Spotify Popularity Regression Analysis

Kiluva James

2025-06-12

Table of contents


```
# ================================#
## RF, XGBoost, and GBM PREDICTIVE CODES ##
# ================================#

## Load the dataset
try:
    spotify_charts_2024 = pd.read_csv("universal_top_spotify_songs.new.csv")
    # If the above fails, try a full path
except FileNotFoundError:
    print("CSV file not found in current directory. Please provide the full path.")



### ------------------------------
## DATA CLEANING
### ------------------------------

# market counts for each song
spotify_charts_2024['market_count'] = spotify_charts_2024.groupby('spotify_id')['country'].t

# Print current state (optional, for debugging)
print("Before removing non-string rows:")
```

```
print(spotify_charts_2024[['spotify_id', 'country']].head()) # Or just print the whole df if
print(spotify_charts_2024['country'].apply(type).value_counts()) # Check types

# Remove rows where 'country' is not a string
# We also check for pd.NA which is a nullable type for object columns in newer pandas
spotify_charts_2024 = spotify_charts_2024[
    spotify_charts_2024['country'].apply(lambda x: isinstance(x, str) or pd.isna(x))
].copy() # .copy() to avoid SettingWithCopyWarning later

# After removing, you might want to convert the column to string type to be safe for future
spotify_charts_2024['country'] = spotify_charts_2024['country'].astype(str)

print("After removing non-string rows and converting to string:")
print(spotify_charts_2024[['spotify_id', 'country']].head())
print(spotify_charts_2024['country'].apply(type).value_counts()) # Should only show <class '

# Now, your original line for 'other_charted_countries' should work smoothly:
spotify_charts_2024['other_charted_countries'] = spotify_charts_2024.groupby('spotify_id')['
    lambda x: ', '.join(x.unique())
)
spotify_charts_2024 = spotify_charts_2024.sort_values(by='popularity', ascending=False).drop_
 keep='first').reset_index(drop=True)

# Function to count the number of artists in each row
spotify_charts_2024['artist_count'] = spotify_charts_2024['artists'].apply(lambda x: len(str

# Convert character date columns to Date objects using mdy()
spotify_charts_2024['snapshot_date'] = pd.to_datetime(spotify_charts_2024['snapshot_date'])
spotify_charts_2024['album_release_date'] = pd.to_datetime(spotify_charts_2024['album_releas
spotify_charts_2024['days_out'] = (spotify_charts_2024['snapshot_date'] - spotify_charts_202

# change boolean into integers
spotify_charts_2024['is_explicit'] = spotify_charts_2024['is_explicit'].astype(int)

# Standardize duration_ms (convert to minutes)
spotify_charts_2024['duration_min'] = spotify_charts_2024['duration_ms'] / 60000
spotify_charts_2024 = spotify_charts_2024.drop(columns=['duration_ms']) # Remove original co

# Remove unneeded columns
columns_to_drop = [
    'country', 'other_charted_countries', 'snapshot_date', 'name',
    'artists', 'album_name', 'album_release_date', 'spotify_id'
```

```
]
spotify_modelling = spotify_charts_2024.drop(columns=columns_to_drop, errors='ignore')

# check for missing values
print("Missing values before handling:")
print(spotify_modelling.isnull().sum())

# Handle missing values by imputing with mean (for numerical columns)
# For categorical columns, you'd typically use mode or a constant, but given the R code's
# `mutate_all(~ifelse(is.na(.), mean(., na.rm = TRUE), .))` and the `select` later,
# it implies all remaining columns are numeric or will be treated as such for imputation.
for col in spotify_modelling.columns:
    if spotify_modelling[col].dtype in ['int64', 'float64']:
        spotify_modelling[col].fillna(spotify_modelling[col].mean(), inplace=True)
    else:
       spotify_modelling[col].fillna(spotify_modelling[col].mode()[0], inplace=True)

print("Missing values after handling:")
print(spotify_modelling.isnull().sum())

# Arrange the columns - Python DataFrames maintain column order
# This step is important to match the R code's column order for operations like ggpairs
ordered_columns = [
    'popularity', 'days_out', 'artist_count', 'market_count', 'daily_rank',
    'daily_movement', 'weekly_movement', 'duration_min', 'is_explicit', 'mode',
    'danceability', 'energy', 'loudness', 'speechiness', 'acousticness',
    'instrumentalness', 'liveness', 'valence', 'tempo', 'key', 'time_signature'
]
spotify_modelling = spotify_modelling[ordered_columns]


# Remove popularity 0
spotify_modelling = spotify_modelling[spotify_modelling['popularity'] != 0].sort_values(by='
```

```
Before removing non-string rows:
               spotify_id country
0  2plbrEY59IikOBgBGLjaoe     NaN
1  4wJ5Qq0jBN4ajy7ouZIV1c     NaN
2  2CGNAOSuO1MEFCbBRgUzjd     NaN
3  6AI3ezQ4o3HUoP6Dhudph3     NaN
4  6dOtVTDdiauQNBQEDOtlAB     NaN
country
```

```
<class 'str'>      1726125
<class 'float'>      23907
Name: count, dtype: int64
After removing non-string rows and converting to string:
               spotify_id country
0  2plbrEY59IikOBgBGLjaoe     nan
1  4wJ5Qq0jBN4ajy7ouZIV1c     nan
2  2CGNAOSuO1MEFCbBRgUzjd     nan
3  6AI3ezQ4o3HUoP6Dhudph3     nan
4  6dOtVTDdiauQNBQEDOtlAB     nan
country
<class 'str'>    1750032
Name: count, dtype: int64
Missing values before handling:
daily_rank           0
daily_movement       0
weekly_movement      0
popularity           0
is_explicit          0
danceability         0
energy               0
key                  0
loudness             0
mode                 0
speechiness          0
acousticness         0
instrumentalness     0
liveness             0
valence              0
tempo                0
time_signature       0
market_count         0
artist_count         0
days_out            12
duration_min         0
dtype: int64
Missing values after handling:
daily_rank          0
daily_movement      0
weekly_movement     0
popularity          0
is_explicit         0
danceability        0
```

```
energy              0
key                 0
loudness            0
mode                0
speechiness         0
acousticness        0
instrumentalness    0
liveness            0
valence             0
tempo               0
time_signature      0
market_count        0
artist_count        0
days_out            0
duration_min        0
dtype: int64

C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\1316865436.py:78: FutureWarni
The behavior will change in pandas 3.0. This inplace method will never work because the inte

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: va


  spotify_modelling[col].fillna(spotify_modelling[col].mean(), inplace=True)
```

==========

3. DESCRIPTIVE STATISTICS, ANOVA AND CORRELATION. ==========

```python
# ------------------------------
### DESCRIPTIVE STATISTICS
# ------------------------------------

# Function to compute basic statistics
def spotify_stats(column):
    stats_dict = {
        'Mean': column.mean(),
        'Median': column.median(),
        'SD': column.std(),
        'Variance': column.var(),
        'IQR': column.quantile(0.75) - column.quantile(0.25)
    }
    return pd.Series(stats_dict)
```

```
# Loop through columns and compute statistics
stats_results = spotify_modelling.apply(spotify_stats).T # .T for transpose to match R's outp

print("Descriptive Statistics:")
print(stats_results)

# Plot 1: Popularity to daily_rank
plot_cols_1 = spotify_modelling.columns[0:5] # Columns 1 to 5 (0-indexed in Python)
fig1 = sns.pairplot(spotify_modelling[plot_cols_1])
fig1.fig.suptitle('Scatter Plot Matrix 1 (Popularity to Daily Rank)', y=1.02) # Adjust title
plt.show()
plt.close(fig1.fig) # Close the figure to free memory

# Plot 2: daily_movement to mode
plot_cols_2 = spotify_modelling.columns[5:10] # Columns 6 to 10
fig2 = sns.pairplot(spotify_modelling[plot_cols_2])
fig2.fig.suptitle('Scatter Plot Matrix 2 (Daily Movement to Mode)', y=1.02)
plt.show()
plt.close(fig2.fig)

# Plot 3: danceability to instrumentalness
plot_cols_3 = spotify_modelling.columns[10:15] # Columns 11 to 15
fig3 = sns.pairplot(spotify_modelling[plot_cols_3])
fig3.fig.suptitle('Scatter Plot Matrix 3 (Danceability to Instrumentalness)', y=1.02)
plt.show()
plt.close(fig3.fig)

# Plot 4: liveness to time_signature
plot_cols_4 = spotify_modelling.columns[15:21] # Columns 16 to 21
fig4 = sns.pairplot(spotify_modelling[plot_cols_4])
fig4.fig.suptitle('Scatter Plot Matrix 4 (Liveness to Time Signature)', y=1.02)
plt.show()
plt.close(fig4.fig)

# check popularity distribution
plt.figure(figsize=(10, 6))
sns.histplot(spotify_modelling['popularity'], bins=50, kde=True, color='skyblue')
plt.title('Popularity Distribution After Cleaning')
plt.xlabel('Popularity')
plt.ylabel('Frequency')
plt.show()
plt.close() # Close the figure
```

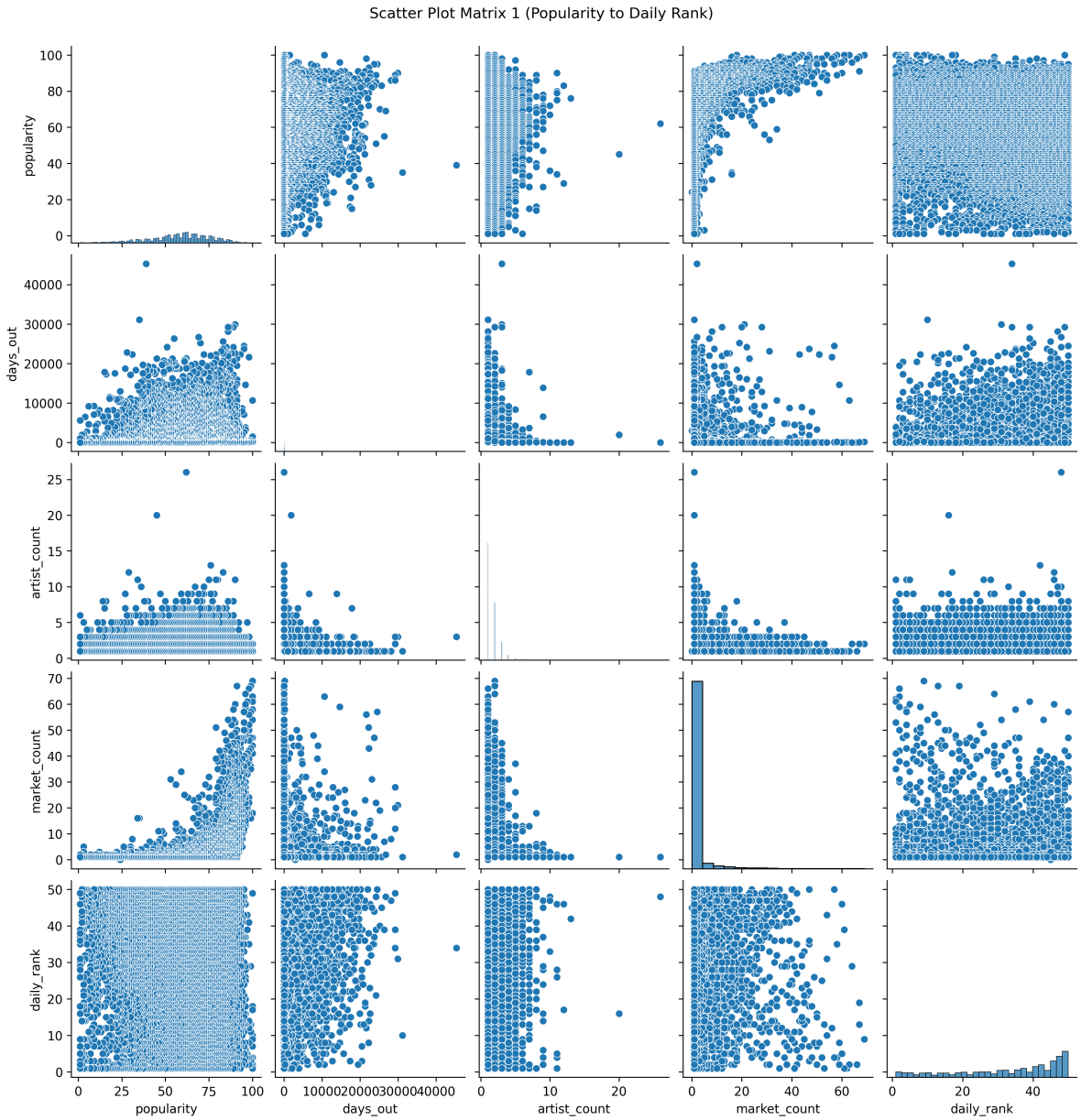
```
# ---------------------------
### CORRELATION
# -------------------------
# Select the audio feature columns
audio_features = spotify_modelling[[
    'popularity', 'market_count', 'daily_rank', 'daily_movement', 'weekly_movement',
    'days_out', 'artist_count', 'duration_min', 'is_explicit', 'mode',
    'danceability', 'energy', 'loudness', 'speechiness', 'acousticness',
    'instrumentalness', 'time_signature', 'liveness', 'valence', 'key', 'tempo'
]]
# Compute correlation matrix
correlation_matrix = audio_features.corr(numeric_only=True)

# Plot heatmap with correlation values
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5,
            cbar_kws={'label': 'Correlation Coefficient'})
plt.title("Correlation Heatmap", fontsize=16)
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
plt.close()

# -------------------------
### ANOVA FOR VARIANCE IMPORTANCE
# ---------------------------

# Convert relevant columns to factors in Python (object/category dtype for categorical)
# We'll make a copy to avoid modifying the original spotify_modelling for other parts
spotify_modelling_anova = spotify_modelling.copy()
categorical_cols = ['mode', 'is_explicit', 'key', 'time_signature']
for col in categorical_cols:
    if col in spotify_modelling_anova.columns:
        spotify_modelling_anova[col] = spotify_modelling_anova[col].astype('category')

anova_results = []
for col in spotify_modelling_anova.columns:
    if col == 'popularity':
        continue # Skip target variable

    formula = f'popularity ~ C({col})' if spotify_modelling_anova[col].dtype == 'category' el
```

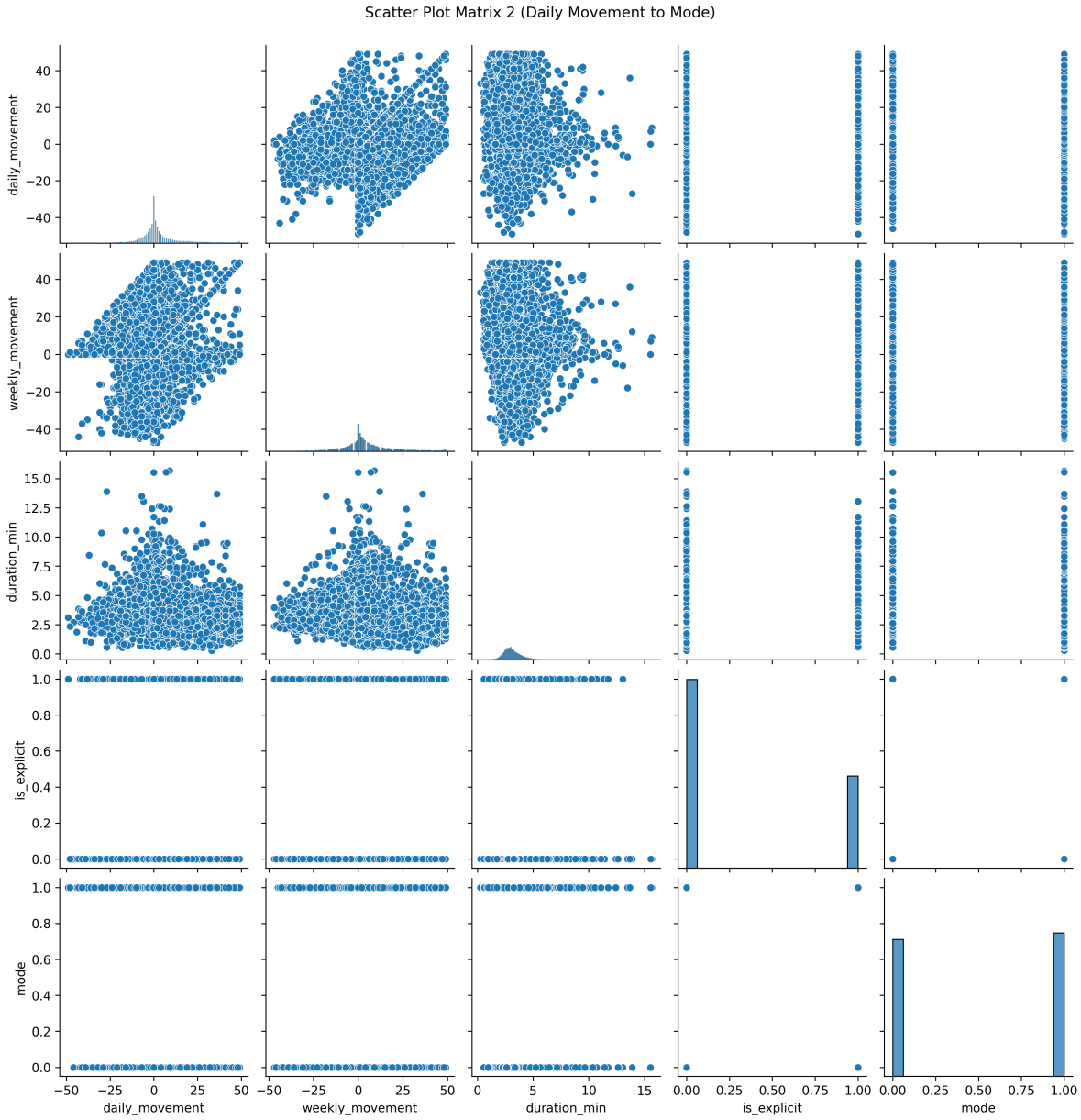
```
    try:
        model = ols(formula, data=spotify_modelling_anova).fit()
        anova_table = sm.stats.anova_lm(model, typ=2) # Type 2 ANOVA for unbalanced designs
        f_value = anova_table['F'][0]
        p_value = anova_table['PR(>F)'][0]
        anova_results.append({'Feature': col, 'F_Value': f_value, 'P_Value': p_value})
    except Exception as e:
        print(f"Could not perform ANOVA/LM for column {col}: {e}")
        anova_results.append({'Feature': col, 'F_Value': np.nan, 'P_Value': np.nan})

# Create a data frame for variance importance based on ANOVA
anova_importance = pd.DataFrame(anova_results)
anova_importance.dropna(subset=['F_Value'], inplace=True) # Remove rows where F_Value is NA
anova_importance = anova_importance.sort_values(by='F_Value', ascending=False).reset_index(dr

print("ANOVA Variance Importance (Combined):")
print(anova_importance)

# --- Plotting ANOVA results ---

# 1. Bar plot of p-values
plt.figure(figsize=(12, 7))
sns.barplot(x='Feature', y='P_Value', data=anova_importance, palette='viridis')
plt.axhline(y=0.05, color='red', linestyle='--', label='Significance (0.05)')
plt.title('P-values from ANOVA and Linear Regression')
plt.xlabel('Feature')
plt.ylabel('P-value')
plt.xticks(rotation=45, ha='right')
plt.legend()
plt.tight_layout()
plt.show()
plt.close()

# 2. Scatter plot of F-values vs. p-values
plt.figure(figsize=(12, 7))
sns.scatterplot(x='F_Value', y='P_Value', data=anova_importance, hue='Feature', s=100, palett
for i, row in anova_importance.iterrows():
    plt.text(row['F_Value'] + 0.1, row['P_Value'], row['Feature'], fontsize=8)
plt.axhline(y=0.05, color='red', linestyle='--', label='Significance (0.05)')
plt.title('F-values vs. P-values')
plt.xlabel('F-value')
plt.ylabel('P-value')
```

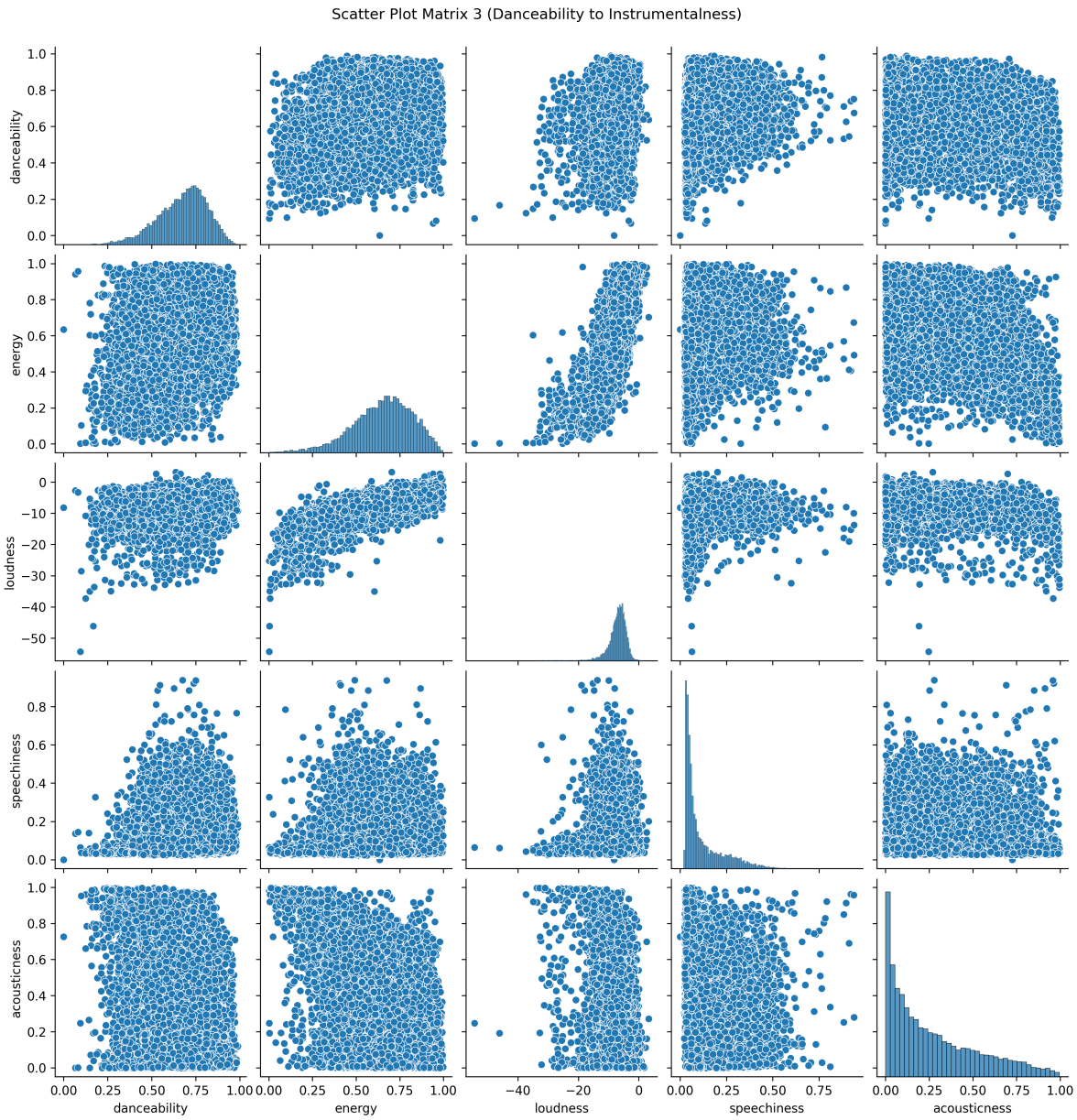
```
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
plt.close()
```

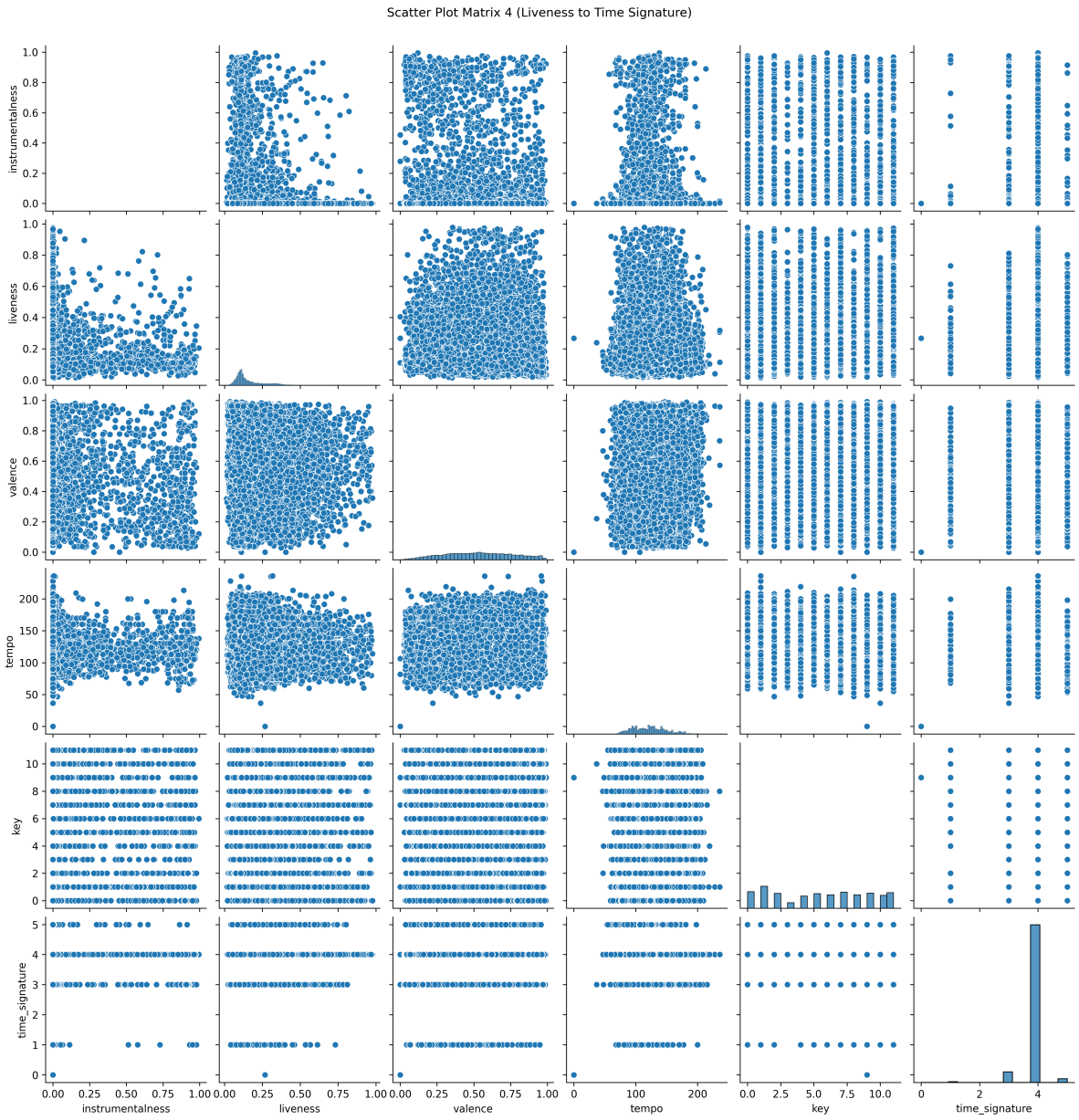
```
Descriptive Statistics:
                         Mean      Median           SD      Variance  \
popularity          58.216856   60.000000    17.932957  3.215910e+02
days_out           942.045425   39.000000  2718.615353  7.390869e+06
artist_count         1.631012    1.000000     1.000867  1.001735e+00
market_count         2.201072    1.000000     4.725443  2.232982e+01
daily_rank          33.333946   38.000000    14.713720  2.164936e+02
daily_movement       2.905743    0.000000    12.264597  1.504203e+02
weekly_movement      4.500620    2.000000    14.583514  2.126789e+02
duration_min         3.184053    3.028083     0.985339  9.708934e-01
is_explicit          0.327840    0.000000     0.469438  2.203720e-01
mode                 0.511590    1.000000     0.499878  2.498781e-01
danceability         0.673112    0.691000     0.140192  1.965372e-02
energy               0.650164    0.668000     0.170803  2.917380e-02
loudness            -7.076410   -6.582000     3.195620  1.021199e+01
speechiness          0.120909    0.069500     0.112578  1.267391e-02
acousticness         0.277781    0.201000     0.251338  6.317067e-02
instrumentalness     0.027357    0.000001     0.129016  1.664518e-02
liveness             0.182934    0.127000     0.139376  1.942577e-02
valence              0.534793    0.534000     0.229416  5.263188e-02
tempo              122.338718  121.031000    27.866344  7.765331e+02
key                  5.372562    6.000000     3.600920  1.296662e+01
time_signature       3.945649    4.000000     0.348977  1.217853e-01

                          IQR
popularity          23.000000
days_out           221.000000
artist_count         1.000000
market_count         0.000000
daily_rank          23.500000
daily_movement       7.000000
weekly_movement     12.000000
duration_min         0.995108
is_explicit          1.000000
mode                 1.000000
danceability         0.191000
```

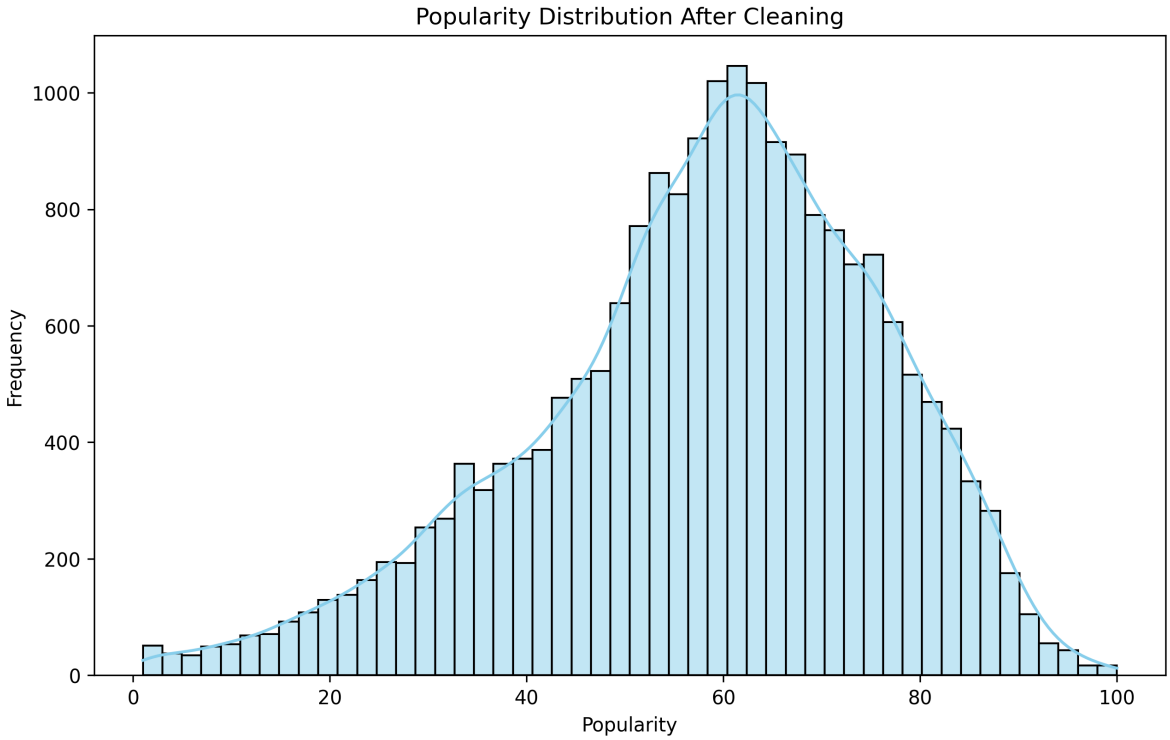
```
energy              0.231000
loudness            3.208000
speechiness         0.122600
acousticness        0.374350
instrumentalness    0.000137
liveness            0.126600
valence             0.356000
tempo              40.051500
key                 7.000000
time_signature      0.000000
```

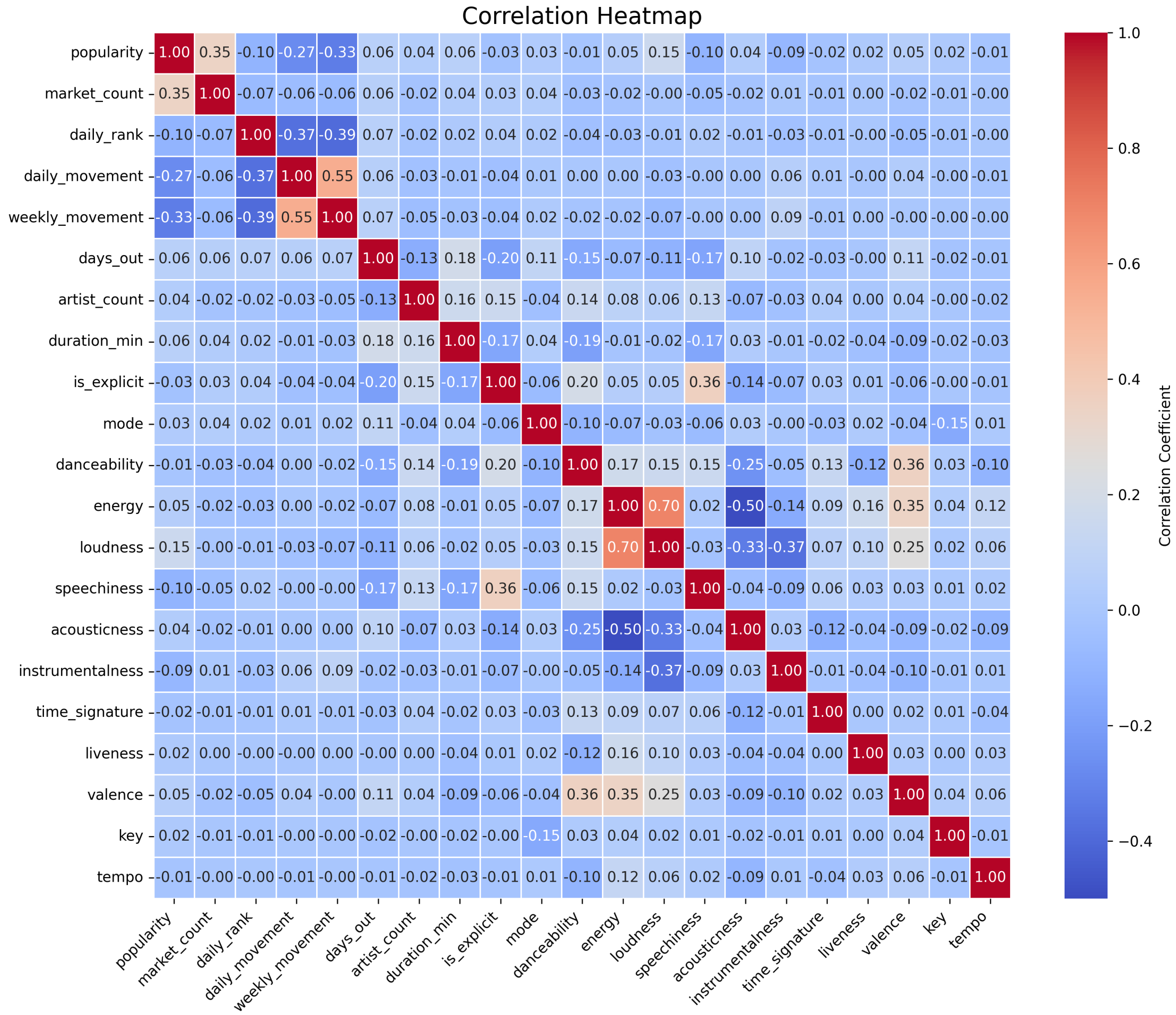

Scatter Plot Matrix 1 (Popularity to Daily Rank)
popularity
days_out
artist_count
market_count
daily_rank


