
qda = QuadraticDiscriminantAnalysis()
qda.fit(X_train_scaled, y_train)
pred_test_qda = qda.predict(X_test_scaled)
accuracy = accuracy_score(y_test, pred_test_qda)
print(f"QDA Test Accuracy: {accuracy}")
print(classification_report(y_test, pred_test_qda))

#check for vif value
vif_data = pd.DataFrame({'feature': X.columns,
                         'VIF': [variance_inflation_factor(X.values, i) for
print("VIF Data:")
print(vif_data)

plt.figure(figsize=(16, 10))
for i, feature in enumerate(features_to_compare, 1):
    plt.subplot(3, 3, i)
    sns.boxplot(x='on_chart', y=feature, data=new_data)
    plt.title(feature)
```

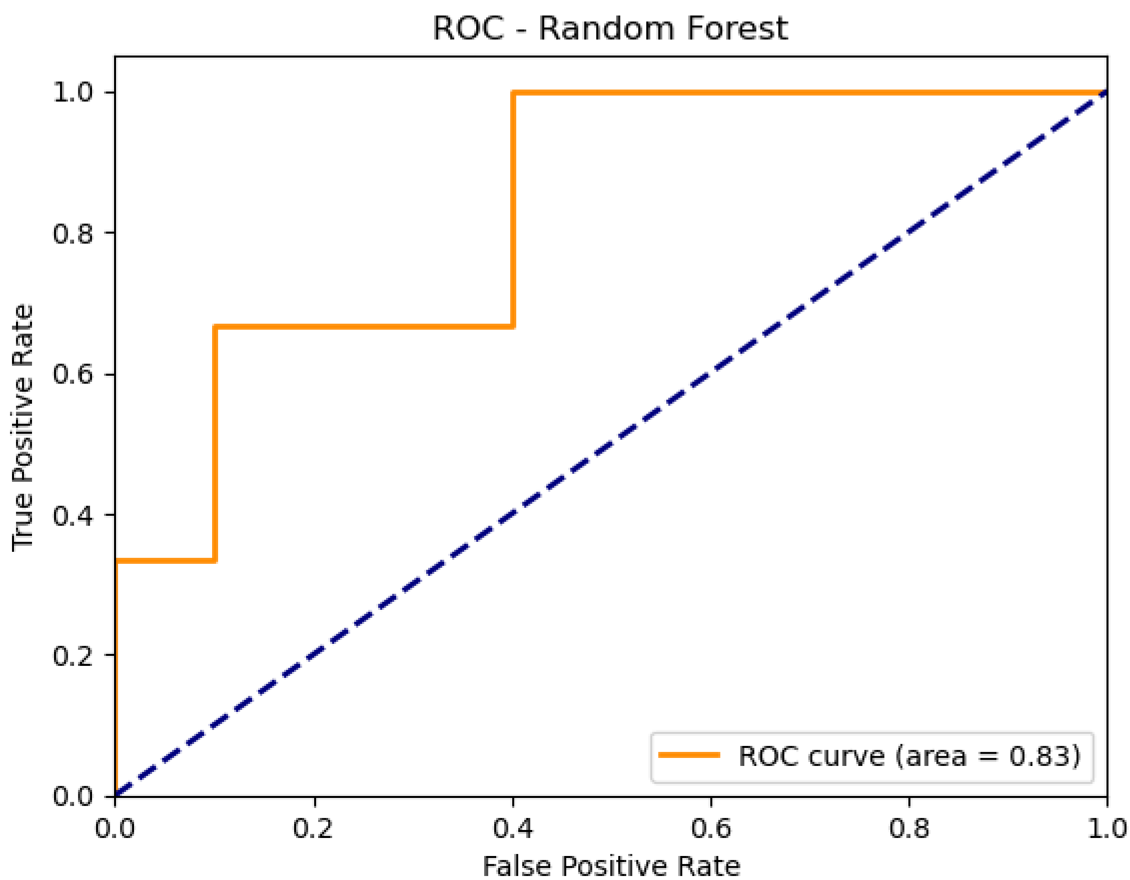
```
plt.tight_layout()
plt.show()

