# 1 Business Understanding

## 1.1 Project Overview

This project outlines analysis based on movie datasets for business-stakeholders new to the movie industry.

## 1.2 Business problem:

Your company now sees all the big companies creating original video content and they want to get in on the fun. They have decided to create a new movie studio, but they don't know anything about creating movies. You are charged with exploring what types of films are currently doing the best at the box office. You must then translate those findings into actionable insights that the head of your company's new movie studio can use to help decide what type of films to create.

## 1.3 Project objectives:

### 1.3.1 Main Objective

To analyze movie data and uncover patterns in sales, popularity, ratings, and director influence across genres, providing actionable insights for business growth and strategy.

### 1.3.2 Specific Objectives

1. Identify which genres generate the most revenue and analyze trends contributing to their sales performance.
2. Understand which genres are most popular among audiences and explore factors driving their popularity. `
3. Examine the ratings of top genres to evaluate their critical reception.
4. Identify top selling movies with their related genre.
5. Determine most popular directors
6. Identify the top selling directors
7. Idenfity top popular movie ratings

### 1.3.3 The Data

We used the folder `zippedData` that are movie datasets from the following websites:

- [Box Office Mojo (https://www.boxofficemojo.com/)](https://www.boxofficemojo.com/)
- [IMDB (https://www.imdb.com/)](https://www.imdb.com/)
- [Rotten Tomatoes (https://www.rottentomatoes.com/)](https://www.rottentomatoes.com/)
- [TheMovieDB (https://www.themoviedb.org/)](https://www.themoviedb.org/)
- [The Numbers (https://www.the-numbers.com/)](https://www.the-numbers.com/)

Here are the datasets our crucial for analysis: `bom.movie_gross.csv` , `rt.movie_info.tsv` , `tmdb.movies.csv` a sql database: `im.db` where tables considered were; `movie_basics` , `movie_rating`

# 2 Data Understanding

Here will need to understand our data. This involves getting the relevant information from each dataset crucial for our analysis.

We start by loading the various datasets reviewing their various information based on the columns and check which information is necessary for our analysis before beginning the data cleaning.

In [1]:
```
#importing libraries for data manipulation (pandas, numpy) and visualizatio
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
import sqlite3
import warnings
import numpy as np
import pandas as pd
from scipy.stats import norm
import statsmodels.api as sm
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_sco


# Suppress all warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
# set the maximum number of columns to 40 to display all columns
pd.set_option('display.max_columns', 40)
```

Loading information from **rt.movie_info.tsv** dataset

In [3]:
```python
#Loading the dataset and checking the top five columns
movie_df = pd.read_csv('Datasets/rt.movie_info.tsv', sep='\t')
movie_df.head()
```

Out[3]:

| | id | synopsis | rating | genre | director | writer | theater_ |
|---|---|---|---|---|---|---|---|
| **0** | 1 | This gritty, fast-paced, and innovative police... | R | Action and Adventure\|Classics\|Drama | William Friedkin | Ernest Tidyman | Oct |
| **1** | 3 | New York City, not-too-distant-future: Eric Pa... | R | Drama\|Science Fiction and Fantasy | David Cronenberg | David Cronenberg\|Don DeLillo | Aug 1 |
| **2** | 5 | Illeana Douglas delivers a superb performance ... | R | Drama\|Musical and Performing Arts | Allison Anders | Allison Anders | Sep 1 |
| **3** | 6 | Michael Douglas runs afoul of a treacherous su... | R | Drama\|Mystery and Suspense | Barry Levinson | Paul Attanasio\|Michael Crichton | Dec |
| **4** | 7 | NaN | NR | Drama\|Romance | Rodney Bennett | Giles Cooper | |

In [4]:
```python
#checking the shape getting information of the rows and columns
movie_df.shape
```

Out[4]: (1560, 12)

In [5]:
```
#checking information  for each column of the dataset
movie_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1560 entries, 0 to 1559
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   id            1560 non-null   int64 
 1   synopsis      1498 non-null   object
 2   rating        1557 non-null   object
 3   genre         1552 non-null   object
 4   director      1361 non-null   object
 5   writer        1111 non-null   object
 6   theater_date  1201 non-null   object
 7   dvd_date      1201 non-null   object
 8   currency      340 non-null    object
 9   box_office    340 non-null    object
 10  runtime       1530 non-null   object
 11  studio        494 non-null    object
dtypes: int64(1), object(11)
memory usage: 146.4+ KB
```

Loading the **bom.movie_gross.csv**

In [6]:
```
#Loading the dataset
gross_df = pd.read_csv("Datasets/bom.movie_gross.csv")
```

In [7]:
```
#checking the top 5 columns
gross_df.head()
```

Out[7]:

| | 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 |

In [8]:
```
#Viewing information for each column
gross_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
```

In [9]:
```
#checking the statistical information for numerical columns
gross_df.describe()
```

Out[9]:

| | domestic_gross | year |
|---|---|---|
| **count** | 3.359000e+03 | 3387.000000 |
| **mean** | 2.874585e+07 | 2013.958075 |
| **std** | 6.698250e+07 | 2.478141 |
| **min** | 1.000000e+02 | 2010.000000 |
| **25%** | 1.200000e+05 | 2012.000000 |
| **50%** | 1.400000e+06 | 2014.000000 |
| **75%** | 2.790000e+07 | 2016.000000 |
| **max** | 9.367000e+08 | 2018.000000 |

Loading the **tmdb.movies.csv** dataset

In [10]:
```
#Loading the dataset and viewing the first five columns
tmdb_df = pd.read_csv("Datasets/tmdb.movies.csv")
tmdb_df.head()
```

Out[10]:

| | Unnamed: 0 | genre_ids | id | original_language | original_title | popularity | releas |
|---|---|---|---|---|---|---|---|
| 0 | 0 | [12, 14, 10751] | 12444 | en | Harry Potter and the Deathly Hallows: Part 1 | 33.533 | 2 |
| 1 | 1 | [14, 12, 16, 10751] | 10191 | en | How to Train Your Dragon | 28.734 | 2 |
| 2 | 2 | [12, 28, 878] | 10138 | en | Iron Man 2 | 28.515 | 2 |
| 3 | 3 | [16, 35, 10751] | 862 | en | Toy Story | 28.005 | 1 |
| 4 | 4 | [28, 878, 12] | 27205 | en | Inception | 27.920 | 2 |

In [11]:
```
#Checking for information of each column from the dataset
tmdb_df.info()
```

```
<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
```

In [12]:
```
#Viewing the statistical information
tmdb_df.describe()
```

Out[12]:

| | Unnamed: 0 | id | popularity | vote_average | vote_count |
|---|---|---|---|---|---|
| **count** | 26517.00000 | 26517.000000 | 26517.000000 | 26517.000000 | 26517.000000 |
| **mean** | 13258.00000 | 295050.153260 | 3.130912 | 5.991281 | 194.224837 |
| **std** | 7654.94288 | 153661.615648 | 4.355229 | 1.852946 | 960.961095 |
| **min** | 0.00000 | 27.000000 | 0.600000 | 0.000000 | 1.000000 |
| **25%** | 6629.00000 | 157851.000000 | 0.600000 | 5.000000 | 2.000000 |
| **50%** | 13258.00000 | 309581.000000 | 1.374000 | 6.000000 | 5.000000 |
| **75%** | 19887.00000 | 419542.000000 | 3.694000 | 7.000000 | 28.000000 |
| **max** | 26516.00000 | 608444.000000 | 80.773000 | 10.000000 | 22186.000000 |

Connecting to the SQL database

In [13]:
```
#connecting to the sql database
conn = sqlite3.Connection('Datasets/im.db')
```

In [14]:
```
#getting table names
cursor = conn.cursor()
cursor.execute("""SELECT name
    FROM sqlite_master
    WHERE type = 'table';""")
print(cursor.fetchall())
```

```
[('movie_basics',), ('directors',), ('known_for',), ('movie_akas',), ('mov
ie_ratings',), ('persons',), ('principals',), ('writers',)]
```

In [15]:
```
#Loading the movie basics table
mbasics_df = pd.read_sql("""SELECT * FROM movie_basics;""",conn)
```

In [16]:
```
#Checking the top five columns
mbasics_df.head()
```

Out[16]:

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

In [17]:
```
#Checking for the information of each column
mbasics_df.info()
```

```
<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
```

In [18]:
```
#Loading the the movie_ratings table from the database
rating_df = pd.read_sql("""SELECT * FROM movie_ratings;""",conn)
```

In [19]:
```
#Checking the top 5 columns
rating_df.head()
```

Out[19]:

| | 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 |

In [20]:
```
#Chcecking for the information from each column
rating_df.info()
```

```
<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
```

In [21]:
```
#checking for the statistical information
rating_df.describe()
```

Out[21]:

| | averagerating | numvotes |
|---|---|---|
| **count** | 73856.000000 | 7.385600e+04 |
| **mean** | 6.332729 | 3.523662e+03 |
| **std** | 1.474978 | 3.029402e+04 |
| **min** | 1.000000 | 5.000000e+00 |
| **25%** | 5.500000 | 1.400000e+01 |
| **50%** | 6.500000 | 4.900000e+01 |
| **75%** | 7.400000 | 2.820000e+02 |
| **max** | 10.000000 | 1.841066e+06 |

In [22]:
```
directors_df = pd.read_sql("""SELECT * FROM directors;""",conn)
```

In [23]:
```
directors_df.head()
```

Out[23]:

| | movie_id | person_id |
|---|---|---|
| **0** | tt0285252 | nm0899854 |
| **1** | tt0462036 | nm1940585 |
| **2** | tt0835418 | nm0151540 |
| **3** | tt0835418 | nm0151540 |
| **4** | tt0878654 | nm0089502 |

In [24]:
```
directors_df.tail()
```

Out[24]:

| | movie_id | person_id |
|---|---|---|
| **291169** | tt8999974 | nm10122357 |
| **291170** | tt9001390 | nm6711477 |
| **291171** | tt9001494 | nm10123242 |
| **291172** | tt9001494 | nm10123248 |
| **291173** | tt9004986 | nm4993825 |

In [25]:
```
directors_df.info()
```

```
<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
```

In [26]:
```
conn.close()
```

# 3 Data Preparation

## 3.1 Data Cleaning

After looking into our dataset and reviewing what categorical and numerical data we will require we begin our data cleaning process by:

1. Dropping columns unecessary for our analysis
2. Checking for missing values
3. Rectifying column arrangement and uniformity
4. Checking for outliers
5. Dropping duplicates
6. Changing the data types

## 3.2 Cleaning for the movie_df dataset

### 3.2.1 Dropping unecessary columns

In [27]:
```
# drop those columns with more than 1000 non-null rows
movie_df = movie_df.drop(['currency', 'studio','writer','synopsis','runtime
```

### 3.2.2 Checking for missing values

In [28]:
```
#Checking for missing values in each column
movie_df.isna().sum()
```

Out[28]:
```
id                0
rating            3
genre             8
director        199
theater_date    359
box_office     1220
dtype: int64
```

In [29]:
```
#replacing movie genre nulls with mode and confirming the changes
genre_mode = movie_df.genre.mode()[0]
movie_df.genre.fillna(genre_mode, inplace=True)
movie_df.genre.isna().sum()
```

Out[29]:
```
0
```

In [30]:
```
#replacing movie rating nulls with mode
rating_mode = movie_df.rating.mode()[0]
movie_df.rating.fillna(rating_mode, inplace=True)
movie_df.rating.isna().sum()
```

Out[30]:
```
0
```

In [31]:
```
#drop the rest with nulls
movie_df.dropna(inplace=True)
```

In [32]:
```
#Confirming that there are no null values
movie_df.isnull().sum().any()
```

Out[32]:
```
False
```

In [33]:
```
#Renaming columns
movie_df.rename(columns={'theater_date':'year'},inplace=True)
```

In [34]:
```
#changing the box office values to numerical(box office sales a movie makes
#noted it was not numerical since it gave out an error when trying to fill r
# Step 1: Remove commas
movie_df['box_office'] = movie_df['box_office'].str.replace(',', '')

# Step 2: Convert to integers
movie_df['box_office'] = movie_df['box_office'].astype(int)
```

In [35]:
```
#Confirming changes to the dataset
movie_df.head()
```

Out[35]:

| | id | rating | genre | director | year | box_office |
|---|---|---|---|---|---|---|
| **1** | 3 | R | Drama\|Science Fiction and Fantasy | David Cronenberg | Aug 17, 2012 | 600000 |
| **6** | 10 | PG-13 | Comedy | Jake Kasdan | Jan 11, 2002 | 41032915 |
| **7** | 13 | R | Drama | Ray Lawrence | Apr 27, 2006 | 224114 |
| **8** | 14 | R | Drama | Taylor Hackford | Jun 30, 2010 | 134904 |
| **15** | 22 | R | Comedy\|Drama\|Mystery and Suspense | George Hickenlooper | Dec 17, 2010 | 1039869 |

## 3.3 Cleaning for the gross_df dataset

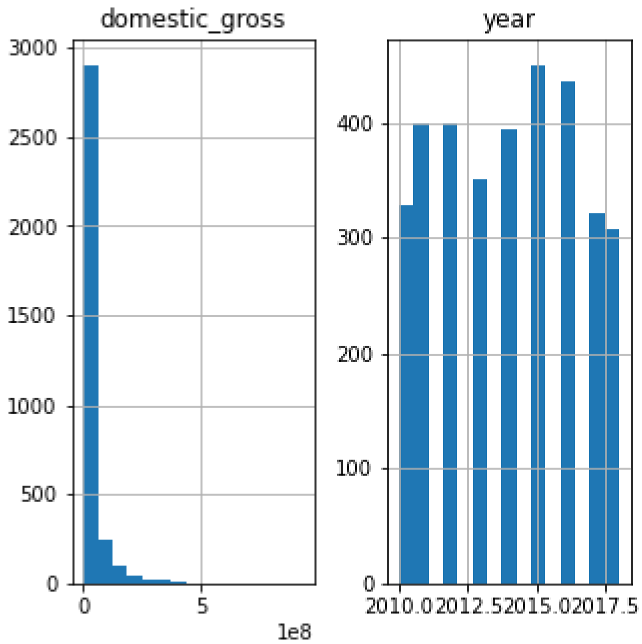
In [36]:
```
#Dropping unnecessary columns
gross_df.drop(["studio"],axis=1,inplace=True)
```

### 3.3.1 Checking for missing values

In [37]:
```
#checking for any missng values
gross_df.isna().sum()
```

Out[37]:
```
title               0
domestic_gross     28
foreign_gross    1350
year                0
dtype: int64
```

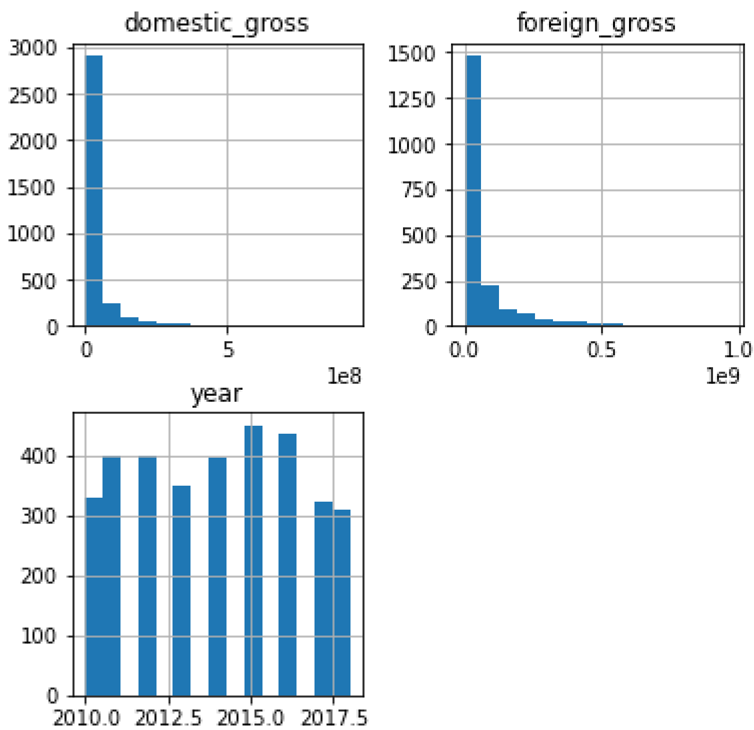
In [38]:
```
#checking for the distribution for the numerical values
gross_df.hist(bins=15,figsize=(5,5));
```




From the histograms we noticed the foreign_gross column is not numerical so we change its data type to float as shown below:

In [39]:
```
# converting 'foreign_gross' to float
gross_df['foreign_gross'] = pd.to_numeric(gross_df['foreign_gross'],errors=
```

In [40]:
```
#confirming the changes
gross_df.hist(bins=15,figsize=(6,6));
```




We notice that the distribution has skewness hence we use median to fill in the missing values

In [41]:
```
#replacing gross for domestic and foreign with median
gross_df["domestic_gross"]=gross_df["domestic_gross"].fillna(gross_df["dome

gross_df['foreign_gross'] = gross_df['foreign_gross'].fillna(gross_df['fore
```

In [42]:
```
# calculating 'total_gross' as the sum of 'domestic_gross' and 'foreign_gro
gross_df['total_gross'] = gross_df['domestic_gross'] + gross_df['foreign_gr

gross_df[['domestic_gross', 'foreign_gross', 'total_gross']].head()
```

Out[42]:

| | domestic_gross | foreign_gross | total_gross |
|---|---|---|---|
| **0** | 415000000.0 | 652000000.0 | 1.067000e+09 |
| **1** | 334200000.0 | 691300000.0 | 1.025500e+09 |
| **2** | 296000000.0 | 664300000.0 | 9.603000e+08 |
| **3** | 292600000.0 | 535700000.0 | 8.283000e+08 |
| **4** | 238700000.0 | 513900000.0 | 7.526000e+08 |

In [43]:
```
#drop the rest with nulls
gross_df.dropna(inplace=True)
#confirming there are no null values
gross_df.isnull().sum().any()
```

Out[43]:
```
False
```

In [44]:
```
#Changing the numerical columns to currency for uniformity

#gross_df['domestic_gross'] = gross_df['domestic_gross'].apply(lambda x: f"$

#gross_df['foreign_gross'] = gross_df['foreign_gross'].apply(lambda x: f"${x

#gross_df['total_gross'] = gross_df['total_gross'].apply(lambda x: f"${x:,.2
```

In [45]:
```
#adding the id as the  first column
gross_df.insert(0, 'id', range(1, len(gross_df) + 1))
```

In [46]:
```
gross_df.head()
```

Out[46]:

| | id | title | domestic_gross | foreign_gross | year | total_gross |
|---|---|---|---|---|---|---|
| **0** | 1 | Toy Story 3 | 415000000.0 | 652000000.0 | 2010 | 1.067000e+09 |
| **1** | 2 | Alice in Wonderland (2010) | 334200000.0 | 691300000.0 | 2010 | 1.025500e+09 |
| **2** | 3 | Harry Potter and the Deathly Hallows Part 1 | 296000000.0 | 664300000.0 | 2010 | 9.603000e+08 |
| **3** | 4 | Inception | 292600000.0 | 535700000.0 | 2010 | 8.283000e+08 |
| **4** | 5 | Shrek Forever After | 238700000.0 | 513900000.0 | 2010 | 7.526000e+08 |

### 3.3.2 Removing Outliers

In [47]:
```
def remove_outliers(df, column_list):
    for column in column_list:
        # Calculate the first (Q1) and third (Q3) quartiles for the column
        Q1 = df[column].quantile(0.25)
        Q3 = df[column].quantile(0.75)

        # Calculate the IQR
        IQR = Q3 - Q1

        # Define the lower and upper bounds for outliers
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        # Filter the dataframe to keep only the rows within the bounds
        df = df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]

    return df

# List of columns to remove outliers from
columns_to_check = ['foreign_gross','domestic_gross']

# Apply the function to remove outliers
gross_df = remove_outliers(gross_df, columns_to_check)

# Check the dataframe info to verify the results
gross_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2280 entries, 96 to 3386
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   id              2280 non-null   int64  
 1   title           2280 non-null   object 
 2   domestic_gross  2280 non-null   float64
 3   foreign_gross   2280 non-null   float64
 4   year            2280 non-null   int64  
 5   total_gross     2280 non-null   float64
dtypes: float64(3), int64(2), object(1)
memory usage: 124.7+ KB
```

## 3.4 Cleaning for the tmdb_df dataset

### 3.4.1 Dropping columns

In [48]:
```
#Dropping the unecessary columns
tmdb_df.drop(["vote_average","vote_count","genre_ids","id",'original_langua
```

### 3.4.2 Checking for missing values

In [49]:
```
#Checking for the missing values
tmdb_df.isna().sum()
```

Out[49]:
```
Unnamed: 0      0
popularity      0
release_date    0
title           0
dtype: int64
```

In [50]:
```
tmdb_df.head()
```

Out[50]:

| | Unnamed: 0 | popularity | release_date | title |
|---|---|---|---|---|
| **0** | 0 | 33.533 | 2010-11-19 | Harry Potter and the Deathly Hallows: Part 1 |
| **1** | 1 | 28.734 | 2010-03-26 | How to Train Your Dragon |
| **2** | 2 | 28.515 | 2010-05-07 | Iron Man 2 |
| **3** | 3 | 28.005 | 1995-11-22 | Toy Story |
| **4** | 4 | 27.920 | 2010-07-16 | Inception |

### 3.4.3 Changing Columns

In [51]:
```
# Renaming columns in tmdb_df
tmdb_df.rename(columns={'Unnamed: 0': 'id'},inplace=True)

tmdb_df.rename(columns={'release_date':'year'},inplace=True)
```

In [52]:
```
tmdb_df.head()
```

Out[52]:

| | id | popularity | year | title |
|---|---|---|---|---|
| **0** | 0 | 33.533 | 2010-11-19 | Harry Potter and the Deathly Hallows: Part 1 |
| **1** | 1 | 28.734 | 2010-03-26 | How to Train Your Dragon |
| **2** | 2 | 28.515 | 2010-05-07 | Iron Man 2 |
| **3** | 3 | 28.005 | 1995-11-22 | Toy Story |
| **4** | 4 | 27.920 | 2010-07-16 | Inception |

## 3.5 Cleaning for mbasic_df dataset

### 3.5.1 Dropping columns

In [53]:
```
#Dropping  columns unnecessary for our analysis
mbasics_df.drop(["original_title","runtime_minutes"],axis=1,inplace=True)
```

### 3.5.2 Checking for missing values

In [54]:
```
#Checking missing values
mbasics_df.isnull().sum().any
```

Out[54]:
```
<bound method NDFrame._add_numeric_operations.<locals>.any of movie_id
0
primary_title       0
start_year          0
genres           5408
dtype: int64>
```

In [55]:
```
#Filling the missing values for the genres columns which is an object
for column in mbasics_df.select_dtypes(include=["object"]).columns:
    mbasics_df[column].fillna(mbasics_df[column].mode()[0],inplace=True)
```

In [56]:
```
#Confirming for no missing values
mbasics_df.isnull().sum().any()
```

Out[56]:
```
False
```

### 3.5.3 Renaming columns

In [57]:
```
#Renaming the columns
mbasics_df.rename(columns={"movie_id":'id'},inplace=True)

mbasics_df.rename(columns={"start_year":'year'},inplace=True)

mbasics_df.rename(columns={"primary_title":'title'},inplace=True)
```

In [58]:
```
mbasics_df.head()
```

Out[58]:

| | id | title | year | genres |
|---|---|---|---|---|
| **0** | tt0063540 | Sunghursh | 2013 | Action,Crime,Drama |
| **1** | tt0066787 | One Day Before the Rainy Season | 2019 | Biography,Drama |
| **2** | tt0069049 | The Other Side of the Wind | 2018 | Drama |
| **3** | tt0069204 | Sabse Bada Sukh | 2018 | Comedy,Drama |
| **4** | tt0100275 | The Wandering Soap Opera | 2017 | Comedy,Drama,Fantasy |

# 3.6 Cleaning for the rating_df dataset

## 3.6.1 Dropping unnecessary columns

In [59]:
```
#Dropping unecessary columns
rating_df.drop(["numvotes"],axis=1,inplace=True)
```

## 3.6.2 Checking for missing values

In [60]:
```
#Checking for missing values
rating_df.isnull().sum().any()
```

Out[60]: False

In [61]:
```
#Renaming columns
rating_df.rename(columns={"movie_id":"id"},inplace=True)

rating_df.rename(columns={"averagerating":"average_rating"},inplace=True)
```

In [62]:
```
rating_df.head()
```

Out[62]:

| | id | average_rating |
|---|---|---|
| **0** | tt10356526 | 8.3 |
| **1** | tt10384606 | 8.9 |
| **2** | tt1042974 | 6.4 |
| **3** | tt1043726 | 4.2 |
| **4** | tt1060240 | 6.5 |

## 3.6.3 Checking Duplicates

In [63]:
```
#Checking for dupicates for movie_df
movie_df.duplicated().sum()
```

Out[63]: 0

In [64]:
```
#Checking for dupicates for gross_df
gross_df.duplicated().sum()
```

Out[64]: 0

In [65]:
```
#Checking for dupicates for tmdb_df
tmdb_df.duplicated().sum()
```

Out[65]: 0

In [66]:
```
#Checking for dupicates for mbasics_df
mbasics_df.duplicated().sum()
```
Out[66]: 0

In [67]:
```
#Checking for dupicates for the rating_df dataset
rating_df.duplicated().sum()
```
Out[67]: 0

In [68]:
```
directors_df.duplicated().sum()
```
Out[68]: 127639

In [69]:
```
directors_df.drop_duplicates(inplace=True)
directors_df.duplicated().sum()
```
Out[69]: 0

### 3.6.4 Feature engineering

In [70]:
```
# Split 'genre' into 'main_genre' and 'supporting_genre'
movie_df['main_genre'] = movie_df['genre'].str.split('|').str[0]
movie_df['supporting_genre'] = movie_df['genre'].str.split('|').apply(lambda

# Preview the result
movie_df[['genre', 'main_genre', 'supporting_genre']].head()
```
Out[70]:

| | genre | main_genre | supporting_genre |
|---|---|---|---|
| **1** | Drama\|Science Fiction and Fantasy | Drama | Science Fiction and Fantasy |
| **6** | Comedy | Comedy | |
| **7** | Drama | Drama | |
| **8** | Drama | Drama | |
| **15** | Comedy\|Drama\|Mystery and Suspense | Comedy | Drama\|Mystery and Suspense |

In [71]:
```
# Convert 'theater_date' and 'dvd_date' columns to datetime format
movie_df["year"]= pd.to_datetime(movie_df["year"]).dt.year

movie_df[['year']].head()
```

Out[71]:

| | year |
|---|---|
| **1** | 2012 |
| **6** | 2002 |
| **7** | 2006 |
| **8** | 2010 |
| **15** | 2010 |

In [72]:
```
#From the new columns we can drop the columns further for easier analysis
movie_df.drop(["genre","supporting_genre"],axis=1,inplace=True)

#Renaming the remaining column
movie_df.rename({"main_genre":"genre"},axis=1,inplace=True)
```

In [73]:
```
#Confirming changes
movie_df.head()
```

Out[73]:

| | id | rating | director | year | box_office | genre |
|---|---|---|---|---|---|---|
| **1** | 3 | R | David Cronenberg | 2012 | 600000 | Drama |
| **6** | 10 | PG-13 | Jake Kasdan | 2002 | 41032915 | Comedy |
| **7** | 13 | R | Ray Lawrence | 2006 | 224114 | Drama |
| **8** | 14 | R | Taylor Hackford | 2010 | 134904 | Drama |
| **15** | 22 | R | George Hickenlooper | 2010 | 1039869 | Comedy |

In [74]:
```
# convert 'release_date' to year
tmdb_df["year"]= pd.to_datetime(tmdb_df["year"]).dt.year
```

In [75]:
```
# merging the movie_df and gross_df on 'id'
movie_basics_rating_df = pd.merge(mbasics_df, rating_df, on='id', how='inne
```

In [76]:
```
movie_basics_rating_df.head()
```

Out[76]:

| | id | title | year | genres | average_rating |
|---|---|---|---|---|---|
| **0** | tt0063540 | Sunghursh | 2013 | Action,Crime,Drama | 7.0 |
| **1** | tt0066787 | One Day Before the Rainy Season | 2019 | Biography,Drama | 7.2 |
| **2** | tt0069049 | The Other Side of the Wind | 2018 | Drama | 6.9 |
| **3** | tt0069204 | Sabse Bada Sukh | 2018 | Comedy,Drama | 6.1 |
| **4** | tt0100275 | The Wandering Soap Opera | 2017 | Comedy,Drama,Fantasy | 6.5 |

In [77]:
```
#Merging the tm1 dataset("tmdb.movies.csv") to the original merged data set
# First merge: Add tmdb1 to the existing merged DataFrame
movie_basics_rating_df_final = pd.merge(movie_basics_rating_df, tmdb_df, on:

# Second merge: Add bom1 to the updated merged DataFrame using the same colu
movie_basics_rating_df_final = pd.merge(movie_basics_rating_df, gross_df, o
# Check the result
print(movie_basics_rating_df_final.head())
```

```
        id_x          title  year_x                    genres  average_rati
ng  \
0  tt0315642          Wazir    2016        Action,Crime,Drama
7.1
1  tt0337692    On the Road    2012  Adventure,Drama,Romance
6.1
2  tt4339118    On the Road    2014                    Drama
6.0
3  tt5647250    On the Road    2016                    Drama
5.7
4  tt0376136  The Rum Diary    2011             Comedy,Drama
6.2

   id_y  domestic_gross  foreign_gross  year_y  total_gross
0  2569       1100000.0     18900000.0    2016   20000000.0
1   905        744000.0      8000000.0    2012    8744000.0
2   905        744000.0      8000000.0    2012    8744000.0
3   905        744000.0      8000000.0    2012    8744000.0
4   475      13100000.0     10800000.0    2011   23900000.0
```

In [78]:
```
#Changing the merged dataset to a dataframe
merged_movies_final_df=pd.DataFrame(movie_basics_rating_df_final)
merged_movies_final_df
```

Out[78]:

| | id_x | title | year_x | genres | average_rating | id_y |
|---|---|---|---|---|---|---|
| **0** | tt0315642 | Wazir | 2016 | Action,Crime,Drama | 7.1 | 2569 |
| **1** | tt0337692 | On the Road | 2012 | Adventure,Drama,Romance | 6.1 | 905 |
| **2** | tt4339118 | On the Road | 2014 | Drama | 6.0 | 905 |
| **3** | tt5647250 | On the Road | 2016 | Drama | 5.7 | 905 |
| **4** | tt0376136 | The Rum Diary | 2011 | Comedy,Drama | 6.2 | 475 |
| **...** | ... | ... | ... | ... | ... | ... |
| **1965** | tt8290698 | The Spy Gone North | 2018 | Drama | 7.2 | 3303 |
| **1966** | tt8331988 | The Chambermaid | 2018 | Drama | 7.1 | 2322 |

In [79]:
```
#Dropping unnecessary columns for the merged dataset
merged_movies_final_df.drop(columns=["id_x","year_x","id_y"],axis=1,inplace
```

In [80]:
```
#Renaming columns
merged_movies_final_df.rename({"year_y":"year"},axis=1,inplace=True)
```

In [81]:
```
#Confirming the changes
merged_movies_final_df
```

Out[81]:

| | title | genres | average_rating | domestic_gross | foreign_gro |
|---|---|---|---|---|---|
| **0** | Wazir | Action,Crime,Drama | 7.1 | 1100000.0 | 18900( |
| **1** | On the Road | Adventure,Drama,Romance | 6.1 | 744000.0 | 8000( |
| **2** | On the Road | Drama | 6.0 | 744000.0 | 8000( |
| **3** | On the Road | Drama | 5.7 | 744000.0 | 8000( |
| **4** | The Rum Diary | Comedy,Drama | 6.2 | 13100000.0 | 10800( |
| **...** | ... | ... | ... | ... | |
| **1965** | The Spy Gone North | Drama | 7.2 | 501000.0 | 18900( |
| **1966** | The Chambermaid | Drama | 7.1 | 300.0 | 18900( |
| **1967** | Helicopter Eela | Drama | 5.4 | 72000.0 | 18900( |
| **1968** | Last Letter | Drama,Romance | 6.4 | 181000.0 | 18900( |
| **1969** | Burn the Stage: The Movie | Documentary,Music | 8.8 | 4200000.0 | 16100( |

1970 rows × 7 columns

### 3.6.5 Saving Dataset

In [82]:
```
#Saving the merged dataset
merged_movies_final_df.to_csv("Datasets/merged_movies_clean.csv")
```

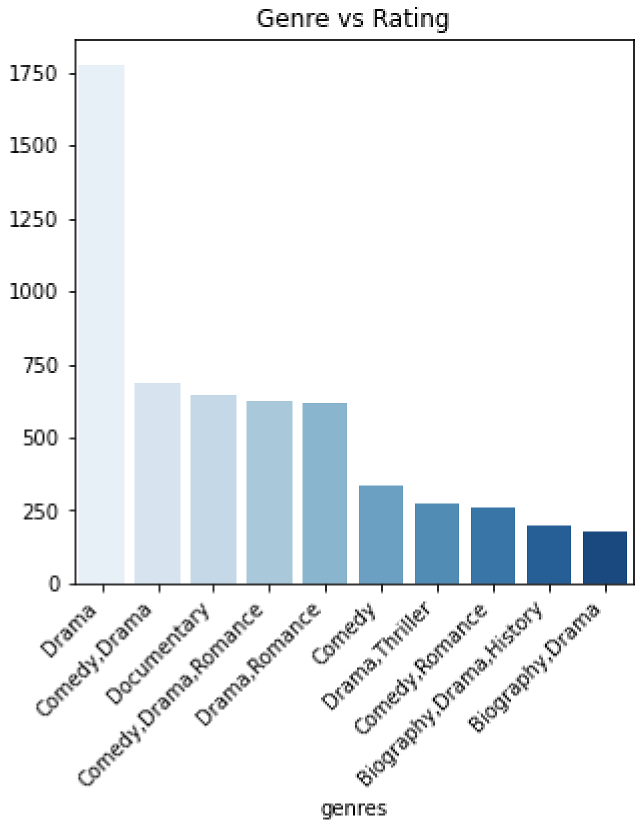
In [83]:
```
#Saving the movie_df dataset
movie_df.to_csv("Datasets/movie_info_clean.csv")
```

# 4 Data Analysis

### 4.0.1 Analysis based on Genre Vs Rating

We will make a visualization for top 10 genres with the highest ratings

In [84]:
```
genre_rating= merged_movies_final_df.groupby("genres")["average_rating"].su
plt.figure(figsize=(5, 5))
sns.barplot(x=genre_rating.index, y=genre_rating.values,palette="Blues")
plt.xticks(rotation=45, ha='right')
plt.title('Genre vs Rating')
plt.show()
```



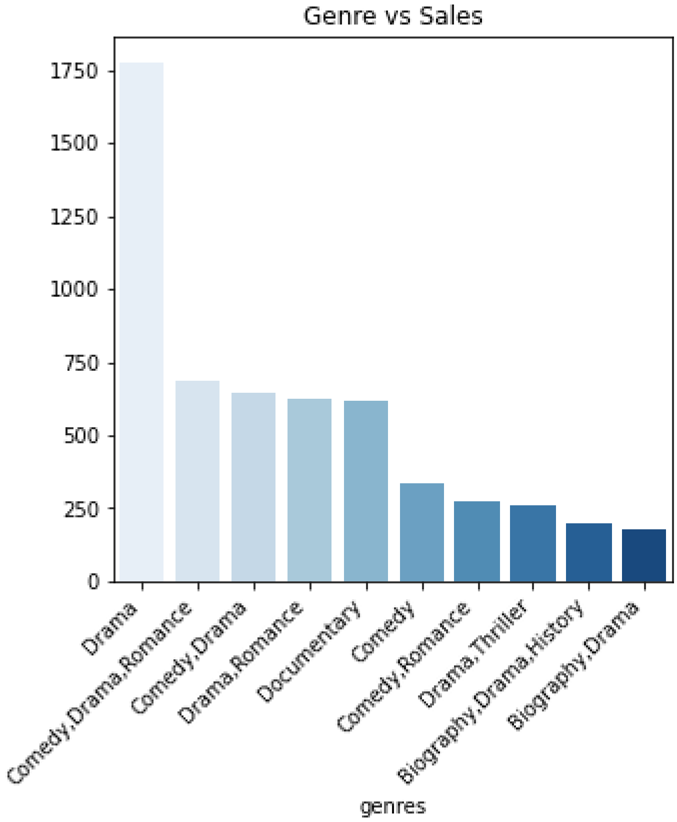

#### 4.0.1.1 Findings:

From the above analysis it is evident that the drama genre is the genre with highest rating with a rating higher than 2000.

### 4.0.2 Analysis based on Genre Vs Sales

Here we will make visualization based on the top 10 highest selling genres based on amount each genre grossed.

In [85]:
```
genre_sales= merged_movies_final_df.groupby("genres")["total_gross"].sum().
plt.figure(figsize=(5, 5))
sns.barplot(x=genre_sales.index, y=genre_rating.values,palette="Blues")
plt.xticks(rotation=45, ha='right')
plt.title('Genre vs Sales')
plt.show()
```



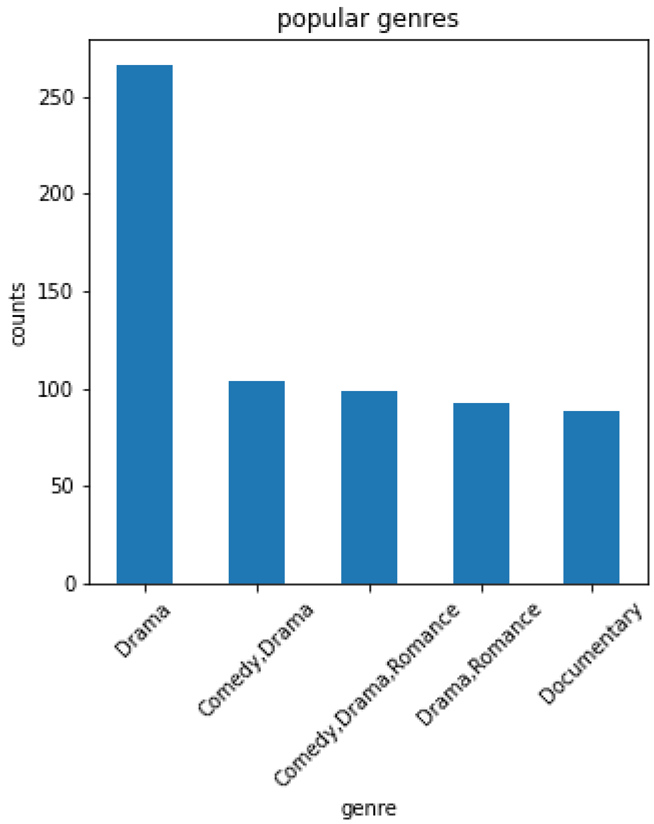

#### 4.0.2.1 Findings:

Here we are able to view that the Adventure, Comedy and Sci-Fi genre has the highest sales. This may be brought by reasons such perfect direction, quality production and entertaining movies from this certain genre.

### 4.0.3 Analysis based on Genre popularity

Here we will make visualizations based on genres with most popularity

In [86]:
```
merged_movies_final_df["genres"].value_counts().head(5).plot(kind='bar',fig
sns.set_palette('Blues')
plt.title("popular genres")
plt.ylabel("counts")
plt.xlabel('genre')
plt.xticks(rotation=45);
```



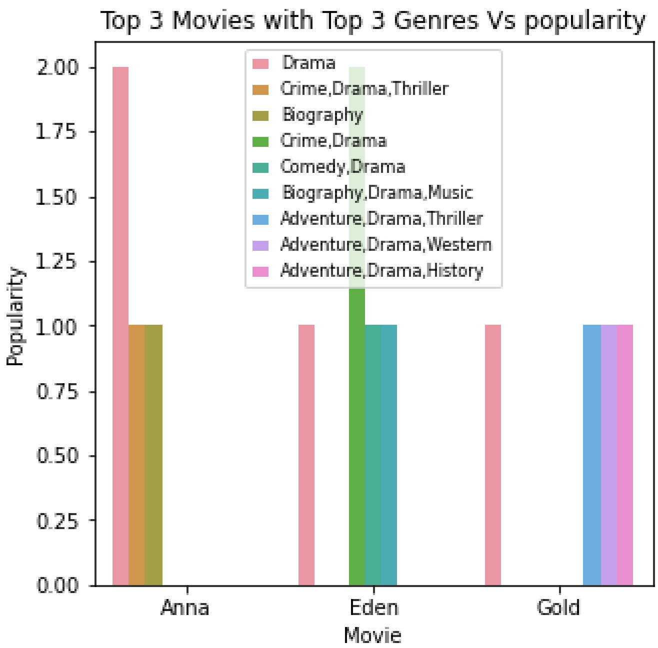

#### 4.0.3.1 Findings:

From the above analysis it is evident that drama genre is also the most popular genre. This may be due audience preferabilty, quality production and perfect direction from the directors associated with this particular genre

### 4.0.4 Analysis on top selling movies with their popular genres

We can further our analysis based on best selling movies by visualizing with their associate genres

In [87]:
```
#top movies based on genre popularity
top_three_movies = merged_movies_final_df["title"].value_counts().head(3).i

filtered_data = merged_movies_final_df[merged_movies_final_df["title"].isin

top_genres=(
    filtered_data.groupby("title")["genres"]
    .value_counts()
    .groupby(level=0).nlargest(3)
    .reset_index(level=0,drop=True)
    .index.get_level_values(1)
)

filtered_data=filtered_data[filtered_data['genres'].isin(top_genres)]

plt.figure(figsize=(5, 5))#select figure size
sns.countplot(data=filtered_data, x='title', hue='genres')#selecting type o
plt.title('Top 3 Movies with Top 3 Genres Vs popularity')#title for the gra
plt.xlabel('Movie')#x-axis label
plt.ylabel('Popularity')#y-axis label
plt.legend(title='Genres')#legend title
plt.legend(fontsize='small')  # You can use 'small', 'medium', 'large' or s

# Alternatively, you can control the size of the legend box:
plt.legend(handlelength=1, fontsize=8)  #
plt.show()#visualize the graph
```



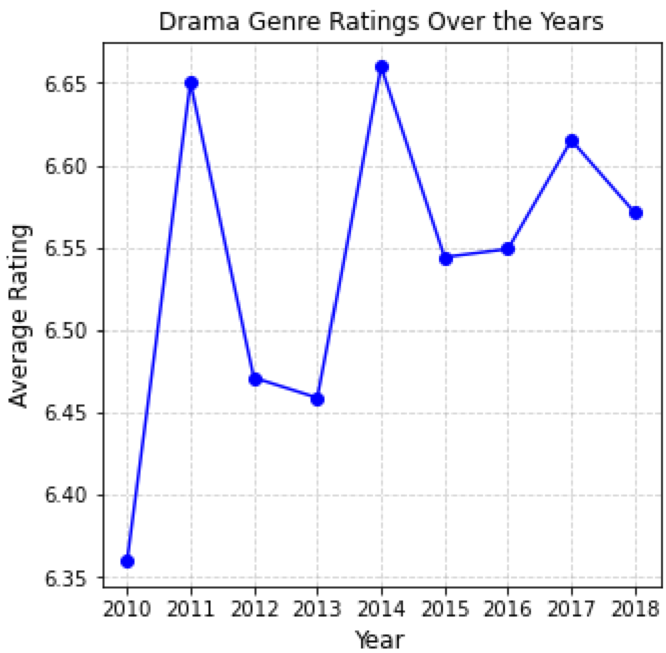

#### 4.0.4.1 Findings:

It is evident that the top selling movies are associated with the best rating and most popular genres such as the drama genre.

### 4.0.5 Analysis on best rated genre yearly

Since drama is the most popular genre we can create a visualization on how it has faired over the years

In [88]:
```
#how genre with best rating has faired over the years(univariate analysis)
# Filter for the genre of interest
genre_of_interest = "Drama"
drama_df = merged_movies_final_df[merged_movies_final_df['genres'].str.cont

# Perform univariate analysis: Focus on average ratings over time
drama_yearly = drama_df.groupby('year')['average_rating'].mean()

# Plotting the univariate trend
plt.figure(figsize=(5, 5))
drama_yearly.plot(kind='line', marker='o', color='blue')
plt.title("Drama Genre Ratings Over the Years")
plt.xlabel("Year", fontsize=12)
plt.ylabel("Average Rating", fontsize=12)
plt.grid(visible=True, linestyle='--', alpha=0.6)
plt.show()
```



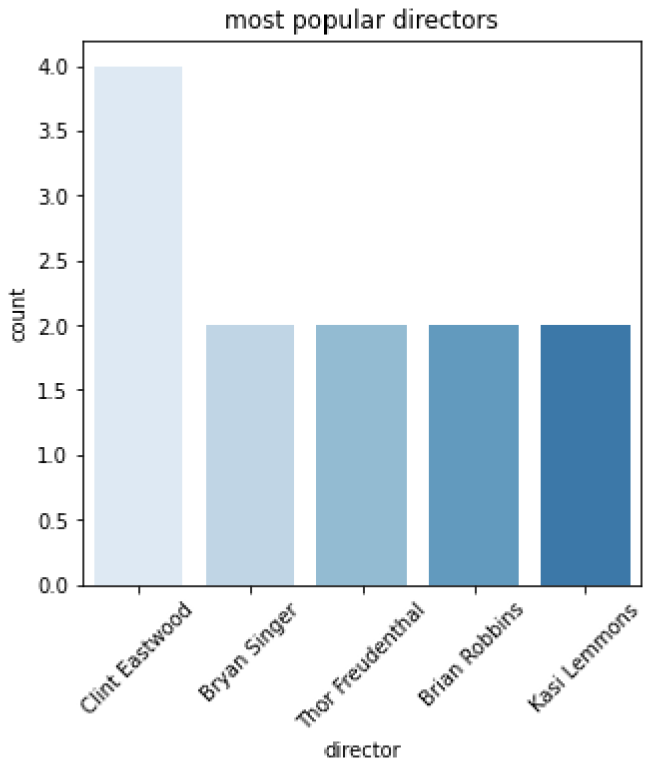

#### 4.0.5.1 Findings:

The graph depicts the trend of average ratings for the drama genre from 2010 to 2018. Here's an analysis: The ratings exhibit fluctuations over the years rather than a consistent trend. The highest average rating occurs around 2014, reaching approximately 6.70. The lowest rating is observed in 2010, near 6.45. There is notable variability, with significant increases from 2010 to 2011 and a sharp rise to the peak in 2014. Post-2014, the ratings dip and rise again, peaking slightly in 2016 and 2017 before a small drop in 2018. From 2015 to 2016, the ratings appear more stable compared to previous years. This analysis suggests a variability in drama ratings, with an overall upward movement from the start to the peak, followed by a decline and stabilization towards the end of the period. This may be brought about by poor direction or production for this particular genre.

### 4.0.6 Analysis based on top popular directors

We can further our analysis based on the top 5 directors

In [89]:
```
#popular directors
director_popularity = movie_df['director'].value_counts().head(5)
plt.figure(figsize=(5,5))
sns.barplot(x=director_popularity.index,y=director_popularity.values)
plt.title("most popular directors")
plt.xlabel("director")
plt.ylabel("count")
plt.xticks(rotation=45);
```




#### 4.0.6.1 Findings:

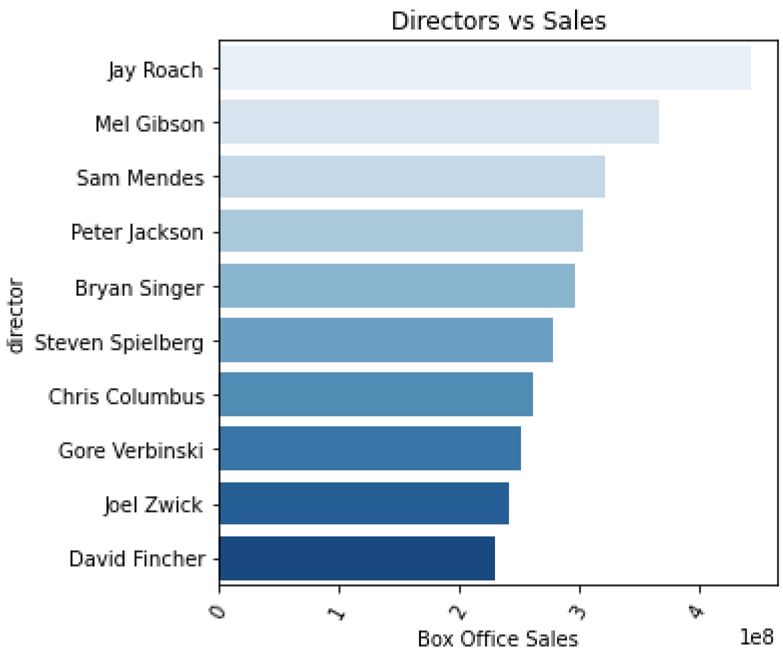
From the above analysis it is evident that Clint Eastwood is the highest selling director. This may be brought by factors such as perfect direction, availability and good work ethics.

### 4.0.7 Analysis based on best selling directors

We can then make a visualization to view the top selling directors based on their box office.

Note: Number of tickets that are sold for a movie, as a measure of how popular and financially successful the movie or director is.

In [90]:
```
#best selling directors

# Group by director and sum the box office sales
director_sales = movie_df.groupby("director")["box_office"].sum().sort_valu

# Create the plot
plt.figure(figsize=(5, 5))
sns.barplot(x=director_sales.values, y=director_sales.index, palette="Blues

# Add title and labels
plt.xlabel('Box Office Sales')
plt.title('Directors vs Sales')

# Include x-tick labels (rotate for readability)
plt.xticks(rotation=60)

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




#### 4.0.7.1 Findings:

It is evident that Jay Roach made the highest sales based on box-office. This may be brought about by factors such as quality direction, good work ethics and professionalism.

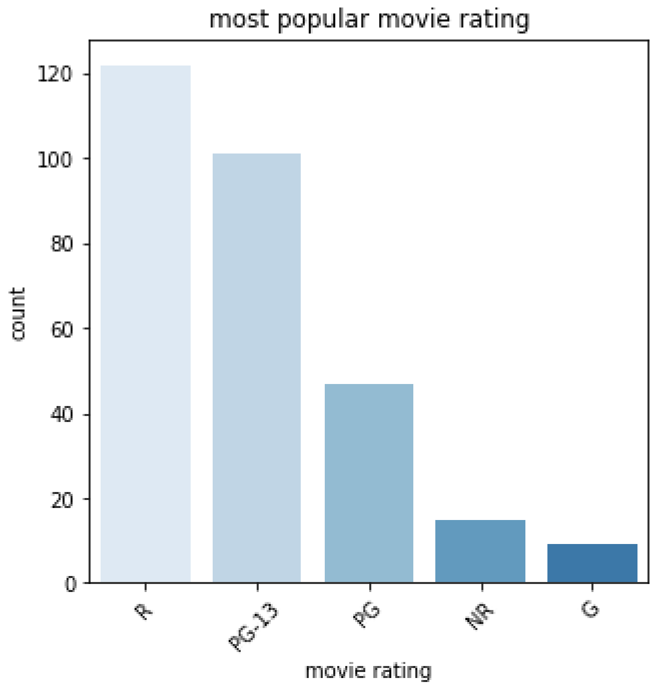
### 4.0.8 Analysis based on popular movie rating

We can further our analysis by movie rating popularity.

Note: In this visualization we are reviwing most popular assigned for each audience as follows:

1. R means restricted for audience under 18 years(adults)
2. PG-13 means restricted for audience under 13 years
3. PG means not restricted but requires parental guidance for audience less than 13 years
4. NR means not rated
5. G means for general audience meaning it is not restricted to any audience

In [91]:
```
#popular movie rating
rating_popularity = movie_df['rating'].value_counts().head(5)
plt.figure(figsize=(5,5))
sns.barplot(x=rating_popularity.index,y=rating_popularity.values)
plt.title("most popular movie rating")
plt.xlabel("movie rating")
plt.ylabel("count")
plt.xticks(rotation=45);
```



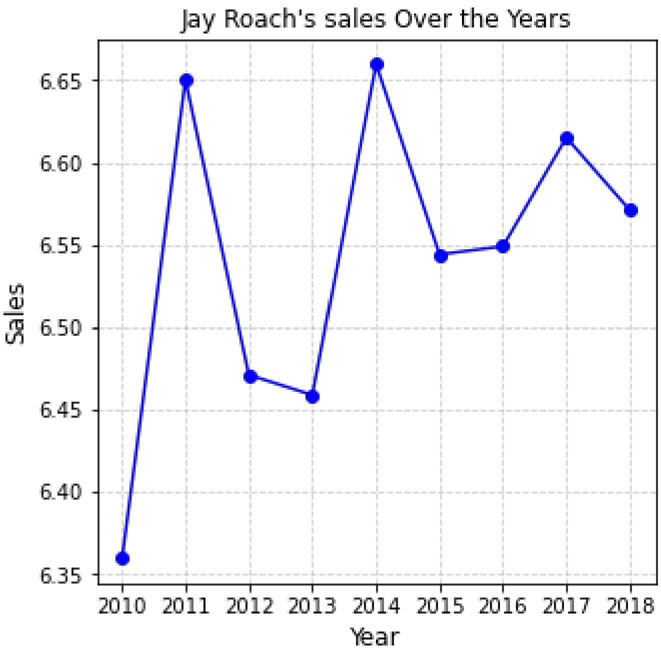

#### 4.0.8.1 Findings

It is evident that the movie rating with the highest popularity is the R rated. This may be due to audience popularity who are adults.

### 4.0.9 Analysis based on the best selling director yearly

Here we did analysis on the best selling director and he faired on yearly

In [92]:
```
#univariate analysis based on the highest selling director
# Filter for the genre of interest
highest_director = "Jay Roach"
director_df = movie_df[movie_df['director'].str.contains(highest_director, 

# Perform univariate analysis: Focus on average ratings over time
performance_yearly = director_df.groupby('year')['box_office'].mean()

# Plotting the univariate trend
plt.figure(figsize=(5, 5))
drama_yearly.plot(kind='line', marker='o', color='blue')
plt.title("Jay Roach's sales Over the Years")
plt.xlabel("Year", fontsize=12)
plt.ylabel("Sales", fontsize=12)
plt.grid(visible=True, linestyle='--', alpha=0.6)
plt.show()
```




#### 4.0.9.1 Findings:

The director sales have exhibit fluctuations over the years, with no consistent upward or downward trend. There are peaks and troughs at various points. The sales peaked significantly in 2014. There was another noticeable high point in 2017. The lowest sales occurred in 2010 and 2011. A decline in sales can also be observed in 2015 and 2018 after prior increases. Between 2012 and 2013 as well as 2015 to 2016, the sales remained relatively stable with minimal fluctuations. The variation in sales could be attributed to the performance of individual projects, changes in market dynamics, or external factors like competition or shifts in audience preferences.

## 4.1 Conclusions/ Results based on our analysis

From the analysis, it is clear that the Drama genre enjoys the highest ratings, but its box office success fluctuates over time. While the Adventure, Comedy, and Sci-Fi genres remain the highest-selling genres, this is likely due to their broad appeal and entertainment value. Directors like Clint Eastwood and Jay Roach contribute significantly to the sales, with their strong reputations and professional standards. The ratings analysis also highlights the R rating's popularity, reflecting the tastes of adult audiences. Overall, the variability in movie sales and ratings over the years is likely driven by factors such as market dynamics, audience preferences, and the quality of direction and production.

## 4.2 Recommendations

1. We recommend the studio produce movies related to the drama genre
2. The studio to consider making movies associated with the Adventure,Comedy,Sci-Fi genres as the highest selling genres
3. Also recommend the studio to work with the director Clint Eastwood based on his popularity
4. The studio can also consider working with the highest selling director Jay Roach
5. Finally we recommend that the studio make adult films as it is the most popular among the audiences

# 5 Hypothesis Testing

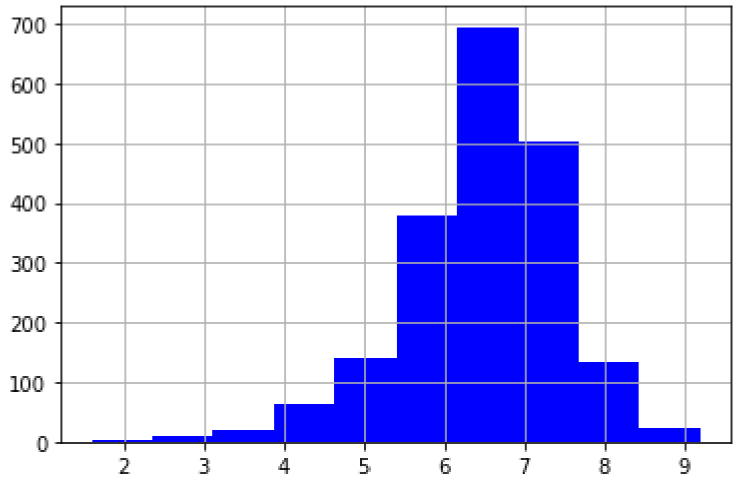
- **95% Confidence Interval (CI)**: The true value of the performance metric is expected to lie within this range with 95% confidence based on the data and model.
- **Alpha value (α)**: The significance level is **0.05**, corresponding to a 95% confidence level, indicating that the likelihood of observing a value outside this range is 5%.

In [93]:
```
merged_movies_final_df.describe()
```

Out[93]:

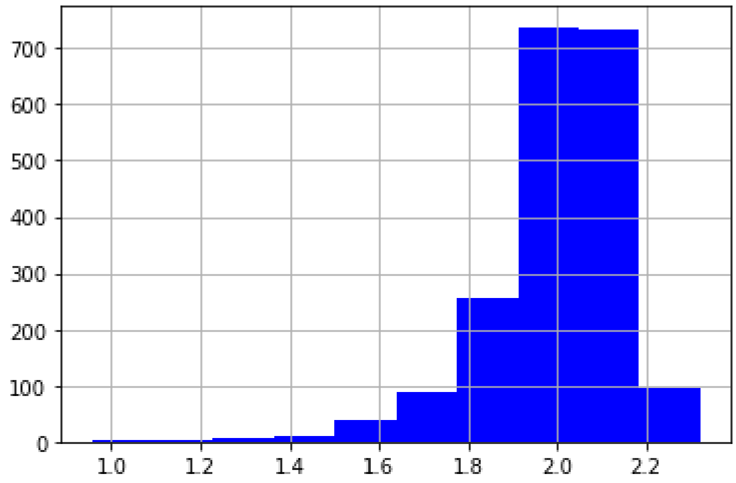
| | average_rating | domestic_gross | foreign_gross | year | total_gross |
|---|---|---|---|---|---|
| **count** | 1970.000000 | 1.970000e+03 | 1.970000e+03 | 1970.000000 | 1.970000e+03 |
| **mean** | 6.460812 | 2.005813e+06 | 1.474976e+07 | 2014.114721 | 1.675557e+07 |
| **std** | 0.999176 | 3.573622e+06 | 8.693627e+06 | 2.366826 | 9.497521e+06 |
| **min** | 1.600000 | 1.000000e+02 | 6.000000e+02 | 2010.000000 | 1.080000e+04 |
| **25%** | 5.900000 | 5.950000e+04 | 7.000000e+06 | 2012.000000 | 1.102500e+07 |
| **50%** | 6.600000 | 3.280000e+05 | 1.890000e+07 | 2014.000000 | 1.895730e+07 |
| **75%** | 7.100000 | 2.000000e+06 | 1.890000e+07 | 2016.000000 | 1.970000e+07 |
| **max** | 9.200000 | 1.710000e+07 | 5.410000e+07 | 2018.000000 | 7.100000e+07 |

In [94]:
```
merged_movies_final_df.average_rating.hist(color="blue");
```

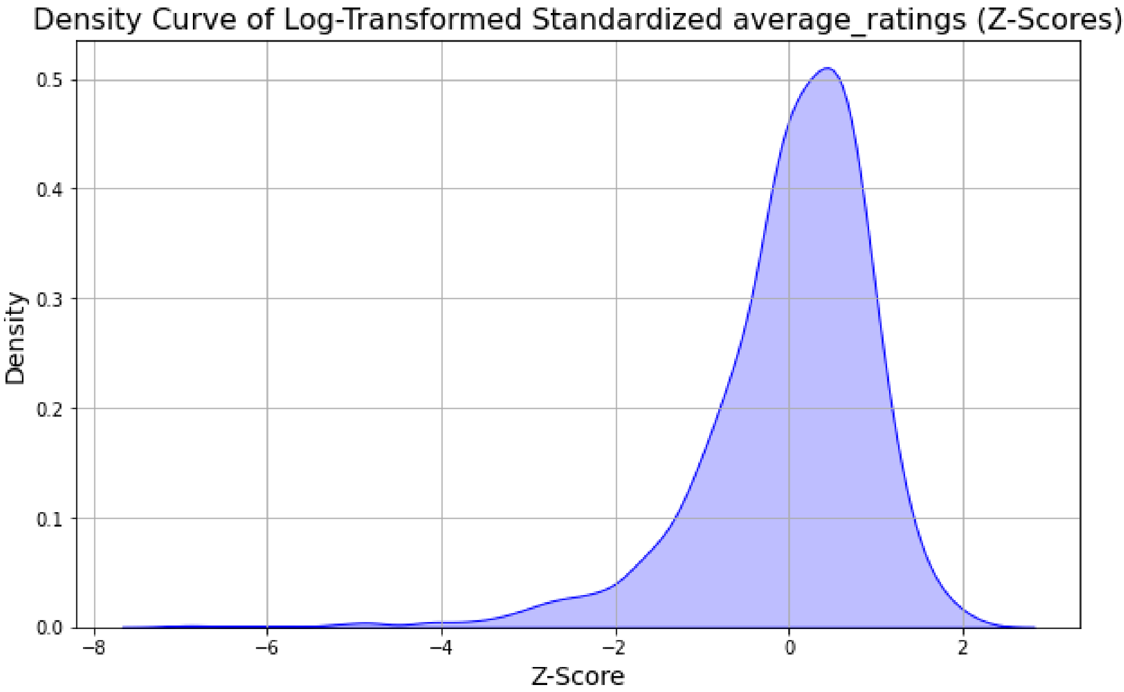



In [95]:
```
# apply log transformation to the 'average_rating' column to reduce skewness
merged_movies_final_df['log_average_rating'] = np.log1p(merged_movies_final_
merged_movies_final_df['log_average_rating'].hist(color="blue")
merged_movies_final_df['log_average_rating'].head()
```

Out[95]:
```
0    2.091864
1    1.960095
2    1.945910
3    1.902108
4    1.974081
Name: log_average_rating, dtype: float64
```



In [96]:
```
# standardize the log-transformed 'average_rating' column
log_average_rating_mean = merged_movies_final_df['log_average_rating'].mean
log_average_rating_std = merged_movies_final_df['log_average_rating'].std()
merged_movies_final_df['log_average_rating_zscore'] = (merged_movies_final_(
# Plot a curve for the log-transformed and standardized average_ratings
plt.figure(figsize=(10, 6))
sns.kdeplot(merged_movies_final_df['log_average_rating_zscore'], shade=True
plt.title('Density Curve of Log-Transformed Standardized average_ratings (Z
plt.xlabel('Z-Score', fontsize=14)
plt.ylabel('Density', fontsize=14)
plt.grid(True)
plt.show()
```




In [97]:
```
merged_movies_final_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1970 entries, 0 to 1969
Data columns (total 9 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   title                      1970 non-null   object
 1   genres                     1970 non-null   object
 2   average_rating             1970 non-null   float64
 3   domestic_gross             1970 non-null   float64
 4   foreign_gross              1970 non-null   float64
 5   year                       1970 non-null   int64
 6   total_gross                1970 non-null   float64
 7   log_average_rating         1970 non-null   float64
 8   log_average_rating_zscore  1970 non-null   float64
dtypes: float64(6), int64(1), object(2)
memory usage: 153.9+ KB
```

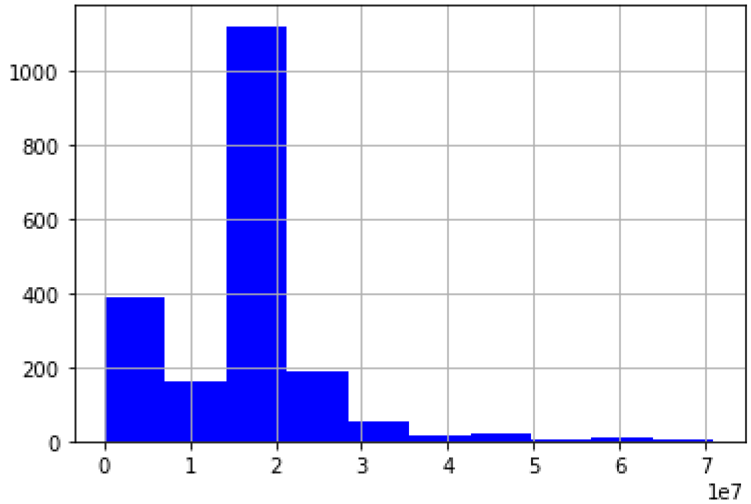
In [98]:
```
print("Null hypothesis\n log_average_rating_mean = ", log_average_rating_mea
print("Alternative hypothesis\n log_average_rating_mean > ", log_average_ra
```

```
Null hypothesis
 log_average_rating_mean =  1.9994422593545513
Alternative hypothesis
 log_average_rating_mean >  1.9994422593545513
```

In [99]:
```
alpha = 0.05  # significance level

# extract the column data
log_average_rating_data = merged_movies_final_df['log_average_rating']

# sample statistics
n = np.random.randint(100,len(log_average_rating_data))  # sample size
sample_mean = np.mean(log_average_rating_data)  # sample mean

# calculate the z-statistic
z_stat = (sample_mean - log_average_rating_mean) / (log_average_rating_std /

# perform one-tailed z-test (alternative hypothesis: mean > mu)
p_value = 1 - norm.cdf(z_stat)

# output the results
print(f"Sample Mean: {sample_mean}")
print(f"Z-Statistic: {z_stat}")
print(f"P-Value: {p_value}")

# make a decision based on the p-value
if p_value < alpha:
    print("Reject the null hypothesis: The mean is significantly greater tha
else:
    print("Fail to reject the null hypothesis: There is no significant evide
```

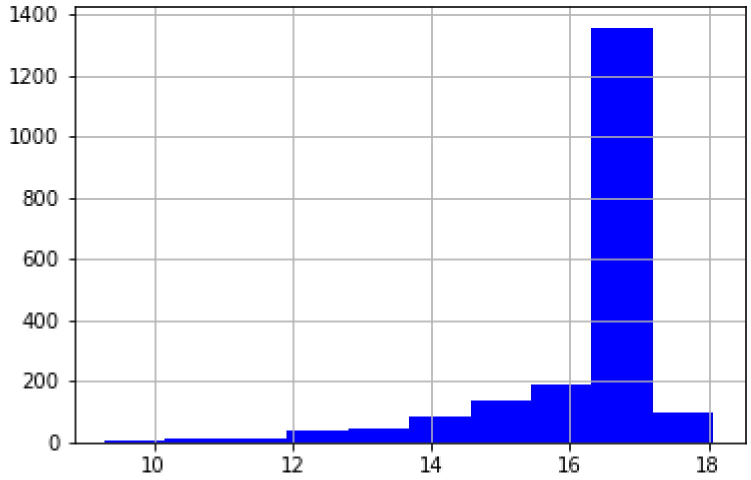
```
Sample Mean: 1.9994422593545513
Z-Statistic: 0.0
P-Value: 0.5
Fail to reject the null hypothesis: There is no significant evidence
 that the mean is greater than  1.9994422593545513 at 95% confidence inter
val
```

In [100]:
```
merged_movies_final_df.total_gross.hist(color="blue");
```



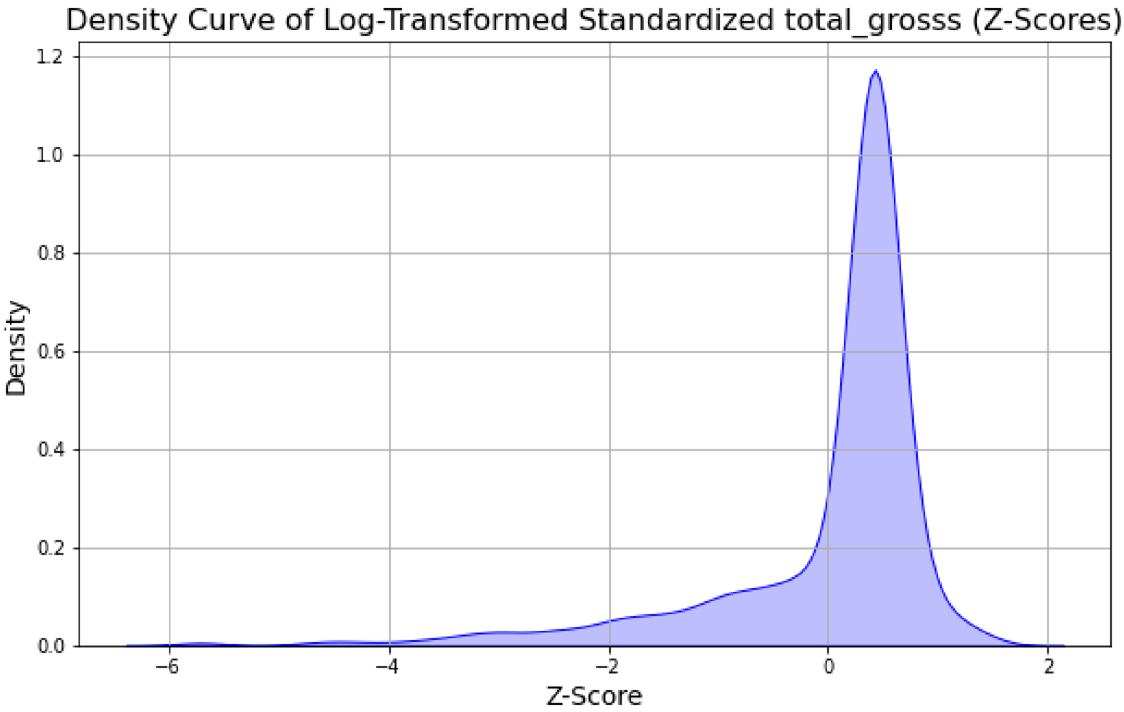

In [101]:
```
# apply log transformation to the 'total_gross' column to reduce skewness
merged_movies_final_df['log_total_gross'] = np.log1p(merged_movies_final_df
merged_movies_final_df['log_total_gross'].hist(color="blue")
merged_movies_final_df['log_total_gross'].head()
```

Out[101]:
```
0    16.811243
1    15.983878
2    15.983878
3    15.983878
4    16.989389
Name: log_total_gross, dtype: float64
```



In [102]:

```
# standardize the log-transformed 'total_gross' column
log_total_gross_mean = merged_movies_final_df['log_total_gross'].mean()
log_total_gross_std = merged_movies_final_df['log_total_gross'].std()
merged_movies_final_df['log_total_gross_zscore'] = (merged_movies_final_df[
# Plot a curve for the log-transformed and standardized total_grosss
plt.figure(figsize=(10, 6))
sns.kdeplot(merged_movies_final_df['log_total_gross_zscore'], shade=True, c
plt.title('Density Curve of Log-Transformed Standardized total_grosss (Z-Sc
plt.xlabel('Z-Score', fontsize=14)
plt.ylabel('Density', fontsize=14)
plt.grid(True)
plt.show()
```




In [103]:

```
print("Null hypothesis\n log_total_gross_mean = ", log_total_gross_mean)
print("Alternative hypothesis\n log_total_gross_mean > ", log_total_gross_m
```

```
Null hypothesis
 log_total_gross_mean =  16.269271504462154
Alternative hypothesis
 log_total_gross_mean >  16.269271504462154
```

In [104]:
```
# extract the column data
log_total_gross_data = merged_movies_final_df['log_total_gross']

# sample statistics
n = np.random.randint(100,len(log_total_gross_data))  # sample size
sample_mean = np.mean(log_total_gross_data)  # sample mean

# calculate the z-statistic
z_stat = (sample_mean - log_total_gross_mean) / (log_total_gross_std / np.s

# perform one-tailed z-test (alternative hypothesis: mean > mu)
p_value = 1 - norm.cdf(z_stat)

# output the results
print(f"Sample Mean: {sample_mean}")
print(f"Z-Statistic: {z_stat}")
print(f"P-Value: {p_value}")

# make a decision based on the p-value
if p_value < alpha:
    print("Reject the null hypothesis: The mean is significantly greater tha
else:
    print("Fail to reject the null hypothesis: There is no significant evide
```

```
Sample Mean: 16.269271504462154
Z-Statistic: 0.0
P-Value: 0.5
Fail to reject the null hypothesis: There is no significant evidence
 that the mean is greater than 16.269271504462154 at 95% confidence interv
al
```

# 6 Modeling

In [105]:
```
merged_movies_final_df.head()
```

Out[105]:

| | title | genres | average_rating | domestic_gross | foreign_gross | ye |
|---|---|---|---|---|---|---|
| 0 | Wazir | Action,Crime,Drama | 7.1 | 1100000.0 | 18900000.0 | |
| 1 | On the Road | Adventure,Drama,Romance | 6.1 | 744000.0 | 8000000.0 | |
| 2 | On the Road | Drama | 6.0 | 744000.0 | 8000000.0 | |
| 3 | On the Road | Drama | 5.7 | 744000.0 | 8000000.0 | |
| 4 | The Rum Diary | Comedy,Drama | 6.2 | 13100000.0 | 10800000.0 | |

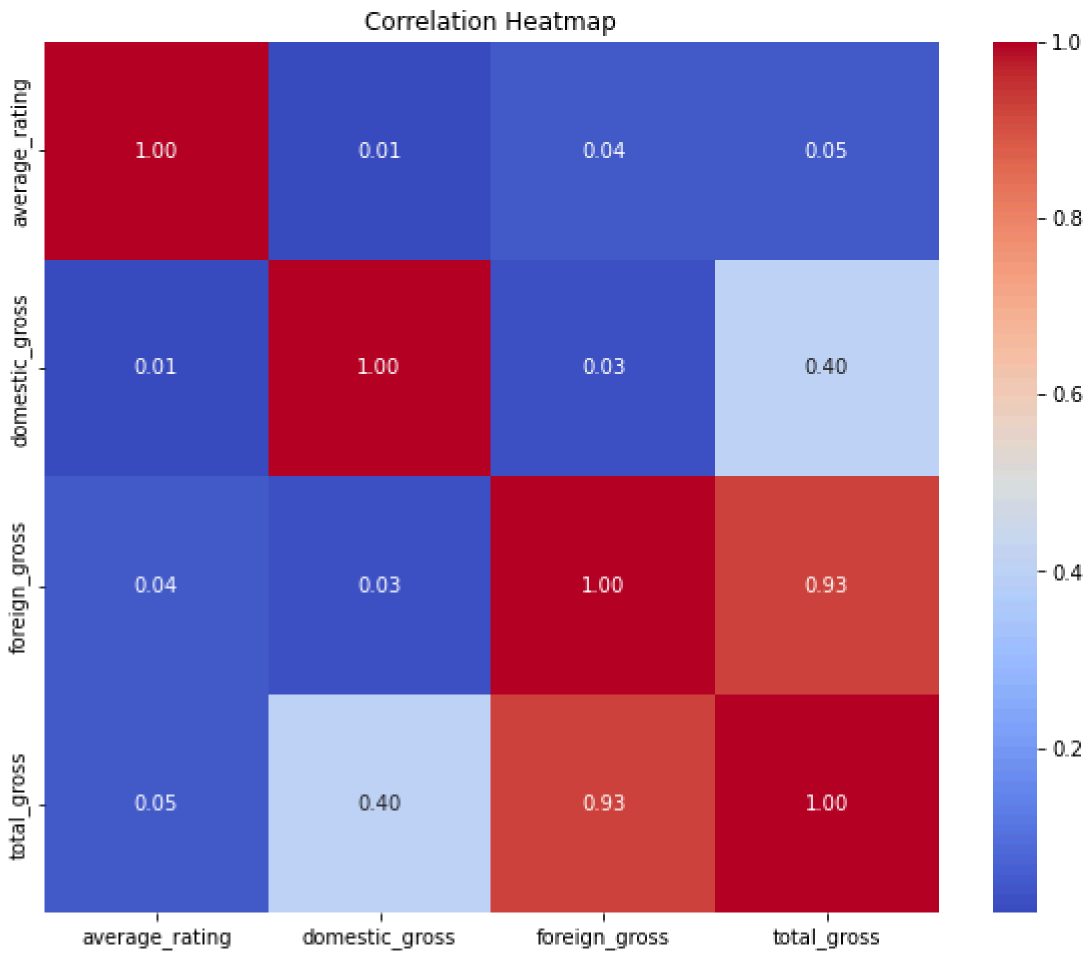
In [106]:
```
#getting data for modelling
#merged_movies_modelling_df= merged_movies_final_df[["average_rating","domes
merged_movies_modelling_df= merged_movies_final_df[["average_rating","domes
```

In [107]:
```
#feature Selection
selected_features = ["average_rating","domestic_gross","foreign_gross",'tota
X = merged_movies_modelling_df[selected_features]
y = merged_movies_final_df['total_gross']
```

In [108]:
```
# Correlation Analysis
correlation_matrix = merged_movies_final_df[selected_features].corr()
print("Correlation Matrix:\n", correlation_matrix)
```

```
Correlation Matrix:
                 average_rating  domestic_gross  foreign_gross  total_gros
s
average_rating        1.000000        0.013149       0.044386    0.045577
domestic_gross        0.013149        1.000000       0.029821    0.403566
foreign_gross         0.044386        0.029821       1.000000    0.926578
total_gross           0.045577        0.403566       0.926578    1.000000
```

In [109]:
```
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap')
plt.show()
```



In [110]:
```
# Step 3: Modeling
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra
```

In [111]:
```
model = sm.OLS(endog=y, exog=X)
result =model.fit()
result.summary()
```

Out[111]:

OLS Regression Results

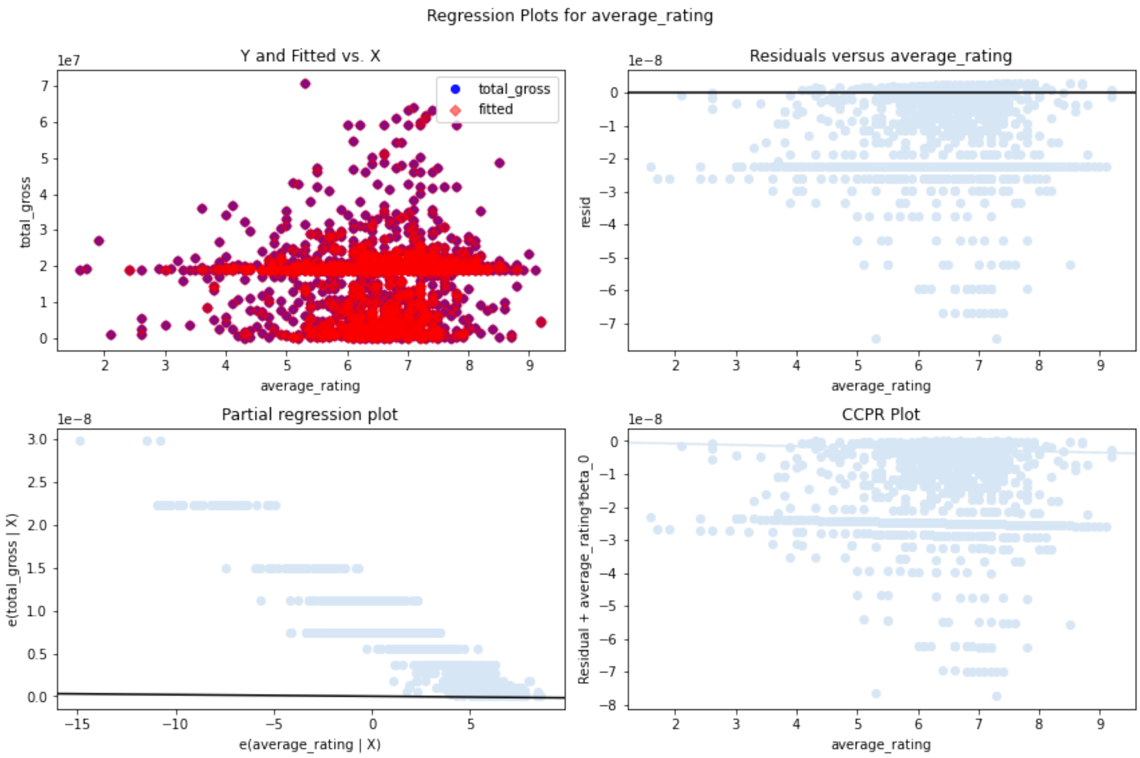
| | | | |
|---|---|---|---|
| **Dep. Variable:** | total_gross | **R-squared (uncentered):** | 1.000 |
| **Model:** | OLS | **Adj. R-squared (uncentered):** | 1.000 |
| **Method:** | Least Squares | **F-statistic:** | 5.507e+32 |
| **Date:** | Sat, 18 Jan 2025 | **Prob (F-statistic):** | 0.00 |
| **Time:** | 20:42:23 | **Log-Likelihood:** | 32030. |
| **No. Observations:** | 1970 | **AIC:** | -6.405e+04 |
| **Df Residuals:** | 1967 | **BIC:** | -6.404e+04 |
| **Df Model:** | 3 | | |
| **Covariance Type:** | nonrobust | | |

| | coef | std err | t | P>\|t\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **average_rating** | -3.893e-10 | 1.44e-10 | -2.703 | 0.007 | -6.72e-10 | -1.07e-10 |
| **domestic_gross** | 0.3333 | 9.08e-17 | 3.67e+15 | 0.000 | 0.333 | 0.333 |
| **foreign_gross** | 0.3333 | 5.79e-17 | 5.76e+15 | 0.000 | 0.333 | 0.333 |
| **total_gross** | 0.6667 | 4.69e-17 | 1.42e+16 | 0.000 | 0.667 | 0.667 |

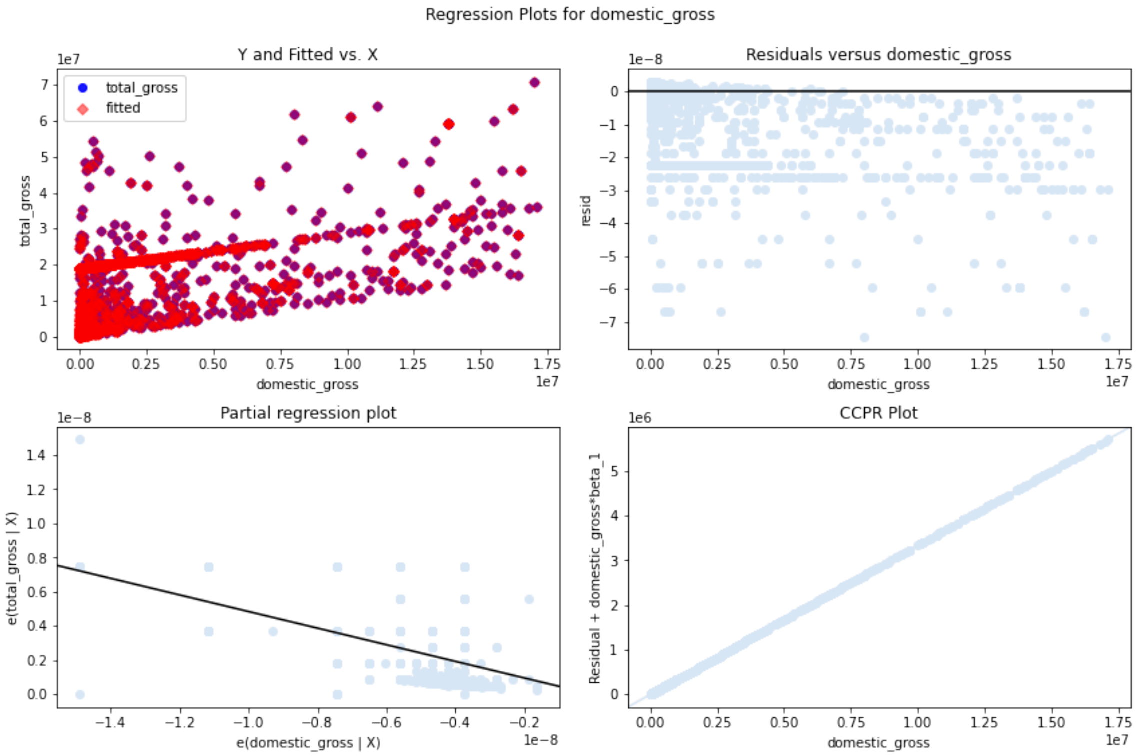
| | | | |
|---|---|---|---|
| **Omnibus:** | 181.928 | **Durbin-Watson:** | 0.453 |
| **Prob(Omnibus):** | 0.000 | **Jarque-Bera (JB):** | 603.703 |
| **Skew:** | -0.440 | **Prob(JB):** | 8.08e-132 |
| **Kurtosis:** | 5.565 | **Cond. No.** | 1.79e+16 |

Notes:
[1] $R^2$ is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3] The smallest eigenvalue is 4.09e-15. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

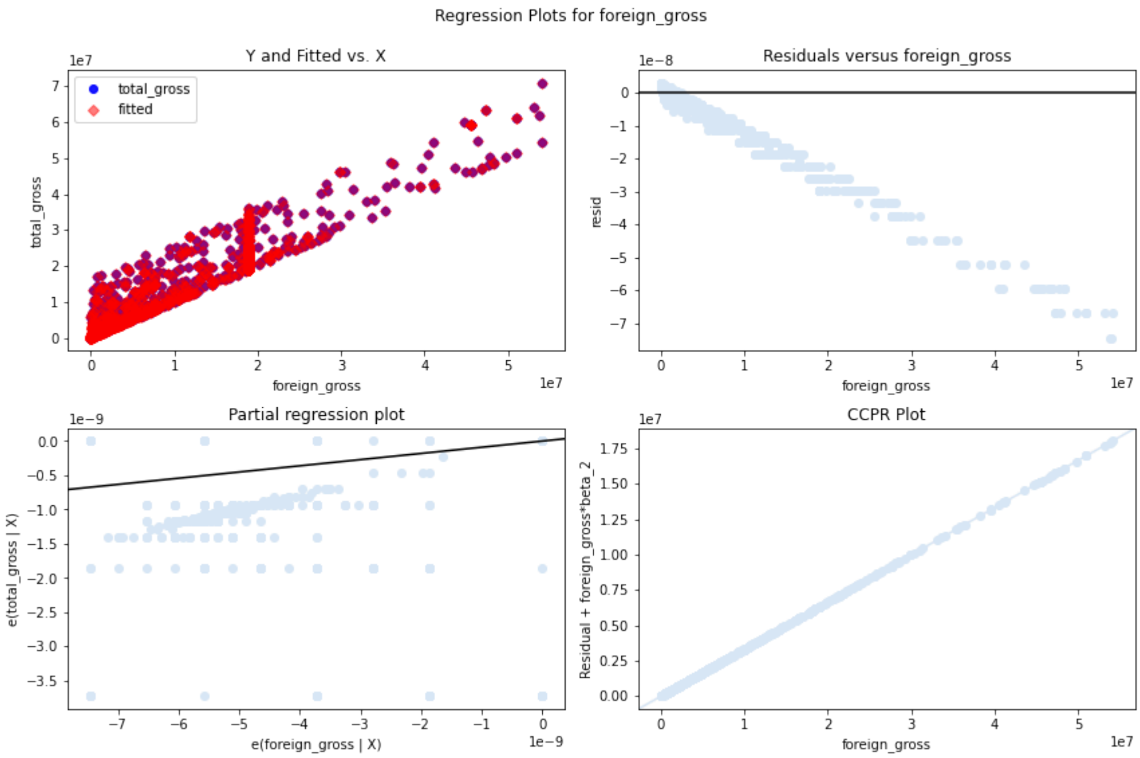
In [112]:
```
#checking resideual for average_rating variable
sm.graphics.plot_regress_exog(result, "average_rating", fig=plt.figure(figs
```




In [113]:
```
#checking resideual for domestic_gross variable
sm.graphics.plot_regress_exog(result, "domestic_gross", fig=plt.figure(figs
```



In [114]:
```
#checking resideual for foreign_gross variable
sm.graphics.plot_regress_exog(result, "foreign_gross", fig=plt.figure(figsi
```




In [115]:
```
merged_movies_final_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1970 entries, 0 to 1969
Data columns (total 11 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   title                       1970 non-null   object 
 1   genres                      1970 non-null   object 
 2   average_rating              1970 non-null   float64
 3   domestic_gross              1970 non-null   float64
 4   foreign_gross               1970 non-null   float64
 5   year                        1970 non-null   int64  
 6   total_gross                 1970 non-null   float64
 7   log_average_rating          1970 non-null   float64
 8   log_average_rating_zscore   1970 non-null   float64
 9   log_total_gross             1970 non-null   float64
 10  log_total_gross_zscore      1970 non-null   float64
dtypes: float64(8), int64(1), object(2)
memory usage: 184.7+ KB
```

## 6.1 linear_regression

### 6.1.1 Feature Preprocessing

In [116]:
```
# Label encode the 'genres' column (target variable)
label_encoder = LabelEncoder()
merged_movies_modelling_df['genres_encoded'] = label_encoder.fit_transform(
```

In [117]:
```
# Features (X) and Target (y)
X = merged_movies_modelling_df[['total_gross', 'average_rating']]  # Includ
y = merged_movies_modelling_df['genres_encoded']

# Split data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra

# Step 2: Train the Logistic Regression Model (since we are predicting a ca
model = LogisticRegression(max_iter=100)
model.fit(X_train, y_train)
```

Out[117]:
```
LogisticRegression()
```

# 7 Evaluation

In [118]:
```
# Calculate evaluation metrics
y_pred = model.predict(X_test)  # Make predictions using the trained model

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print(f"MAE: {mae}")
print(f"MSE: {mse}")
print(f"RMSE: {rmse}")
print(f"R²: {r2}")
```

```
MAE: 66.00507614213198
MSE: 8115.197969543147
RMSE: 90.08439359591176
R²: -0.38027741022485917
```

**Observation** The model's performance is suboptimal, as reflected by the following key observations: High Errors: The MAE (65.66), MSE (8101.98), and RMSE (90.01) indicate significant prediction errors on average. Negative $R^2$ (-0.378): The negative $R^2$ suggests the model performs worse than simply predicting the mean of the target variable. Model Improvements Needed: The model likely requires better feature engineering, data preprocessing, and possibly a different model approach to improve accuracy and fit.

In [119]:
```
# Step 3: Modify the Random Prediction Function to Predict Genre

def predict_random_genre(model, merged_movies_modelling_df):
    # Randomly select a row from the dataset
    random_row = merged_movies_modelling_df.sample(1)

    # Extract the values for total_gross and average_rating
    total_gross = random_row['total_gross'].values[0]
    average_rating = random_row['average_rating'].values[0]

    # Make a prediction using the trained model
    genre_encoded_prediction = model.predict([[total_gross, average_rating]

    # Convert the encoded prediction back to the original genre
    predicted_genre = label_encoder.inverse_transform([genre_encoded_predic

    # Return the random row and the predicted genre
    return random_row[['genres', 'total_gross', 'average_rating']].values, 

# Example usage:
random_movie, predicted_genre = predict_random_genre(model, merged_movies_m
print("Random Movie Data: ", random_movie)
print("Predicted Genre: ", predicted_genre)
```

```
Random Movie Data:  [['Comedy,Drama' 19031000.0 6.4]]
Predicted Genre:  Drama
```