Predicting Audience Ratings: A Comprehensive Machine Learning Workflow

You can find the code and the repository for this project on GitHub.

In this notebook, we explore an end-to-end machine learning pipeline for predicting audience ratings of movies based on various features. The notebook follows these key steps:

1. **Data Preprocessing**:
    - We handle missing values in both categorical and numerical features, ensuring the dataset is ready for modeling
2. **Feature Engineering**:
    - To enhance the predictive power of the models, we perform feature engineering by generating vectorized representations for text-based columns and breaking down multi-valued categorical columns into binary indicator columns.
3. **Model Selection and Training**:
    - A diverse set of regression models are trained, including Random Forest, Gradient Boosting, AdaBoost, Linear Regression, and more. Each model is evaluated using standard metrics such as Mean Squared Error (MSE), R-squared (R2), and Mean Absolute Error (MAE).
4. **Model Evaluation**:
    - The models are evaluated to determine which provides the best balance between bias and variance. Cross-validation is employed to validate the performance of the best model.
    - The model performance is visualized to facilitate easy comparison and interpretation.
5. **Feature Importance Analysis**:
    - For tree-based models, we calculate and visualize the importance of features to understand which variables contribute most significantly to the model's predictions.
6. **Results and Conclusion**:
    - The best model is selected based on its R2 score, providing the most accurate predictions for audience ratings. Insights into model performance are summarized, offering a clear understanding of the most influential features in the dataset.

This notebook serves as a practical guide to building and evaluating regression models in a real-world setting.

Importing the Dependencies

```
import pandas as pd
import numpy as np
```

```python
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn import metrics
from sklearn import preprocessing
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from scipy.stats import iqr
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
from sklearn.utils.class_weight import compute_class_weight
from xgboost import XGBClassifier
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer, KNNImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder,
PolynomialFeatures
```

Loading Dataset

```python
# Load the Rotten Tomatoes Movie Dataset from a CSV file into a pandas
DataFrame
# Replace the file path with your desired location if needed
audience_data = pd.read_csv('/content/Rotten_Tomatoes_Movies3.csv')
```

Exploratory Data Analysis

Data Preprocessing

```python
# Displaying the first five rows of the DataFrame
# This provides a quick glimpse of the dataset, including column names
and sample data
audience_data.head()
```


```
{"summary":"{\n  \"name\": \"audience_data\",\n  \"rows\": 16638,\n
\"fields\": [\n    {\n      \"column\": \"movie_title\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 16106,\n        \"samples\": [\n
\"White Dog\",\n          \"Out to Sea\",\n          \"The Social
Network\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
```

```
\"movie_info\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 16613,\n
\"samples\": [\n          \"The Wild West meets the Far East in a
battle for honor, royalty and a trunk full of gold when acrobatic
Imperial Guard Chong Wang comes to America to rescue a beautiful
kidnapped Chinese princess. With the help of a partner he doesn't
trust, a wife he doesn't want, a horse he cannot ride and martial arts
moves that no one can believe, Chan finds himself facing the meanest
gunslingers in the West.\",\n          \"Yusef is a first-generation
Pakistani-American engineering student who moves off-campus with a
group of Muslim punks in Buffalo, New York. His new \\\"un-
orthodox\\\" housemates soon introduce him to Taqwacore - a hardcore,
Muslim punk rock scene. As the seasons change, Taqwacore influences
the house more and more. The living room becomes a mosque during the
day, while it continues to host punk shows at night. Ultimately, Yusef
begins to challenge his own faith and ideologies. A powerful and
original story of punk Islam in the USA and the discovery of oneself
within the confines of religion. -- (C) Strand\",\n          \"Filthy.
Rich. Spoiled. Rotten. A band of overprivileged rich boys run wild in
this savagely funny satire of money, sex and power. In the elite realm
of Oxford University, no society is more exclusive than The Riot Club,
the ultra-selective fraternity for Britain's most privileged sons.
When he's recruited to join, down-to-earth first-year student Miles
(Max Irons) is at first amused-but he's about to get a taste of upper-
crust entitlement at its ugliest when a hedonistic night of drinking
and drugs spins out of control. The Hunger Games' Sam Claflin co-stars
in this deliciously dark look at boys behaving badly from the
Oscar(R)-nominated director of An Education. (C) IFC\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"critics_consensus\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 8307,\n        \"samples\": [\n          \"Lost
and Delirious becomes exactly that, as the film sinks into overwrought
melodrama and cliched, obvious symbolism.\",\n          \"Churchill
gets sterling work out of Brian Cox in the leading role, but it isn't
enough to overcome a muddled and ultimately unsuccessful approach to
an incredible real-life story.\",\n          \"The Last Face's noble
intentions are nowhere near enough to carry a fundamentally misguided
story that arguably demeans the demographic it wants to defend.\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"rating\",\n      \"properties\":
{\n        \"dtype\": \"category\",\n        \"num_unique_values\":
6,\n        \"samples\": [\n          \"PG\",\n          \"R\",\n
\"NC17\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"genre\",\n      \"properties\": {\n        \"dtype\": \"category\",\
n        \"num_unique_values\": 1080,\n        \"samples\": [\n
\"Drama, Kids & Family, Musical & Performing Arts\",\n
\"Drama, Western, Romance\",\n          \"Art House & International,
```

```
Comedy, Drama, Musical & Performing Arts, Science Fiction & Fantasy\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"directors\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 8314,\n
\"samples\": [\n          \"David Caesar\",\n          \"Mikkel Br\\
u221a\\u00b6nne Sandemose\",\n          \"K.C. Bascombe\"\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"writers\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 12121,\n        \"samples\": [\n
\"Peter Tolan\",\n          \"Pedro Gonz\\u221a\\u00b0lez-Rubio\",\n
\"Fran\\u221a\\u00dfois Truffaut, Jean-Louis Richard, Helen Scott,
David Rudkin\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"cast\",\n      \"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 16326,\n        \"samples\": [\n
\"Casper Van Dien, Jane March, Steven Waddington, Winston Ntshona,
Rapulana Seiphemo, Sean Taylor, Gys de Villiers, Russel Savadier, Paul
Buckby, Zane Meas, Barry Berk, Michael Gritten, Dimitri Cassar, Tony
Caprari, Kurt Wurstman, Chris Olley, Joshua Lindberg, Henry van der
Berg, Pete Janschk, Danie van Reinsberg, Aubrey Lovett, Paolo Tocha,
Nickie Grigg, Neville Strydom, Dieter Hoffman, Pierre van Rensburg,
Bismulah Mdaka, Sello Sebotiane, Sello Dlamini, Chester Fukazi, Grant
Swanby, Adam Crousdale, Nick Rujewick, Brendan Stapelton, Amy Pearson,
Jeneane Wyatt-Mair, Cheryl Lang, Flash Trobajane\",\n
\"Jeanne Bell, Robert De Niro, Harvey Keitel, Amy Robinson, David
Proval, Richard Romanus, Cesare Danova, George Memmoli, Julie
Andelman, Lenny Scaletta, Jeannie Bell, Victor Argo, Murray Moston,
David Carradine, Robert Carradine, Jeanie Bell, Lois Walden, Harry
Northrup, Dino Seragusa, D'Mitch Davis, Peter Fain, Juli Andelman,
Robert Wilder, Ken Sinclair, Catherine Scorsese, Martin Scorsese,
Jaime Alba\",\n          \"Billy Crudup, Ezra Miller, Michael
Angarano, Tye Sheridan, Johnny Simmons, Olivia Thirlby, Logan Miller,
Thomas Mann, Keir Gilchrist, Gaius Charles, Ki Hong Lee, James Wolk,
Moises Arias, Jack Kilmer, Chris Sheffield, James Frecheville,
Nicholas Braun, Nelsan Ellis, Matt Bennett, Jesse Carere, Brett
Davern, Miles Heizer, Callan McAuliffe, Benedict Samuel, Harrison
Thomas, Albert Malafronte, Danielle Lauder, Kate Butler, Jim Klock,
Fred Ochs, Alec Holden, Jack Foley, Ross Philips, Aidan Sussman,
Armand Vasquez, Kim Robert Koscki\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"in_theaters_date\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 5586,\n        \"samples\": [\n          \"27-
01-1984\",\n          \"02-11-2001\",\n          \"18-01-2007\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"on_streaming_date\",\n
\"properties\": {\n        \"dtype\": \"object\",\n
```

\"num_unique_values\": 2260,\n   \"samples\": [\n   \"14-
03-1995\",\n   \"27-07-2018\",\n   \"27-08-2013\"\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"runtime_in_minutes\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
25.02801086709842,\n   \"min\": 1.0,\n   \"max\": 2000.0,\n
\"num_unique_values\": 201,\n   \"samples\": [\n   66.0,\n
86.0,\n   91.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"studio_name\",\n   \"properties\": {\n   \"dtype\":
\"category\",\n   \"num_unique_values\": 2886,\n
\"samples\": [\n   \"Three Kings\",\n   \"Original
Cinema\",\n   \"Rockstone Releasing\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"tomatometer_status\",\n
\"properties\": {\n   \"dtype\": \"category\",\n
\"num_unique_values\": 3,\n   \"samples\": [\n
\"Rotten\",\n   \"Certified Fresh\",\n   \"Fresh\"\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"tomatometer_rating\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
28,\n   \"min\": 0,\n   \"max\": 100,\n
\"num_unique_values\": 101,\n   \"samples\": [\n   95,\n
72,\n   12\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"tomatometer_count\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 66,\n   \"min\": 5,\n
\"max\": 497,\n   \"num_unique_values\": 393,\n
\"samples\": [\n   65,\n   250,\n   308\
n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"audience_rating\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 20.462367575517256,\n   \"min\":
0.0,\n   \"max\": 100.0,\n   \"num_unique_values\": 98,\n
\"samples\": [\n   44.0,\n   55.0,\n   99.0\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   }\n]\n}","type":"dataframe","variable_name":"audience_data"}


```
#Find Shape of Our Dataset (Number of Rows And Number of Columns)
audience_data.shape
```

```
(16638, 16)
```

```
# Getting Information About Our Dataset Like Total Number Rows, Total
Number of Columns, Datatypes of Each Column And Memory Requirement
audience_data.info()
```

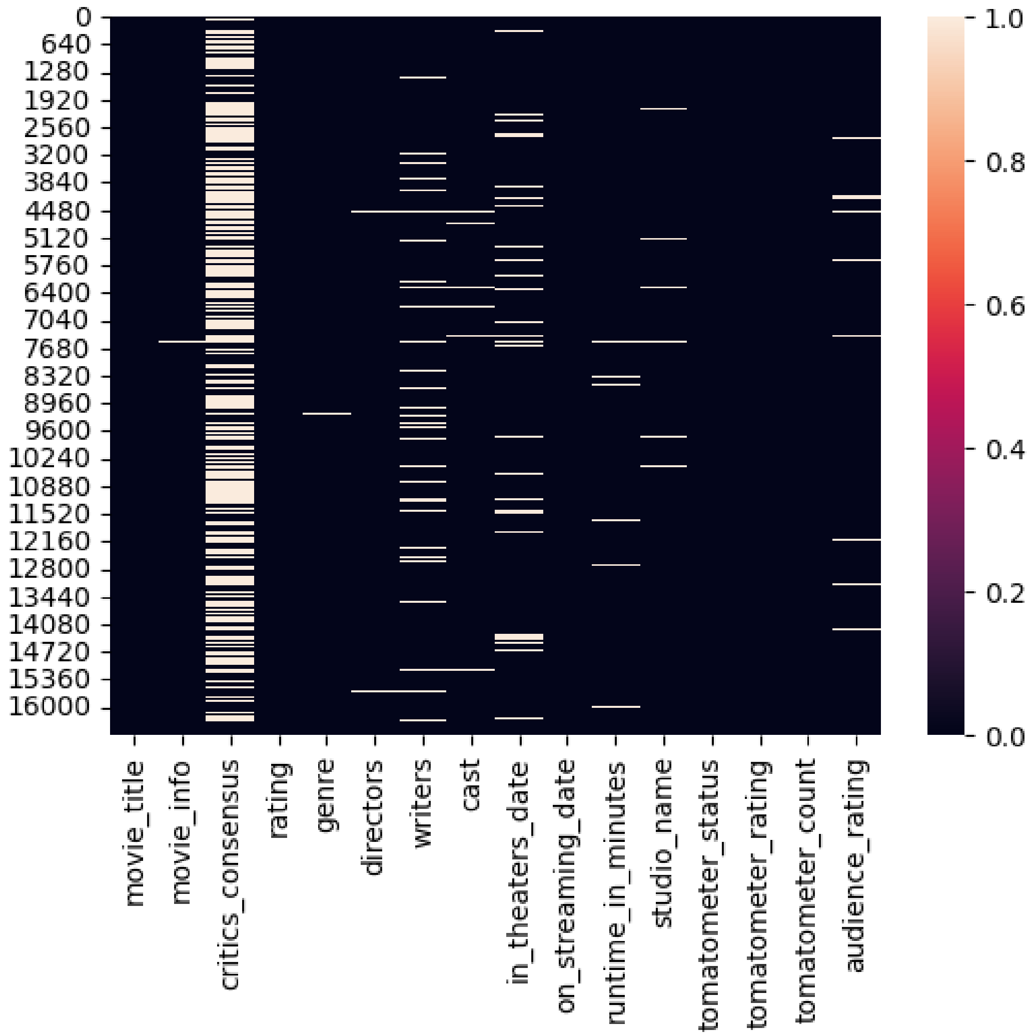
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16638 entries, 0 to 16637
Data columns (total 16 columns):
```

```
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   movie_title           16638 non-null  object
 1   movie_info            16614 non-null  object
 2   critics_consensus     8309 non-null   object
 3   rating                16638 non-null  object
 4   genre                 16621 non-null  object
 5   directors             16524 non-null  object
 6   writers               15289 non-null  object
 7   cast                  16354 non-null  object
 8   in_theaters_date      15823 non-null  object
 9   on_streaming_date     16636 non-null  object
 10  runtime_in_minutes    16483 non-null  float64
 11  studio_name           16222 non-null  object
 12  tomatometer_status    16638 non-null  object
 13  tomatometer_rating    16638 non-null  int64
 14  tomatometer_count     16638 non-null  int64
 15  audience_rating       16386 non-null  float64
dtypes: float64(2), int64(2), object(12)
memory usage: 2.0+ MB
```

```
# checking for null values
audience_data.isnull().sum()
```

```
movie_title              0
movie_info              24
critics_consensus     8329
rating                   0
genre                   17
directors              114
writers               1349
cast                   284
in_theaters_date       815
on_streaming_date        2
runtime_in_minutes     155
studio_name            416
tomatometer_status       0
tomatometer_rating       0
tomatometer_count        0
audience_rating        252
dtype: int64
```

```
# Visualizing missing values in the dataset using a heatmap
# sns.heatmap() creates a visual representation of null values in the
DataFrame
sns.heatmap(audience_data.isnull())

# Displaying the plot
plt.show()
```




```
#Check Proportion of Missing Data
missing_percentage = (audience_data.isnull().sum() /
len(audience_data)) * 100
print(missing_percentage.sort_values(ascending=False))
```

```
critics_consensus     50.060103
writers                8.107946
in_theaters_date       4.898425
studio_name            2.500301
cast                   1.706936
audience_rating        1.514605
runtime_in_minutes     0.931602
directors              0.685179
movie_info             0.144248
genre                  0.102176
on_streaming_date      0.012021
movie_title            0.000000
rating                 0.000000
```

```
tomatometer_status     0.000000
tomatometer_rating     0.000000
tomatometer_count      0.000000
dtype: float64
```

```python
# Calculate median for numerical columns
median_runtime = audience_data['runtime_in_minutes'].median()
median_rating = audience_data['audience_rating'].median()

# Drop columns with too many missing values
audience_data.drop(['critics_consensus'], axis=1, inplace=True)
audience_data['runtime_in_minutes'] =
audience_data['runtime_in_minutes'].fillna(median_runtime)
audience_data['audience_rating'] =
audience_data['audience_rating'].fillna(median_rating)
audience_data=audience_data.dropna(axis=0)

#check for duplicate data
dup_data=audience_data.duplicated().any()
print("Are there any duplicated values in data?",dup_data)
```

```
Are there any duplicated values in data? False
```

```python
#Get Overall Statistics About The DataFrame
audience_data.describe()
```


