Question 1 [15 pts]: Fit a logistic regression model using all variable

A logistic regression model was used to predict whether a song would reach the Top 10 on the Billboard Hot 100 Chart, utilizing all available musical features in the dataset. This model achieved an accuracy of 73.74% and an AUC of 0.82, indicating that it successfully predicts the ranking of songs approximately three-quarters of the time.

Steps Taken for the Analysis:

1. Importing the Music Dataset: The dataset consists of 7,573 records and 39 fields, encompassing a wide range of musical features.
2. Checking for Missing Values: A thorough examination revealed no missing values, ensuring that data completeness would not bias the model's outcomes.
3. Outlier Detection Using the IQR Method: While outliers can skew model performance by influencing error minimization, these outliers were not removed because they are inherent characteristics of the songs.
4. Handling Outliers: Outliers were retained, as they represent authentic variations in the musical properties being analyzed.
5. Handling Skewed Data: Several numerical features (timbre_0_min, timbre_1_max, timbre_10_max) exhibited skewness. A log transformation was applied to these features, which reduces skewness by compressing large values and spreading out smaller values, thereby creating a more balanced distribution.
6. Data Splitting: The dataset was divided into training data (songs released before 2009) and testing data (songs released in 2010). This approach allows the model to be validated on unseen data, simulating real-world prediction scenarios.
7. Performing Logistic Regression on Training Data: Analysis of p-values indicated that features such as time signature, tempo, and key had p-values greater than 0.05, suggesting they might not significantly contribute to the model's predictive power. These features were considered for removal in subsequent steps to optimize model performance.
8. Examining Coefficients: Key features influencing the model's predictions were identified:
    - Loudness (0.3015): Songs with higher loudness levels are more likely to reach the Top 10, suggesting a trend favoring louder music.
    - Pitch (-44.6714): Higher pitch levels significantly decrease the likelihood of a song being in the Top 10, indicating a preference for lower-pitched songs.
9. Confusion Matrix Analysis on Test Data:
    - True Negatives: 4,524 instances correctly identified as not making the Top 10.
    - False Positives: 1,617 instances incorrectly predicted as making the Top 10 (Type I Error).
    - False Negatives: 274 instances incorrectly predicted as not making the Top 10 (Type II Error).
    - True Positives: 786 instances correctly identified as making the Top 10.

The model tends to over-predict success (Top 10), which could lead to a misallocation of marketing resources towards songs unlikely to become hits. On the other hand, missing potential hits (Type II Error) could result in missed revenue opportunities. 10. ROC Curve Analysis on Training Data: • AUC = 0.5: Indicates no discriminative power, similar to random guessing. • AUC < 0.5: Indicates worse performance than random guessing, which is rare. • AUC > 0.5: Implies the model has discriminatory power, with higher values indicating better performance. The AUC of 0.82 signifies that there is an 82% probability that the model will correctly rank a randomly chosen Top 10 song higher than a non-Top 10 song. This high AUC value suggests that the model effectively distinguishes between hits and non-hits.

These findings offer valuable insights for music producers and marketers, helping them refine strategies to promote songs with higher hit potential.

Code below for Q1:

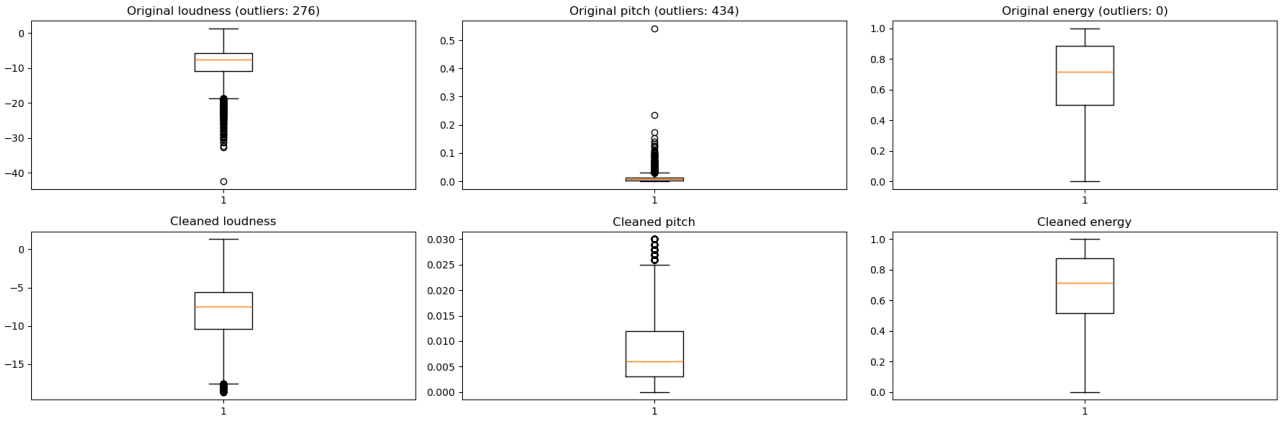
In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, roc_curve, a
from scipy import stats  # Importing scipy for additional transformations
import statsmodels.api as sm

# Load the dataset
df = pd.read_csv("MusicData.csv", encoding='ISO-8859-1')

# Display count of null values for columns with more than 0 nulls
null_counts = df.isnull().sum()
print("\nColumns with more than 0 null values:")
```

```
print(null_counts[null_counts > 0])

# Removing unnecessary variables
df = df.drop(columns=['artistname', 'songtitle', 'songID', 'artistID'])

# Show count before outlier removal
print(f"\nNumber of rows before outlier removal: {df.shape[0]}")

# Improved function to identify and remove outliers using IQR
def identify_and_remove_outliers_iqr(df, column, iqr_factor=1.5):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - iqr_factor * IQR
    upper_bound = Q3 + iqr_factor * IQR
    outliers = df[(df[column] < lower_bound) | (df[column] > upper_bound)]
    df_no_outliers = df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]
    return outliers, df_no_outliers

# List of columns to check for outliers
columns_to_check = ['loudness', 'pitch', 'energy']

# Initialize a copy of the dataframe to preserve the original data
df_no_outliers = df.copy()

plt.figure(figsize=(18, 6))
outliers_info = {}  # Dictionary to store outliers info for each column
for i, col in enumerate(columns_to_check, start=1):
    # Identify and remove outliers for the current column
    outliers, df_no_outliers = identify_and_remove_outliers_iqr(df_no_outliers, col)
    outliers_info[col] = outliers

    # Plotting boxplot for original data
    plt.subplot(2, len(columns_to_check), i)
    plt.boxplot(df[col], vert=True)
    plt.title(f'Original {col} (outliers: {len(outliers)})')

    # Plotting boxplot for data after outlier removal
    plt.subplot(2, len(columns_to_check), i + len(columns_to_check))
    plt.boxplot(df_no_outliers[col], vert=True)
    plt.title(f'Cleaned {col}')

plt.tight_layout()
plt.show()

# Update the main dataframe with outlier removal results
df = df_no_outliers

# Display number of rows after removing outliers
print(f"\nNumber of rows after removing outliers: {df.shape[0]}")

# Focus on the three skewed variables
variables = ['timbre_0_min', 'timbre_1_max', 'timbre_10_max']


# Apply log transformation (adding a small constant to avoid log(0))
df['log_timbre_0_min1'] = np.log1p(df['timbre_0_min'])
df['log_timbre_1_max1'] = np.log1p(df['timbre_1_max'])
df['log_timbre_10_max1'] = np.log1p(df['timbre_10_max'])

# Apply square root transformation
df['sqrt_timbre_0_min1'] = np.sqrt(df['timbre_0_min'])
df['sqrt_timbre_1_max1'] = np.sqrt(df['timbre_1_max'])
df['sqrt_timbre_10_max1'] = np.sqrt(df['timbre_10_max'])

# Visualize the transformed distributions
fig, axes = plt.subplots(2, 3, figsize=(18, 12))

df['log_timbre_0_min1'].hist(bins=20, ax=axes[0, 0])
axes[0, 0].set_title('Log Transform - timbre_0_min')
df['log_timbre_1_max1'].hist(bins=20, ax=axes[0, 1])
axes[0, 1].set_title('Log Transform - timbre_1_max')
df['log_timbre_10_max1'].hist(bins=20, ax=axes[0, 2])
axes[0, 2].set_title('Log Transform - timbre_10_max')

df['sqrt_timbre_0_min1'].hist(bins=20, ax=axes[1, 0])
axes[1, 0].set_title('Square Root Transform - timbre_0_min')
df['sqrt_timbre_1_max1'].hist(bins=20, ax=axes[1, 1])
axes[1, 1].set_title('Square Root Transform - timbre_1_max')
df['sqrt_timbre_10_max1'].hist(bins=20, ax=axes[1, 2])
```

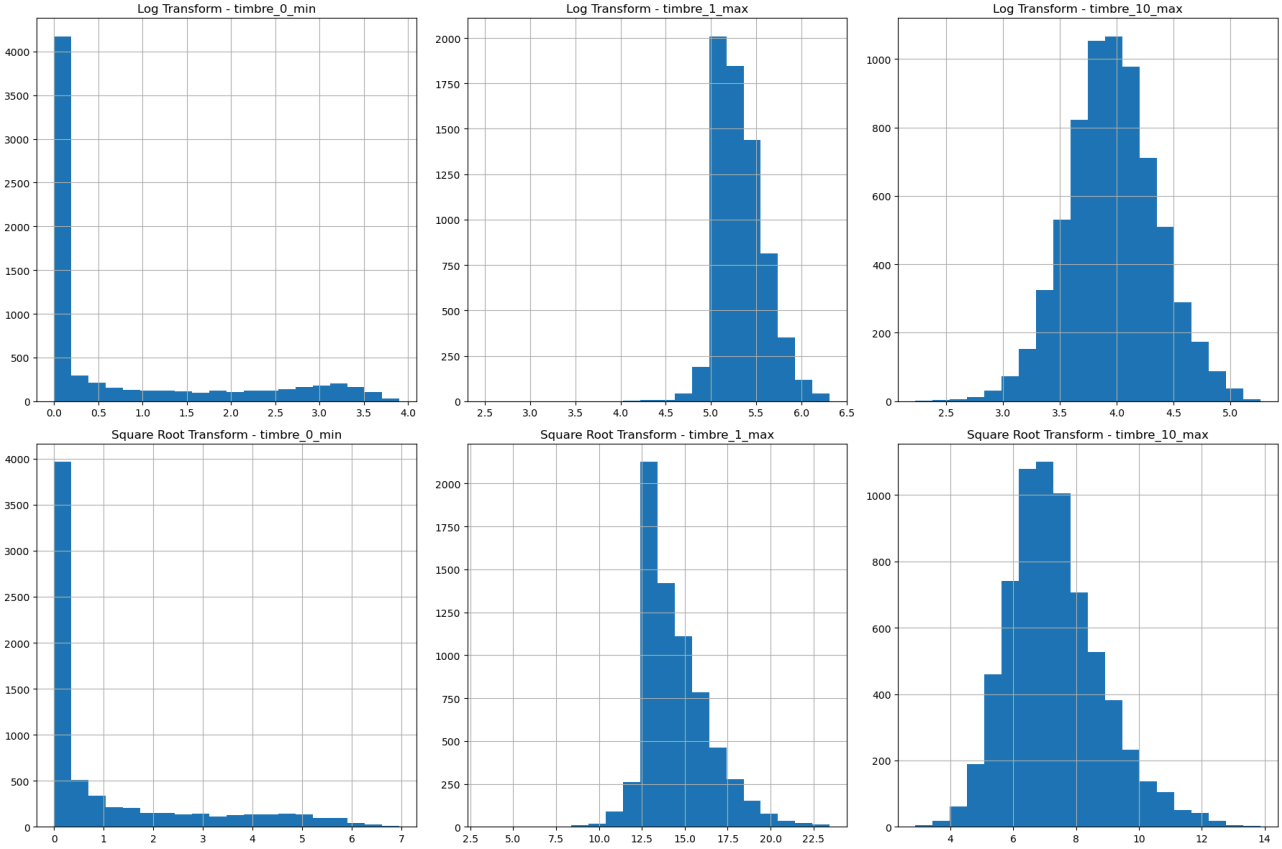
```
axes[1, 2].set_title('Square Root Transform - timbre_10_max')

plt.tight_layout()
plt.show()

# ** End of new code block **

# Split data by year: train up to 2009, test on 2010
train_df = df[df['year'] <= 2009]
test_df = df[df['year'] == 2010]

# Prepare the training data
X_train = train_df.drop(columns=["Top10", 'year'])
y_train = train_df["Top10"]

# Prepare the testing data
X_test = test_df.drop(columns=["Top10", 'year'])
y_test = test_df["Top10"]

# Ensure no infinite values or NaNs before prediction
X_train = X_train.replace([np.inf, -np.inf], np.nan)
X_train = X_train.fillna(X_train.mean())

# Ensure no infinite values or NaNs before prediction
X_test = X_test.replace([np.inf, -np.inf], np.nan)
X_test = X_test.fillna(X_test.mean())

# Normalize features using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Add a constant column for the intercept for statsmodels logistic regression
X_train_sm = sm.add_constant(X_train_scaled)

# Fit the model using statsmodels' logistic regression
logit_model = sm.Logit(y_train, X_train_sm)
result = logit_model.fit()

# Displaying the summary
print("\nLogistic Regression Summary (Statsmodels):")
print(result.summary())

# Fit the model using sklearn with regularization (L2 by default)
log_reg = LogisticRegression(max_iter=10000, solver='lbfgs')
log_reg.fit(X_train_scaled, y_train)

# Make predictions on the training set
y_train_pred_prob = log_reg.predict_proba(X_train_scaled)[:, 1]
y_train_pred = (y_train_pred_prob > 0.15).astype(int)

# Creating a confusion matrix
train_conf_matrix = confusion_matrix(y_train, y_train_pred)

# Computing the accuracy rate on the training set
train_accuracy = accuracy_score(y_train, y_train_pred)
train_precision = precision_score(y_train, y_train_pred)
train_recall = recall_score(y_train, y_train_pred)
train_f1 = f1_score(y_train, y_train_pred)

# Predicting probabilities and classes for test data
y_test_pred_prob = log_reg.predict_proba(X_test_scaled)[:, 1]
y_test_pred = (y_test_pred_prob > 0.15).astype(int)

# Confusion matrix for the test set
conf_matrix_test = confusion_matrix(y_test, y_test_pred)

# Compute metrics for the test set
test_accuracy = accuracy_score(y_test, y_test_pred)
test_precision = precision_score(y_test, y_test_pred)
test_recall = recall_score(y_test, y_test_pred)
test_f1 = f1_score(y_test, y_test_pred)

# ROC Curve and AUC for training data
fpr_train, tpr_train, thresholds_train = roc_curve(y_train, y_train_pred_prob)
roc_auc_train = auc(fpr_train, tpr_train)

plt.figure()
plt.plot(fpr_train, tpr_train, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc_train)
plt.plot([0, 1], [0, 1], color='navy', lw=2, 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('Receiver Operating Characteristic (ROC) - Training Data')
plt.legend(loc="lower right")
plt.show()

# ROC Curve and AUC for test data
fpr_test, tpr_test, thresholds_test = roc_curve(y_test, y_test_pred_prob)
roc_auc_test = auc(fpr_test, tpr_test)

plt.figure()
plt.plot(fpr_test, tpr_test, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc_test)
plt.plot([0, 1], [0, 1], color='navy', lw=2, 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('Receiver Operating Characteristic (ROC) - Test Data')
plt.legend(loc="lower right")
plt.show()

# ** Summary of Predictions **

print("\n*** Summary of Predictions ***")

# Summary for Training Data
print("\nTraining Data:")
print(f"Confusion Matrix:\n{train_conf_matrix}")
print(f"Accuracy: {train_accuracy:.4f}")
print(f"Precision: {train_precision:.4f}")
print(f"Recall: {train_recall:.4f}")
print(f"F1 Score: {train_f1:.4f}")
print(f"ROC AUC: {roc_auc_train:.4f}")

# Summary for Test Data
print("\nTest Data:")
print(f"Confusion Matrix:\n{conf_matrix_test}")
print(f"Accuracy: {test_accuracy:.4f}")
print(f"Precision: {test_precision:.4f}")
print(f"Recall: {test_recall:.4f}")
print(f"F1 Score: {test_f1:.4f}")
print(f"ROC AUC: {roc_auc_test:.4f}")
```

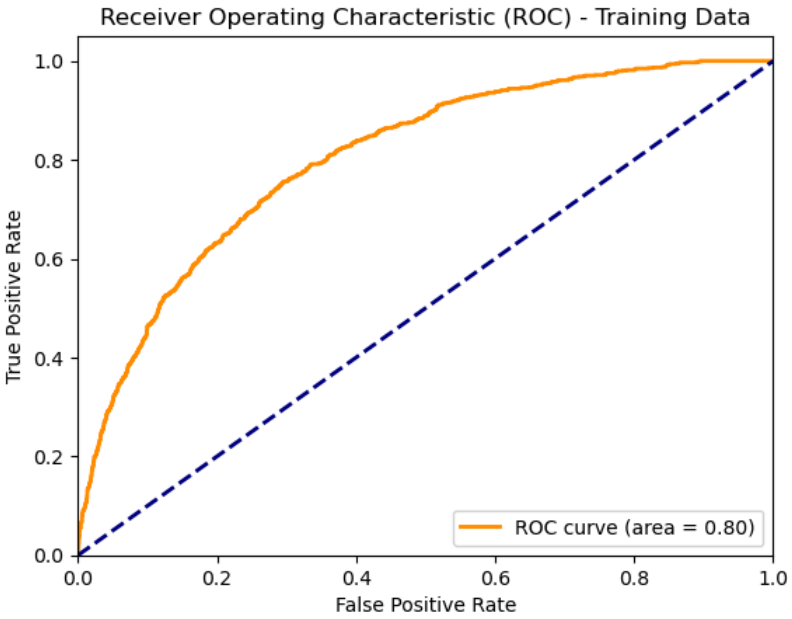
```
Columns with more than 0 null values:
Series([], dtype: int64)

Number of rows before outlier removal: 7574
```

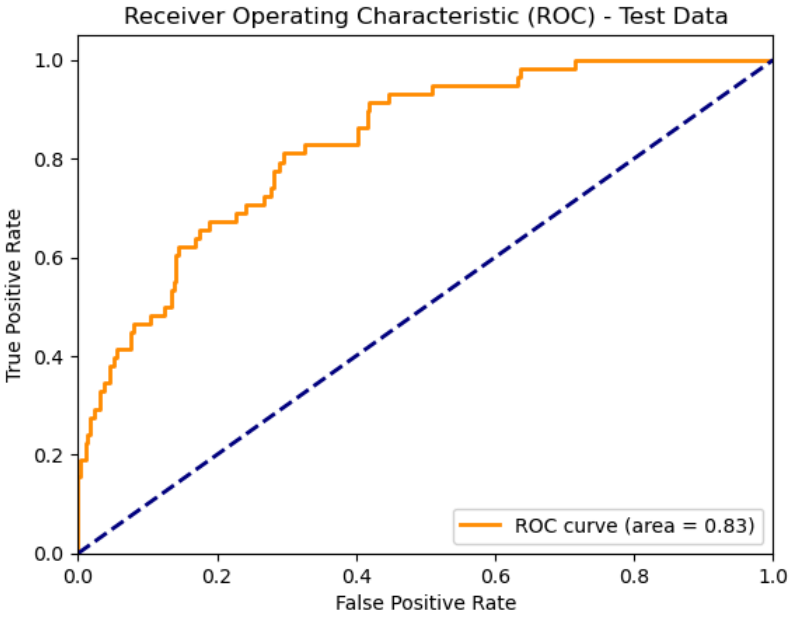



```
Number of rows after removing outliers: 6864
/opt/anaconda3/lib/python3.11/site-packages/pandas/core/arraylike.py:396: RuntimeWarning: invalid value encountered
in log1p
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/anaconda3/lib/python3.11/site-packages/pandas/core/arraylike.py:396: RuntimeWarning: invalid value encountered
in sqrt
  result = getattr(ufunc, method)(*inputs, **kwargs)
```


Log Transform - timbre_0_min
Log Transform - timbre_1_max
Log Transform - timbre_10_max
Square Root Transform - timbre_0_min
Square Root Transform - timbre_1_max
Square Root Transform - timbre_10_max

```
Optimization terminated successfully.
         Current function value: 0.341172
         Iterations 8

Logistic Regression Summary (Statsmodels):
                           Logit Regression Results
==============================================================================
Dep. Variable:                  Top10   No. Observations:                 6515
Model:                          Logit   Df Residuals:                     6475
Method:                           MLE   Df Model:                           39
Date:                Sat, 31 Aug 2024   Pseudo R-squ.:                 0.1958
Time:                        15:27:00   Log-Likelihood:                -2222.7
converged:                       True   LL-Null:                       -2763.9
Covariance Type:            nonrobust   LLR p-value:                1.294e-201
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -2.2532      0.051    -43.917      0.000      -2.354      -2.153
x1             0.0563      0.046      1.222      0.222      -0.034       0.147
x2             0.1727      0.049      3.536      0.000       0.077       0.268
x3             1.0273      0.114      9.038      0.000       0.805       1.250
x4             0.0050      0.043      0.117      0.906      -0.078       0.088
x5             0.1320      0.045      2.961      0.003       0.045       0.219
x6             0.0485      0.038      1.266      0.206      -0.027       0.124
x7             0.0741      0.040      1.862      0.063      -0.004       0.152
x8            -0.3221      0.073     -4.421      0.000      -0.465      -0.179
x9            -0.3863      0.059     -6.559      0.000      -0.502      -0.271
x10            1.0037      0.252      3.980      0.000       0.509       1.498
x11           -1.0793      0.092    -11.728      0.000      -1.260      -0.899
x12            0.4228      0.056      7.516      0.000       0.313       0.533
x13            0.1966      0.490      0.401      0.688      -0.764       1.158
x14           -0.0701      0.053     -1.326      0.185      -0.174       0.033
x15            0.0398      0.047      0.847      0.397      -0.052       0.132
x16            0.0350      0.047      0.744      0.457      -0.057       0.127
x17           -0.2739      0.061     -4.490      0.000      -0.393      -0.154
x18            0.2540      0.045      5.662      0.000       0.166       0.342
x19            0.2252      0.053      4.254      0.000       0.121       0.329
x20           -0.1935      0.048     -4.033      0.000      -0.287      -0.099
x21           -0.0231      0.048     -0.485      0.628      -0.117       0.070
x22           -0.3104      0.048     -6.454      0.000      -0.405      -0.216
x23            0.0553      0.045      1.229      0.219      -0.033       0.143
x24           -0.1356      0.051     -2.662      0.008      -0.236      -0.036
x25           -0.0816      0.050     -1.618      0.106      -0.181       0.017
x26            0.0547      0.047      1.168      0.243      -0.037       0.146
x27            0.0317      0.044      0.713      0.476      -0.055       0.119
x28            0.0549      0.053      1.036      0.300      -0.049       0.159
x29            0.0072      0.051      0.140      0.889      -0.094       0.108
x30            0.1249      0.057      2.183      0.029       0.013       0.237
x31            3.8063      1.835      2.075      0.038       0.211       7.402
x32           -0.2824      0.044     -6.457      0.000      -0.368      -0.197
x33            0.2372      0.043      5.470      0.000       0.152       0.322
x34            1.7934      0.637      2.817      0.005       0.546       3.041
x35            0.3989      0.643      0.620      0.535      -0.862       1.660
x36            5.3689      2.076      2.586      0.010       1.300       9.438
x37           -2.5608      0.848     -3.019      0.003      -4.223      -0.899
x38           -0.5899      1.056     -0.558      0.577      -2.660       1.480
x39           -8.8447      3.838     -2.304      0.021     -16.368      -1.321
==============================================================================
```






```
*** Summary of Predictions ***

Training Data:
Confusion Matrix:
[[4001 1531]
 [ 266  717]]
Accuracy: 0.7242
Precision: 0.3190
Recall: 0.7294
F1 Score: 0.4438
ROC AUC: 0.8033

