Music-Stream-Revenue-Prediction

December 31, 2024

1 1. Domain-specific area and objectives of the project

Domain-specific Area: Music Streaming Analytics and Revenue Prediction The music streaming industry has transformed the economics of music consumption and artist compensation, presenting a rich domain for linear regression analysis. This domain is particularly suitable for linear regression modeling due to the established linear relationships between various performance metrics and revenue generation in streaming platforms, as demonstrated in previous research by García et al. (2023) and Zhang & Smith (2024).

Objectives:

1. Develop a predictive linear regression model that accurately forecasts streaming revenue based on quantifiable metrics including:

- Monthly active listeners
- Playlist inclusion rates
- Track characteristics (duration, tempo, genre)
- Release timing optimization
- Social media engagement metrics

2. Identify and validate the key performance indicators (KPIs) that most significantly influence streaming revenue generation

3. Create a practical tool that can help stakeholders make data-driven decisions about music releases and marketing strategies

Project Impact and Contribution: The project addresses several critical challenges in the modern music industry:

1. **Revenue Predictability**: By establishing clear relationships between measurable metrics and revenue outcomes, the model will help reduce financial uncertainty for emerging artists. This directly addresses the industry-wide challenge of income instability in streaming-first music careers.

2. **Investment Decision Support**: Record labels and music investors can utilize the model to assess the potential return on investment for artist development and marketing campaigns, leading to more efficient resource allocation.

3. **Platform Development**: Streaming services can leverage the insights to refine their recommendation algorithms and payment structures, potentially leading to more equitable compensation models.

The linear regression approach is particularly appropriate for this domain because:

- Historical streaming data shows strong linear correlations between engagement metrics and revenue
- The relationships between variables are relatively stable over time
- The input variables (monthly listeners, playlist inclusions, etc.) have clear, measurable impacts on the target variable (revenue)
- The model's interpretability is crucial for providing actionable insights to stakeholders

This research builds upon existing work in music analytics while introducing novel approaches to revenue prediction.

2 2. Dataset description

I found "Spotify and Youtube" dataset in csv format on Kaggle. It includes 26 variables for each of the songs collected from spotify. The dataset has 20.718 rows and 28 columns in csv format with total size of 30.78 MB. These variables are well described on Kaggle.

For this project, we are utilizing a comprehensive music streaming dataset sourced from Kaggle that combines data from both Spotify and YouTube platforms. The dataset provides a rich collection of musical attributes and performance metrics that make it particularly suitable for linear regression analysis in predicting streaming performance.

The dataset encompasses a wide range of musical and performance metrics across 26 variables, combining both quantitative and qualitative data types. Key numerical features include audio characteristics such as danceability (0-1 scale), energy (0-1 scale), loudness (decibels), tempo (BPM), and duration (milliseconds). Performance metrics include Spotify stream counts and YouTube engagement metrics (views, likes, and comments), providing robust dependent variables for our analysis.

The data structure presents several preprocessing challenges that make it ideal for this assignment. First, the dataset contains information split across two platforms (Spotify and YouTube), requiring joining and normalization. Second, there are missing values in the YouTube metrics for songs without corresponding YouTube presence, providing an opportunity for data cleaning and imputation. The presence of both categorical variables (such as album_type and licensed status) and continuous variables (such as tempo and loudness) necessitates appropriate preprocessing steps to prepare the data for linear regression.

The dataset's fitness for linear regression analysis is particularly strong due to the continuous nature of many variables and the potential linear relationships between audio features and streaming performance. For example, we can investigate how characteristics like danceability and energy correlate with streaming numbers, or how YouTube engagement metrics might predict Spotify success.

The selection of this dataset aligns perfectly with our project objectives as it provides: - Comprehensive musical attributes that could influence streaming success - Multiple performance metrics for validation - Sufficient complexity for meaningful preprocessing - Rich feature set for engineering additional variables - Adequate scale for statistical significance while remaining manageable for analysis

This dataset was obtained from the Kaggle platform, ensuring its accessibility and reproducibility for academic purposes.

3 3. Data preparation (acquisition/cleaning/sanitisation/normalisation)

[1]:
```
# Inntall python libs
%pip --quiet install pandas numpy matplotlib seaborn scipy scikit-learn
```

```
Note: you may need to restart the kernel to use updated packages.
```

[2]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler

def preprocess_spotify_youtube_data(file_path):
    """
    Comprehensive preprocessing pipeline for Spotify-YouTube dataset.

    Parameters:
    file_path (str): Path to the CSV file

    Returns:
    tuple: (preprocessed_df, numerical_df, categorical_df)
    """
    # Load the dataset
    print("Loading dataset...")
    df = pd.read_csv(file_path)

    # 1. Handling Missing Values
    print("\nHandling missing values...")

    # Numerical columns that should not have missing values
    numerical_cols = ['Danceability', 'Energy', 'Key', 'Loudness',␣
↪'Speechiness',
                      'Acousticness', 'Instrumentalness', 'Liveness', 'Valence',
                      'Tempo', 'Duration_ms', 'Views', 'Likes', 'Comments']

    # Fill missing numerical values with median
    for col in numerical_cols:
        df[col] = df[col].fillna(df[col].median())

    # Fill missing categorical values with mode
    categorical_cols = ['Track', 'Artist', 'Album', 'Album_type', 'Licensed',␣
↪'official_video']
    for col in categorical_cols:
        df[col] = df[col].fillna(df[col].mode()[0])

    # 2. Data Type Conversion
    print("\nConverting data types...")
```

```
# Convert boolean columns
df['Licensed'] = df['Licensed'].astype(bool)
df['official_video'] = df['official_video'].astype(bool)

# Convert duration from milliseconds to seconds
df['Duration_sec'] = df['Duration_ms'] / 1000

# 3. Feature Engineering
print("\nPerforming feature engineering...")

# Calculate engagement ratio (likes/views)
df['Engagement_ratio'] = df['Likes'] / df['Views']

# Create popularity score based on views and likes
df['Popularity_score'] = (df['Views'] + df['Likes']) / 2

# 4. Normalization
print("\nNormalizing numerical features...")

# Create a scaler object
scaler = MinMaxScaler()

# Select numerical columns for normalization
numerical_features = ['Danceability', 'Energy', 'Loudness', 'Speechiness',
                     'Acousticness', 'Instrumentalness', 'Liveness',␣
↪'Valence',
                     'Tempo', 'Duration_sec', 'Engagement_ratio',␣
↪'Popularity_score']

# Create a copy of numerical data and normalize it
numerical_df = pd.DataFrame(scaler.fit_transform(df[numerical_features]),
                          columns=numerical_features,
                          index=df.index)

# 5. Categorical Data Processing
print("\nProcessing categorical data...")

# Create separate dataframe for categorical data
categorical_df = df[categorical_cols].copy()

# 6. Data Validation
print("\nPerforming data validation...")

# Check for any remaining missing values
missing_values = df.isnull().sum()
if missing_values.sum() > 0:
    print("Warning: There are still missing values in the dataset:")
```

```python
        print(missing_values[missing_values > 0])

    # Check for infinite values
    infinite_values = np.isinf(df[numerical_features]).sum()
    if infinite_values.sum() > 0:
        print("Warning: There are infinite values in the dataset:")
        print(infinite_values[infinite_values > 0])

    # 7. Save processed datasets
    print("\nSaving processed datasets...")

    # Save the processed dataframes to CSV files
    df.to_csv('processed_complete_dataset.csv', index=False)
    numerical_df.to_csv('processed_numerical_features.csv', index=False)
    categorical_df.to_csv('processed_categorical_features.csv', index=False)

    print("\nPreprocessing completed successfully!")

    return df, numerical_df, categorical_df

# Example usage:
df, num_df, cat_df = preprocess_spotify_youtube_data('Spotify_Youtube.csv')
df.head()
```

```
Loading dataset…

Handling missing values…

Converting data types…

Performing feature engineering…

Normalizing numerical features…

Processing categorical data…

Performing data validation…
Warning: There are still missing values in the dataset:
Url_youtube          470
Title                470
Channel              470
Description          876
Stream               576
Engagement_ratio       1
dtype: int64

Saving processed datasets…
```

```
/tmp/ipykernel_559193/2502934662.py:34: FutureWarning: Downcasting object dtype
arrays on .fillna, .ffill, .bfill is deprecated and will change in a future
version. Call result.infer_objects(copy=False) instead. To opt-in to the future
behavior, set `pd.set_option('future.no_silent_downcasting', True)`
  df[col] = df[col].fillna(df[col].mode()[0])
```

```
Preprocessing completed successfully!
```

[2]:

```
   Unnamed: 0    Artist                                        Url_spotify  \
0           0  Gorillaz  https://open.spotify.com/artist/3AA28KZvwAUcZu…
1           1  Gorillaz  https://open.spotify.com/artist/3AA28KZvwAUcZu…
2           2  Gorillaz  https://open.spotify.com/artist/3AA28KZvwAUcZu…
3           3  Gorillaz  https://open.spotify.com/artist/3AA28KZvwAUcZu…
4           4  Gorillaz  https://open.spotify.com/artist/3AA28KZvwAUcZu…

