Identifying Top Films Using Box Office Data

Overview

Our company is preparing to create a new movie studio, so it is important to understand what kinds of films perform best. Exploratory data analysis insights derived from this project, will be translated into actionable recommendations for the new movie studio.

Business Understanding

This project analyzes Movie Datasets to uncover patterns in film performance. The goal is to support the company's entry into the film market by answering key business questions, such as:

- Which genres have the most production over the years?
- Which movie genres have the highest average ratings?
- Who are our competitors?
- Do movies that perform domestically perform internationally?

The insights will help the head of our company’s new movie studio to make informed decisions about what kinds of films to produce.

Data Understanding

The data used in this project comes from two popular film industry sources:

- `Box Office Mojo`
- `IMDb`

These datasets are stored in the `Data` folder. Since the files were collected from different sources, they vary in structure and formatting, hence I will be using pandas and SQLite3 to explore the datasets.

1. Data Exploration

With the project clearly explained, i will now load and explore the datasets to be used for this analysis.

1.1 Importing relevant libraries

In [1]:
```
# Import relevant libraries
import pandas as pd
import sqlite3
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

1.2 Load the IMdb Sqlite dataset and query to get the relevant tables

In [2]:
```
# Establish a connection to database

conn = sqlite3.connect('Data/im.db')

# Display the tables From the database using pandas

pd.read_sql("""SELECT  *
            FROM sqlite_master ;
            """, conn)
```

Out[2]:

	type	name	tbl_name	rootpage	sql
0	table	movie_basics	movie_basics	2	CREATE TABLE "movie_basics" (\n"movie_id" TEXT...
1	table	directors	directors	3	CREATE TABLE "directors" (\n"movie_id" TEXT,\n...
2	table	known_for	known_for	4	CREATE TABLE "known_for" (\n"person_id" TEXT,\...
3	table	movie_akas	movie_akas	5	CREATE TABLE "movie_akas" (\n"movie_id" TEXT,\...
4	table	movie_ratings	movie_ratings	6	CREATE TABLE "movie_ratings" (\n"movie_id" TEX...
5	table	persons	persons	7	CREATE TABLE "persons" (\n"person_id" TEXT,\n ...
6	table	principals	principals	8	CREATE TABLE "principals" (\n"movie_id" TEXT,\...
7	table	writers	writers	9	CREATE TABLE "writers" (\n"movie_id" TEXT,\n ...

Query and preview the tables that are relevant that is `movie_basics` and `movie_ratings` .

In [3]:
```
# Query the movie_basic table
pd.read_sql("""SELECT *
               FROM movie_basics
               LIMIT 10;
             """, conn)
```

Out[3]:

	movie_id	primary_title	original_title	start_year	runtime_minutes	genres
0	tt0063540	Sunghursh	Sunghursh	2013	175.0	Action,Crime,Drama
1	tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	114.0	Biography,Drama
2	tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018	122.0	Drama
3	tt0069204	Sabse Bada Sukh	Sabse Bada Sukh	2018	NaN	Comedy,Drama
4	tt0100275	The Wandering Soap Opera	La Telenovela Errante	2017	80.0	Comedy,Drama,Fantasy
5	tt0111414	A Thin Life	A Thin Life	2018	75.0	Comedy
6	tt0112502	Bigfoot	Bigfoot	2017	NaN	Horror,Thriller
7	tt0137204	Joe Finds Grace	Joe Finds Grace	2017	83.0	Adventure,Animation,Comedy
8	tt0139613	O Silêncio	O Silêncio	2012	NaN	Documentary,History
9	tt0144449	Nema aviona za Zagreb	Nema aviona za Zagreb	2012	82.0	Biography

In [4]:
```
# Query the movie_ratings table
pd.read_sql("""SELECT *
                FROM movie_ratings
                LIMIT 10;
            """, conn)
```

Out[4]:

	movie_id	averagerating	numvotes
0	tt10356526	8.3	31
1	tt10384606	8.9	559
2	tt1042974	6.4	20
3	tt1043726	4.2	50352
4	tt1060240	6.5	21
5	tt1069246	6.2	326
6	tt1094666	7.0	1613
7	tt1130982	6.4	571
8	tt1156528	7.2	265
9	tt1161457	4.2	148

Merge the two tables using the unique identifier `movie_id` present in both tables and call the new dataframe **`df_movies`** .

In [5]:
```
# Merge the two tables using Sql
df_movies = pd.read_sql("""SELECT *
               FROM movie_basics
               JOIN movie_ratings
               USING(movie_id);
            """, conn)

df_movies.head()
```

Out[5]:

	movie_id	primary_title	original_title	start_year	runtime_minutes	genres	ave
0	tt0063540	Sunghursh	Sunghursh	2013	175.0	Action,Crime,Drama	
1	tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	114.0	Biography,Drama	
2	tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018	122.0	Drama	
3	tt0069204	Sabse Bada Sukh	Sabse Bada Sukh	2018	NaN	Comedy,Drama	
4	tt0100275	The Wandering Soap Opera	La Telenovela Errante	2017	80.0	Comedy,Drama,Fantasy	

With the merged dataframe we can inspect the dataframe to check the summary, check null values and duplicates

In [6]:
```
# Check the summary of the dataframe
df_movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 73856 entries, 0 to 73855
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   movie_id         73856 non-null  object 
 1   primary_title    73856 non-null  object 
 2   original_title   73856 non-null  object 
 3   start_year       73856 non-null  int64  
 4   runtime_minutes  66236 non-null  float64
 5   genres           73052 non-null  object 
 6   averagerating    73856 non-null  float64
 7   numvotes         73856 non-null  int64  
dtypes: float64(2), int64(2), object(4)
memory usage: 4.5+ MB
```

The summary shows 73856 rows and 8 columns. The dataframe seems to have missing values.

Next we get an overview of the distributions of the data.

In [7]:
```
# Get descriptive statistics for numerical columns
print("Descriptive Statistics for numerical columns:")
df_movies.describe()
```

```
Descriptive Statistics for numerical columns:
```

Out[7]:

	start_year	runtime_minutes	averagerating	numvotes
count	73856.000000	66236.000000	73856.000000	7.385600e+04
mean	2014.276132	94.654040	6.332729	3.523662e+03
std	2.614807	208.574111	1.474978	3.029402e+04
min	2010.000000	3.000000	1.000000	5.000000e+00
25%	2012.000000	81.000000	5.500000	1.400000e+01
50%	2014.000000	91.000000	6.500000	4.900000e+01
75%	2016.000000	104.000000	7.400000	2.820000e+02
max	2019.000000	51420.000000	10.000000	1.841066e+06

In [8]:
```
# Check for missing values
print("Missing values per column:")
print(df_movies.isnull().sum())
```

```
Missing values per column:
movie_id              0
primary_title         0
original_title        0
start_year            0
runtime_minutes    7620
genres              804
averagerating         0
numvotes              0
dtype: int64
```

Two columns runtime_minutes and genres have missing values which we will clean during the cleaning step.

In [9]:
```
# Close DB connection
conn.close()
```

1.2 Load the Box Office CSV dataset and explore

In [10]:
```
# Load the CSV file using pandas
df = pd.read_csv("Data/bom.movie_gross.csv.gz")

# Display the dataframe
df
```

Out[10]:

	title	studio	domestic_gross	foreign_gross	year
0	Toy Story 3	BV	415000000.0	652000000	2010
1	Alice in Wonderland (2010)	BV	334200000.0	691300000	2010
2	Harry Potter and the Deathly Hallows Part 1	WB	296000000.0	664300000	2010
3	Inception	WB	292600000.0	535700000	2010
4	Shrek Forever After	P/DW	238700000.0	513900000	2010
...	...	...	...	...	...
3382	The Quake	Magn.	6200.0	NaN	2018
3383	Edward II (2018 re-release)	FM	4800.0	NaN	2018
3384	El Pacto	Sony	2500.0	NaN	2018
3385	The Swan	Synergetic	2400.0	NaN	2018
3386	An Actor Prepares	Grav.	1700.0	NaN	2018

3387 rows × 5 columns

We now inspect our dataframe to understand it by checking the summary, missing values and duplicates.

In [11]:
```
# Preview the  dataframe summary
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3387 entries, 0 to 3386
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   title           3387 non-null   object 
 1   studio          3382 non-null   object 
 2   domestic_gross  3359 non-null   float64
 3   foreign_gross   2037 non-null   object 
 4   year            3387 non-null   int64  
dtypes: float64(1), int64(1), object(3)
memory usage: 132.4+ KB
```

The dataframe has 3387 rows and 5 columns. The dataframe clearly has missing values.

In [12]:
```
# Check for missing values
df.isnull().sum()
```

Out[12]:
```
title                0
studio               5
domestic_gross      28
foreign_gross     1350
year                 0
dtype: int64
```

Three columns have missing values to be addressed during the cleaning part.

In [13]:
```
# Check for duplicates
df.duplicated().sum()
```

Out[13]:
```
0
```

2. Data Cleaning

This step is crucial as it enables cleaning of data by standardizing formats, handling missing values and handling outliers.Since there is two DataFrames, i will clean each one separately before merging them to a single Dataframe.

2.1 Cleaning the merged IMdb SQL DataFrame

In [14]:
```
# Create a copy of the df_movies to work on it
df_movie_copy = df_movies.copy()

# Display the summary
df_movie_copy.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 73856 entries, 0 to 73855
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   movie_id         73856 non-null  object 
 1   primary_title    73856 non-null  object 
 2   original_title   73856 non-null  object 
 3   start_year       73856 non-null  int64  
 4   runtime_minutes  66236 non-null  float64
 5   genres           73052 non-null  object 
 6   averagerating    73856 non-null  float64
 7   numvotes         73856 non-null  int64  
dtypes: float64(2), int64(2), object(4)
memory usage: 4.5+ MB
```

2.1.1 Handling Missing Values

Missing data can cause bias hence the need to accurately ensure the data has no NaNs

In [15]:
```
# Check for missing values in each column
df_movie_copy.isnull().sum()
```

Out[15]:
```
movie_id              0
primary_title         0
original_title        0
start_year            0
runtime_minutes    7620
genres              804
averagerating         0
numvotes              0
dtype: int64
```

The two columns `runtime_minutes` and `genres` have missing values which will be handled now. I will begin with the `runtime_minutes` column.

In [16]:
```
# Check the value_counts for the column
df_movie_copy['runtime_minutes'].value_counts()
```

Out[16]:
```
90.0     4742
80.0     2166
85.0     2057
100.0    1957
95.0     1933
         ...
212.0       1
278.0       1
225.0       1
467.0       1
746.0       1
Name: runtime_minutes, Length: 289, dtype: int64
```

Since majority of the films have a runtime of between 90 and 100 it is likely that i fill missing values with the median.

In [17]:
```
# Calculate the median
median_runtime = df_movie_copy['runtime_minutes'].median()

df_movie_copy['runtime_minutes'].fillna(median_runtime, inplace = True) #  Fil

# Check to see if there are still NaN values
df_movie_copy['runtime_minutes'].isnull().sum()
```

Out[17]:
```
0
```

Next is the `genres` column.

In [18]:
```
# Check the value_counts
df_movies['genres'].value_counts()
```

Out[18]:
```
Drama                          11612
Documentary                    10313
Comedy                          5613
Horror                          2692
Comedy,Drama                    2617
                               ...
Animation,Family,Mystery           1
Comedy,Music,Thriller              1
Adventure,Crime                    1
Comedy,Documentary,Western         1
Biography,Drama,Western            1
Name: genres, Length: 923, dtype: int64
```

Since the `genres` column is crucial for the analysis i will fill NaNs with ' `Unknown` ' to prevent loss of data.

In [19]:
```
# Fill missing values with Uknown
df_movie_copy['genres'].fillna('Unknown', inplace = True)

# Check if there is still missing values
df_movie_copy['genres'].isnull().sum()
```

Out[19]:
```
0
```

In [20]:
```
# Check if there is missing values in the dataframe
df_movie_copy.isnull().sum()
```

Out[20]:
```
movie_id           0
primary_title      0
original_title     0
start_year         0
runtime_minutes    0
genres             0
averagerating      0
numvotes           0
dtype: int64
```

2.1.2 Handling Duplicates

In [21]:
```
print(f"Number of exact duplicate rows: {df_movie_copy.duplicated().sum()}")

# If there are exact duplicates, drop them:
df_movie_copy.drop_duplicates(inplace = True)
```

```
Number of exact duplicate rows: 0
```

2.1.3 Check for Outliers

This will enabe us to see data points that significantly deviate from other observations.

In [22]:
```
# Check statistical summary for the numerical columns
df_movie_copy.describe()
```

Out[22]:

	start_year	runtime_minutes	averagerating	numvotes
count	73856.000000	73856.000000	73856.000000	7.385600e+04
mean	2014.276132	94.277039	6.332729	3.523662e+03
std	2.614807	197.524557	1.474978	3.029402e+04
min	2010.000000	3.000000	1.000000	5.000000e+00
25%	2012.000000	83.000000	5.500000	1.400000e+01
50%	2014.000000	91.000000	6.500000	4.900000e+01
75%	2016.000000	101.000000	7.400000	2.820000e+02
max	2019.000000	51420.000000	10.000000	1.841066e+06

The first column is the `start_year` .

In [23]:
```
# Check Outliers in start_year to see years that are not possible
print("Outliers in 'start_year' column:")
print(f"Min year: {df_movie_copy['start_year'].min()}")
print(f"Max year: {df_movie_copy['start_year'].max()}")
```

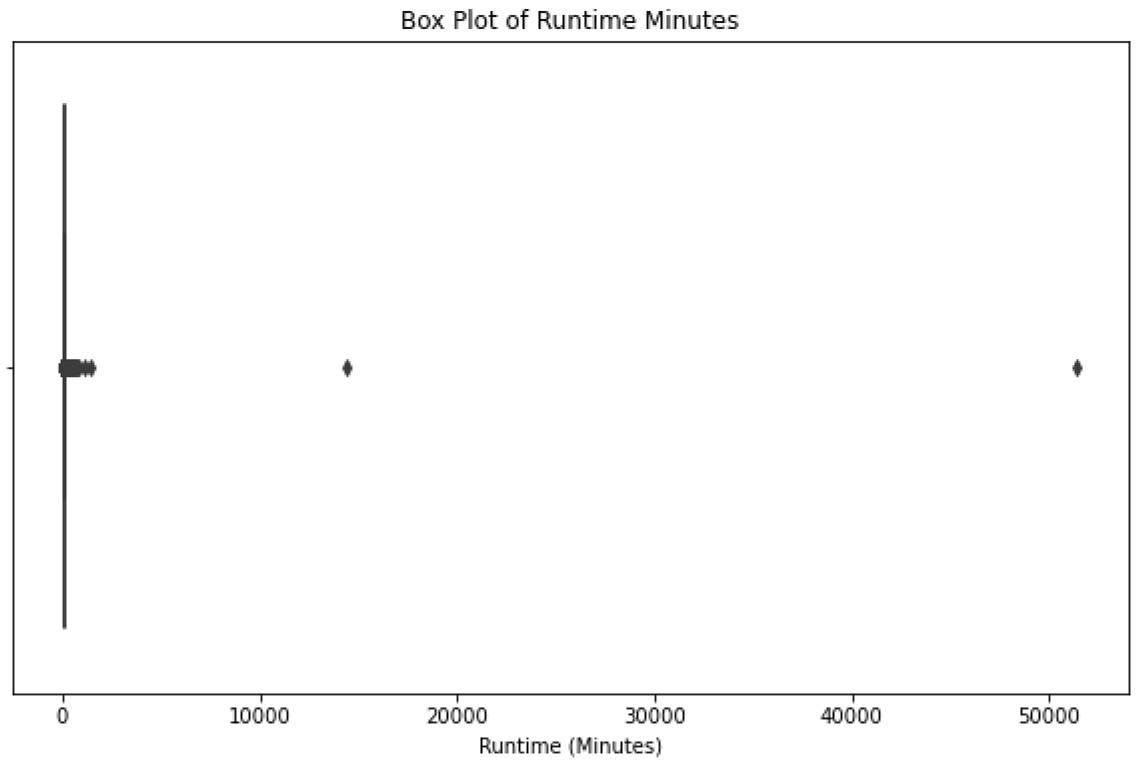
```
Outliers in 'start_year' column:
Min year: 2010
Max year: 2019
```

The years recorded is valid.

We move to `runtime_minutes` column

In [24]:
```
# Outliers in runtime_minutes
print(df_movie_copy['runtime_minutes'].describe())

# Plot a box plot to visualize
plt.figure(figsize = (10, 6))
sns.boxplot(x = df_movie_copy['runtime_minutes'])
plt.title('Box Plot of Runtime Minutes')
plt.xlabel('Runtime (Minutes)')
plt.show()
```

```
count    73856.000000
mean        94.277039
std        197.524557
min          3.000000
25%         83.000000
50%         91.000000
75%        101.000000
max      51420.000000
Name: runtime_minutes, dtype: float64
```



The maximum runtime looks like an abnormal runtime so i will handle the outliers in this column.

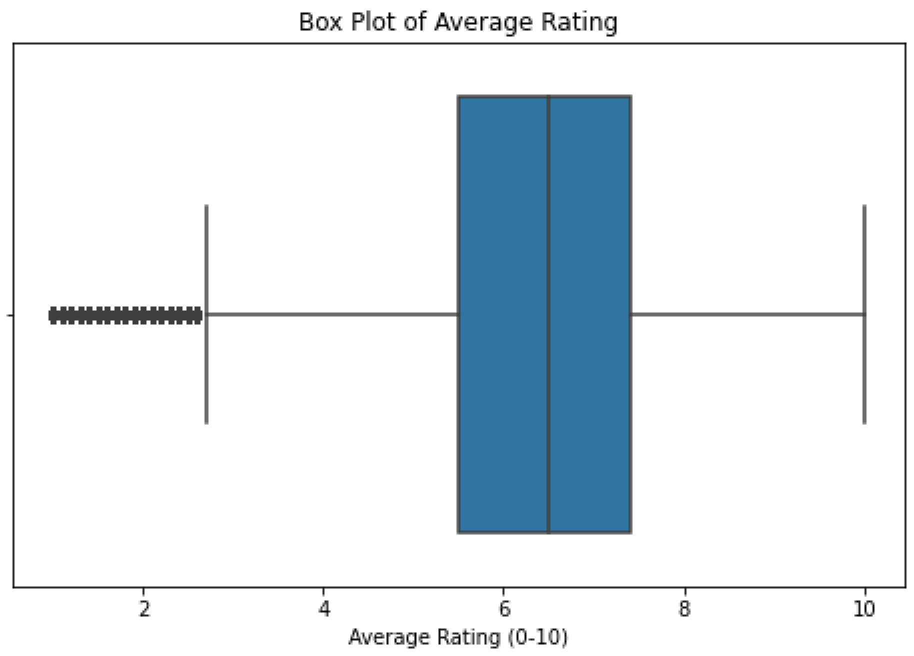
In [25]:
```
# Remove runtime_minutes above 300 mins
df_movie_copy = df_movie_copy[df_movie_copy['runtime_minutes'] <= 300]

df_movie_copy['runtime_minutes'].value_counts()
```

Out[25]:
```
91.0     8893
90.0     4742
80.0     2166
85.0     2057
100.0    1957
         ... 
254.0       1
287.0       1
223.0       1
202.0       1
248.0       1
Name: runtime_minutes, Length: 252, dtype: int64
```

Next is the `averagerating` column

In [26]:
```python
# Outliers in averagerating
print(df_movie_copy['averagerating'].describe())

plt.figure(figsize = (8, 5))
sns.boxplot(x = df_movie_copy['averagerating'])
plt.title('Box Plot of Average Rating')
plt.xlabel('Average Rating (0-10)')
plt.show()
```

```
count    73813.000000
mean         6.332096
std          1.474822
min          1.000000
25%          5.500000
50%          6.500000
75%          7.400000
max         10.000000
Name: averagerating, dtype: float64
```



This column is pretty okay and also the `numvotes` column seems to have films that have many votes so i will not handle outliers.

2.2 Cleaning the CSV dataset

This is the box office dataset- `bom.movie_gross.csv.gz` located in the Data folder.

In [27]:
```
# Create a copy of the df
df_bom =  df.copy()

# Preview the summary
df_bom.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3387 entries, 0 to 3386
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   title           3387 non-null   object 
 1   studio          3382 non-null   object 
 2   domestic_gross  3359 non-null   float64
 3   foreign_gross   2037 non-null   object 
 4   year            3387 non-null   int64  
dtypes: float64(1), int64(1), object(3)
memory usage: 132.4+ KB
```

2.2.1 Handling Missing values

3 columns namely `studio` , `domestic_gross` , `foreign_gross` have missing values which need to be dealt with. I will begin with `studio` column.

In [28]:
```
# Print value_counts for the studio column
print(df_bom['studio'].describe())
```

```
count     3382
unique     257
top        IFC
freq       166
Name: studio, dtype: object
```

In [29]:
```
# Since it’s a categorical column, i will Fill with "Unknown"
df_bom['studio'].fillna("Unknown", inplace = True)

# Check the Dataframe
df_bom.isnull().sum()
```

Out[29]:
```
title                0
studio               0
domestic_gross      28
foreign_gross     1350
year                 0
dtype: int64
```

Assuming no domestic earnings reported i will fill the `domestic_gross` column with 0.

In [30]:
```
# Fill NaNs with 0
df_bom['domestic_gross'].fillna(0, inplace = True)

# Check the Dataframe
df_bom.isnull().sum()
```

Out[30]:
```
title                0
studio               0
domestic_gross       0
foreign_gross     1350
year                 0
dtype: int64
```

`foreign_gross` column has a lot of missing values. Best to fill with 0, to avoid overestimating earnings.

In [31]:
```
# Convert foreign_gross to numeric
df_bom['foreign_gross'] = pd.to_numeric(df_bom['foreign_gross'], errors = 'coe

df_bom['foreign_gross'].fillna(0, inplace = True)

# Check the Dataframe
df_bom.isnull().sum()
```

Out[31]:
```
title             0
studio            0
domestic_gross    0
foreign_gross     0
year              0
dtype: int64
```

2.2.2 Handling Duplicates

In [32]:
```
# Check duplicates

print(f"Number of exact duplicate rows: {df_bom.duplicated().sum()}")

# If there are exact duplicates, drop them:
print(df_bom.drop_duplicates(inplace = True))
```

```
Number of exact duplicate rows: 0
None
```

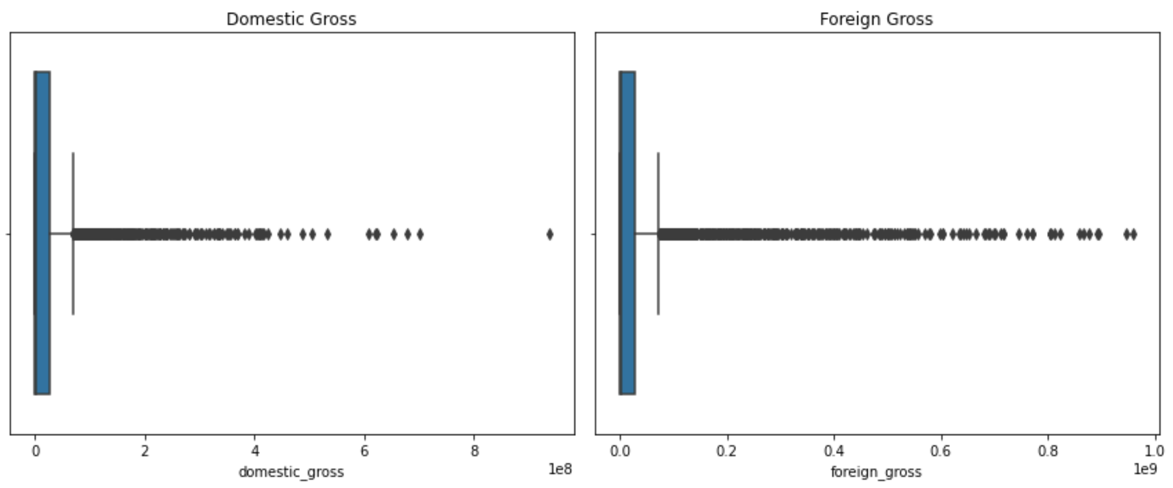
2.2.3 Checking Outliers

In [33]:
```
# Check statistical summary
df_bom.describe()
```

Out[33]:

	domestic_gross	foreign_gross	year
count	3.387000e+03	3.387000e+03	3387.000000
mean	2.850821e+07	4.502979e+07	2013.958075
std	6.675575e+07	1.126843e+08	2.478141
min	0.000000e+00	0.000000e+00	2010.000000
25%	1.115000e+05	0.000000e+00	2012.000000
50%	1.300000e+06	1.500000e+06	2014.000000
75%	2.750000e+07	2.915000e+07	2016.000000
max	9.367000e+08	9.605000e+08	2018.000000

In [34]:
```
# Plot boxplots
plt.figure(figsize = (12, 5))

plt.subplot(1, 2, 1)
sns.boxplot(x = df_bom['domestic_gross'])
plt.title('Domestic Gross')

plt.subplot(1, 2, 2)
sns.boxplot(x = df_bom['foreign_gross'])
plt.title('Foreign Gross')

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



While some extreme values exist in the gross data, I will retain as they represent major box office successes crucial to our business insights.

2.3 Feature Engineering

The `df_movies` DataFrame extracted from the database

In [35]:
```
# Bin runtime_minutes
# Define bins and labels
runtime_bins = [0, 90, 120, df_movie_copy['runtime_minutes'].max()]
runtime_labels = ['Short', 'Medium', 'Long']

# Apply binning
df_movie_copy['runtime_category'] = pd.cut(df_movie_copy['runtime_minutes'], b
```

In [36]:
```
# bin average ratings
# Define bins and labels
bins = [0, 6, 8, 10]
labels = ['Low', 'Medium', 'High']

# Apply binning
df_movie_copy['rating_category'] = pd.cut(df_movie_copy['averagerating'], bins

# Check df summary
df_movie_copy.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 73813 entries, 0 to 73855
Data columns (total 10 columns):
 #   Column            Non-Null Count  Dtype   
---  ------            --------------  -----   
 0   movie_id          73813 non-null  object  
 1   primary_title     73813 non-null  object  
 2   original_title    73813 non-null  object  
 3   start_year        73813 non-null  int64   
 4   runtime_minutes   73813 non-null  float64 
 5   genres            73813 non-null  object  
 6   averagerating     73813 non-null  float64 
 7   numvotes          73813 non-null  int64   
 8   runtime_category  73813 non-null  category
 9   rating_category   73813 non-null  category
dtypes: category(2), float64(2), int64(2), object(4)
memory usage: 5.2+ MB
```

The `bom_movies` CSV

In [37]:
```
# Create total_gross column
df_bom['total_gross'] = df_bom['domestic_gross'].fillna(0) + df_bom['foreign_g

# check summary
df_bom.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3387 entries, 0 to 3386
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   title           3387 non-null   object 
 1   studio          3387 non-null   object 
 2   domestic_gross  3387 non-null   float64
 3   foreign_gross   3387 non-null   float64
 4   year            3387 non-null   int64  
 5   total_gross     3387 non-null   float64
dtypes: float64(3), int64(1), object(2)
memory usage: 185.2+ KB
```

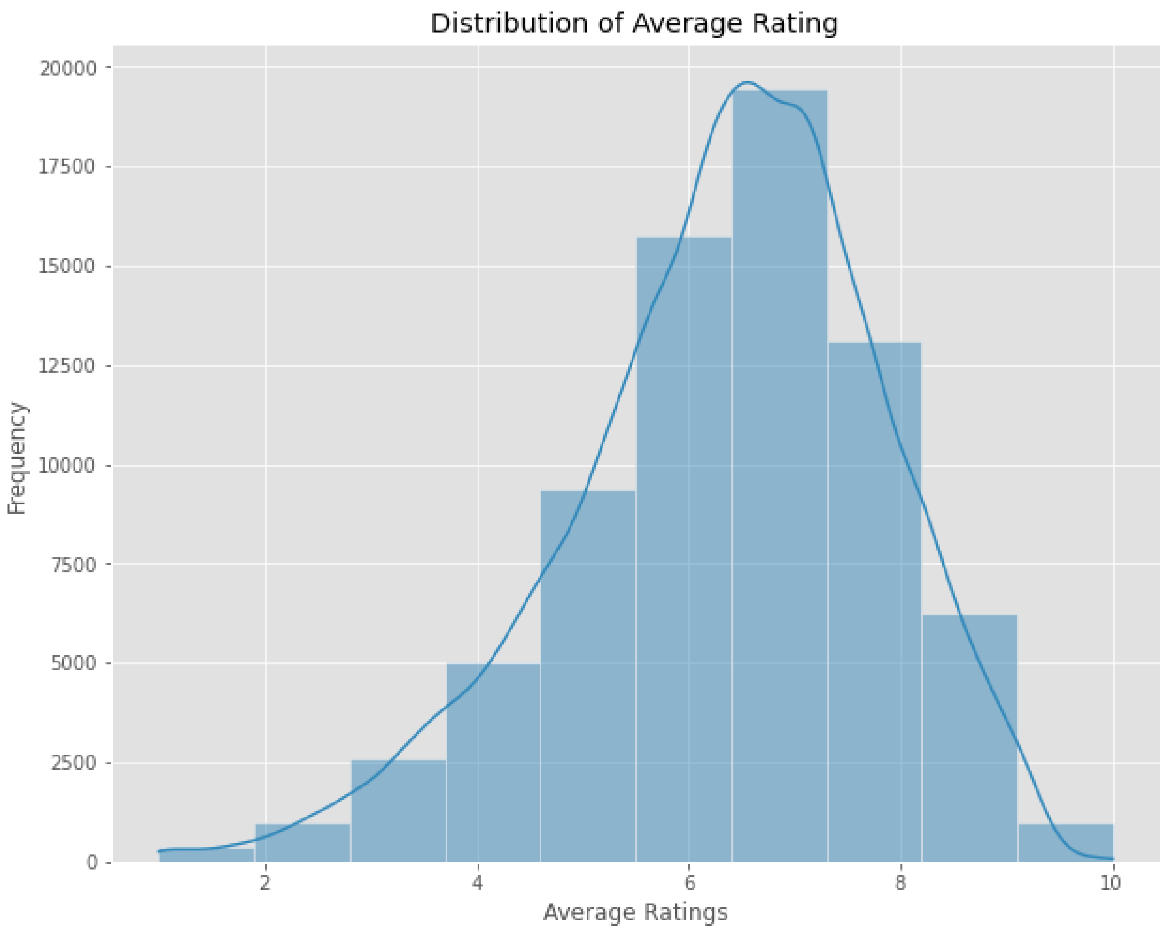
3. Exploratory Data Analysis

In this section, we perform Exploratory Data Analysis (EDA) to understand the patterns and relationships within our datasets.

3.1 Univariate Analysis

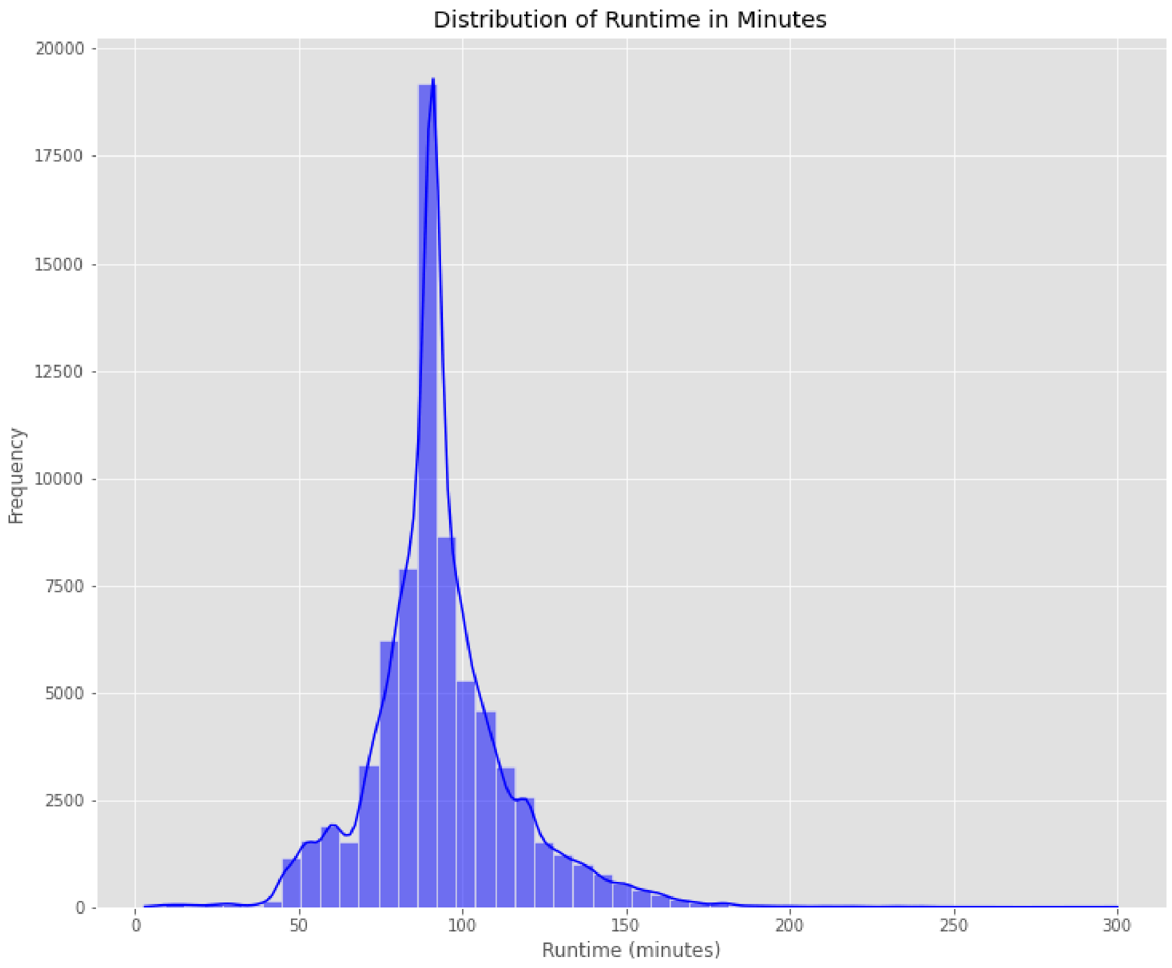
We start with univariate analysis to look at each column on its own and understand the basic patterns in the data.

In [38]:
```
# Plot to see the Distributio of average ratings of genres in our df_movie_cop
plt.figure(figsize = (10, 8))

plt.style.use('ggplot')

sns.histplot(df_movie_copy['averagerating'], bins = 10, kde = True) # Histogra
plt.title('Distribution of Average Rating')
plt.xlabel('Average Ratings')
plt.ylabel('Frequency')
plt.show()
```



Most ratings are concentrated in the 5.5 – 8.0 range. The tail on the left includes some movies with very low ratings below 5.

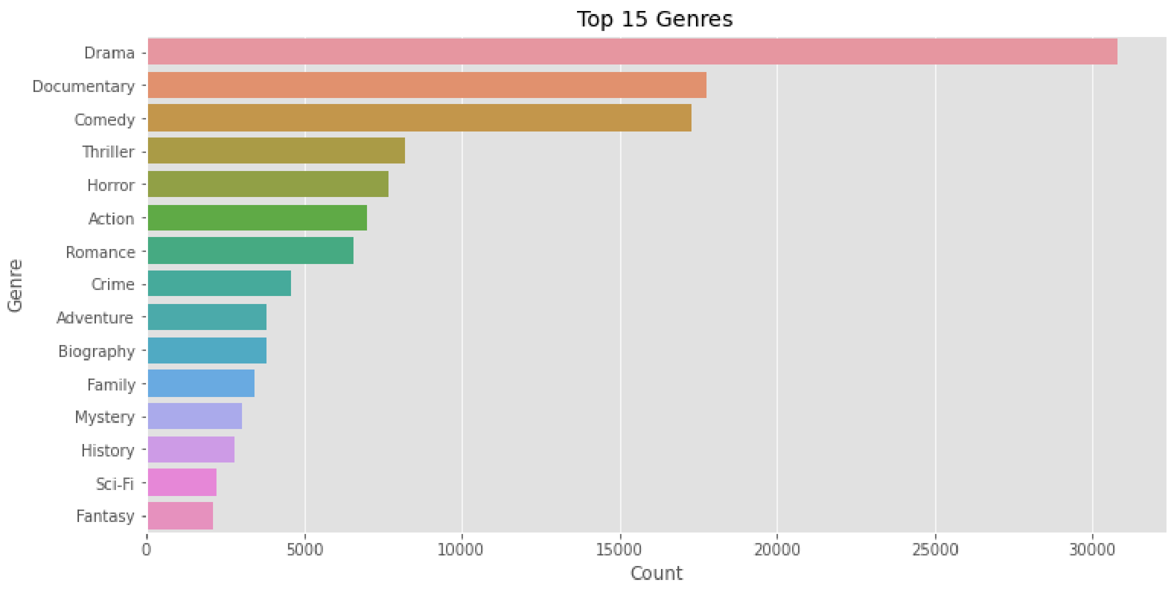
```
In [39]: # Plot to see the average movie minutes mostly watched
         plt.figure(figsize = (12, 10))

         sns.histplot(df_movie_copy['runtime_minutes'], bins = 50,color = 'blue', kde =
         plt.title("Distribution of Runtime in Minutes")
         plt.xlabel("Runtime (minutes)")
         plt.ylabel("Frequency")
         plt.show()
```



The mean lies at 90 mins and most watched movies have a runtime minutes of between 80 to 120 minutes.

For the genres, I exploded the `genres` column since a movie can belong to multiple genres hence need to analyze each genre separately. This helped to get clearer insights on genre trends, frequency, and performance.

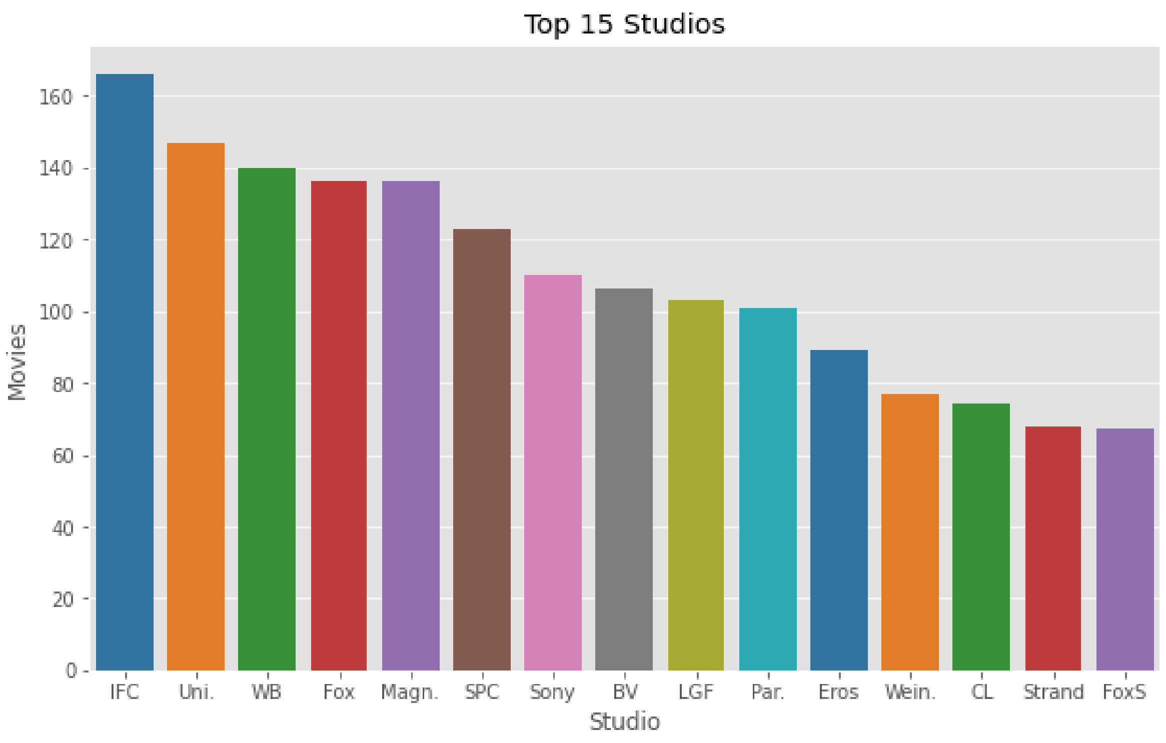
In [40]:
```
# Exploding genres columns
df_exploded = df_movies.copy() # Make a copy of the dataframe
df_exploded['genres'] = df_exploded['genres'].str.split(',') # splits the comm

df_exploded = df_exploded.explode('genres') # Explode the list so each genre g

plt.figure(figsize = (12, 6))
sns.countplot(data = df_exploded, y = 'genres', order = df_exploded['genres'].
plt.title("Top 15 Genres")
plt.xlabel("Count")
plt.ylabel("Genre")
plt.show()
```



Drama is the most popular, then Documentaries and Comedy follow in the top 3 with over 15000. Fantasy and Sci-Fi have less than 5000 counts.

Next is to see the top studios using the box office dataset.

In [41]:
```
# Get top 15 studios by count
top_studios = df_bom['studio'].value_counts().head(15).index

# Filter the dataset to only include those top studios
filtered_bom = df_bom[df_bom['studio'].isin(top_studios)]

plt.figure(figsize = (10,6))
sns.countplot(data = filtered_bom, x = 'studio', order = top_studios, palette
plt.title("Top 15 Studios")
plt.xlabel("Studio")
plt.ylabel("Movies")

plt.show()
```



IFC(Independent Film Company), Universal(uni), Warner Bros(WB) are in the top 3 best performing studios.

3.2 Bivariate Analysis

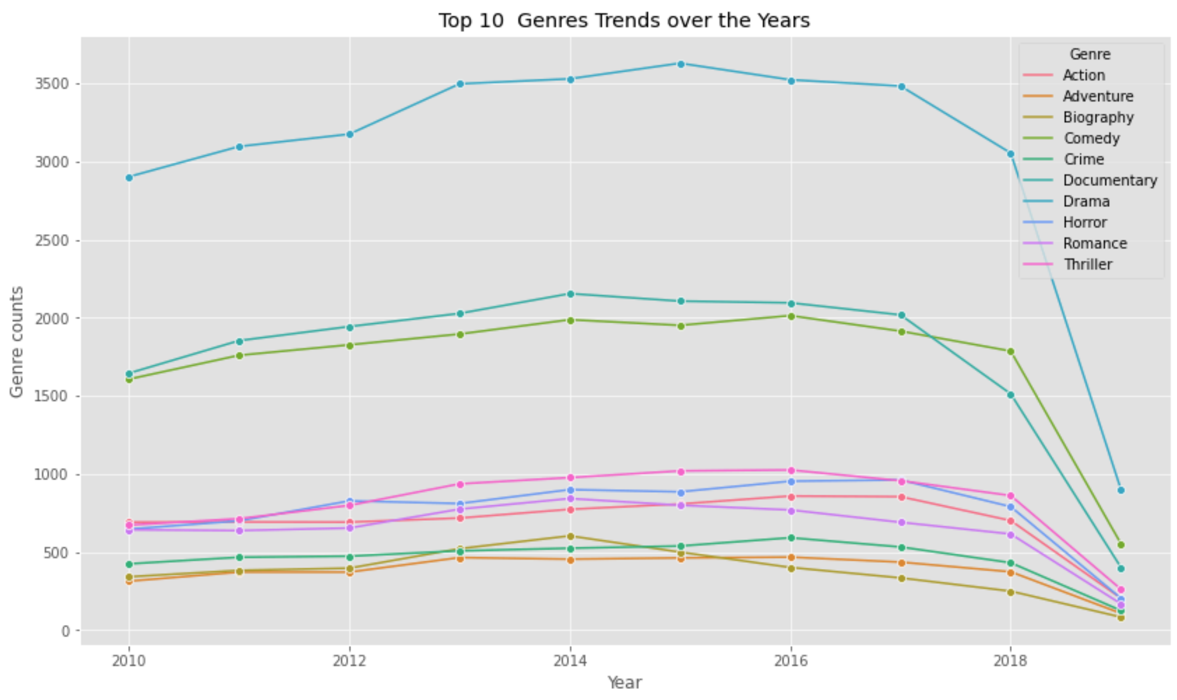
Here, I will explore how two variables interact to uncover patterns that inform box office performance.

We begin by asking which genres have been the most produced over the years?

In [42]:
```
# Group by year and genre, count how many movies
genre_year_counts = df_exploded.groupby(['start_year', 'genres']).size().reset

# top 10 genres
top_genres = genre_year_counts.groupby('genres')['count'].sum().nlargest(10).i

print(top_genres)


# Filter for only top genres
top_10 = genre_year_counts[genre_year_counts['genres'].isin(top_genres)]
```

```
Index(['Drama', 'Documentary', 'Comedy', 'Thriller', 'Horror', 'Action',
       'Romance', 'Crime', 'Adventure', 'Biography'],
      dtype='object', name='genres')
```

In [43]:
```
# Plot
plt.figure(figsize = (14, 8))
sns.lineplot(data = top_10, x = 'start_year', y = 'count', hue = 'genres', mar
plt.title('Top 10  Genres Trends over the Years')
plt.xlabel('Year')
plt.ylabel('Genre counts')
plt.legend(title = 'Genre')

plt.savefig('Images/Trends_year.png', dpi=300, bbox_inches='tight')
plt.show()
```



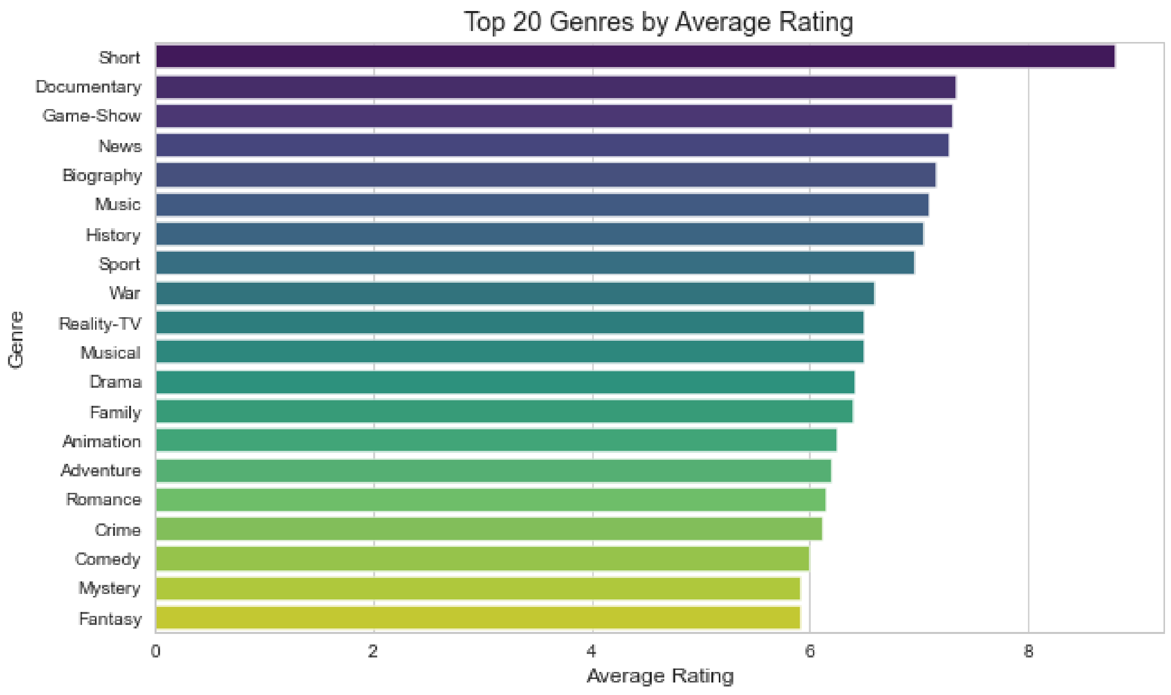
Drama, Documentary and comedy genres have been top performing produced over the years.

Next is which movie genres consistently receive the highest average ratings?

In [44]:
```
# Group by individual genre and calculate average ratings then sort descending
genre_ratings = df_exploded.groupby('genres')['averagerating'].mean().sort_va

# Plot top 20 genres
top20 = genre_ratings.head(20)
print(top20)
```

```
genres
Short          8.800000
Documentary    7.332090
Game-Show      7.300000
News           7.271330
Biography      7.162274
Music          7.091972
History        7.040956
Sport          6.961493
War            6.584291
Reality-TV     6.500000
Musical        6.498336
Drama          6.401559
Family         6.394725
Animation      6.248308
Adventure      6.196201
Romance        6.146608
Crime          6.115441
Comedy         6.002689
Mystery        5.920401
Fantasy        5.919473
Name: averagerating, dtype: float64
```

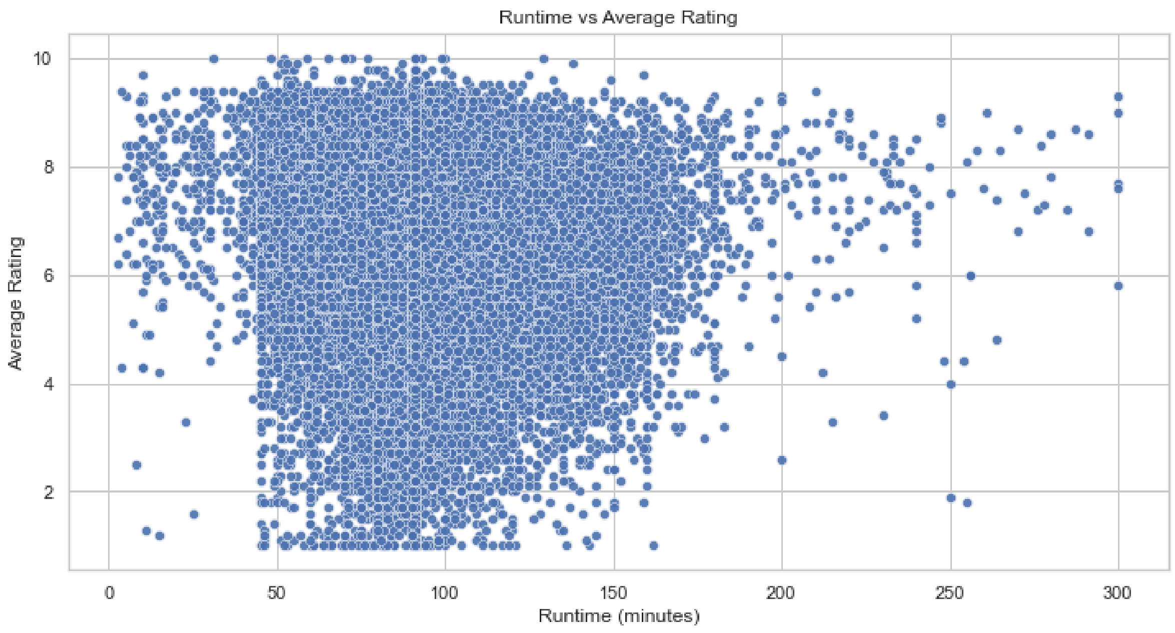
In [45]:
```
# Plot
plt.figure(figsize = (10, 6))
sns.set_style("whitegrid")
sns.barplot(x = top20.values, y = top20.index, palette = 'viridis')
plt.title('Top 20 Genres by Average Rating')
plt.xlabel('Average Rating')
plt.ylabel('Genre')

plt.savefig('Images/top_genres.png', dpi=300, bbox_inches='tight')
plt.show()
```



Short genres has the highest rating of 8.8. Documentaries and Game-show have 7.3. News also in the top 3 ratings.

We now look at how runtime in minutes relates to average ratings.

```
In [46]: # Runtime vs Average Rating
plt.figure(figsize = (12, 6))

sns.set(style = "whitegrid")

sns.scatterplot(data = df_movie_copy, x = 'runtime_minutes', y = 'averageratin
plt.title("Runtime vs Average Rating")
plt.xlabel("Runtime (minutes)")
plt.ylabel("Average Rating")

plt.show()
```



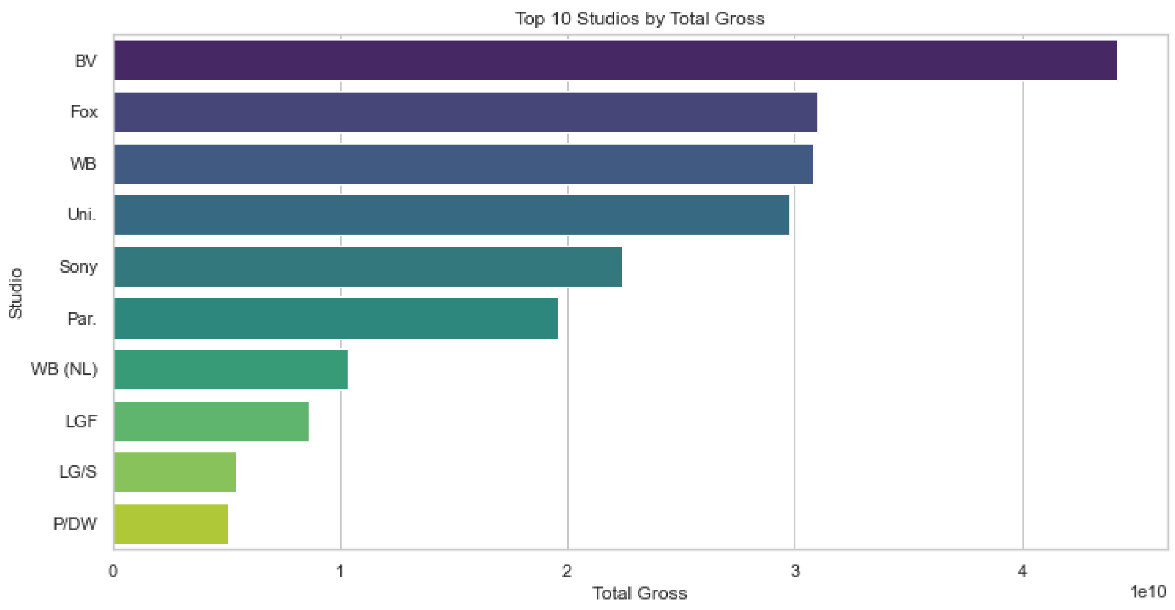
There's a dense cluster of movies with runtimes between roughly 80–120 minutes, where most ratings lie between 5 and 8.

We now answer which are the top 10 studios based on total gross revenue?

In [47]:
```
# Top 10 Studios by Total Gross Revenue
top_total = df_bom.groupby('studio')['total_gross'].sum().sort_values(ascendin

print(top_total)
```

```
studio
BV          4.421288e+10
Fox         3.100537e+10
WB          3.083595e+10
Uni.        2.975716e+10
Sony        2.240504e+10
Par.        1.954926e+10
WB (NL)     1.033470e+10
LGF         8.601583e+09
LG/S        5.431924e+09
P/DW        5.076500e+09
Name: total_gross, dtype: float64
```

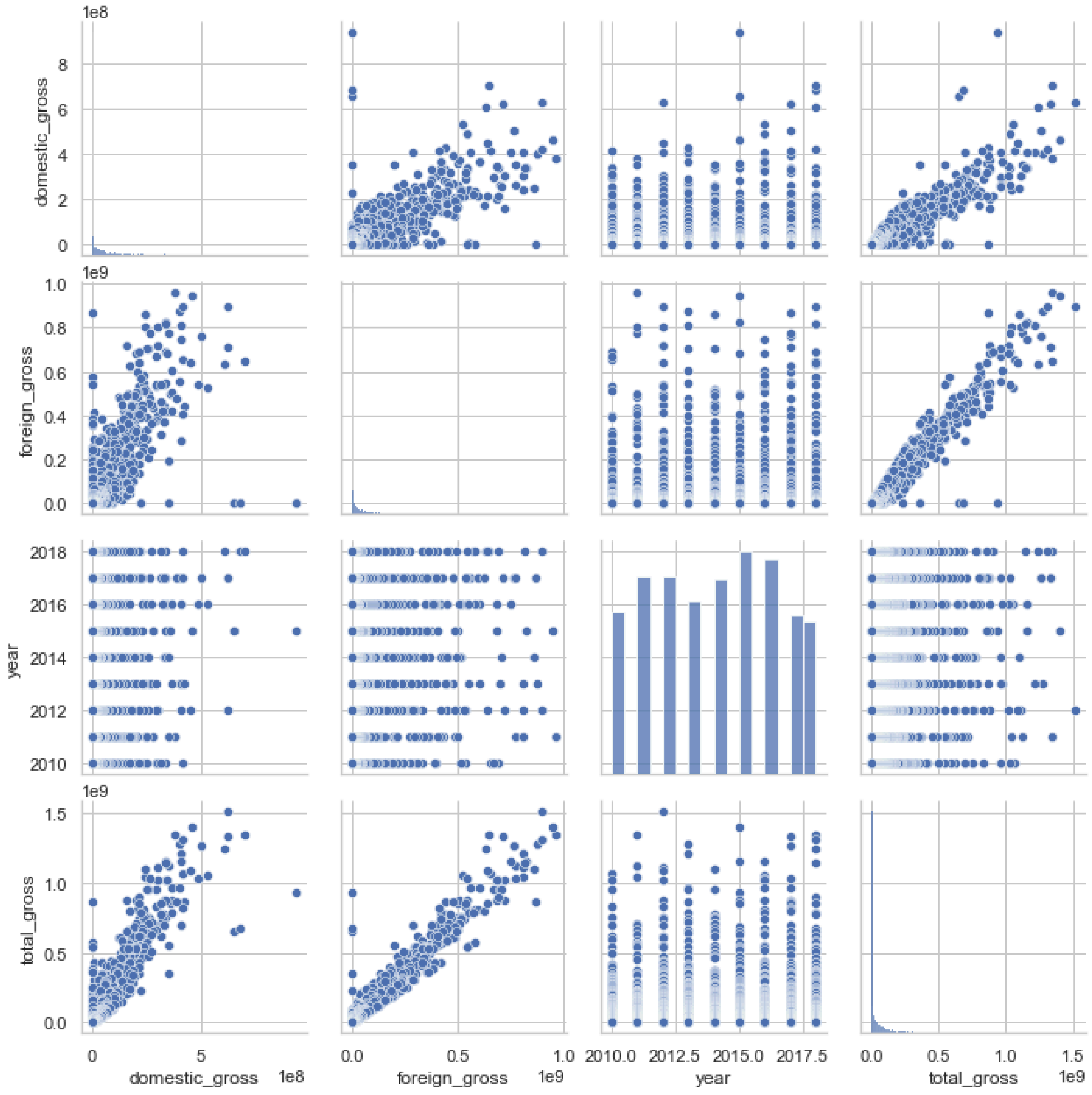
In [48]:
```
plt.figure(figsize = (12, 6))
sns.barplot(x = top_total.values, y = top_total.index, palette = 'viridis')
plt.title('Top 10 Studios by Total Gross')
plt.xlabel('Total Gross')
plt.ylabel('Studio')

plt.savefig('Images/top_studio.png', dpi=300, bbox_inches='tight')
plt.show()
```



BV studio tops with total gross of 44,212,880,000, Fox studio has 31,005,370,000 and WB studio has 30,835,950,000.

3.3 Multi-variate analysis

In [49]:
```
numeric_df = df_bom.select_dtypes(include = 'number')
sns.pairplot(numeric_df)
plt.show()
```



1. Domestic_gross/Foreign_gross vs Total_gross:

Both have a very strong linear relationship with total gross.

Insight: High domestic or foreign revenue is a strong indicator of total success.

2. Year Does Not Strongly Influence Gross Revenue

The points for year are spread evenly, with no strong trend upward or downward.

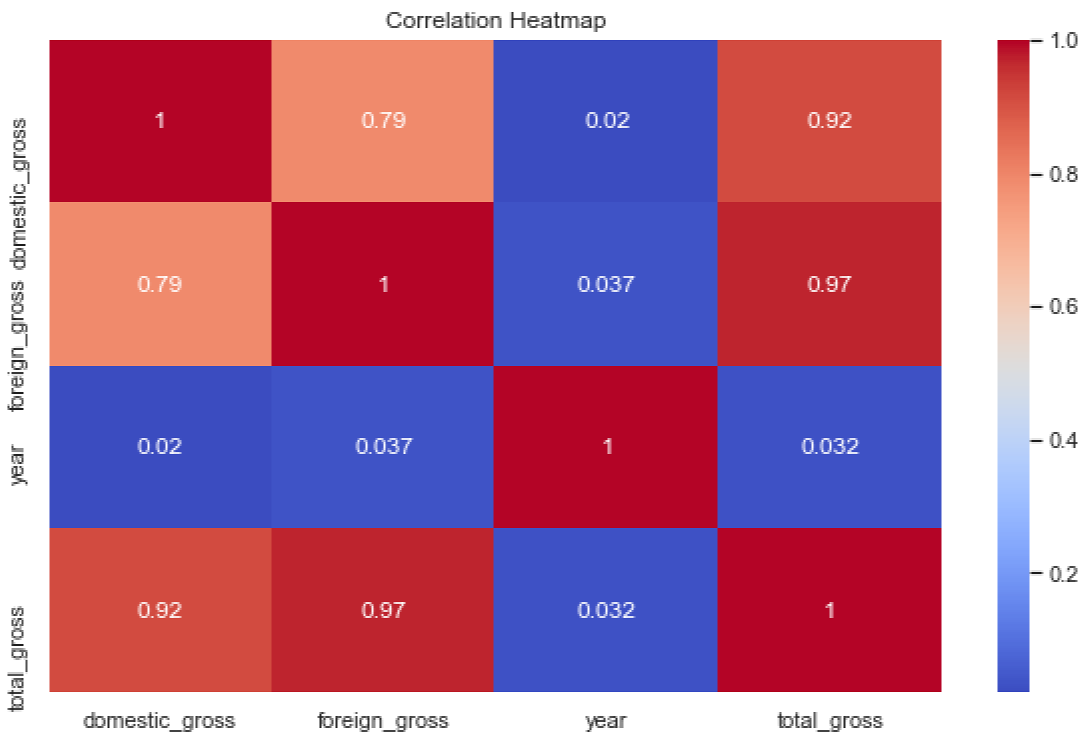
Insight: Year of release alone doesn’t significantly affect a movie’s gross revenue.

3. Total Gross is Driven More by Foreign Gross

The scatter for foreign_gross vs total_gross is tighter and more linear than that of domestic_gross vs total_gross.

Insight: Foreign markets may have a slightly stronger contribution to total gross than domestic markets.

In [50]:
```
# Select relevant numeric columns
num_df = df_movie_copy[['start_year', 'runtime_minutes', 'averagerating', 'nur

# Compute correlation
corr = numeric_df.corr()

# Plot heatmap
plt.figure(figsize = (10,6))
sns.heatmap(corr, annot = True, cmap = 'coolwarm')
plt.title("Correlation Heatmap")
plt.show()
```



No strong correlations among the variables:

All correlation values are between -0.04 and 0.12, which indicates very weak or no linear relationships between the variables.

Recommendations

1. Focus on Drama, Documentary, and Comedy

   These genres are produced the most showing they're popular and less risky.

Recommendation: Spend more of the budget investing in these genres.

2. Leverage High-Rated Genres Like Short, Documentary, and News

   Short films have the highest average rating (8.8), with Documentary, News, and Game-Shows also ranking highly.

   Recommendation: Invest in short films and factual content to enhance content quality and build audience trust.

3. Partner with Top-Grossing Studios Like BV, Fox, and Warner Bros

   BV, Fox and Warner Bros lead in total revenue each surpassing $30B, indicating strong market performance and audience reach.

   Recommendation: Work with top studios through partnerships or licensing to increase revenue and reach more viewers.

4. Strong Link Between Domestic and International Success

   Movies that earn more domestically tend to earn more internationally too, as shown by the high correlation (0.79).

   Recommendation: Prioritize movies with global reach and plan forboth domestic and international marketing to maximize revenue.

Conclusion

1. Drama, Documentary, and Comedy are the most produced genres, showing strong industry preference and consistent audience demand.
2. Short films have the highest average ratings (8.8) followed by Documentary, News and Game-Shows, indicating that factual and concise content is well-received.
3. Top studios like BV, Fox and Warner Bros dominate in total revenue, highlighting their strong market influence and production power.
4. Movies that perform well domestically often succeed internationally, supported by a strong correlation coefficient of 0.79.