avg_features_by_chart_status = new_data.groupby('on_chart')[features_to_comp
avg_features_by_chart_status.T.plot(kind='bar', figsize=(12, 6))
plt.title('Average Feature Values: Charted vs Not Charted')
plt.ylabel('Average Value')
plt.xticks(rotation=45)
plt.legend(title='Charted', labels=['Not Charted', 'Charted'])
plt.show()
```

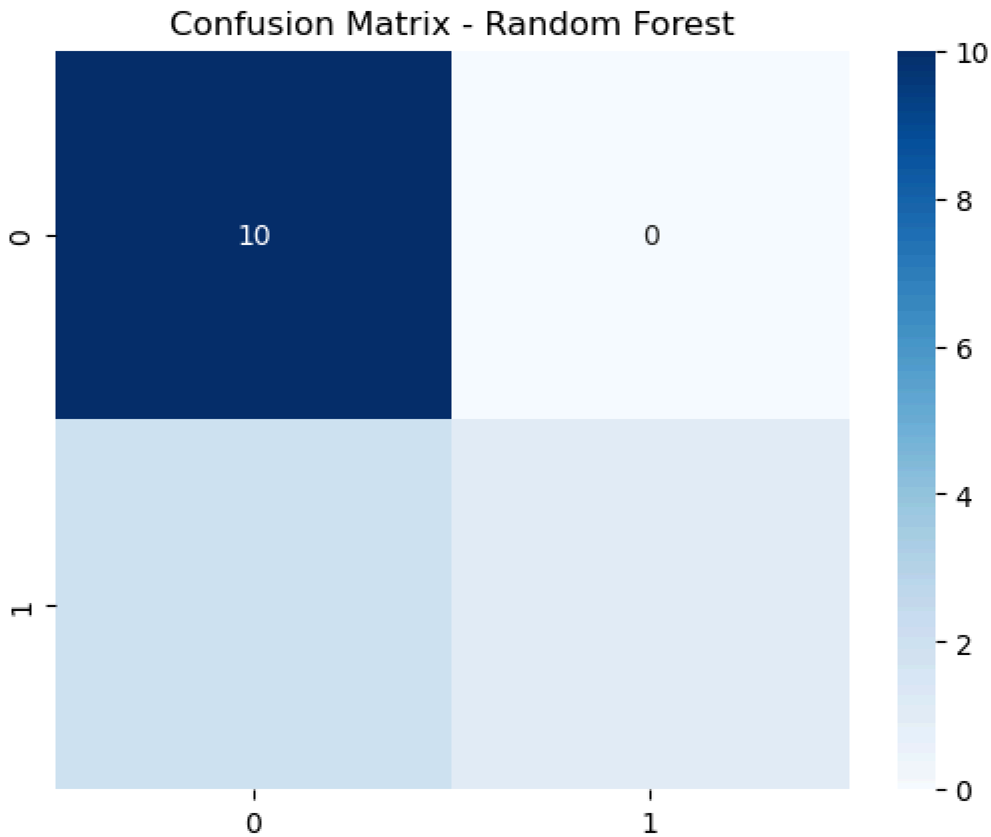
```
                     tempo   valence  liveness  instrumentalness  \
tempo             1.000000 -0.023047 -0.024614         -0.009727   
valence          -0.023047  1.000000  0.093202         -0.078181   
liveness         -0.024614  0.093202  1.000000         -0.121425   
instrumentalness -0.009727 -0.078181 -0.121425          1.000000   
acousticness     -0.257692 -0.026307 -0.168289          0.316509   
speechiness       0.186888  0.111957  0.360015         -0.217171   
loudness         -0.013209  0.224281  0.278834         -0.745402   
energy            0.069052  0.333075  0.363174         -0.381465   
danceability     -0.087213  0.071724 -0.027177         -0.322906   
on_chart         -0.240399  0.156262 -0.067126          0.239609   

                  acousticness  speechiness  loudness    energy  danceabilit
y  \
tempo                -0.257692     0.186888 -0.013209  0.069052     -0.08721
3   
valence              -0.026307     0.111957  0.224281  0.333075      0.07172
4   
liveness             -0.168289     0.360015  0.278834  0.363174     -0.02717
7   
instrumentalness      0.316509    -0.217171 -0.745402 -0.381465     -0.32290
6   
acousticness          1.000000    -0.167270 -0.433886 -0.432453     -0.31067
0   
speechiness          -0.167270     1.000000  0.259347  0.146025      0.05080
9   
loudness             -0.433886     0.259347  1.000000  0.680207      0.22533
5   
energy               -0.432453     0.146025  0.680207  1.000000     -0.01002
5   
danceability         -0.310670     0.050809  0.225335 -0.010025      1.00000
0   
on_chart              0.287250    -0.208633 -0.078500  0.046559      0.02744
4   