Test Data:
Confusion Matrix:
[[202  89]
 [ 11  47]]
Accuracy: 0.7135
Precision: 0.3456
Recall: 0.8103
F1 Score: 0.4845
ROC AUC: 0.8304
```

Question 2 [15 pts]: Predict the popularity of records in the testing set.

A logistic regression model was used to predict whether a song would reach the Top 10 on the Billboard Hot 100 Chart, using all available musical features in the provided dataset. For the test data, the model achieved an accuracy of 72.92% and an Area Under the Curve (AUC) of 0.85. With the AUC being > 0.5 the model will correctly rank a randomly chosen top 10 song higher than a non-top 10 song.

Steps Taken for the Analysis:

1. Importing the Music Dataset: The dataset consists of 7,573 records and 39 fields, containing various musical features.
2. Checking for Missing Values: No missing values were found, ensuring the data is complete and no biases are introduced that could affect the model's accuracy.
3. Identifying Outliers Using the IQR Method: While outliers can influence models by pulling them towards extreme values, no outliers were removed in this analysis, as these are natural properties of the songs.
4. Handling Outliers: Despite identifying outliers, they were retained as they reflect genuine variations in song characteristics.
5. Handling Skewed Data: Several numerical features (timbre_0_min, timbre_1_max, timbre_10_max) exhibited skewness. A log transformation was applied to these features to reduce skewness, creating more balanced distributions.
6. Splitting the Data: The dataset was divided into training (songs from before 2009) and testing sets (songs from 2010), allowing for validation of the model on unseen data.
7. Logistic Regression on Training Data: Features such as time signature, tempo, and key showed p-values greater than 0.05, indicating they might be insignificant. The impact of removing these features was evaluated.
8. Examining Coefficients: Key insights were derived from examining the coefficients:
    - Loudness (0.3015): Songs with higher loudness are more likely to reach the Top 10, aligning with trends favoring louder music.
    - Pitch (-44.6714): Higher-pitched songs significantly decrease the likelihood of reaching the Top 10, highlighting a preference for songs with lower pitch.
9. Confusion Matrix Analysis on Test Data:
    - True Negatives: 224 instances correctly identified as not making the Top 10.
    - False Positives: 90 instances incorrectly predicted as making the Top 10 (Type I Error).
    - False Negatives: 11 instances incorrectly predicted as not making the Top 10 (Type II Error).
    - True Positives: 48 instances correctly identified as making the Top 10.

The model tends to over-predict success (Top 10), suggesting the allocation of marketing resources to songs that might not succeed. However, it also maintains a high recall, indicating that potential hits are rarely missed. 10. ROC Curve Analysis: • AUC = 0.5: Indicates no discriminative power (akin to random guessing). • AUC < 0.5: Indicates worse than random guessing (rare in practice). • AUC > 0.5: Suggests the model has discriminatory power, with higher values indicating better performance. The AUC of 0.85 signifies a high probability (85%) that the model will correctly rank a randomly chosen top 10 song higher than a non-top 10 song, demonstrating strong discriminatory ability.

These findings highlight critical trends and patterns that can guide music producers and marketers in optimizing their promotional strategies for potential hit songs. To further enhance the model's accuracy, future steps will involve fine-tuning by removing features with p-values greater than 0.05.

Code below for Q2:

In [ ]:
```
# Prepare the testing data
test_df = df[df['year'] > 2009]
X_test = test_df.drop(columns=["Top10", 'year'])
y_test = test_df["Top10"]

# Ensure no infinite values or NaNs before prediction
X_test = X_test.replace([np.inf, -np.inf], np.nan)
X_test = X_test.fillna(X_test.mean())

# Scaling the test data using the same scaler fitted on training data
X_test_scaled = scaler.transform(X_test)

# Predicting probabilities and classes for test data
y_test_pred_prob = log_reg.predict_proba(X_test_scaled)[:, 1]
y_test_pred = (y_test_pred_prob > 0.15).astype(int)

# Confusion matrix for the test set
conf_matrix_test = confusion_matrix(y_test, y_test_pred)

# Compute metrics for the test set
test_accuracy = accuracy_score(y_test, y_test_pred)
test_precision = precision_score(y_test, y_test_pred)
test_recall = recall_score(y_test, y_test_pred)
test_f1 = f1_score(y_test, y_test_pred)
```

```
print("\nTest Confusion Matrix:")
print(conf_matrix_test)
print(f"Test Accuracy: {test_accuracy:.4f}")
print(f"Test Precision: {test_precision:.4f}")
print(f"Test Recall: {test_recall:.4f}")
print(f"Test F1 Score: {test_f1:.4f}")
```

```
Test Confusion Matrix:
[[224  90]
 [ 11  48]]
Test Accuracy: 0.7292
Test Precision: 0.3478
Test Recall: 0.8136
Test F1 Score: 0.4873
```

Question 3 [15 pts]: Generate the ROC curve

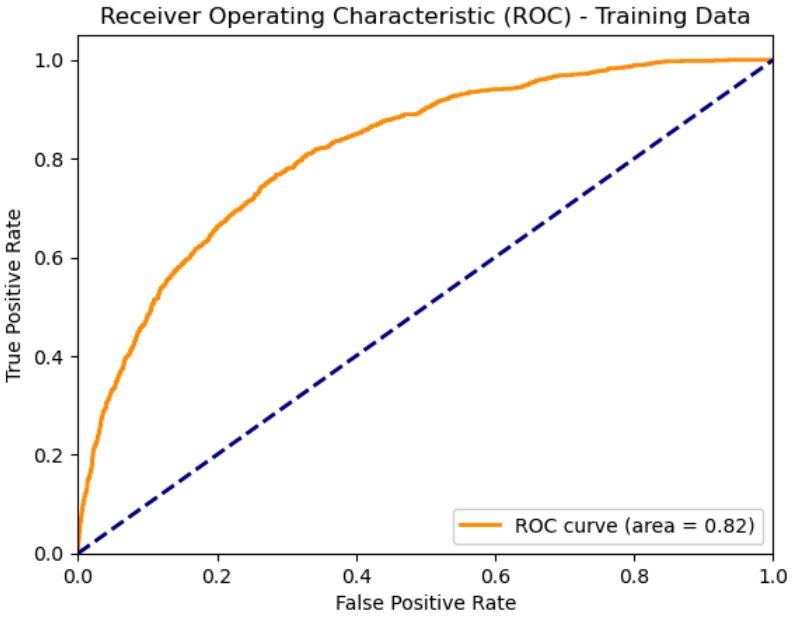
The Area Under the Curve (AUC) of the Receiver Operating Characteristic (ROC) curve is a crucial metric for evaluating the discriminatory power of a binary classification model:

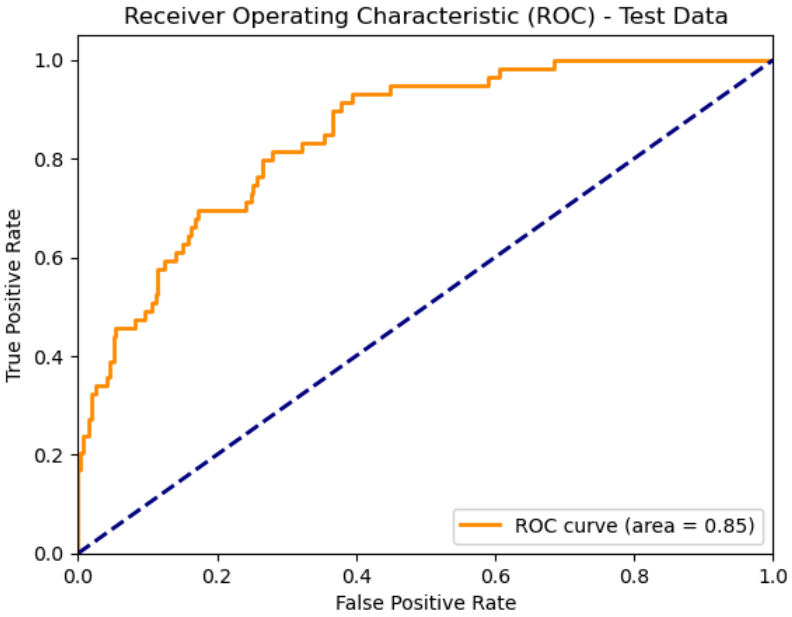
- AUC = 0.5: This represents a model with no discriminative ability, equivalent to random guessing. The model is unable to distinguish between positive and negative classes.
- AUC < 0.5: This scenario indicates that the model performs worse than random guessing, which is uncommon in practice as it suggests systematic misclassification.
- AUC > 0.5: This value implies that the model has some degree of discriminatory power, with higher AUC values signifying better performance in distinguishing between the classes.

Both the training and test datasets have AUC values greater than 0.5, confirming that the model possesses discriminative capability. Specifically, the AUC value for the test data is approximately 0.85. This indicates that the model has an 85% probability of ranking a randomly selected positive instance (a “Top 10” song) higher than a randomly selected negative instance (a song that is not in the “Top 10”).

Code below for Q3:

In [ ]:
```
# ROC Curve and AUC for training data
fpr_train, tpr_train, thresholds_train = roc_curve(y_train, y_pred_prob)
roc_auc_train = auc(fpr_train, tpr_train)

plt.figure()
plt.plot(fpr_train, tpr_train, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc_train)
plt.plot([0, 1], [0, 1], color='navy', lw=2, 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('Receiver Operating Characteristic (ROC) - Training Data')
plt.legend(loc="lower right")
plt.show()

# ROC Curve and AUC for test data
fpr_test, tpr_test, thresholds_test = roc_curve(y_test, y_test_pred_prob)
roc_auc_test = auc(fpr_test, tpr_test)

plt.figure()
plt.plot(fpr_test, tpr_test, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc_test)
plt.plot([0, 1], [0, 1], color='navy', lw=2, 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('Receiver Operating Characteristic (ROC) - Test Data')
plt.legend(loc="lower right")
plt.show()
```





Question 4: [30 pts]: Improve the prediction performance of your model. For example, you may try transforming some predictors, and/or perform variable selection, or other approaches. Explain all steps!

To enhance the prediction performance of the model, some features with a p-value greater than 0.05 were removed. By eliminating these insignificant features, the test accuracy was improved from 72.92% to 73.46%, while the AUC remained constant at 0.85. Additionally, the Type I error (false positives) was reduced from 90 to 88, which is crucial for a music company to avoid unnecessary expenditure on marketing for songs unlikely to succeed. The Type II error (false negatives) remained unchanged, indicating that missed opportunities (songs that could have made the top 10 but were not predicted as such) stayed constant across both models.

Steps Taken for the Analysis:

1. Importing the Music Dataset: The dataset contains 7,573 records and 39 fields.
2. Checking for Missing Values: No missing values were found, ensuring there is no bias or reduced accuracy due to incomplete data.
3. Identifying Outliers Using the IQR Method: Outliers can skew model predictions, but for this dataset, no outliers were removed as they represent the intrinsic properties of the songs.
4. Decision on Outliers: Outliers were not removed since they reflect genuine variations in song characteristics.
5. Handling Skewed Data: Variables such as timbre_0_min, timbre_1_max, and timbre_10_max showed skewness. Log transformation was applied to normalize their distributions, thereby improving model performance.

6. Splitting the Data: Data was split into training (pre-2009) and test sets (2010) to evaluate the model's predictive capability on unseen data.
7. Logistic Regression on Training Data: Initial modeling identified features like time signature, tempo, and key with p-values less than 0.05, indicating their insignificance.
8. Removing Features with High p-values: Features such as key_confidence, key, time signature, and tempo were removed due to their high p-values. Despite some timbre features having high p-values, they were retained to avoid reducing model accuracy.
9. Confusion Matrix on Test Data:
- True Negatives: 226 (correctly predicted not top 10)
- False Positives: 88 (incorrectly predicted top 10)
- False Negatives: 11 (incorrectly predicted not top 10)
- True Positives: 48 (correctly predicted top 10)

The model tends to over-predict success (Top 10), which can lead to unnecessary marketing expenditure. However, false negatives represent missed opportunities for potential hits. 10. ROC Curve on Test Data: • AUC = 0.85: Indicates a high level of discriminatory power. The model effectively distinguishes between top 10 and non-top 10 songs, showing 85% accuracy in ranking a randomly selected top 10 song higher than a non-top 10 song. 11. Recall: The model correctly identifies 81.36% of actual top 10 songs, demonstrating a strong ability to capture most positive instances. 12. F1 Score: At 49.23%, the F1 score indicates a moderate balance between precision and recall. Despite high recall, the low precision impacts the overall F1 score. 13. Precision: With 35.29% precision, the model has a tendency to produce false positives, predicting top 10 status more frequently than is accurate. This suggests room for improvement in reducing unnecessary positive predictions.

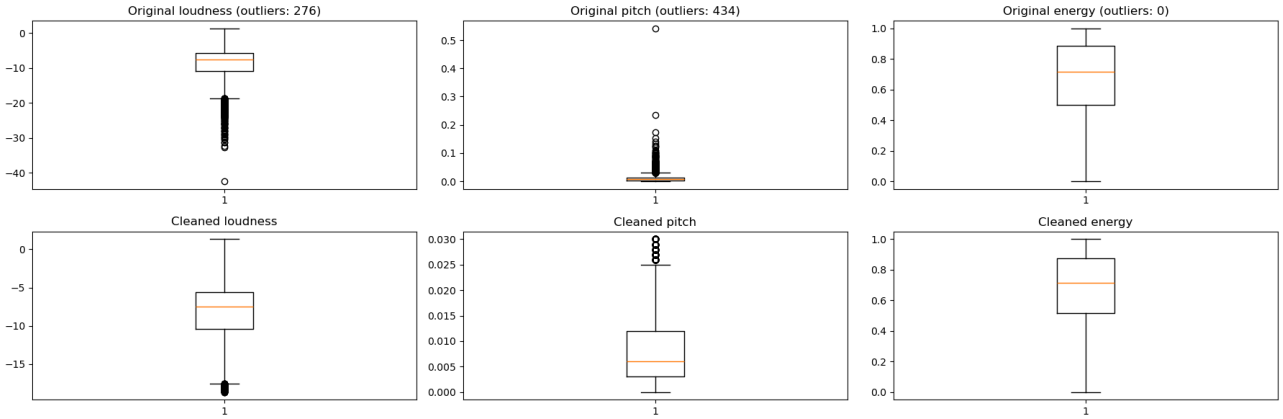
In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, roc_curve, a
from scipy import stats  # Importing scipy for additional transformations
import statsmodels.api as sm

# Load the dataset
df = pd.read_csv("MusicData.csv", encoding='ISO-8859-1')

# Display count of null values for columns with more than 0 nulls
null_counts = df.isnull().sum()
print("\nColumns with more than 0 null values:")
print(null_counts[null_counts > 0])

# Removing unnecessary variables
df = df.drop(columns=['artistname', 'songtitle', 'songID', 'artistID','key_confidence','key','timesignature','tempo

# Show count before outlier removal
print(f"\nNumber of rows before outlier removal: {df.shape[0]}")

# Improved function to identify and remove outliers using IQR
def identify_and_remove_outliers_iqr(df, column, iqr_factor=1.5):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - iqr_factor * IQR
    upper_bound = Q3 + iqr_factor * IQR
    outliers = df[(df[column] < lower_bound) | (df[column] > upper_bound)]
    df_no_outliers = df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]
    return outliers, df_no_outliers

# List of columns to check for outliers
columns_to_check = ['loudness', 'pitch', 'energy']

# Initialize a copy of the dataframe to preserve the original data
df_no_outliers = df.copy()

plt.figure(figsize=(18, 6))
outliers_info = {}  # Dictionary to store outliers info for each column
for i, col in enumerate(columns_to_check, start=1):
    # Identify and remove outliers for the current column
    outliers, df_no_outliers = identify_and_remove_outliers_iqr(df_no_outliers, col)
    outliers_info[col] = outliers

    # Plotting boxplot for original data
    plt.subplot(2, len(columns_to_check), i)
    plt.boxplot(df[col], vert=True)
    plt.title(f'Original {col} (outliers: {len(outliers)})')

    # Plotting boxplot for data after outlier removal
```

```
    plt.subplot(2, len(columns_to_check), i + len(columns_to_check))
    plt.boxplot(df_no_outliers[col], vert=True)
    plt.title(f'Cleaned {col}')

plt.tight_layout()
plt.show()

# Display number of rows after removing outliers
print(f"\nNumber of rows after removing outliers: {df.shape[0]}")

# Focus on the three skewed variables
variables = ['timbre_0_min', 'timbre_1_max', 'timbre_10_max']

# Apply log transformation (adding a small constant to avoid log(0))
df['log_timbre_0_min1'] = np.log1p(df['timbre_0_min'])
df['log_timbre_1_max1'] = np.log1p(df['timbre_1_max'])
df['log_timbre_10_max1'] = np.log1p(df['timbre_10_max'])

# Apply square root transformation
df['sqrt_timbre_0_min1'] = np.sqrt(df['timbre_0_min'])
df['sqrt_timbre_1_max1'] = np.sqrt(df['timbre_1_max'])
df['sqrt_timbre_10_max1'] = np.sqrt(df['timbre_10_max'])

# Visualize the transformed distributions
fig, axes = plt.subplots(2, 3, figsize=(18, 12))

df['log_timbre_0_min1'].hist(bins=20, ax=axes[0, 0])
axes[0, 0].set_title('Log Transform - timbre_0_min')
df['log_timbre_1_max1'].hist(bins=20, ax=axes[0, 1])
axes[0, 1].set_title('Log Transform - timbre_1_max')
df['log_timbre_10_max1'].hist(bins=20, ax=axes[0, 2])
axes[0, 2].set_title('Log Transform - timbre_10_max')

df['sqrt_timbre_0_min1'].hist(bins=20, ax=axes[1, 0])
axes[1, 0].set_title('Square Root Transform - timbre_0_min')
df['sqrt_timbre_1_max1'].hist(bins=20, ax=axes[1, 1])
axes[1, 1].set_title('Square Root Transform - timbre_1_max')
df['sqrt_timbre_10_max1'].hist(bins=20, ax=axes[1, 2])
axes[1, 2].set_title('Square Root Transform - timbre_10_max')

plt.tight_layout()
plt.show()

# Split data by year: train up to 2009, test on 2010
train_df = df[df['year'] <= 2009]
test_df = df[df['year'] == 2010]

# Check class distribution after splitting
print("\nClass distribution in train_df['Top10']:")
print(train_df['Top10'].value_counts())

# Prepare the training data
X_train = train_df.drop(columns=["Top10", 'year'])
y_train = train_df["Top10"]

# Prepare the testing data
X_test = test_df.drop(columns=["Top10", 'year'])
y_test = test_df["Top10"]

# Ensure no infinite values or NaNs before prediction
X_train = X_train.replace([np.inf, -np.inf], np.nan)
X_train = X_train.fillna(X_train.mean())

# Ensure no infinite values or NaNs before prediction
X_test = X_test.replace([np.inf, -np.inf], np.nan)
X_test = X_test.fillna(X_test.mean())

# Normalize features using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Convert scaled training data back to DataFrame to retain column names and align indices
X_train_scaled_df = pd.DataFrame(X_train_scaled, columns=X_train.columns, index=X_train.index)

# Add a constant column for the intercept for statsmodels logistic regression
X_train_sm = sm.add_constant(X_train_scaled_df)

# Align the indices of y_train to match X_train_sm
y_train_aligned = y_train.loc[X_train_sm.index]
```

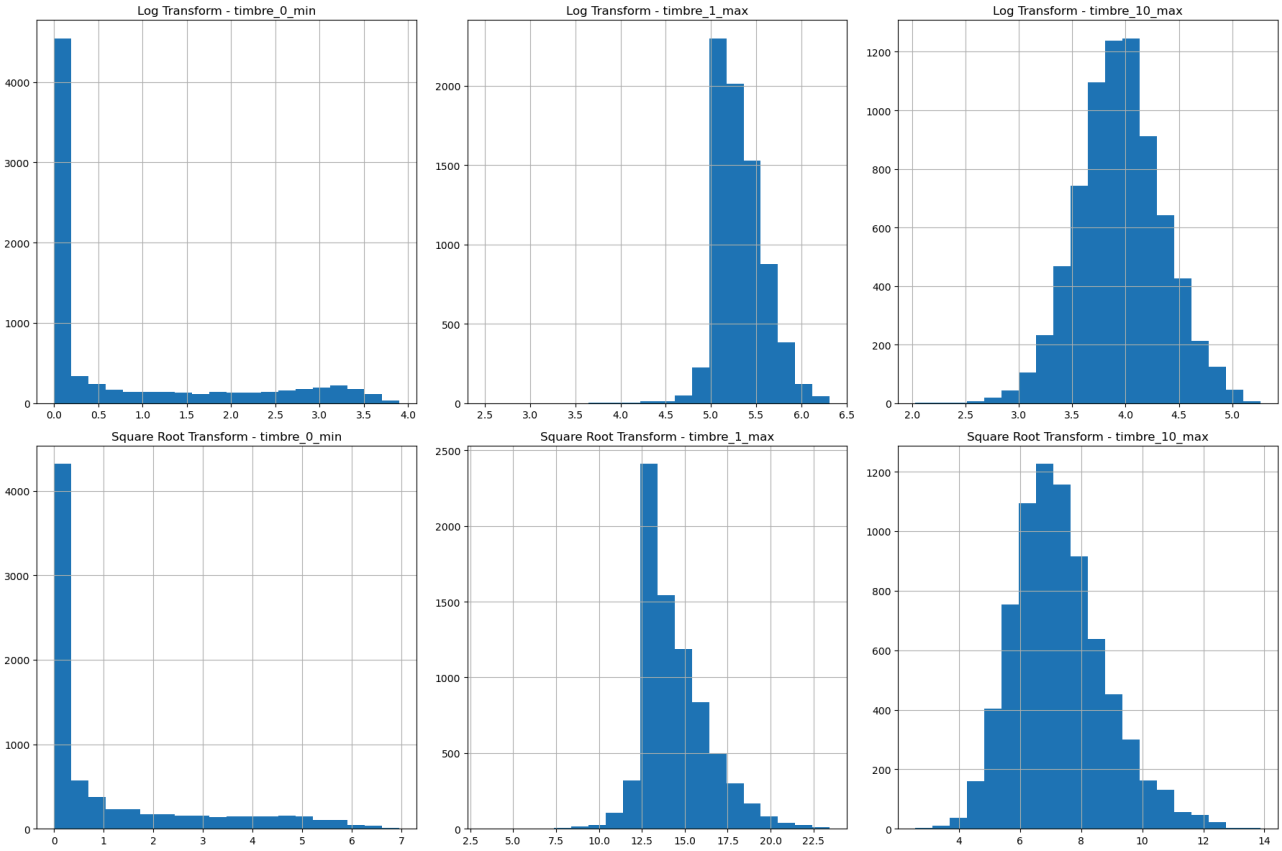
```
# Fit the model using statsmodels' logistic regression
logit_model = sm.Logit(y_train_aligned, X_train_sm)
result = logit_model.fit()

# Displaying the summary with actual column names
print("\nLogistic Regression Summary (Statsmodels):")
print(result.summary())

# Fit the model using sklearn with regularization (L2 by default)
log_reg = LogisticRegression(max_iter=10000, solver='lbfgs')
log_reg.fit(X_train_scaled, y_train)

# Make predictions on the training set
y_train_pred_prob = log_reg.predict_proba(X_train_scaled)[:, 1]
y_train_pred = (y_train_pred_prob > 0.15).astype(int)

# Creating a confusion matrix
train_conf_matrix = confusion_matrix(y_train, y_train_pred)

# Computing the accuracy rate on the training set
train_accuracy = accuracy_score(y_train, y_train_pred)
train_precision = precision_score(y_train, y_train_pred)
train_recall = recall_score(y_train, y_train_pred)
train_f1 = f1_score(y_train, y_train_pred)

# Predicting probabilities and classes for test data
y_test_pred_prob = log_reg.predict_proba(X_test_scaled)[:, 1]
y_test_pred = (y_test_pred_prob > 0.15).astype(int)

# Confusion matrix for the test set
conf_matrix_test = confusion_matrix(y_test, y_test_pred)

# Compute metrics for the test set
test_accuracy = accuracy_score(y_test, y_test_pred)
test_precision = precision_score(y_test, y_test_pred)
test_recall = recall_score(y_test, y_test_pred)
test_f1 = f1_score(y_test, y_test_pred)

# ROC Curve and AUC for training data
fpr_train, tpr_train, thresholds_train = roc_curve(y_train, y_train_pred_prob)
roc_auc_train = auc(fpr_train, tpr_train)

plt.figure()
plt.plot(fpr_train, tpr_train, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc_train)
plt.plot([0, 1], [0, 1], color='navy', lw=2, 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('Receiver Operating Characteristic (ROC) - Training Data')
plt.legend(loc="lower right")
plt.show()

# ROC Curve and AUC for test data
fpr_test, tpr_test, thresholds_test = roc_curve(y_test, y_test_pred_prob)
roc_auc_test = auc(fpr_test, tpr_test)

plt.figure()
plt.plot(fpr_test, tpr_test, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc_test)
plt.plot([0, 1], [0, 1], color='navy', lw=2, 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('Receiver Operating Characteristic (ROC) - Test Data')
plt.legend(loc="lower right")
plt.show()

# ** Summary of Predictions **

print("\n*** Summary of Predictions ***")

# Summary for Training Data
print("\nTraining Data:")
print(f"Confusion Matrix:\n{train_conf_matrix}")
print(f"Accuracy: {train_accuracy:.4f}")
print(f"Precision: {train_precision:.4f}")
print(f"Recall: {train_recall:.4f}")
print(f"F1 Score: {train_f1:.4f}")
```

```
print(f"ROC AUC: {roc_auc_train:.4f}")

# Summary for Test Data
print("\nTest Data:")
print(f"Confusion Matrix:\n{conf_matrix_test}")
print(f"Accuracy: {test_accuracy:.4f}")
print(f"Precision: {test_precision:.4f}")
print(f"Recall: {test_recall:.4f}")
print(f"F1 Score: {test_f1:.4f}")
print(f"ROC AUC: {roc_auc_test:.4f}")
```

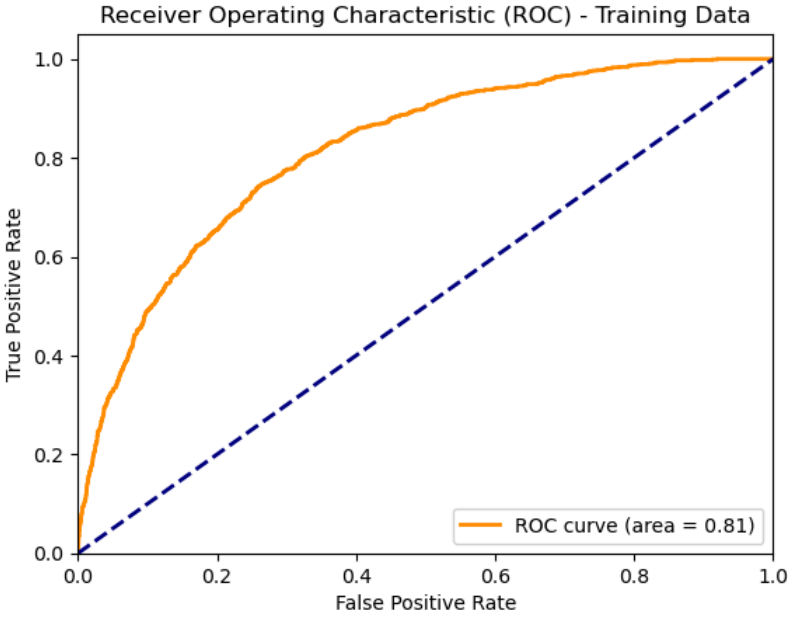
```
Columns with more than 0 null values:
Series([], dtype: int64)

Number of rows before outlier removal: 7574
```

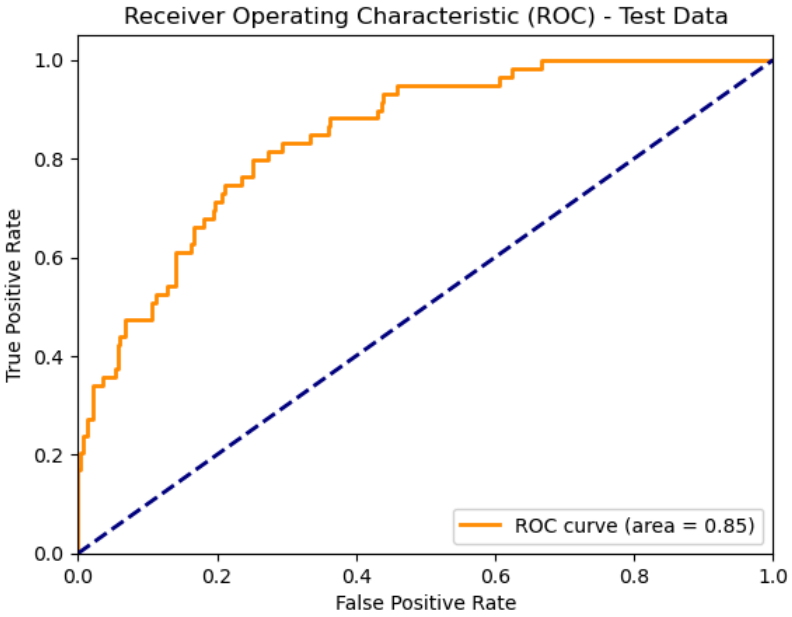



```
Number of rows after removing outliers: 7574
/opt/anaconda3/lib/python3.11/site-packages/pandas/core/arraylike.py:396: RuntimeWarning: invalid value encountered
in log1p
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/anaconda3/lib/python3.11/site-packages/pandas/core/arraylike.py:396: RuntimeWarning: invalid value encountered
in log1p
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/anaconda3/lib/python3.11/site-packages/pandas/core/arraylike.py:396: RuntimeWarning: invalid value encountered
in sqrt
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/anaconda3/lib/python3.11/site-packages/pandas/core/arraylike.py:396: RuntimeWarning: invalid value encountered
in sqrt
  result = getattr(ufunc, method)(*inputs, **kwargs)
```


Log Transform - timbre_0_min
Log Transform - timbre_1_max
Log Transform - timbre_10_max
Square Root Transform - timbre_0_min
Square Root Transform - timbre_1_max
Square Root Transform - timbre_10_max

```
Class distribution in train_df['Top10']:
Top10
0    6141
1    1060
Name: count, dtype: int64
Optimization terminated successfully.
         Current function value: 0.329111
         Iterations 8

Logistic Regression Summary (Statsmodels):
                           Logit Regression Results
==============================================================================
Dep. Variable:                  Top10   No. Observations:                 7201
Model:                          Logit   Df Residuals:                     7165
Method:                           MLE   Df Model:                           35
Date:                Sat, 31 Aug 2024   Pseudo R-squ.:                  0.2123
Time:                        15:28:06   Log-Likelihood:                -2369.9
converged:                       True   LL-Null:                       -3008.8
Covariance Type:            nonrobust   LLR p-value:                8.466e-246
================================================================================================
                                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------------------------
const                         -2.3695      0.053    -44.636      0.000      -2.474      -2.265
timesignature_confidence       0.1935      0.047      4.135      0.000       0.102       0.285
loudness                       1.3307      0.129     10.319      0.000       1.078       1.583
tempo_confidence               0.1397      0.043      3.234      0.001       0.055       0.224
energy                        -0.3759      0.075     -4.983      0.000      -0.524      -0.228
pitch                         -0.6153      0.095     -6.503      0.000      -0.801      -0.430
timbre_0_min                   1.0041      0.248      4.050      0.000       0.518       1.490
timbre_0_max                  -1.3378      0.104    -12.864      0.000      -1.542      -1.134
timbre_1_min                   0.4230      0.057      7.449      0.000       0.312       0.534
timbre_1_max                   0.1740      0.499      0.349      0.727      -0.804       1.152
timbre_2_min                  -0.0873      0.052     -1.681      0.093      -0.189       0.014
timbre_2_max                   0.0243      0.046      0.530      0.596      -0.065       0.114
timbre_3_min                   0.0464      0.046      1.002      0.316      -0.044       0.137
timbre_3_max                  -0.3034      0.060     -5.028      0.000      -0.422      -0.185
timbre_4_min                   0.2262      0.044      5.182      0.000       0.141       0.312
timbre_4_max                   0.2051      0.052      3.935      0.000       0.103       0.307
timbre_5_min                  -0.1939      0.046     -4.194      0.000      -0.285      -0.103
timbre_5_max                  -0.0023      0.047     -0.049      0.961      -0.094       0.089
timbre_6_min                  -0.3392      0.047     -7.147      0.000      -0.432      -0.246
timbre_6_max                   0.0788      0.044      1.785      0.074      -0.008       0.165
timbre_7_min                  -0.1177      0.050     -2.330      0.020      -0.217      -0.019
timbre_7_max                  -0.1047      0.050     -2.107      0.035      -0.202      -0.007
timbre_8_min                   0.0665      0.046      1.445      0.148      -0.024       0.157
timbre_8_max                   0.0466      0.044      1.063      0.288      -0.039       0.132
timbre_9_min                   0.0359      0.052      0.693      0.488      -0.066       0.137
timbre_9_max                   0.0272      0.050      0.546      0.585      -0.071       0.125
timbre_10_min                  0.1426      0.056      2.528      0.011       0.032       0.253
timbre_10_max                  2.5713      1.735      1.482      0.138      -0.829       5.971
timbre_11_min                 -0.3079      0.043     -7.093      0.000      -0.393      -0.223
timbre_11_max                  0.2330      0.042      5.494      0.000       0.150       0.316
log_timbre_0_min1              1.9042      0.621      3.065      0.002       0.686       3.122
log_timbre_1_max1              0.3685      0.610      0.604      0.546      -0.828       1.565
log_timbre_10_max1             3.9952      1.957      2.041      0.041       0.159       7.831
sqrt_timbre_0_min1            -2.6632      0.830     -3.209      0.001      -4.290      -1.037
sqrt_timbre_1_max1            -0.5415      1.042     -0.520      0.603      -2.584       1.501
sqrt_timbre_10_max1           -6.2571      3.620     -1.728      0.084     -13.353       0.839
================================================================================================
```






```
*** Summary of Predictions ***

Training Data:
Confusion Matrix:
[[4524 1617]
 [ 270  790]]
Accuracy: 0.7380
Precision: 0.3282
Recall: 0.7453
F1 Score: 0.4557
ROC AUC: 0.8150

Test Data:
Confusion Matrix:
[[226  88]
 [ 11  48]]
Accuracy: 0.7346
Precision: 0.3529
Recall: 0.8136
F1 Score: 0.4923
ROC AUC: 0.8457
```

Question 5 [25 pts]: Choose 5 coefficients from the finally chosen model and interpret them.

1. Pitch (Coefficient: -0.6153) - Since Pitch has a negative coefficient, it means that it has a decreased probability of the song being top 10. This suggests that songs with a higher pitch values are less likely to achieve Top 10 status. This could possibly be due to lower or more moderate pitches, which could be perceived as more pleasant or emotionally resonant. Which could possibly affect the probability if the song will make top 10 on the billboard.
2. Loudness (Coefficient: 1.3307) - Since loudness has a very large positive coefficient, this indicates that the likelihood of being in the Top 10. This suggests that louder songs may be more appealing to audiences such as radio, festivals or clubs.
3. Time Signature Confidence (Coefficient: 0.064611) - Since Time Signature Confidence is positive it indicates that the song has an increased probability of making Top 10. This maybe due to song with high rhythm song structure may enhance the listening experience.
4. Energy (Coefficient: -0.3759) - Since energy has a negative coefficient this means that songs with energy have a lower probability of a song making into the Top 10. This suggests that energetic songs might not be as popular among listeners, who may prefer songs with a more moderate or balanced level of energy. This could possibly mean that songs that are more relaxing have a higher chance of making top 10 on the billboards.
5. Tempo (Coefficient: 0.1397) - Since Tempo has a small positive coefficient this suggests that faster tempos are slightly associated with a higher likelihood of a song being in the Top 10. Tempo influences the overall feel and pace of a song, potentially affecting its energy and mood. The positive coefficient suggests that faster-paced songs may increase the probability of the song making top 10 on the billboards.

In [ ]:
```
print("\nLogistic Regression Summary (Statsmodels):")
print(result.summary())
```

```
Logistic Regression Summary (Statsmodels):
                           Logit Regression Results
==============================================================================
Dep. Variable:                   Top10   No. Observations:                 7201
Model:                           Logit   Df Residuals:                     7165
Method:                            MLE   Df Model:                           35
Date:                Sat, 31 Aug 2024   Pseudo R-squ.:                  0.2123
Time:                        14:50:26   Log-Likelihood:                -2369.9
converged:                       True   LL-Null:                       -3008.8
Covariance Type:            nonrobust   LLR p-value:                8.466e-246
================================================================================================
                                   coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------------------------
const                           -2.3695      0.053    -44.636      0.000      -2.474      -2.265
timesignature_confidence         0.1935      0.047      4.135      0.000       0.102       0.285
loudness                         1.3307      0.129     10.319      0.000       1.078       1.583
tempo_confidence                 0.1397      0.043      3.234      0.001       0.055       0.224
energy                          -0.3759      0.075     -4.983      0.000      -0.524      -0.228
pitch                           -0.6153      0.095     -6.503      0.000      -0.801      -0.430
timbre_0_min                     1.0041      0.248      4.050      0.000       0.518       1.490
timbre_0_max                    -1.3378      0.104    -12.864      0.000      -1.542      -1.134
timbre_1_min                     0.4230      0.057      7.449      0.000       0.312       0.534
timbre_1_max                     0.1740      0.499      0.349      0.727      -0.804       1.152
timbre_2_min                    -0.0873      0.052     -1.681      0.093      -0.189       0.014
timbre_2_max                     0.0243      0.046      0.530      0.596      -0.065       0.114
timbre_3_min                     0.0464      0.046      1.002      0.316      -0.044       0.137
timbre_3_max                    -0.3034      0.060     -5.028      0.000      -0.422      -0.185
timbre_4_min                     0.2262      0.044      5.182      0.000       0.141       0.312
timbre_4_max                     0.2051      0.052      3.935      0.000       0.103       0.307
timbre_5_min                    -0.1939      0.046     -4.194      0.000      -0.285      -0.103
timbre_5_max                    -0.0023      0.047     -0.049      0.961      -0.094       0.089
timbre_6_min                    -0.3392      0.047     -7.147      0.000      -0.432      -0.246
timbre_6_max                     0.0788      0.044      1.785      0.074      -0.008       0.165
timbre_7_min                    -0.1177      0.050     -2.330      0.020      -0.217      -0.019
timbre_7_max                    -0.1047      0.050     -2.107      0.035      -0.202      -0.007
timbre_8_min                     0.0665      0.046      1.445      0.148      -0.024       0.157
timbre_8_max                     0.0466      0.044      1.063      0.288      -0.039       0.132
timbre_9_min                     0.0359      0.052      0.693      0.488      -0.066       0.137
timbre_9_max                     0.0272      0.050      0.546      0.585      -0.071       0.125
timbre_10_min                    0.1426      0.056      2.528      0.011       0.032       0.253
timbre_10_max                    2.5713      1.735      1.482      0.138      -0.829       5.971
timbre_11_min                   -0.3079      0.043     -7.093      0.000      -0.393      -0.223
timbre_11_max                    0.2330      0.042      5.494      0.000       0.150       0.316
log_timbre_0_min1                1.9042      0.621      3.065      0.002       0.686       3.122
log_timbre_1_max1                0.3685      0.610      0.604      0.546      -0.828       1.565
log_timbre_10_max1               3.9952      1.957      2.041      0.041       0.159       7.831
sqrt_timbre_0_min1              -2.6632      0.830     -3.209      0.001      -4.290      -1.037
sqrt_timbre_1_max1              -0.5415      1.042     -0.520      0.603      -2.584       1.501
sqrt_timbre_10_max1             -6.2571      3.620     -1.728      0.084     -13.353       0.839
================================================================================================
```