```
{"summary":"{\n  \"name\": \"audience_data\",\n  \"rows\": 8,\n
\"fields\": [\n    {\n      \"column\": \"runtime_in_minutes\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
4982.853943372883,\n        \"min\": 1.0,\n        \"max\": 14311.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
103.55397945636224,\n          100.0,\n          14311.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"tomatometer_rating\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
5041.097289765921,\n        \"min\": 0.0,\n        \"max\": 14311.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
58.789253022150795,\n          63.0,\n          14311.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"tomatometer_count\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
5023.534871819521,\n        \"min\": 5.0,\n        \"max\": 14311.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
61.82502969743554,\n          32.0,\n          14311.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"audience_rating\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
5041.4491953636025,\n        \"min\": 0.0,\n        \"max\": 14311.0,\
n        \"num_unique_values\": 8,\n        \"samples\": [\n
60.031514219830896,\n          62.0,\n          14311.0\n        ],\n
```

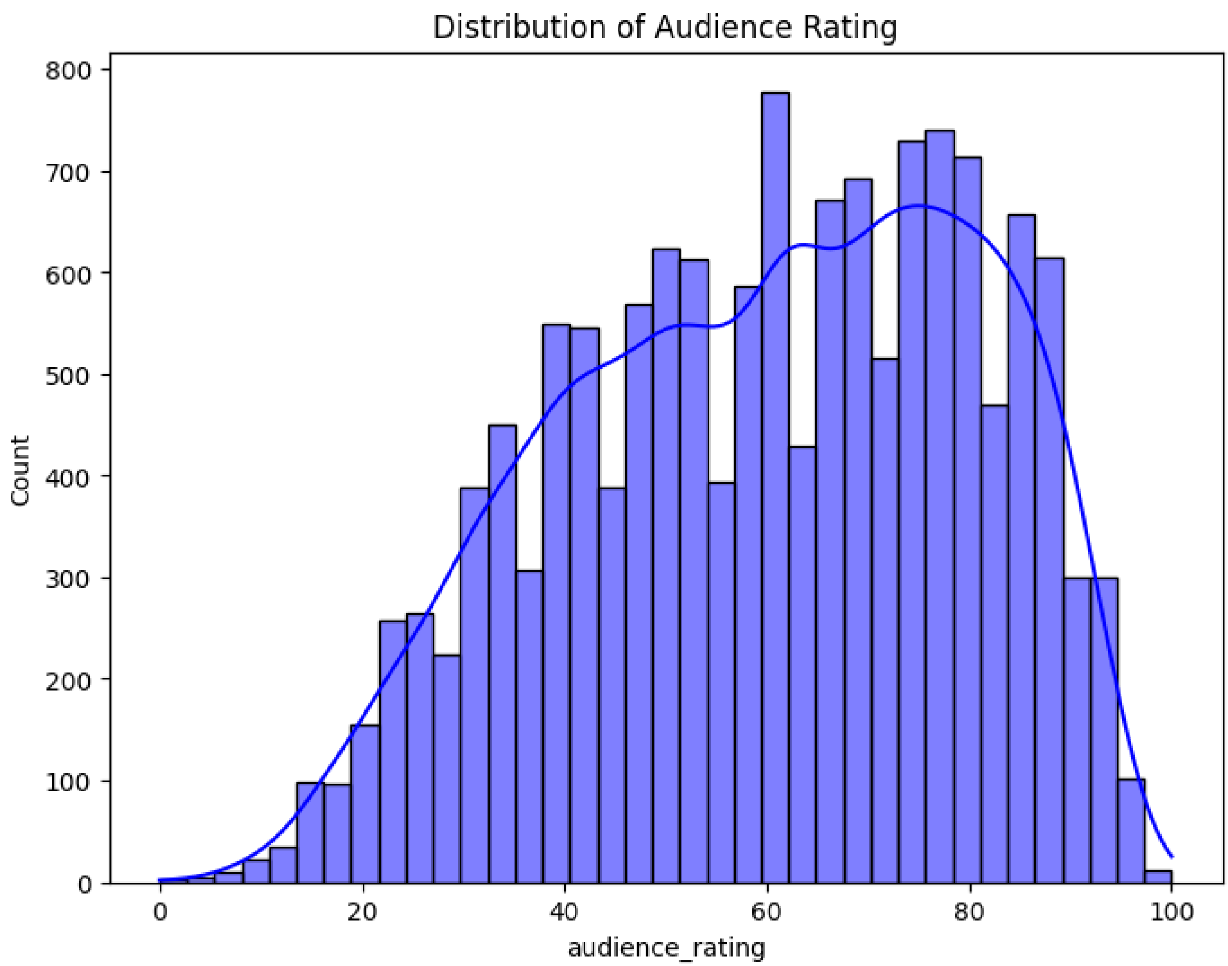
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe"}


```
#Display Title of The Movie Having Runtime >= 300 Minutes
audience_data[audience_data['runtime_in_minutes']>=300]['movie_title']
```

```
913                                   Love on the Run
1844                                 1900 (Novecento)
4104                                           Carlos
10372    Never Sleep Again: The Elm Street Legacy
13539                                    Terror Tract
Name: movie_title, dtype: object
```

Visualizing Audience Rating Distribution

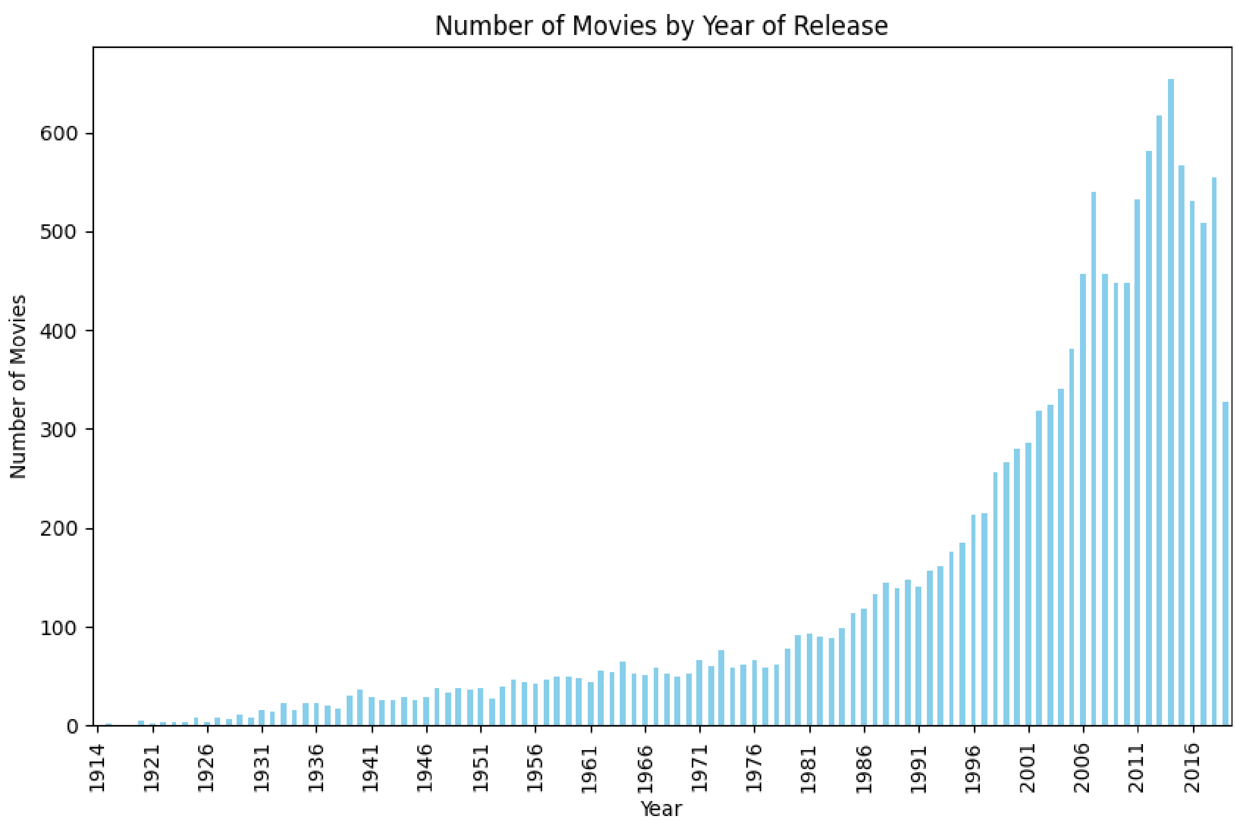
```
# Adjusting the figure size for better visualization
plt.figure(figsize=(8, 6))

# Plotting the distribution of the 'audience_rating' column
# sns.histplot creates a histogram with a Kernel Density Estimate
(KDE) overlay for smoother visualization
sns.histplot(audience_data['audience_rating'], kde=True, color='blue')

# Adding a title to the plot for clarity
plt.title('Distribution of Audience Rating')

# Displaying the plot
plt.show()
```



Visualizing Number of Movies by Year of Release

```
# Convert the 'in_theaters_date' column to datetime format and extract
the release year
# This helps in aggregating data by release year
audience_data['release_year'] =
pd.to_datetime(audience_data['in_theaters_date'], format='%d-%m-
%Y').dt.year

# Creating a histogram to visualize the number of movies released each
year
plt.figure(figsize=(10, 6))

# Counting movies for each release year and sorting by year
release_year_counts =
audience_data['release_year'].value_counts().sort_index()
release_year_counts.plot(kind='bar', color='skyblue')

# Adding plot title and axis labels
plt.title("Number of Movies by Year of Release")
```

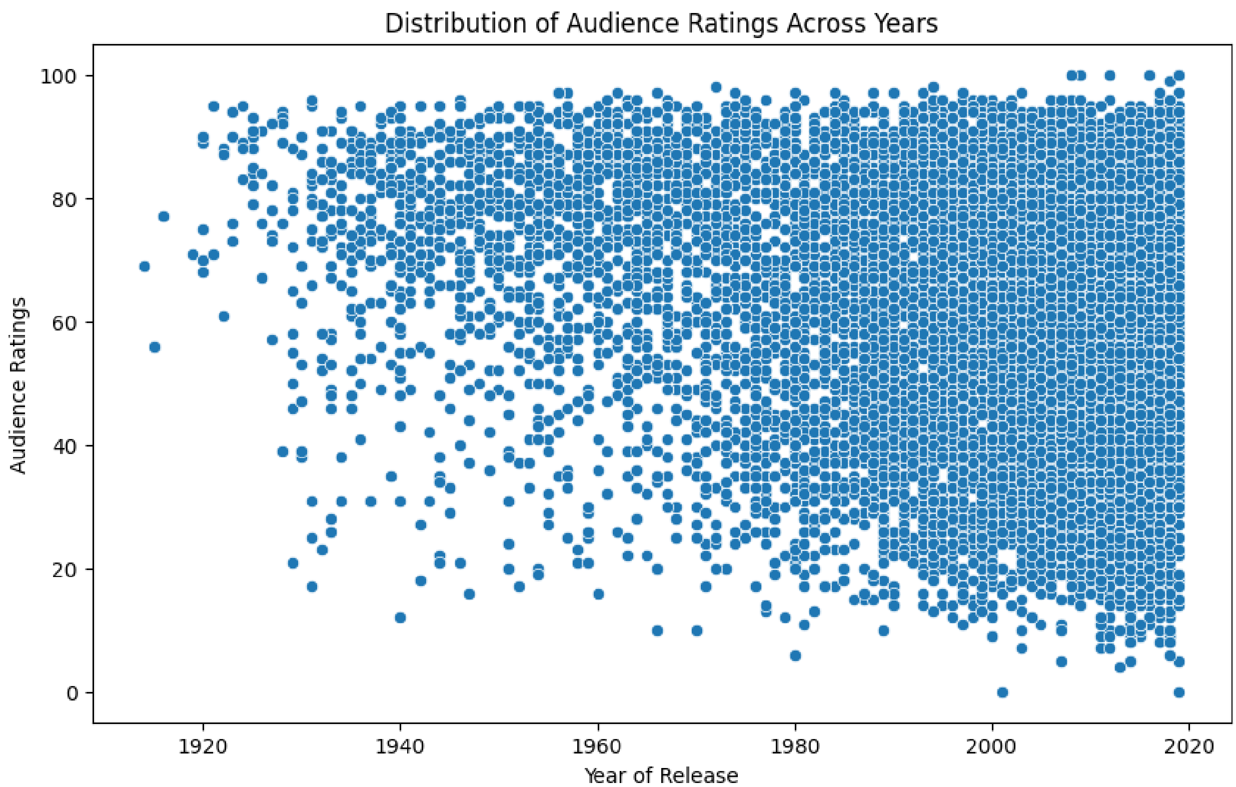
```
plt.xlabel("Year")
plt.ylabel("Number of Movies")

# Customizing x-axis labels to show every 5th year for better
readability
years = release_year_counts.index  # Get unique years from the data
plt.xticks(
    ticks=range(0, len(years), 5),  # Adjust tick spacing to every 5th
year
    labels=[str(year) for year in years[::5]]  # Display labels for
every 5th year
)

# Display the plot
plt.show()
```



Distribution of Audience Ratings Across Years

```
# Create a scatter plot to visualize the distribution of audience
ratings across release years
plt.figure(figsize=(10, 6))

# sns.scatterplot creates a scatter plot with 'release_year' on the x-
axis and 'audience_rating' on the y-axis
```

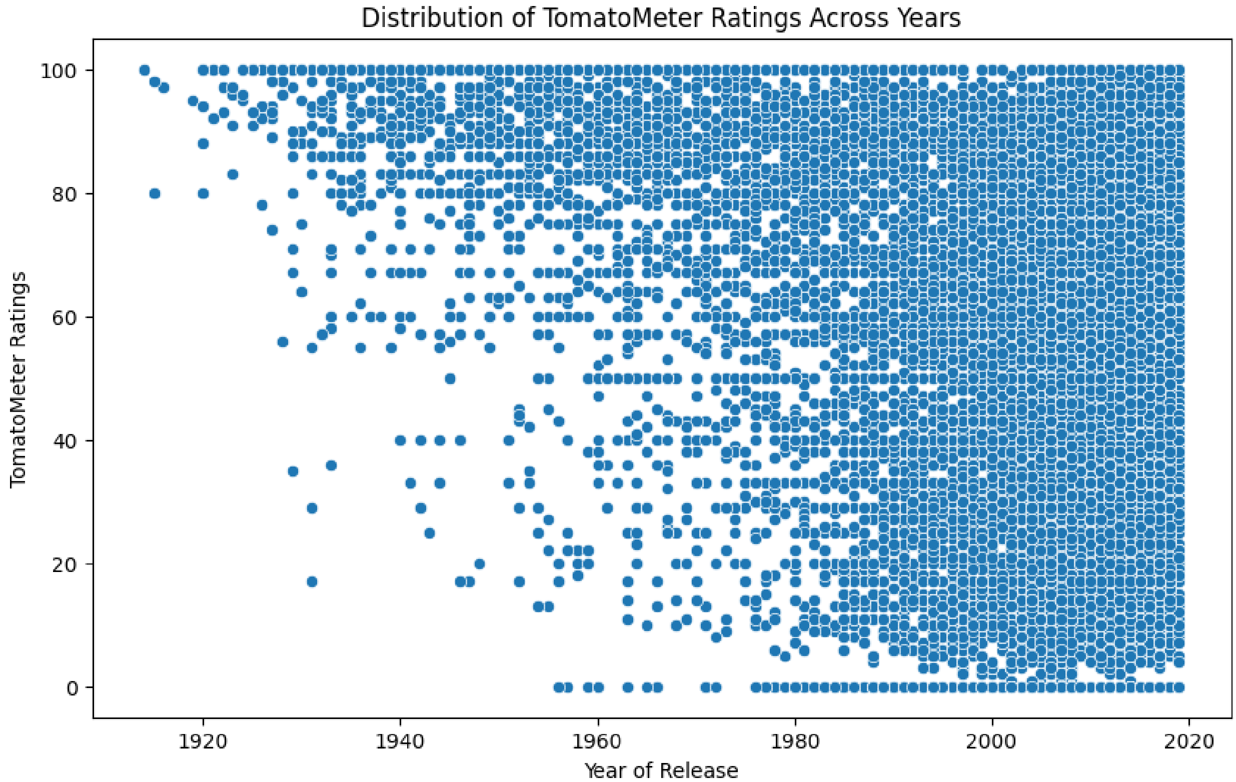
```
sns.scatterplot(x='release_year', y='audience_rating',
data=audience_data)

# Adding a title and axis labels for clarity
plt.title("Distribution of Audience Ratings Across Years")
plt.xlabel("Year of Release")
plt.ylabel("Audience Ratings")

# Display the plot
plt.show()
```



Distribution of TomatoMeter Ratings Across Years

```
# Create a scatter plot to visualize the distribution of TomatoMeter
ratings across release years
plt.figure(figsize=(10, 6))

# sns.scatterplot creates a scatter plot with 'release_year' on the x-
axis and 'tomatometer_rating' on the y-axis
sns.scatterplot(x='release_year', y='tomatometer_rating',
data=audience_data)

# Adding a title and axis labels for clarity
plt.title("Distribution of TomatoMeter Ratings Across Years")
plt.xlabel("Year of Release")
```

```
plt.ylabel("TomatoMeter Ratings")

# Display the plot
plt.show()
```



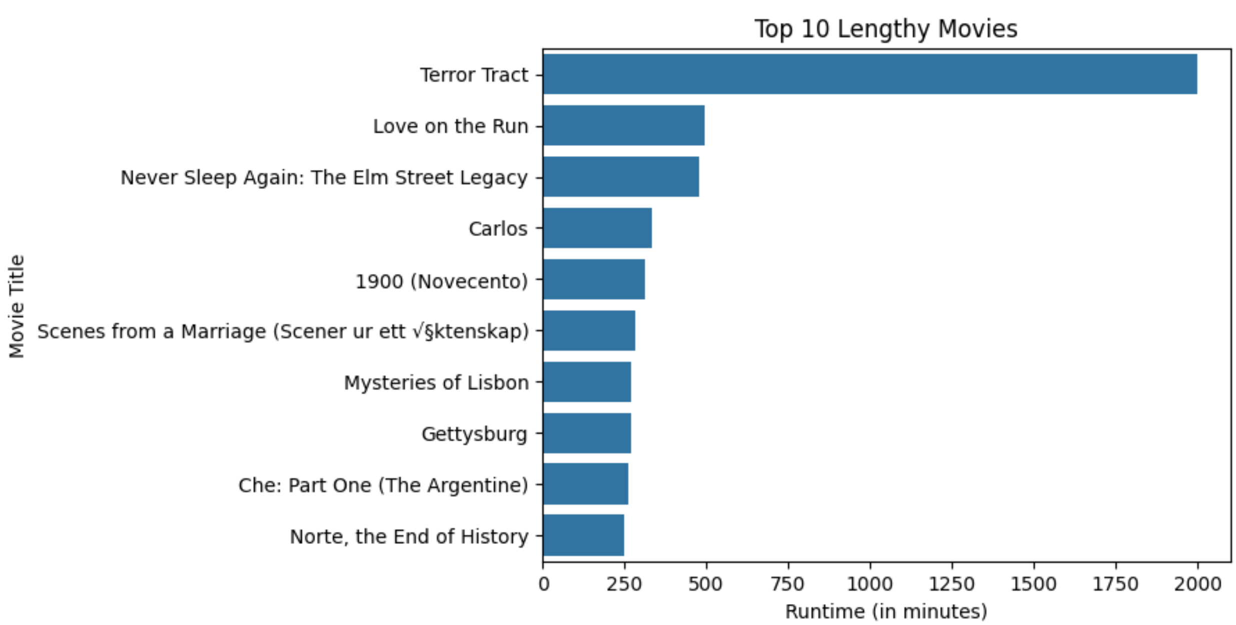
Finding the Average Audience Rating for Each Writer

```
# Grouping the data by 'writers' and calculating the average
'audience_rating' for each writer
# Sorting the results in descending order to get the highest average
ratings first
audience_data.groupby('writers')
['audience_rating'].mean().sort_values(ascending=False)

writers
Abe Forsythe                         100.0
Bertrand Normand                     100.0
Anne Aghion                          100.0
Michele Mitchell                     100.0
Scott Beck, Bryan Woods              100.0
                                     ...
Naman Barsoom, Daniel Wallner          5.0
Roy Sallows                            5.0
Ted Kupper                             0.0
Russell DeGrazier                      0.0
```

```
Rob Gilmer                                0.0
Name: audience_rating, Length: 11332, dtype: float64
```

Visualizing Top 10 Lengthy Movies

```
# Find the top 10 longest movies by selecting the 10 movies with the
largest runtime
# The 'nlargest' function is used to select the top 10 rows based on
'runtime_in_minutes'
le = audience_data.nlargest(10, 'runtime_in_minutes')[['movie_title',
'runtime_in_minutes']].set_index('movie_title')

# Create a bar plot to visualize the top 10 longest movies
# Fix: Use keyword arguments `x` and `y` explicitly in sns.barplot for
better readability
sns.barplot(x=le['runtime_in_minutes'], y=le.index)

# Adding title and labels to the plot for clarity
plt.title('Top 10 Lengthy Movies')
plt.xlabel('Runtime (in minutes)')
plt.ylabel('Movie Title')

# Display the plot
plt.show()
```



Displaying Top 10 Highest Audience-Rated Movie Titles, Writers, and Directors

```
# Find the top 10 highest audience-rated movies by selecting the 10
movies with the highest 'audience_rating'
# The 'nlargest' function is used to select the top 10 rows based on
```

```
'audience_rating'
top_10 = audience_data.nlargest(10, 'audience_rating')[['movie_title', 'audience_rating', 'writers', 'directors']].set_index('movie_title')

# Display the top 10 movies with their audience ratings, writers, and directors
top_10
```


```
{"summary":"{\n  \"name\": \"top_10\",\n  \"rows\": 10,\n  \"fields\": [\n    {\n      \"column\": \"movie_title\",\n      \"properties\": {\n        \"dtype\": \"string\",\n        \"num_unique_values\": 10,\n        \"samples\": [\n          \"Maktub\",\n          \"Ice People\",\n          \"The Uncondemned\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"audience_rating\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 0.6992058987801011,\n        \"min\": 98.0,\n        \"max\": 100.0,\n        \"num_unique_values\": 3,\n        \"samples\": [\n          100.0,\n          99.0,\n          98.0\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"writers\",\n      \"properties\": {\n        \"dtype\": \"string\",\n        \"num_unique_values\": 10,\n        \"samples\": [\n          \"Guy Amir, Hanan Savyon\",\n          \"Anne Aghion\",\n          \"Michele Mitchell\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"directors\",\n      \"properties\": {\n        \"dtype\": \"string\",\n        \"num_unique_values\": 10,\n        \"samples\": [\n          \"Oded Raz\",\n          \"Anne Aghion\",\n          \"Nick Louvel, Michele Mitchell\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe","variable_name":"top_10"}
```


Average Audience Rating of Movies Year-wise

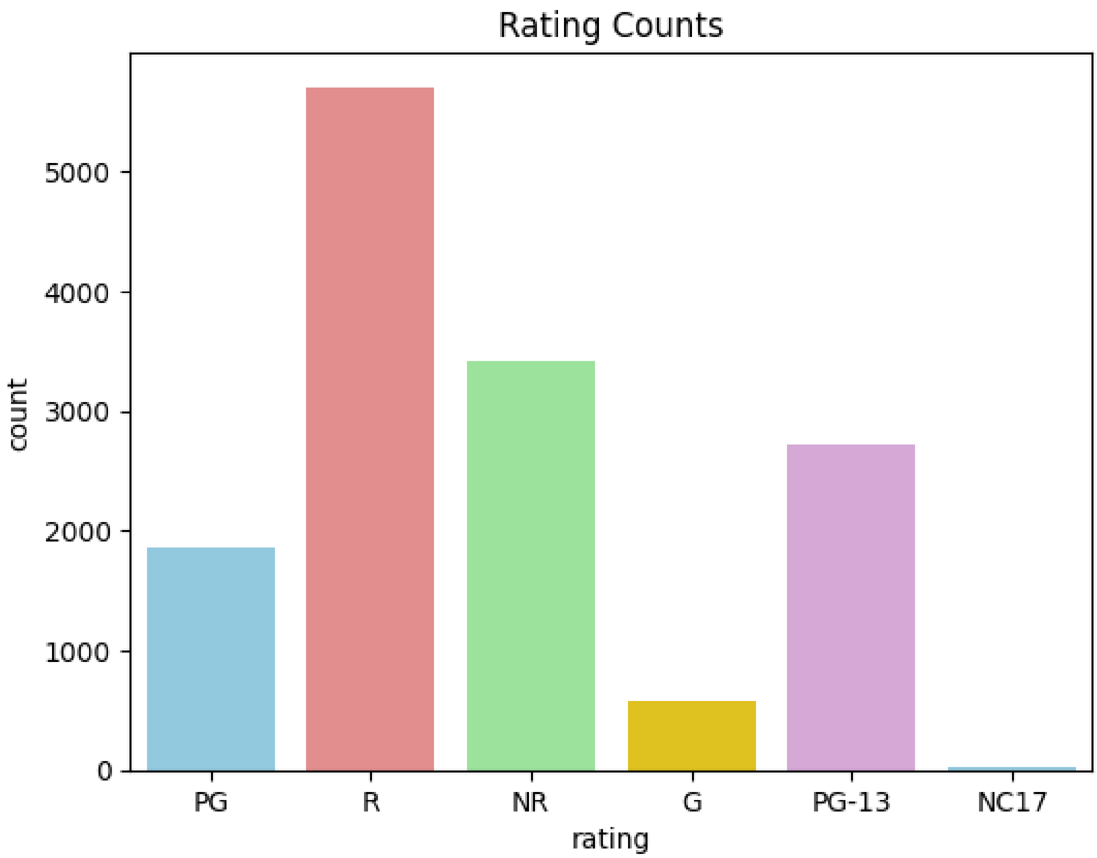
```
# Calculate the average audience rating for movies year-wise
# Grouping by 'release_year' and calculating the mean of 'audience_rating' for each year
data1 = (
    audience_data.groupby('release_year')[['release_year', 'audience_rating']]  # Group by 'release_year'
    .mean()  # Calculate the mean audience rating for each year
    .sort_values(by='audience_rating', ascending=False)  # Sort by average rating in descending order
    .set_index('release_year')  # Set 'release_year' as the index for better visualization
)

# Display the result
data1
```

```
{"summary":"{\n  \"name\": \"data1\",\n  \"rows\": 104,\n  \"fields\": [\n    {\n      \"column\": \"release_year\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 30.265436380185346,\n        \"min\": 1914.0,\n        \"max\": 2019.0,\n        \"num_unique_values\": 104,\n        \"samples\": [\n          1959.0,\n          2019.0,\n          1978.0\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"audience_rating\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 8.21125275058634,\n        \"min\": 53.37461773700306,\n        \"max\": 88.75,\n        \"num_unique_values\": 103,\n        \"samples\": [\n          72.41379310344827,\n          61.60227272727273,\n          61.87777777777778\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe","variable_name":"data1"}
```


Visualizing Rating Counts of Movies

```
# Calculate the count of each unique rating in the 'rating' column
audience_data['rating'].value_counts()

# Create a count plot to visualize the distribution of movie ratings
# The palette is customized with different colors for each rating
category
sns.countplot(audience_data, x='rating', palette=['skyblue',
'lightcoral', 'lightgreen', 'gold', 'plum'])

# Add a title to the plot for clarity
plt.title('Rating Counts')

# Display the plot
plt.show()
```

```
<ipython-input-23-e74125159c82>:6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(audience_data, x='rating', palette=['skyblue',
'lightcoral', 'lightgreen', 'gold', 'plum'])
<ipython-input-23-e74125159c82>:6: UserWarning:
The palette list has fewer values (5) than needed (6) and will cycle,
which may produce an uninterpretable plot.
  sns.countplot(audience_data, x='rating', palette=['skyblue',
'lightcoral', 'lightgreen', 'gold', 'plum'])
```

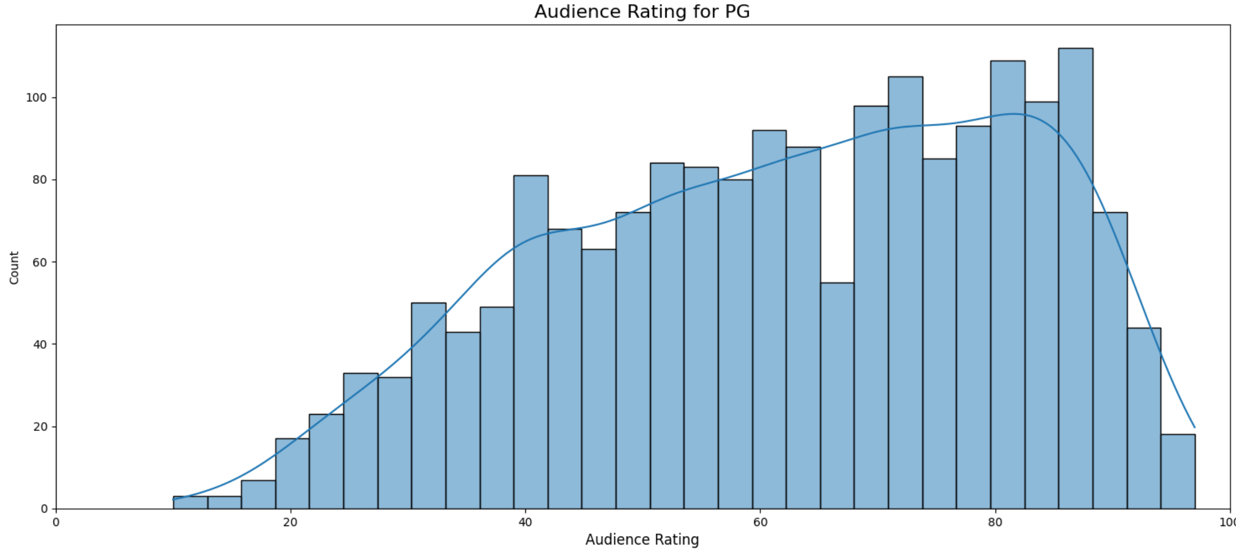


Visualizing Audience Ratings for Different Movie Ratings (PG, R, NR, G, PG-13, NC-17)

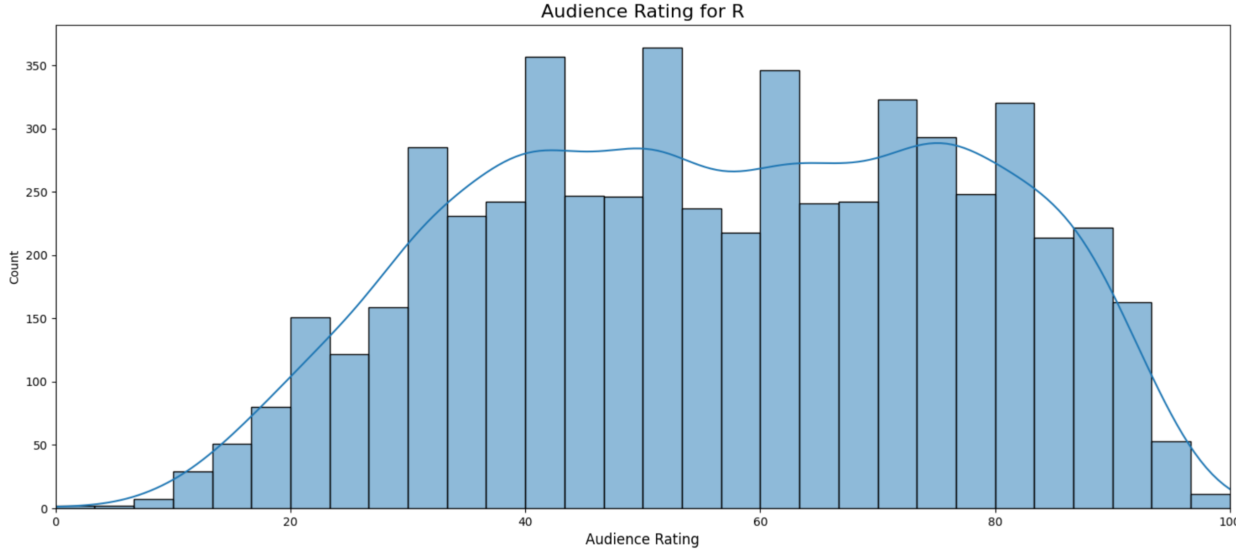
```
# Create a 6-row subplot to visualize the distribution of audience
ratings for different movie ratings
f, ax = plt.subplots(6, 1, figsize=(15, 40))

# Plot for PG rating
sns.histplot(audience_data[(audience_data['rating'] == 'PG') &
                             (audience_data['audience_rating'] <=
100)].audience_rating,
             ax=ax[0], bins=30, kde=True)
ax[0].set_title('Audience Rating for PG', fontsize=16)
ax[0].set_xlabel("Audience Rating", fontsize=12)
ax[0].set_xlim([0, 100])

# Plot for R rating
sns.histplot(audience_data[(audience_data['rating'] == 'R') &
                             (audience_data['audience_rating'] <=
100)].audience_rating,
             ax=ax[1], bins=30, kde=True)
ax[1].set_title('Audience Rating for R', fontsize=16)
ax[1].set_xlabel("Audience Rating", fontsize=12)
ax[1].set_xlim([0, 100])
```

```
# Plot for NR (Not Rated) rating
sns.histplot(audience_data[(audience_data['rating'] == 'NR') &
                               (audience_data['audience_rating'] <=
100)].audience_rating,
             ax=ax[2], bins=30, kde=True)
ax[2].set_title('Audience Rating for NR', fontsize=16)
ax[2].set_xlabel("Audience Rating", fontsize=12)
ax[2].set_xlim([0, 100])

# Plot for G rating
sns.histplot(audience_data[(audience_data['rating'] == 'G') &
                               (audience_data['audience_rating'] <=
100)].audience_rating,
             ax=ax[3], bins=30, kde=True)
ax[3].set_title('Audience Rating for G', fontsize=16)
ax[3].set_xlabel("Audience Rating", fontsize=12)
ax[3].set_xlim([0, 100])

# Plot for PG-13 rating
sns.histplot(audience_data[(audience_data['rating'] == 'PG-13') &
                               (audience_data['audience_rating'] <=
100)].audience_rating,
             ax=ax[4], bins=30, kde=True)
ax[4].set_title('Audience Rating for PG-13', fontsize=16)
ax[4].set_xlabel("Audience Rating", fontsize=12)
ax[4].set_xlim([0, 100])

# Plot for NC-17 rating
sns.histplot(audience_data[(audience_data['rating'] == 'NC17') &
                               (audience_data['audience_rating'] <=
100)].audience_rating,
             ax=ax[5], bins=30, kde=True)
ax[5].set_title('Audience Rating for NC-17', fontsize=16)
ax[5].set_xlabel("Audience Rating", fontsize=12)
ax[5].set_xlim([0, 100])

# Adjust layout to ensure proper spacing between subplots
plt.tight_layout()

# Display the plots
plt.show()
```

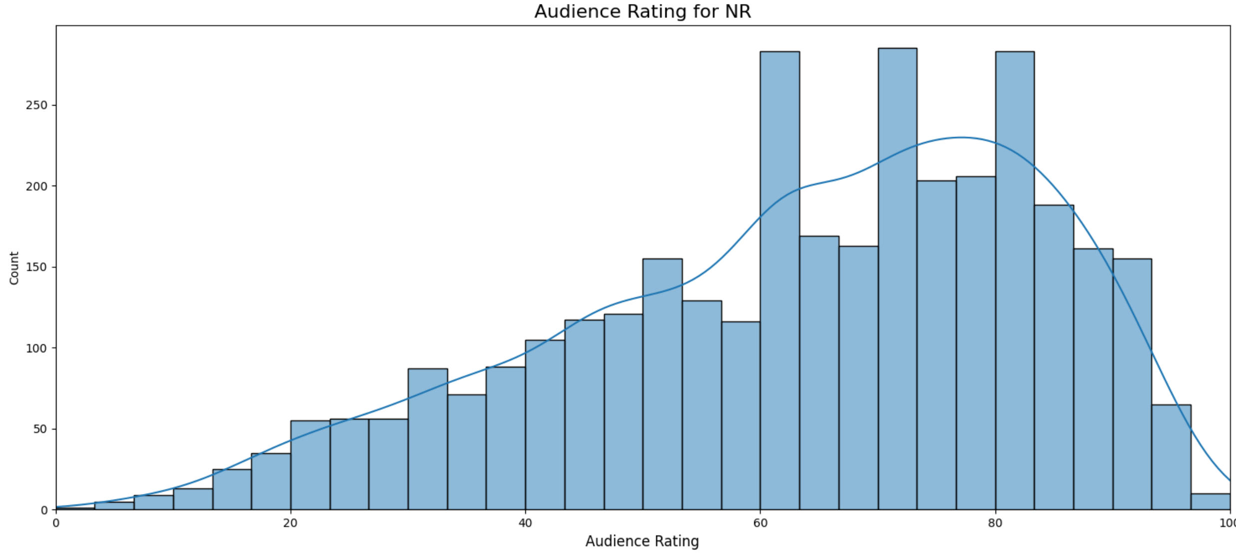
Audience Rating for PG



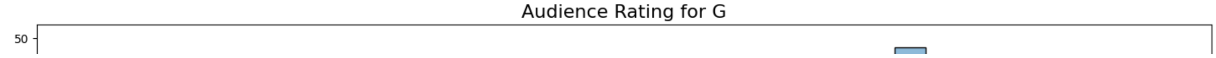
Audience Rating for R



Audience Rating for NR



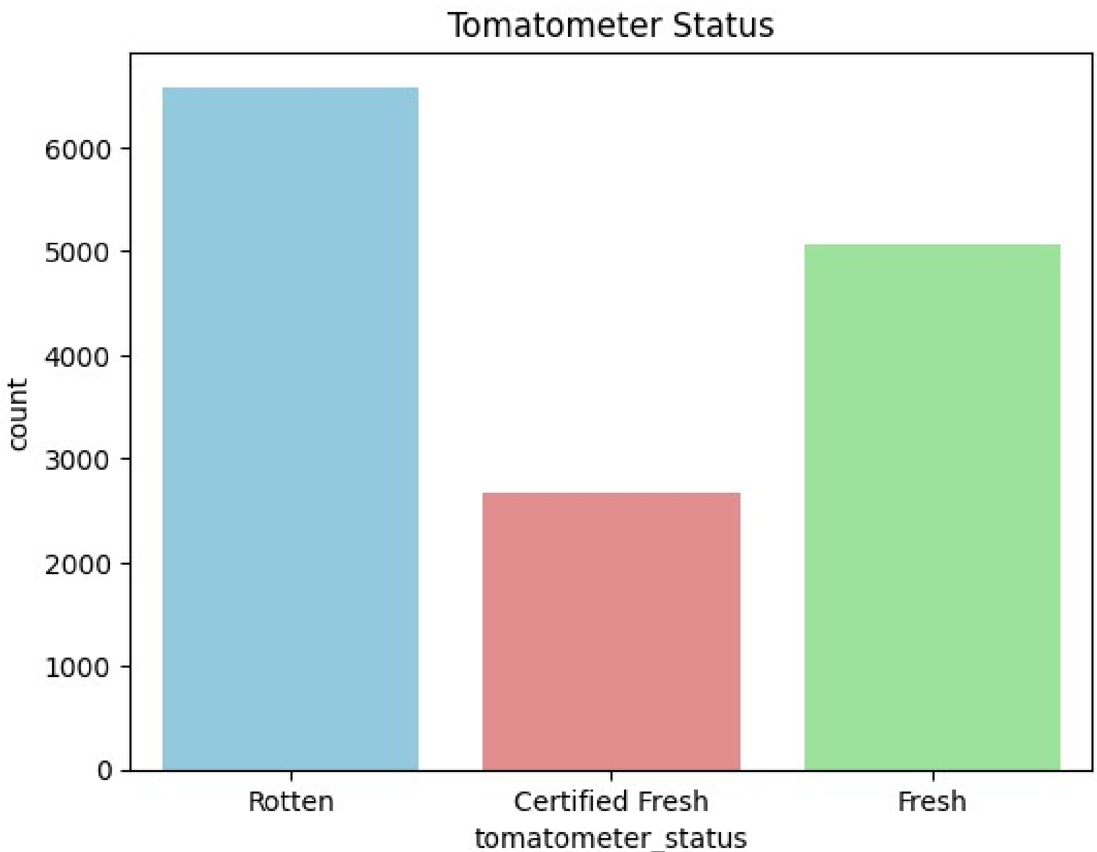
Audience Rating for G



Visualizing TomatoMeter Status Counts