                  on_chart  
tempo            -0.240399  
valence           0.156262  
liveness         -0.067126  
instrumentalness  0.239609  
acousticness      0.287250  
speechiness      -0.208633  
loudness         -0.078500  
energy            0.046559  
danceability      0.027444  
on_chart          1.000000  
Logistic Regression Test Accuracy: 0.9230769230769231
              precision    recall  f1-score   support

           0       1.00      0.90      0.95        10
           1       0.75      1.00      0.86         3

    accuracy                           0.92        13
   macro avg       0.88      0.95      0.90        13
weighted avg       0.94      0.92      0.93        13
```


ROC - Logistic Regression
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate
ROC curve (area = 1.00)

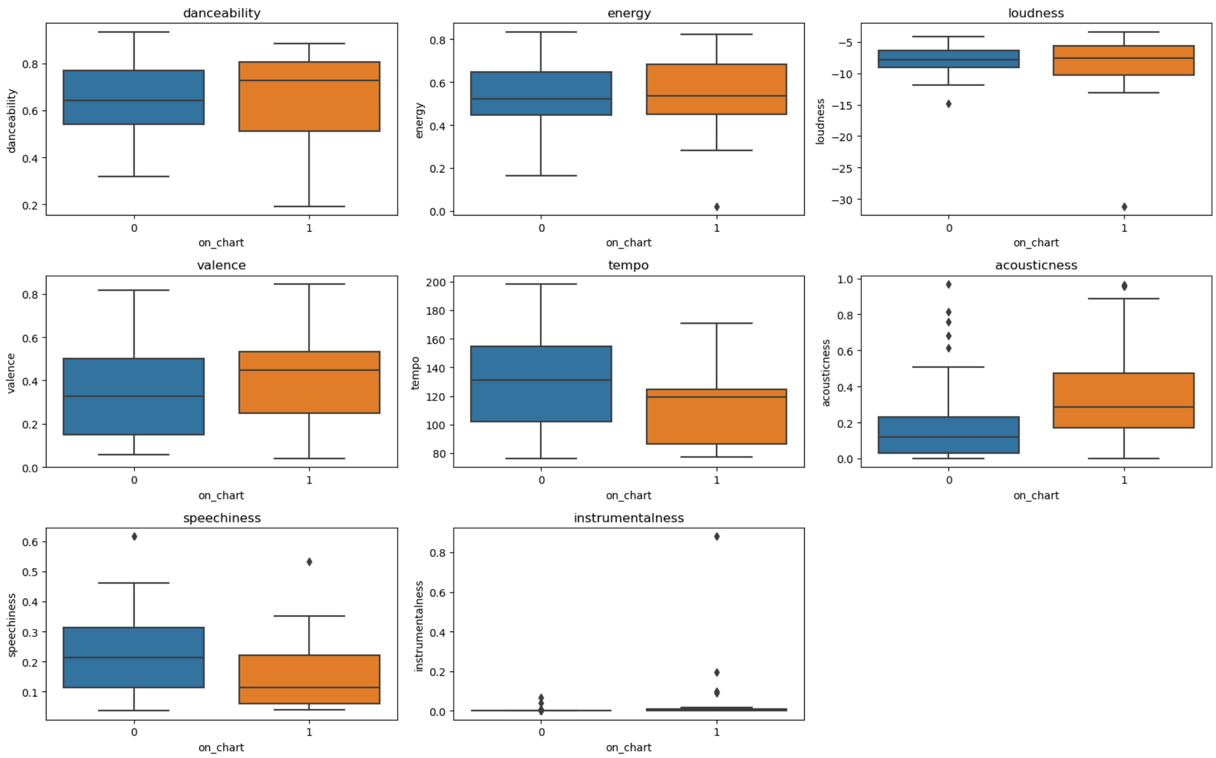

Confusion Matrix - Logistic Regression
9
1
0
1
0
1
9
8
7
6
5
4
3
2
1
0

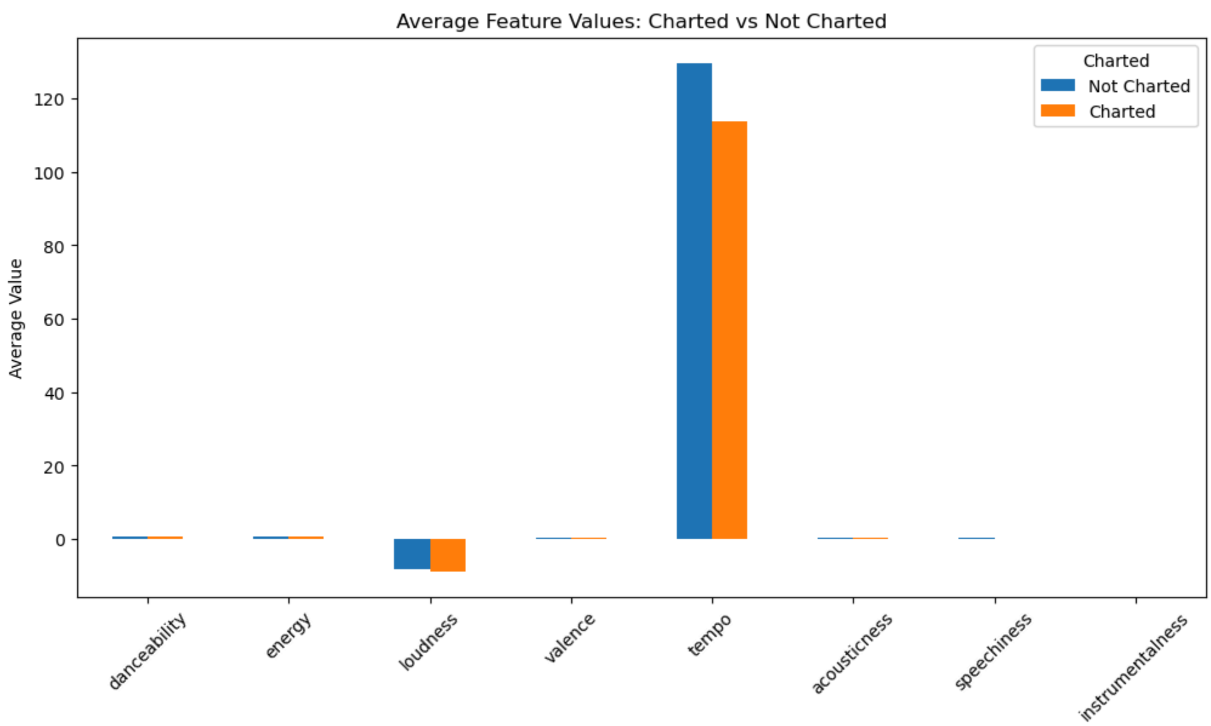
```
Random Forest Test Accuracy: 0.8461538461538461
              precision    recall  f1-score   support