Scatter Plot Matrix 2 (Daily Movement to Mode)
daily_movement
weekly_movement
duration_min
is_explicit
mode


Scatter Plot Matrix 3 (Danceability to Instrumentalness)
danceability
energy
loudness
speechiness
acousticness


Scatter Plot Matrix 4 (Liveness to Time Signature)
instrumentalness
liveness
valence
tempo
key
time_signature


Popularity Distribution After Cleaning
Frequency
Popularity
0
200
400
600
800
1000
0
20
40
60
80
100



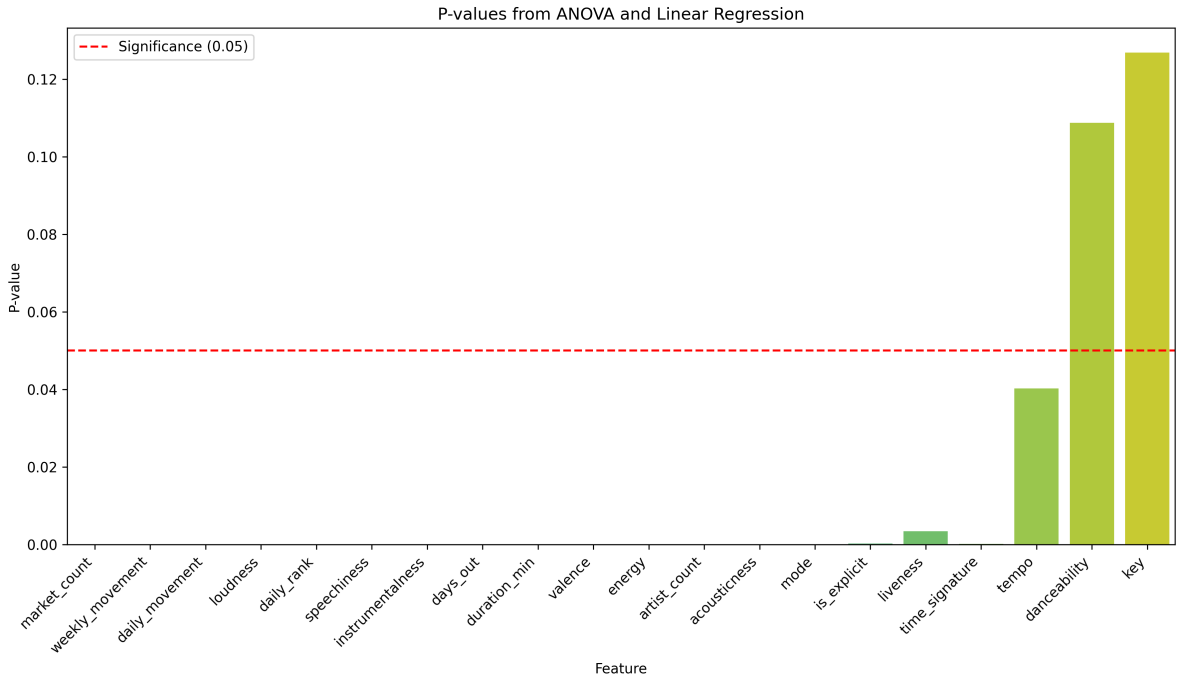

```
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:104: FutureWarn
  f_value = anova_table['F'][0]
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:105: FutureWarn
  p_value = anova_table['PR(>F)'][0]
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:104: FutureWarn
  f_value = anova_table['F'][0]
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:105: FutureWarn
  p_value = anova_table['PR(>F)'][0]
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:104: FutureWarn
  f_value = anova_table['F'][0]
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:105: FutureWarn
  p_value = anova_table['PR(>F)'][0]
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:104: FutureWarn
```

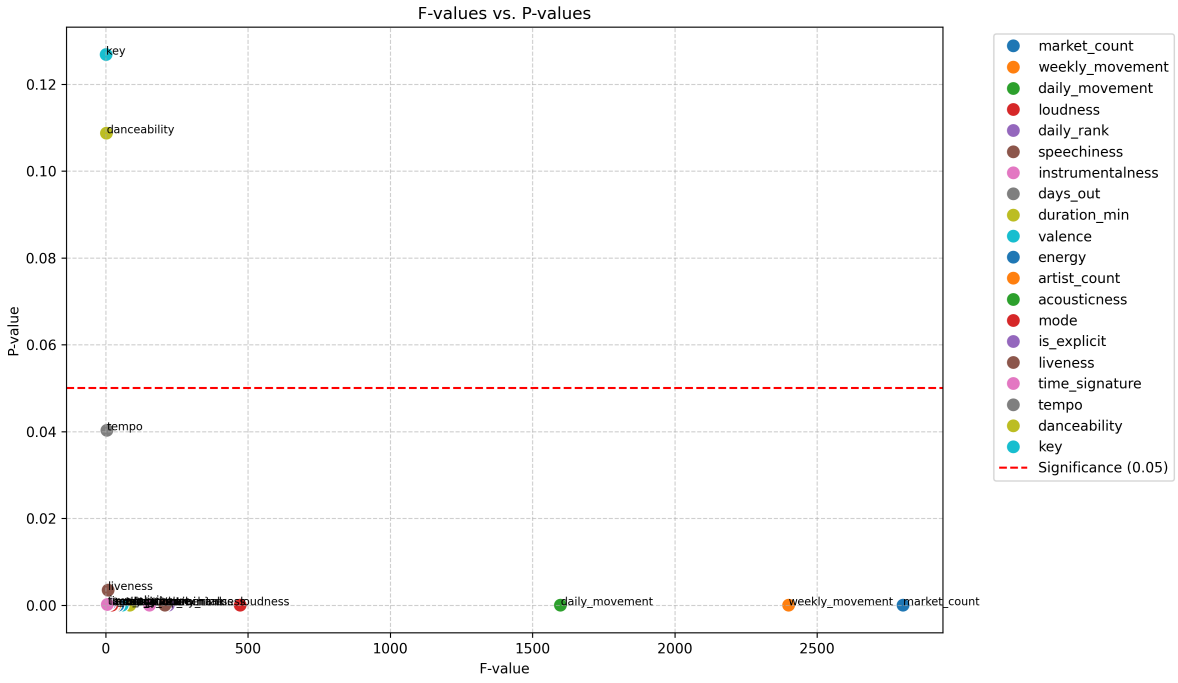
```
  f_value = anova_table['F'][0]
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:105: FutureWarn
  p_value = anova_table['PR(>F)'][0]
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:104: FutureWarn
  f_value = anova_table['F'][0]
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:105: FutureWarn
  p_value = anova_table['PR(>F)'][0]
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:104: FutureWarn
  f_value = anova_table['F'][0]
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:105: FutureWarn
  p_value = anova_table['PR(>F)'][0]
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:104: FutureWarn
  f_value = anova_table['F'][0]
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:105: FutureWarn
  p_value = anova_table['PR(>F)'][0]
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:104: FutureWarn
  f_value = anova_table['F'][0]
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:105: FutureWarn
  p_value = anova_table['PR(>F)'][0]
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:104: FutureWarn
  f_value = anova_table['F'][0]
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:105: FutureWarn
  p_value = anova_table['PR(>F)'][0]
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:104: FutureWarn
  f_value = anova_table['F'][0]
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:105: FutureWarn
  p_value = anova_table['PR(>F)'][0]
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:104: FutureWarn
  f_value = anova_table['F'][0]
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:105: FutureWarn
  p_value = anova_table['PR(>F)'][0]
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:104: FutureWarn
  f_value = anova_table['F'][0]
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:105: FutureWarn
  p_value = anova_table['PR(>F)'][0]
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:104: FutureWarn
  f_value = anova_table['F'][0]
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:105: FutureWarn
  p_value = anova_table['PR(>F)'][0]
```

```
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:104: FutureWarn
  f_value = anova_table['F'][0]
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:105: FutureWarn
  p_value = anova_table['PR(>F)'][0]
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:104: FutureWarn
  f_value = anova_table['F'][0]
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:105: FutureWarn
  p_value = anova_table['PR(>F)'][0]
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:104: FutureWarn
  f_value = anova_table['F'][0]
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:105: FutureWarn
  p_value = anova_table['PR(>F)'][0]
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:104: FutureWarn
  f_value = anova_table['F'][0]
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:105: FutureWarn
  p_value = anova_table['PR(>F)'][0]
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:104: FutureWarn
  f_value = anova_table['F'][0]
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:105: FutureWarn
  p_value = anova_table['PR(>F)'][0]
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:104: FutureWarn
  f_value = anova_table['F'][0]
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:105: FutureWarn
  p_value = anova_table['PR(>F)'][0]

ANOVA Variance Importance (Combined):
             Feature      F_Value        P_Value
0       market_count  2802.118963   0.000000e+00
1    weekly_movement  2399.985319   0.000000e+00
2     daily_movement  1598.022570   0.000000e+00
3           loudness   472.545990  1.386045e-103
4         daily_rank   219.741565   1.877037e-49
5        speechiness   208.137340   6.003006e-47
6   instrumentalness   153.807716   3.429673e-35
7           days_out    84.952470   3.348759e-20
8       duration_min    84.798300   3.619138e-20
9            valence    57.860831   2.936628e-14
10            energy    43.021824   5.544713e-11
11      artist_count    31.795854   1.735529e-08
12      acousticness    25.090714   5.515806e-07
13              mode    21.596230   3.386430e-06
14       is_explicit    13.087214   2.980451e-04
```

```
15          liveness     8.543747   3.470968e-03
16    time_signature     5.635186   1.572127e-04
17             tempo     4.206484   4.028262e-02
18      danceability     2.572726   1.087359e-01
19               key     1.491359   1.268530e-01
```

```
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\3067862557.py:123: FutureWarn
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assig
  sns.barplot(x='Feature', y='P_Value', data=anova_importance, palette='viridis')
```





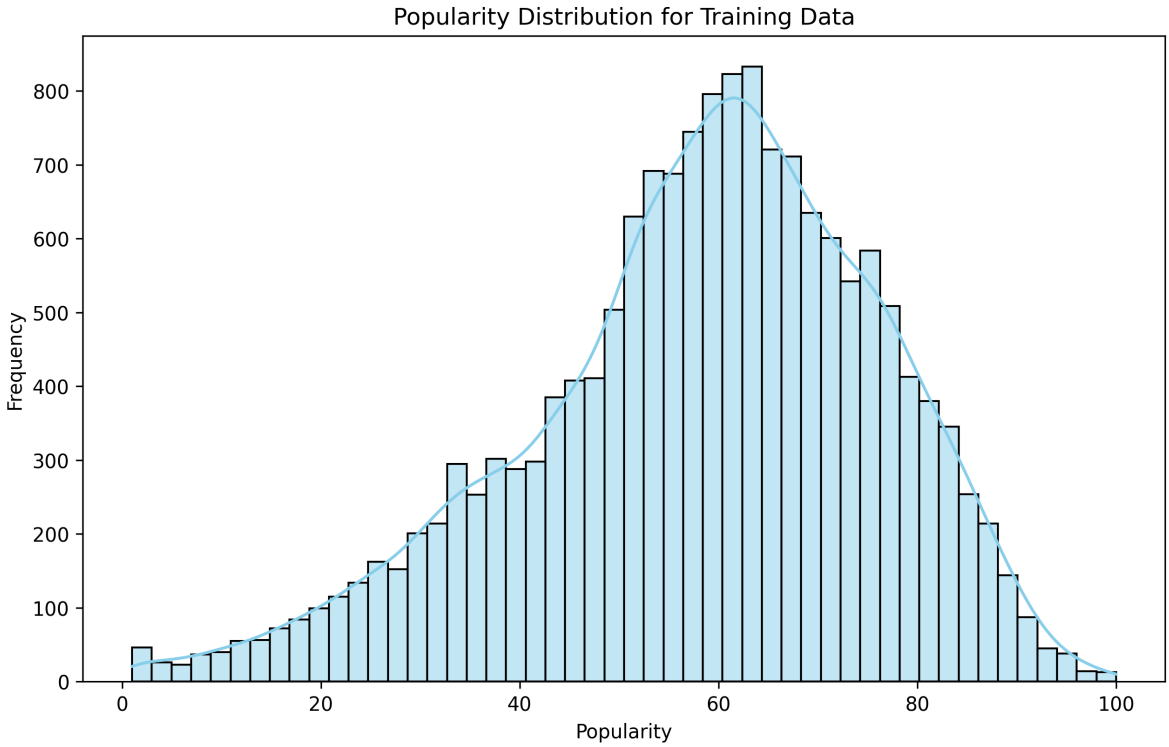
==========

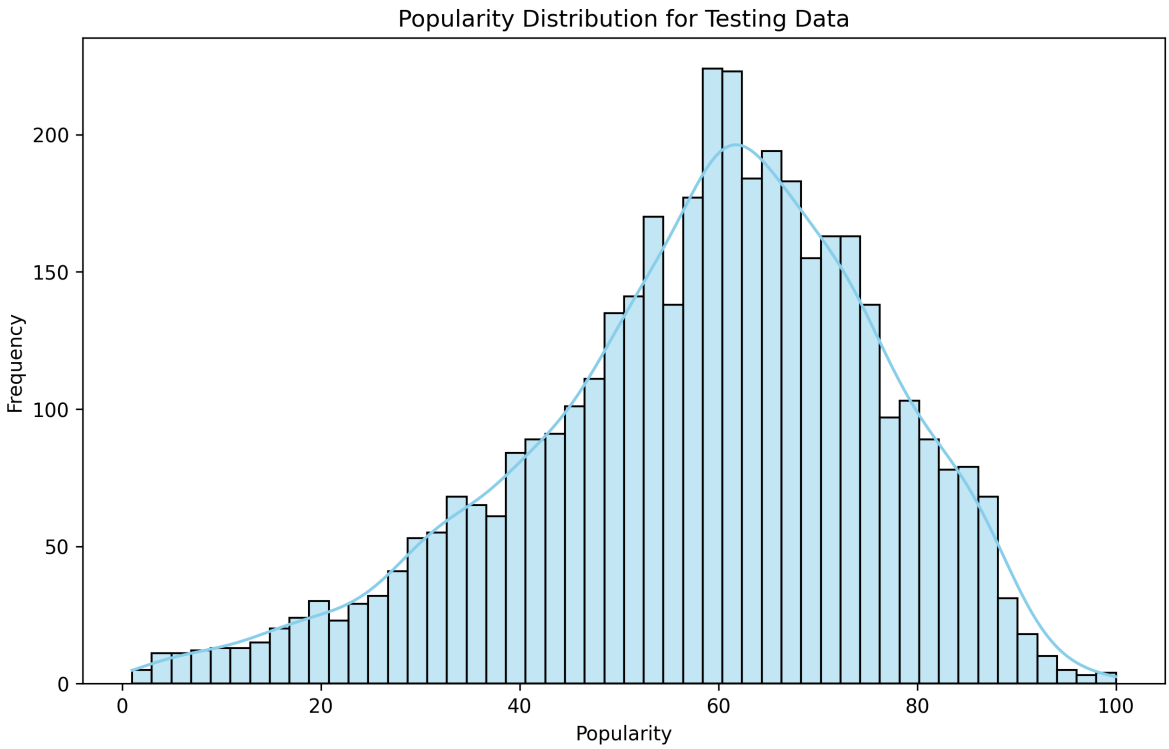
4. DATA PATITION AND PREPARATION FOR MODELLING. ==========

```
# --------------------------
### SKLEARN DATA PARTITION
# --------------------------

# Define feature columns and target variable
X = spotify_modelling.drop(columns=['popularity'])
y = spotify_modelling['popularity']

# Split the data into training and testing sets
# set.seed(50) in R is equivalent to random_state=50 in Python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=50, sti

# Converting to DataFrame is not strictly necessary for scikit-learn,
# but done to mimic R's explicit conversion.
train_dframe_features = X_train.copy()
test_dframe_features = X_test.copy()
train_dframe_target = y_train.to_frame(name='y_train')
test_dframe_target = y_test.to_frame(name='y_test')

# Combine features and target for distribution plotting, if needed
```

```
train_dframe = pd.concat([train_dframe_features, train_dframe_target], axis=1)
test_dframe = pd.concat([test_dframe_features, test_dframe_target], axis=1)


# testing popularity distribution for training data
plt.figure(figsize=(10, 6))
sns.histplot(y_train, bins=50, kde=True, color='skyblue')
plt.title('Popularity Distribution for Training Data')
plt.xlabel('Popularity')
plt.ylabel('Frequency')
plt.show()
plt.close()

# testing popularity distribution for testing data
plt.figure(figsize=(10, 6))
sns.histplot(y_test, bins=50, kde=True, color='skyblue')
plt.title('Popularity Distribution for Testing Data')
plt.xlabel('Popularity')
plt.ylabel('Frequency')
plt.show()
plt.close()

# -------------------------------------
## DATA PREPARATION FOR MODELING
# -------------------------------------

# Identify numerical and categorical columns
numerical_cols = X_train.select_dtypes(include=np.number).columns.tolist()
categorical_cols = X_train.select_dtypes(include='object').columns.tolist() # Ensure 'mode',

# Re-check categorical columns, as they might have been converted to int/float earlier
# based on the R script's `as.integer` for is_explicit and `as.factor` for others.
# In Python, we want true categorical columns for OneHotEncoder.
# Assuming 'mode', 'key', 'time_signature' are originally integers acting as categories
# and 'is_explicit' is int 0/1. Let's explicitly define them based on original intent.
categorical_cols_for_ohe = [col for col in ['is_explicit', 'mode', 'key', 'time_signature'] i
numerical_cols_for_scaling = [col for col in numerical_cols if col not in categorical_cols_fo


# Create a preprocessing pipeline
# `step_center` and `step_scale` are `StandardScaler` in Python.
# `step_dummy` is `OneHotEncoder`.
```

```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_cols_for_scaling),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_cols_for_ohe)
    ],
    remainder='passthrough' # Keep other columns not specified (if any)
)

# Apply preprocessing
# The `fit_transform` and `transform` methods handle fitting on training and transforming bot
train_processed_array = preprocessor.fit_transform(X_train)
test_processed_array = preprocessor.transform(X_test)

# Get feature names after one-hot encoding
feature_names = preprocessor.get_feature_names_out()

# Convert back to DataFrame
train_processed = pd.DataFrame(train_processed_array, columns=feature_names, index=X_train.in
test_processed = pd.DataFrame(test_processed_array, columns=feature_names, index=X_test.index


print("Processed Training Data (First few rows):")
print(train_processed.head())
print("Processed Testing Data (First few rows):")
print(test_processed.head())

# Combine processed features with the target variable (for modeling libraries that prefer a s
# Note: For scikit-learn models, X_train_processed and y_train are often passed separately.
# This step is more for mimicking R's `cbind` structure.
train_processed_combined = pd.concat([train_processed, y_train.reset_index(drop=True)], axis=
test_processed_combined = pd.concat([test_processed, y_test.reset_index(drop=True)], axis=1)
train_processed_combined.rename(columns={y_train.name: 'y_train'}, inplace=True)
test_processed_combined.rename(columns={y_test.name: 'y_test'}, inplace=True)
```


Popularity Distribution for Training Data
Frequency
Popularity
0
100
200
300
400
500
600
700
800
0
20
40
60
80
100


Popularity Distribution for Testing Data
Frequency
Popularity
0
50
100
150
200
0
20
40
60
80
100

```
Processed Training Data (First few rows):
       num__days_out  num__artist_count  num__market_count  num__daily_rank  \
13535      -0.224137          -0.630518          -0.253529        -2.196366
9686       -0.334784          -0.630518          -0.253529        -0.431651
15680      -0.342581          -0.630518          -0.253529         0.925823
5165       -0.341096          -0.630518          -0.043337        -0.160156
10436      -0.203715           0.361290          -0.253529        -1.517629

       num__daily_movement  num__weekly_movement  num__duration_min  \
13535            -0.153363             -0.303227           1.067532
9686              0.416652             -0.371464           0.079572
15680            -0.886238             -0.098517           1.309202
5165              0.823805             -0.644411          -0.046354
10436             0.498082             -0.030280           0.020173

       num__danceability  num__energy  num__loudness  ...  cat__key_7  \
13535           0.089659     0.213958      -0.351809  ...         0.0
9686            0.111187     1.221695       0.693972  ...         1.0
15680          -2.816718    -1.165051      -0.848508  ...         0.0
5165            0.922102     0.703093       1.090321  ...         0.0
10436           1.316795    -1.271128      -1.823603  ...         0.0

       cat__key_8  cat__key_9  cat__key_10  cat__key_11  \
13535         0.0         0.0          0.0          1.0
9686          0.0         0.0          0.0          0.0
15680         0.0         0.0          0.0          0.0
5165          0.0         0.0          0.0          0.0
10436         1.0         0.0          0.0          0.0

       cat__time_signature_0  cat__time_signature_1  cat__time_signature_3  \
13535                    0.0                    0.0                    0.0
9686                     0.0                    0.0                    0.0
15680                    0.0                    0.0                    0.0
5165                     0.0                    0.0                    0.0
10436                    0.0                    0.0                    0.0

       cat__time_signature_4  cat__time_signature_5
13535                    1.0                    0.0
9686                     1.0                    0.0
15680                    1.0                    0.0
5165                     1.0                    0.0
10436                    1.0                    0.0
```

```
[5 rows x 37 columns]
Processed Testing Data (First few rows):
       num__days_out  num__artist_count  num__market_count  num__daily_rank  \
9555       -0.303223          -0.630518          -0.253529         0.925823
400        -0.337011           1.353097           5.001254        -0.703145
1233       -0.323645          -0.630518          -0.253529        -0.635272
1948       -0.151362          -0.630518          -0.043337        -1.314008
15621      -0.342210          -0.630518          -0.253529         0.790075

       num__daily_movement  num__weekly_movement  num__duration_min  \
9555             -0.316224             -0.234991          -0.726558
400              -0.316224             -0.507937          -1.346719
1233             -0.071932             -0.303227          -0.046201
1948             -0.316224             -0.303227           1.919808
15621            -0.967669              0.037956          -0.152627

       num__danceability  num__energy  num__loudness  ...  cat__key_7  \
9555            2.034419    -0.605197      -0.535468  ...         0.0
400            -0.534674     1.374918       0.264489  ...         0.0
1233            0.771401    -0.328216      -1.536439  ...         0.0
1948           -0.541851    -1.365419      -1.448081  ...         0.0
15621          -1.123126     1.422064       0.778544  ...         0.0

       cat__key_8  cat__key_9  cat__key_10  cat__key_11  \
9555          1.0         0.0          0.0          0.0
400           0.0         0.0          0.0          1.0
1233          1.0         0.0          0.0          0.0
1948          0.0         0.0          1.0          0.0
15621         0.0         0.0          0.0          0.0

       cat__time_signature_0  cat__time_signature_1  cat__time_signature_3  \
9555                     0.0                    0.0                    0.0
400                      0.0                    0.0                    0.0
1233                     0.0                    0.0                    0.0
1948                     0.0                    0.0                    0.0
15621                    0.0                    0.0                    0.0

       cat__time_signature_4  cat__time_signature_5
9555                     1.0                    0.0
400                      1.0                    0.0
1233                     1.0                    0.0
1948                     1.0                    0.0
15621                    1.0                    0.0
```

```
[5 rows x 37 columns]
```

========== 5. MODEL TRAINING: R_F, XGB and GBM. ==========

```
# ------------------------
### RANDOM FOREST MODEL
# ------------------------

# Hyperparameter tuning using GridSearchCV/RandomizedSearchCV (caret::train equivalent)
# RandomForestRegressor from sklearn doesn't directly expose `mtry` (n_features_to_consider)
# but `max_features` is the equivalent. `min.node.size` is `min_samples_leaf`.
# `splitrule` is implicitly handled by the algorithm (usually 'mse' or 'squared_error').
# `num.trees` is `n_estimators`.

from sklearn.model_selection import GridSearchCV

# Define the model
rf_model = RandomForestRegressor(random_state=50, n_jobs=-1) # n_jobs=-1 uses all available c

# Define the hyperparameter grid
# A more robust way to define max_features is as a fraction of total features
num_features = train_processed.shape[1]
# Example: mtry=5 becomes max_features=5
# If R's mtry is absolute number of features, then we can use int values for max_features
param_grid_rf = {
    'n_estimators': [200],
    'max_features': [5, 7, 9, 11],
    'min_samples_leaf': [3, 5, 7]
}


# GridSearchCV for hyperparameter tuning (equivalent to caret::train with 'cv')
grid_search_rf = GridSearchCV(estimator=rf_model, param_grid=param_grid_rf,
                              cv=5, n_jobs=-1, scoring='neg_mean_squared_error', verbose=0)
grid_search_rf.fit(train_processed, y_train)

rf_best_model = grid_search_rf.best_estimator_

# Model results
print("Random Forest - Best Parameters:", grid_search_rf.best_params_)
```

```
print("Random Forest - Best Cross-Validation RMSE:", np.sqrt(-grid_search_rf.best_score_))

### Save RF model
import joblib
joblib.dump(rf_best_model, "rf_model.pkl")

# Predictions on test set
rf_pred = rf_best_model.predict(test_processed)

# Evaluate model performance
MAE_rf = mean_absolute_error(y_test, rf_pred)
RMSE_rf = np.sqrt(mean_squared_error(y_test, rf_pred))
R_squared_rf = r2_score(y_test, rf_pred)

print("Random Forest - Mean Absolute Error (MAE):", MAE_rf)
print("Random Forest - Root Mean Squared Error (RMSE):", RMSE_rf)
print("Random Forest - R-squared (R²):", R_squared_rf)

# Feature Importance
importance_df_rf = pd.DataFrame({
    'Feature': train_processed.columns,
    'Importance': rf_best_model.feature_importances_
})
sorted_importance_df_rf = importance_df_rf.sort_values(by='Importance', ascending=False).rese
print("Random Forest - Feature Importance:")
print(sorted_importance_df_rf[1:10])

# -------------------------
### XGBOOST MODEL
# -------------------------

# Prepare data for XGBoost (already done by preprocessor and X_train/X_test)
# XGBoost models in scikit-learn handle DataFrames directly.

# Define the model
xgb_model = XGBRegressor(objective='reg:squarederror', random_state=50, n_jobs=-1)

# Define the hyperparameter grid
# nrounds -> n_estimators
# max_depth, eta (learning_rate), gamma, colsample_bytree, min_child_weight, subsample are di
param_grid_xgb = {
    'n_estimators': [100, 200],
```

```
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.05, 0.1], # eta
    'gamma': [0],
    'colsample_bytree': [0.7, 0.9],
    'min_child_weight': [1],
    'subsample': [0.8]
}

# GridSearchCV for hyperparameter tuning
grid_search_xgb = GridSearchCV(estimator=xgb_model, param_grid=param_grid_xgb,
                               cv=5, n_jobs=-1, scoring='neg_mean_squared_error', verbose=0)
grid_search_xgb.fit(train_processed, y_train)

xgb_best_model = grid_search_xgb.best_estimator_

# Model results
print("XGBoost - Best Parameters:", grid_search_xgb.best_params_)
print("XGBoost - Best Cross-Validation RMSE:", np.sqrt(-grid_search_xgb.best_score_))

### Save XGBoost model
joblib.dump(xgb_best_model, "xgb_model.pkl")

# Predictions on test set
xgb_pred = xgb_best_model.predict(test_processed)

# Evaluate model performance
MAE_xgb = mean_absolute_error(y_test, xgb_pred)
RMSE_xgb = np.sqrt(mean_squared_error(y_test, xgb_pred))
R_squared_xgb = r2_score(y_test, xgb_pred)

print("XGBoost - Mean Absolute Error (MAE):", MAE_xgb)
print("XGBoost - Root Mean Squared Error (RMSE):", RMSE_xgb)
print("XGBoost - R-squared (R²):", R_squared_xgb)

# Feature Importance
importance_df_xgb = pd.DataFrame({
    'Feature': train_processed.columns,
    'Importance': xgb_best_model.feature_importances_
})
sorted_importance_df_xgb = importance_df_xgb.sort_values(by='Importance', ascending=False).re
print("XGBoost - Feature Importance:")
print(sorted_importance_df_xgb[1:10])
```

```
# -------------------------
### GBM MODEL
# -------------------------

# Define the model
gbm_model = GradientBoostingRegressor(random_state=50)

# Define the hyperparameter grid
# n.trees -> n_estimators
# interaction.depth -> max_depth
# shrinkage -> learning_rate
# n.minobsinnode -> min_samples_leaf
param_grid_gbm = {
    'n_estimators': [100, 200],
    'max_depth': [3, 5],
    'learning_rate': [0.01, 0.025],
    'min_samples_leaf': [10, 20]
}

# GridSearchCV for hyperparameter tuning
grid_search_gbm = GridSearchCV(estimator=gbm_model, param_grid=param_grid_gbm,
                               cv=5, n_jobs=-1, scoring='neg_mean_squared_error', verbose=0)
grid_search_gbm.fit(train_processed, y_train)

gbm_best_model = grid_search_gbm.best_estimator_

# Model results
print("GBM - Best Parameters:", grid_search_gbm.best_params_)
print("GBM - Best Cross-Validation RMSE:", np.sqrt(-grid_search_gbm.best_score_))

### Save GBM model
joblib.dump(gbm_best_model, "gbm_model.pkl")

# Predictions on test set
gbm_pred = gbm_best_model.predict(test_processed)

# Evaluate model performance
MAE_gbm = mean_absolute_error(y_test, gbm_pred)
RMSE_gbm = np.sqrt(mean_squared_error(y_test, gbm_pred))
R_squared_gbm = r2_score(y_test, gbm_pred)

print("Gradient Boosting Machine - Mean Absolute Error (MAE):", MAE_gbm)
```

```
print("Gradient Boosting Machine - Root Mean Squared Error (RMSE):", RMSE_gbm)
print("Gradient Boosting Machine - R-squared (R²):", R_squared_gbm)

# Feature Importance
importance_df_gbm = pd.DataFrame({
    'Feature': train_processed.columns,
    'Importance': gbm_best_model.feature_importances_
})
sorted_importance_df_gbm = importance_df_gbm.sort_values(by='Importance', ascending=False).re
print("GBM - Feature Importance:")
print(sorted_importance_df_gbm[1:10])
```

```
Random Forest - Best Parameters: {'max_features': 11, 'min_samples_leaf': 3, 'n_estimators':
Random Forest - Best Cross-Validation RMSE: 11.279759359525071
Random Forest - Mean Absolute Error (MAE): 8.698847749798059
Random Forest - Root Mean Squared Error (RMSE): 11.512042143690442
Random Forest - R-squared (R²): 0.591993818932163
Random Forest - Feature Importance:
                Feature  Importance
1     num__market_count    0.191807
2   num__daily_movement    0.101856
3  num__weekly_movement    0.090112
4         num__loudness    0.042242
5       num__daily_rank    0.034868
6     num__duration_min    0.030150
7     num__acousticness    0.028182
8          num__valence    0.027582
9      num__speechiness    0.027221
XGBoost - Best Parameters: {'colsample_bytree': 0.7, 'gamma': 0, 'learning_rate': 0.05, 'max_
XGBoost - Best Cross-Validation RMSE: 11.192787600177862
XGBoost - Mean Absolute Error (MAE): 8.606953620910645
XGBoost - Root Mean Squared Error (RMSE): 11.424507808308828
XGBoost - R-squared (R²): 0.5981749296188354
XGBoost - Feature Importance:
                 Feature  Importance
1          num__days_out    0.136925
2   num__weekly_movement    0.065269
3    num__daily_movement    0.055341
4     cat__is_explicit_0    0.026648
5        num__daily_rank    0.026213
6          num__loudness    0.024811
7     cat__is_explicit_1    0.024414
```

```
8  cat__time_signature_4    0.018940
9  num__instrumentalness    0.018915
GBM - Best Parameters: {'learning_rate': 0.025, 'max_depth': 5, 'min_samples_leaf': 20, 'n_es
GBM - Best Cross-Validation RMSE: 11.307317653696343
Gradient Boosting Machine - Mean Absolute Error (MAE): 8.765571014055027
Gradient Boosting Machine - Root Mean Squared Error (RMSE): 11.586005855728859
Gradient Boosting Machine - R-squared (R²): 0.5867341794180254
GBM - Feature Importance:
                 Feature  Importance
1      num__market_count    0.267058
2   num__weekly_movement    0.080457
3    num__daily_movement    0.077172
4          num__loudness    0.029760
5        num__daily_rank    0.021412
6  num__instrumentalness    0.008340
7      num__acousticness    0.008053
8      num__duration_min    0.006433
9            num__energy    0.004726
```

==========

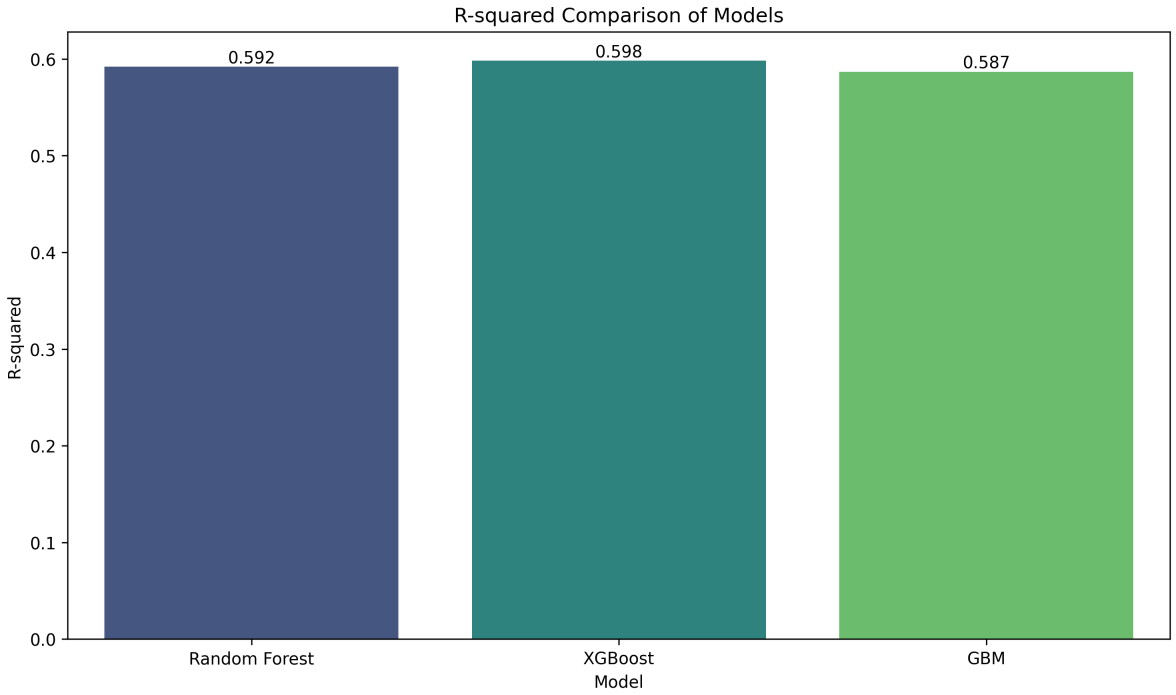
6. MODEL COMPARISON EVALUATION AND PLOTTING. ==========

```
# ----------------------
## MODEL COMPARISON:
# -------------------------

# Create a data frame to compare model performance
model_comparison = pd.DataFrame({
    'Model': ["Random Forest", "XGBoost", "GBM"],
    'MAE': [MAE_rf, MAE_xgb, MAE_gbm],
    'RMSE': [RMSE_rf, RMSE_xgb, RMSE_gbm],
    'R_squared': [R_squared_rf, R_squared_xgb, R_squared_gbm]
})

print("Model Performance Comparison:")
print(model_comparison)

# Create a bar plot for R-squared comparison
plt.figure(figsize=(10, 6))
sns.barplot(x='Model', y='R_squared', data=model_comparison, palette='viridis')
for index, row in model_comparison.iterrows():
    plt.text(index, row['R_squared'], round(row['R_squared'], 3), color='black', ha="center"
```

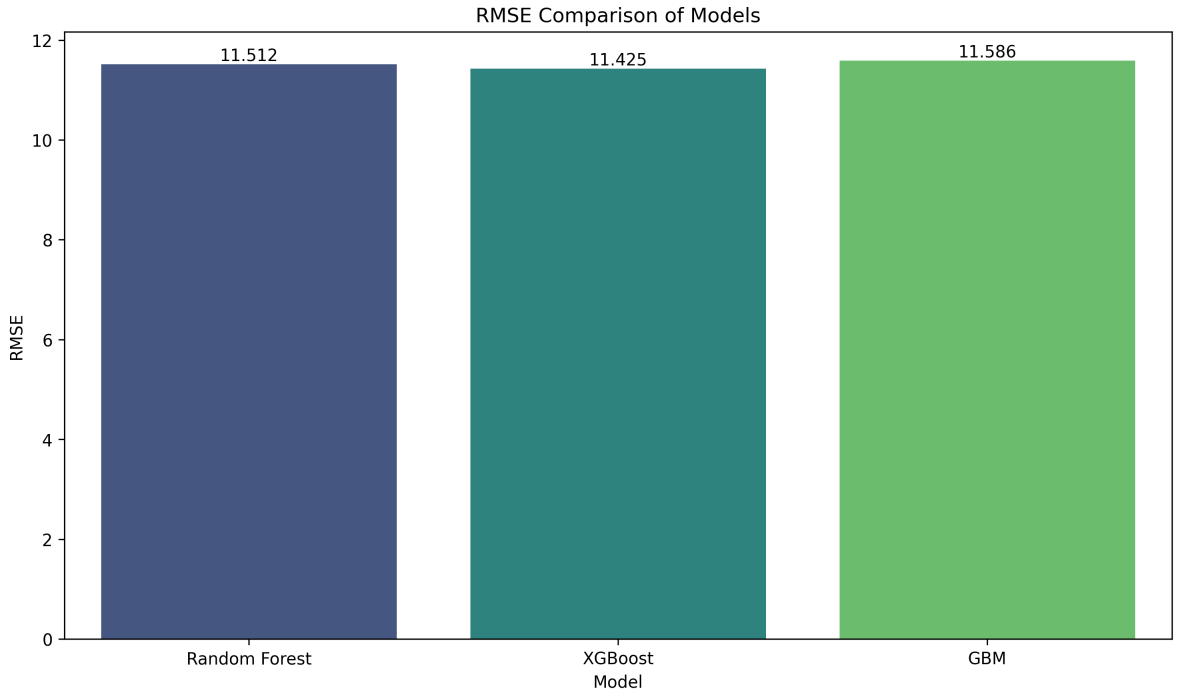
```
plt.title("R-squared Comparison of Models")
plt.ylabel("R-squared")
plt.tight_layout()
plt.show()
plt.close()

# Create a bar plot for RMSE comparison
plt.figure(figsize=(10, 6))
sns.barplot(x='Model', y='RMSE', data=model_comparison, palette='viridis')
for index, row in model_comparison.iterrows():
    plt.text(index, row['RMSE'], round(row['RMSE'], 3), color='black', ha="center", va='botto
plt.title("RMSE Comparison of Models")
plt.ylabel("RMSE")
plt.tight_layout()
plt.show()
plt.close()

# Create a bar plot for MAE comparison
plt.figure(figsize=(10, 6))
sns.barplot(x='Model', y='MAE', data=model_comparison, palette='viridis')
for index, row in model_comparison.iterrows():
    plt.text(index, row['MAE'], round(row['MAE'], 3), color='black', ha="center", va='bottom
plt.title("MAE Comparison of Models")
plt.ylabel("MAE")
plt.tight_layout()
plt.show()
plt.close()

# -----------------------------------
## ACTUAL VS PREDICTED PLOTS:
# -----------------------------------

# Random Forest
plt.figure(figsize=(8, 8))
plot_data_rf = pd.DataFrame({'Actual': y_test, 'Predicted': rf_pred})
sns.scatterplot(x='Actual', y='Predicted', data=plot_data_rf, color='blue', alpha=0.6)
sns.regplot(x='Actual', y='Predicted', data=plot_data_rf, scatter=False, color='red', line_kw
plt.title("Actual vs Predicted Popularity (Random Forest)")
plt.xlabel("Actual Popularity")
plt.ylabel("Predicted Popularity")
plt.grid(True, linestyle='--', alpha=0.5)
plt.show()
```

```
plt.close()

# XGBoost
plt.figure(figsize=(8, 8))
plot_data_xgb = pd.DataFrame({'Actual': y_test, 'Predicted': xgb_pred})
sns.scatterplot(x='Actual', y='Predicted', data=plot_data_xgb, color='green', alpha=0.6)
sns.regplot(x='Actual', y='Predicted', data=plot_data_xgb, scatter=False, color='red', line_
plt.title("Actual vs Predicted Popularity (XGBoost)")
plt.xlabel("Actual Popularity")
plt.ylabel("Predicted Popularity")
plt.grid(True, linestyle='--', alpha=0.5)
plt.show()
plt.close()

# GBM
plt.figure(figsize=(8, 8))
plot_data_gbm = pd.DataFrame({'Actual': y_test, 'Predicted': gbm_pred})
sns.scatterplot(x='Actual', y='Predicted', data=plot_data_gbm, color='purple', alpha=0.6)
sns.regplot(x='Actual', y='Predicted', data=plot_data_gbm, scatter=False, color='red', line_
plt.title("Actual vs Predicted Popularity (GBM)")
plt.xlabel("Actual Popularity")
plt.ylabel("Predicted Popularity")
plt.grid(True, linestyle='--', alpha=0.5)
plt.show()
plt.close()
```

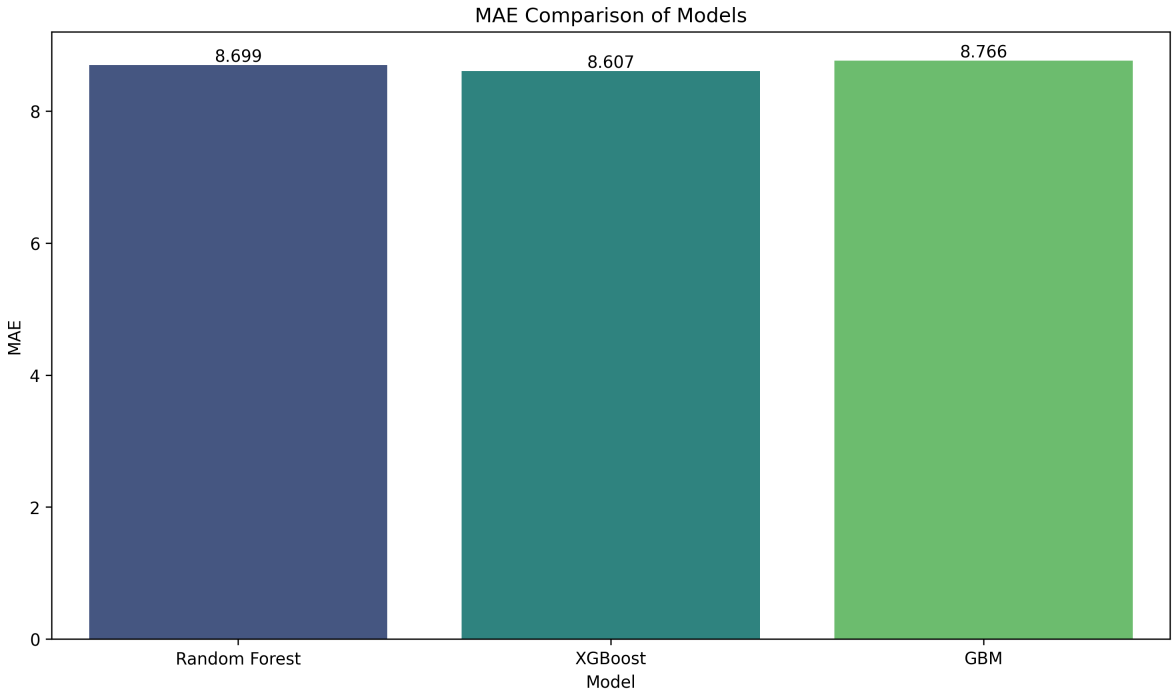
```
Model Performance Comparison:
           Model       MAE       RMSE  R_squared
0  Random Forest  8.698848  11.512042   0.591994
1        XGBoost  8.606954  11.424508   0.598175
2            GBM  8.765571  11.586006   0.586734

C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\1266757217.py:18: FutureWarni

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assig

  sns.barplot(x='Model', y='R_squared', data=model_comparison, palette='viridis')
```

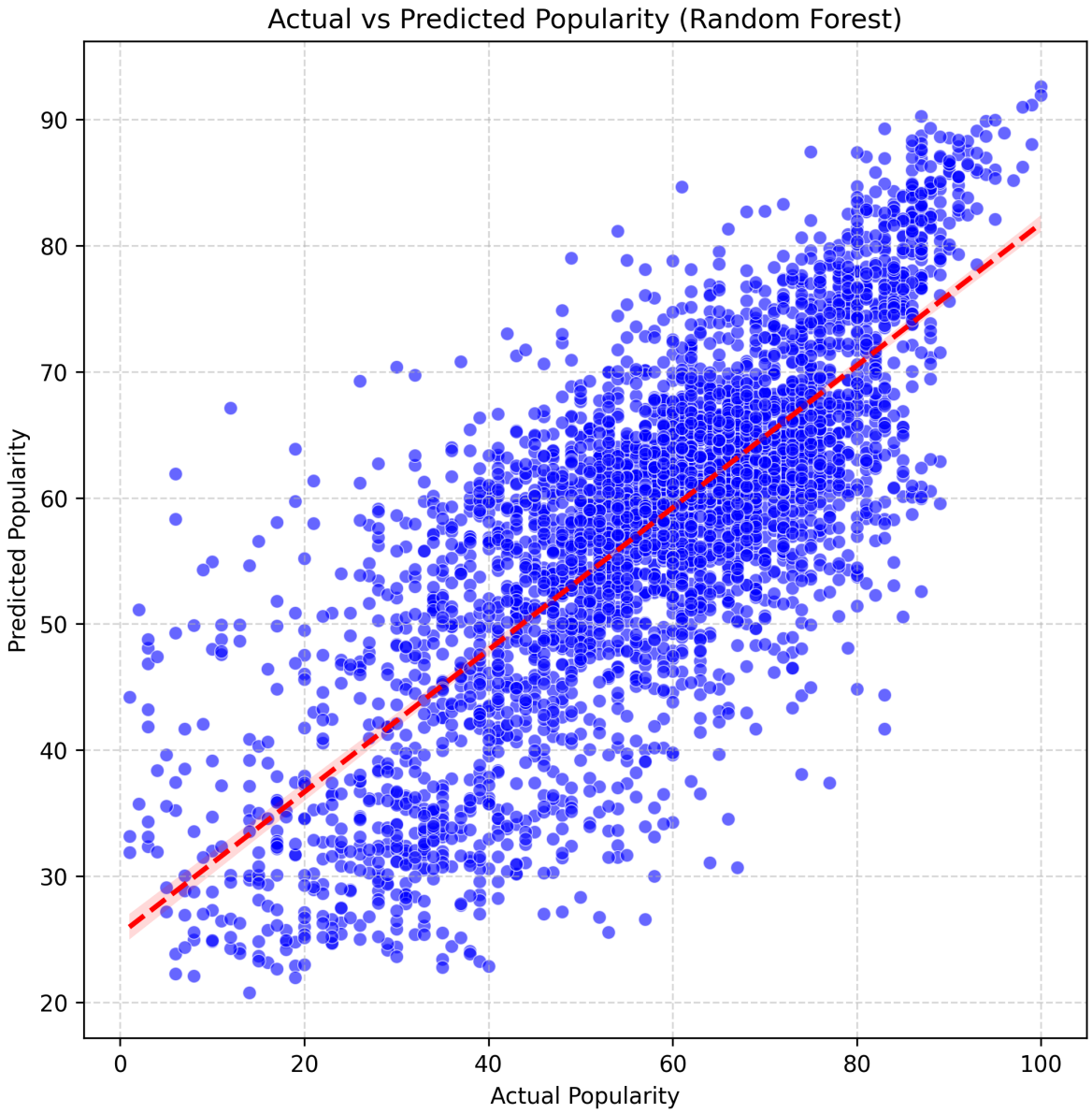


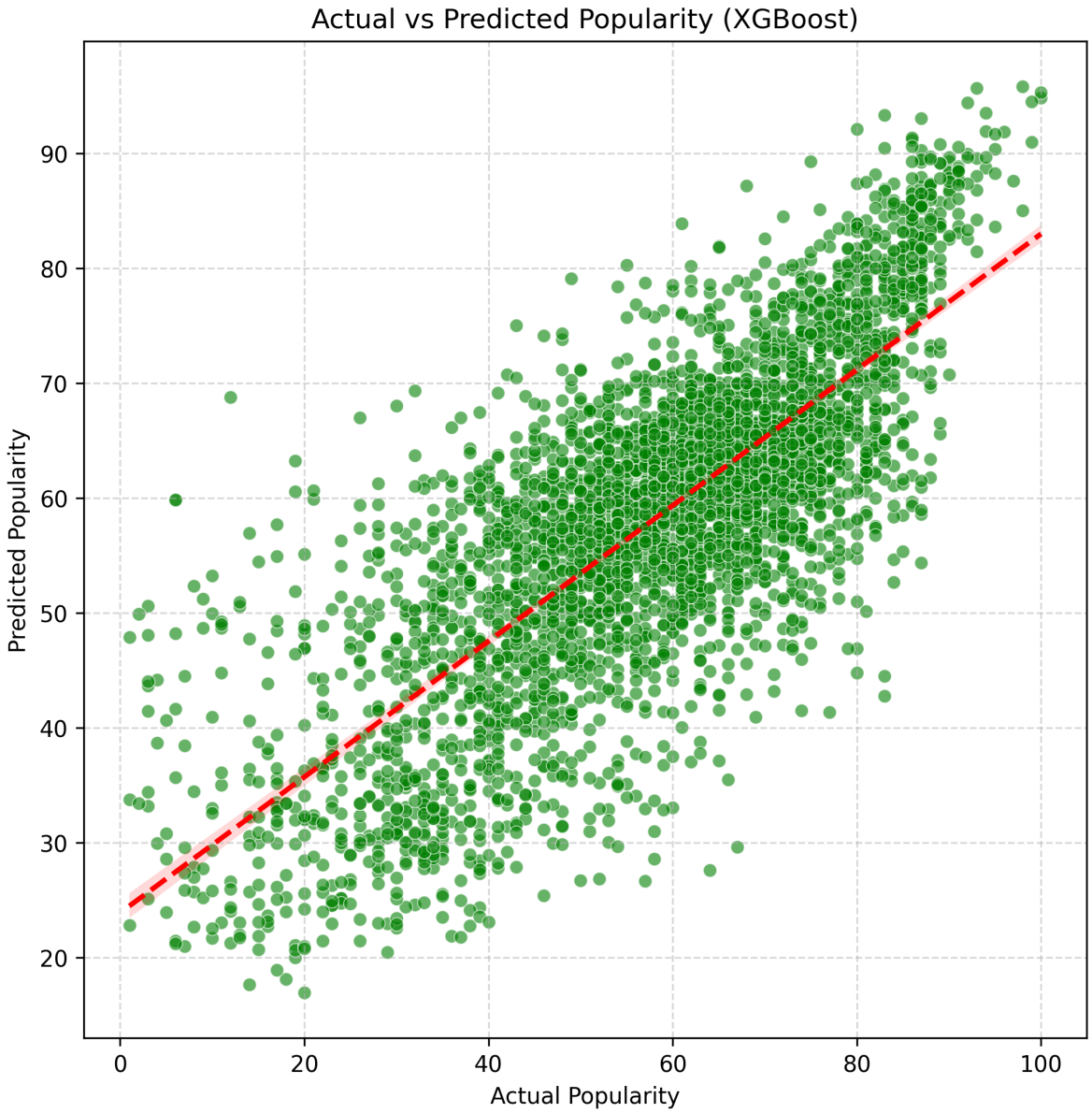

```
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\1266757217.py:29: FutureWarni
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assig
  sns.barplot(x='Model', y='RMSE', data=model_comparison, palette='viridis')
```

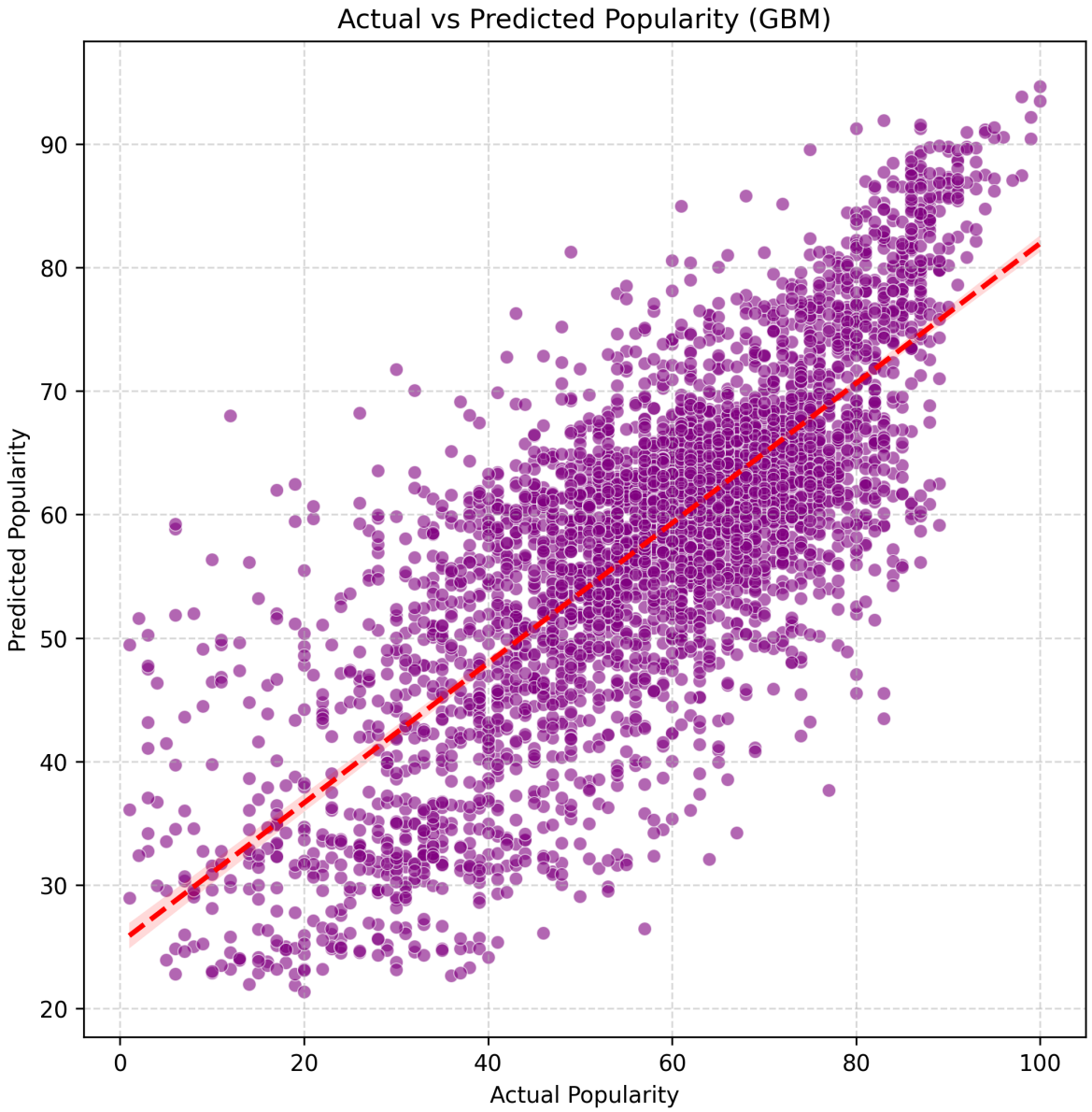



```
C:\Users\HP ELITEBOOK 820 G4\AppData\Local\Temp\ipykernel_17096\1266757217.py:40: FutureWarni
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assig
  sns.barplot(x='Model', y='MAE', data=model_comparison, palette='viridis')
```


MAE Comparison of Models
8.699
8.607
8.766
MAE
0
2
4
6
8
Random Forest
XGBoost
GBM
Model


Actual vs Predicted Popularity (Random Forest)
Predicted Popularity
Actual Popularity
0
20
40
60
80
100
20
30
40
50
60
70
80
90


Actual vs Predicted Popularity (XGBoost)
Predicted Popularity
Actual Popularity
0
20
40
60
80
100
20
30
40
50
60
70
80
90



THE END; j-k. :ALWAYS