```
# Calculate the count of each unique value in the 'tomatometer_status'
column
audience_data['tomatometer_status'].value_counts()

# Create a count plot to visualize the distribution of tomato meter
status
# The palette is customized with different colors for each tomato
meter status category
sns.countplot(audience_data, x='tomatometer_status',
palette=['skyblue', 'lightcoral', 'lightgreen'])

# Add a title to the plot for clarity
plt.title('Tomatometer Status')

# Display the plot
plt.show()
```

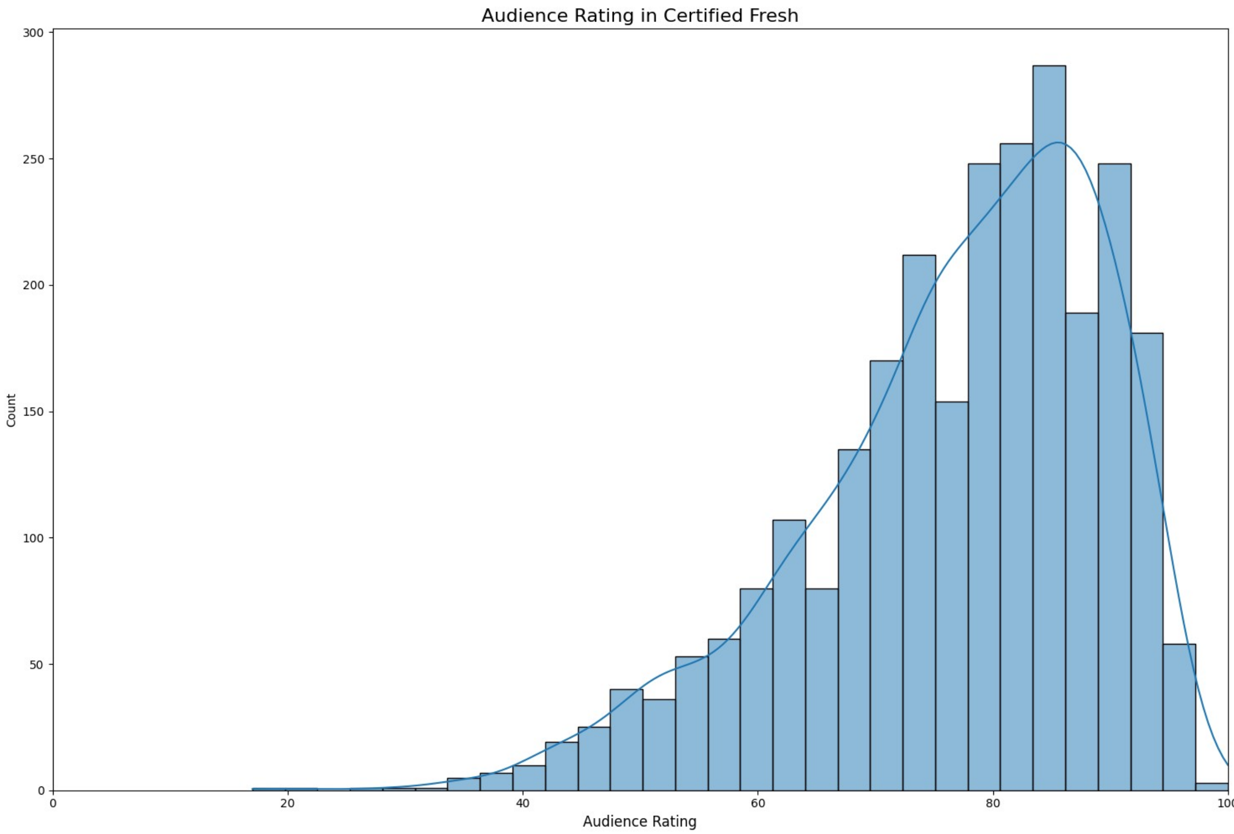
```
<ipython-input-25-960b3596be46>:6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(audience_data, x='tomatometer_status',
palette=['skyblue', 'lightcoral', 'lightgreen'])
```

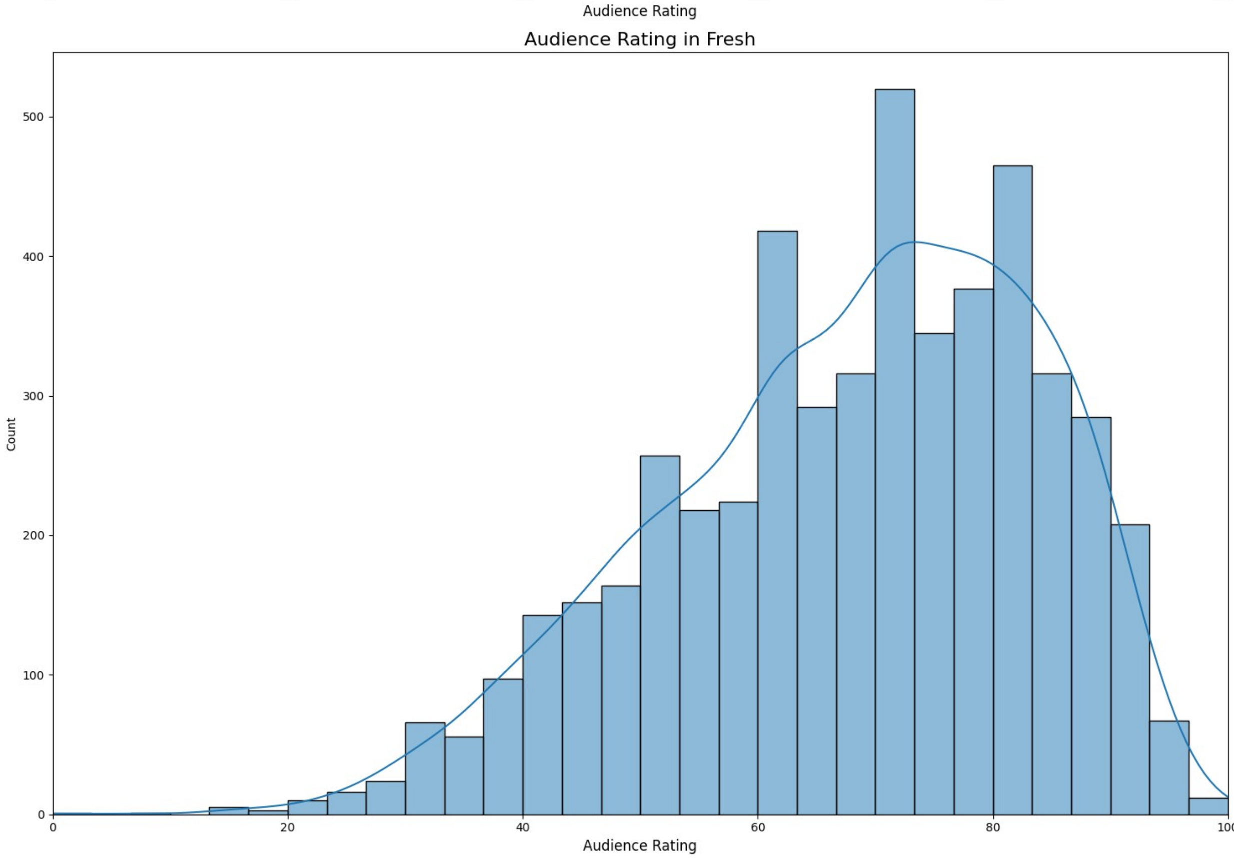


Visualizing Audience Ratings for Different TomatoMeter Statuses (Certified Fresh, Fresh, Rotten)

```
# Create a 3-row subplot to visualize the distribution of audience
ratings for different tomato meter statuses
f, ax = plt.subplots(3, 1, figsize=(15, 30))

# Plot for Certified Fresh status
sns.histplot(audience_data[(audience_data['tomatometer_status'] ==
'Certified Fresh') &
                             (audience_data['audience_rating'] <=
100)].audience_rating,
             ax=ax[0], bins=30, kde=True)
ax[0].set_title('Audience Rating in Certified Fresh', fontsize=16)
ax[0].set_xlabel("Audience Rating", fontsize=12)
ax[0].set_xlim([0, 100])

# Plot for Fresh status
sns.histplot(audience_data[(audience_data['tomatometer_status'] ==
'Fresh') &
                             (audience_data['audience_rating'] <=
100)].audience_rating,
             ax=ax[1], bins=30, kde=True)
```

```
ax[1].set_title('Audience Rating in Fresh', fontsize=16)
ax[1].set_xlabel("Audience Rating", fontsize=12)
ax[1].set_xlim([0, 100])

# Plot for Rotten status
sns.histplot(audience_data[(audience_data['tomatometer_status'] ==
'Rotten') &
                             (audience_data['audience_rating'] <=
100)].audience_rating,
             ax=ax[2], bins=30, kde=True)
ax[2].set_title('Audience Rating in Rotten', fontsize=16)
ax[2].set_xlabel("Audience Rating", fontsize=12)
ax[2].set_xlim([0, 100])

# Adjust layout to ensure proper spacing between subplots
plt.tight_layout()

# Display the plots
plt.show()
```

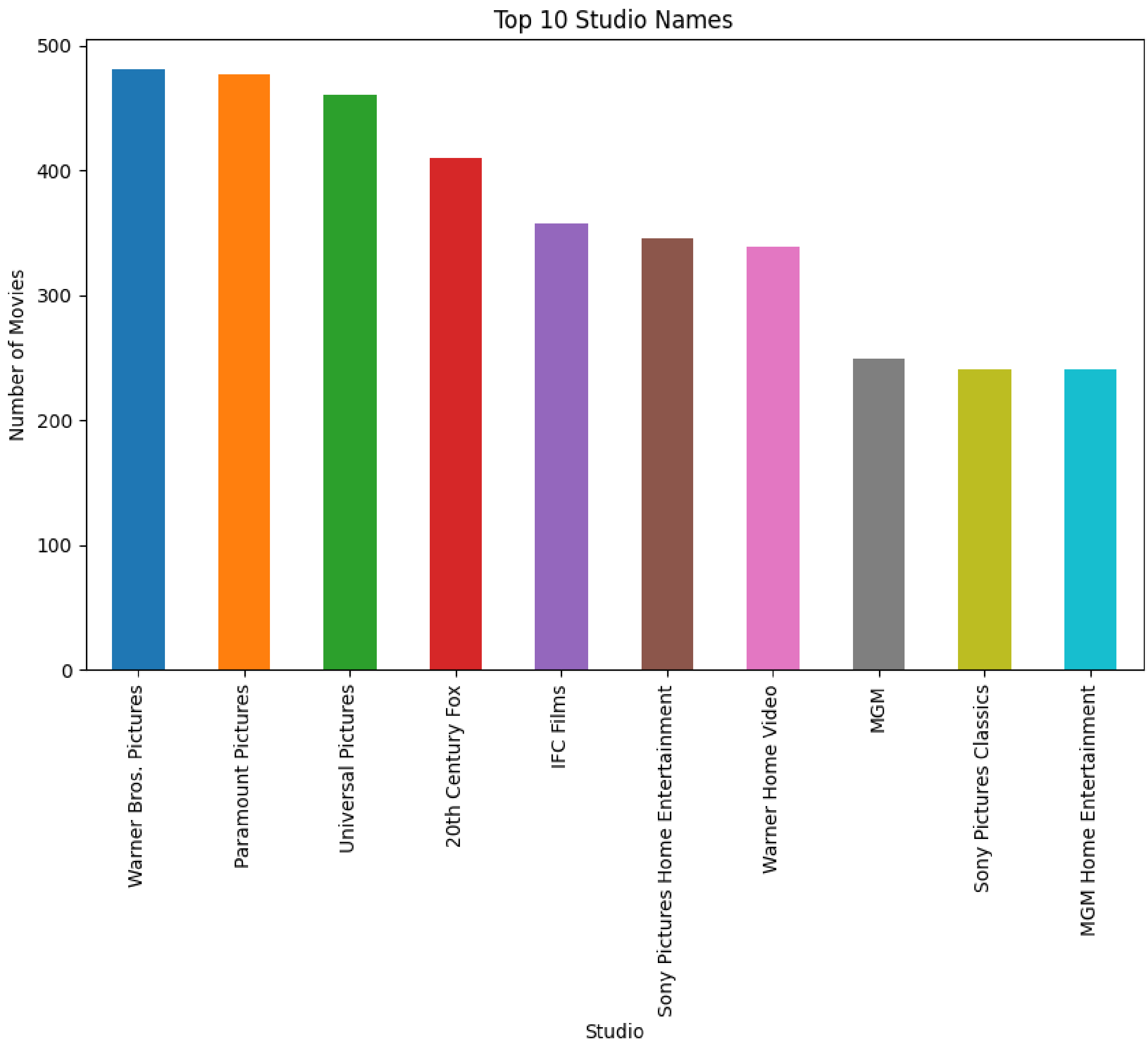

Audience Rating in Certified Fresh
Count
Audience Rating
0
50
100
150
200
250
300
0
20
40
60
80
100


Audience Rating in Fresh
Count
Audience Rating
0
100
200
300
400
500
0
20
40
60
80
100


Audience Rating in Rotten
600

Top 10 Studios by Number of Movies Produced

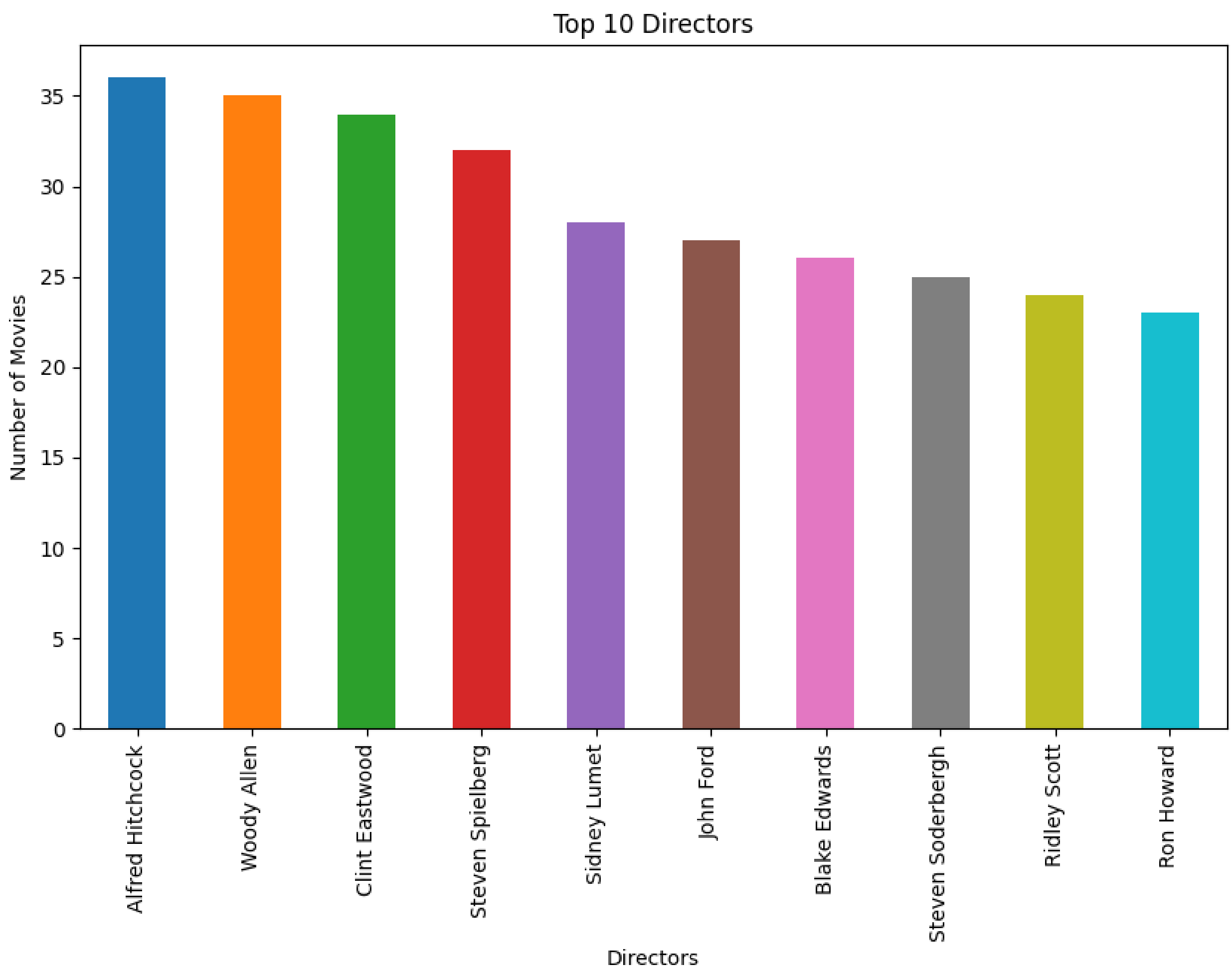
```
# Count the number of movies produced by each studio and select the top 10
studio_10 = audience_data['studio_name'].value_counts().head(10)

# Generate a color palette with a unique color for each bar using 'tab10' colormap
colors = plt.cm.tab10(range(len(studio_10)))  # 'tab10' colormap ensures distinct colors for each bar

# Create a bar plot for the top 10 studios and their movie counts
plt.figure(figsize=(10, 6))
studio_10.plot(kind='bar', color=colors)

# Add a title and labels to the plot for clarity
plt.title("Top 10 Studio Names")
plt.xlabel("Studio")
plt.ylabel("Number of Movies")

# Display the plot
plt.show()
```



Top 10 Directors by Number of Movies Directed

```
# Count the number of movies directed by each director and select the
top 10
top_10_directors = audience_data['directors'].value_counts().head(10)

# Generate a color palette with a unique color for each bar using
'tab10' colormap
colors = plt.cm.tab10(range(len(top_10_directors)))  # 'tab10'
colormap ensures distinct colors for each bar

# Create a bar plot for the top 10 directors and their movie counts
plt.figure(figsize=(10, 6))
top_10_directors.plot(kind='bar', color=colors)

# Add a title and labels to the plot for clarity
```

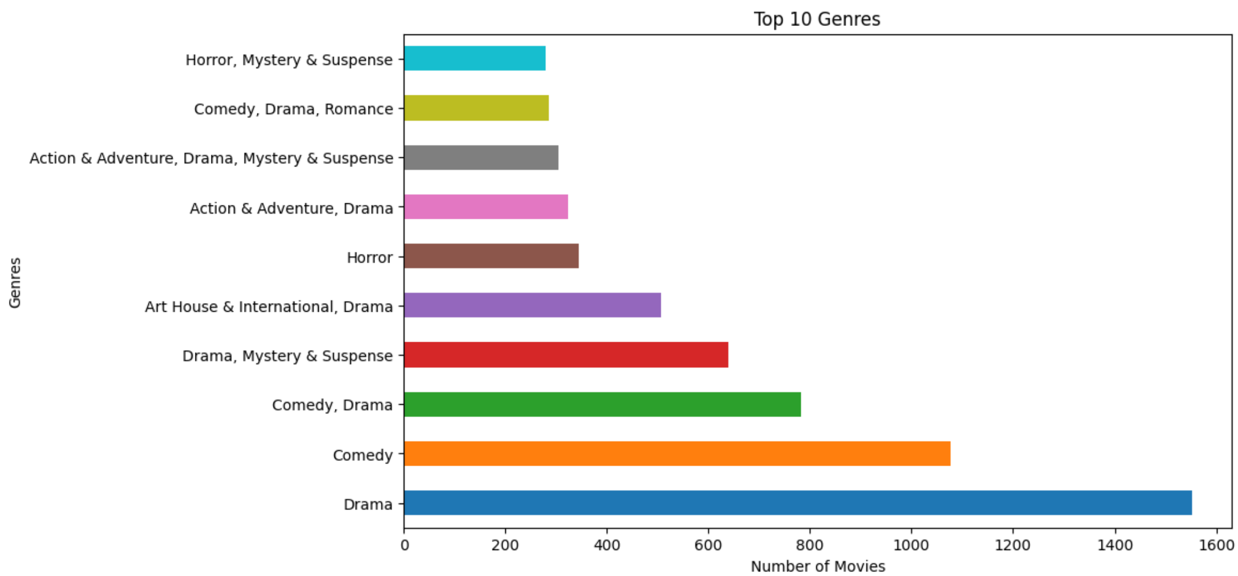
```
plt.title("Top 10 Directors")
plt.xlabel("Directors")
plt.ylabel("Number of Movies")

# Display the plot
plt.show()
```



Top 10 Genres by Number of Movies

```
# Count the number of movies in each genre and select the top 10
genres
top_10_genres =
audience_data['genre'].value_counts().head(10).sort_values(ascending=F
alse)

# Generate a color palette with a unique color for each bar using
'tab10' colormap
colors = plt.cm.tab10(range(len(top_10_genres)))  # 'tab10' colormap
ensures distinct colors for each bar
```

```python
# Create a horizontal bar plot for the top 10 genres and their movie counts
plt.figure(figsize=(10, 6))
top_10_genres.plot(kind='barh', color=colors)

# Add a title and labels to the plot for clarity
plt.title("Top 10 Genres")
plt.xlabel("Number of Movies")
plt.ylabel("Genres")

# Display the plot
plt.show()
```



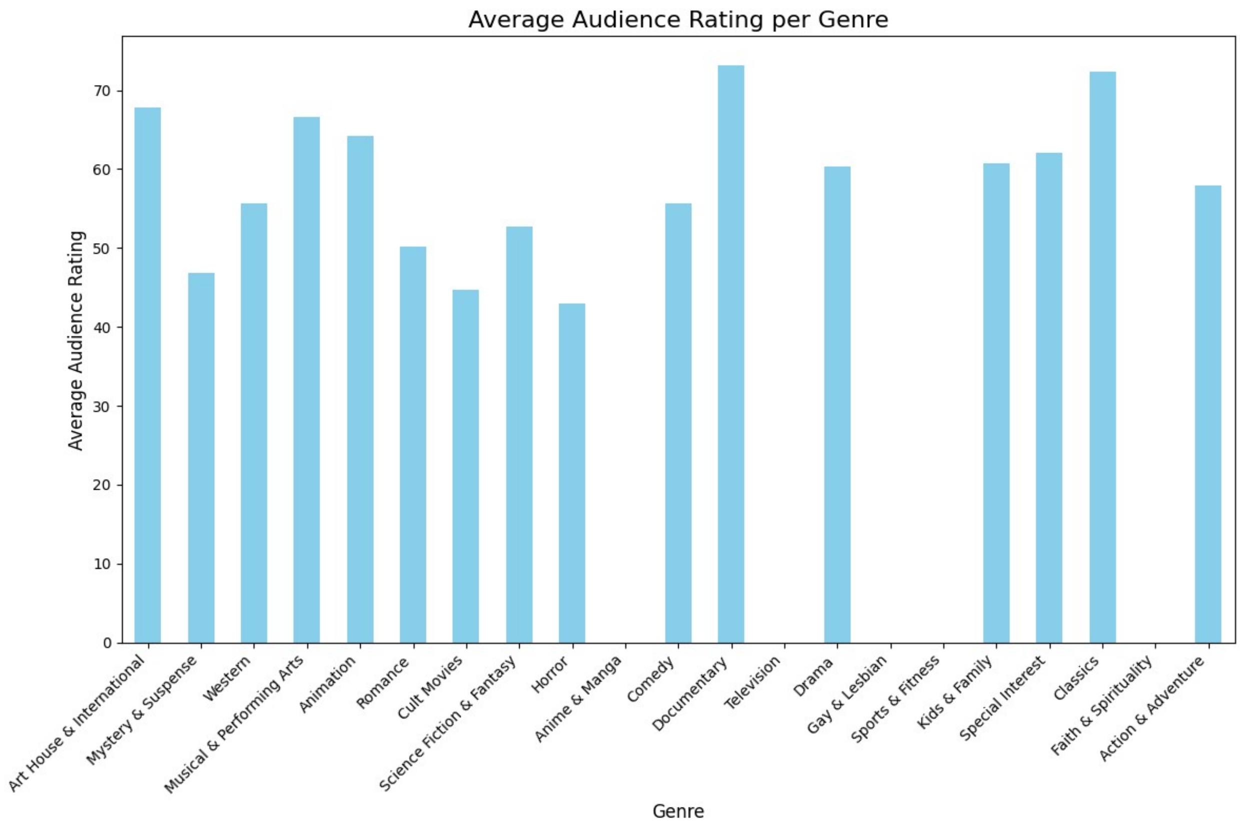
Extracting and Analyzing Unique Genres in the Dataset

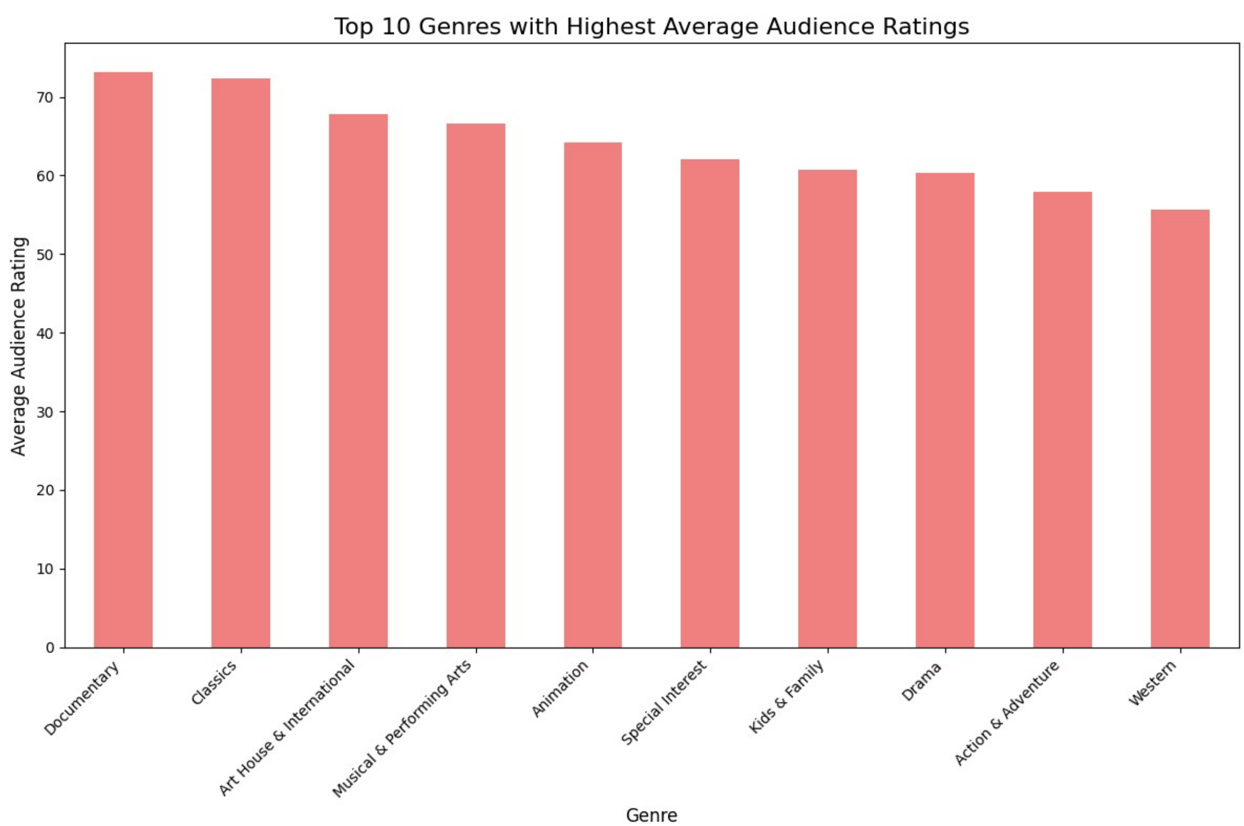
```python
# Extract all unique genres from the 'genre' column, handle missing values, and split genres by comma
unique_genres = set(genre.strip() for genres in audience_data['genre'].dropna() for genre in genres.split(','))

# Print the number of unique genres and the list of genres
print(f"Number of unique genres: {len(unique_genres)}")
print(f"Unique genres: {unique_genres}")

# Split the genres for each movie and flatten the list to count occurrences
genres = audience_data['genre'].str.split(',').explode()

# Count the occurrences of each genre
genre_counts = genres.value_counts()

# Ensure all unique genres are included, even if a genre has a count
```

```
of 0
genre_counts = genre_counts.reindex(unique_genres, fill_value=0)

# Display the genre counts
print("Unique genre counts:", genre_counts)

# Combine the exploded genres with the original ratings to calculate
the average rating for each genre
ratings_by_genre = pd.DataFrame({
    'genre': genres,
    'audience_rating': audience_data.loc[genres.index,
'audience_rating'].values
})

# Calculate the average rating for each genre
avg_ratings_by_genre = ratings_by_genre.groupby('genre')
['audience_rating'].mean()

# Ensure all unique genres are included, even if a genre has a count
of 0
avg_ratings_by_genre = avg_ratings_by_genre.reindex(unique_genres,
fill_value=0)

# Plot the average ratings for each genre
plt.figure(figsize=(12, 8))
avg_ratings_by_genre.plot(kind='bar', color='skyblue')
plt.title('Average Audience Rating per Genre', fontsize=16)
plt.xlabel('Genre', fontsize=12)
plt.ylabel('Average Audience Rating', fontsize=12)
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()

# Get the top 10 genres with the highest average audience ratings
top_10_genres =
avg_ratings_by_genre.sort_values(ascending=False).head(10)

# Plot the top 10 genres with the highest average ratings
plt.figure(figsize=(12, 8))
top_10_genres.plot(kind='bar', color='lightcoral')
plt.title('Top 10 Genres with Highest Average Audience Ratings',
fontsize=16)
plt.xlabel('Genre', fontsize=12)
plt.ylabel('Average Audience Rating', fontsize=12)
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

```
Number of unique genres: 21
Unique genres: {'Art House & International', 'Mystery & Suspense',
```

```
'Western', 'Musical & Performing Arts', 'Animation', 'Romance', 'Cult
Movies', 'Science Fiction & Fantasy', 'Horror', 'Anime & Manga',
'Comedy', 'Documentary', 'Television', 'Drama', 'Gay & Lesbian',
'Sports & Fitness', 'Kids & Family', 'Special Interest', 'Classics',
'Faith & Spirituality', 'Action & Adventure'}
Unique genre counts: genre
Art House & International    1661
Mystery & Suspense            234
Western                         7
Musical & Performing Arts      17
Animation                     291
Romance                        11
Cult Movies                    17
Science Fiction & Fantasy      29
Horror                        735
Anime & Manga                   0
Comedy                       3235
Documentary                   712
Television                      0
Drama                        3207
Gay & Lesbian                   0
Sports & Fitness                0
Kids & Family                  36
Special Interest                1
Classics                     1019
Faith & Spirituality            0
Action & Adventure           3099
Name: count, dtype: int64
```

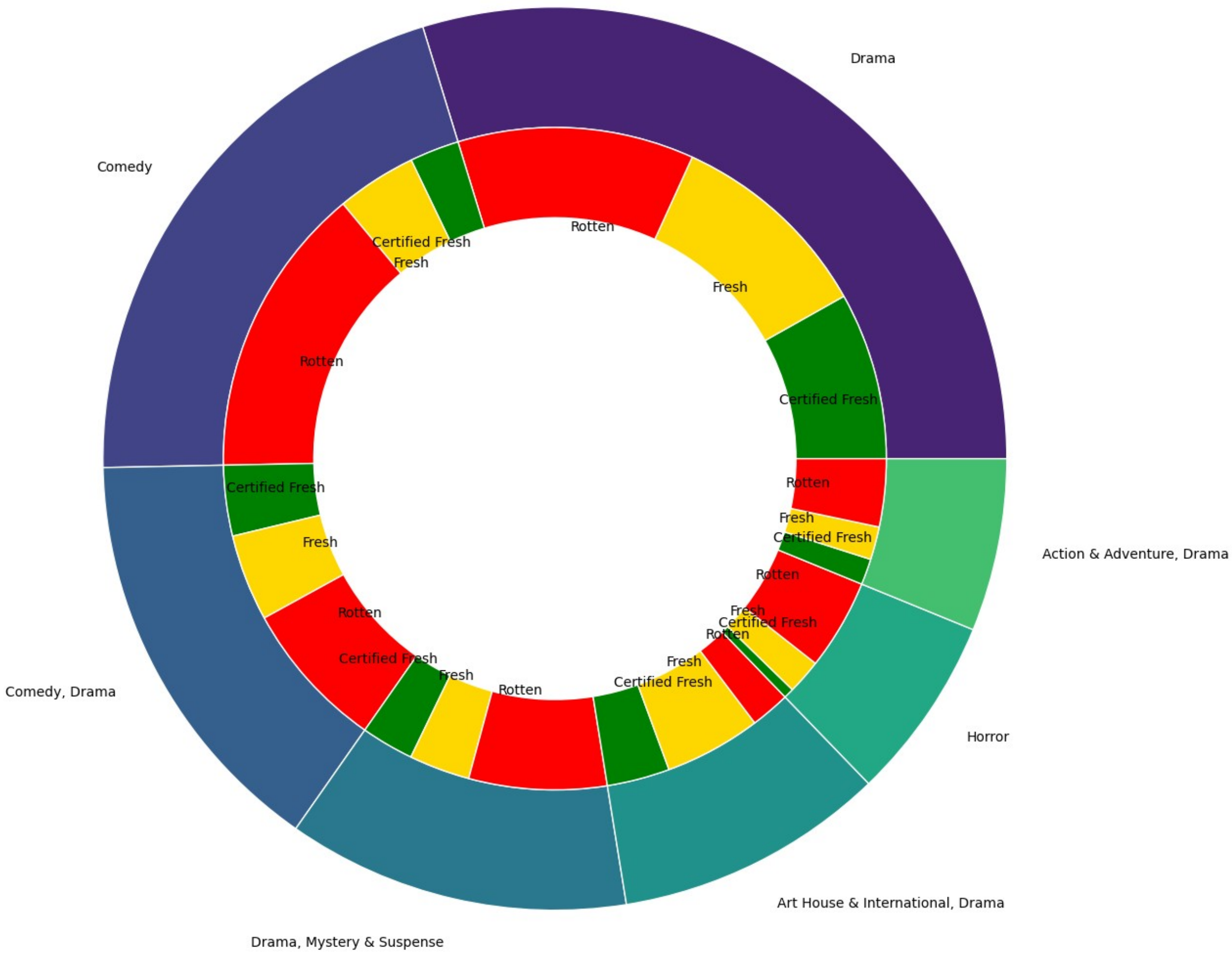

Average Audience Rating per Genre
Average Audience Rating
0
10
20
30
40
50
60
70
Art House & International
Mystery & Suspense
Western
Musical & Performing Arts
Animation
Romance
Cult Movies
Science Fiction & Fantasy
Horror
Anime & Manga
Comedy
Documentary
Television
Drama
Gay & Lesbian
Sports & Fitness
Kids & Family
Special Interest
Classics
Faith & Spirituality
Action & Adventure
Genre


Top 10 Genres with Highest Average Audience Ratings
Average Audience Rating
0
10
20
30
40
50
60
70
Documentary
Classics
Art House & International
Musical & Performing Arts
Animation
Special Interest
Kids & Family
Drama
Action & Adventure
Western
Genre

Genres and Tomatometer Status Distribution: Nested Pie Chart

```
# Genres and Tomatometer Status Distribution

# Step 1: Define data for genres and tomatometer status
# Get the top 7 genres by counting their occurrences in the 'genre'
column
genres = audience_data['genre'].value_counts().head(7)  # Top 7 genres
group_names = genres.index  # Genre names (indices)
group_size = genres.values  # Genre counts (values)

# Step 2: Tomatometer status distribution for each genre
# Group the data by genre and tomatometer status, then count the
occurrences for each combination
genre_status = audience_data.groupby(['genre',
'tomatometer_status']).size().unstack(fill_value=0)

# Step 3: Prepare subgroup names and sizes for the pie chart
# The subgroup names are 'Certified Fresh', 'Fresh', and 'Rotten'
repeated for each genre
subgroup_names = ['Certified Fresh', 'Fresh', 'Rotten'] *
len(group_names)
subgroup_size = []  # List to store the size of each subgroup

# Loop through each genre to get the count of each tomatometer status
within that genre
for genre in group_names:
    subgroup_size.append(genre_status.loc[genre, 'Certified Fresh'])
# Count of 'Certified Fresh' for each genre
    subgroup_size.append(genre_status.loc[genre, 'Fresh'])  # Count of
'Fresh' for each genre
    subgroup_size.append(genre_status.loc[genre, 'Rotten'])  # Count
of 'Rotten' for each genre

# Step 4: Define the colors for the pie chart
# Outer pie chart uses a viridis colormap
outer_colors = plt.cm.viridis([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7])

# Inner pie chart uses a fixed set of colors for the three subgroups
inner_colors = ['green', 'gold', 'red'] * len(group_names)

# Step 5: Create the nested pie chart
fig, ax = plt.subplots(figsize=(10, 10))  # Create a figure with a
specified size
ax.axis('equal')  # Ensure the pie chart is a circle

# Outer pie chart represents the genres
outer_pie, _ = ax.pie(group_size, radius=1.5, labels=group_names,
colors=outer_colors)
plt.setp(outer_pie, width=0.4, edgecolor='white')  # Set the width and
```

```
edgecolor for the outer pie chart

# Inner pie chart represents the tomatometer status for each genre
inner_pie, _ = ax.pie(subgroup_size, radius=1.1,
labels=subgroup_names, labeldistance=0.7, colors=inner_colors)
plt.setp(inner_pie, width=0.3, edgecolor='white')  # Set the width and
edgecolor for the inner pie chart

# Step 6: Display the chart
plt.margins(0, 0)  # Remove margins around the plot
plt.show()  # Show the final plot
```

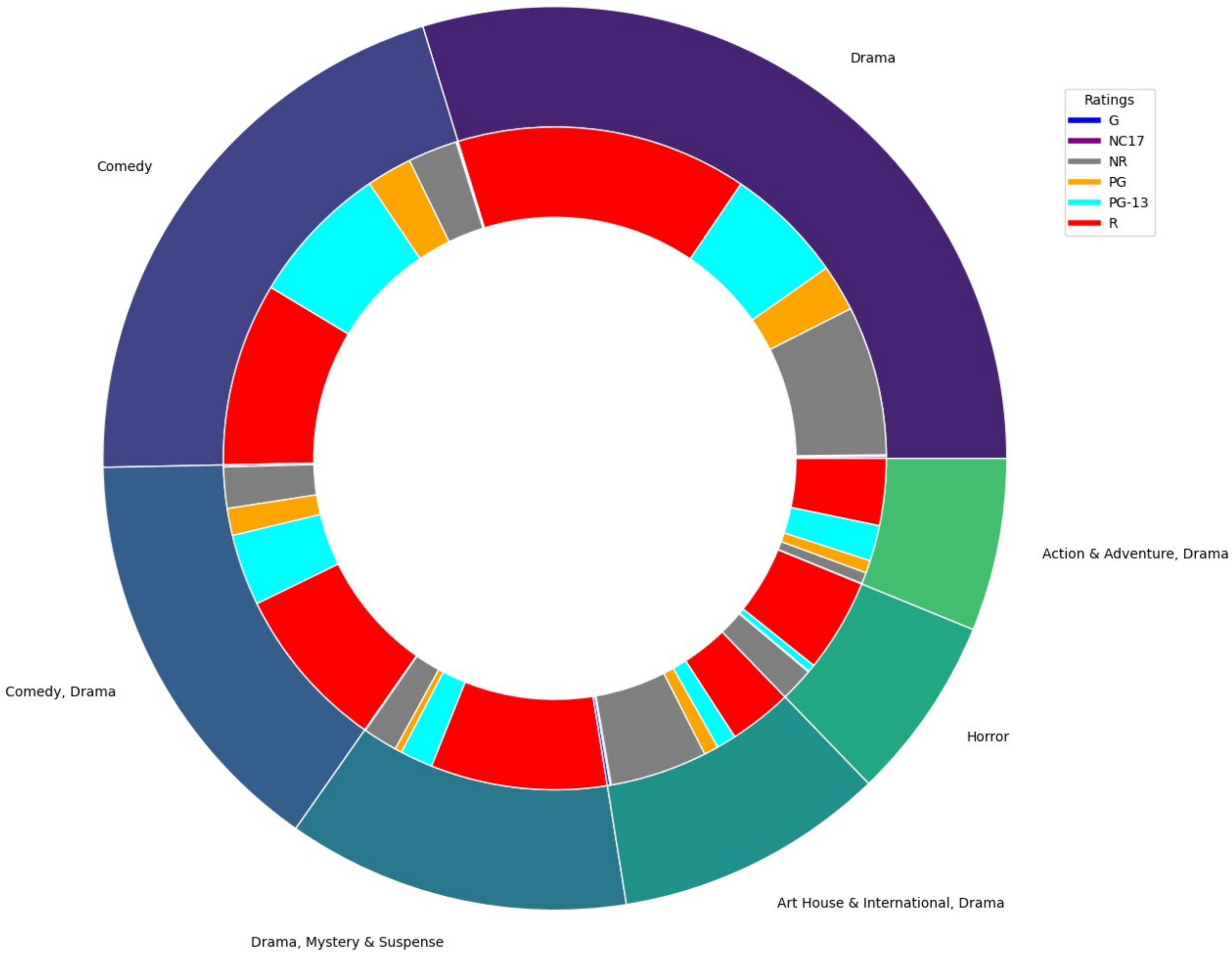


Genres and Ratings Distribution: Nested Pie Chart

```
# Define the top genres (Top 7 genres based on their count)
top_genres = audience_data['genre'].value_counts().head(7)  # Top 7
genres
group_names = top_genres.index  # Genre names
group_size = top_genres.values  # Genre counts
```

```
# Ratings distribution for each genre
# Group by genre and rating to count occurrences of each rating per
genre
genre_rating = audience_data.groupby(['genre',
'rating']).size().unstack(fill_value=0)

# Prepare subgroup names and sizes for the inner pie chart (Ratings)
# Create a list of rating categories (G, NC17, NR, PG, PG-13, R)
subgroup_names = ['G', 'NC17', 'NR', 'PG', 'PG-13', 'R'] *
len(group_names)
subgroup_size = []  # List to store the sizes for the ratings
subgroups

# Populate the subgroup_size list by extracting the rating counts for
each genre
for genre in group_names:
    subgroup_size.extend(genre_rating.loc[genre, ['G', 'NC17', 'NR',
'PG', 'PG-13', 'R']])

# Colors for the outer (Genres) and inner (Ratings) pie charts
outer_colors = plt.cm.viridis([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7])  #
Colors for genres
inner_colors = ['blue', 'purple', 'gray', 'orange', 'cyan', 'red'] *
len(group_names)  # Colors for ratings

# Create the nested pie chart
fig, ax = plt.subplots(figsize=(10, 10))  # Create a figure and axis
for the pie chart
ax.axis('equal')  # Ensure the pie chart is a circle

# Outer pie chart (Genres) - Displays the distribution of the top
genres
outer_pie, _ = ax.pie(group_size, radius=1.5, labels=group_names,
colors=outer_colors)
plt.setp(outer_pie, width=0.4, edgecolor='white')  # Set width and
edge color for the outer pie chart

# Inner pie chart (Ratings) - Displays the distribution of ratings
within each genre
inner_pie, _ = ax.pie(subgroup_size, radius=1.1, labels=None,
colors=inner_colors)
plt.setp(inner_pie, width=0.3, edgecolor='white')  # Set width and
edge color for the inner pie chart

# Add legend for the inner pie chart (Ratings)
ratings_labels = ['G', 'NC17', 'NR', 'PG', 'PG-13', 'R']  # Rating
labels
ratings_patches = [plt.Line2D([0], [0], color=color, lw=4) for color
in inner_colors[:6]]  # Create legend patches for ratings
plt.legend(ratings_patches, ratings_labels, title="Ratings",
```

```
loc="upper right", bbox_to_anchor=(1.3, 1))  # Add legend

# Display the chart
plt.margins(0, 0)  # Remove margins for better display
plt.show()  # Show the plot
```



Model Building

Loading the Dataset

```
audience_df = pd.read_csv('/content/Rotten_Tomatoes_Movies3.csv')
```

Sentiment Analysis on Critics' Consensus

```
# Importing TextBlob library for performing sentiment analysis
from textblob import TextBlob

# Creating a flag for rows where 'critics_consensus' is not null
```

```
audience_df['critics_flag'] =
audience_df['critics_consensus'].notna().astype(int)

# Define the sentiment analysis function using TextBlob
def perform_sentiment_analysis(text):
    # Using TextBlob to get the polarity of the text
    sentiment = TextBlob(text).sentiment.polarity  # Example: TextBlob
polarity score
    return sentiment

# Apply sentiment analysis to 'critics_consensus' only for rows where
'critics_flag' is 1
audience_df['sentiment_score'] = audience_df.apply(
    lambda row: perform_sentiment_analysis(row['critics_consensus'])
if row['critics_flag'] == 1 else None,
    axis=1
)
```

Checking Null Values

```
# checking for null values
audience_df.isnull().sum()
```

```
movie_title              0
movie_info              24
critics_consensus     8329
rating                   0
genre                   17
directors              114
writers               1349
cast                   284
in_theaters_date       815
on_streaming_date        2
runtime_in_minutes     155
studio_name            416
tomatometer_status       0
tomatometer_rating       0
tomatometer_count        0
audience_rating        252
critics_flag             0
sentiment_score       8329
dtype: int64
```

Handling Missing Values and Date Feature Engineering

```
# Fill NaN values in date columns with empty strings
audience_df['in_theaters_date'] =
audience_df['in_theaters_date'].fillna('')  # Replace NaN in
'in_theaters_date' with empty string
```

```
audience_df['on_streaming_date'] =
audience_df['on_streaming_date'].fillna('')  # Replace NaN in
'on_streaming_date' with empty string

# Fill other object columns with "<column_name>_null" to indicate
missing values
for col in audience_df.select_dtypes('object').columns:
    audience_df[col] = audience_df[col].fillna(f"{col}_null")  #
Replace NaN in object columns with a placeholder string indicating
null

# Convert the date columns to datetime format to enable date-related
calculations
audience_df['in_theaters_date'] =
pd.to_datetime(audience_df['in_theaters_date'], errors='coerce')  #
Convert 'in_theaters_date' to datetime
audience_df['on_streaming_date'] =
pd.to_datetime(audience_df['on_streaming_date'], errors='coerce')  #
Convert 'on_streaming_date' to datetime

# Extract year and month from the 'in_theaters_date' and
'on_streaming_date' columns
audience_df['in_theaters_year'] =
audience_df['in_theaters_date'].dt.year  # Extract year from
'in_theaters_date'
audience_df['in_theaters_month'] =
audience_df['in_theaters_date'].dt.month  # Extract month from
'in_theaters_date'
audience_df['on_streaming_year'] =
audience_df['on_streaming_date'].dt.year  # Extract year from
'on_streaming_date'
audience_df['on_streaming_month'] =
audience_df['on_streaming_date'].dt.month  # Extract month from
'on_streaming_date'

# Handle rows where dates could not be parsed (if any) by filling
missing days_to_streaming with -1
# audience_df['days_to_streaming'] =
audience_df['days_to_streaming'].fillna(-1).astype(int)  # Optionally,
calculate the number of days between the dates

# Display the resulting DataFrame to confirm changes
print(audience_df.head())  # Display the first few rows of the
modified DataFrame
```

```
                                         movie_title  \
0  Percy Jackson & the Olympians: The Lightning T...
1                                        Please Give
2                                                 10
3                    12 Angry Men (Twelve Angry Men)
```

```
4                        20,000 Leagues Under The Sea

                                           movie_info  \
0  A teenager discovers he's the descendant of a ...
1  Kate has a lot on her mind. There's the ethics...
2  Blake Edwards' 10 stars Dudley Moore as George...
3  A Puerto Rican youth is on trial for murder, a...
4  This 1954 Disney version of Jules Verne's 20,0...

                                   critics_consensus rating  \
0  Though it may seem like just another Harry Pot...     PG
1  Nicole Holofcener's newest might seem slight i...      R
2                             critics_consensus_null      R
3  Sidney Lumet's feature debut is a superbly wri...     NR
4  One of Disney's finest live-action adventures,...      G

                                                genre
directors  \
0  Action & Adventure, Comedy, Drama, Science Fic...     Chris
Columbus
1                                              Comedy  Nicole
Holofcener
2                                     Comedy, Romance      Blake
Edwards
3                                     Classics, Drama       Sidney
Lumet
4           Action & Adventure, Drama, Kids & Family  Richard
Fleischer

             writers
cast  \
0       Craig Titley  Logan Lerman, Brandon T. Jackson, Alexandra
Da...
1  Nicole Holofcener  Catherine Keener, Amanda Peet, Oliver Platt,
R...
2      Blake Edwards  Dudley Moore, Bo Derek, Julie Andrews,
Robert ...
3      Reginald Rose  Martin Balsam, John Fiedler, Lee J. Cobb,
E.G....
4        Earl Felton  James Mason, Kirk Douglas, Paul Lukas, Peter
L...

  in_theaters_date on_streaming_date  ...  tomatometer_status  \
0       2010-12-02        2010-06-29  ...              Rotten
1              NaT        2010-10-19  ...     Certified Fresh
2       1979-05-10        1997-08-27  ...               Fresh
3              NaT        2001-03-06  ...     Certified Fresh
4       1954-01-01        2003-05-20  ...               Fresh

  tomatometer_rating tomatometer_count  audience_rating  critics_flag
```

```
\
0                  49                144                53.0              1

1                  86                140                64.0              1

2                  68                 22                53.0              0

3                 100                 51                97.0              1

4                  89                 27                74.0              1


   sentiment_score  in_theaters_year  in_theaters_month
on_streaming_year  \
0         0.245833            2010.0               12.0
2010.0
1         0.055556               NaN                NaN
2010.0
2              NaN            1979.0                5.0
1997.0
3         0.491667               NaN                NaN
2001.0
4         0.322917            1954.0                1.0
2003.0

   on_streaming_month
0                 6.0
1                10.0
2                 8.0
3                 3.0
4                 5.0

[5 rows x 22 columns]

<ipython-input-36-f1f43c6f4145>:11: UserWarning: Parsing dates in %d-
%m-%Y format when dayfirst=False (the default) was specified. Pass
`dayfirst=True` or specify a format to silence this warning.
  audience_df['on_streaming_date'] =
pd.to_datetime(audience_df['on_streaming_date'], errors='coerce')  #
Convert 'on_streaming_date' to datetime
```

Dropping Unnecessary Columns from the DataFrame

```
# List of columns to drop from the DataFrame
drop_cols = ['movie_info', 'critics_consensus', 'critics_flag']  #
Columns that are no longer needed for analysis

# Drop the specified columns from the DataFrame
audience_df.drop(columns=drop_cols, axis=1, inplace=True)  # Removes
the columns from the DataFrame in-place
```

Removing Missing Values and Resetting Index

```
# Drop rows with missing values in the 'audience_rating' column
audience_df.dropna(subset=['audience_rating'], inplace=True)  #
Removes rows where 'audience_rating' is NaN

# Reset the index of the DataFrame after dropping rows
audience_df.reset_index(drop=True, inplace=True)  # Resets the index
and drops the old index
```

Analyzing the Number of Directors per Movie

```
# Calculate the number of directors for each movie
audience_df['num_directors'] = audience_df['directors'].apply(lambda x
: len(x.split(',')))  # Split by ',' and count

# Find the movie(s) with the highest number of directors
audience_df[audience_df['num_directors'] ==
audience_df['num_directors'].max()]  # Filter movie(s) with max
directors
```


```
{"summary":"{\n  \"name\": \"audience_df[audience_df['num_directors']
== audience_df['num_directors']]\",\n  \"rows\": 2,\n  \"fields\": [\n
{\n      \"column\": \"movie_title\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 2,\n
\"samples\": [\n          \"The ABCs of Death\",\n          \"ABCs of
Death 2\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"rating\",\n      \"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n          \"R\",\n
\"NR\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"genre\",\n      \"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n          \"Horror,
Mystery & Suspense\",\n          \"Horror\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"directors\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n
\"Christopher Smith (VIII), Angela Bettis, Simon Rumley, Bruno
Forzani, Kaare Andrews, Jason Eisener, Ernesto D\\u221a\\u2260az
Espinoza, Xavier Gens, Noboru Iguchi, Thomas Malling Cappelen, Jorge
Michel Grau, Anders Morgenthaler, Yoshihiro Nishimura, H\\u221a\\
u00a9l\\u221a\\u00aene Cattet, Banjong Pisanthanakun, Marcel
Sarmiento, Tak Sakaguchi, Adrian Garc\\u221a\\u2260a Bogliano, Jon
Schnepp, Srdjan Spasojevic, Timo Tjahjanto, Andrew Traucki, Nacho
Vigalondo, Jake West, Ti West, Ben Wheatley, Adam Wingard, Yudai
Yamaguchi, Lee Hardcastle, Simon Barrette, Simon Barrett\",\n
\"Ahron Keshales, Navot Papushado, Bill Plympton, Chris Nash, Dennison
Ramalho, Erik Matti, E.L. Katz, Hajime Ohata, Jen Soska, Sylvia Soska,
```