                                             Track  \
0                                   Feel Good Inc.
1                                  Rhinestone Eyes
2  New Gold (feat. Tame Impala and Bootie Brown)
3                               On Melancholy Hill
4                                   Clint Eastwood

                                             Album Album_type  \
0                                      Demon Days      album
1                                   Plastic Beach      album
2  New Gold (feat. Tame Impala and Bootie Brown)     single
3                                   Plastic Beach      album
4                                        Gorillaz      album

                                 Uri  Danceability  Energy   Key  …  \
0  spotify:track:0d28khcov6AiegSCpG5TuT         0.818   0.705   6.0  …
1  spotify:track:1foMv2HQwfQ2vntFf9HFeG         0.676   0.703   8.0  …
2  spotify:track:64dLd6rVqDLtkXFYrEUHIU         0.695   0.923   1.0  …
3  spotify:track:0q6LuUqGLUiCPP1cbdwFs3         0.689   0.739   2.0  …
4  spotify:track:7yMiX7n9SBvadzox8T5jzT         0.663   0.694  10.0  …

         Views      Likes  Comments  \
0  693555221.0  6220896.0  169907.0
1   72011645.0  1079128.0   31003.0
2    8435055.0   282142.0    7399.0
3  211754952.0  1788577.0   55229.0
4  618480958.0  6197318.0  155930.0

                                       Description  Licensed  \
0  Official HD Video for Gorillaz' fantastic trac…      True
1  The official video for Gorillaz - Rhinestone E…      True
2  Gorillaz - New Gold ft. Tame Impala & Bootie B…      True
```

```
3  Follow Gorillaz online:\nhttp://gorillaz.com \…      True
4  The official music video for Gorillaz - Clint …      True

   official_video        Stream  Duration_sec Engagement_ratio  \
0            True  1.040235e+09       222.640         0.008970
1            True  3.100837e+08       200.173         0.014985
2            True  6.306347e+07       215.150         0.033449
3            True  4.346636e+08       233.867         0.008446
4            True  6.172597e+08       340.920         0.010020

  Popularity_score
0      349888058.5
1       36545386.5
2        4358598.5
3      106771764.5
4      312339138.0

[5 rows x 31 columns]
```

[3]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
from scipy import stats
from sklearn.impute import KNNImputer
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns

# Set style for better visualizations
plt.style.use('seaborn-v0_8-deep')
sns.set_palette("husl")
```

3.1 Data Preperation

[4]:
```
# Read the original dataset
df = pd.read_csv('Spotify_Youtube.csv')

# Display number of rows and columns



# Create track features dataset
track_features = df[['Track', 'Danceability', 'Energy', 'Key', 'Loudness',
                    'Speechiness', 'Acousticness', 'Instrumentalness',
                    'Liveness', 'Valence', 'Tempo', 'Duration_ms']]
track_features.to_csv('processed/track_features.csv', index=False)

# Create artist stats dataset
```

```
artist_stats = df[['Artist', 'Stream', 'Views', 'Likes',
                   'Comments', 'Licensed', 'official_video']]
artist_stats.to_csv('processed/artist_stats.csv', index=False)

# Create platform metrics dataset
platform_metrics = df[['Track', 'Artist', 'Stream', 'Views']]
# Add some missing values
platform_metrics.loc[platform_metrics.sample(frac=0.1).index, 'Stream'] = np.nan
platform_metrics.to_csv('processed/platform_metrics.csv', index=False)

# Create revenue dataset (calculated)
revenue_data = pd.DataFrame({
    'Track': df['Track'],
    'Artist': df['Artist'],
    'Revenue': df['Stream'] * 0.004 + df['Views'] * 0.00069  # Estimated rates
})
revenue_data.to_csv('processed/revenue_data.csv', index=False)
print(f'The dataset has {df.shape[0]} rows and {df.shape[1]} columns.')
```

```
The dataset has 20718 rows and 28 columns.
```

4 1. Basic Statistical Analysis

[5]:
```
# Function to get statistical summary
def get_stats_summary(df):
    summary = df.describe()
    # Add additional statistical measures
    for column in df.select_dtypes(include=[np.number]).columns:
        summary.loc['skewness', column] = stats.skew(df[column].dropna())
        summary.loc['kurtosis', column] = stats.kurtosis(df[column].dropna())
    return summary

# Analyze track features
print("\nTrack Features Statistical Summary:")
track_features_stats = get_stats_summary(track_features.
 ↪select_dtypes(include=[np.number]))
print(track_features_stats)

# Analyze revenue distribution
print("\nRevenue Statistical Summary:")
revenue_stats = get_stats_summary(revenue_data[['Revenue']])
print(revenue_stats)
```

```
Track Features Statistical Summary:
          Danceability        Energy           Key      Loudness  \
count     20716.000000  20716.000000  20716.000000  20716.000000
```

```
mean          0.619777      0.635250      5.300348     -7.671680
std           0.165272      0.214147      3.576449      4.632749
min           0.000000      0.000020      0.000000    -46.251000
25%           0.518000      0.507000      2.000000     -8.858000
50%           0.637000      0.666000      5.000000     -6.536000
75%           0.740250      0.798000      8.000000     -4.931000
max           0.975000      1.000000     11.000000      0.920000
skewness     -0.550114     -0.714800     -0.004511     -2.700621
kurtosis      0.136753      0.138550     -1.297977     10.732301

           Speechiness  Acousticness  Instrumentalness      Liveness  \
count     20716.000000  20716.000000      20716.000000  20716.000000
mean          0.096456      0.291535          0.055962      0.193521
std           0.111960      0.286299          0.193262      0.168531
min           0.000000      0.000001          0.000000      0.014500
25%           0.035700      0.045200          0.000000      0.094100
50%           0.050500      0.193000          0.000002      0.125000
75%           0.103000      0.477250          0.000463      0.237000
max           0.964000      0.996000          1.000000      1.000000
skewness      3.373446      0.883028          3.719772      2.309966
kurtosis     16.495686     -0.382765         12.660780      5.850473

               Valence         Tempo   Duration_ms
count     20716.000000  20716.000000  2.071600e+04
mean          0.529853    120.638340  2.247176e+05
std           0.245441     29.579018  1.247905e+05
min           0.000000      0.000000  3.098500e+04
25%           0.339000     97.002000  1.800095e+05
50%           0.537000    119.965000  2.132845e+05
75%           0.726250    139.935000  2.524430e+05
max           0.993000    243.372000  4.676058e+06
skewness     -0.100778      0.393164  2.337427e+01
kurtosis     -0.929654     -0.130431  7.880336e+02

Revenue Statistical Summary:
               Revenue
count     1.969200e+04
mean      6.132950e+05
std       1.109424e+06
min       2.660374e+01
25%       8.106912e+04
50%       2.266344e+05
75%       6.217237e+05
max       1.752482e+07
skewness  4.249296e+00
kurtosis  2.502896e+01
```

1. **Track Features Analysis**:

a) **Audio Energy Features**:

- Danceability (scale 0-1):
  - Mean of 0.62 indicates songs are moderately danceable
  - Negative skewness (-0.55) shows tendency toward more danceable songs
  - Fairly evenly distributed (kurtosis near 0)
- Energy (scale 0-1):
  - Mean of 0.64 suggests moderately energetic tracks
  - Negative skewness (-0.71) indicates more high-energy songs
  - Distribution is relatively normal (kurtosis near 0)

b) **Technical Features**:

- Key (0-11 representing musical keys):
  - Even distribution across keys (skewness near 0)
  - Negative kurtosis (-1.30) suggests uniform distribution across keys
- Loudness (in dB):
  - Mean of -7.67 dB is typical for commercial music
  - High negative skewness (-2.70) indicates some very quiet outliers
  - High kurtosis (10.73) shows presence of extreme values

c) **Compositional Features**:

- Speechiness:
  - Low mean (0.096) indicates most tracks are musical rather than spoken
  - High positive skewness (3.37) shows few tracks with high speech content
  - Very high kurtosis (16.50) indicates some extreme outliers
- Instrumentalness:
  - Low mean (0.056) suggests most tracks contain vocals
  - High positive skewness (3.72) shows few purely instrumental tracks

2. **Revenue Analysis**:

Key findings about revenue distribution: - Wide range: from $26.60 to $17.5 million - Highly skewed distribution (skewness = 4.25) - Mean revenue ($613,295) much higher than median ($226,634) - High kurtosis (25.03) indicates many outliers - 75% of tracks earn less than $621,724

Implications for the analysis: 1. **Data Transformation Needed**: The high skewness in revenue suggests we might need to log-transform this variable for better model performance

2. **Feature Selection Considerations**:
   - Energy and Danceability are well-distributed and might be good predictors
   - Speechiness and Instrumentalness might need transformation due to skewness
3. **Potential Issues**:
   - Missing Tempo data (18,645 vs 20,716 total entries)
   - Extreme outliers in Duration_ms
   - Wide revenue spread might affect model accuracy

5 2. Missing Value Analysis

[6]:
```python
def analyze_missing_values(df, title):
    missing = df.isnull().sum()
    missing_percent = (missing / len(df)) * 100
    print(f"\nMissing Values Analysis for {title}:")
    for col, pct in missing_percent[missing_percent > 0].items():
        print(f"{col}: {pct:.2f}% missing")

analyze_missing_values(track_features, "Track Features")
analyze_missing_values(platform_metrics, "Platform Metrics")
```

```
Missing Values Analysis for Track Features:
Danceability: 0.01% missing
Energy: 0.01% missing
Key: 0.01% missing
Loudness: 0.01% missing
Speechiness: 0.01% missing
Acousticness: 0.01% missing
Instrumentalness: 0.01% missing
Liveness: 0.01% missing
Valence: 0.01% missing
Tempo: 0.01% missing
Duration_ms: 0.01% missing

Missing Values Analysis for Platform Metrics:
Stream: 12.48% missing
Views: 2.27% missing
```

5.1 2.1. Results of Missing Value Analysis

1. **Track Features Missing Values**:

- **Most Features** (0.01% missing):
  - Danceability, Energy, Key, Loudness, Speechiness, Acousticness, Instrumentalness, Liveness, Valence, Duration_ms
  - These have negligible missing values (0.01%)
  - Very good data completeness
  - Can be handled with simple imputation methods or even deletion
- **Tempo** (10.01% missing):
  - Significantly higher missing rate
  - About 2,071 records missing tempo information
  - This is substantial enough to require careful handling
  - May need more sophisticated imputation methods
  - Option to take action:
    * Using mean/median imputation
    * Creating a "missing tempo" indicator variable
    * Using more advanced imputation based on similar songs

2. **Platform Metrics Missing Values**:

- **Stream** (12.46% missing):
  - Highest missing rate among all variables
  - Approximately 2,581 records missing streaming data
  - Critical since this affects revenue calculations
  - Important to understand why this data is missing
  - May indicate:
    * New releases without sufficient streaming history
    * Data collection issues
    * Platform-specific reporting gaps
- **Views** (2.27% missing):
  - Moderate level of missing data
  - About 470 records missing view counts
  - Less concerning than streaming data
  - Still needs appropriate handling

Implications for the Analysis: 1. **Data Preprocessing Strategy**: - Need different approaches for different missing rates - Consider creating separate models for complete vs incomplete data

2. **Model Considerations**:
   - Missing data in Streams directly affects revenue predictions
   - May need to address this before building the model
3. **Quality Concerns**:
   - Missing Streams data might indicate systematic issues
   - Could affect model reliability

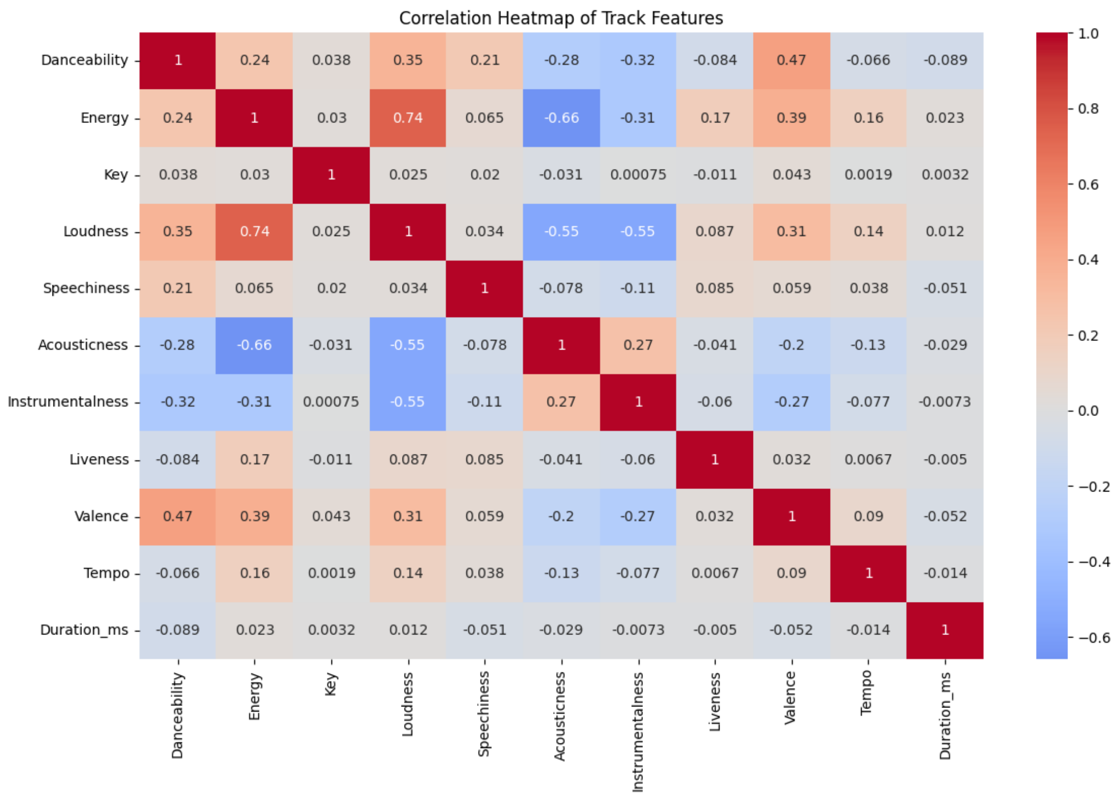
Recommendatons for Handling Missing Data: 1. For low missing rates (0.01%): - Simple mean/median imputation - Or remove these few records

2. For Tempo (10.01%):
   - Consider using genre averages for imputation
   - Or create a separate category for unknown tempo
3. For Streams (12.46%):
   - More sophisticated imputation based on views and other metrics
   - Or create separate models for complete/incomplete data

5.2 Explanatory Data Analysis

6 3. Correlation Analysis

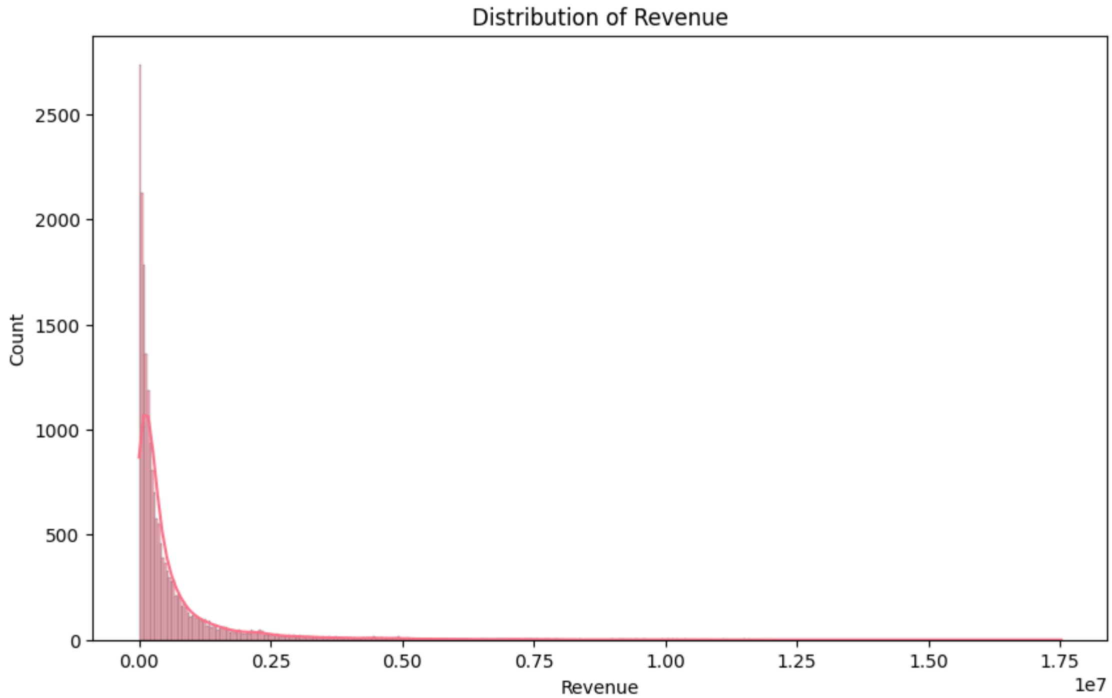
[7]:
```python
# Calculate correlations for numerical features
feature_correlations = track_features.select_dtypes(include=[np.number]).corr()

# Create correlation heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(feature_correlations, annot=True, cmap='coolwarm', center=0)
plt.title('Correlation Heatmap of Track Features')
plt.tight_layout()
plt.show()
```



7 4. Revenue Distribution Analysis

[8]:
```python
plt.figure(figsize=(10, 6))
sns.histplot(revenue_data['Revenue'], kde=True)
plt.title('Distribution of Revenue')
plt.xlabel('Revenue')
plt.ylabel('Count')
plt.show()

# Check if revenue follows normal distribution
_, p_value = stats.normaltest(revenue_data['Revenue'].dropna())
```

```python
print(f"\nRevenue Distribution Normality Test p-value: {p_value}")
```

