In [5]:

```
# Import necessary libraries
import pandas as pd
from datetime import datetime
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import accuracy_score


# Function to convert date string to datetime object
def convert_date(date_str):
    return datetime.strptime(date_str, '%d %b %Y, %H:%M')

# Load the data
df = pd.read_csv('lastfm_data2.csv')

# Remove leading and trailing whitespace from column names
df.columns = df.columns.str.strip()

# Drop rows with missing date values
df = df.dropna(subset=['date'])

# Apply the function to the 'date' column
df['date'] = df['date'].apply(convert_date)

# Set the date as the DataFrame's index
df.set_index('date', inplace=True)

# Reset the index of the DataFrame
df_reset = df.reset_index()

# Extract the hour and day of week from the date
df_reset['hour'] = df_reset['date'].dt.hour
df_reset['day_of_week'] = df_reset['date'].dt.dayofweek

# Create a new DataFrame with the 'hour', 'day_of_week', and 'counts' features
df_reset['counts'] = 1  # For each row, the count is 1
data = df_reset.groupby(['day_of_week', 'hour']).size().reset_index(name='counts')

# Split the data into features (X) and target variable (y)
X = data[['day_of_week', 'hour']]
y = data['counts']

# Split the data into a training set and a test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42

# Standardize the features to have zero mean and unit variance
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create and train the Random Forest Regressor
regressor = RandomForestRegressor(n_estimators=100, random_state=42)
regressor.fit(X_train_scaled, y_train)

# Predict the number of songs listened to in the test set
y_pred = regressor.predict(X_test_scaled)

# Calculate the mean absolute error of the predictions
mae = mean_absolute_error(y_test, y_pred)
```

```
mae
```

Out[5]:
```
61.21303030303031
```

In [6]:
```
# Function to predict the number of songs listened to at a specific hour and day of the
def predict_scrobbles(day_of_week, hour):
    # Prepare the feature vector
    X_pred = pd.DataFrame({'day_of_week': [day_of_week], 'hour': [hour]})

    # Standardize the features
    X_pred_scaled = scaler.transform(X_pred)

    # Make the prediction
    y_pred = regressor.predict(X_pred_scaled)

    return y_pred[0]

# Test the function with an example: predict the number of songs listened to on Monday a
print(predict_scrobbles(0, 10))
print(predict_scrobbles(1, 15))
print(predict_scrobbles(5, 1))
```

```
12.16
507.45
1223.17
```

In [7]:
```
# For each track, get the most common tag and consider it as the genre
df_reset['tags'] = df_reset['tags'].apply(lambda x: str(x).split(',')[0] if pd.notnull(x

# Create a new DataFrame for the classification model
data_cls = df_reset.groupby(['day_of_week', 'hour'])['tags'].agg(lambda x: x.value_count

# Split the data into features (X) and target variable (y)
X_cls = data_cls[['day_of_week', 'hour']]
y_cls = data_cls['tags']

# Split the data into a training set and a test set
X_train_cls, X_test_cls, y_train_cls, y_test_cls = train_test_split(X_cls, y_cls, test_s

# Standardize the features to have zero mean and unit variance
scaler_cls = StandardScaler()
X_train_cls_scaled = scaler_cls.fit_transform(X_train_cls)
X_test_cls_scaled = scaler_cls.transform(X_test_cls)

# Create and train the Random Forest Classifier
classifier = RandomForestClassifier(n_estimators=100, random_state=42)
classifier.fit(X_train_cls_scaled, y_train_cls)

# Predict the genre of music listened to in the test set
y_pred_cls = classifier.predict(X_test_cls_scaled)

# Calculate the accuracy of the predictions
accuracy_cls = accuracy_score(y_test_cls, y_pred_cls)

accuracy_cls
```

Out[7]:
```
0.5454545454545454
```

In [8]:
```
# Function to predict the genre of music listened to at a specific hour and day of the w
def predict_genre(day_of_week, hour):
    # Prepare the feature vector
    X_pred = pd.DataFrame({'day_of_week': [day_of_week], 'hour': [hour]})
```

```
    # Standardize the features
    X_pred_scaled = scaler_cls.transform(X_pred)

    # Make the prediction
    y_pred = classifier.predict(X_pred_scaled)

    return y_pred[0]

# Test the function with an example: predict the genre of music listened to on Monday at
print(predict_genre(0, 10))
print(predict_genre(1, 15))
print(predict_genre(5, 1))
```

```
alternative
alternative
k-pop
```

In [ ]: