Final Analysis: IMDb and Financial Data Integration

Business Understanding

IMDB Objectives

Our analysis will focus on the following objectives:

1. **Identify Popular Genres**: Analyze which genres tend to have higher ratings and more viewer engagement.
2. **Analyze Characteristics of High-Rated Movies**: Examine factors such as runtime, year of release, and genre combinations to see if they correlate with higher ratings.
3. **Investigate Trends Over Time**: Look at how preferences in ratings, movie length, and genres have evolved over the years, highlighting trends that may be valuable for the new studio to consider..

Financial Datasets Objectives

Our analysis will focus on the following objectives:

1. **Identify Most Profitable Genres**: Analyze which genres tend to have higher profitability and the trends in profits by genre

2. **Identify genres with best return on Investments**: Analyze which genres tend to have higher ROI

3. **Determine costs to produce different genres**: Analyze production budgets by genre

4. **Determine the factors that affect profitability**: Does popularity, vote count, ratings,affect profitability

Data Sources

1. **IMDb Database**:
   - Contains movie ratings, genres, and key details.
2. **TMDb Dataset**:
   - Includes popularity metrics and genre encodings.
3. **Budget Dataset**:
   - Provides production budgets, domestic, and worldwide revenue.

CRISP-DM Framework

1. **Business Understanding**: Define the objectives and questions.
2. **Data Understanding**: Explore the datasets to understand their structure and content.

3. **Data Preparation**: Clean, transform, and merge data for analysis.
4. **Modeling/Analysis**: Uncover trends and correlations through visualizations and metrics.
5. **Evaluation**: Summarize key findings and actionable insights.
6. **Deployment**: Present the final results in a structured manner.

Data Understanding

We work with the following datasets:

- **IMDb Database**: Contains movie ratings and genres.
- **TMDb Data**: Offers genre encodings, popularity scores, and audience ratings.
- **Budget Data**: Includes production budgets, domestic, and worldwide revenue.

Data Preparation

Steps:

1. Clean and format financial and popularity data (TMDb and budget datasets).
2. Extract relevant information from the IMDb database (`movie_basics` and `movie_ratings`).

```
#install all libraries to be used
import pandas as pd
import numpy as np
import seaborn as sns
sns.set_style('whitegrid')
import scipy as sp
import scipy.stats as st
import sqlite3
from zipfile import ZipFile
import os
import statsmodels.api as sm
from matplotlib.colors import ListedColormap
from statsmodels.stats.power import TTestIndPower, TTestPower
import statsmodels.formula as smf
import matplotlib.pyplot as plt
%matplotlib inline
# Ignore all warnings
import warnings
warnings.filterwarnings('ignore')
```

READING DATASETS

1.1 Data\tn.movie_budgets.csv

```
# Reading movie_budgets dataset
df_movie_budgets =
pd.read_csv("data/tn.movie_budgets.csv",encoding='latin1')

df_movie_budgets.shape # checking shape
```

```python
"""Has 5782 rows and 6 columns"""

df_movie_budgets.info()
df_movie_budgets.head() # displaying sample data (first 5 rows)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5782 entries, 0 to 5781
Data columns (total 6 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 5782 non-null   int64
 1   release_date       5782 non-null   object
 2   movie              5782 non-null   object
 3   production_budget  5782 non-null   object
 4   domestic_gross     5782 non-null   object
 5   worldwide_gross    5782 non-null   object
dtypes: int64(1), object(5)
memory usage: 271.2+ KB

   id  release_date                                        movie  \
0   1  Dec 18, 2009                                       Avatar
1   2  May 20, 2011  Pirates of the Caribbean: On Stranger Tides
2   3   Jun 7, 2019                                 Dark Phoenix
3   4   May 1, 2015                      Avengers: Age of Ultron
4   5  Dec 15, 2017            Star Wars Ep. VIII: The Last Jedi

  production_budget domestic_gross worldwide_gross
0      $425,000,000   $760,507,625  $2,776,345,279
1      $410,600,000   $241,063,875  $1,045,663,875
2      $350,000,000    $42,762,350    $149,762,350
3      $330,600,000   $459,005,868  $1,403,013,963
4      $317,000,000   $620,181,382  $1,316,721,747
```

1.2 Data\tmdb.movies.csv

```python
# Reading tmdb_movies dataset
df_movies = pd.read_csv("data/tmdb.movies.csv",encoding='latin1')
df_movies.shape # checking shape
"""Has 26517 rows and 10 columns"""

df_movies.info()
df_movies.head().T # displaying sample data (first 5 rows)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26517 entries, 0 to 26516
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Unnamed: 0         26517 non-null  int64
 1   genre_ids          26517 non-null  object
 2   id                 26517 non-null  int64
```

```
 3   original_language  26517 non-null  object 
 4   original_title     26517 non-null  object 
 5   popularity         26517 non-null  float64
 6   release_date       26517 non-null  object 
 7   title              26517 non-null  object 
 8   vote_average       26517 non-null  float64
 9   vote_count         26517 non-null  int64  
dtypes: float64(2), int64(3), object(5)
memory usage: 2.0+ MB

                                                               0  \
Unnamed: 0                                                     0   
genre_ids                                        [12, 14, 10751]   
id                                                         12444   
original_language                                             en   
original_title     Harry Potter and the Deathly Hallows: Part 1   
popularity                                                33.533   
release_date                                          2010-11-19   
title              Harry Potter and the Deathly Hallows: Part 1   
vote_average                                                 7.7   
vote_count                                                 10788   

                                         1              2  
3  \
Unnamed: 0                               1              2  
3   
genre_ids              [14, 12, 16, 10751]  [12, 28, 878]  [16, 35, 
10751]   
id                                   10191          10138  
862   
original_language                       en             en  
en   
original_title     How to Train Your Dragon     Iron Man 2        Toy 
Story   
popularity                          28.734         28.515  
28.005   
release_date                    2010-03-26     2010-05-07       1995-
11-22   
title              How to Train Your Dragon     Iron Man 2        Toy 
Story   
vote_average                           7.7            6.8  
7.9   
vote_count                            7610          12368  
10174   

                               4  
Unnamed: 0                     4  
genre_ids          [28, 878, 12]  
id                         27205  
original_language             en  
```

```
original_title         Inception
popularity                 27.92
release_date          2010-07-16
title                  Inception
vote_average                 8.3
vote_count                 22186
```

IMDb Data

This is a SQLite database file containing information about the movies production, detailing movie basics, directors, cast, writers, etc.

```
# Define paths
zip_file_path = "data\im.db.zip"
extracted_dir = "data\im.db.extracted"

# Step 1: Unzip the file
with ZipFile(zip_file_path, "r") as zip_ref:
    zip_ref.extractall(extracted_dir)

# Define the path to the extracted database file
db_path = os.path.join(extracted_dir, "im.db")

# Step 2: Check if the database file exists and connect to it
if os.path.exists(db_path) and os.path.getsize(db_path) > 0:
    # Connect to the SQLite database
    conn = sqlite3.connect(db_path)

# Step 3: Check if the database file exists and has a reasonable size
if os.path.exists(db_path) and os.path.getsize(db_path) > 0:
    print("Database file exists and is not empty. Proceeding with
connection.")

    # Connect to the SQLite database
    conn = sqlite3.connect(db_path)

    # Step 3: Check tables in the database
    tables = pd.read_sql_query("SELECT name FROM sqlite_master WHERE
type='table';", conn)

    if tables.empty:
        print("No tables found in the database. The database might be
empty or corrupted.")
    else:
        print("Tables found:", tables)
else:
    print("Database file is either missing or empty.")
```

```
Database file exists and is not empty. Proceeding with connection.
Tables found:                  name
```

```
0   movie_basics
1      directors
2      known_for
3     movie_akas
4  movie_ratings
5        persons
6     principals
7        writers
```

```python
# Step 4: Load tables into DataFrames for analysis
movie_basics = pd.read_sql_query("SELECT * FROM movie_basics;", conn)
movie_ratings = pd.read_sql_query("SELECT * FROM movie_ratings;",
conn)
principals = pd.read_sql_query("SELECT * FROM principals;", conn)
persons = pd.read_sql_query("SELECT * FROM persons;", conn)
known_for = pd.read_sql_query("SELECT * FROM known_for;", conn)
directors = pd.read_sql_query("SELECT * FROM directors;", conn)
writers = pd.read_sql_query("SELECT * FROM writers;", conn)
movie_akas = pd.read_sql_query("SELECT * FROM movie_akas;", conn)

# Display brief info summary for each DataFrame
dataframes_info = {
        "movie_basics": movie_basics.info(),
        "movie_ratings": movie_ratings.info(),
        "principals": principals.info(),
        "persons": persons.info(),
        "known_for": known_for.info(),
        "directors": directors.info(),
        "writers": writers.info(),
        "movie_akas": movie_akas.info()
    }

dataframes_info  # This will display summaries for all loaded tables
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 146144 entries, 0 to 146143
Data columns (total 6 columns):
 #   Column           Non-Null Count   Dtype
---  ------           --------------   -----
 0   movie_id         146144 non-null  object
 1   primary_title    146144 non-null  object
 2   original_title   146123 non-null  object
 3   start_year       146144 non-null  int64
 4   runtime_minutes  114405 non-null  float64
 5   genres           140736 non-null  object
dtypes: float64(1), int64(1), object(4)
memory usage: 6.7+ MB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 73856 entries, 0 to 73855
Data columns (total 3 columns):
```

```
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   movie_id       73856 non-null  object 
 1   averagerating  73856 non-null  float64
 2   numvotes       73856 non-null  int64  
dtypes: float64(1), int64(1), object(1)
memory usage: 1.7+ MB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1028186 entries, 0 to 1028185
Data columns (total 6 columns):
 #   Column      Non-Null Count    Dtype 
---  ------      --------------    ----- 
 0   movie_id    1028186 non-null  object
 1   ordering    1028186 non-null  int64 
 2   person_id   1028186 non-null  object
 3   category    1028186 non-null  object
 4   job         177684 non-null   object
 5   characters  393360 non-null   object
dtypes: int64(1), object(5)
memory usage: 47.1+ MB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 606648 entries, 0 to 606647
Data columns (total 5 columns):
 #   Column              Non-Null Count   Dtype  
---  ------              --------------   -----  
 0   person_id           606648 non-null  object 
 1   primary_name        606648 non-null  object 
 2   birth_year          82736 non-null   float64
 3   death_year          6783 non-null    float64
 4   primary_profession  555308 non-null  object 
dtypes: float64(2), object(3)
memory usage: 23.1+ MB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1638260 entries, 0 to 1638259
Data columns (total 2 columns):
 #   Column     Non-Null Count    Dtype 
---  ------     --------------    ----- 
 0   person_id  1638260 non-null  object
 1   movie_id   1638260 non-null  object
dtypes: object(2)
memory usage: 25.0+ MB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 291174 entries, 0 to 291173
Data columns (total 2 columns):
 #   Column     Non-Null Count   Dtype 
---  ------     --------------   ----- 
 0   movie_id   291174 non-null  object
 1   person_id  291174 non-null  object
dtypes: object(2)
```

```
memory usage: 4.4+ MB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 255873 entries, 0 to 255872
Data columns (total 2 columns):
 #   Column     Non-Null Count   Dtype 
---  ------     --------------   ----- 
 0   movie_id   255873 non-null  object
 1   person_id  255873 non-null  object
dtypes: object(2)
memory usage: 3.9+ MB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 331703 entries, 0 to 331702
Data columns (total 8 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   movie_id           331703 non-null  object 
 1   ordering           331703 non-null  int64  
 2   title              331703 non-null  object 
 3   region             278410 non-null  object 
 4   language           41715 non-null   object 
 5   types              168447 non-null  object 
 6   attributes         14925 non-null   object 
 7   is_original_title  331678 non-null  float64
dtypes: float64(1), int64(1), object(6)
memory usage: 20.2+ MB

{'movie_basics': None,
 'movie_ratings': None,
 'principals': None,
 'persons': None,
 'known_for': None,
 'directors': None,
 'writers': None,
 'movie_akas': None}
```

CLEANING DATASETS

TmDB and tn.movies Datasets

```
# Function to remove special columns from movies and movies_budgets
def special_characters( df,columns):
    # for each colums specified the below is perfomed
    for col in columns:
        # Check if the column is of type object (string) before 
cleaning
        if df[col].dtype == 'object':
            df[col] = df[col].replace({'\$': '', ',': ''}, 
regex=True).astype(float)
            # Assertion to ensure the column has been converted to 
float
```

```python
        assert df[col].dtype == 'float64', f"Column {col} was not
converted to float"
    return df

#specifying dataframes
df1 = df_movies
df2 = df_movie_budgets

# specifying columns from the budgets and gross dataframes
columns_to_clean_df2 =
['production_budget','domestic_gross','worldwide_gross']

# cleaning specificified columns
df2 = special_characters(df2, columns_to_clean_df2)

# output sample
df2.head()
```

```
   id  release_date                                        movie  \
0   1  Dec 18, 2009                                       Avatar
1   2  May 20, 2011  Pirates of the Caribbean: On Stranger Tides
2   3   Jun 7, 2019                                 Dark Phoenix
3   4   May 1, 2015                      Avengers: Age of Ultron
4   5  Dec 15, 2017            Star Wars Ep. VIII: The Last Jedi

   production_budget  domestic_gross  worldwide_gross
0        425000000.0     760507625.0     2.776345e+09
1        410600000.0     241063875.0     1.045664e+09
2        350000000.0      42762350.0     1.497624e+08
3        330600000.0     459005868.0     1.403014e+09
4        317000000.0     620181382.0     1.316722e+09
```

```python
# Creating a new columns in df_movies_budget
#year in the df_movies_budget by extracting years from release_date
df2['year'] = pd.to_datetime(df2['release_date']).dt.year.astype(int)
#year in the df_movies by extracting years from release_date
df1['year'] = pd.to_datetime(df1['release_date']).dt.year.astype(int)

# Checking column creation of year and datatype
assert 'year' in df2.columns, "The 'year' column was not created."
assert df2['year'].dtype == 'int32', "The 'year' column is not type
integer."
assert 'year' in df1.columns, "The 'year' column was not created."
assert df1['year'].dtype == 'int32', "The 'year' column is not type
integer."

# Feature engineering a foreign_gross column
# foreign_gross = worldwide_gross - domestic_gross
df2['foreign_gross'] = df2['worldwide_gross'] - df2 ['domestic_gross']
# assertion
```

```
assert 'foreign_gross' in df2.columns,"The 'foreign_gross column was
not created"

# renaming the movie column to title
df2 = df2.rename(columns = {'movie':'title'},errors='ignore')

# Function to change all titles to title case and removing any leading
whitespaces
def clean_df(df):
    # Title case for all columns containing 'title' in the column name
    for col in df.columns:
        if 'title' in col.lower():
            df[col] = df[col].str.title()  # Converts to title case

        # Remove leading and trailing whitespace from all columns
        df[col] = df[col].str.strip() if df[col].dtype == "object"
else df[col]

    return df

# ouput:
df1 = clean_df(df1)
df2 = clean_df(df2)

# Movies dataset cleaning
# Convert both columns to title case
df1['original_title'] = df1['original_title'].str.title()
# Identify rows where titles are not identical after standardization
mismatched_titles = df1[df1['original_title'] != df1['title']]

# View mismatched titles if any
mismatched_titles.count()
"""A total of 2532 mismatched titles"""
```

```
'A total of 2532 mismatched titles'
```

```
import ast

genre_map = {
    28: "Action",
    12: "Adventure",
    16: "Animation",
    35: "Comedy",
    80: "Crime",
    99: "Documentary",
    18: "Drama",
    10751: "Family",
    14: "Fantasy",
    36: "History",
    27: "Horror",
    10402: "Music",
```

```
    9648: "Mystery",
    10749: "Romance",
    878: "Science Fiction",
    10770: "TV Movie",
    53: "Thriller",
    10752: "War",
    37: "Western",
    # TV Show-specific genres
    10759: "Action & Adventure",
    10762: "Kids",
    10763: "News",
    10764: "Reality",
    10765: "Sci-Fi & Fantasy",
    10766: "Soap",
    10767: "Talk",
    10768: "War & Politics"
}

# Function to convert a list of genre IDs to genre names
def ids_to_names(genre_ids_str):
    # Convert the string representation of a list to an actual list of
integers
    genre_ids = ast.literal_eval(genre_ids_str)
    # Map each genre ID to its name using the genre_map dictionary
    return [genre_map[genre_id] for genre_id in genre_ids if genre_id
in genre_map]

# Apply the function to the 'genre_ids' column
df1['genre_names'] = df1['genre_ids'].apply(ids_to_names)


df1.head(3)
```

```
   Unnamed: 0            genre_ids     id original_language  \
0           0      [12, 14, 10751]  12444                en
1           1  [14, 12, 16, 10751]  10191                en
2           2        [12, 28, 878]  10138                en

                                original_title  popularity
release_date  \
0  Harry Potter And The Deathly Hallows: Part 1      33.533   2010-11-
19
1                       How To Train Your Dragon      28.734   2010-03-
26
2                                     Iron Man 2      28.515   2010-05-
07

                                         title  vote_average
vote_count  \
0  Harry Potter And The Deathly Hallows: Part 1           7.7
```

```
10788
1                        How To Train Your Dragon           7.7
7610
2                                      Iron Man 2           6.8
12368

   year                              genre_names
0  2010             [Adventure, Fantasy, Family]
1  2010  [Fantasy, Adventure, Animation, Family]
2  2010     [Adventure, Action, Science Fiction]
```

```
# Lets drop columns not to be used for analysis in budget tables
columns_to_drop = ['id','release_date']
df2 = df2.drop(columns = columns_to_drop,errors = 'ignore')
# dropping columns in movies datasets
cols_to_drop = ['Unnamed: 
0','original_language','original_title','release_date']
df1 = df1.drop(columns = cols_to_drop,errors='ignore')

# Merging dataframes to use for Analysis
clean_movie_df = df2.merge(df1, on='title', how = 'inner') # performed 
an inner join
clean_movie_df.shape
```

```
(2446, 13)
```

```
# checking any null values
clean_movie_df.isna().sum()
# dropping null values
clean_movie_df = clean_movie_df.dropna()
clean_movie_df.head()
```

```
                                         title  production_budget  \
0                                       Avatar        425000000.0   
1  Pirates Of The Caribbean: On Stranger Tides        410600000.0   
2                      Avengers: Age Of Ultron        330600000.0   
3                       Avengers: Infinity War        300000000.0   
4                               Justice League        300000000.0   

   domestic_gross  worldwide_gross  year_x  foreign_gross  
genre_ids  \
0     760507625.0     2.776345e+09    2009   2.015838e+09  [28, 12, 
14, 878]   
1     241063875.0     1.045664e+09    2011   8.046000e+08      [12, 
28, 14]   
2     459005868.0     1.403014e+09    2015   9.440081e+08     [28, 
12, 878]   
3     678815482.0     2.048134e+09    2018   1.369319e+09      [12, 
28, 14]   
4     229024295.0     6.559452e+08    2017   4.269209e+08  [28, 12, 
14, 878]
```

```
        id  popularity  vote_average  vote_count  year_y  \
0   19995      26.526           7.4       18676    2009
1    1865      30.579           6.4        8571    2011
2   99861      44.383           7.3       13457    2015
3  299536      80.773           8.3       13948    2018
4  141052      34.953           6.2        7510    2017

                                     genre_names
0  [Action, Adventure, Fantasy, Science Fiction]
1                 [Adventure, Action, Fantasy]
2         [Action, Adventure, Science Fiction]
3                 [Adventure, Action, Fantasy]
4  [Action, Adventure, Fantasy, Science Fiction]
```

```
#dropping unwanted columns
col_to_drop = ['year_y','genre_ids','id' ]
clean_movie_df = clean_movie_df.drop(columns = col_to_drop,axis=1)
# Renaming columns
clean_movie_df = clean_movie_df.rename(columns = {'year_x':'year'})

# creating new_column
# profit = worldwide_gross-production_budget
clean_movie_df['profit'] = clean_movie_df['worldwide_gross']-
clean_movie_df['production_budget']
# Return on investment
# ROI = profit / production_budget
clean_movie_df['ROI'] = clean_movie_df['profit'] /
clean_movie_df['production_budget']

# Extracting the first genre name as the main genre
clean_movie_df_1 = clean_movie_df
clean_movie_df_1['main_genres'] =
clean_movie_df_1['genre_names'].apply(lambda x: x[0] if isinstance(x,
list) and len(x) > 0 else None)
clean_movie_df_1
```

```
                                              title
production_budget  \
0                                            Avatar        425000000.0

1     Pirates Of The Caribbean: On Stranger Tides        410600000.0

2                           Avengers: Age Of Ultron        330600000.0

3                            Avengers: Infinity War        300000000.0

4                                    Justice League        300000000.0

...                                             ...                ...
```

```
2441                                             Exeter         25000.0

2442                                                Ten         25000.0

2443                                          Dry Spell         22000.0

2444                          All Superheroes Must Die          20000.0

2445                                          Newlyweds          9000.0


      domestic_gross  worldwide_gross  year  foreign_gross  popularity  
\
0        760507625.0     2.776345e+09  2009   2.015838e+09      26.526   

1        241063875.0     1.045664e+09  2011   8.046000e+08      30.579   

2        459005868.0     1.403014e+09  2015   9.440081e+08      44.383   

3        678815482.0     2.048134e+09  2018   1.369319e+09      80.773   

4        229024295.0     6.559452e+08  2017   4.269209e+08      34.953   

...              ...              ...   ...            ...         ...   

2441             0.0     4.897920e+05  2015   4.897920e+05       5.934   

2442             0.0     0.000000e+00  2015   0.000000e+00       1.575   

2443             0.0     0.000000e+00  2014   0.000000e+00       0.600   

2444             0.0     0.000000e+00  2013   0.000000e+00       2.078   

2445          4584.0     4.584000e+03  2012   0.000000e+00       1.973   


      vote_average  vote_count                                        
genre_names  \
0              7.4       18676  [Action, Adventure, Fantasy, Science 
Fiction]   
1              6.4        8571                [Adventure, Action, 
Fantasy]   
2              7.3       13457           [Action, Adventure, Science 
Fiction]   
3              8.3       13948                [Adventure, Action, 
Fantasy]   
4              6.2        7510  [Action, Adventure, Fantasy, Science 
Fiction]   
...            ...         ...                                        
...   
2441           4.7         121                           [Thriller, 
```

```
Horror]
2442             5.4            5          [Adventure, Horror, Mystery,
Thriller]
2443             6.0            1                                     [Comedy,
Romance]
2444             3.9           19                           [Science Fiction,
Thriller]
2445             5.4            7                                     [Comedy,
Romance]

            profit        ROI      main_genres
0     2.351345e+09   5.532577           Action
1     6.350639e+08   1.546673        Adventure
2     1.072414e+09   3.243841           Action
3     1.748134e+09   5.827114        Adventure
4     3.559452e+08   1.186484           Action
...            ...        ...              ...
2441  4.647920e+05  18.591680         Thriller
2442 -2.500000e+04  -1.000000        Adventure
2443 -2.200000e+04  -1.000000           Comedy
2444 -2.000000e+04  -1.000000  Science Fiction
2445 -4.416000e+03  -0.490667           Comedy

[2446 rows x 13 columns]
```

IMDb Data Cleaning

```
# Inspect the movie_basics data and drop unnecessary columns
movie_basics.columns
movie_basics = pd.read_sql("""SELECT movie_id, primary_title,
start_year, runtime_minutes, genres FROM movie_basics;""", conn)

# Inspect the directors data and drop unnecessary columns
directors.columns
directors = pd.read_sql("""SELECT movie_id, person_id FROM
directors""", conn)

# Inspect the known_for data and drop unnecessary columns
known_for.columns
known_for = pd.read_sql("""SELECT person_id, movie_id FROM
known_for""", conn)

# Inspect the movie_akas data and drop unnecessary columns
movie_akas.columns
movie_akas = pd.read_sql("""SELECT movie_id, title, region, language
FROM movie_akas""", conn)

# Inspect the movie_ratings data and drop unnecessary columns
movie_ratings.columns
movie_ratings = pd.read_sql("""SELECT movie_id, averagerating,
```

```
numvotes  FROM movie_ratings""", conn)

# Inspect the principals data and drop unnecessary columns
principals.columns
principals = pd.read_sql("""SELECT movie_id, person_id, category FROM
principals""", conn)

# Inspect the persons data and drop unnecessary columns
persons.columns
persons = pd.read_sql("""SELECT person_id, primary_name FROM
persons""", conn)

# Inspect the writers data and drop unnecessary columns
writers.columns
writers = pd.read_sql("""SELECT movie_id, person_id FROM writers""",
conn)

# Identify unique movie_id and person_id values in the primary
tables(movie_basics and persons)
# Filter each table to only inlcude rows with valid movie_id and
person_id
# Create sets of unique movie and person IDs from primary tables
valid_movie_ids = set(movie_basics['movie_id'])
valid_person_ids = set(persons['person_id'])

# Directors: filter on movie_id and person_id
directors = directors[directors['movie_id'].isin(valid_movie_ids) &
                      directors['person_id'].isin(valid_person_ids)]

# Known_for: filter on movie_id and person_id
known_for = known_for[known_for['movie_id'].isin(valid_movie_ids) &
                      known_for['person_id'].isin(valid_person_ids)]

# Movie_akas: filter on movie_id only
movie_akas = movie_akas[movie_akas['movie_id'].isin(valid_movie_ids)]

# Movie_ratings: filter on movie_id only
movie_ratings =
movie_ratings[movie_ratings['movie_id'].isin(valid_movie_ids)]

# Principals: filter on movie_id and person_id
principals = principals[principals['movie_id'].isin(valid_movie_ids) &


principals['person_id'].isin(valid_person_ids)]

# Writers: filter on movie_id and person_id
writers = writers[writers['movie_id'].isin(valid_movie_ids) &
                  writers['person_id'].isin(valid_person_ids)]
```

```
# Check if all movie_id in foreign tables exist in movie_basics
assert directors['movie_id'].isin(valid_movie_ids).all()
assert known_for['movie_id'].isin(valid_movie_ids).all()
assert movie_akas['movie_id'].isin(valid_movie_ids).all()
assert movie_ratings['movie_id'].isin(valid_movie_ids).all()
assert principals['movie_id'].isin(valid_movie_ids).all()
assert writers['movie_id'].isin(valid_movie_ids).all()

# Check if all person_id in foreign tables exist in persons
assert directors['person_id'].isin(valid_person_ids).all()
assert known_for['person_id'].isin(valid_person_ids).all()
assert principals['person_id'].isin(valid_person_ids).all()
assert writers['person_id'].isin(valid_person_ids).all()

# Update cleaned Tables in SQLite
movie_basics.to_sql("movie_basics_clean", conn, if_exists="replace",
index=False)
directors.to_sql("directors_clean", conn, if_exists="replace",
index=False)
known_for.to_sql("known_for_clean", conn, if_exists="replace",
index=False)
movie_akas.to_sql("movie_akas_clean", conn, if_exists="replace",
index=False)
movie_ratings.to_sql("movie_ratings_clean", conn, if_exists="replace",
index=False)
principals.to_sql("principals_clean", conn, if_exists="replace",
index=False)
persons.to_sql("persons_clean", conn, if_exists="replace",
index=False)
writers.to_sql("writers_clean", conn, if_exists="replace",
index=False)

# Retrieve cleaned tables
movie_basics = pd.read_sql("SELECT * FROM movie_basics_clean", conn)
directors = pd.read_sql("SELECT * FROM directors_clean", conn)
known_for = pd.read_sql("SELECT * FROM known_for_clean", conn)
movie_akas = pd.read_sql("SELECT * FROM movie_akas_clean", conn)
movie_ratings = pd.read_sql("SELECT * FROM movie_ratings_clean", conn)
principals = pd.read_sql("SELECT * FROM principals_clean", conn)
persons = pd.read_sql("SELECT * FROM persons_clean", conn)
writers = pd.read_sql("SELECT * FROM writers_clean", conn)

# Check values before dropping duplicates
print(f"movie_basics shape: {movie_basics.shape}")
print(f"directors shape: {directors.shape}")
print(f"movie_ratings shape: {movie_ratings.shape}")
print(f"known_for shape: {known_for.shape}")
print(f"movie_akas shape: {movie_akas.shape}")
print(f"persons shape: {persons.shape}")
```

```
print(f"principals shape: {principals.shape}")
print(f"writers shape before dropping duplicates: {writers.shape}")
```

```
movie_basics shape: (146144, 5)
directors shape: (291171, 2)
movie_ratings shape: (73856, 3)
known_for shape: (791006, 2)
movie_akas shape: (331703, 4)
persons shape: (606648, 2)
principals shape: (1027912, 3)
writers shape before dropping duplicates: (255871, 2)
```

```
# Drop duplicates
def drop_duplicates(df, subset=None, keep='first', inplace=True):
    # Drop duplicates based on the subset of columns
    if subset:
        df.drop_duplicates(subset=subset, keep=keep, inplace=inplace)
    else:
        df.drop_duplicates(keep=keep, inplace=inplace)

    return df

#Apply the function with appropriate subset of columns for each
DataFrame
drop_duplicates(movie_basics, subset=['movie_id'], keep='first')
drop_duplicates(directors, subset=['person_id'], keep='first')
drop_duplicates(movie_ratings, subset=['movie_id'], keep='first')
drop_duplicates(known_for, subset=['movie_id', 'person_id'],
keep='first')
drop_duplicates(movie_akas, subset=['movie_id'], keep='first')
drop_duplicates(persons, subset=['person_id'], keep='first')
drop_duplicates(principals, subset=['movie_id', 'person_id'],
keep='first')
drop_duplicates(writers, subset=['person_id', 'movie_id'],
keep='first')

# Verify the results after dropping duplicates
print(f"clean movie_basics shape: {movie_basics.shape}")
print(f"clean directors shape: {directors.shape}")
print(f"clean movie_ratings shape: {movie_ratings.shape}")
print(f"clean known_for shape: {known_for.shape}")
print(f"clean movie_akas shape: {movie_akas.shape}")
print(f"clean persons shape: {persons.shape}")
print(f"clean principals shape: {principals.shape}")
print(f"clean writers shape: {writers.shape}")
```

```
clean movie_basics shape: (146144, 5)
clean directors shape: (109251, 2)
clean movie_ratings shape: (73856, 3)
clean known_for shape: (791006, 2)
```

```
clean movie_akas shape: (122302, 4)
clean persons shape: (606648, 2)
clean principals shape: (1027874, 3)
clean writers shape: (178350, 2)

# Drop null values in the tables
movie_basics.dropna(subset=['genres'], inplace=True)

# Save the cleaned data back to SQLite
movie_basics.to_sql("movie_basics", conn, if_exists="replace",
index=False)
```

Feature Engineering (IMDB)

The primary focus is on analyzing genres. To perform genre-specific analysis, it's beneficial to split these genres into individual entries. This way, each genre can be treated independently, allowing us to analyze trends, popularity, and ratings by genre.

Objectives

1. **Split `genres`**: Separate the `genres` column into individual genre entries. This will create multiple rows for movies that belong to more than one genre.
2. **Merge with `movie_ratings`**: Later, we'll merge this split dataset with `movie_ratings` to analyze ratings for each genre.

```
# split the genres column and keep first occurence as primary_genre
movie_basics['genres'] = movie_basics.genres.str.split(',').str[0]
movie_basics.head()

    movie_id                      primary_title  start_year
runtime_minutes  \
0  tt0063540                          Sunghursh        2013
175.0
1  tt0066787  One Day Before the Rainy Season        2019
114.0
2  tt0069049       The Other Side of the Wind        2018
122.0
3  tt0069204                    Sabse Bada Sukh        2018
NaN
4  tt0100275         The Wandering Soap Opera        2017
80.0

      genres
0     Action
1  Biography
2      Drama
3     Comedy
4     Comedy
```

```
# Save the cleaned data back to SQLite
movie_basics.to_sql("movie_basics", conn, if_exists="replace",
index=False)
```

Merge `movie_basics` with `movie_ratings`

Now, we're merging `movie_basics` with the `movie_ratings` DataFrame. This will allow us to associate each genre with its corresponding movie ratings, enabling us to analyze popularity and high-rated genres.

Objectives

1. **Merge DataFrames**: Merge `movie_basics` with `movie_ratings` on `movie_id` to create a comprehensive dataset for genre-based rating analysis.
2. **Check Merge Results**: Verify the merged DataFrame to ensure it includes `averagerating` and `numvotes` alongside the genres.

```
# Merge movie_basics with movie_ratings on movie_id
movie_data = movie_basics.merge(movie_ratings, on="movie_id",
how="left")

# Display a sample of the merged DataFrame to verify
print("Sample of merged movie_data:\n", movie_data[['movie_id',
'primary_title', 'genres', 'averagerating', 'numvotes']].head())

Sample of merged movie_data:
     movie_id                    primary_title     genres
averagerating  \
0  tt0063540                        Sunghursh     Action
7.0
1  tt0066787  One Day Before the Rainy Season  Biography
7.2
2  tt0069049       The Other Side of the Wind      Drama
6.9
3  tt0069204                  Sabse Bada Sukh     Comedy
6.1
4  tt0100275         The Wandering Soap Opera     Comedy
6.5

   numvotes
0      77.0
1      43.0
2    4517.0
3      13.0
4     119.0
```

EDA Analysis on Financial Datasets (Tmdb & tn.movies Budget)

Exploratory Data Analysis Outline

- **Univariate Analysis**: Look at the distribution of Earnings and Production Budgets.

- **Bivariate Analysis**: Analyze Profitability & ROI by genre.
- **Multivariate Analysis**: identify strong linear relationships between variables in the dataset.

Univariate Analysis

Objective: Look at the distribution of Earnings and Production Budgets

```
# Brand colors
colors = ('#C5FFF8','#96EFFF','#5FBDFF','#7B66FF')

# Datasets for EDA
clean_movie_df
clean_movie_df_1
```

```
                                                 title  production_budget  \
0                                               Avatar        425000000.0
1      Pirates Of The Caribbean: On Stranger Tides        410600000.0
2                              Avengers: Age Of Ultron        330600000.0
3                               Avengers: Infinity War        300000000.0
4                                       Justice League        300000000.0
...                                                ...                ...
2441                                            Exeter            25000.0
2442                                               Ten            25000.0
2443                                         Dry Spell            22000.0
2444                          All Superheroes Must Die            20000.0
2445                                         Newlyweds             9000.0

      domestic_gross  worldwide_gross  year  foreign_gross  popularity  \
0        760507625.0     2.776345e+09  2009   2.015838e+09      26.526
1        241063875.0     1.045664e+09  2011   8.046000e+08      30.579
2        459005868.0     1.403014e+09  2015   9.440081e+08      44.383
3        678815482.0     2.048134e+09  2018   1.369319e+09      80.773
4        229024295.0     6.559452e+08  2017   4.269209e+08      34.953
```

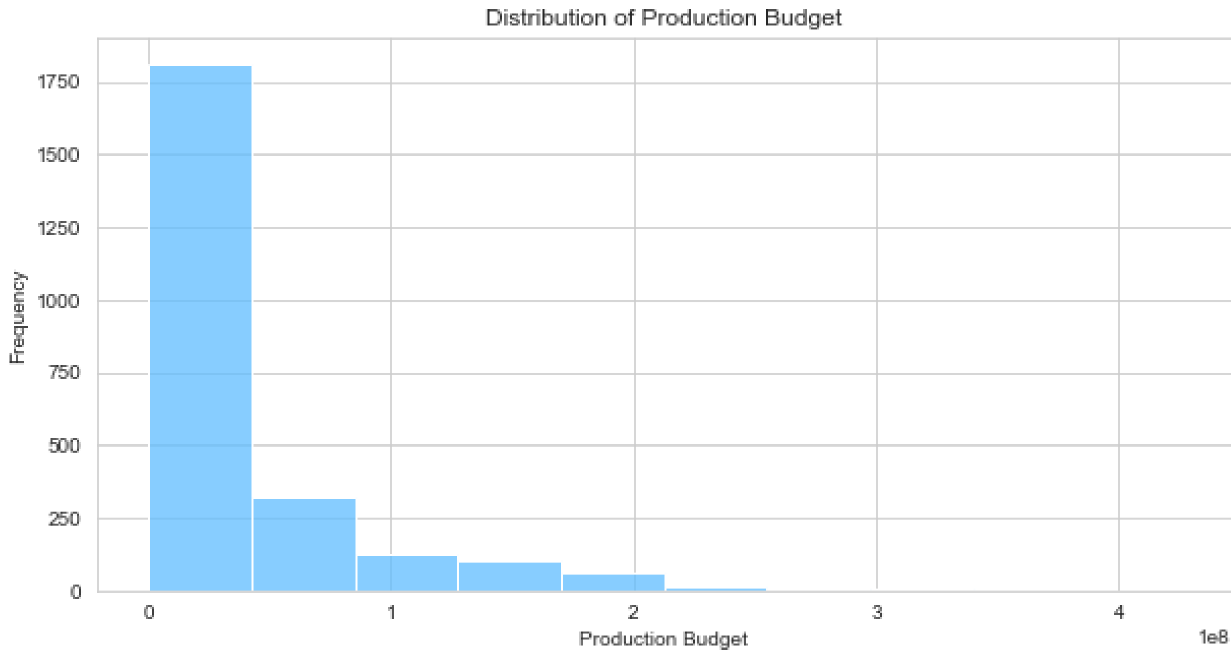
```
...               ...               ...   ...            ...         ...

2441              0.0      4.897920e+05  2015   4.897920e+05       5.934

2442              0.0      0.000000e+00  2015   0.000000e+00       1.575

2443              0.0      0.000000e+00  2014   0.000000e+00       0.600

2444              0.0      0.000000e+00  2013   0.000000e+00       2.078

2445           4584.0      4.584000e+03  2012   0.000000e+00       1.973


      vote_average  vote_count
genre_names  \
0              7.4       18676  [Action, Adventure, Fantasy, Science
Fiction]
1              6.4        8571                 [Adventure, Action,
Fantasy]
2              7.3       13457         [Action, Adventure, Science
Fiction]
3              8.3       13948                 [Adventure, Action,
Fantasy]
4              6.2        7510  [Action, Adventure, Fantasy, Science
Fiction]
...            ...         ...
...
2441           4.7         121                           [Thriller,
Horror]
2442           5.4           5       [Adventure, Horror, Mystery,
Thriller]
2443           6.0           1                             [Comedy,
Romance]
2444           3.9          19                  [Science Fiction,
Thriller]
2445           5.4           7                             [Comedy,
Romance]

            profit        ROI      main_genres
0     2.351345e+09   5.532577           Action
1     6.350639e+08   1.546673        Adventure
2     1.072414e+09   3.243841           Action
3     1.748134e+09   5.827114        Adventure
4     3.559452e+08   1.186484           Action
...            ...        ...              ...
2441  4.647920e+05  18.591680         Thriller
2442 -2.500000e+04  -1.000000        Adventure
2443 -2.200000e+04  -1.000000           Comedy
2444 -2.000000e+04  -1.000000  Science Fiction
2445 -4.416000e+03  -0.490667           Comedy
```

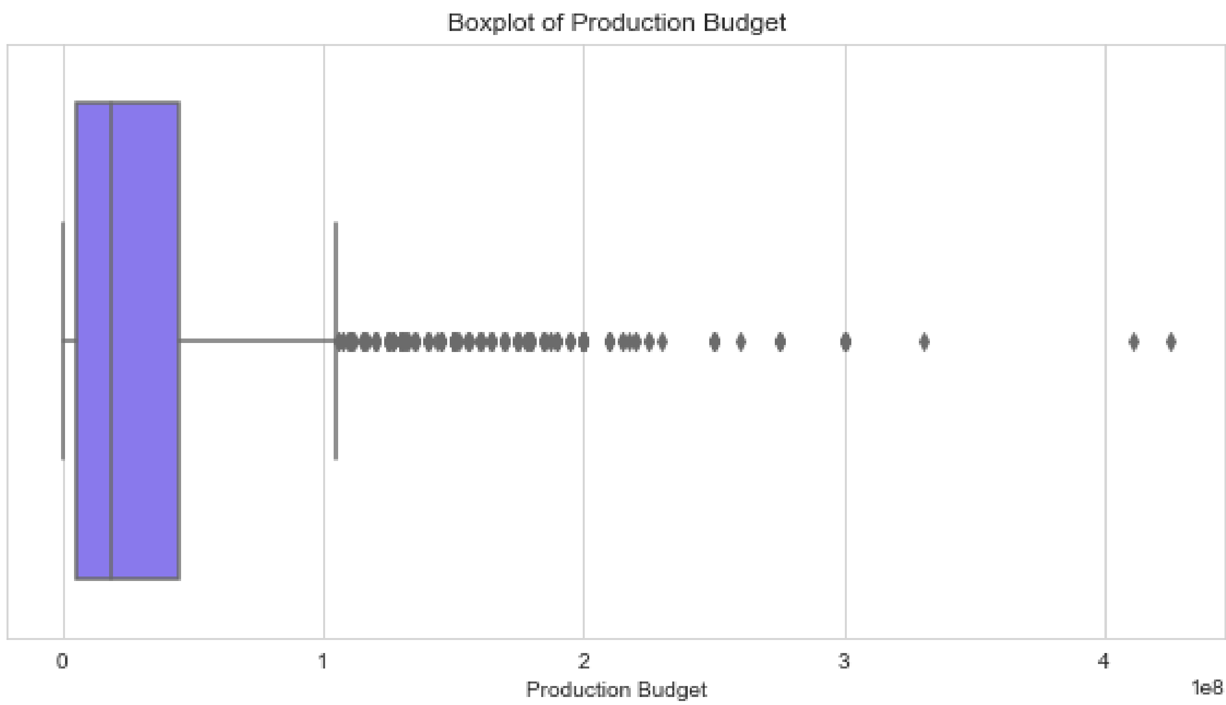
```
[2446 rows x 13 columns]
```

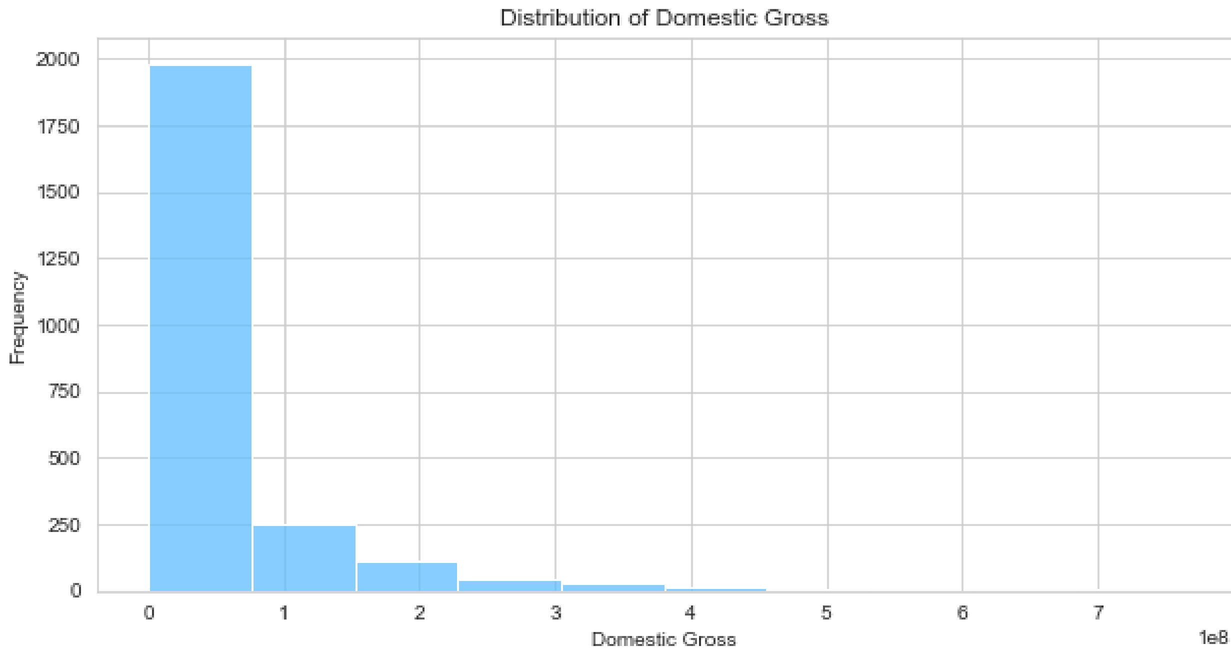
```
#summary statics for numerical columns
clean_movie_df.describe()
```

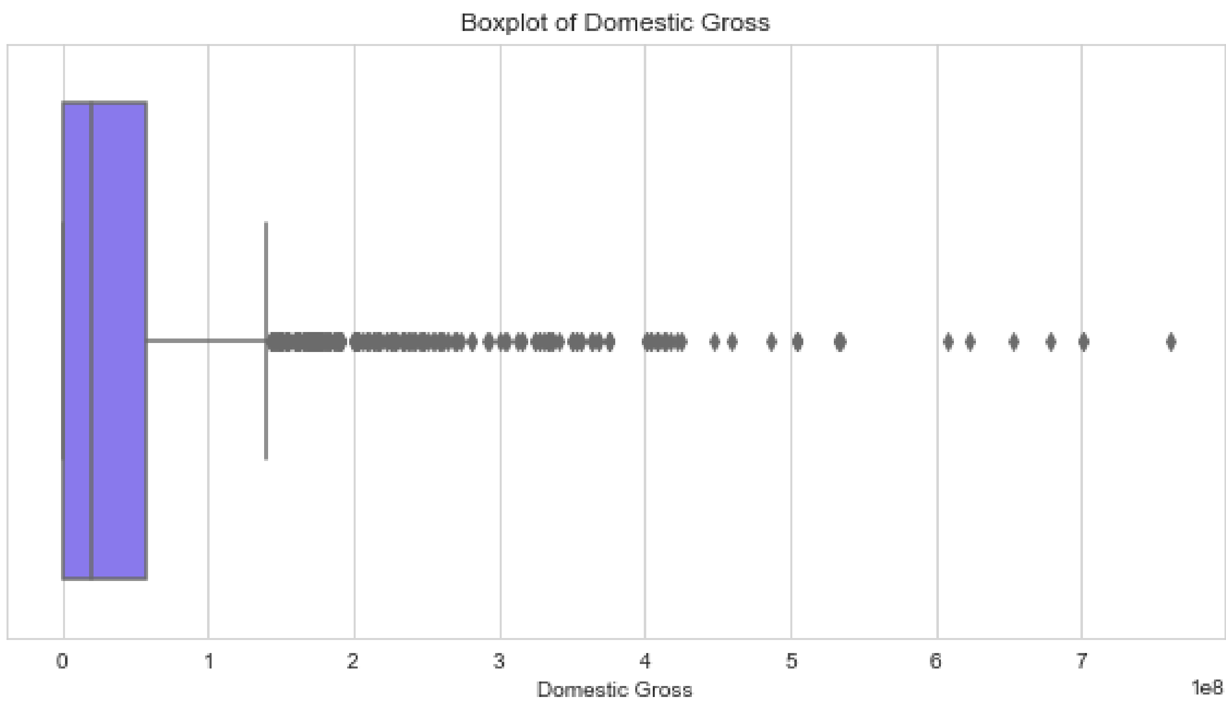
```
       production_budget  domestic_gross  worldwide_gross         year  \
count       2.446000e+03    2.446000e+03     2.446000e+03  2446.000000

mean        3.755172e+07    4.916926e+07     1.187520e+08  2011.223630

std         5.110760e+07    8.217618e+07     2.194741e+08     9.046408

min         9.000000e+03    0.000000e+00     0.000000e+00  1915.000000

25%         5.000000e+06    8.545342e+05     3.186113e+06  2011.000000

50%         1.900000e+07    1.981497e+07     3.668101e+07  2013.000000

75%         4.500000e+07    5.732941e+07     1.226038e+08  2015.000000

max         4.250000e+08    7.605076e+08     2.776345e+09  2019.000000


       foreign_gross   popularity  vote_average    vote_count
profit  \
count   2.446000e+03  2446.000000   2446.000000   2446.000000
2.446000e+03
mean    6.958273e+07    10.405210      6.206705   1646.101799
8.120027e+07
std     1.442189e+08     8.257414      1.188181   2658.721581
1.815758e+08
min     0.000000e+00     0.600000      0.000000      1.000000 -
1.104502e+08
25%     2.397542e+05     4.713250      5.600000     48.000000 -
1.812698e+06
50%     1.355679e+07     9.310000      6.300000    548.500000
1.369037e+07
75%     6.286721e+07    14.200000      6.900000   2035.750000
7.631341e+07
max     2.015838e+09    80.773000     10.000000  22186.000000
2.351345e+09

               ROI
count  2446.000000
mean      3.308609
std      14.157063
min      -1.000000
25%      -0.578497
50%       0.767637
```

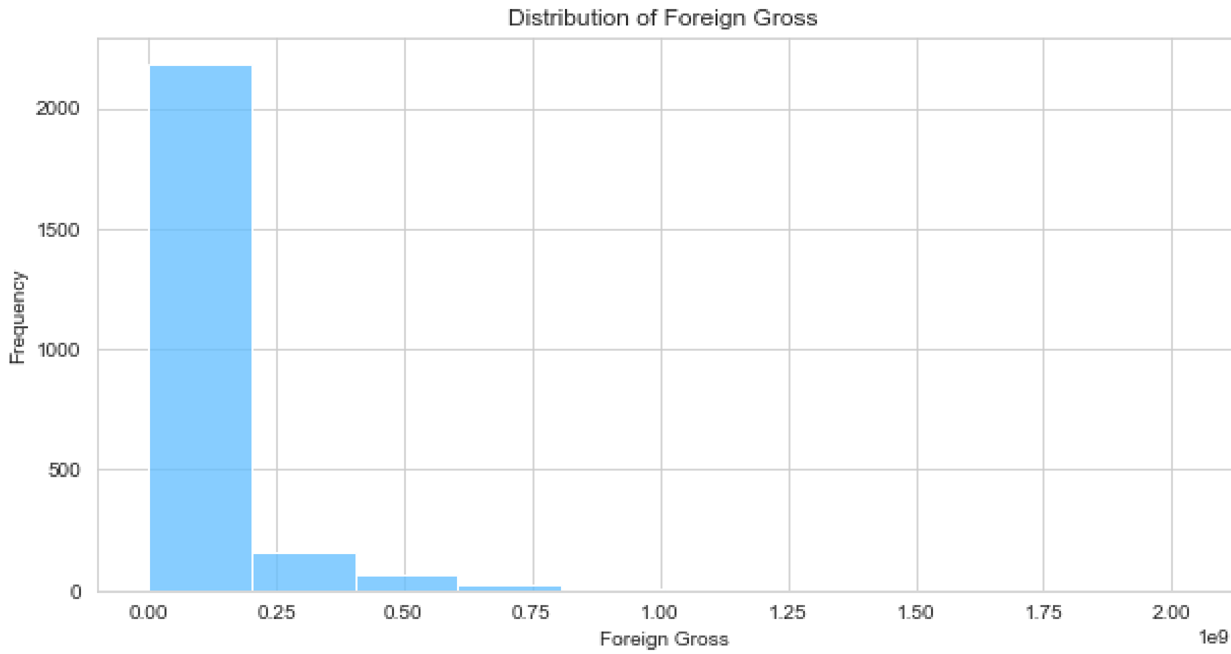
```
75%        2.880348
max      415.564740

# Function to plot distribution and boxplot for any given column in
the dataset
def plot_column_distribution(df, column_name, bin_size=10,
figsize=(10, 5)):
    # Creating a copy of dataframe is copied and we add version for
readability
    df = df.copy()
    df.loc[:,f'{column_name}'] = df[column_name]

    # Plotting the histogram
    plt.figure(figsize=figsize)
    sns.histplot(df[f'{column_name}'], bins=bin_size, kde=False,
color='#5FBDFF')
    plt.title(f'Distribution of {column_name.replace("_", "
").title()}')
    plt.xlabel(f'{column_name.replace("_", " ").title()}')
    plt.ylabel('Frequency')
    plt.show()

    # Plotting the boxplot for outliers
    plt.figure(figsize=figsize)
    sns.boxplot(x=df[f'{column_name}'], color='#7B66FF')
    plt.title(f'Boxplot of {column_name.replace("_", " ").title()}')
    plt.xlabel(f'{column_name.replace("_", " ").title()}')
    plt.show()

# usage of the function with final_df for multiple columns:
columns_to_plot = ['production_budget',
'domestic_gross','foreign_gross' ,'worldwide_gross']
# Loop through the columns and plot each one
for column in columns_to_plot:
    #plot_column_distribution(final_df, column, bin_size=30,
figsize=(10, 5))
    plot_column_distribution(clean_movie_df,column,
bin_size=10,figsize=(10,5))
```

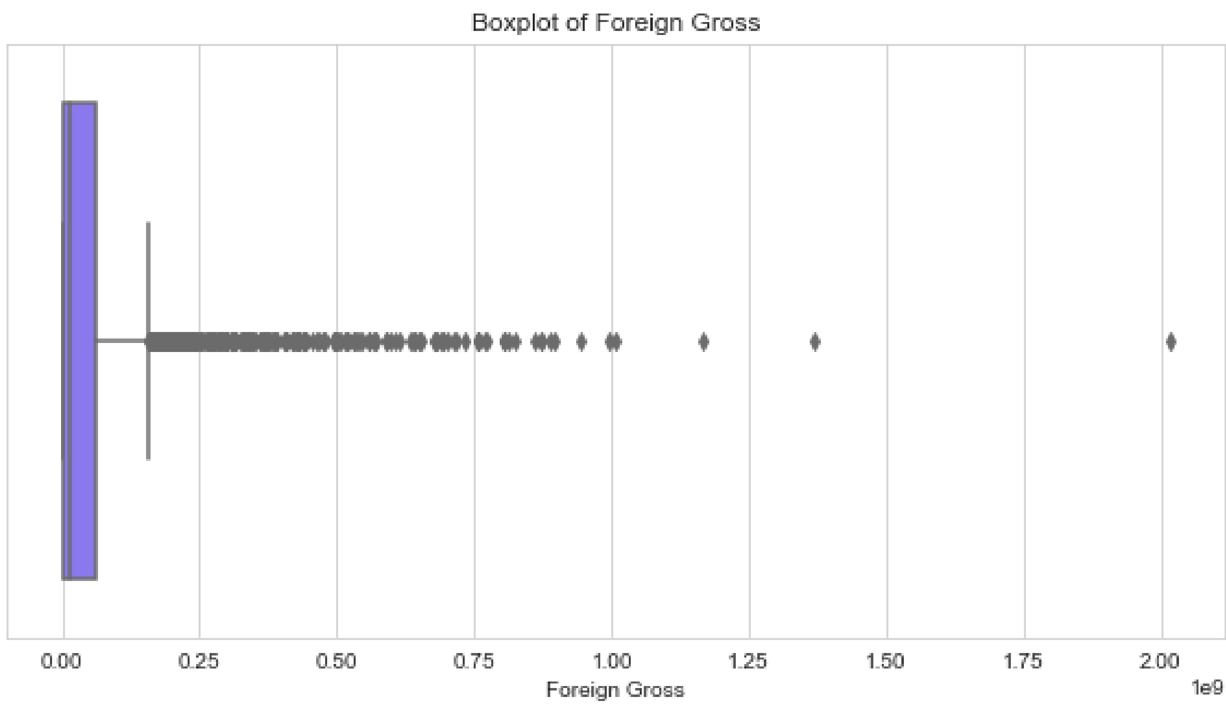

Distribution of Production Budget
Frequency
1750
1500
1250
1000
750
500
250
0
0
1
2
3
4
1e8
Production Budget

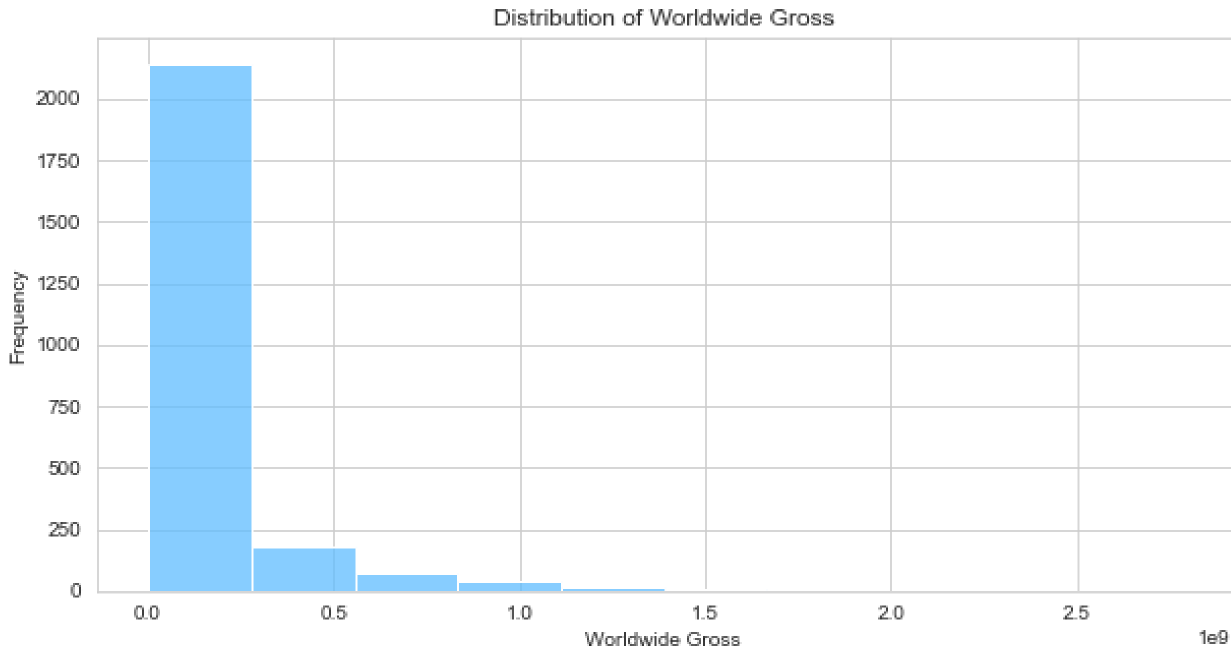

Boxplot of Production Budget
0
1
2
3
4
1e8
Production Budget

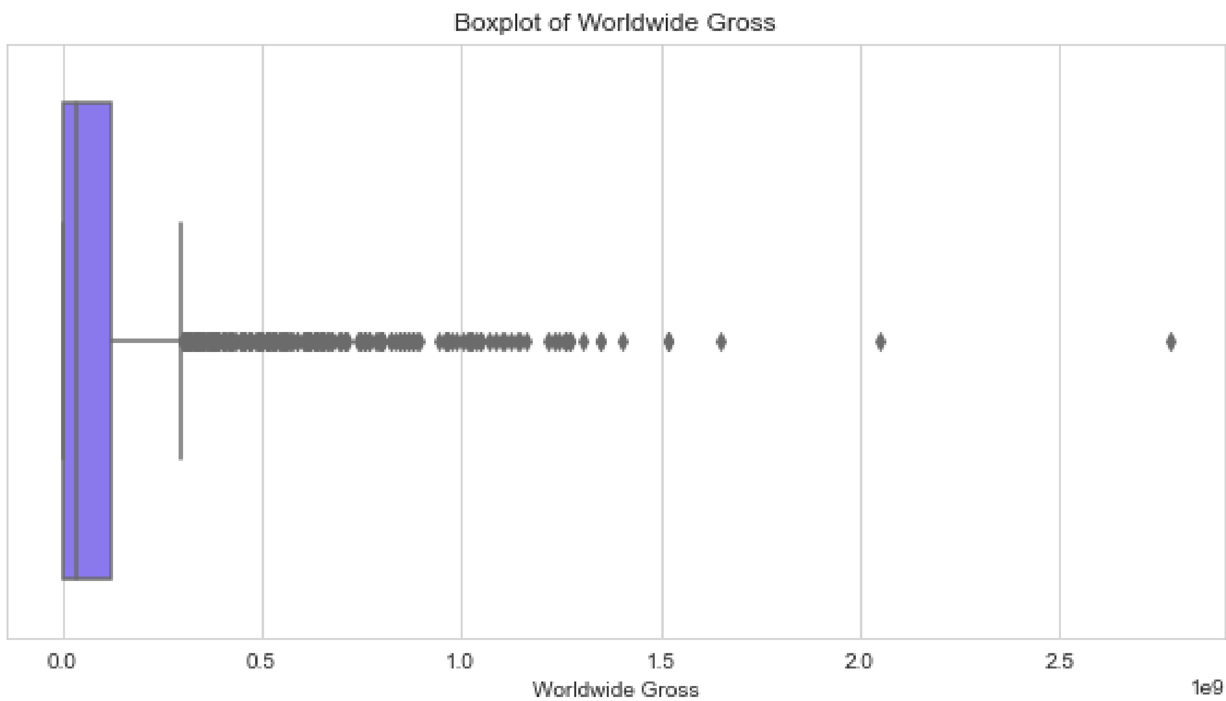

Distribution of Domestic Gross
Frequency
0
250
500
750
1000
1250
1500
1750
2000
0
1
2
3
4
5
6
7
Domestic Gross
1e8


Boxplot of Domestic Gross
0
1
2
3
4
5
6
7
Domestic Gross
1e8


Distribution of Foreign Gross
Frequency
2000
1500
1000
500
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Foreign Gross
1e9


Boxplot of Foreign Gross
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Foreign Gross
1e9





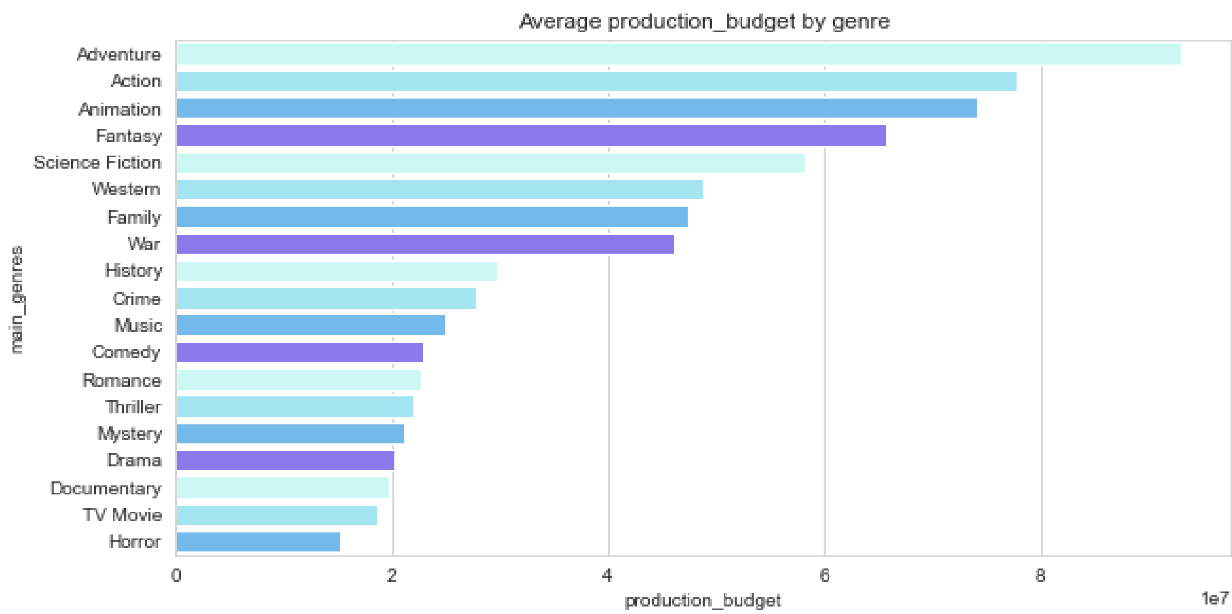
Observations

- Production budget, and all the earnings distribution are left skewed.
- Production budget in particular shows that most budgets are less than $100 million.
- The average production budget observed to be $ 37.55 million.
- There are cases of outliers observed, however they are important. In cases of earnings the outliers might depict blockbusters.

Bivariate Analysis

Objective: Analyze Profitability & ROI by genre.

```
# Production budget by genres
budget_genres =
clean_movie_df_1.groupby(['main_genres']).agg({"production_budget":"me
an"}).reset_index().sort_values(by='production_budget',

ascending=False)
#Plotting bar plot

fig, ax = plt.subplots(figsize=(10,5))
sns.barplot(x='production_budget', y='main_genres', data =
budget_genres, palette = colors)
plt.title("Average production_budget by genre")
plt.show()
```

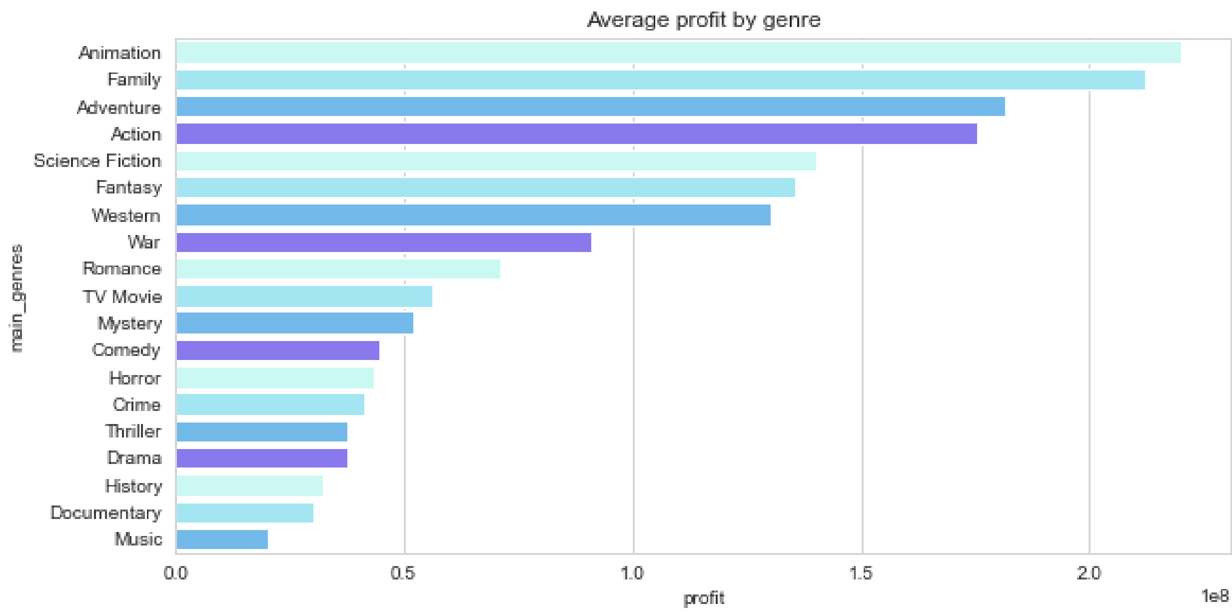


Insights on production budget by genre

- Most expensive genre to produce is Adventure
- Other top 5 genres to produce are Action, Animation,Fantasy and Sci-Fi
- Least expensive genre to produce is Horror

```
# Profitability by genres
profit_genre =
clean_movie_df_1.groupby(['main_genres']).agg({"profit":"mean"}).reset
_index().sort_values(by='profit',

ascending=False)
```

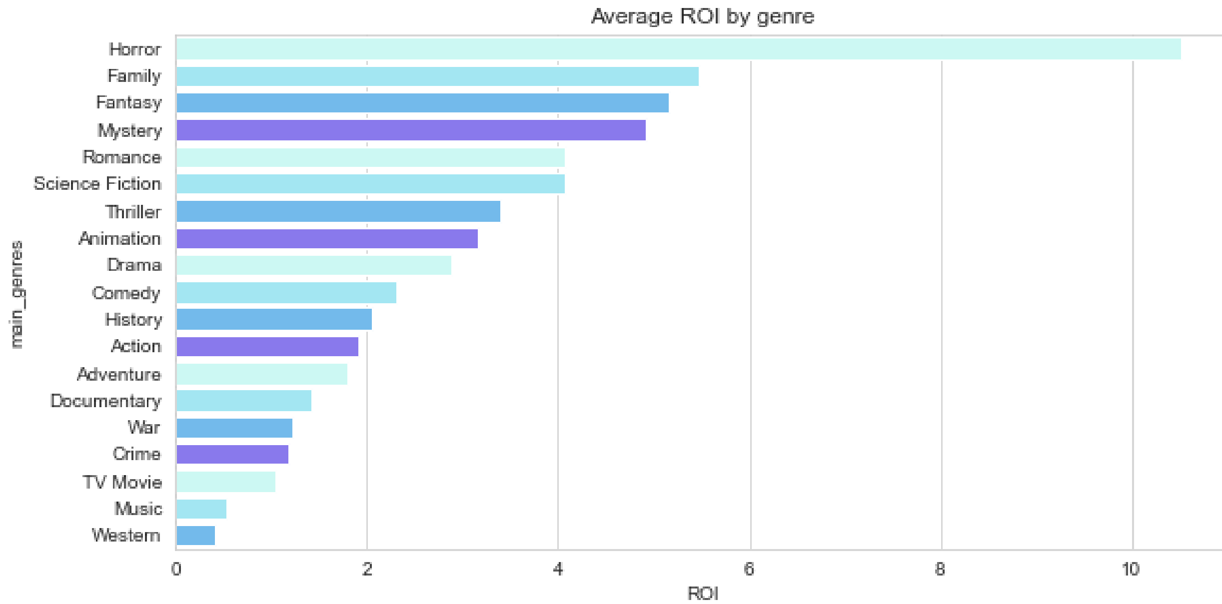
```
#Plotting bar plot
fig, ax = plt.subplots(figsize=(10,5))
sns.barplot(x='profit', y='main_genres', data=profit_genre, palette =
colors)
plt.title("Average profit by genre")
plt.show()
```



Insights on profitability by genre

- Top 5 most profitable genres are Animation,Family,Adventure, Action, Sci_Fi.
- Genres that generate profits above $150 million are Animation, Family,Adventure,and Action
- Bottom 3 genres in terms of profit generation are Music, Documentary and History

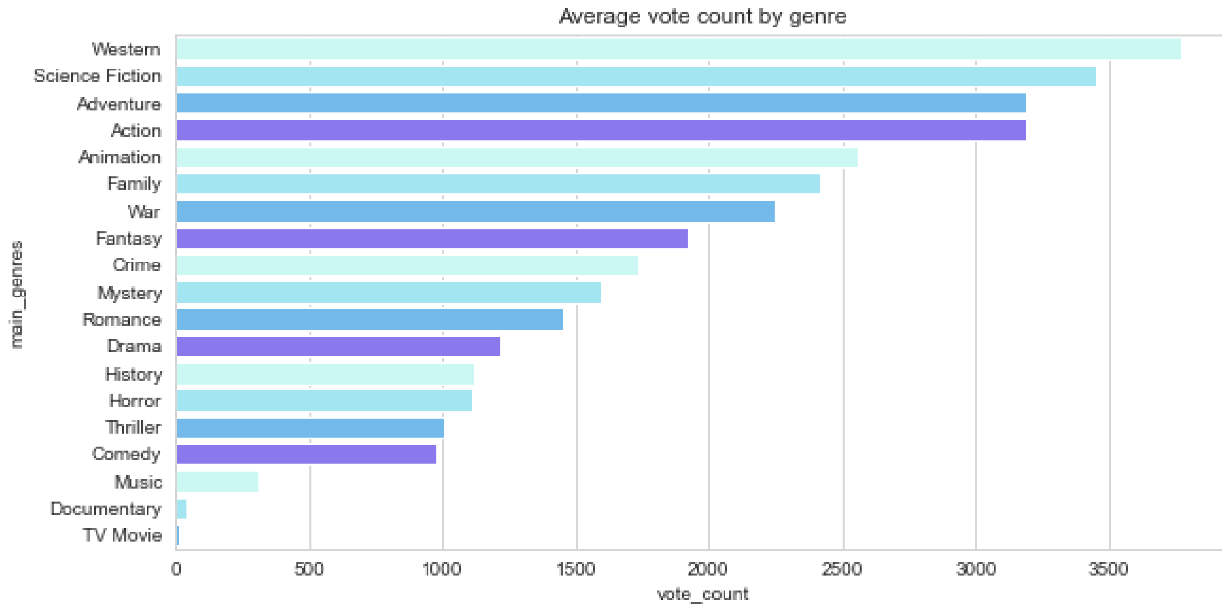
```
# Return on Investment by Genre
ROI_genre =
clean_movie_df_1.groupby(['main_genres']).agg({"ROI":"mean"}).reset_in
dex().sort_values(by='ROI',

ascending=False)

#Plotting bar plot
fig, ax = plt.subplots(figsize=(10,5))
sns.barplot(x='ROI', y='main_genres', data=ROI_genre, palette =
colors)
plt.title("Average ROI by genre")
plt.show()
```



Insights on ROI by Genre

- Horror genre has the highest ROI
- Other top genres are Family, Fantasy, Mystery

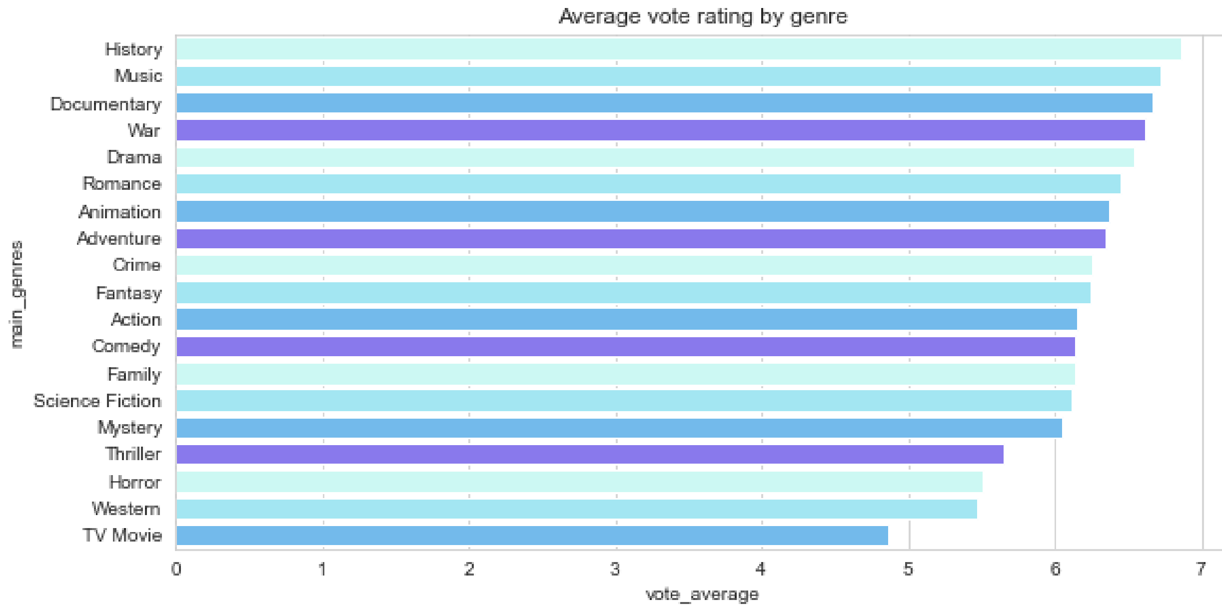
```
# Genre Analysis by Vote count (how many people voted )
avg_vote_count =
clean_movie_df_1.groupby(['main_genres']).agg({"vote_count":"mean"}).r
eset_index().sort_values(by='vote_count',

ascending=False)

#Plotting bar plot
fig, ax = plt.subplots(figsize=(10,5))
sns.barplot(x='vote_count', y='main_genres', data = avg_vote_count,
palette=colors)
plt.title("Average vote count by genre")
plt.show()
```



Insights on Vote Count by Genre

- Top 5 most voted genres were Western, Sci-Fi, Adventure, Action and Animation

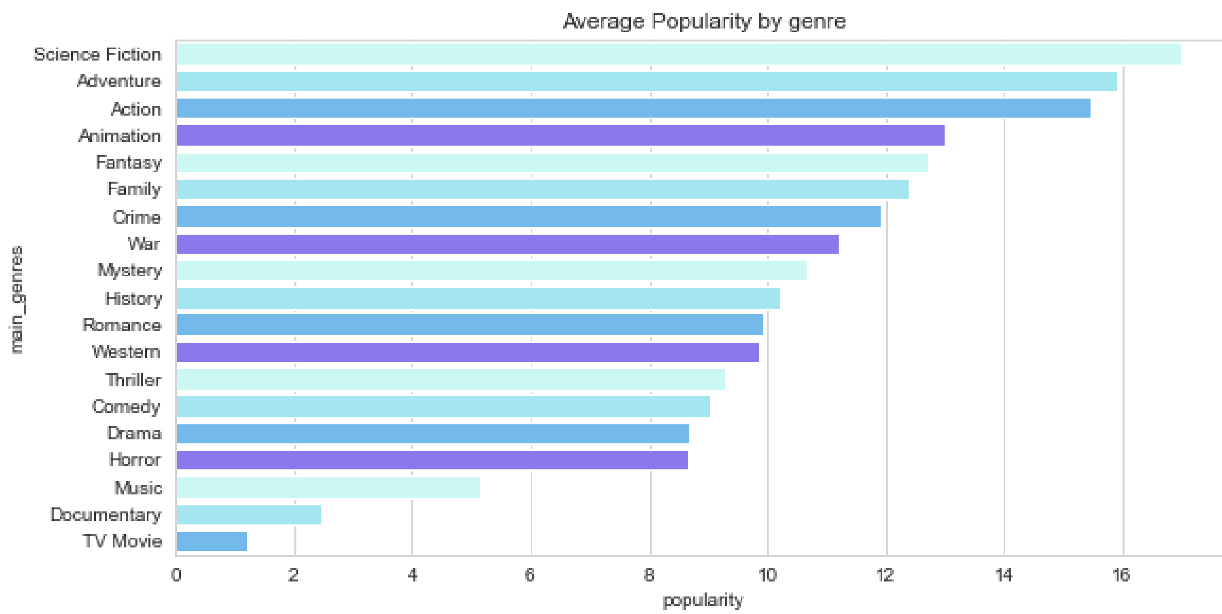
```
# Genre Analysis by Rating
avg_vote_rating =
clean_movie_df_1.groupby(['main_genres']).agg({"vote_average":"mean"})
.reset_index().sort_values(by='vote_average',

ascending=False)

#Plotting bar plot
fig, ax = plt.subplots(figsize=(10,5))
sns.barplot(x='vote_average', y='main_genres', data=avg_vote_rating,
palette=colors)
plt.title("Average vote rating by genre")
plt.show()
```



Insights on Rating by Genre

- Of all the Genres only TV movies had a less than 5 rating, with History having the highest rating

```
# Genre Analysis by popularity
avg_popularity =
clean_movie_df_1.groupby(['main_genres']).agg({"popularity":"mean"}).r
eset_index().sort_values(by='popularity',

ascending=False)

#Plotting bar plot
fig, ax = plt.subplots(figsize=(10,5))
sns.barplot(x='popularity', y='main_genres', data=avg_popularity,
palette = colors)
plt.title("Average Popularity by genre")
plt.show()
```



Insights on Popularity by Genre

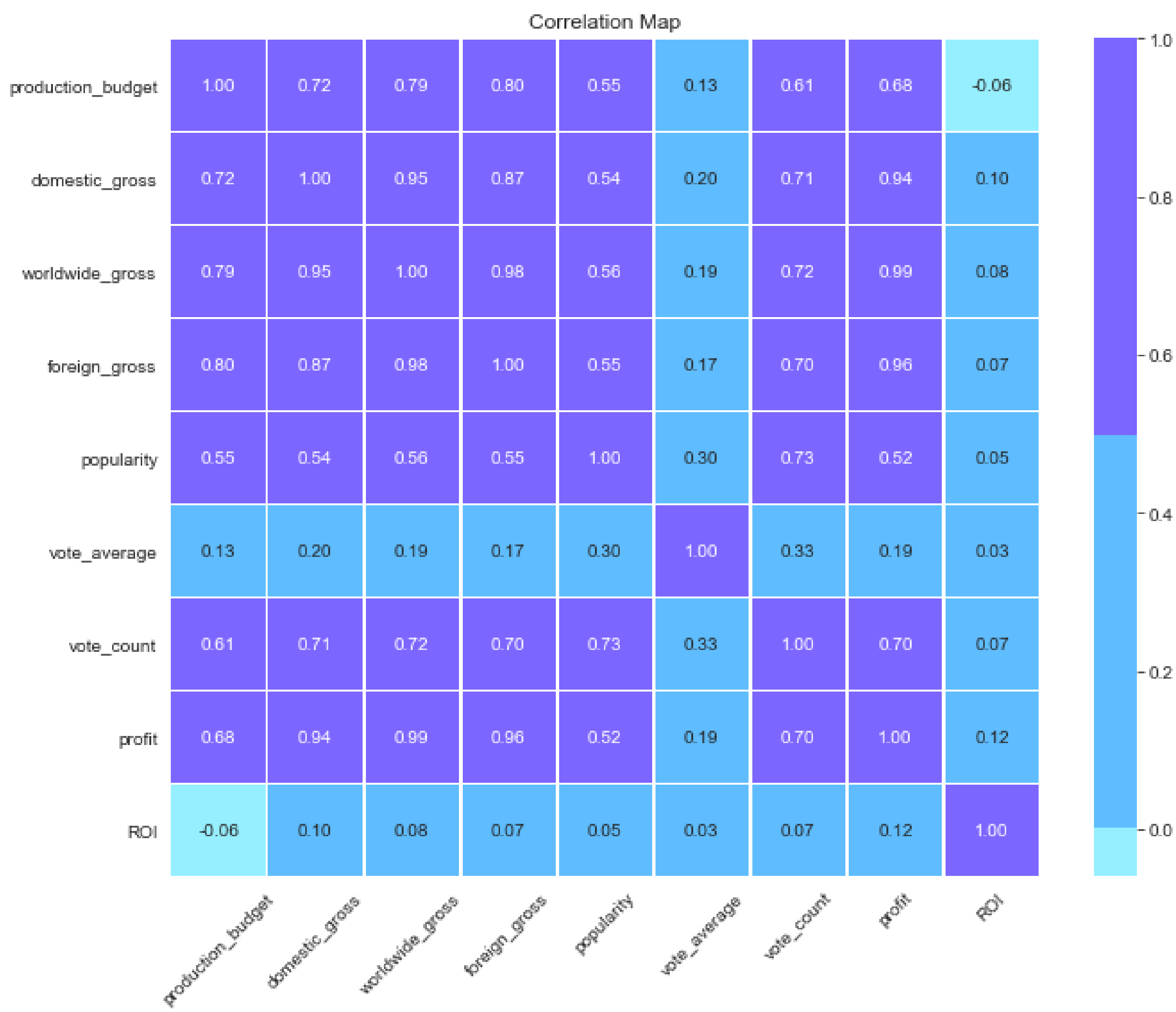
- Sci-Fi was the most popular genre
- Other genres in top 5 category by popularity were Adventure, Action, Animation, Fantasy
- Documentary and TV movie were the least popular genres

Multivariate Analysis

Objective: Identify strong linear relationships between variables in the dataset.

```
def plot_corr_heatmap(df, cols=None, figsize=(10, 8)):

    # If columns is not provided, use all numeric columns in the
DataFrame
    if cols is None:
        cols = df.select_dtypes(include=['float64', 'int64']).columns

    # Calculate the correlation matrix for the specified columns
    correlation_matrix = df[cols].corr()

    # Create a custom colormap using your colors
    colors = ['#C5FFF8', '#96EFFF', '#5FBDFF', '#7B66FF']
    cmap = ListedColormap(colors)

    # Set up the figure size
    plt.figure(figsize=figsize)

    # Plot the heatmap
    sns.heatmap(correlation_matrix, annot=True, cmap=cmap, fmt='.2f',
linewidths=0.5, cbar=True, center=0)
```

```
    # Title and labels
    plt.title('Correlation Map')
    plt.xticks(rotation=45)  # Rotate x-axis labels for readability
    plt.yticks(rotation=0)   # Keep y-axis labels horizontal

    # Show the plot
    plt.tight_layout()
    plt.show()

# Example usage:
#plot_corr_heatmap(final_df)  # Plot the correlation map for all
columns
plot_corr_heatmap(clean_movie_df_1)
```



Insights Correlation Production budget vs gross earnings

- Observed to have strong positive correlation(>0.7) with gross earnings. This suggests that higher spending on production often lead to higher revenue

Domestic Gross vs Foreign Gross

- Observed to have strong positive correlation of 0.87. This suggests that movies that do well domestically tend to perform well internationally.

Vote average (audience score) vs earnings

- Observed to generally have a weak correlation. However this may also be affected by other factors such as genre

```
# Function to get correlation values
def get_corr(df, threshold=0.7):

    # Calculating correlation matrix and stacking it to create pairs
    correlations = df.corr().stack().reset_index()
    correlations.columns = ['Variable 1', 'Variable 2', 'correlation']

    # Filtering out self-correlations
    correlations = correlations[correlations['Variable 1'] != 
correlations['Variable 2']]

    # Add 'correlation_strength' column based on threshold
    correlations['correlation_strength'] = 
correlations['correlation'].apply(
        lambda x: 'high' if abs(x) > threshold else 'low'
    )

    # Sort by correlation values and drop duplicate pairs
    correlations = correlations.sort_values(by='correlation', 
ascending=False).drop_duplicates(subset=['correlation'])

    return correlations.reset_index(drop=True)

# Usage example:
correlation_data = get_corr(clean_movie_df)
print(correlation_data)
```

```
          Variable 1        Variable 2  correlation 
correlation_strength
0     worldwide_gross            profit     0.985249 
high
1       foreign_gross   worldwide_gross     0.982788 
high
2              profit     foreign_gross     0.963065 
high
3      domestic_gross   worldwide_gross     0.945985 
high
4              profit    domestic_gross     0.941203 
high
5      domestic_gross     foreign_gross     0.869810 
high
```

```
6   production_budget      foreign_gross     0.798850
high
7   production_budget    worldwide_gross     0.793946
high
8          popularity         vote_count     0.732381
high
9          vote_count    worldwide_gross     0.723513
high
10  production_budget     domestic_gross     0.718474
high
11     domestic_gross         vote_count     0.705918
high
12         vote_count             profit     0.703930
high
13      foreign_gross         vote_count     0.698817
low
14             profit  production_budget     0.678190
low
15         vote_count  production_budget     0.606085
low
16    worldwide_gross         popularity     0.562936
low
17         popularity  production_budget     0.554525
low
18         popularity      foreign_gross     0.551389
low
19         popularity     domestic_gross     0.535790
low
20         popularity             profit     0.524350
low
21       vote_average         vote_count     0.328530
low
22       vote_average         popularity     0.296557
low
23     domestic_gross       vote_average     0.201330
low
24             profit       vote_average     0.194685
low
25    worldwide_gross       vote_average     0.190341
low
26      foreign_gross       vote_average     0.174945
low
27               year         popularity     0.163895
low
28  production_budget       vote_average     0.125714
low
29             profit                ROI     0.115468
low
30  production_budget               year     0.107106
```

```
low
31                 ROI      domestic_gross     0.099977
low
32                year       foreign_gross     0.089023
low
33                 ROI    worldwide_gross     0.081670
low
34          vote_count               year     0.081420
low
35     worldwide_gross               year     0.073014
low
36          vote_count                ROI     0.072714
low
37       foreign_gross                ROI     0.067320
low
38              profit               year     0.058107
low
39                 ROI         popularity     0.045618
low
40                year     domestic_gross     0.038769
low
41                year       vote_average     0.034285
low
42                 ROI       vote_average     0.031204
low
43                 ROI  production_budget    -0.059515
low
44                 ROI               year    -0.194758
low
```

: Exploratory Data Analysis (EDA)-IMDB Database

With `movie_data` prepared, let's dive into EDA to address our objectives:

1. **Identify trends in genres, styles, and themes**: We'll explore genre distributions, trends over time, and average ratings per genre.
2. **Identify popular and high-rated genres**: We'll calculate the average rating per genre and explore the frequency of high-rated movies (e.g., movies with ratings above a threshold) within each genre.

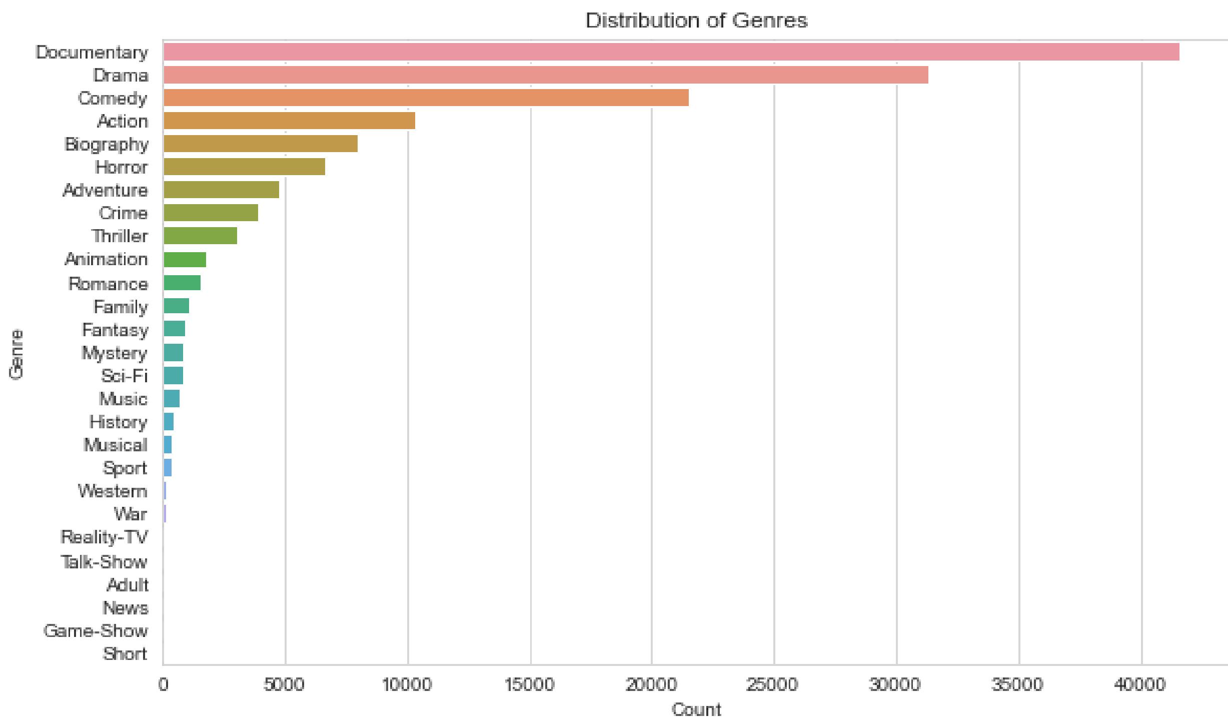
EDA Outline

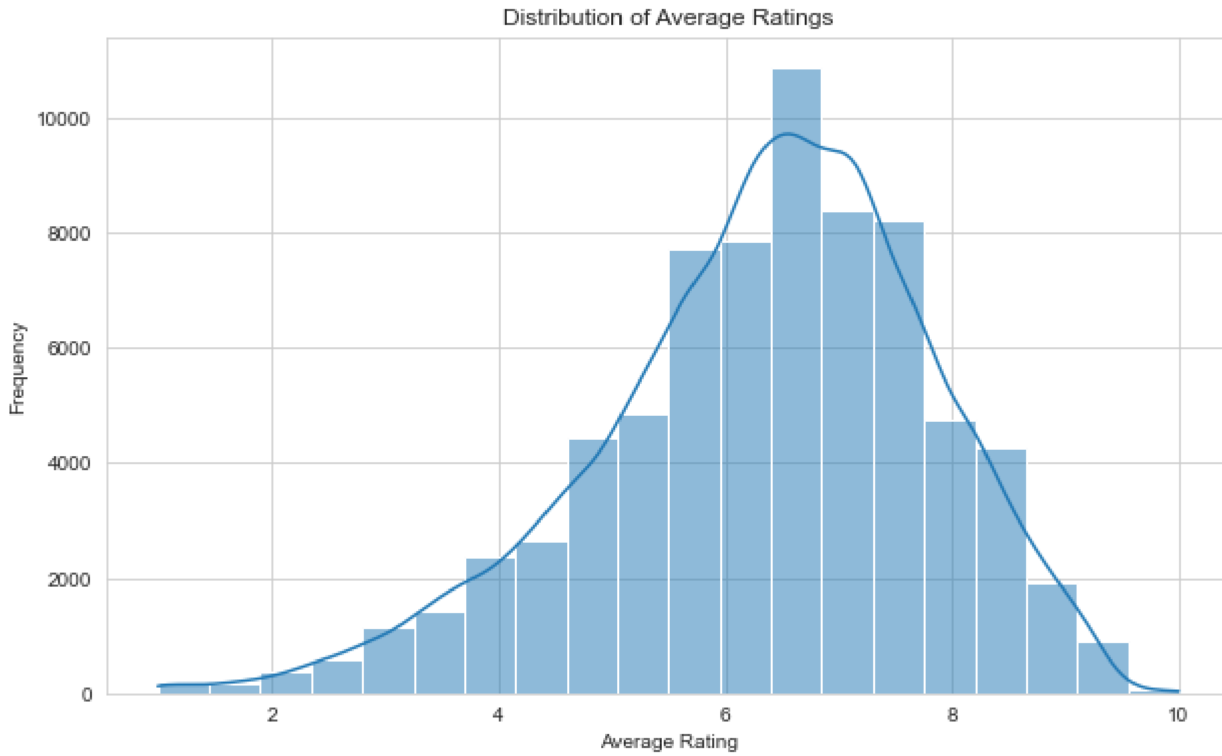
- **Univariate Analysis**: Look at the distribution of genres and ratings.
- **Bivariate Analysis**: Analyze average ratings by genre to identify popular genres.
- **Multivariate Analysis**: Analyze how genre ratings have changed over time, which will show genre popularity trends.

Univariate Analysis

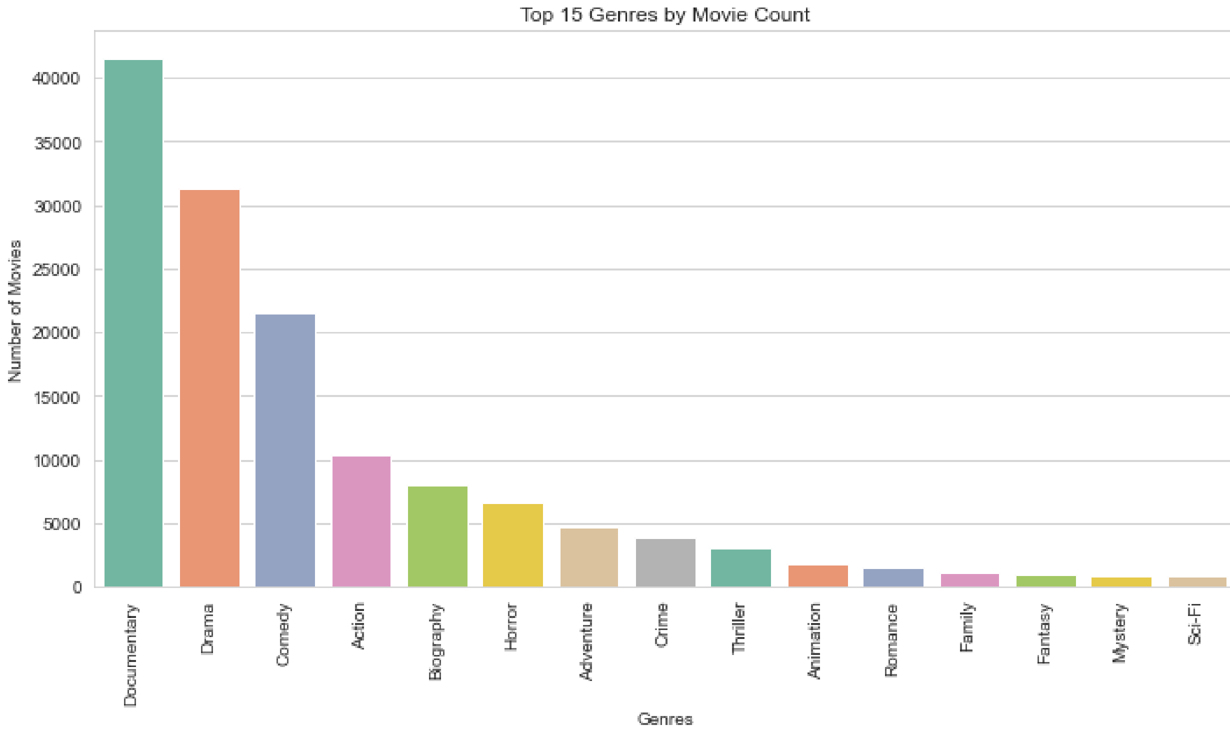
Objective: Understand the distribution of genres and average ratings.

```python
# Plot the genre distribution
plt.figure(figsize=(10, 6))
sns.countplot(data=movie_data, y='genres',
order=movie_data['genres'].value_counts().index)
plt.title("Distribution of Genres")
plt.xlabel("Count")
plt.ylabel("Genre")
plt.show()

# Plot the distribution of average ratings
plt.figure(figsize=(10, 6))
sns.histplot(movie_data['averagerating'], bins=20, kde=True)
plt.title("Distribution of Average Ratings")
plt.xlabel("Average Rating")
plt.ylabel("Frequency")
plt.show()
```

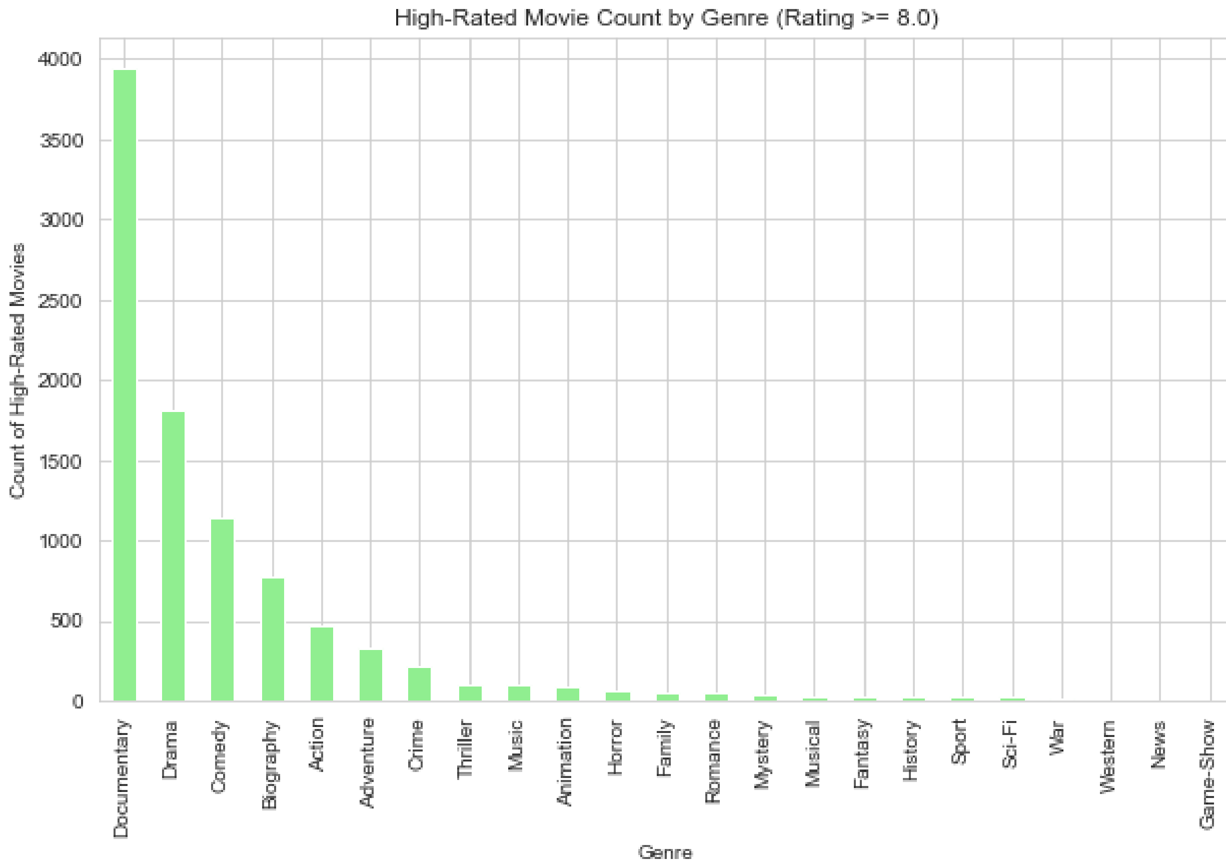





```
# Top 15 genres by movie_count
top_15_genres = """ SELECT genres, COUNT(DISTINCT movie_id) AS
genre_count
FROM movie_basics
GROUP BY genres
ORDER BY genre_count DESC
LIMIT 15;"""
top_15_genres = pd.read_sql(top_15_genres, conn)
top_15_genres

# Plot barplot of the genre count
plt.figure(figsize=(10,6))
ax= sns.barplot(y='genre_count', x='genres', data=top_15_genres,
palette='Set2')

plt.title('Top 15 Genres by Movie Count')
plt.xlabel('Genres')
plt.ylabel('Number of Movies')
plt.xticks(rotation=90)

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




```
# Filter for high-rated movies
high_rated_movies = movie_data[movie_data['averagerating'] >= 8.0]

# Count the number of high-rated movies per genre
high_rated_genre_counts = high_rated_movies['genres'].value_counts()

# Plot high-rated movie count per genre
plt.figure(figsize=(10, 6))
high_rated_genre_counts.plot(kind='bar', color='lightgreen')
plt.title("High-Rated Movie Count by Genre (Rating >= 8.0)")
plt.xlabel("Genre")
plt.ylabel("Count of High-Rated Movies")
plt.show()
```

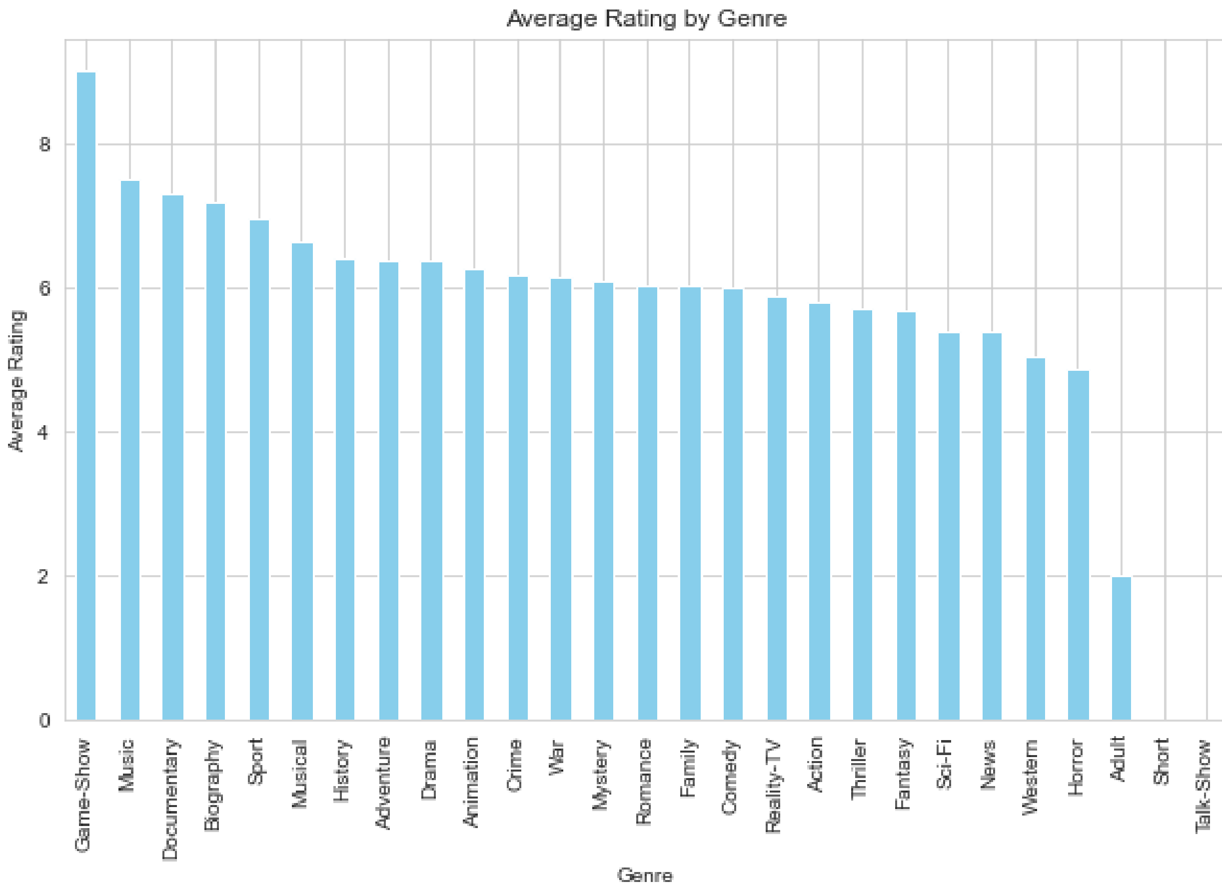


Bivariate Analysis

Objective:

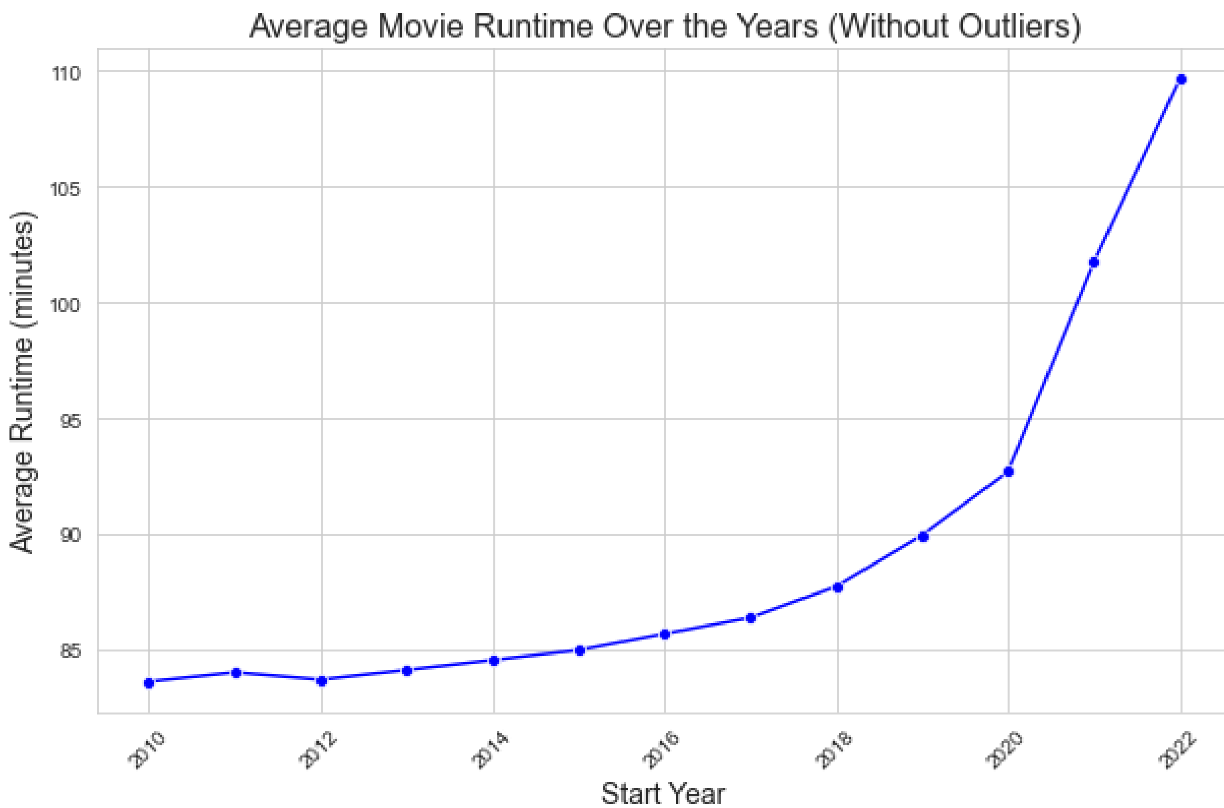
- Identify popular genres based on average ratings.
- Analyze trends in movie runtimes over the years
- Determine the average ratings for the top 10 most-rated genres to identify popular genres with high audience engagement

```
# Calculate the average rating by genre
average_rating_by_genre = movie_data.groupby('genres')
['averagerating'].mean().sort_values(ascending=False)

# Plot average rating by genre
plt.figure(figsize=(10, 6))
average_rating_by_genre.plot(kind='bar', color='skyblue')
plt.title("Average Rating by Genre")
plt.xlabel("Genre")
plt.ylabel("Average Rating")
plt.show()
```



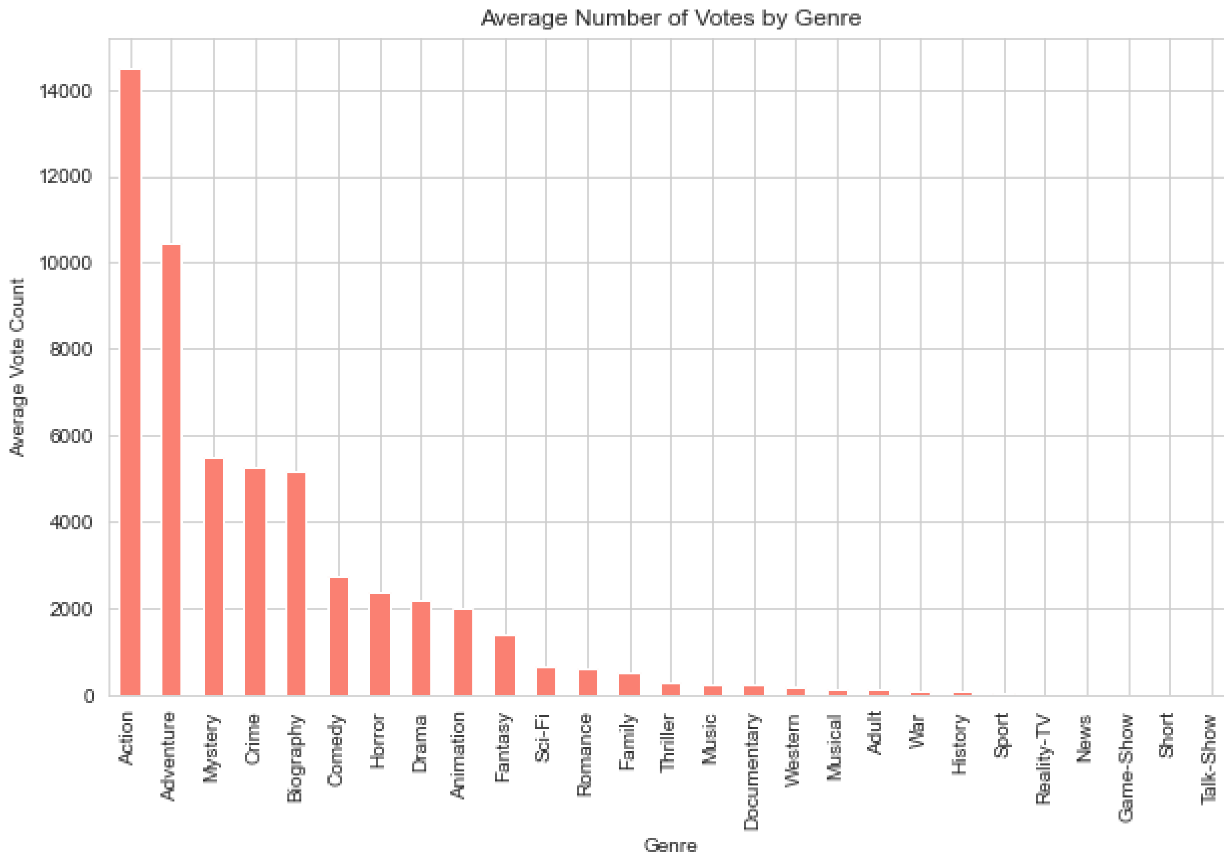

```
# Visualize how movie runtimes change over the years and filter out
the outliers based on IQR method
# Calculate the IQR to remove outliers
Q1 = movie_basics['runtime_minutes'].quantile(0.25)
Q3 = movie_basics['runtime_minutes'].quantile(0.75)
IQR = Q3 - Q1

# Calculate the lower and upper bounds for outliers
lower_val = Q1 - 1.5 * IQR
upper_val = Q3 + 1.5 * IQR

# Filter the dataset to exclude outliers based on runtime_minutes
filtered_movie_basics = movie_basics[(movie_basics['runtime_minutes']
>= lower_val) &
                                     (movie_basics['runtime_minutes']
<= upper_val)]

# Aggregate data by release year and calculate the mean runtime for
each year
average_runtime_by_year = filtered_movie_basics.groupby('start_year')
['runtime_minutes'].mean().reset_index()
```

```python
# Plot the data without outliers (line plot)
plt.figure(figsize=(10, 6))
sns.lineplot(x='start_year', y='runtime_minutes',
data=average_runtime_by_year, marker='o', color='b')

# Adding titles and labels
plt.title('Average Movie Runtime Over the Years (Without Outliers)',
fontsize=16)
plt.xlabel('Start Year', fontsize=14)
plt.ylabel('Average Runtime (minutes)', fontsize=14)

# Display the plot
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.show()
```



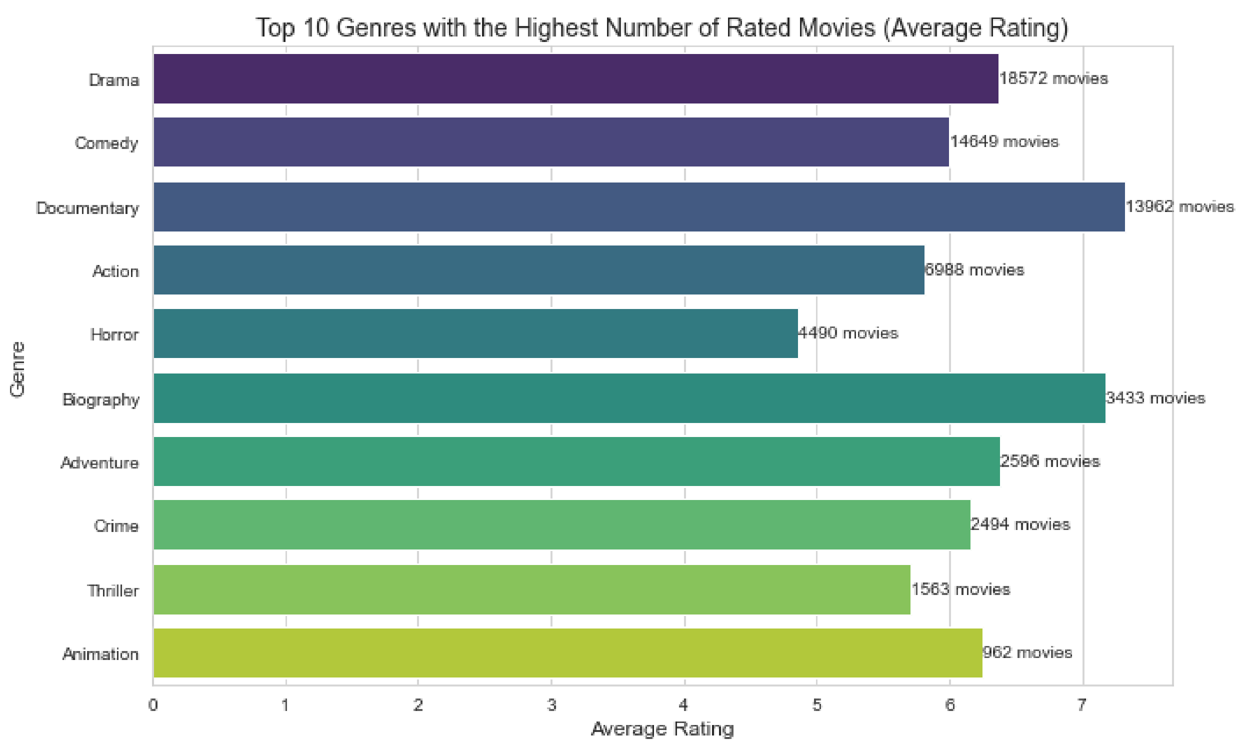

```python
# Calculate the average number of votes per genre
average_votes_by_genre = movie_data.groupby('genres')
['numvotes'].mean().sort_values(ascending=False)

# Plot average number of votes by genre
plt.figure(figsize=(10, 6))
average_votes_by_genre.plot(kind='bar', color='salmon')
plt.title("Average Number of Votes by Genre")
plt.xlabel("Genre")
```

```python
plt.ylabel("Average Vote Count")
plt.show()
```




```python
# Calculate average ratings for the top 10 genres with the highest
number of rated movies
top_rated_genres_query = """
SELECT mb.genres, COUNT(DISTINCT mb.movie_id) AS num_movies_rated,
AVG(mr.averagerating) AS avg_rating
FROM movie_basics mb
INNER JOIN movie_ratings mr ON mb.movie_id = mr.movie_id
GROUP BY mb.genres
ORDER BY num_movies_rated DESC
LIMIT 10;
"""

# Execute the query and load the result into a DataFrame
top_10_genres = pd.read_sql(top_rated_genres_query, conn)

# Plot using Seaborn
plt.figure(figsize=(10, 6))
sns.barplot(
    data=top_10_genres,
```

```
    x='avg_rating',
    y='genres',
    palette='viridis'  # Color palette similar to 'Viridis' in Plotly
)

# Customize the plot
plt.title("Top 10 Genres with the Highest Number of Rated Movies
(Average Rating)", fontsize=14)
plt.xlabel("Average Rating", fontsize=12)
plt.ylabel("Genre", fontsize=12)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)

# Annotate bars with the number of rated movies
for index, row in top_10_genres.iterrows():
    plt.text(row['avg_rating'], index, f"{row['num_movies_rated']}
movies", va='center', ha='left', fontsize=10)

# Show the plot
plt.tight_layout()
plt.show()
```



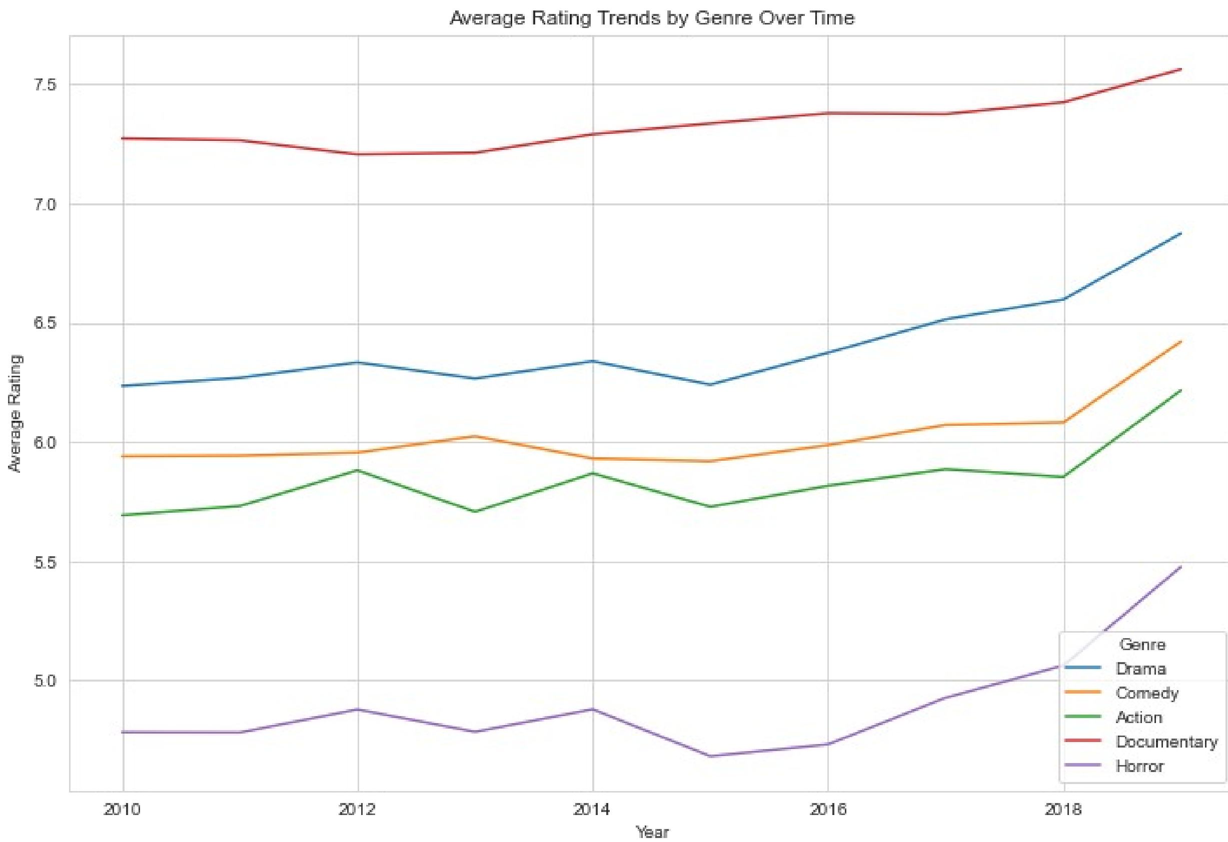
Multivariate Analysis

Objective:

- Analyze how genre popularity has evolved over time by calculating the average rating per genre per year.

- Identify the relationship between runtime_minutes, average rating and genres

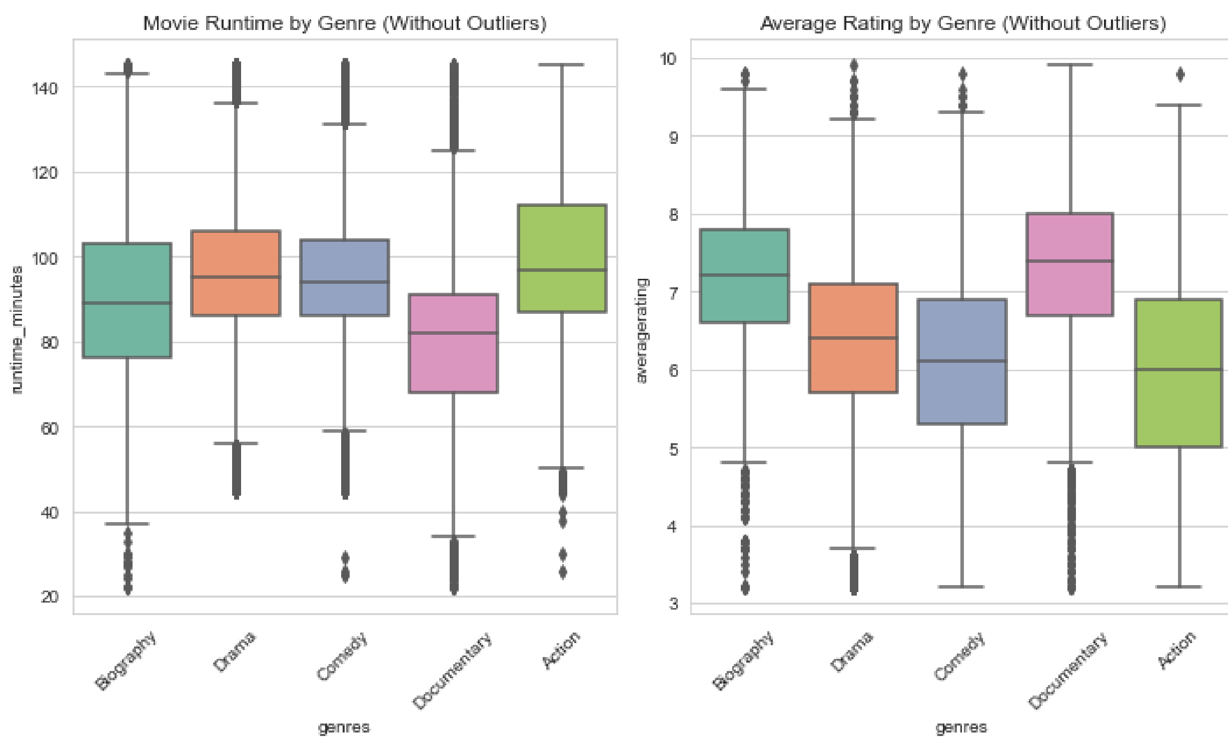
```
# Calculate the average rating by genre and year
genre_yearly_ratings = movie_data.groupby(['start_year', 'genres'])
['averagerating'].mean().unstack()

# Select a few popular genres to plot over time
selected_genres = ['Drama', 'Comedy', 'Action', 'Documentary',
'Horror',]  # Adjust based on genre distribution
genre_yearly_ratings[selected_genres].plot(figsize=(12, 8))
plt.title("Average Rating Trends by Genre Over Time")
plt.xlabel("Year")
plt.ylabel("Average Rating")
plt.legend(title="Genre")
plt.show()
```




```
# Visualize the top 10 genres and the relationship between
runtime_minutes, avg_rating and genre, while excluding outliers
# Identify the top 10 genres by the number of movies or average rating
# First, count the number of movies per genre
genre_counts = movie_basics['genres'].value_counts().head(5).index

# Filter the movie_basics and movie_ratings datasets to only include
these top 10 genres
```

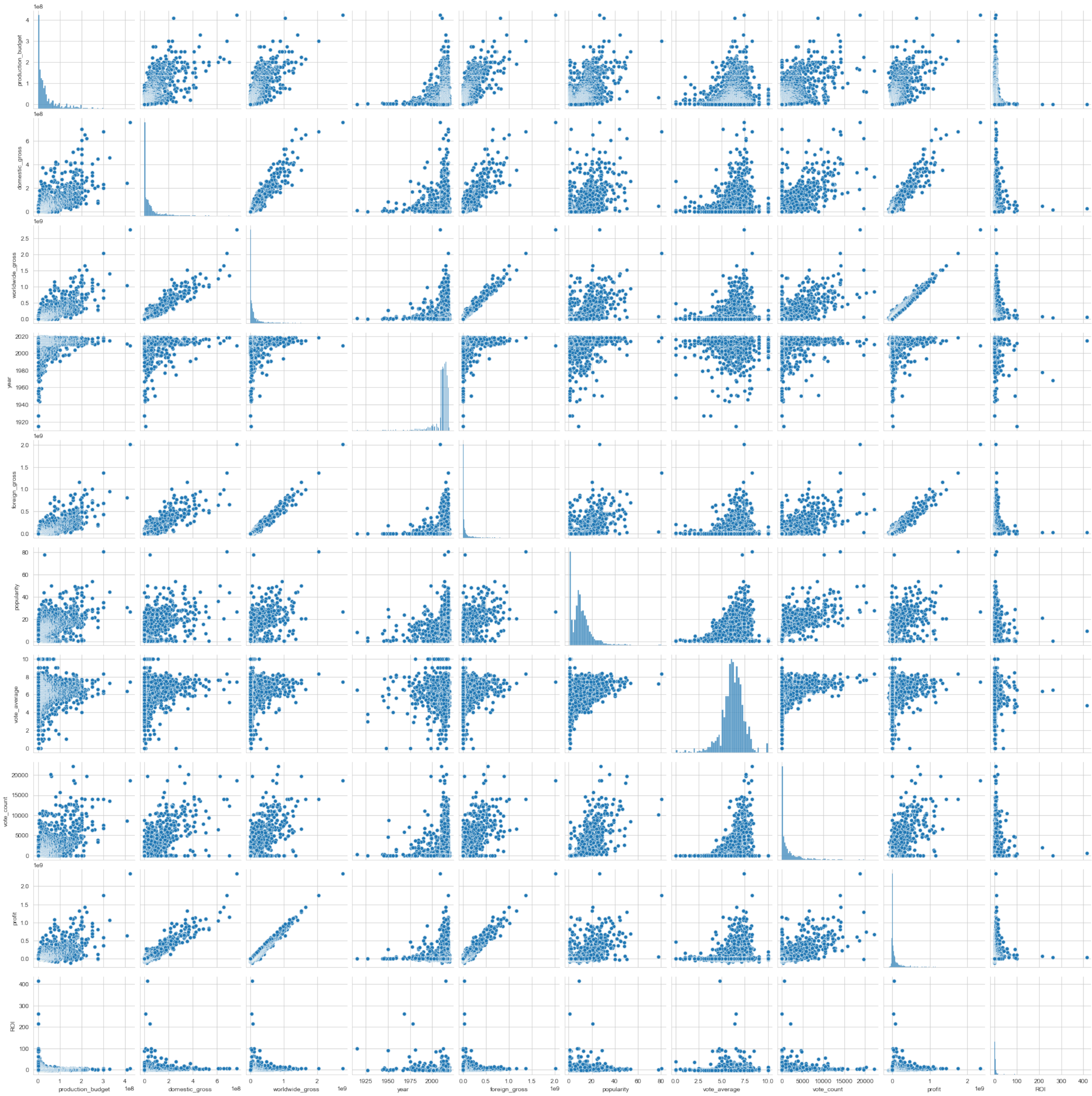
```
top_10_genres_data =
movie_basics[movie_basics['genres'].isin(genre_counts)]

# Filter the movie_ratings to include only movies in top 10 genres
top_10_ratings_data =
movie_ratings[movie_ratings['movie_id'].isin(top_10_genres_data['movie
_id'])]

# Calculate the IQR to remove outliers from both runtime_minutes and
avg_rating
# For runtime_minutes:
Q1_runtime = top_10_genres_data['runtime_minutes'].quantile(0.25)
Q3_runtime = top_10_genres_data['runtime_minutes'].quantile(0.75)
IQR_runtime = Q3_runtime - Q1_runtime
lower_val_runtime = Q1_runtime - 1.5 * IQR_runtime
upper_val_runtime = Q3_runtime + 1.5 * IQR_runtime

# For avg_rating:
Q1_rating = top_10_ratings_data['averagerating'].quantile(0.25)
Q3_rating = top_10_ratings_data['averagerating'].quantile(0.75)
IQR_rating = Q3_rating - Q1_rating
lower_val_rating = Q1_rating - 1.5 * IQR_rating
upper_val_rating = Q3_rating + 1.5 * IQR_rating

# Filter out outliers based on IQR method for runtime_minutes and
avg_rating
filtered_top_10_runtime_data =
top_10_genres_data[(top_10_genres_data['runtime_minutes'] >=
lower_val_runtime) &

(top_10_genres_data['runtime_minutes'] <= upper_val_runtime)]

filtered_top_10_rating_data =
top_10_ratings_data[(top_10_ratings_data['averagerating'] >=
lower_val_rating) &

(top_10_ratings_data['averagerating'] <= upper_val_rating)]

# Merge the two dataframes on 'movie_id' to get both runtime_minutes
and averagerating in one dataframe
merged_filtered_data =
pd.merge(filtered_top_10_runtime_data[['movie_id', 'runtime_minutes',
'genres']],

filtered_top_10_rating_data[['movie_id', 'averagerating']],
                                on='movie_id')

#Plot the relationship between runtime_minutes and avg_rating by genre
using boxplots
plt.figure(figsize=(10, 6))
```

```python
# Boxplot for runtime_minutes by genre
plt.subplot(1, 2, 1)
sns.boxplot(x='genres', y='runtime_minutes',
data=merged_filtered_data, palette='Set2')
plt.xticks(rotation=45)  # Rotate genre names for readability
plt.title('Movie Runtime by Genre (Without Outliers)')

# Boxplot for averagerating by genre
plt.subplot(1, 2, 2)
sns.boxplot(x='genres', y='averagerating', data=merged_filtered_data,
palette='Set2')
plt.xticks(rotation=45)  # Rotate genre names for readability
plt.title('Average Rating by Genre (Without Outliers)')

# Adjust layout
plt.tight_layout()
plt.show()
```



Regression Modelling (Financial Datasets)

```python
# pairplot for linearity comparisons
sns.pairplot(clean_movie_df,palette=colors)
plt.show()
```

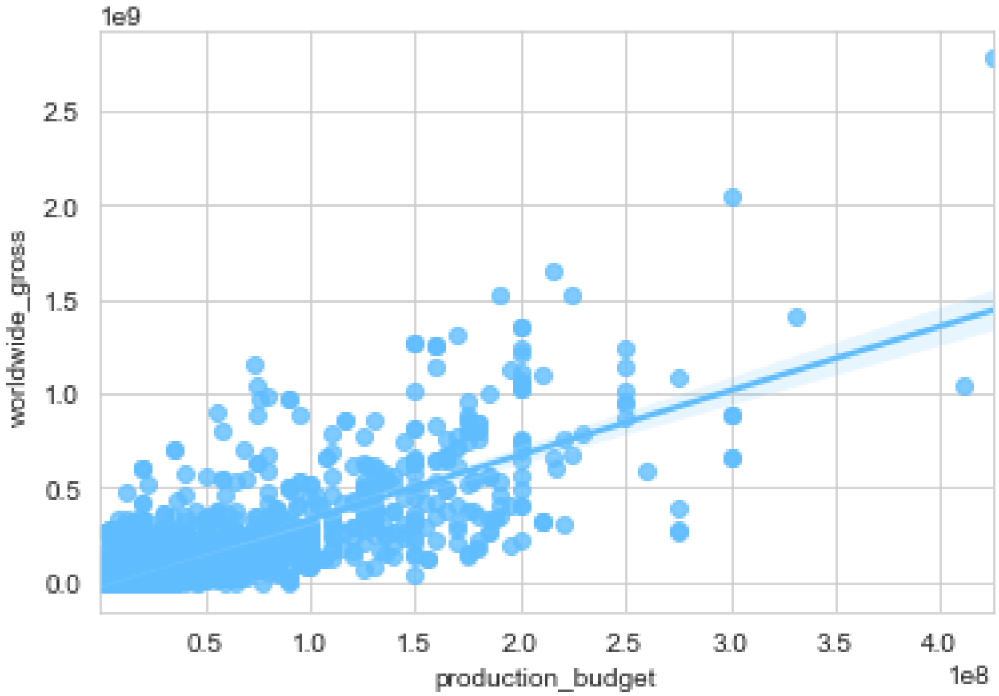


Predictive Modelling

```
# Modelling
X = clean_movie_df[['production_budget']]
y = clean_movie_df['worldwide_gross']

model = sm.OLS(endog=y, exog=sm.add_constant(X))

results = model.fit()

results.summary()

<class 'statsmodels.iolib.summary.Summary'>
"""
```

```
                                 OLS Regression Results

==============================================================================
========
Dep. Variable:        worldwide_gross   R-squared:
0.630
Model:                            OLS   Adj. R-squared:
0.630
Method:                 Least Squares   F-statistic:
4168.
Date:                Fri, 15 Nov 2024   Prob (F-statistic):
0.00
Time:                        22:03:07   Log-Likelihood:
-49233.
No. Observations:                2446   AIC:
9.847e+04
Df Residuals:                    2444   BIC:
9.848e+04
Df Model:                           1

Covariance Type:            nonrobust

==============================================================================
===============
                        coef    std err          t      P>|t|
[0.025      0.975]
------------------------------------------------------------------------------
---------------
const             -9.28e+06   3.35e+06     -2.771      0.006   -
1.58e+07   -2.71e+06
production_budget     3.4095      0.053     64.558      0.000
3.306       3.513
==============================================================================
========
Omnibus:                     1402.780   Durbin-Watson:
1.134
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
27164.448
Skew:                           2.309   Prob(JB):
0.00
Kurtosis:                      18.659   Cond. No.
7.87e+07
==============================================================================
========

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 7.87e+07. This might indicate that
there are
```

```
strong multicollinearity or other numerical problems.
"""

sns.regplot(x="production_budget", y="worldwide_gross",
data=clean_movie_df,color='#5FBDFF');
```




```
results.params

const                    -9.280052e+06
production_budget         3.409485e+00
dtype: float64


# Predict for a new production budget
new_production_budget = 250000000  # Define the new production budget

# Use the model to predict the worldwide gross for the given
production budget
predicted_worldwide_gross = new_production_budget * results.params[1]
+ results.params[0]

# Output the predicted value
print(f"Predicted Worldwide Gross for a Production Budget of $
{new_production_budget:}: ${predicted_worldwide_gross:,}")

Predicted Worldwide Gross for a Production Budget of $250000000:
$843,091,276.3123922
```

Model insights

- model explains about 63% of variance in worldwide-gross
- p-value is 0.0 which is less than alpha = 0.05, model is stastistically significant

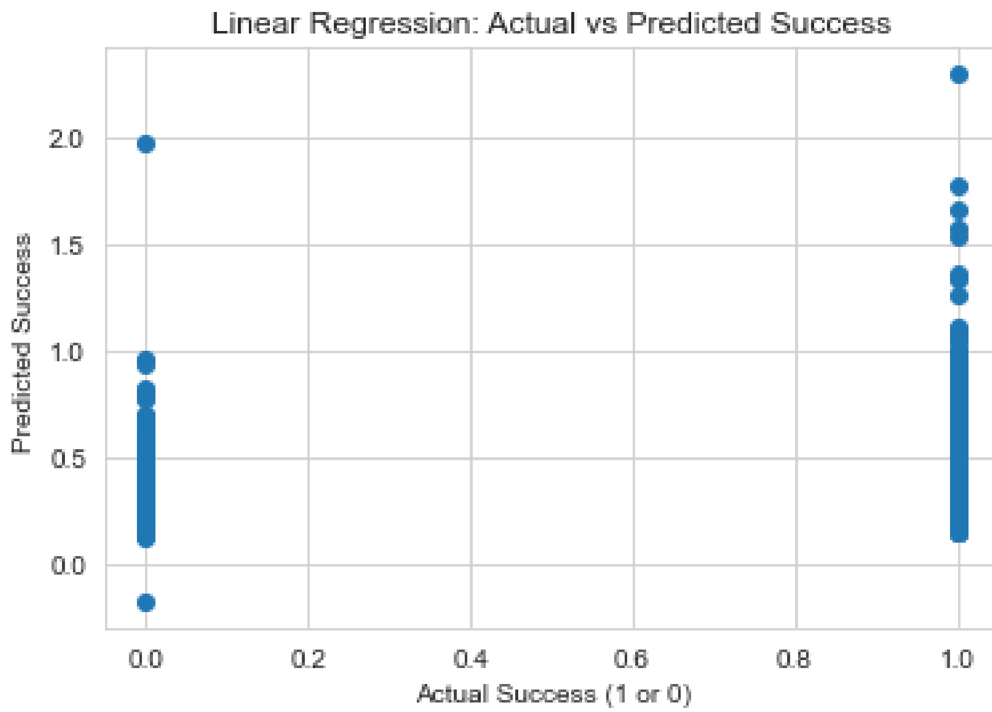
Logistic Regression Modelling on IMDB Database

```
# Merge movie_basics and movie_ratings by movie_id
data = pd.merge(movie_basics, movie_ratings, on='movie_id',
how='left')
# Create the 'success' column based on averagerating (1 if rating > 7,
else 0)
success_threshold = 7
data['success'] = np.where(data['averagerating'] > success_threshold,
1, 0)
# Create movie age (current year - start year)
data['movie_age'] = 2024 - data['start_year']  # Assuming current year
is 2024
#  Add a constant (intercept) term to the model
X = sm.add_constant(X)
# Define the target variable 'success' (dependent variable)
y = data['success']

# Train the Linear Regression model (OLS - Ordinary Least Squares)
model = sm.OLS(y, X)  # OLS model for linear regression
result = model.fit()  # Fit the model

# Output model summary
print(result.summary())

# Make predictions (continuous values, not probabilities)
y_pred = result.predict(X)

# Evaluate the model using R-squared and Mean Squared Error
r2 = result.rsquared  # R-squared value
mse = np.mean((y - y_pred)**2)  # Mean Squared Error
print(f'R-squared: {r2}')
print(f'Mean Squared Error: {mse}')
# Visualize the predictions vs actual values (optional)
plt.scatter(y, y_pred)
plt.xlabel('Actual Success (1 or 0)')
plt.ylabel('Predicted Success')
plt.title('Linear Regression: Actual vs Predicted Success')
plt.show()
```

```
                            OLS Regression Results

==============================================================================
Dep. Variable:                success   R-squared:
0.005
```

```
Model:                            OLS   Adj. R-squared:
0.005
Method:                 Least Squares   F-statistic:
253.1
Date:                Fri, 15 Nov 2024   Prob (F-statistic):
8.48e-164
Time:                        23:26:24   Log-Likelihood:
-62483.
No. Observations:              140736   AIC:
1.250e+05
Df Residuals:                  140732   BIC:
1.250e+05
Df Model:                           3

Covariance Type:            nonrobust

======================================================================
=============
                      coef    std err          t      P>|t|
[0.025      0.975]
----------------------------------------------------------------------
-------------
const               0.1354      0.004     37.177      0.000
0.128       0.142
runtime_minutes  3.477e-05   6.71e-06      5.184      0.000    2.16e-
05    4.79e-05
numvotes         1.142e-06   4.57e-08     24.996      0.000    1.05e-
06    1.23e-06
movie_age           0.0035      0.000      9.408      0.000
0.003       0.004
======================================================================
========
Omnibus:                    37656.288   Durbin-Watson:
1.967
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
75909.940
Skew:                           1.726   Prob(JB):
0.00
Kurtosis:                       4.015   Cond. No.
8.04e+04
======================================================================
========

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 8.04e+04. This might indicate that
there are
strong multicollinearity or other numerical problems.
```

```
R-squared: 0.005365633581671969
Mean Squared Error: 0.1422833282018419
```



Insights and Recommendations on Financial Datasets

Production budgets

- Average production budgets for all genres are $37.55 million.
- However, the top 5 most expensive movies to produce have a budget of more than $ 50 million.
- Despite the high production budgets some of these genres are profitable and have good ROIs.

Profitability by Genres

- Top 5 most profitable genres are Animation, Family, Adventure, Action and Sci-Fi
- Based on profitability, the company should consider investing in these genres Animation, Family, Adventure,or Action as they generate profits above $150 million when compared to the mean of 81.2 million

ROI by Genre

- Top 5 genres with the best return on investment are Horror, Family, Fantasy, Mystery, Romance and Sci-Fi.
- Horror has the best ROI.
- The company should consider Family or Sci-Fi, as they are among top performing genres in terms of profit and ROI.

- Despite Horror having the best ROI, the votecounts and popularity suggests that it might be a niched market, meaning producing the horror will be for a specific demographic.

Factors affecting profitability

Some key insights from correlation analysis were : **Production budget vs gross earnings**

- Observed to have strong positive correlation(>0.7) with gross earnings. This suggests that higher spending on production often lead to higher revenue

  Domestic Gross vs Foreign Gross

- Observed to have strong positive correlation of 0.87. This suggests that movies that do well domestically tend to perform well internationally.

  Vote average (audience score) vs earnings

- Observed to generally have a weak correlation. However this may also be affected by other factors such as genre

Insights and Evaluation on IMDB Database

Summarizing Key Findings

For each of the objectives, I've summarized the main insights.

Insights and Evaluation

1. **Popular Genres**: Genres like Drama, Comedy, and Action are the most common, indicating strong general audience engagement.
2. **High-Rated Genres**: Genres such as Documentary and Biography have higher ratings, suggesting a preference for quality in these categories.
3. **Rising Genre Trends**: Genres like Science Fiction and Thriller show increasing ratings over time, suggesting growing audience interest.
4. **Successful Genre Combinations**: Action-Comedy and Drama-Thriller combinations show high ratings, indicating successful blending of audience-favorite themes.

Conclusions and Recommendations

1. **Invest in High-Rated Niche Genres**: Given the high ratings in Documentary and Biography, consider producing quality films in these genres to appeal to a selective audience.
2. **Leverage Broad-Engagement Genres**: Action and Adventure genres have strong audience engagement, as indicated by vote counts, suggesting they are safe for larger audience appeal.
3. **Explore Successful Genre Combinations**: Hybrid genres like Action-Comedy could attract a wide range of viewers, combining popular genres in a single film.

Document Limitations

I've highlighted limitations (e.g., limited data on recent movies, potential bias in genre labels) and suggest areas for future research.

Limitations

1. **Data Limitations**: Some genres are underrepresented in recent years, possibly affecting trend analysis.
2. **Potential Bias**: IMDb ratings may reflect a subset of audience preferences and could vary by region. Subsequently, there was potential bias in genre labels and selections.