Jerome Sable, Jim Hosking, Juan Mart\\u221a\\u2260nez Moreno, Julian
Barratt, Julian Gilbey, Alexandre Bustillo, Julien Maury, Kristina
Buozyte, Bruno Samper, Lancelot Imasuen, Alejandro Brugu\\u221a\\
u00a9s, Larry Fessenden, Marvin Kren, Rob Boocheck, Robert Morgan,
Rodney Ascher, S\\u221a\\u00a5ichi Umezawa, Steven Kostanski, Todd
Rohal, Vincenzo Natali, Lancelot Oduwa Imasuen\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"writers\",\n   \"properties\":
{\n   \"dtype\": \"string\",\n   \"num_unique_values\": 1,\n
\"samples\": [\n   \"writers_null\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"cast\",\n   \"properties\": {\n
\"dtype\": \"string\",\n   \"num_unique_values\": 2,\n
\"samples\": [\n   \"Ingrid Bolso Berdal, Neil Maskell, Kyra
Zagorsky, Michael Smiley, Michael Rogers, Elisabeth Rosen\"\
n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"in_theaters_date\",\n   \"properties\": {\n   \"dtype\":
\"date\",\n   \"min\": \"2013-08-03 00:00:00\",\n   \"max\":
\"2013-08-03 00:00:00\",\n   \"num_unique_values\": 1,\n
\"samples\": [\n   \"2013-08-03 00:00:00\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"on_streaming_date\",\n
\"properties\": {\n   \"dtype\": \"date\",\n   \"min\":
\"2013-05-21 00:00:00\",\n   \"max\": \"2015-02-03 00:00:00\",\n
\"num_unique_values\": 2,\n   \"samples\": [\n   \"2013-
05-21 00:00:00\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"runtime_in_minutes\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 2.8284271247461903,\n   \"min\":
125.0,\n   \"max\": 129.0,\n   \"num_unique_values\": 2,\n
\"samples\": [\n   129.0\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"studio_name\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 2,\n   \"samples\": [\n
\"Magnolia Pictures\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"tomatometer_status\",\n   \"properties\": {\n   \"dtype\":
\"string\",\n   \"num_unique_values\": 2,\n   \"samples\":
[\n   \"Rotten\"\n   ],\n   \"semantic_type\": \"\",\
n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"tomatometer_rating\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 25,\n   \"min\": 37,\n
\"max\": 73,\n   \"num_unique_values\": 2,\n   \"samples\":
[\n   37\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"tomatometer_count\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 24,\n   \"min\": 33,\n

```
\"max\": 68,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          68\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"audience_rating\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 11.313708498984761,\n        \"min\":
23.0,\n        \"max\": 39.0,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          23.0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"sentiment_score\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.035355339059327355,\n
\"min\": -0.04999999999999999,\n        \"max\": -1.850371707708594e-
17,\n        \"num_unique_values\": 2,\n        \"samples\": [\n
-0.04999999999999999\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"in_theaters_year\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": null,\n        \"min\": 2013.0,\n
\"max\": 2013.0,\n        \"num_unique_values\": 1,\n
\"samples\": [\n          2013.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"in_theaters_month\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
null,\n        \"min\": 8.0,\n        \"max\": 8.0,\n
\"num_unique_values\": 1,\n        \"samples\": [\n          8.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"on_streaming_year\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1.4142135623730951,\n        \"min\": 2013.0,\n        \"max\":
2015.0,\n        \"num_unique_values\": 2,\n        \"samples\": [\n
2013.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"on_streaming_month\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 2.1213203435596424,\n        \"min\":
2.0,\n        \"max\": 5.0,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          5.0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"num_directors\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0,\n        \"min\": 31,\n
\"max\": 31,\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          31\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


Analyzing the Number of Writers per Movie

```
#Number of writers in each movie
audience_df['num_writers'] = audience_df['writers'].apply(lambda x :
len(x.split(',')))
# Maximum No. of writers
audience_df[audience_df['num_writers'] ==
audience_df['num_writers'].max()]
```

{"type":"dataframe"}

Analyzing the Number of casts per Movie

```
# Number of casts in each movie
audience_df['num_casts'] = audience_df['cast'].apply(lambda x :
len(x.split(',')))
# Maximum No.of Casts
audience_df[audience_df['num_casts'] ==
audience_df['num_casts'].max()]
```

{"type":"dataframe"}

Replacing Tomatometer Status with Numeric Values

```
# Replacing the string labels in 'tomatometer_status' with numeric
values
audience_df['tomatometer_status'].replace(['Rotten','Fresh','Certified
Fresh'], [0, 1, 2], inplace=True)
```

```
<ipython-input-42-9912a4507f03>:2: FutureWarning: A value is trying to
be set on a copy of a DataFrame or Series through chained assignment
using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.



audience_df['tomatometer_status'].replace(['Rotten','Fresh','Certified
Fresh'], [0, 1, 2], inplace=True)
<ipython-input-42-9912a4507f03>:2: FutureWarning: Downcasting behavior
in `replace` is deprecated and will be removed in a future version. To
retain the old behavior, explicitly call
`result.infer_objects(copy=False)`. To opt-in to the future behavior,
set `pd.set_option('future.no_silent_downcasting', True)`

audience_df['tomatometer_status'].replace(['Rotten','Fresh','Certified
Fresh'], [0, 1, 2], inplace=True)
```

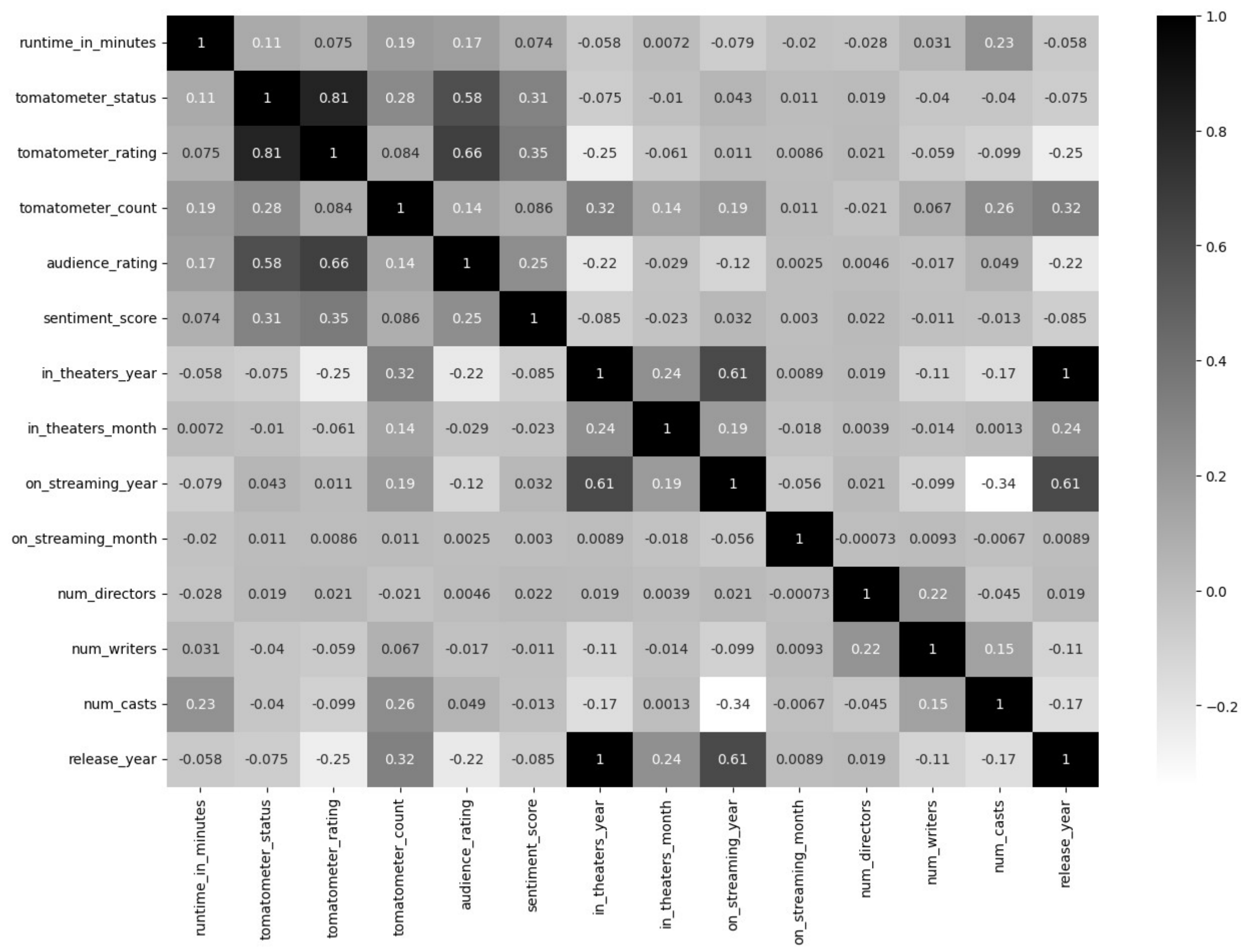
Converting 'in_theaters_date' to Datetime and Extracting Release Year

```
# Convert the 'in_theaters_date' column to datetime format,
considering day-first format
```

```python
audience_df['in_theaters_date'] =
pd.to_datetime(audience_df['in_theaters_date'], dayfirst=True)

# Extract the release year from the 'in_theaters_date' column
audience_df['release_year'] = audience_df['in_theaters_date'].dt.year
```

Correlation Matrix of Numerical Features

```python
# Calculate the correlation matrix for numeric columns in the dataset
corr = audience_df.select_dtypes('number').corr()  # Selects only
numerical columns and calculates their correlation

# Create a heatmap for visualizing the correlation matrix
plt.figure(figsize=(15, 10))  # Set the figure size for the heatmap
sns.heatmap(corr, annot=True, cmap='binary')  # Draw the heatmap with
annotations and 'binary' color map

<Axes: >
```

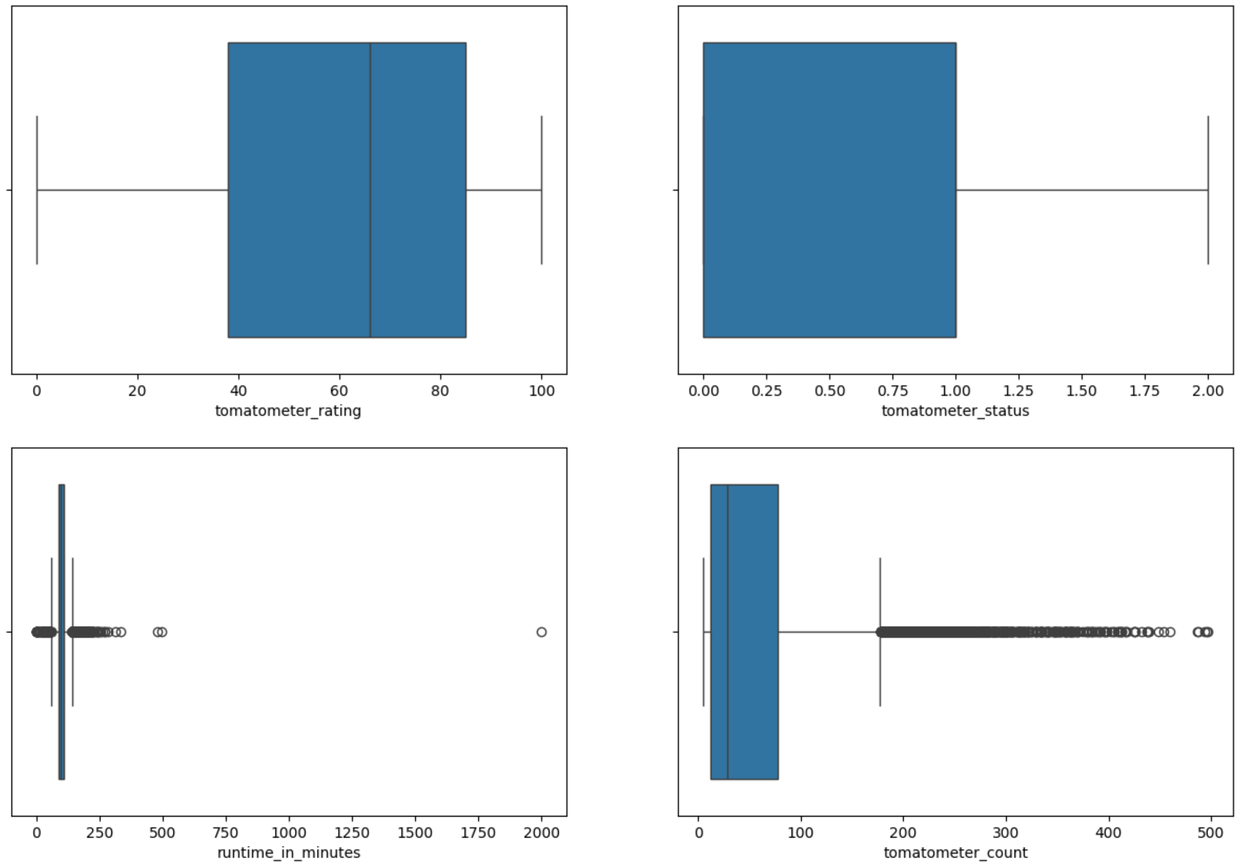


Correlation of Numerical Features with Audience Rating

```python
# Calculate the correlation between all numerical columns and the
target variable 'audience_rating'
audience_df.select_dtypes('number').corr()
['audience_rating'].sort_values(ascending=False)
```

```
audience_rating       1.000000
tomatometer_rating    0.660111
tomatometer_status    0.582228
sentiment_score       0.253643
runtime_in_minutes    0.168507
tomatometer_count     0.141012
num_casts             0.049081
num_directors         0.004587
on_streaming_month    0.002474
num_writers          -0.016563
in_theaters_month    -0.028938
on_streaming_year    -0.120884
in_theaters_year     -0.223488
release_year         -0.223488
Name: audience_rating, dtype: float64
```

```python
# Columns with high correlation
# checking for outliers

box_col =
['tomatometer_rating','tomatometer_status','runtime_in_minutes','tomat
ometer_count']
fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(15, 10))
ax = ax.flatten()
for index, value in enumerate(box_col):
    sns.boxplot(data=audience_df, x=value, ax=ax[index])
```



Outlier Removal Based on IQR (Interquartile Range)

```
# Creating masks to filter out outliers based on the IQR for each
relevant column
mask1 = audience_df['tomatometer_rating'] <
iqr(audience_df['tomatometer_rating']) * 1.5 +
np.percentile(audience_df['tomatometer_rating'], 75)
mask2 = audience_df['tomatometer_status'] <
iqr(audience_df['tomatometer_status']) * 1.5 +
np.percentile(audience_df['tomatometer_status'], 75)
mask3 = audience_df['runtime_in_minutes'] <
iqr(audience_df['runtime_in_minutes']) * 1.5 +
np.percentile(audience_df['runtime_in_minutes'], 75)
mask4 = audience_df['tomatometer_count'] <
iqr(audience_df['tomatometer_count']) * 1.5 +
np.percentile(audience_df['tomatometer_count'], 75)

# Combining all masks to filter out rows where any of the columns have
outliers
filtered_df = audience_df  # In this case, no rows are removed

# Print the shape of the original and filtered DataFrame
```

```
print("Original DataFrame shape:", audience_df.shape)
print("Filtered DataFrame shape:", filtered_df.shape)
```

```
Original DataFrame shape: (16386, 23)
Filtered DataFrame shape: (16386, 23)
```

Word2Vec for Cast Feature Preprocessing and Document Vector Creation

```
import numpy as np
from gensim.models import Word2Vec
# Step 1: Preprocess the 'Cast' feature to tokenize the cast names
# Here, the 'cast' feature is split by commas to get individual names
as tokens
audience_df['Cast_Tokens'] = audience_df['cast'].apply(lambda x:
x.split(','))

# Step 2: Train Word2Vec model on the tokens from the 'Cast_Tokens'
column
# Flatten the list of tokenized cast names to create training data for
Word2Vec
tokenized_cast = audience_df['Cast_Tokens'].tolist()
model = Word2Vec(sentences=tokenized_cast, vector_size=120, window=5,
min_count=1, workers=4)

# Step 3: Create document vectors for each movie
# The function calculates the average of the word vectors for the
tokens (cast names) in each document
def get_document_vector(tokens, model):
    # Fetch the vector for each token if it exists in the model's
vocabulary
    vectors = [model.wv[token] for token in tokens if token in
model.wv]
    # Return the average vector or a zero vector if no valid token
vectors exist
    if vectors:
        return np.mean(vectors, axis=0)
    else:
        return np.zeros(model.vector_size)  # Fallback to zero vector
for missing tokens

# Apply the function to create a vector for each document (movie's
cast)
audience_df['Cast_Vector'] = audience_df['Cast_Tokens'].apply(lambda
tokens: get_document_vector(tokens, model))

# Step 4: Transform the list of vectors into a DataFrame where each
column represents a component of the vector
# Each element of 'Cast_Vector' (which is a list) will be split into
individual columns
vector_df = pd.DataFrame(audience_df['Cast_Vector'].tolist(),
```

```python
columns=[f'vec_{i}' for i in range(model.vector_size)])

# Concatenate the vector DataFrame with the original DataFrame
(dropping the original cast-related columns)
audience_df = pd.concat([audience_df.drop(columns=['Cast_Vector']),
vector_df], axis=1)

# Drop unnecessary columns (cast and Cast_Tokens) after vectorization
audience_df.drop(columns=['cast', 'Cast_Tokens'], inplace=True,
axis=1)

# Final Dataset with the cast feature represented as vectors
audience_df
```

{"type":"dataframe","variable_name":"audience_df"}

```python
filtered_df=audience_df.copy()
```

Feature and Target Separation with Categorical and Numerical Column Identification

```python
# Step 1: Separate the target variable ('audience_rating') from the
features
# Copy the 'audience_rating' column as the target (y)
target = filtered_df['audience_rating'].copy()

# Drop the 'audience_rating' column from the features DataFrame (X)
filtered_df.drop('audience_rating', axis=1, inplace=True)

# Assign the target to 'y' and features to 'X'
y = target
X = filtered_df

# Step 2: Identify categorical and numerical columns in the features
DataFrame (X)
# Select columns of type 'object' (categorical data)
categorical_cols = X.select_dtypes(include=['object']).columns

# Select columns of type 'int64' or 'float64' (numerical data)
numerical_cols = X.select_dtypes(include=['int64', 'float64']).columns

# Step 3: Split the categorical columns that need special processing
(e.g., directors, genre, cast)
# These columns will be excluded from the general categorical
processing (handled separately)
categorical_split = ['directors', 'genre']

# Step 4: Final list of categorical columns excluding those that need
special handling
categorical_final = [i for i in categorical_cols if i not in
categorical_split]
```

```
# Now, 'categorical_final' contains the categorical columns that don't
require special handling
# 'categorical_split' holds columns like 'directors' and 'genre' that
may need separate preprocessing

print(y.shape)
audience_df.shape
```

```
(16386,)
```

```
(16386, 142)
```

```
print(f"Length of X: {len(X)}")
print(f"Length of y: {len(y)}")
```

```
Length of X: 16386
Length of y: 16386
```

Feature Engineering and Preprocessing with Stratified Split for Categorical and Numerical Columns

```
# Function to perform feature engineering on categorical columns
def feature_engineer_split(df, column_name, unique_vals):
    # Fill NaN values with 'null' to handle missing values
    df[column_name] = df[column_name].fillna('null')

    # Split string values into lists
    _list = df[column_name].str.split(",")

    # Create a dictionary for new columns based on unique values
    _cols = {f'{val}': [] for val in unique_vals}

    # Populate the dictionary with 1 (if value exists) or 0 (if value
doesn't exist)
    for row in _list:
        row_set = set(row) if row else set()
        for val in unique_vals:
            _cols[f'{val}'].append(1 if val in row_set else 0)

    # Convert the dictionary into a DataFrame with the same index as
the original DataFrame
    _df = pd.DataFrame(_cols, index=df.index)

    # Concatenate the new columns to the original DataFrame
    df = pd.concat([df, _df], axis=1)

    return df

# Loop through each column in 'categorical_split' to apply the feature
engineering
```

```
for column in categorical_split:
    print(f'Processing column: {column}')

    # Print the length of the DataFrame before processing
    print(f"Length of X before processing column {column}: {len(X)}")

    # Fill missing values with 'null'
    X[column] = X[column].fillna('null')

    # Split values and collect unique values
    _list = X[column].str.split(",")
    unique_vals = set()
    for row in _list:
        if row:  # Skip empty rows
            unique_vals.update(row)
    unique_vals = sorted(unique_vals)  # Sort unique values for
consistency

    # Apply feature engineering using the helper function
    X = feature_engineer_split(X, column, unique_vals)

    # Drop the original column after processing
    X.drop(columns=[column], inplace=True)

    # Print the length of the DataFrame after processing
    print(f"Length of X after processing column {column}: {len(X)}")
    print(f'Finished processing column: {column}')

# Update the list of categorical columns after feature engineering
categorical_cols = [col for col in
X.select_dtypes(include=['object']).columns if col not in
categorical_split]

# Preprocessing pipelines for numerical and categorical data
numerical_transformer = Pipeline(steps=[
    ('imputer', KNNImputer(n_neighbors=5)),  # Impute missing values
using KNN
    ('scaler', StandardScaler())  # Standardize numerical features
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),  # Impute
missing values using the most frequent value
    ('onehot', OneHotEncoder(handle_unknown='ignore'))  # One-hot
encode categorical features
])

# Column transformer to apply transformations to specific columns
preprocessor = ColumnTransformer(
    transformers=[
```

```
        ('num', numerical_transformer, numerical_cols),  # Apply
numerical transformer to numerical columns
        ('cat', categorical_transformer, categorical_cols)  # Apply
categorical transformer to categorical columns
    ])

# Stratified split: ensuring equal distribution of target values in
training and testing sets
if len(y.unique()) > 1:  # Check if the target has more than one
unique value
    x_train, x_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)
else:
    # If there's only one unique value in the target, perform a normal
split
    x_train, x_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

print("Data split completed successfully.")
```

```
Processing column: directors
Length of X before processing column directors: 16386
Length of X after processing column directors: 16386
Finished processing column: directors
Processing column: genre
Length of X before processing column genre: 16386
Length of X after processing column genre: 16386
Finished processing column: genre
Data split completed successfully.
```

```
from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor, AdaBoostRegressor
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, r2_score
```

1. **RandomForestRegressor**

- A tree-based ensemble method that combines multiple decision trees to improve prediction accuracy. It reduces the risk of overfitting and can handle large datasets effectively.

2. **GradientBoostingRegressor**

- An ensemble technique that builds trees sequentially, where each tree corrects the errors of the previous one. It is particularly useful for handling complex data with non-linear relationships.

3. **AdaBoostRegressor**

- A boosting algorithm that focuses on correcting the mistakes made by weak learners. It assigns more weight to instances that are difficult to predict and combines them into a stronger model.

4. **LinearRegression**

- A simple regression model that assumes a linear relationship between the dependent and independent variables. It is quick and easy to interpret but may not capture complex patterns.

5. **Ridge**

- A variant of linear regression with L2 regularization, which helps prevent overfitting by penalizing large coefficients. It is useful when there is multicollinearity in the data.

6. **Lasso**

- Similar to Ridge but uses L1 regularization, which forces some coefficients to be zero, effectively performing feature selection. It can help reduce the complexity of the model.

7. **DecisionTreeRegressor**

- A decision tree model that splits data into subsets based on feature thresholds. It is highly interpretable but can easily overfit, so it is often used in ensemble methods.

8. **SVR (Support Vector Regression)**

- A regression model based on Support Vector Machines. It tries to find the hyperplane that best fits the data while allowing some tolerance for error. It is effective in high-dimensional spaces.

```
# Dictionary to store different regression models
models = {
    'RandomForest': RandomForestRegressor(random_state=42),
    'GradientBoosting': GradientBoostingRegressor(random_state=42),
    'AdaBoost': AdaBoostRegressor(random_state=42),
    'LinearRegression': LinearRegression(),
    'Ridge': Ridge(),
    'Lasso': Lasso(),
    'DecisionTree': DecisionTreeRegressor(random_state=42),
    'SVR': SVR()
}
```

Model Evaluation and Selection Using MSE, MAE, and R-squared (R2) & Pipeline Demonstration

```
from sklearn.metrics import mean_absolute_error  # Import MAE function

# Initialize variables to track the best model and results
results = []
best_model = None
```

```
best_r2_score = -np.inf

# Iterate through each model to train and evaluate
for model_name, model in models.items():
    # Create a pipeline with preprocessing and the model regressor
    pipeline = Pipeline(steps=[
        ('preprocessor', preprocessor),  # Apply preprocessing
(scaling, imputation, etc.)
        ('regressor', model)  # Apply the regressor model
    ])

    # Train the model on the training data
    pipeline.fit(x_train, y_train)

    # Predict the target values for the test data
    y_pred = pipeline.predict(x_test)

    # Calculate Mean Squared Error (MSE) to evaluate model performance
    mse = mean_squared_error(y_test, y_pred)

    # Calculate Mean Absolute Error (MAE) to evaluate model
performance
    mae = mean_absolute_error(y_test, y_pred)

    # Calculate R-squared (R2) score to evaluate model fit
    r2 = r2_score(y_test, y_pred)

    # Print the evaluation metrics for the current model
    print(f'\n{model_name} Results:\nMean Squared Error (MSE):
{mse:.2f}\nMean Absolute Error (MAE): {mae:.2f}\nR-squared (R2) Score:
{r2:.2f}\n')

    # Store the results for each model in a list
    results.append({'Model': model_name, 'MSE': mse, 'MAE': mae, 'R2':
r2})

    # Track the best model based on the highest R2 score
    if r2 > best_r2_score:
        best_r2_score = r2
        best_model = pipeline


RandomForest Results:
Mean Squared Error (MSE): 202.55
Mean Absolute Error (MAE): 11.08
R-squared (R2) Score: 0.52


GradientBoosting Results:
Mean Squared Error (MSE): 197.90
Mean Absolute Error (MAE): 11.19
```

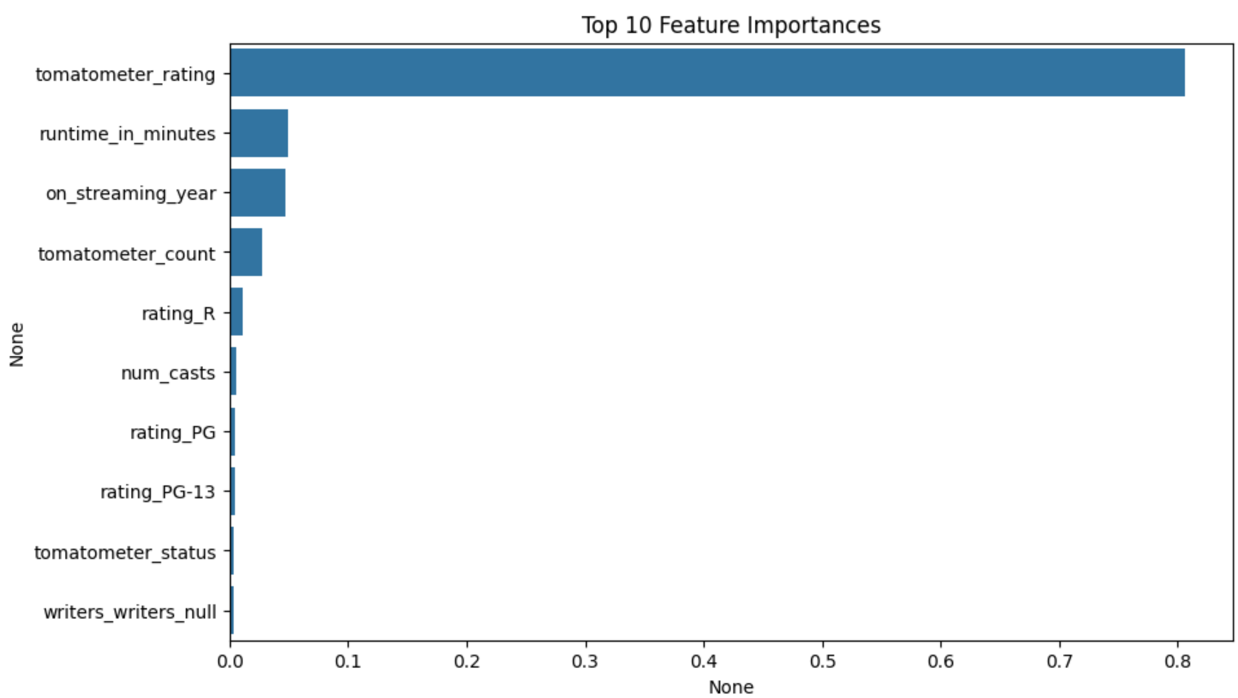
```
R-squared (R2) Score: 0.53


AdaBoost Results:
Mean Squared Error (MSE): 231.82
Mean Absolute Error (MAE): 12.61
R-squared (R2) Score: 0.45


LinearRegression Results:
Mean Squared Error (MSE): 235.85
Mean Absolute Error (MAE): 12.04
R-squared (R2) Score: 0.44


Ridge Results:
Mean Squared Error (MSE): 208.42
Mean Absolute Error (MAE): 11.32
R-squared (R2) Score: 0.50


Lasso Results:
Mean Squared Error (MSE): 224.07
Mean Absolute Error (MAE): 12.08
R-squared (R2) Score: 0.46


DecisionTree Results:
Mean Squared Error (MSE): 361.27
Mean Absolute Error (MAE): 14.73
R-squared (R2) Score: 0.14


SVR Results:
Mean Squared Error (MSE): 216.97
Mean Absolute Error (MAE): 11.36
R-squared (R2) Score: 0.48
```

Cross-validation and feature Importance for Model Evaluation

```
from sklearn.model_selection import cross_val_score
# Cross-validation for best model
cross_val_scores = cross_val_score(best_model, X, y, cv=5,
scoring='r2')
print(f'\nCross-validated R2 scores: {cross_val_scores}\nMean Cross-
validated R2 score: {np.mean(cross_val_scores):.2f}\n')

# Feature importance for tree-based models
if hasattr(best_model.named_steps['regressor'],
```

```
'feature_importances_'):
    importance =
best_model.named_steps['regressor'].feature_importances_
    feature_names = numerical_cols.tolist() +
list(best_model.named_steps['preprocessor'].transformers_[1]
[1].named_steps['onehot'].get_feature_names_out(categorical_cols))
    feature_importances = pd.Series(importance, index=feature_names)

    top_features =
feature_importances.sort_values(ascending=False).head(10)
    plt.figure(figsize=(10, 6))
    sns.barplot(x=top_features, y=top_features.index)
    plt.title('Top 10 Feature Importances')
    plt.show()
```

```
Cross-validated R2 scores: [0.55997872 0.52066642 0.50494501
0.52558321 0.51797633]
Mean Cross-validated R2 score: 0.53
```

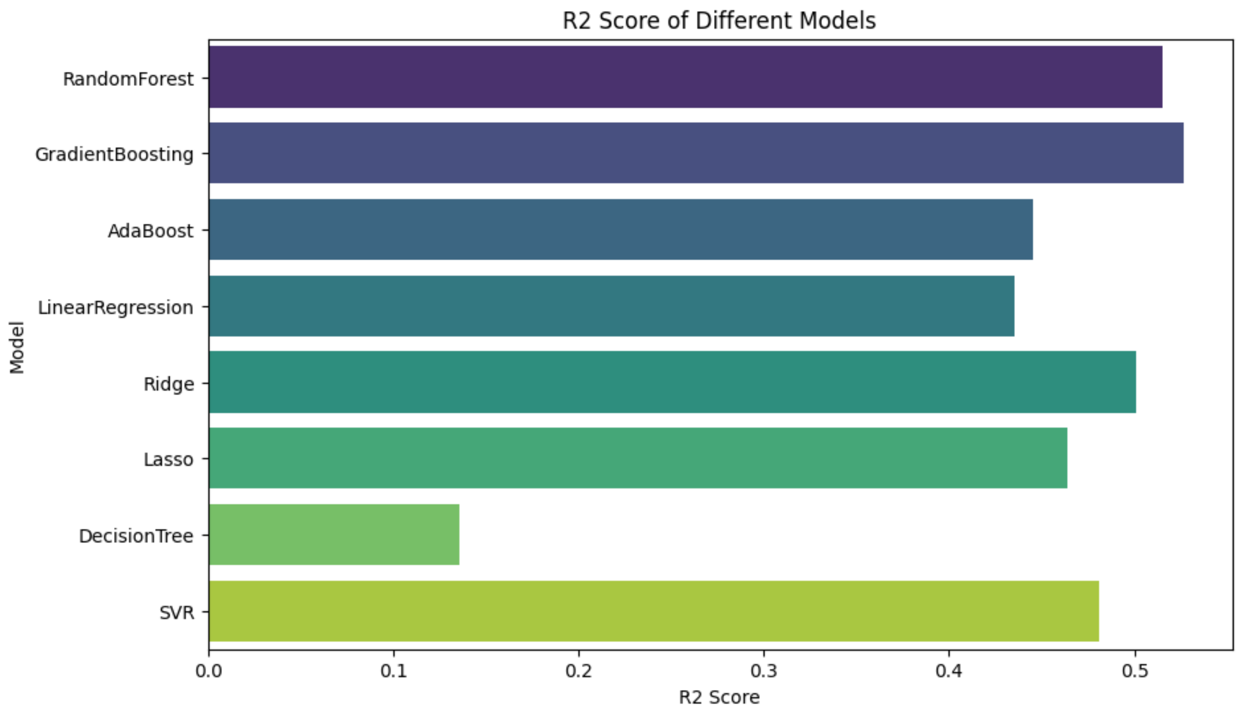


Summary of Model Performance and Best Model Identification

```
# Summary of Model Performance
results_df = pd.DataFrame(results)
plt.figure(figsize=(10, 6))
sns.barplot(x='R2', y='Model', data=results_df, palette='viridis')
plt.title('R2 Score of Different Models')
```

```
plt.xlabel('R2 Score')
plt.ylabel('Model')
plt.show()

# Print the best model based on R2 score
best_model_name =
best_model.named_steps['regressor'].__class__.__name__
best_model_r2 = best_r2_score

print("\nOverall Model Results:\n")
print(results_df.sort_values(by='R2', ascending=False))

print(f"\nBest Model: {best_model_name} with R2 Score:
{best_model_r2:.2f}")
```

```
<ipython-input-59-b75bcc350f93>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `y` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(x='R2', y='Model', data=results_df, palette='viridis')
```




```
Overall Model Results:

              Model         MSE        MAE        R2
1  GradientBoosting  197.899943  11.188252  0.526458
```

```
0      RandomForest  202.551073  11.075485  0.515328
4             Ridge  208.420286  11.323606  0.501284
7               SVR  216.971371  11.357742  0.480823
5             Lasso  224.074057  12.080760  0.463827
2          AdaBoost  231.816817  12.608415  0.445300
3  LinearRegression  235.854735  12.042916  0.435638
6      DecisionTree  361.269677  14.730323  0.135540

Best Model: GradientBoostingRegressor with R2 Score: 0.53
```

Conclusion

In this project, we evaluated multiple regression models to predict audience ratings based on a variety of movie features. The models tested include GradientBoosting, RandomForest, Ridge, SVR, Lasso, AdaBoost, LinearRegression, and DecisionTree.

After assessing the models on key performance metrics such as Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-squared (R2), we concluded the following:

- The **GradientBoostingRegressor** outperforms all other models with an R2 score of **0.53**.
- The **RandomForest** and **Ridge** models follow closely with R2 scores of **0.51** and **0.50**, respectively.
- The **DecisionTree** model performed poorly with an R2 score of just **0.14**, indicating that tree depth and feature interactions might not be adequately tuned.

Based on the performance metrics, the **GradientBoostingRegressor** is the best model for this task, showing the highest predictive accuracy for audience ratings. Further fine-tuning and model optimization can still enhance these results.