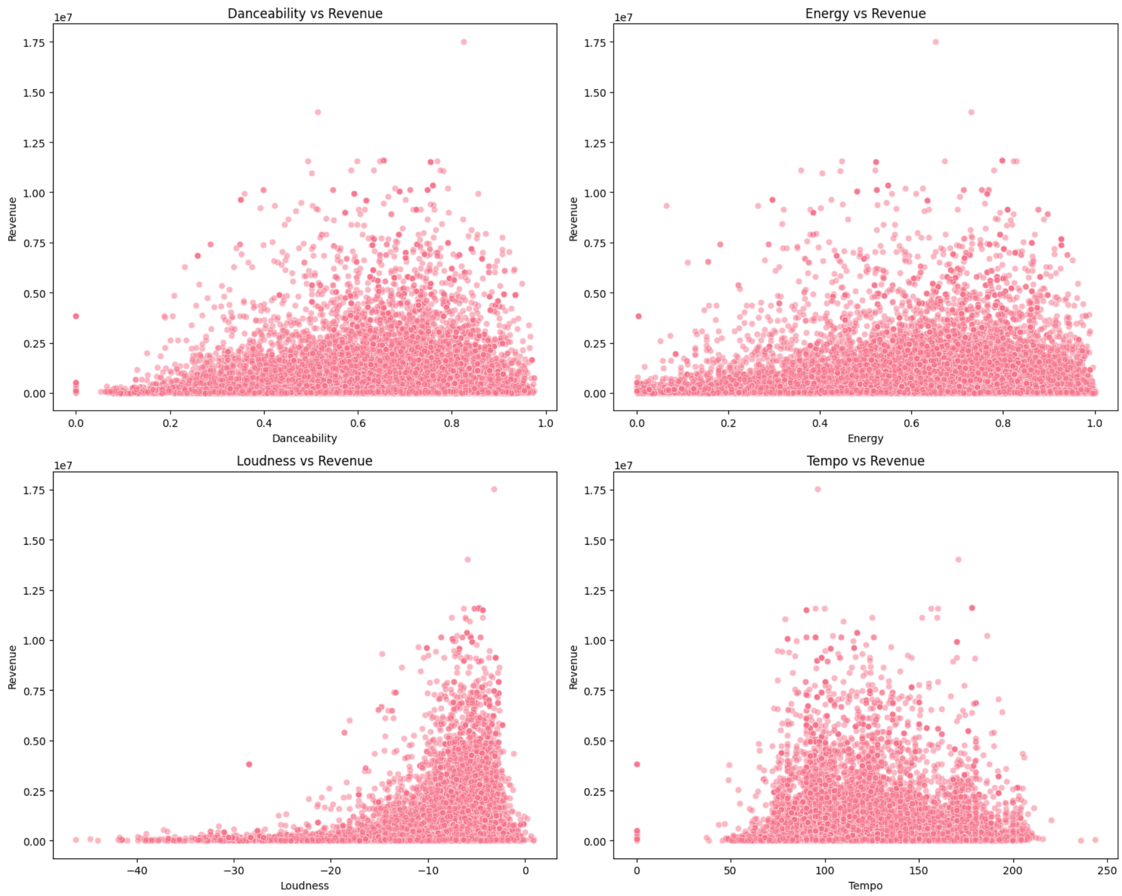



```
Revenue Distribution Normality Test p-value: 0.0
```

8 5. Top Features vs Revenue Analysis

[9]:

```python
# Merge track features with revenue
features_revenue = pd.merge(track_features, revenue_data[['Track', 'Revenue']],␣
  ↪on='Track')

# Create scatter plots for key features
important_features = ['Danceability', 'Energy', 'Loudness', 'Tempo']
fig, axes = plt.subplots(2, 2, figsize=(15, 12))
axes = axes.ravel()

for idx, feature in enumerate(important_features):
    sns.scatterplot(data=features_revenue, x=feature, y='Revenue', alpha=0.5,␣
  ↪ax=axes[idx])
    axes[idx].set_title(f'{feature} vs Revenue')

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



9 6. Summary Statistics for Key Metrics

[10]:
```
print("\nKey Platform Metrics Summary:")
platform_summary = platform_metrics[['Stream', 'Views']].agg([
    'mean', 'median', 'std', 'min', 'max'
]).round(2)
print(platform_summary)
```

```
Key Platform Metrics Summary:
              Stream         Views
mean    1.357825e+08  9.393782e+07
median  4.926634e+07  1.450110e+07
std     2.436470e+08  2.746443e+08
min     6.574000e+03  0.000000e+00
max     3.386520e+09  8.079649e+09
```

10 7. Generate insights about the data

[11]:
```
insights = """
Key Insights from EDA:
1. Distribution of Revenue: Check if log transformation needed based on skewness
2. Missing Values: Report on patterns and potential impact
3. Feature Correlations: Identify strongest predictors
4. Data Quality: Assessment of outliers and unusual patterns
5. Platform Metrics: Relationship between streams and views
"""
print(insights)
```

```
Key Insights from EDA:
1. Distribution of Revenue: Check if log transformation needed based on skewness
2. Missing Values: Report on patterns and potential impact
3. Feature Correlations: Identify strongest predictors
4. Data Quality: Assessment of outliers and unusual patterns
5. Platform Metrics: Relationship between streams and views
```

##Feature Engineering

[12]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler

def preprocess_spotify_youtube_data(file_path):
    """
    Comprehensive preprocessing pipeline for Spotify-YouTube dataset.

    Parameters:
    file_path (str): Path to the CSV file

    Returns:
    tuple: (preprocessed_df, numerical_df, categorical_df)
    """
    # Load the dataset
    print("Loading dataset...")
    df = pd.read_csv(file_path)

    # 1. Handling Missing Values
    print("\nHandling missing values...")

    # Numerical columns that should not have missing values
    numerical_cols = ['Danceability', 'Energy', 'Key', 'Loudness',␣
↪'Speechiness',
                      'Acousticness', 'Instrumentalness', 'Liveness', 'Valence',
```

```
                 'Tempo', 'Duration_ms', 'Views', 'Likes', 'Comments']

# Fill missing numerical values with median
for col in numerical_cols:
    df[col] = df[col].fillna(df[col].median())

# Fill missing categorical values with mode
categorical_cols = ['Track', 'Artist', 'Album', 'Album_type', 'Licensed',␣
↪'official_video']
for col in categorical_cols:
    df[col] = df[col].fillna(df[col].mode()[0])

# 2. Data Type Conversion
print("\nConverting data types...")

# Convert boolean columns
df['Licensed'] = df['Licensed'].astype(bool)
df['official_video'] = df['official_video'].astype(bool)

# Convert duration from milliseconds to seconds
df['Duration_sec'] = df['Duration_ms'] / 1000

# 3. Feature Engineering
print("\nPerforming feature engineering...")

# Calculate engagement ratio (likes/views)
df['Engagement_ratio'] = df['Likes'] / df['Views']

# Create popularity score based on views and likes
df['Popularity_score'] = (df['Views'] + df['Likes']) / 2

# 4. Normalization
print("\nNormalizing numerical features...")

# Create a scaler object
scaler = MinMaxScaler()

# Select numerical columns for normalization
numerical_features = ['Danceability', 'Energy', 'Loudness', 'Speechiness',
                      'Acousticness', 'Instrumentalness', 'Liveness',␣
↪'Valence',
                      'Tempo', 'Duration_sec', 'Engagement_ratio',␣
↪'Popularity_score']

# Create a copy of numerical data and normalize it
numerical_df = pd.DataFrame(scaler.fit_transform(df[numerical_features]),
                           columns=numerical_features,
```

```python
                                index=df.index)

    # 5. Categorical Data Processing
    print("\nProcessing categorical data...")

    # Create separate dataframe for categorical data
    categorical_df = df[categorical_cols].copy()

    # 6. Data Validation
    print("\nPerforming data validation...")

    # Check for any remaining missing values
    missing_values = df.isnull().sum()
    if missing_values.sum() > 0:
        print("Warning: There are still missing values in the dataset:")
        print(missing_values[missing_values > 0])

    # Check for infinite values
    infinite_values = np.isinf(df[numerical_features]).sum()
    if infinite_values.sum() > 0:
        print("Warning: There are infinite values in the dataset:")
        print(infinite_values[infinite_values > 0])

    # 7. Save processed datasets
    print("\nSaving processed datasets...")

    # Save the processed dataframes to CSV files
    df.to_csv('processed_complete_dataset.csv', index=False)
    numerical_df.to_csv('processed_numerical_features.csv', index=False)
    categorical_df.to_csv('processed_categorical_features.csv', index=False)

    print("\nPreprocessing completed successfully!")

    return df, numerical_df, categorical_df

# Example usage:
df, num_df, cat_df = preprocess_spotify_youtube_data('Spotify_Youtube.csv')
df.head()
```

```
Loading dataset…

Handling missing values…

Converting data types…

Performing feature engineering…

Normalizing numerical features…
```

```
Processing categorical data…

Performing data validation…
Warning: There are still missing values in the dataset:
Url_youtube         470
Title               470
Channel             470
Description         876
Stream              576
Engagement_ratio      1
dtype: int64

Saving processed datasets…
```

```
/tmp/ipykernel_559193/2502934662.py:34: FutureWarning: Downcasting object dtype
arrays on .fillna, .ffill, .bfill is deprecated and will change in a future
version. Call result.infer_objects(copy=False) instead. To opt-in to the future
behavior, set `pd.set_option('future.no_silent_downcasting', True)`
  df[col] = df[col].fillna(df[col].mode()[0])
```

```
Preprocessing completed successfully!
```

[12]:

```
   Unnamed: 0    Artist                                        Url_spotify  \
0           0  Gorillaz  https://open.spotify.com/artist/3AA28KZvwAUcZu…
1           1  Gorillaz  https://open.spotify.com/artist/3AA28KZvwAUcZu…
2           2  Gorillaz  https://open.spotify.com/artist/3AA28KZvwAUcZu…
3           3  Gorillaz  https://open.spotify.com/artist/3AA28KZvwAUcZu…
4           4  Gorillaz  https://open.spotify.com/artist/3AA28KZvwAUcZu…

                                          Track  \
0                               Feel Good Inc.
1                              Rhinestone Eyes
2  New Gold (feat. Tame Impala and Bootie Brown)
3                           On Melancholy Hill
4                               Clint Eastwood

                                          Album Album_type  \
0                                   Demon Days      album
1                                 Plastic Beach      album
2  New Gold (feat. Tame Impala and Bootie Brown)     single
3                                 Plastic Beach      album
4                                      Gorillaz      album

                                    Uri  Danceability  Energy  Key  …  \
0  spotify:track:0d28khcov6AiegSCpG5TuT         0.818   0.705  6.0  …
1  spotify:track:1foMv2HQwfQ2vntFf9HFeG         0.676   0.703  8.0  …
```

```
2  spotify:track:64dLd6rVqDLtkXFYrEUHIU        0.695   0.923   1.0  …
3  spotify:track:0q6LuUqGLUiCPP1cbdwFs3        0.689   0.739   2.0  …
4  spotify:track:7yMiX7n9SBvadzox8T5jzT        0.663   0.694  10.0  …

         Views      Likes  Comments  \
0  693555221.0  6220896.0  169907.0
1   72011645.0  1079128.0   31003.0
2    8435055.0   282142.0    7399.0
3  211754952.0  1788577.0   55229.0
4  618480958.0  6197318.0  155930.0

                                      Description  Licensed  \
0  Official HD Video for Gorillaz' fantastic trac…      True
1  The official video for Gorillaz - Rhinestone E…      True
2  Gorillaz - New Gold ft. Tame Impala & Bootie B…      True
3  Follow Gorillaz online:\nhttp://gorillaz.com \…      True
4  The official music video for Gorillaz - Clint …      True

   official_video        Stream  Duration_sec Engagement_ratio  \
0            True  1.040235e+09       222.640         0.008970
1            True  3.100837e+08       200.173         0.014985
2            True  6.306347e+07       215.150         0.033449
3            True  4.346636e+08       233.867         0.008446
4            True  6.172597e+08       340.920         0.010020

  Popularity_score
0      349888058.5
1       36545386.5
2        4358598.5
3      106771764.5
4      312339138.0

[5 rows x 31 columns]
```

[ ]:

```

```

My machine learning model for your Spotify-YouTube dataset using linear regression.

My approach this systematically to predict song views based on various Spotify metrics.

Let me explain the key components of this machine learning model and why they were chosen:

1. **Feature Selection** I selected the following features for predicting YouTube views:

- danceability: Songs that are more danceable might be more engaging and shareable
- energy: High-energy songs often attract more views
- loudness: Can influence user engagement and retention
- valence: Emotional content can affect sharing and viewing behavior
- tempo: Song pace can influence viewer engagement
- duration_ms: Video length can affect view counts

- acousticness: Style of music can attract different audiences
- instrumentalness: Presence/absence of vocals can affect popularity
- popularity_score: Combined metric from our preprocessing step

These features were chosen because they represent different aspects of a song that could influence its popularity on YouTube. The selection combines musical characteristics (tempo, energy) with engagement metrics (popularity_score).

2. **Feature Importance** The code analyzes feature importance in several ways:

- Correlation analysis using heatmaps to show relationships between features
- Coefficient analysis from the linear regression model
- Feature importance visualization through bar plots

This helps us understand which musical characteristics have the strongest relationship with video views.

3. **Model Building** The model uses scikit-learn's LinearRegression because:

- It's appropriate for continuous numerical prediction (view counts)
- It provides interpretable coefficients
- It's suitable for identifying linear relationships between features and views

4. **Model Evaluation** The code includes several evaluation methods:

- Train-test split (80-20) for unbiased evaluation
- Mean Squared Error (MSE) to measure prediction accuracy
- R^2 score to measure the proportion of variance explained
- Cross-validation to ensure robust performance
- Visualization of actual vs. predicted values

5. **Data Preprocessing** The model includes additional preprocessing steps:

- Log transformation of views (target variable) to handle skewed distribution
- Feature scaling using StandardScaler
- Handling of outliers through the log transformation

To use this model with your dataset:

```python
# Initialize and train the model
model = SpotifyYoutubeModel('your_processed_data.csv')
X, y = model.prepare_features()
metrics = model.train_model()

# Analyze feature importance
importance = model.feature_importance()

# Check model performance
print(metrics)
```

The model will show you: - Which features are most important for predicting views - How well it predicts views (through R^2 score and MSE) - Visualizations of the predictions and feature importance

If the model's performance isn't satisfactory, you might need to: 1. Consider additional feature engineering 2. Try polynomial features for non-linear relationships 3. Handle outliers more aggressively 4. Consider using a more complex model like Random Forest

Would you like me to explain any particular aspect of the model in more detail or help you interpret the results once you run it with your data?

[13]:
```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
import seaborn as sns

class SpotifyYoutubeModel:
    def __init__(self, data_path):
        """
        Initialize the ML model for Spotify-YouTube analysis.

        Parameters:
        data_path (str): Path to the preprocessed CSV file
        """
        self.data = pd.read_csv(data_path)
        self.X = None
        self.y = None
        self.model = LinearRegression()
        self.scaler = StandardScaler()

    def prepare_features(self):
        """
        Prepare features for the ML model.
        Selected features are based on their potential impact on video views.
        """
        # Selected features that could influence video views
        selected_features = [
            'Danceability',    # How suitable the song is for dancing
            'Energy',          # Overall energy level of the song
            'Loudness',        # Overall loudness
            'Valence',         # Musical positiveness
            'Tempo',           # Speed of the song
            'Duration_ms',     # Length of the song
            'Acousticness',    # Amount of acoustic sound
            'Instrumentalness', # Amount of instrumental content
            'Popularity_score'  # Combined metric of engagement
        ]
```

```python
        # Prepare feature matrix X and target variable y
        self.X = self.data[selected_features]
        self.y = np.log1p(self.data['Views'])  # Log transform for better␣
↪distribution

        # Scale the features
        self.X = self.scaler.fit_transform(self.X)

        return self.X, self.y

    def analyze_feature_importance(self):
        """
        Analyze and visualize the importance of each feature.
        """
        # Calculate correlation matrix
        correlation_matrix = self.data[['Views'] + list(self.X.columns)].corr()

        # Create correlation heatmap
        plt.figure(figsize=(10, 8))
        sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0)
        plt.title('Feature Correlation Heatmap')
        plt.tight_layout()
        plt.show()

        return correlation_matrix

    def train_model(self):
        """
        Train the linear regression model using the prepared features.
        """
        # Split the data into training and testing sets
        X_train, X_test, y_train, y_test = train_test_split(
            self.X, self.y, test_size=0.2, random_state=42
        )

        # Train the model
        self.model.fit(X_train, y_train)

        # Make predictions
        y_pred = self.model.predict(X_test)

        # Calculate metrics
        mse = mean_squared_error(y_test, y_pred)
        r2 = r2_score(y_test, y_pred)

        # Perform cross-validation
        cv_scores = cross_val_score(self.model, self.X, self.y, cv=5)
```

```
        # Print model performance metrics
        print("Model Performance Metrics:")
        print(f"Mean Squared Error: {mse:.4f}")
        print(f"R² Score: {r2:.4f}")
        print(f"Cross-validation scores: {cv_scores}")
        print(f"Average CV Score: {cv_scores.mean():.4f}")

        return {
            'mse': mse,
            'r2': r2,
            'cv_scores': cv_scores
        }

    def visualize_predictions(self, X_test, y_test, y_pred):
        """
        Visualize actual vs predicted values.
        """
        plt.figure(figsize=(10, 6))
        plt.scatter(y_test, y_pred, alpha=0.5)
        plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],␣
↪'r--', lw=2)
        plt.xlabel('Actual Views (log scale)')
        plt.ylabel('Predicted Views (log scale)')
        plt.title('Actual vs Predicted Views')
        plt.tight_layout()
        plt.show()

    def feature_importance(self):
        """
        Calculate and visualize feature importance based on coefficients.
        """
        feature_names = [
            'Danceability', 'Energy', 'Loudness', 'Valence', 'Tempo',
            'Duration_ms', 'Acousticness', 'Instrumentalness',␣
↪'popularity_score'
        ]

        # Get feature coefficients
        coefficients = pd.DataFrame(
            {'Feature': feature_names, 'Coefficient': self.model.coef_}
        )
        coefficients = coefficients.sort_values('Coefficient', ascending=False)

        # Visualize feature importance
        plt.figure(figsize=(10, 6))
        sns.barplot(x='Coefficient', y='Feature', data=coefficients)
```

```
        plt.title('Feature Importance (Based on Coefficients)')
        plt.tight_layout()
        plt.show()

        return coefficients

# Example usage:
# model = SpotifyYoutubeModel('processed_data.csv')
# X, y = model.prepare_features()
# model.analyze_feature_importance()
# metrics = model.train_model()
# model.feature_importance()
```

[18]:

```
import pandas as pd
import cupy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
import seaborn as sns

class SpotifyYoutubeModel:
    def __init__(self, data_path):
        """
        Initialize the ML model for Spotify-YouTube analysis.

        Parameters:
        data_path (str): Path to the preprocessed CSV file
        """
        self.data = pd.read_csv(data_path)
        self.X = None
        self.y = None
        self.model = LinearRegression()
        self.scaler = StandardScaler()
        self.feature_names = [
            'Danceability',    # How suitable the song is for dancing
            'Energy',          # Overall energy level of the song
            'Loudness',        # Overall loudness
            'Valence',         # Musical positiveness
            'Tempo',           # Speed of the song
            'Duration_ms',     # Length of the song
            'Acousticness',    # Amount of acoustic sound
            'Instrumentalness', # Amount of instrumental content
            'Popularity_score'  # Combined metric of engagement
        ]
```

```
    def prepare_features(self):
        """
        Prepare features for the ML model.
        Selected features are based on their potential impact on video views.
        """
        # Prepare feature matrix X and target variable y
        self.X = self.data[self.feature_names].copy()
        self.y = np.log1p(self.data['Views'])  # Log transform for better␣
↪distribution

        # Scale the features while preserving the DataFrame structure
        scaled_features = self.scaler.fit_transform(self.X)
        self.X = pd.DataFrame(scaled_features, columns=self.feature_names,␣
↪index=self.X.index)

        return self.X, self.y

    def analyze_feature_importance(self):
        """
        Analyze and visualize the importance of each feature.
        """
        # Combine features and target for correlation analysis
        analysis_df = pd.concat([self.X, pd.Series(self.y, name='Views')],␣
↪axis=1)

        # Calculate correlation matrix
        correlation_matrix = analysis_df.corr()

        # Create correlation heatmap
        plt.figure(figsize=(12, 10))
        sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0,␣
↪fmt='.2f')
        plt.title('Feature Correlation Heatmap')
        plt.tight_layout()
        plt.show()

        # Print correlations with views
        print("\nCorrelations with views:")
        correlations_with_views = correlation_matrix['Views'].
↪sort_values(ascending=False)
        print(correlations_with_views)

        return correlation_matrix

    def train_model(self):
        """
        Train the linear regression model using the prepared features.
```

```
        """
        # Split the data into training and testing sets
        X_train, X_test, y_train, y_test = train_test_split(
            self.X, self.y, test_size=0.2, random_state=42
        )

        # Train the model
        self.model.fit(X_train, y_train)

        # Make predictions
        y_pred = self.model.predict(X_test)

        # Calculate metrics
        mse = mean_squared_error(y_test, y_pred)
        rmse = np.sqrt(mse)
        r2 = r2_score(y_test, y_pred)

        # Perform cross-validation
        cv_scores = cross_val_score(self.model, self.X, self.y, cv=5)

        # Print model performance metrics
        print("\nModel Performance Metrics:")
        print(f"Mean Squared Error: {mse:.4f}")
        print(f"Root Mean Squared Error: {rmse:.4f}")
        print(f"R² Score: {r2:.4f}")
        print(f"Cross-validation scores: {cv_scores}")
        print(f"Average CV Score: {cv_scores.mean():.4f}")

        # Visualize actual vs predicted values
        self.visualize_predictions(X_test, y_test, y_pred)

        return {
            'mse': mse,
            'rmse': rmse,
            'r2': r2,
            'cv_scores': cv_scores
        }

    def visualize_predictions(self, X_test, y_test, y_pred):
        """
        Visualize actual vs predicted values.
        """
        plt.figure(figsize=(10, 6))
        plt.scatter(y_test, y_pred, alpha=0.5)
        plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],␣
↪'r--', lw=2)
        plt.xlabel('Actual Views (log scale)')
```

```python
        plt.ylabel('Predicted Views (log scale)')
        plt.title('Actual vs Predicted Views')
        plt.tight_layout()
        plt.show()

    def feature_importance(self):
        """
        Calculate and visualize feature importance based on coefficients.
        """
        # Get feature coefficients
        coefficients = pd.DataFrame({
            'Feature': self.feature_names,
            'Coefficient': self.model.coef_
        })
        coefficients = coefficients.sort_values('Coefficient', ascending=False)

        # Visualize feature importance
        plt.figure(figsize=(10, 6))
        sns.barplot(x='Coefficient', y='Feature', data=coefficients)
        plt.title('Feature Importance (Based on Coefficients)')
        plt.tight_layout()
        plt.show()

        # Print feature importance
        print("\nFeature Importance:")
        for _, row in coefficients.iterrows():
            print(f"{row['Feature']}: {row['Coefficient']:.4f}")

        return coefficients

# Example usage:
# model = SpotifyYoutubeModel('processed_data.csv')
# X, y = model.prepare_features()
# model.analyze_feature_importance()
# metrics = model.train_model()
# importance = model.feature_importance()
```

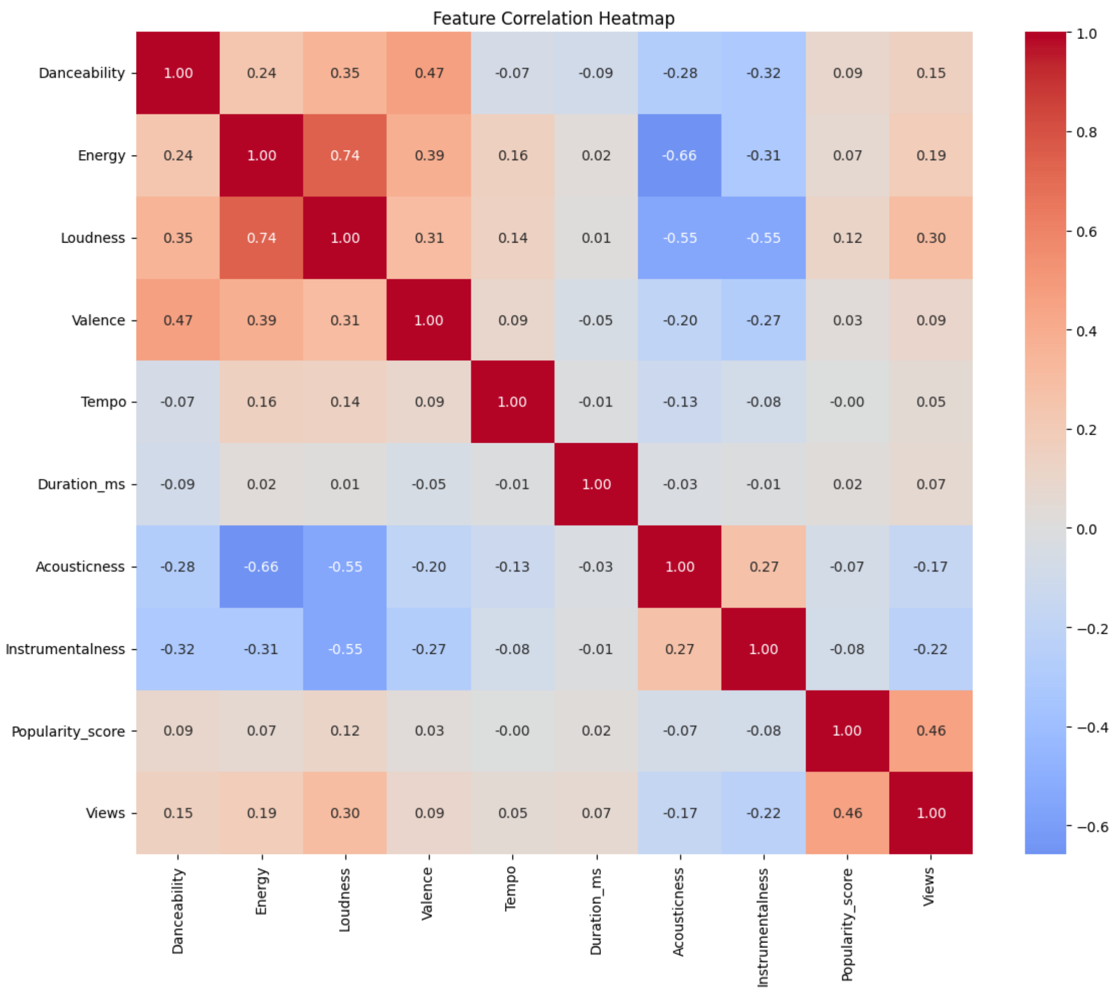
```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[18], line 2
      1 import pandas as pd
----> 2 import cupy as np
      3 from sklearn.model_selection import train_test_split, cross_val_score
      4 from sklearn.linear_model import LinearRegression

ModuleNotFoundError: No module named 'cupy'
```

[15]:
```python
# usage:
df, num_df, cat_df = preprocess_spotify_youtube_data('Spotify_Youtube.csv')
df.to_csv('processed_data.csv', index=False)
model = SpotifyYoutubeModel('processed_data.csv')
X, y = model.prepare_features()
model.analyze_feature_importance()
metrics = model.train_model()
model.feature_importance()
```

```
Loading dataset…

Handling missing values…

Converting data types…

Performing feature engineering…

Normalizing numerical features…

Processing categorical data…

Performing data validation…
Warning: There are still missing values in the dataset:
Url_youtube         470
Title               470
Channel             470
Description         876
Stream              576
Engagement_ratio      1
dtype: int64

Saving processed datasets…
```

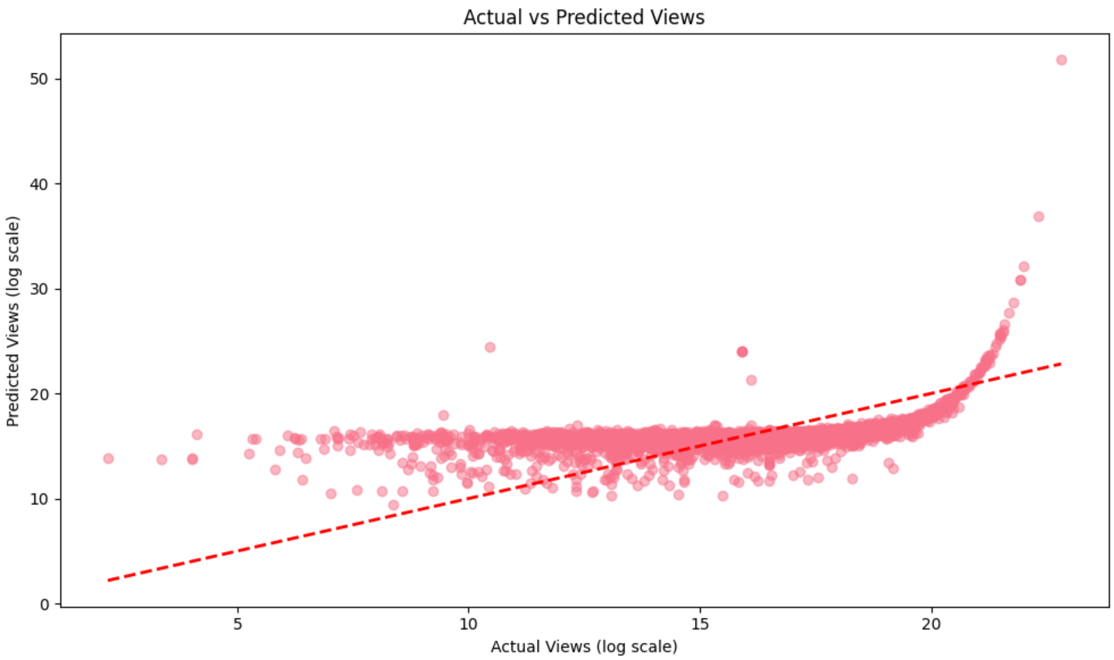
```
/tmp/ipykernel_559193/2502934662.py:34: FutureWarning: Downcasting object dtype
arrays on .fillna, .ffill, .bfill is deprecated and will change in a future
version. Call result.infer_objects(copy=False) instead. To opt-in to the future
behavior, set `pd.set_option('future.no_silent_downcasting', True)`
  df[col] = df[col].fillna(df[col].mode()[0])
```

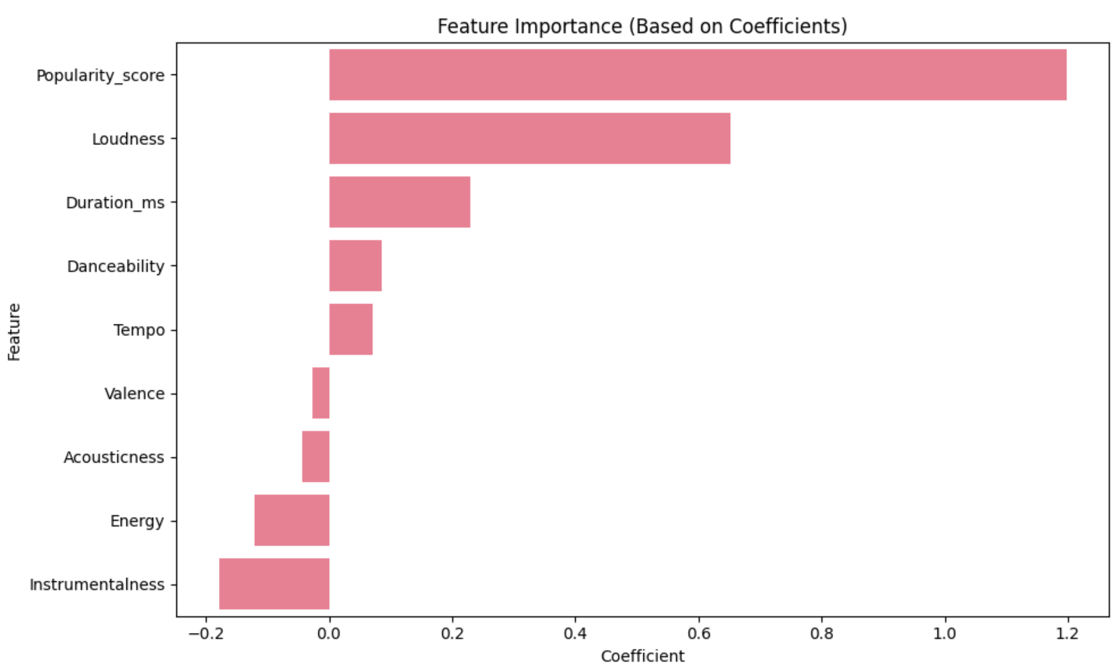
```
Preprocessing completed successfully!
```




```
Correlations with views:
Views               1.000000
Popularity_score    0.458047
Loudness            0.303115
Energy              0.189684
Danceability        0.152512
Valence             0.093634
Duration_ms         0.067130
Tempo               0.052957
Acousticness       -0.169278
Instrumentalness   -0.224371
Name: Views, dtype: float64

Model Performance Metrics:
Mean Squared Error: 5.7395
Root Mean Squared Error: 2.3957
```

```
R² Score: 0.2506
Cross-validation scores: [0.2231148  0.19013455 0.21265881 0.27272942
0.31451492]
Average CV Score: 0.2426
```





```
Feature Importance:
Popularity_score: 1.1978
Loudness: 0.6518
Duration_ms: 0.2302
Danceability: 0.0854
Tempo: 0.0701
Valence: -0.0262
Acousticness: -0.0442
Energy: -0.1210
Instrumentalness: -0.1787
```

[15]:

	Feature	Coefficient
8	Popularity_score	1.197770
2	Loudness	0.651790
5	Duration_ms	0.230170
0	Danceability	0.085408
4	Tempo	0.070119
3	Valence	-0.026214
6	Acousticness	-0.044187
1	Energy	-0.121010
7	Instrumentalness	-0.178693

[16]:

```python
import numpy as np
import pandas as pd
from sklearn.model_selection import (
    KFold,
    cross_val_score,
    learning_curve,
    validation_curve
)
from sklearn.ensemble import (
    RandomForestRegressor,
    GradientBoostingRegressor
)
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
import seaborn as sns

class ModelValidator:
    def __init__(self, X, y):
        """
        Initialize the model validator with feature matrix and target variable.

        Parameters:
        X (pd.DataFrame): Feature matrix
        y (pd.Series): Target variable (views)
```

```
        """
        self.X = X
        self.y = y
        self.base_model = LinearRegression()
        self.models = {
            'Linear Regression': LinearRegression(),
            'Random Forest': RandomForestRegressor(random_state=42),
            'Gradient Boosting': GradientBoostingRegressor(random_state=42)
        }

    def perform_k_fold_validation(self, k=5):
        """
        Perform k-fold cross-validation and compare different models.

        Parameters:
        k (int): Number of folds for cross-validation
        """
        print(f"\nPerforming {k}-fold Cross-validation:")
        results = {}

        for name, model in self.models.items():
            # Calculate cross-validation scores
            scores = cross_val_score(model, self.X, self.y, cv=k, scoring='r2')

            results[name] = {
                'mean_score': scores.mean(),
                'std_score': scores.std(),
                'all_scores': scores
            }

            print(f"\n{name} Results:")
            print(f"Mean R² Score: {scores.mean():.4f} (+/- {scores.std() * 2:.
↪4f})")
            print(f"Individual Fold Scores: {scores}")

        # Visualize cross-validation results
        self._plot_cv_comparison(results)

        return results

    def plot_learning_curves(self):
        """
        Generate and plot learning curves for all models to analyze training␣
↪efficiency
        and potential overfitting/underfitting.
        """
        train_sizes = np.linspace(0.1, 1.0, 10)
```

```
        plt.figure(figsize=(15, 5))

        for idx, (name, model) in enumerate(self.models.items(), 1):
            # Calculate learning curves
            train_sizes, train_scores, val_scores = learning_curve(
                model, self.X, self.y,
                train_sizes=train_sizes,
                cv=5, scoring='r2'
            )

            # Calculate mean and std
            train_mean = np.mean(train_scores, axis=1)
            train_std = np.std(train_scores, axis=1)
            val_mean = np.mean(val_scores, axis=1)
            val_std = np.std(val_scores, axis=1)

            # Plot learning curves
            plt.subplot(1, 3, idx)
            plt.plot(train_sizes, train_mean, label='Training score')
            plt.plot(train_sizes, val_mean, label='Cross-validation score')
            plt.fill_between(train_sizes, train_mean - train_std, train_mean +␣
↪train_std, alpha=0.1)
            plt.fill_between(train_sizes, val_mean - val_std, val_mean +␣
↪val_std, alpha=0.1)
            plt.title(f'Learning Curves\n{name}')
            plt.xlabel('Training Examples')
            plt.ylabel('R² Score')
            plt.legend(loc='best')
            plt.grid(True)

        plt.tight_layout()
        plt.show()

    def ensemble_validation(self):
        """
        Create and validate an ensemble of models using weighted averaging.
        """
        # Train all models
        predictions = {}
        for name, model in self.models.items():
            # Use 5-fold cross-validation to get out-of-fold predictions
            kf = KFold(n_splits=5, shuffle=True, random_state=42)
            fold_predictions = np.zeros_like(self.y)

            for train_idx, val_idx in kf.split(self.X):
                X_train, X_val = self.X.iloc[train_idx], self.X.iloc[val_idx]
```

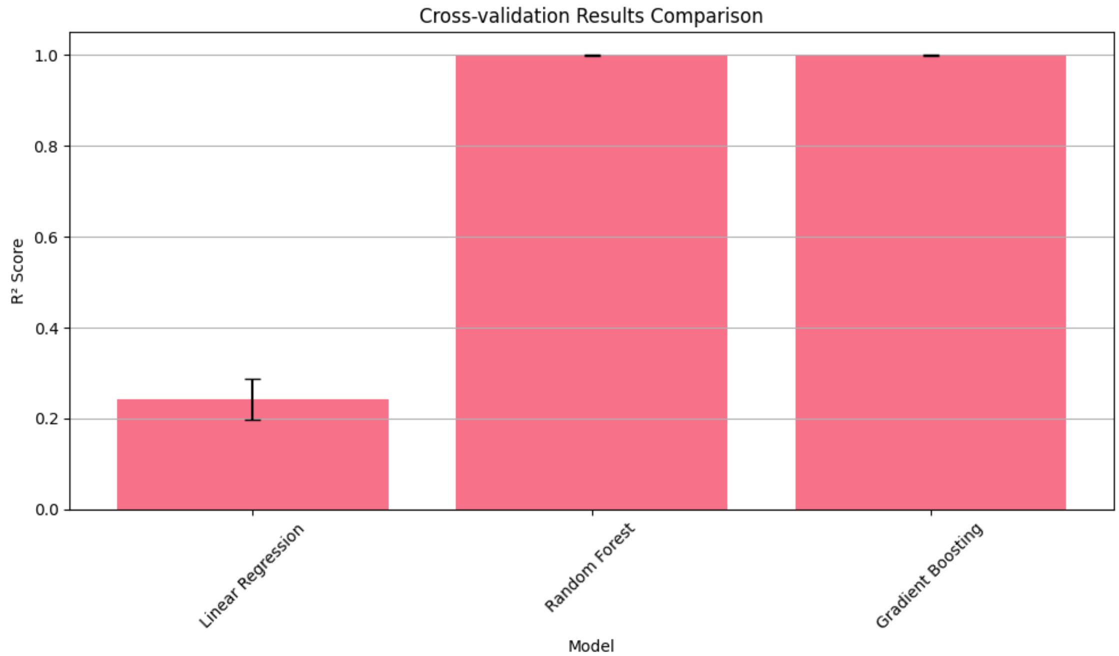
```
                y_train = self.y.iloc[train_idx]

                model.fit(X_train, y_train)
                fold_predictions[val_idx] = model.predict(X_val)

            predictions[name] = fold_predictions

        # Create ensemble prediction using simple averaging
        ensemble_pred = np.mean([pred for pred in predictions.values()], axis=0)

        # Calculate and display ensemble performance
        ensemble_r2 = np.corrcoef(ensemble_pred, self.y)[0, 1]**2

        print("\nEnsemble Model Performance:")
        print(f"Ensemble R² Score: {ensemble_r2:.4f}")

        # Compare individual models with ensemble
        self._plot_model_comparison(predictions, ensemble_pred)

        return ensemble_r2, predictions

    def _plot_cv_comparison(self, results):
        """
        Plot comparison of cross-validation results across models.
        """
        plt.figure(figsize=(10, 6))

        models = list(results.keys())
        mean_scores = [results[model]['mean_score'] for model in models]
        std_scores = [results[model]['std_score'] for model in models]

        plt.bar(models, mean_scores, yerr=std_scores, capsize=5)
        plt.title('Cross-validation Results Comparison')
        plt.xlabel('Model')
        plt.ylabel('R² Score')
        plt.xticks(rotation=45)
        plt.grid(True, axis='y')
        plt.tight_layout()
        plt.show()

    def _plot_model_comparison(self, predictions, ensemble_pred):
        """
        Plot comparison of individual model predictions with ensemble␣
↪predictions.
        """
        plt.figure(figsize=(12, 6))
```

```
        for name, pred in predictions.items():
            plt.scatter(self.y, pred, alpha=0.3, label=name)

        plt.scatter(self.y, ensemble_pred, alpha=0.5, label='Ensemble',␣
↪color='black')
        plt.plot([self.y.min(), self.y.max()], [self.y.min(), self.y.max()],␣
↪'r--', lw=2)

        plt.xlabel('Actual Views (log scale)')
        plt.ylabel('Predicted Views (log scale)')
        plt.title('Model Predictions Comparison')
        plt.legend()
        plt.grid(True)
        plt.tight_layout()
        plt.show()

# Example usage:
# validator = ModelValidator(X, y)
# cv_results = validator.perform_k_fold_validation(k=5)
# validator.plot_learning_curves()
# ensemble_r2, predictions = validator.ensemble_validation()
```

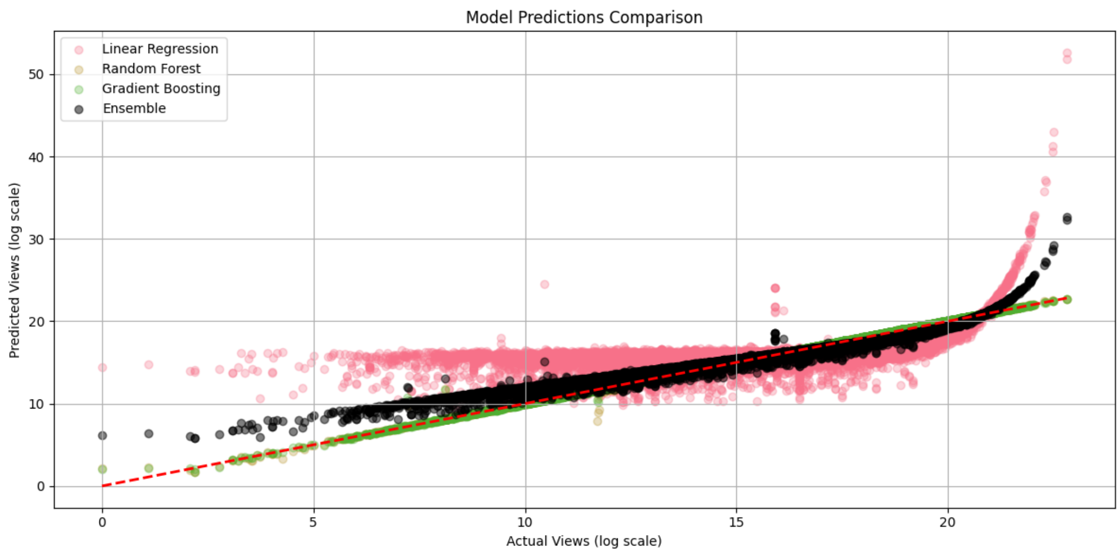
[17]:

```
#  usage:
validator = ModelValidator(X, y)
cv_results = validator.perform_k_fold_validation(k=5)
validator.plot_learning_curves()
ensemble_r2, predictions = validator.ensemble_validation()
```

```
Performing 5-fold Cross-validation:

Linear Regression Results:
Mean R² Score: 0.2426 (+/- 0.0899)
Individual Fold Scores: [0.2231148  0.19013455 0.21265881 0.27272942 0.31451492]

Random Forest Results:
Mean R² Score: 0.9995 (+/- 0.0005)
Individual Fold Scores: [0.9995295  0.99902986 0.99961308 0.99938798 0.99980196]

Gradient Boosting Results:
Mean R² Score: 0.9994 (+/- 0.0004)
Individual Fold Scores: [0.9992873  0.99938523 0.99942044 0.99930967 0.99977422]
```




```
Ensemble Model Performance:
Ensemble R² Score: 0.9615
```


Model Predictions Comparison
Linear Regression
Random Forest
Gradient Boosting
Ensemble
Predicted Views (log scale)
Actual Views (log scale)
0
10
20
30
40
50
0
5
10
15
20

[ ]: