Read Input Data from Kaggle

```
! pip install -q kaggle
```

```
from google.colab import files
files.upload()
```

```
Choose Files No file chosen  Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving kaggle.json to kaggle.json
{'kaggle.json': b'{"username":"minfeishen","key":"7ed3293191e3b63c97dbfc0d352330fc"}'}
```

```
! mkdir ~/.kaggle
```

```
mkdir: cannot create directory ‘/root/.kaggle’: File exists
```

```
! cp kaggle.json ~/.kaggle/
```

```
! chmod 600 ~/.kaggle/kaggle.json
```

```
! kaggle datasets list
```

ref	title	size	lastUpdated	downloadCount	voteCount	usabilit
jayaantanaath/student-habits-vs-academic-performance	Student Habits vs Academic Performance	19512	2025-04-12 10:49:08.663000	19905	341	1.0
adilshamim8/cost-of-international-education	Cost of International Education	18950	2025-05-07 15:41:53.213000	2826	45	1.0
fatemehmohammadinia/heart-attack-dataset-tarik-a-rashid	Heart Attack Dataset	16250	2025-04-30 21:58:22.740000	3667	64	1.0
ivankmk/thousand-ml-jobs-in-usa	Machine Learning Job Postings in the US	1682058	2025-04-20 16:11:59.347000	2814	54	1.0
glowstudygram/spotify-songs-and-artists-dataset	Spotify Songs and Artists Dataset | Audio Features	68415	2025-04-27 12:38:36.850000	1558	27	0.82352
khushikyad001/impact-of-screen-time-on-mental-health	Impact of Screen Time on Mental Health	64873	2025-04-20 18:01:47.570000	2273	42	1.0
aryan208/financial-transactions-dataset-for-fraud-detection	Financial Transactions Dataset for Fraud Detection	290256858	2025-05-02 09:12:28.203000	822	25	1.0
palvinder2006/ola-bike-ride-request	Ola Bike Ride Request	174975	2025-04-28 03:55:33.860000	953	23	1.0
mahdimashayekhi/fake-news-detection-dataset	Fake News Detection Dataset	11735585	2025-04-27 14:52:10.607000	1028	11	1.0
umeradnaan/daily-social-media-active-users	Daily Social Media Active Users	126814	2025-05-05 02:11:50.873000	1379	21	1.0
zahidmughal2343/global-cancer-patients-2015-2024	global_cancer_patients_2015_2024	1261049	2025-04-14 00:05:23.367000	4683	57	1.0
adilshamim8/greenhouse-plant-growth-metrics	Greenhouse Plant Growth	3041046	2025-04-19 07:33:57.787000	1553	23	1.0
khushikyad001/screen-time-and-app-usage-dataset-iosandroid	Screen Time and App Usage Dataset (iOS/Android)	157038	2025-04-19 13:23:41.067000	1828	29	1.0
adilshamim8/predict-students-dropout-and-academic-success	Student Dropout & Success Prediction Dataset	106181	2025-04-23 06:34:06.433000	2229	33	1.0
nikolasgegenava/sneakers-classification	Popular Sneakers Classification	17981294	2025-05-01 12:00:45.517000	1372	36	1.0
samithsachidanandan/1000-most-trending-youtube-videos	1000 Most Trending YouTube Videos	43395	2025-04-19 10:11:14.973000	2242	34	1.0
kapturovalexander/bank-credit-risk-assessment	Bank credit risk assessment 💵🏦💼	2537159	2025-05-06 05:04:50.793000	1174	32	1.0
zahidmughal2343/amazon-sales-2025	Amazon Sales 2025	3617	2025-04-03 22:08:13.607000	8058	93	1.0
sahirmaharajj/bird-migration-dataset-data-visualization-eda	Bird Migration Dataset (Data Visualization / EDA)	1002468	2025-04-23 18:33:29.913000	1223	23	1.0
dansbecker/melbourne-housing-snapshot	Melbourne Housing Snapshot	461423	2018-06-05 12:52:24.087000	179137	1614	0.70588

```
! kaggle competitions download -c playground-series-s5e4
```

```
! mkdir train
```

```
! unzip playground-series-s5e4.zip -d train
```

```
Archive:  playground-series-s5e4.zip
  inflating: train/sample_submission.csv
  inflating: train/test.csv
  inflating: train/train.csv
```

EDA

```
#Imports
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd



train_df = pd.read_csv('train/train.csv')
train_df.info()
train_df.describe()
train_df.isnull().sum()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 750000 entries, 0 to 749999
Data columns (total 12 columns):
 #   Column                       Non-Null Count   Dtype
---  ------                       --------------   -----
 0   id                           750000 non-null  int64
 1   Podcast_Name                 750000 non-null  object
 2   Episode_Title                750000 non-null  object
 3   Episode_Length_minutes       662907 non-null  float64
 4   Genre                        750000 non-null  object
 5   Host_Popularity_percentage   750000 non-null  float64
 6   Publication_Day              750000 non-null  object
 7   Publication_Time             750000 non-null  object
 8   Guest_Popularity_percentage  603970 non-null  float64
 9   Number_of_Ads                749999 non-null  float64
 10  Episode_Sentiment            750000 non-null  object
 11  Listening_Time_minutes       750000 non-null  float64
dtypes: float64(5), int64(1), object(6)
memory usage: 68.7+ MB
```

	0
id	0
Podcast_Name	0
Episode_Title	0
Episode_Length_minutes	87093
Genre	0
Host_Popularity_percentage	0
Publication_Day	0
Publication_Time	0
Guest_Popularity_percentage	146030
Number_of_Ads	1
Episode_Sentiment	0
Listening_Time_minutes	0

dtype: int64

```
test_df = pd.read_csv('train/test.csv')
test_df.info()
test_df.describe()
test_df.isnull().sum()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 250000 entries, 0 to 249999
Data columns (total 11 columns):
 #   Column                       Non-Null Count   Dtype
---  ------                       --------------   -----
 0   id                           250000 non-null  int64
 1   Podcast_Name                 250000 non-null  object
 2   Episode_Title                250000 non-null  object
 3   Episode_Length_minutes       221264 non-null  float64
 4   Genre                        250000 non-null  object
 5   Host_Popularity_percentage   250000 non-null  float64
 6   Publication_Day              250000 non-null  object
 7   Publication_Time             250000 non-null  object
 8   Guest_Popularity_percentage  201168 non-null  float64
 9   Number_of_Ads                250000 non-null  float64
 10  Episode_Sentiment            250000 non-null  object
dtypes: float64(4), int64(1), object(6)
memory usage: 21.0+ MB
```

	0
id	0
Podcast_Name	0
Episode_Title	0
Episode_Length_minutes	28736
Genre	0
Host_Popularity_percentage	0
Publication_Day	0
Publication_Time	0
Guest_Popularity_percentage	48832
Number_of_Ads	0
Episode_Sentiment	0

dtype: int64

Numerical exploration

```
podcast_num = train_df["Podcast_Name"].nunique()
print(f"There are {podcast_num} podcast names in the train dataset.")
```

```
There are 48 podcast names in the train dataset.
```

```
np.sum(~np.isin(test_df["Podcast_Name"].unique().tolist(), train_df["Podcast_Name"].unique().tolist()))
```

```
np.int64(0)
```

```
print("Train Data: Number_of_Ads Frequency Table\n")
print(train_df["Number_of_Ads"].value_counts())
```

```
print("\n")
print("Test Data: Number_of_Ads Frequency Table\n")
print(test_df["Number_of_Ads"].value_counts())
```

```
Train Data: Number_of_Ads Frequency Table

Number_of_Ads
0.00      217592
1.00      214069
3.00      160173
2.00      158156
103.25         2
53.37          1
103.00         1
103.91         1
53.42          1
103.75         1
12.00          1
103.88         1
Name: count, dtype: int64


Test Data: Number_of_Ads Frequency Table

Number_of_Ads
0.00       72863
1.00       71015
3.00       53556
2.00       52564
89.12          1
2063.00        1
Name: count, dtype: int64
```

```
print("Train Data: Host_Popularity_percentage Summary Statistics\n")
print(train_df["Host_Popularity_percentage"].describe())
print("\n")
print("Test Data: Host_Popularity_percentage Summary Statistics\n")
print(test_df["Host_Popularity_percentage"].describe())
```

```
Train Data: Host_Popularity_percentage Summary Statistics

count    750000.000000
mean         59.859901
std          22.873098
min           1.300000
25%          39.410000
50%          60.050000
75%          79.530000
max         119.460000
Name: Host_Popularity_percentage, dtype: float64


Test Data: Host_Popularity_percentage Summary Statistics

count    250000.000000
mean         59.716491
std          22.880028
min           2.490000
25%          39.250000
50%          59.900000
75%          79.390000
max         117.760000
Name: Host_Popularity_percentage, dtype: float64
```

```
print("Train Data: Guest_Popularity_percentage Summary Statistics\n")
print(train_df["Guest_Popularity_percentage"].describe())
print("\n")
print("Test Data: Guest_Popularity_percentage Summary Statistics\n")
print(test_df["Guest_Popularity_percentage"].describe())
```

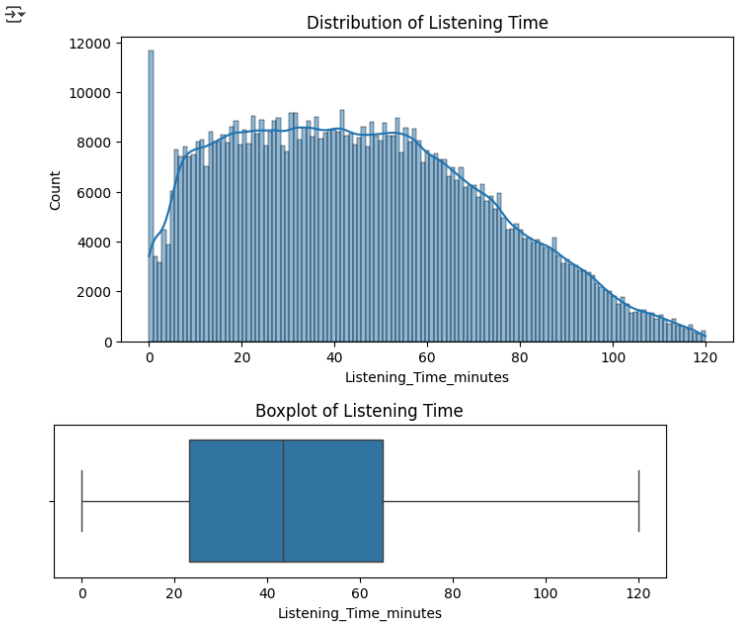
```
Train Data: Guest_Popularity_percentage Summary Statistics

count    603970.000000
mean         52.236449
std          28.451241
min           0.000000
25%          28.380000
50%          53.580000
75%          76.600000
max         119.910000
Name: Guest_Popularity_percentage, dtype: float64


Test Data: Guest_Popularity_percentage Summary Statistics

count    201168.000000
mean         52.192796
std          28.445034
min           0.000000
25%          28.320000
50%          53.360000
75%          76.560000
max         116.820000
Name: Guest_Popularity_percentage, dtype: float64
```

Graphical Visualization

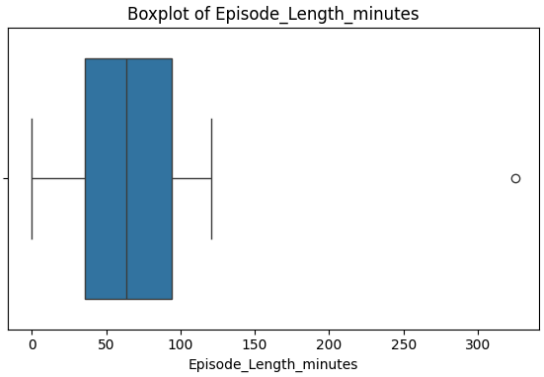
Numerical Feature

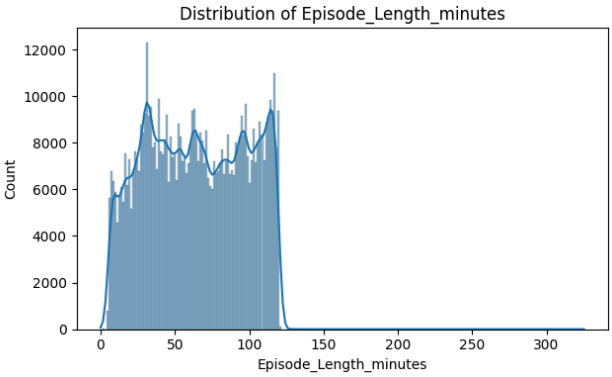
```
#check for skewness/outlier


plt.figure(figsize=(8, 4))
sns.histplot(train_df['Listening_Time_minutes'], kde=True)
```

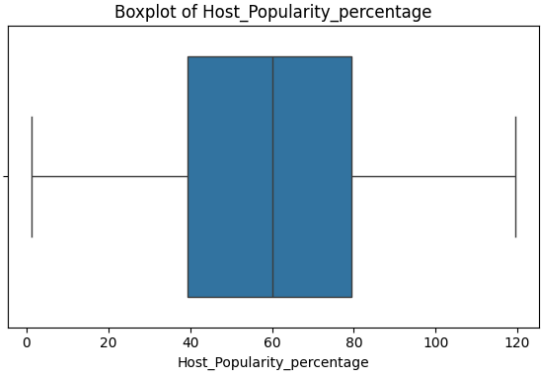
```
plt.title("Distribution of Listening Time")
plt.xlabel("Listening_Time_minutes")
plt.show()


plt.figure(figsize=(8, 2))
sns.boxplot(x=train_df['Listening_Time_minutes'])
plt.title("Boxplot of Listening Time")
plt.show()
```

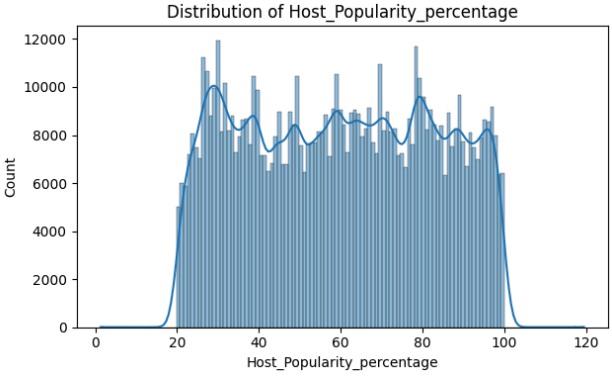


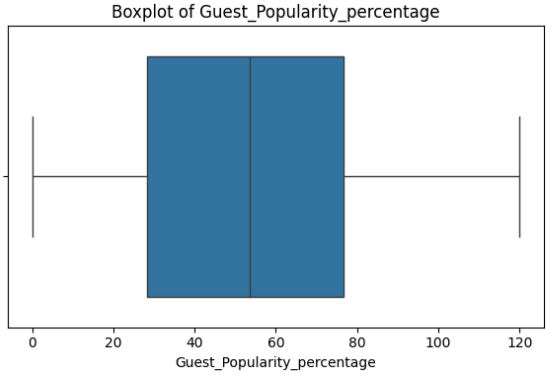

```
import matplotlib.pyplot as plt
import seaborn as sns

# List of numerical columns (adjust as needed)
numerical_cols = [
    'Episode_Length_minutes',
    'Host_Popularity_percentage',
    'Guest_Popularity_percentage',
    'Number_of_Ads'
]

# Loop through and generate boxplot + histplot for each
for col in numerical_cols:
    plt.figure(figsize=(12, 4))

    # Boxplot
    plt.subplot(1, 2, 1)
    sns.boxplot(x=train_df[col])
    plt.title(f'Boxplot of {col}')
    if col == 'Number_of_Ads':
        plt.ylim(0, 5)

    # Histogram
    plt.subplot(1, 2, 2)
    sns.histplot(train_df[col], kde=True)
    plt.title(f'Distribution of {col}')
    if col == 'Number_of_Ads':
        plt.ylim(0, 5)

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

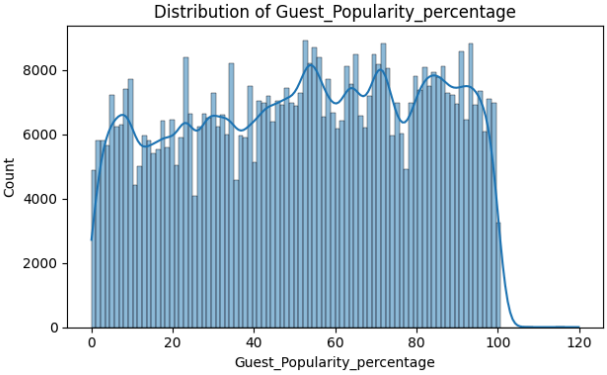


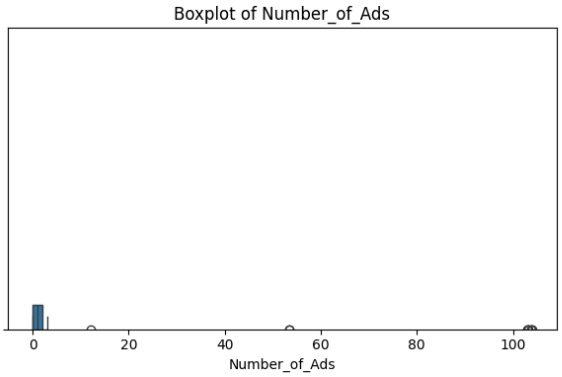


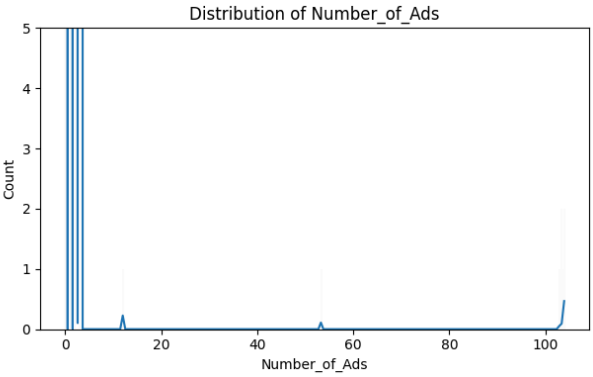




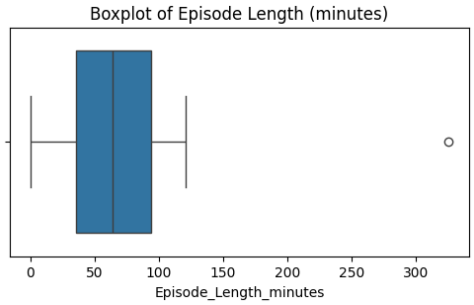


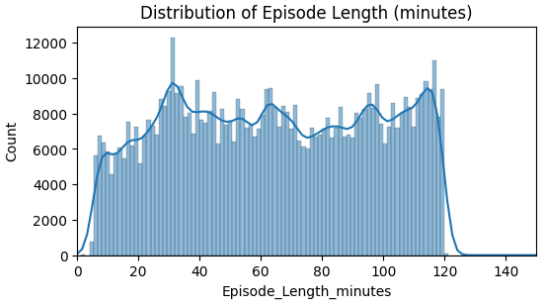




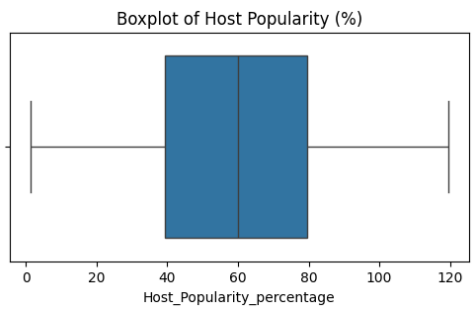


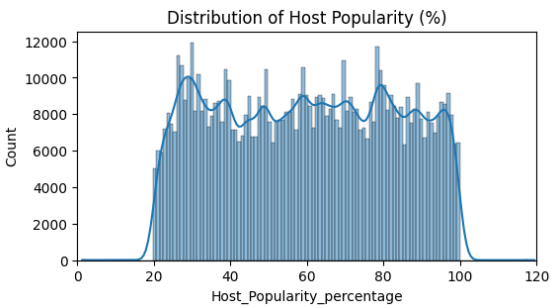

```
# Boxplot
plt.figure(figsize=(6, 3))
sns.boxplot(x=train_df['Episode_Length_minutes'])
plt.title('Boxplot of Episode Length (minutes)')
plt.show()

# Histogram
plt.figure(figsize=(6, 3))
sns.histplot(train_df['Episode_Length_minutes'], kde=True)
plt.title('Distribution of Episode Length (minutes)')
plt.xlim(0, 150)
plt.show()
```



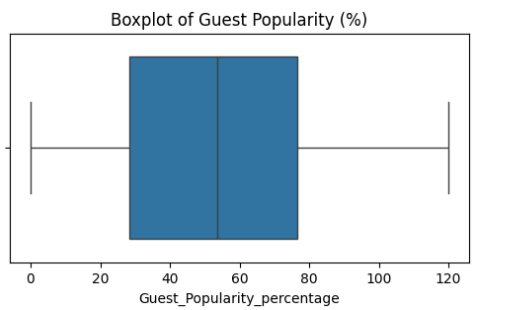


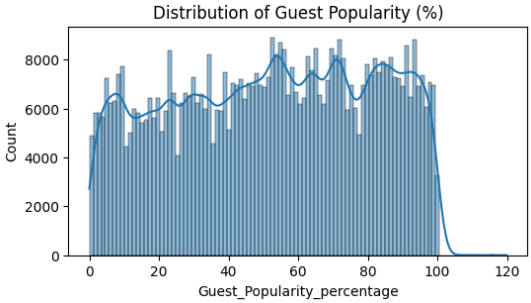

```
# Boxplot
plt.figure(figsize=(6, 3))
sns.boxplot(x=train_df['Host_Popularity_percentage'])
plt.title('Boxplot of Host Popularity (%)')
plt.show()

# Histogram
plt.figure(figsize=(6, 3))
sns.histplot(train_df['Host_Popularity_percentage'], kde=True)
plt.title('Distribution of Host Popularity (%)')
plt.xlim(0, 120)
plt.show()
```



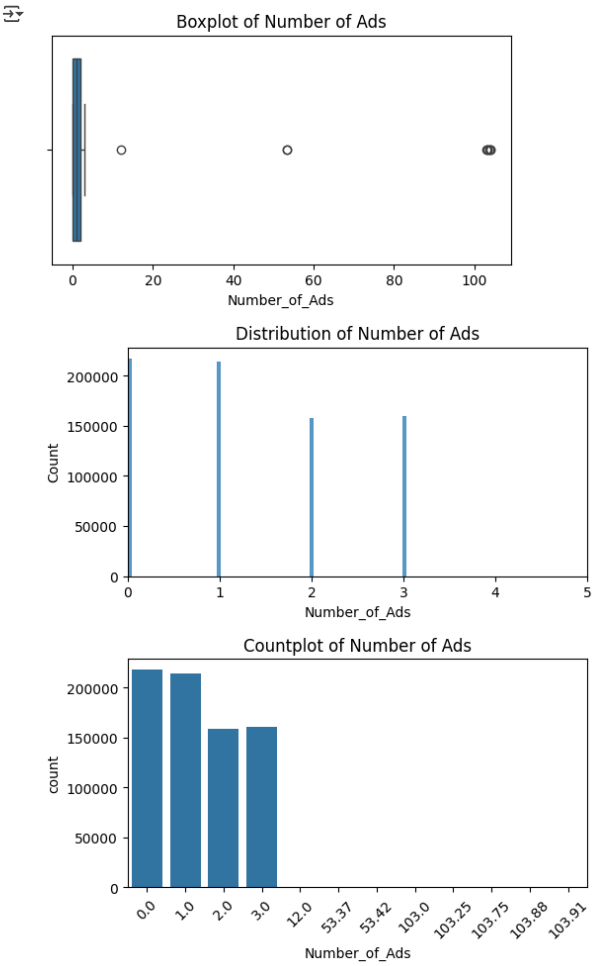



```
plt.figure(figsize=(6, 3))
sns.boxplot(x=train_df['Guest_Popularity_percentage'])
plt.title('Boxplot of Guest Popularity (%)')
plt.show()

# Histogram
plt.figure(figsize=(6, 3))
sns.histplot(train_df['Guest_Popularity_percentage'], kde=True)
plt.title('Distribution of Guest Popularity (%)')
plt.show()
```

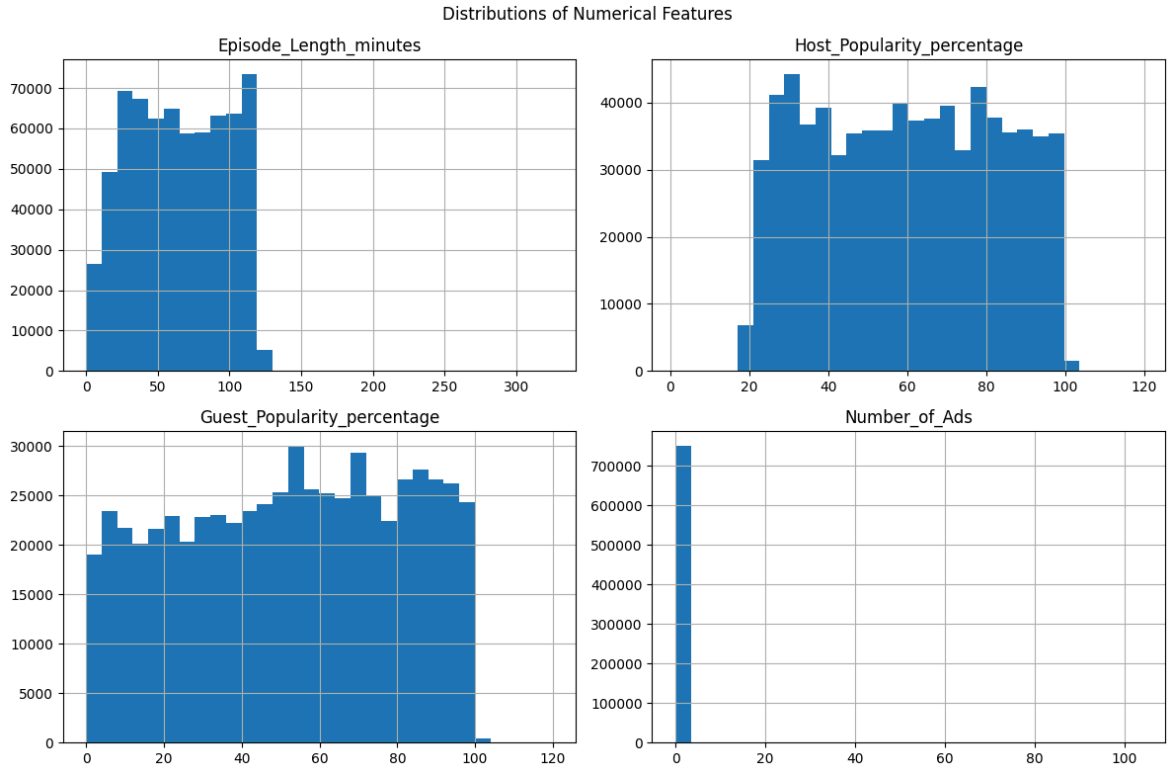




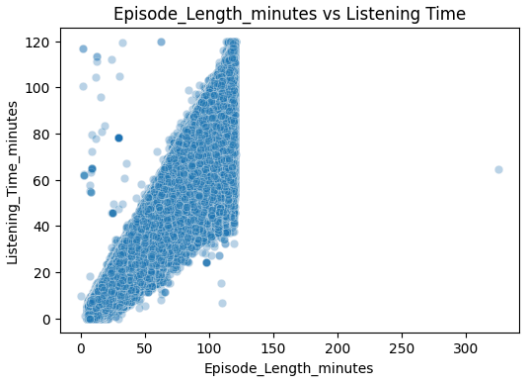

```
# Boxplot
plt.figure(figsize=(6, 3))
sns.boxplot(x=train_df['Number_of_Ads'])
plt.title('Boxplot of Number of Ads')

plt.show()

# Histogram
plt.figure(figsize=(6, 3))
sns.histplot(train_df['Number_of_Ads'], kde=False)
plt.title('Distribution of Number of Ads')
plt.xlim(0, 5)
plt.show()

#countplot
plt.figure(figsize=(6, 3))
sns.countplot(x='Number_of_Ads', data=train_df)
plt.xticks(rotation=45)
plt.title('Countplot of Number of Ads')

plt.show()
```

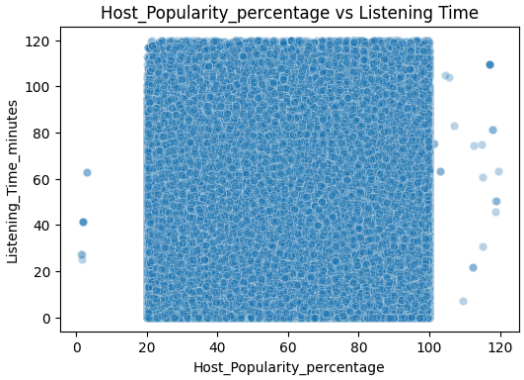


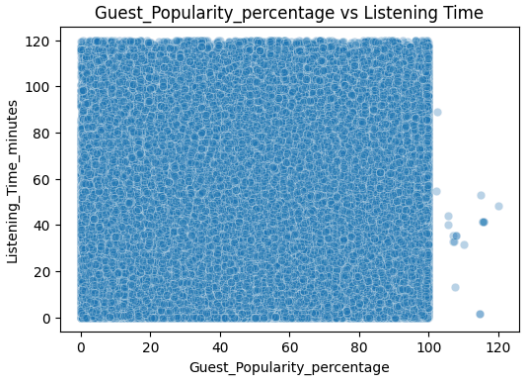

```
numerical_cols = ['Episode_Length_minutes', 'Host_Popularity_percentage',
                  'Guest_Popularity_percentage', 'Number_of_Ads']


train_df[numerical_cols].hist(bins=30, figsize=(12, 8), layout=(2, 2))
plt.suptitle("Distributions of Numerical Features")
plt.tight_layout()
plt.show()
```

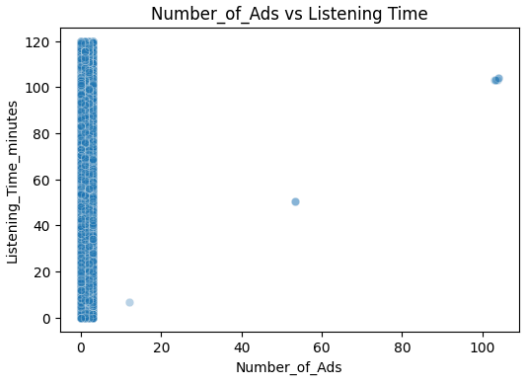



```
#scatterplot
num_cols = ['Episode_Length_minutes', 'Host_Popularity_percentage',
            'Guest_Popularity_percentage', 'Number_of_Ads']

for col in num_cols:
    plt.figure(figsize=(6, 4))
    sns.scatterplot(data=train_df, x=col, y='Listening_Time_minutes', alpha=0.3)
    plt.title(f"{col} vs Listening Time")
    plt.xlabel(col)
    plt.ylabel("Listening_Time_minutes")
    plt.show()
```









Categorical Feature

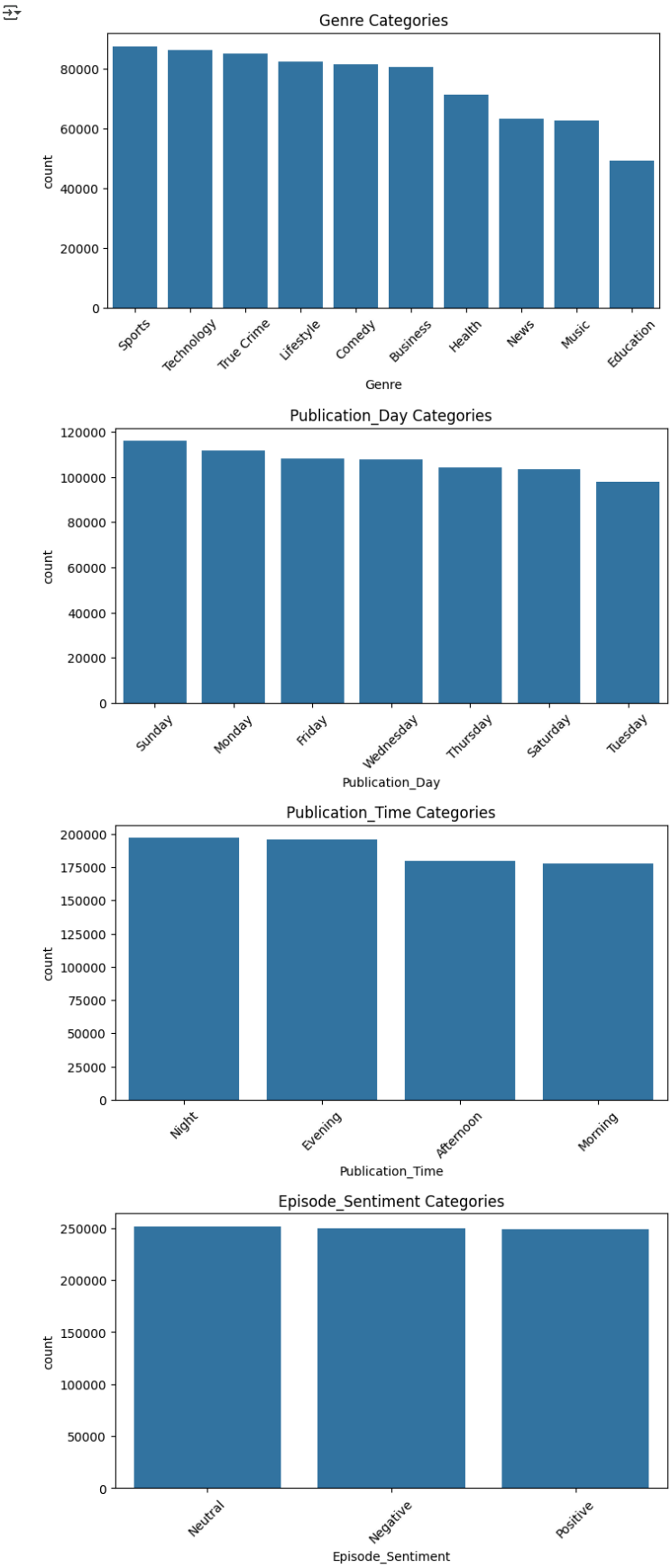
```
print(f"Episode_Title unique values: {train_df['Episode_Title'].nunique()}")
print(f"Podcast_Name unique values: {train_df['Podcast_Name'].nunique()}")
print(f"Number of unique genres: {train_df['Genre'].nunique()}")
```

```
Episode_Title unique values: 100
Podcast_Name unique values: 48
Number of unique genres: 10
```

```
categorical_cols = ['Genre',
                    'Publication_Day', 'Publication_Time', 'Episode_Sentiment']
```

```
for col in categorical_cols:
    plt.figure(figsize=(8, 4))
    sns.countplot(data=train_df, x=col, order=train_df[col].value_counts().iloc[:10].index)
    plt.title(f"{col} Categories")
```

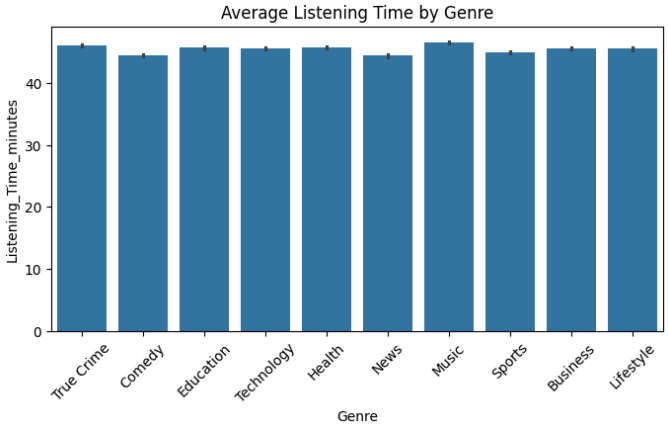
```
    plt.xticks(rotation=45)
    plt.show()
```

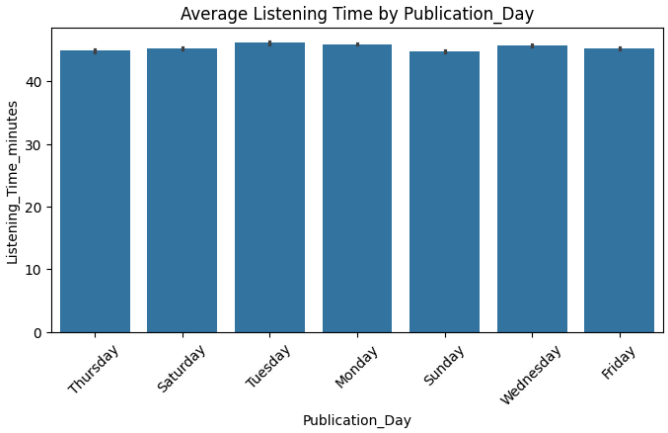


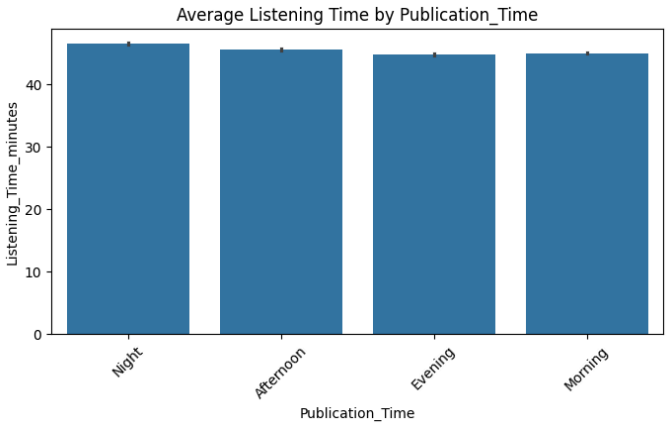

```
#mean listening time by each categorical variable
for col in ['Genre', 'Publication_Day', 'Publication_Time', 'Episode_Sentiment']:
    plt.figure(figsize=(8, 4))
    sns.barplot(data=train_df, x=col, y='Listening_Time_minutes', estimator='mean')
    plt.title(f"Average Listening Time by {col}")
    plt.xticks(rotation=45)
```

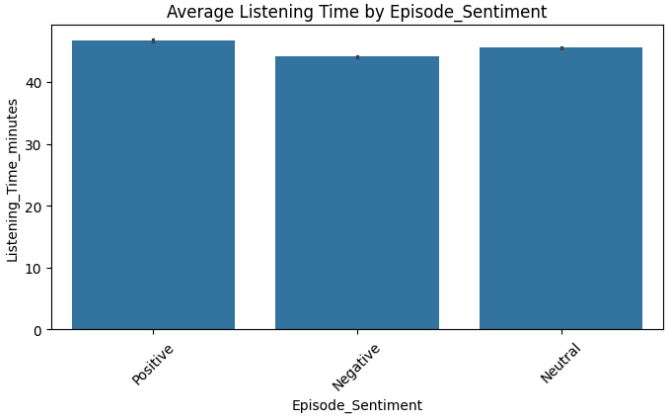
```
    plt.ylabel("Listening_Time_minutes")
    plt.xlabel(col)
    plt.show()
```

[+]





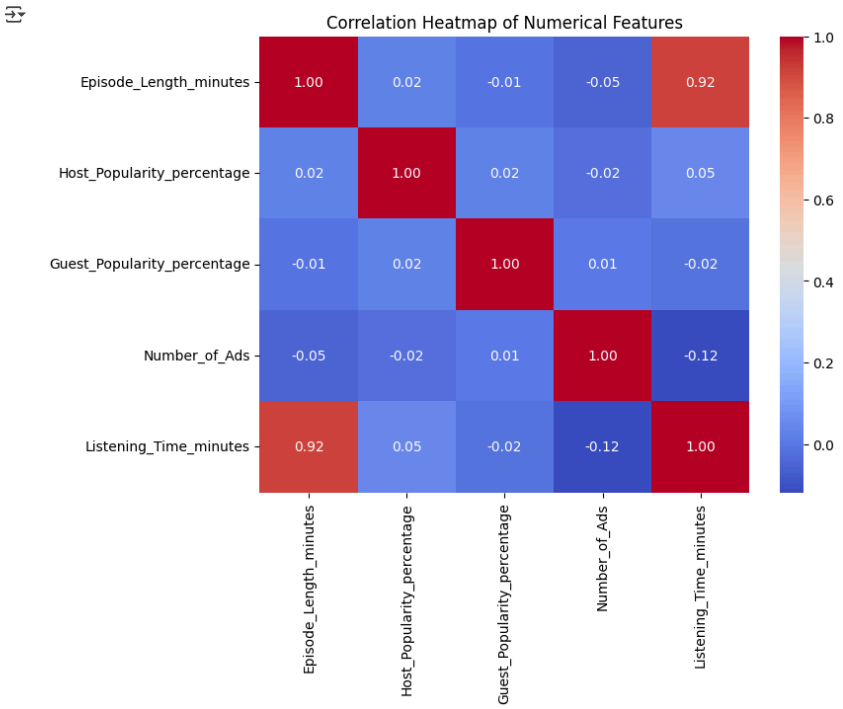




Multicollinearity using Corrleation Heatmap

```
#inspect multicollinearily between numerical features to inform preprocessing
```

```
num_cols = ['Episode_Length_minutes', 'Host_Popularity_percentage',
            'Guest_Popularity_percentage', 'Number_of_Ads', 'Listening_Time_minutes']

plt.figure(figsize=(8, 6))
corr_matrix = train_df[num_cols].corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Correlation Heatmap of Numerical Features")
plt.show()
```



Preprocessing

Training Data Preparation

```
#Preprocess Training Data


train_df_cleaned = train_df.copy()

#missing value imputation
train_df_cleaned['Episode_Length_minutes'] = train_df_cleaned['Episode_Length_minutes'].fillna(train_df_cleaned['Episode_Length_minutes'].median())
train_df_cleaned['Guest_Popularity_percentage'] = train_df_cleaned['Guest_Popularity_percentage'].fillna(train_df_cleaned['Guest_Popularity_percentage'].median())
train_df_cleaned['Number_of_Ads'] = train_df_cleaned['Number_of_Ads'].fillna(train_df_cleaned['Number_of_Ads'].mode()[0])

#clip outlier
train_df_cleaned['Episode_Length_minutes'] = np.clip(train_df_cleaned['Episode_Length_minutes'], None, 120)
train_df_cleaned['Host_Popularity_percentage'] = np.clip(train_df_cleaned['Host_Popularity_percentage'], 0, 100)
train_df_cleaned['Guest_Popularity_percentage'] = np.clip(train_df_cleaned['Guest_Popularity_percentage'], 0, 100)
train_df_cleaned['Number_of_Ads'] = np.clip(train_df_cleaned['Number_of_Ads'], None, 3)

#correct overflow in target
train_df_cleaned['Listening_Time_minutes'] = np.where(
    train_df_cleaned['Listening_Time_minutes'] > train_df_cleaned['Episode_Length_minutes'],
    train_df_cleaned['Episode_Length_minutes'],
    train_df_cleaned['Listening_Time_minutes']
)


#Deal with Episode_Title/Podcast_Name
train_df_cleaned['Episode_Title_Numeric'] = train_df_cleaned['Episode_Title'].str.extract(r'(\d+)').astype(int)
train_df_cleaned.drop(columns=['Episode_Title', 'Podcast_Name'], inplace=True)



train_df_cleaned.info()
train_df_cleaned.describe()
train_df_cleaned.isnull().sum()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 750000 entries, 0 to 749999
Data columns (total 11 columns):
 #   Column                       Non-Null Count   Dtype
---  ------                       --------------   -----
 0   id                           750000 non-null  int64
 1   Episode_Length_minutes       750000 non-null  float64
 2   Genre                        750000 non-null  object
 3   Host_Popularity_percentage   750000 non-null  float64
 4   Publication_Day              750000 non-null  object
 5   Publication_Time             750000 non-null  object
 6   Guest_Popularity_percentage  750000 non-null  float64
 7   Number_of_Ads                750000 non-null  float64
 8   Episode_Sentiment            750000 non-null  object
 9   Listening_Time_minutes       750000 non-null  float64
 10  Episode_Title_Numeric        750000 non-null  int64
dtypes: float64(5), int64(2), object(4)
memory usage: 62.9+ MB
```

	0
id	0
Episode_Length_minutes	0
Genre	0
Host_Popularity_percentage	0
Publication_Day	0
Publication_Time	0
Guest_Popularity_percentage	0
Number_of_Ads	0
Episode_Sentiment	0
Listening_Time_minutes	0
Episode_Title_Numeric	0

dtype: int64

```
#encoding block
sentiment_map = {'Negative': 0, 'Neutral': 1, 'Positive': 2}
train_df_cleaned['Episode_Sentiment'] = train_df_cleaned['Episode_Sentiment'].map(sentiment_map)

train_df_cleaned = pd.get_dummies(
    train_df_cleaned,
    columns=['Genre', 'Publication_Day', 'Publication_Time'],
    drop_first=True
)

train_df_cleaned.head()
```

	id	Episode_Length_minutes	Host_Popularity_percentage	Guest_Popularity_percentage	Number_of_Ads	Episode_Sentiment	Listening_Time_minutes	Episode_Title_Numeric	Genre_Comedy	Genr
0	0	63.84	74.81	53.58	0.0	2	31.41998	98	False	
1	1	119.80	66.95	75.95	2.0	0	88.01241	26	True	
2	2	73.90	69.97	8.97	0.0	0	44.92531	16	False	
3	3	67.17	57.22	78.70	2.0	2	46.27824	45	False	
4	4	110.51	80.07	58.68	3.0	1	75.61031	86	False	

5 rows × 26 columns

```
bool_cols = train_df_cleaned.select_dtypes(include='bool').columns
train_df_cleaned[bool_cols] = train_df_cleaned[bool_cols].astype(int)


train_df_cleaned.info()
train_df_cleaned.describe()
train_df_cleaned.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 750000 entries, 0 to 749999
Data columns (total 26 columns):
 #   Column                       Non-Null Count   Dtype  
---  ------                       --------------   -----  
 0   id                           750000 non-null  int64  
 1   Episode_Length_minutes       750000 non-null  float64
 2   Host_Popularity_percentage   750000 non-null  float64
 3   Guest_Popularity_percentage  750000 non-null  float64
 4   Number_of_Ads                750000 non-null  float64
 5   Episode_Sentiment            750000 non-null  int64  
 6   Listening_Time_minutes       750000 non-null  float64
 7   Episode_Title_Numeric        750000 non-null  int64  
 8   Genre_Comedy                 750000 non-null  int64  
 9   Genre_Education              750000 non-null  int64  
 10  Genre_Health                 750000 non-null  int64  
 11  Genre_Lifestyle              750000 non-null  int64  
 12  Genre_Music                  750000 non-null  int64  
 13  Genre_News                   750000 non-null  int64  
 14  Genre_Sports                 750000 non-null  int64  
 15  Genre_Technology             750000 non-null  int64  
 16  Genre_True Crime             750000 non-null  int64  
 17  Publication_Day_Monday       750000 non-null  int64  
 18  Publication_Day_Saturday     750000 non-null  int64  
 19  Publication_Day_Sunday       750000 non-null  int64  
 20  Publication_Day_Thursday     750000 non-null  int64  
 21  Publication_Day_Tuesday      750000 non-null  int64  
 22  Publication_Day_Wednesday    750000 non-null  int64  
 23  Publication_Time_Evening     750000 non-null  int64  
 24  Publication_Time_Morning     750000 non-null  int64  
 25  Publication_Time_Night       750000 non-null  int64  
dtypes: float64(5), int64(21)
memory usage: 148.8 MB
```

	id	Episode_Length_minutes	Host_Popularity_percentage	Guest_Popularity_percentage	Number_of_Ads	Episode_Sentiment	Listening_Time_minutes	Episode_Title_Numeric	Genre_Comedy	Genr
0	0	63.84	74.81	53.58	0.0	2	31.41998	98	0	
1	1	119.80	66.95	75.95	2.0	0	88.01241	26	1	
2	2	73.90	69.97	8.97	0.0	0	44.92531	16	0	
3	3	67.17	57.22	78.70	2.0	2	46.27824	45	0	
4	4	110.51	80.07	58.68	3.0	1	75.61031	86	0	

5 rows × 26 columns

Testing Data Preparation

```
test_df_cleaned = test_df.copy()


test_df_cleaned['Episode_Length_minutes'] = test_df_cleaned['Episode_Length_minutes'].fillna(train_df['Episode_Length_minutes'].median())
test_df_cleaned['Guest_Popularity_percentage'] = test_df_cleaned['Guest_Popularity_percentage'].fillna(train_df['Guest_Popularity_percentage'].median())
test_df_cleaned['Number_of_Ads'] = test_df_cleaned['Number_of_Ads'].fillna(train_df['Number_of_Ads'].mode()[0])


test_df_cleaned['Episode_Length_minutes'] = np.clip(test_df_cleaned['Episode_Length_minutes'], None, 120)
test_df_cleaned['Host_Popularity_percentage'] = np.clip(test_df_cleaned['Host_Popularity_percentage'], 0, 100)
test_df_cleaned['Guest_Popularity_percentage'] = np.clip(test_df_cleaned['Guest_Popularity_percentage'], 0, 100)
test_df_cleaned['Number_of_Ads'] = np.clip(test_df_cleaned['Number_of_Ads'], None, 3)


test_df_cleaned['Episode_Title_Numeric'] = test_df_cleaned['Episode_Title'].str.extract(r'(\d+)').astype(int)
test_df_cleaned.drop(columns=['Podcast_Name', 'Episode_Title'], inplace=True)


sentiment_map = {'Negative': 0, 'Neutral': 1, 'Positive': 2}
test_df_cleaned['Episode_Sentiment'] = test_df_cleaned['Episode_Sentiment'].map(sentiment_map)


test_df_cleaned = pd.get_dummies(
    test_df_cleaned,
    columns=['Genre', 'Publication_Day', 'Publication_Time'],
    drop_first=True
)
bool_cols = test_df_cleaned.select_dtypes(include='bool').columns
test_df_cleaned[bool_cols] = test_df_cleaned[bool_cols].astype(int)

#ensure columns match training set
for col in train_df_cleaned.columns:
    if col not in test_df_cleaned.columns and col != 'Listening_Time_minutes':
        test_df_cleaned[col] = 0

extra_cols = set(test_df_cleaned.columns) - set(train_df_cleaned.columns)
test_df_cleaned.drop(columns=extra_cols, inplace=True)

test_df_cleaned = test_df_cleaned[[col for col in train_df_cleaned.columns if col != 'Listening_Time_minutes']]


test_df_cleaned.info()
test_df_cleaned.describe()
test_df_cleaned.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 250000 entries, 0 to 249999
Data columns (total 25 columns):
 #   Column                       Non-Null Count   Dtype  
---  ------                       --------------   -----  
 0   id                           250000 non-null  int64  
 1   Episode_Length_minutes       250000 non-null  float64
 2   Host_Popularity_percentage   250000 non-null  float64
 3   Guest_Popularity_percentage  250000 non-null  float64
 4   Number_of_Ads                250000 non-null  float64
 5   Episode_Sentiment            250000 non-null  int64  
 6   Episode_Title_Numeric        250000 non-null  int64  
 7   Genre_Comedy                 250000 non-null  int64  
 8   Genre_Education              250000 non-null  int64  
 9   Genre_Health                 250000 non-null  int64  
 10  Genre_Lifestyle              250000 non-null  int64  
 11  Genre_Music                  250000 non-null  int64  
 12  Genre_News                   250000 non-null  int64  
 13  Genre_Sports                 250000 non-null  int64  
 14  Genre_Technology             250000 non-null  int64  
 15  Genre_True Crime             250000 non-null  int64  
 16  Publication_Day_Monday       250000 non-null  int64  
 17  Publication_Day_Saturday     250000 non-null  int64  
 18  Publication_Day_Sunday       250000 non-null  int64  
 19  Publication_Day_Thursday     250000 non-null  int64  
 20  Publication_Day_Tuesday      250000 non-null  int64  
 21  Publication_Day_Wednesday    250000 non-null  int64  
 22  Publication_Time_Evening     250000 non-null  int64  
 23  Publication_Time_Morning     250000 non-null  int64  
 24  Publication_Time_Night       250000 non-null  int64  
dtypes: float64(4), int64(21)
memory usage: 47.7 MB
```

	id	Episode_Length_minutes	Host_Popularity_percentage	Guest_Popularity_percentage	Number_of_Ads	Episode_Sentiment	Episode_Title_Numeric	Genre_Comedy	Genre_Education	Genre_H
0	750000	78.96	38.11	53.33	1.0	1	73	0	1	
1	750001	27.87	71.29	53.58	0.0	1	23	0	0	
2	750002	69.10	67.89	97.51	0.0	2	11	1	0	
3	750003	115.39	23.40	51.75	2.0	2	73	1	0	
4	750004	72.32	58.10	11.30	2.0	1	50	0	0	

5 rows × 25 columns

Modeling

```
# Separate features and target
X = train_df_cleaned.drop(columns=['id', 'Listening_Time_minutes']).values
Y = train_df_cleaned[['Listening_Time_minutes']].values

X.shape, Y.shape
```

```
((750000, 24), (750000, 1))
```

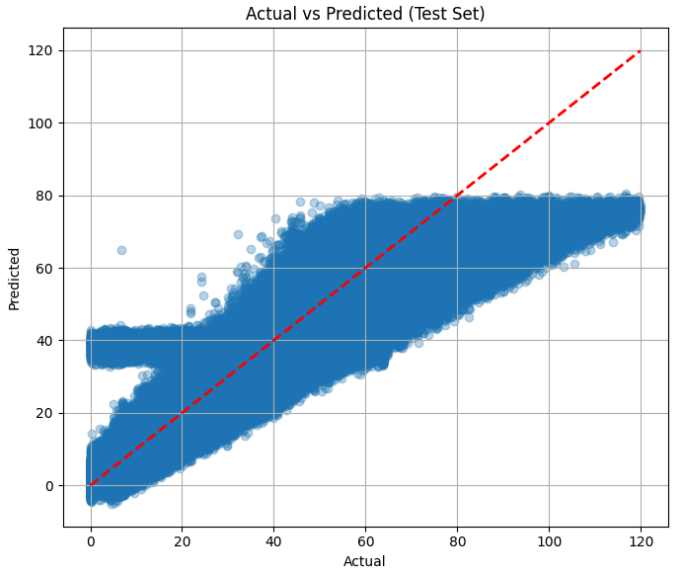
```
class LinearRegression:
    def __init__(self, learning_rate=0.001, num_iterations=500):
        self.learning_rate = learning_rate
        self.num_iterations = num_iterations
        self.cost_history = []
        self.mean = None
        self.std = None

    def normalize(self, X, is_training=True):
        if is_training:
            self.mean = np.mean(X, axis=0)
            self.std = np.std(X, axis=0) + 1e-7
        return (X - self.mean) / self.std

    def predict(self, X):
        num_examples = X.shape[0]
        X_transform = np.append(np.ones((num_examples, 1)), X, axis=1)
        X_normalized = self.normalize(X_transform[:, 1:], is_training=False)
        X_normalized = np.insert(X_normalized, 0, 1, axis=1)
        prediction = X_normalized.dot(self.W)
        return prediction

    def update_weights(self):
        num_examples = self.X.shape[0]
        X_transform = np.append(np.ones((num_examples, 1)), self.X, axis=1)
        X_normalized = self.normalize(X_transform[:, 1:], is_training=False)
        X_normalized = np.insert(X_normalized, 0, 1, axis=1)
        Y_pred = self.predict(self.X)
        dW = - (2 * X_normalized.T.dot(self.Y - Y_pred)) / num_examples
        cost = np.sqrt(np.mean(np.square(self.Y - Y_pred)))
        self.cost_history.append(cost)
        self.W = self.W - self.learning_rate * dW

    def fit(self, X, Y):
        self.X = X
        self.Y = Y
        _, num_features = X.shape
        self.W = np.zeros((num_features + 1, 1))

        _ = self.normalize(X, is_training=True)

        for _ in range(self.num_iterations):
            self.update_weights()
        return self
```

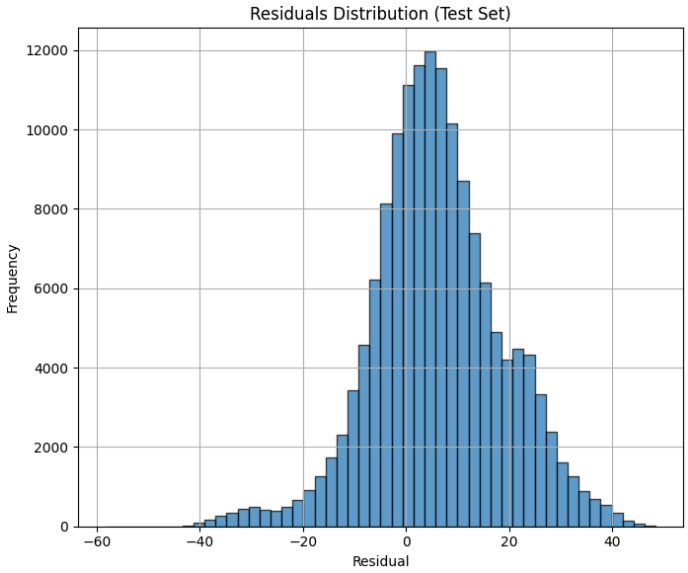
```
def r2_score(y_true, y_pred):
    ss_total = np.sum((y_true - np.mean(y_true)) ** 2)
    ss_residual = np.sum((y_true - y_pred) ** 2)
    return 1 - ss_residual / ss_total

def mean_squared_error(y_true, y_pred):
    return np.mean((y_true - y_pred) ** 2)

def root_mean_squared_error(y_true, y_pred):
    return np.sqrt(mean_squared_error(y_true, y_pred))

def mean_absolute_error(y_true, y_pred):
    return np.mean(np.abs(y_true - y_pred))

def max_error(y_true, y_pred):
    return np.max(np.abs(y_true - y_pred))

def mean_error(y_true, y_pred):
    return np.mean(y_true - y_pred)

def evaluate_model(model, X_train, y_train, X_eval, y_eval):
    y_train_pred = model.predict(X_train)
    y_eval_pred = model.predict(X_eval)

    def report(y_true, y_pred, label):
        print(f"{label} SET:")
        print(f"  MSE         = {mean_squared_error(y_true, y_pred):.4f}")
        print(f"  RMSE        = {root_mean_squared_error(y_true, y_pred):.4f}")
        print(f"  MAE         = {mean_absolute_error(y_true, y_pred):.4f}")
        print(f"  R²          = {r2_score(y_true, y_pred):.4f}")
        print(f"  Max Error   = {max_error(y_true, y_pred):.4f}")
        print(f"  Mean Error  = {mean_error(y_true, y_pred):.4f}")
        print()

    report(y_train, y_train_pred, "TRAIN")
    report(y_eval, y_eval_pred, "TEST")

    import matplotlib.pyplot as plt
    fig, axs = plt.subplots(1, 2, figsize=(14, 6))

    axs[0].scatter(y_eval, y_eval_pred, alpha=0.3)
    axs[0].plot([y_eval.min(), y_eval.max()], [y_eval.min(), y_eval.max()], 'r--', lw=2)
    axs[0].set_title('Actual vs Predicted (Test Set)')
    axs[0].set_xlabel('Actual')
    axs[0].set_ylabel('Predicted')
    axs[0].grid(True)

    residuals = y_eval - y_eval_pred
    axs[1].hist(residuals, bins=50, edgecolor='k', alpha=0.7)
    axs[1].set_title('Residuals Distribution (Test Set)')
    axs[1].set_xlabel('Residual')
    axs[1].set_ylabel('Frequency')
    axs[1].grid(True)

    plt.tight_layout()
    plt.show()


df_filtered = train_df_cleaned.drop(columns=['id', 'Episode_Title_Numeric'])

np.random.seed(50)
shuffled_indices = np.random.permutation(len(df_filtered))
split_index = int(len(df_filtered) * 0.8)
train_indices = shuffled_indices[:split_index]
eval_indices = shuffled_indices[split_index:]

feature_columns = [col for col in df_filtered.columns if col != 'Listening_Time_minutes']
X_all = df_filtered[feature_columns].values
Y_all = df_filtered[['Listening_Time_minutes']].values

X_train = X_all[train_indices]
y_train = Y_all[train_indices]
X_eval = X_all[eval_indices]
y_eval = Y_all[eval_indices]



print(X_train.shape, y_train.shape)
print(X_eval.shape, y_eval.shape)
```

```
(600000, 23) (600000, 1)
(150000, 23) (150000, 1)
```

```
linear_model = LinearRegression(learning_rate=0.01, num_iterations=100)
linear_model.fit(X_train, y_train)

evaluate_model(linear_model, X_train, y_train, X_eval, y_eval)
```

```
TRAIN SET:
  MSE          = 195.8690
  RMSE         = 13.9953
  MAE          = 10.7362
  R²           = 0.7239
  Max Error    = 51.8909
  Mean Error   = 5.9671

TEST SET:
  MSE          = 195.7126
  RMSE         = 13.9897
  MAE          = 10.7533
  R²           = 0.7225
  Max Error    = 58.2576
  Mean Error   = 6.0187
```






```
feature_names = feature_columns

coefs = sk_model.coef_.flatten()
intercept = sk_model.intercept_.item() if hasattr(sk_model.intercept_, 'item') else sk_model.intercept_

weights = pd.DataFrame({
    'Feature': feature_names,
    'Coefficient': coefs
})
weights['Abs_Coefficient'] = weights['Coefficient'].abs()
weights = weights.sort_values(by='Abs_Coefficient', ascending=False)

print(f"Intercept: {intercept:.4f}")
weights.head(40)
```