           0       0.83      1.00      0.91        10
           1       1.00      0.33      0.50         3

    accuracy                           0.85        13
   macro avg       0.92      0.67      0.70        13
weighted avg       0.87      0.85      0.81        13
```






```
Lasso Coefficients:
danceability       -0.0
energy              0.0
loudness           -0.0
valence             0.0
tempo              -0.0
acousticness        0.0
speechiness        -0.0
instrumentalness    0.0
dtype: float64
QDA Test Accuracy: 0.9230769230769231
              precision    recall  f1-score   support

           0       0.91      1.00      0.95        10
           1       1.00      0.67      0.80         3

    accuracy                           0.92        13
   macro avg       0.95      0.83      0.88        13
weighted avg       0.93      0.92      0.92        13

VIF Data:
            feature        VIF
0      danceability  14.389701
1            energy  12.810418
2          loudness  15.879496
3           valence   4.532060
4             tempo  16.824353
5      acousticness   2.592262
6       speechiness   3.562391
7  instrumentalness   2.542151
```


danceability
energy
loudness
valence
tempo
acousticness
speechiness
instrumentalness
on_chart


Average Feature Values: Charted vs Not Charted
Charted
Not Charted
Charted
Average Value
danceability
energy
loudness
valence
tempo
acousticness
speechiness
instrumentalness

In [ ]: