Project: Investigate a Dataset - TMDB 5000 Movie Dataset

Table of Contents

Introduction

Dataset Description

The **TMDB 5000 Movie Dataset** offers a comprehensive overview of approximately 10,000 movies sourced from The Movie Database (TMDB). This dataset encompasses various movie details, including user ratings, revenue, genres, cast, and crew information. Key aspects of the dataset include multiple values in certain columns, special characters, and adjusted financial figures. Columns like `cast` and `genres` contain multiple values separated by a pipe (`|`) character, allowing for the inclusion of multiple actors or genres. Certain columns, such as `cast` , also include special characters that do not require additional cleaning for this analysis. Additionally, the `budget_adj` and `revenue_adj` columns present budget and revenue figures adjusted to 2010 dollars, providing a standardized financial comparison across different time periods by accounting for inflation.

Key Columns

- **id**: Unique movie identifier.
- **imdb_id**: IMDB code specific to the movie.
- **popularity**: Metric indicating the movie's popularity.
- **budget**: Amount spent to produce the movie.
- **revenue**: Earnings from the movie.
- **original_title**: The movie's original name.
- **cast**: Leading actors involved in the movie.
- **homepage**: Official website of the movie.
- **director**: The movie's director.
- **tagline**: A catchy phrase representing the movie.
- **keywords**: Terms associated with the movie.
- **overview**: A brief summary of the movie.
- **runtime**: Total duration of the movie in minutes.

- **genres**: Categories describing the movie's style and content.
- **production_companies**: Firms that produced the movie.
- **release_date**: When the movie was first released.
- **vote_count**: Total votes received by the movie.
- **vote_average**: Average rating given to the movie.
- **release_year**: Year the movie was released.
- **budget_adj**: Movie's budget adjusted for inflation.
- **revenue_adj**: Movie's revenue adjusted for inflation.

Questions for Analysis

Q1. Which movies had the highest budgets?

Q2. Which movies received the most votes from viewers?

Q3. What is the average rating for movies in each genre?

Q4. Is the budget related to a higher average vote?

Q5. What are the most common release years in the dataset?

Q6. Is there a relationship between movie popularity and revenue?

Q7. How has the average budget for movies changed over the years?

Q8. Which movies have the highest revenue, adjusted for inflation?

Q9. What is the average runtime for movies in each genre?

Q10. Which movies had the highest popularity scores each year?

Now, we need to **import the necessary libraries** to uesd it later on analysis data.

In [118…

```
# Import necessary libraries
import numpy as np              # For numerical operations
import pandas as pd             # For data manipulation and analysis
import matplotlib.pyplot as plt # For plotting graphs and charts
import seaborn as sns           # For advanced data visualization
import json                     # For handling JSON data

# Display plots directly in the notebook (only for Jupyter Notebook)
%matplotlib inline

# Print confirmation message if libraries are imported successfully
print("Libraries imported successfully.")
```

```
Libraries imported successfully.
```

Data Wrangling

In this section, we'll load the TMDB 5000 Movie Dataset and check its quality to ensure it's ready for analysis. We'll examine the data for any issues, like missing values or inconsistencies, and clean or trim where needed. Each step of the cleaning process will be documented, explaining why each decision was made to improve the data's quality and reliability.

General Properties

First, we'll load the dataset and take a quick look at its structure, data types, and any missing or unusual values. We'll break down each step clearly, so it's easy to follow along. Once we're done exploring, we'll organize our findings and refine our approach to make the data wrangling process straightforward and clear in the final report.

In [119…
```python
# Import pandas for data manipulation
import pandas as pd

# Load the TMDB movies dataset
try:
    df = pd.read_csv('tmdb-movies.csv')
    print("Success: Dataset loaded correctly!")
except FileNotFoundError:
    print("Error: File not found. Please check the file path.")
except Exception as e:
    print(f"Error: {e}")
```

```
Success: Dataset loaded correctly!
```

Now, I will **display the first few rows of the dataframe** to understand its structure

In [120…
```python
#display the first few rows of the dataframe
df.head()
```

Out[120…

	id	imdb_id	popularity	budget	revenue	original_title	cast	
0	135397	tt0369610	32.985763	150000000	1513528810	Jurassic World	Chris Pratt|Bryce Dallas Howard|Irrfan Khan|Vi...	
1	76341	tt1392190	28.419936	150000000	378436354	Mad Max: Fury Road	Tom Hardy|Charlize Theron|Hugh Keays-Byrne|Nic...	
2	262500	tt2908446	13.112507	110000000	295238201	Insurgent	Shailene Woodley|Theo James|Kate Winslet|Ansel...	htt
3	140607	tt2488496	11.173104	200000000	2068178225	Star Wars: The Force Awakens	Harrison Ford|Mark Hamill|Carrie Fisher|Adam D...	
4	168259	tt2820852	9.335014	190000000	1506249360	Furious 7	Vin Diesel|Paul Walker|Jason Statham|Michelle ...	

5 rows × 21 columns

Here, I will **display a summary of the DataFrame** to understand its structure, data types, and any missing values.

In [121…

```
# Display summary information about the DataFrame
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10866 entries, 0 to 10865
Data columns (total 21 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   id                    10866 non-null  int64
 1   imdb_id               10856 non-null  object
 2   popularity            10866 non-null  float64
 3   budget                10866 non-null  int64
 4   revenue               10866 non-null  int64
 5   original_title        10866 non-null  object
 6   cast                  10790 non-null  object
 7   homepage              2936 non-null   object
 8   director              10822 non-null  object
 9   tagline               8042 non-null   object
 10  keywords              9373 non-null   object
 11  overview              10862 non-null  object
 12  runtime               10866 non-null  int64
 13  genres                10843 non-null  object
 14  production_companies  9836 non-null   object
 15  release_date          10866 non-null  object
 16  vote_count            10866 non-null  int64
 17  vote_average          10866 non-null  float64
 18  release_year          10866 non-null  int64
 19  budget_adj            10866 non-null  float64
 20  revenue_adj           10866 non-null  float64
dtypes: float64(4), int64(6), object(11)
memory usage: 1.7+ MB
```

we noticed that the DataFrame has 10,866 rows and 21 columns, with some missing values in columns like `imdb_id`, `cast`, `homepage`, and `tagline`. Most data types are appropriate, and memory usage is around 1.7 MB.

Now, I will **generate descriptive summary statistics** for the dataset.

In [122…
```
# Generate descriptive statistics summary
df.describe()
```

Out[122…

	id	popularity	budget	revenue	runtime	vote_count
count	10866.000000	10866.000000	1.086600e+04	1.086600e+04	10866.000000	10866.000000
mean	66064.177434	0.646441	1.462570e+07	3.982332e+07	102.070863	217.389748
std	92130.136561	1.000185	3.091321e+07	1.170035e+08	31.381405	575.619058
min	5.000000	0.000065	0.000000e+00	0.000000e+00	0.000000	10.000000
25%	10596.250000	0.207583	0.000000e+00	0.000000e+00	90.000000	17.000000
50%	20669.000000	0.383856	0.000000e+00	0.000000e+00	99.000000	38.000000
75%	75610.000000	0.713817	1.500000e+07	2.400000e+07	111.000000	145.750000
max	417859.000000	32.985763	4.250000e+08	2.781506e+09	900.000000	9767.000000

Data Cleaning

In this section, we'll clean up the TMDB 5000 Movie Dataset to get it ready for analysis. This will involve these:

1. Removing duplicate rows.
2. Handling missing values in columns.
3. Adjusting data types where necessary.

Each step will be explained, so the reasons behind our cleaning choices are clear.

At the first, i'm **droped the columes** that not important and not needed on the dataset

In [123…
```
# here, I will drop columns using the list of column names
df.drop(['homepage','overview','keywords', 'imdb_id', 'production_companies','cast',
df.columns
```

Out[123…
```
Index(['id', 'popularity', 'budget', 'revenue', 'original_title', 'runtime',
       'genres', 'release_date', 'vote_count', 'vote_average', 'release_year',
       'budget_adj', 'revenue_adj'],
      dtype='object')
```

In [124…
```
# Print the first 5 rows after dropping colunms operation
df.head()
```

Out[124…

	id	popularity	budget	revenue	original_title	runtime	genre
0	135397	32.985763	150000000	1513528810	Jurassic World	124	Action|Adventure|Scienc Fiction|Thrille
1	76341	28.419936	150000000	378436354	Mad Max: Fury Road	120	Action|Adventure|Scienc Fiction|Thrille
2	262500	13.112507	110000000	295238201	Insurgent	119	Adventure|Scienc Fiction|Thrille
3	140607	11.173104	200000000	2068178225	Star Wars: The Force Awakens	136	Action|Adventure|Scienc Fiction|Fantas
4	168259	9.335014	190000000	1506249360	Furious 7	137	Action|Crime|Thrille

Then, this is a Dataset after dropping unwanted columns.

In [125…
```
# Display summary information about the DataFrame
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10866 entries, 0 to 10865
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   id              10866 non-null  int64  
 1   popularity      10866 non-null  float64
 2   budget          10866 non-null  int64  
 3   revenue         10866 non-null  int64  
 4   original_title  10866 non-null  object 
 5   runtime         10866 non-null  int64  
 6   genres          10843 non-null  object 
 7   release_date    10866 non-null  object 
 8   vote_count      10866 non-null  int64  
 9   vote_average    10866 non-null  float64
 10  release_year    10866 non-null  int64  
 11  budget_adj      10866 non-null  float64
 12  revenue_adj     10866 non-null  float64
dtypes: float64(4), int64(6), object(3)
memory usage: 1.1+ MB
```

The DataFrame now has 10,866 rows and 13 columns. Most columns are complete, except for a few missing values in `genres`. Numeric data is in place for budget, revenue, and ratings, while titles and genres are stored as text. Memory usage is now about 1.1 MB.

After that, i will remove the rows that be duplicate

In [126…

```python
# Check for duplicate rows in the dataset
duplicates = df.duplicated().sum()
print(f"Number of duplicate rows: {duplicates}")

# If duplicates are found, remove them; otherwise, print a message
if duplicates > 0:
    df = df.drop_duplicates()
    print("Duplicate rows removed.")
else:
    print("No duplicate rows found.")
```

```
Number of duplicate rows: 1
Duplicate rows removed.
```

We found one duplicate row in the dataset, which could impact our analysis, so we removed it to keep the data accurate and reliable.

Now, We started by **checking for missing values** in each column to see where data was incomplete.

In [127…

```python
# Check for missing values
missing_values = df.isnull().sum()
print("Missing values per column:\n", missing_values)
```

```
Missing values per column:
 id                 0
popularity         0
budget             0
revenue            0
original_title     0
runtime            0
genres            23
release_date       0
vote_count         0
vote_average       0
release_year       0
budget_adj         0
revenue_adj        0
dtype: int64
```

As we noticed above, We found missing values in a `genres` columne. So, after that, we reviewed each column with missing data to decide whether to fill in the missing values or remove those rows.

For `genres`, we filled missing values with 'Unknown' because it's important for analysis as shown below

In [128…
```
# Fill missing values in `genres` with 'Unknown'
df['genres'].fillna('Unknown', inplace=True)
```

Finally, we check the dataset again to confirm there were no remaining missing values as shown below

In [129…
```
# Final check for missing values
print("Remaining missing values per column:\n", df.isnull().sum())
```

```
Remaining missing values per column:
 id                 0
popularity         0
budget             0
revenue            0
original_title     0
runtime            0
genres             0
release_date       0
vote_count         0
vote_average       0
release_year       0
budget_adj         0
revenue_adj        0
dtype: int64
```

The dataset is now complete, with missing values handled, and ready for analysis.

Exploratory Data Analysis

With our data cleaned and ready, we can start exploring the dataset to answer our research questions. In this section, we'll calculate statistics and create visuals to help us understand the data better.

Our goal is to gain insights by looking at both individual variables and the relationships between them. We'll use different types of plots to reveal trends, patterns, and connections within the data. Pandas provides built-in visualization tools that we'll use to make this analysis clear and effective.

Research Question 1: Which movies had the highest budgets?

To find the movies with the highest budgets, we'll sort the dataset by the `budget` column in descending order. This will allow us to identify and list the top-budget movies in our dataset.

Sort the Movies by Budget

In [130…

```
# Sort movies by budget in descending order and select the top 10
top_budget_movies = df.sort_values(by='budget', ascending=False).head(10)

# Display the movie titles and their budgets for the top 10 highest-budget movies
top_budget_movies[['original_title', 'budget']]
```

Out[130…

	original_title	budget
2244	The Warrior's Way	425000000
3375	Pirates of the Caribbean: On Stranger Tides	380000000
7387	Pirates of the Caribbean: At World's End	300000000
14	Avengers: Age of Ultron	280000000
6570	Superman Returns	270000000
4411	John Carter	260000000
1929	Tangled	260000000
7394	Spider-Man 3	258000000
5508	The Lone Ranger	255000000
4367	The Hobbit: An Unexpected Journey	250000000

The movies with the biggest budgets include **The Warrior's Way** is 425 dollar million , **Pirates of the Caribbean: On Stranger Tides** 380 dollar million, and **Pirates of the Caribbean: At World's End** 300 dollar million. High-budget films tend to be major franchises or large productions with extensive special effects.

The Visualization (Plot)

At the start, I created a function called `plot_top10` to simplify the plotting of top 10 bar charts for various analyses. This function takes in key parameters such as the dataset, x and y variables, plot title, axis labels, bar color, and orientation (horizontal or vertical). By using this function, I avoided repetitive code and made it easier to customize each plot.

The function is defined as follows:

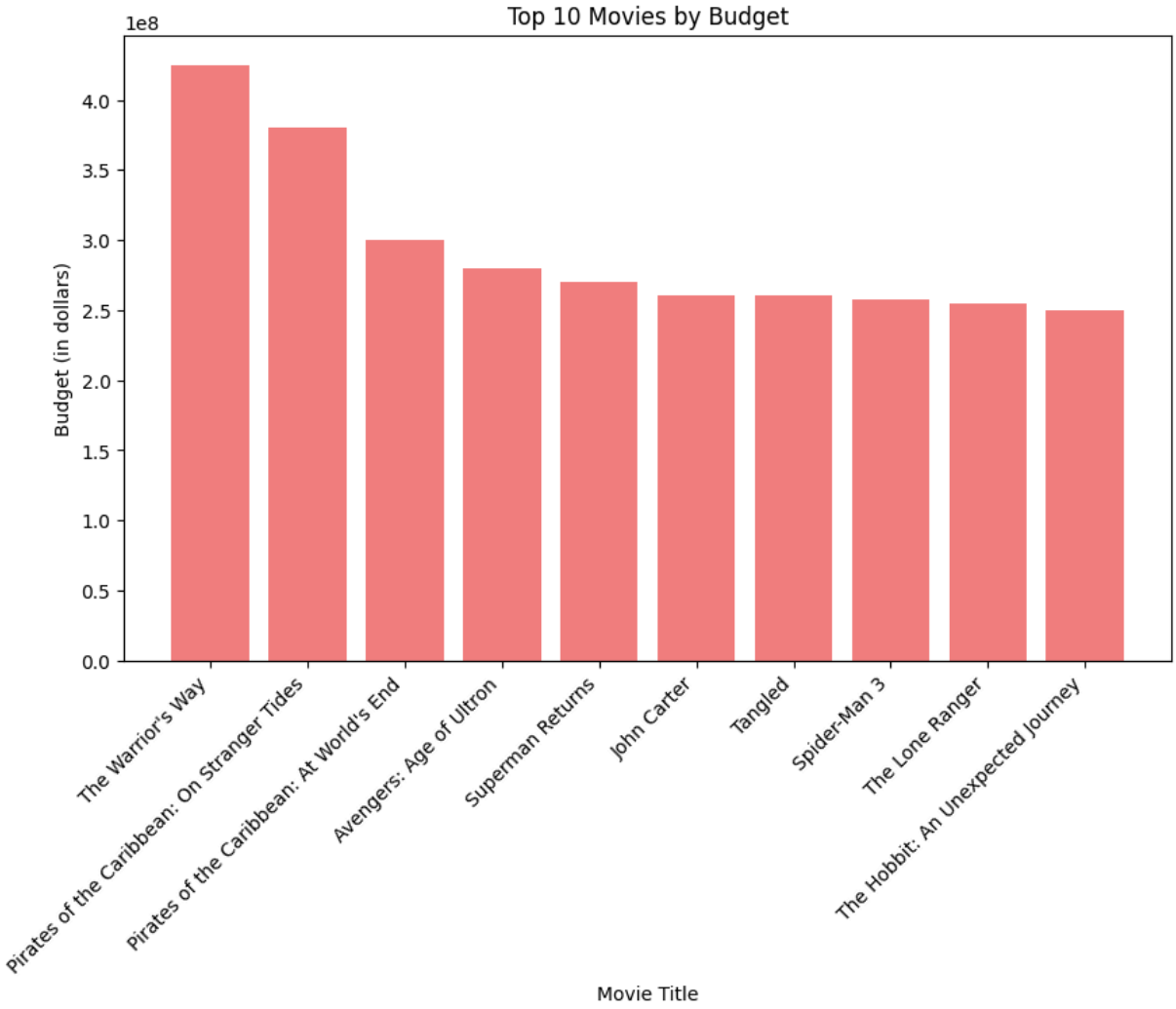
In [131…

```
def plot_top10(data, x_var, y_var, title, x_label, y_label, color='skyblue', horizon
    """
    Plots the top 10 values for a given x and y variable from a DataFrame.

    Parameters:
        data (DataFrame): The dataset to plot.
        x_var (str): Column name for the x-axis.
        y_var (str): Column name for the y-axis.
        title (str): Title of the plot.
        x_label (str): Label for the x-axis.
        y_label (str): Label for the y-axis.
        color (str): Color of the bars.
        horizontal (bool): If True, makes a horizontal bar plot.
    """
    # Sort data by y_var in descending order and select the top 10 rows
    top10_data = data.sort_values(by=y_var, ascending=False).head(10)

    # Set up the figure size
    plt.figure(figsize=(10, 6))

    # Plot horizontally if specified, otherwise plot vertically
    if horizontal:
        plt.barh(top10_data[x_var], top10_data[y_var], color=color)
        plt.xlabel(x_label)
        plt.ylabel(y_label)
        plt.gca().invert_yaxis()  # Invert y-axis to have the highest value at the t
    else:
        plt.bar(top10_data[x_var], top10_data[y_var], color=color)
        plt.xlabel(x_label)
        plt.ylabel(y_label)
        plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for readability

    # Set the title of the plot and display it
    plt.title(title)
    plt.show()
```

Here, I created a bar chart showing the budgets of the `top 10 movies` . by call plot_top10 function and change the argument

In [132…

```
plot_top10(top_budget_movies, 'original_title', 'budget',
           'Top 10 Movies by Budget', 'Movie Title', 'Budget (in dollars)', color='l
```



Research Question 2: Which Movies Received the Most Votes from Viewers?

To find the movies with the most viewer engagement, we sorted the dataset by the `vote_count` column in descending order. This helped us identify the movies with the highest number of votes, showing which movies generated the most interest from audiences.

Sort the Movies by vote count

In [133…

```
# Sort movies by vote count in descending order and display the top 10
top_voted_movies = df.sort_values(by='vote_count', ascending=False).head(10)
top_voted_movies[['original_title', 'vote_count']]
```

Out[133…

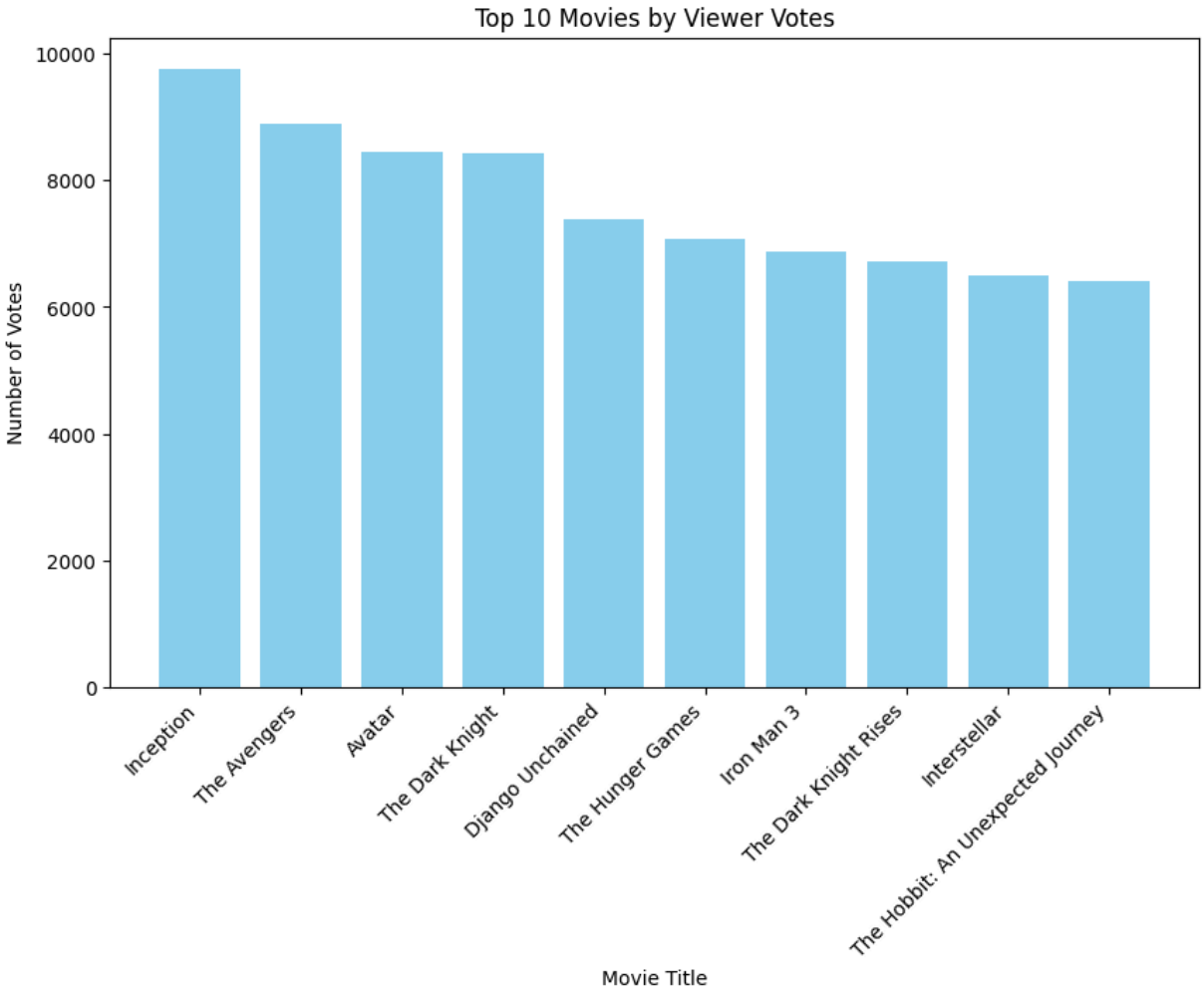
	original_title	vote_count
1919	Inception	9767
4361	The Avengers	8903
1386	Avatar	8458
2875	The Dark Knight	8432
4364	Django Unchained	7375
4382	The Hunger Games	7080
5425	Iron Man 3	6882
4363	The Dark Knight Rises	6723
629	Interstellar	6498
4367	The Hobbit: An Unexpected Journey	6417

The table above lists the top 10 movies with the highest vote counts. These are the movies that received the most viewer votes, indicating they were widely discussed and popular.

The Visualization (Plot)

In [134…

```
plot_top10(top_voted_movies, 'original_title', 'vote_count',
           'Top 10 Movies by Viewer Votes', 'Movie Title', 'Number of Votes', color=
```



Research Question 3: What is the Average Rating for Movies in Each Genre?

To find the average rating for each genre, we:

1. Split the `genres` column (since some movies have multiple genres).
2. Calculated the average `vote_average` for each genre.

In [135…

```
# Remove rows with missing genres and split genres
df_genres = df.dropna(subset=['genres'])
df_genres['genres'] = df_genres['genres'].str.split('|')
df_genres = df_genres.explode('genres')

# Calculate average rating by genre
average_rating_by_genre = df_genres.groupby('genres')['vote_average'].mean()
average_rating_by_genre
```

Out[135…

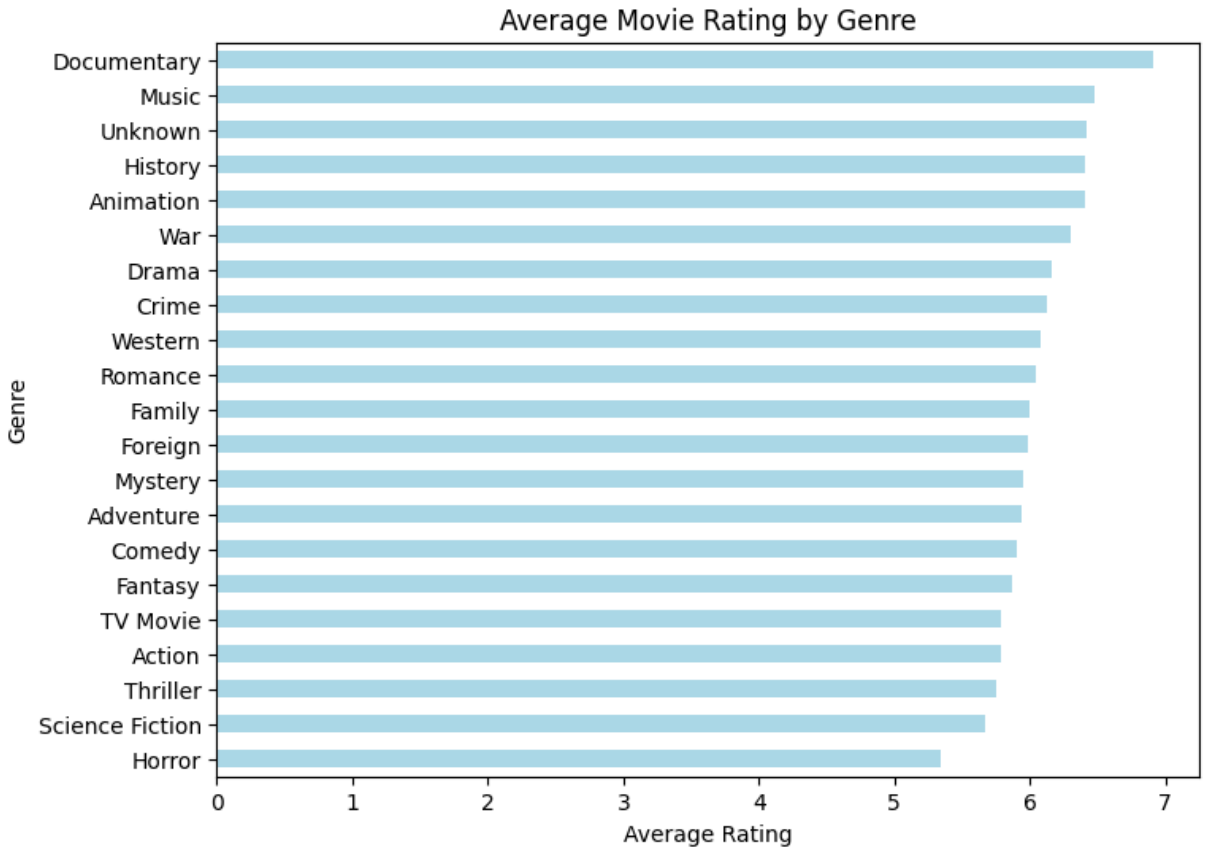
```
genres
Action              5.787752
Adventure           5.940585
Animation           6.403147
Comedy              5.905167
Crime               6.124889
Documentary         6.908462
Drama               6.165546
Family              5.997563
Fantasy             5.863537
Foreign             5.981383
History             6.410479
Horror              5.337447
Music               6.480392
Mystery             5.946790
Romance             6.042874
Science Fiction     5.665582
TV Movie            5.788024
Thriller            5.750671
Unknown             6.421739
War                 6.297778
Western             6.083030
Name: vote_average, dtype: float64
```

The table above shows the average rating for each genre, it help us to see which genres tend to have higher viewer ratings.

The Visualization (Plot)

In [136…

```
# Plot average rating by genre with a horizontal bar plot
plt.figure(figsize=(8, 6))
average_rating_by_genre.sort_values().plot(kind='barh', color='lightblue')
plt.title('Average Movie Rating by Genre')
plt.xlabel('Average Rating')
plt.ylabel('Genre')
plt.show()
```



Research Question 4: Is the Budget Related to a Higher Average Vote?

To see if there's a relation between a movie's budget and its average viewer rating, we'll calculate the correlation between the `budget` and `vote_average` columns. A positive correlation would mean that higher budgets tend to result in higher ratings, while a negative correlation would suggest the opposite.

- If the correlation value is close to 1, it means there's a strong positive relationship (higher budgets might lead to higher ratings).
- If it's close to -1, it means there's a strong negative relationship.
- If it's near 0, it suggests no strong connection between budget and ratings.

In [137…
```
# Calculate correlation between budget and average vote
budget_vote_correlation = df[['budget', 'vote_average']].corr().iloc[0, 1]
print(f"Correlation between budget and average vote is: {budget_vote_correlation}")
```

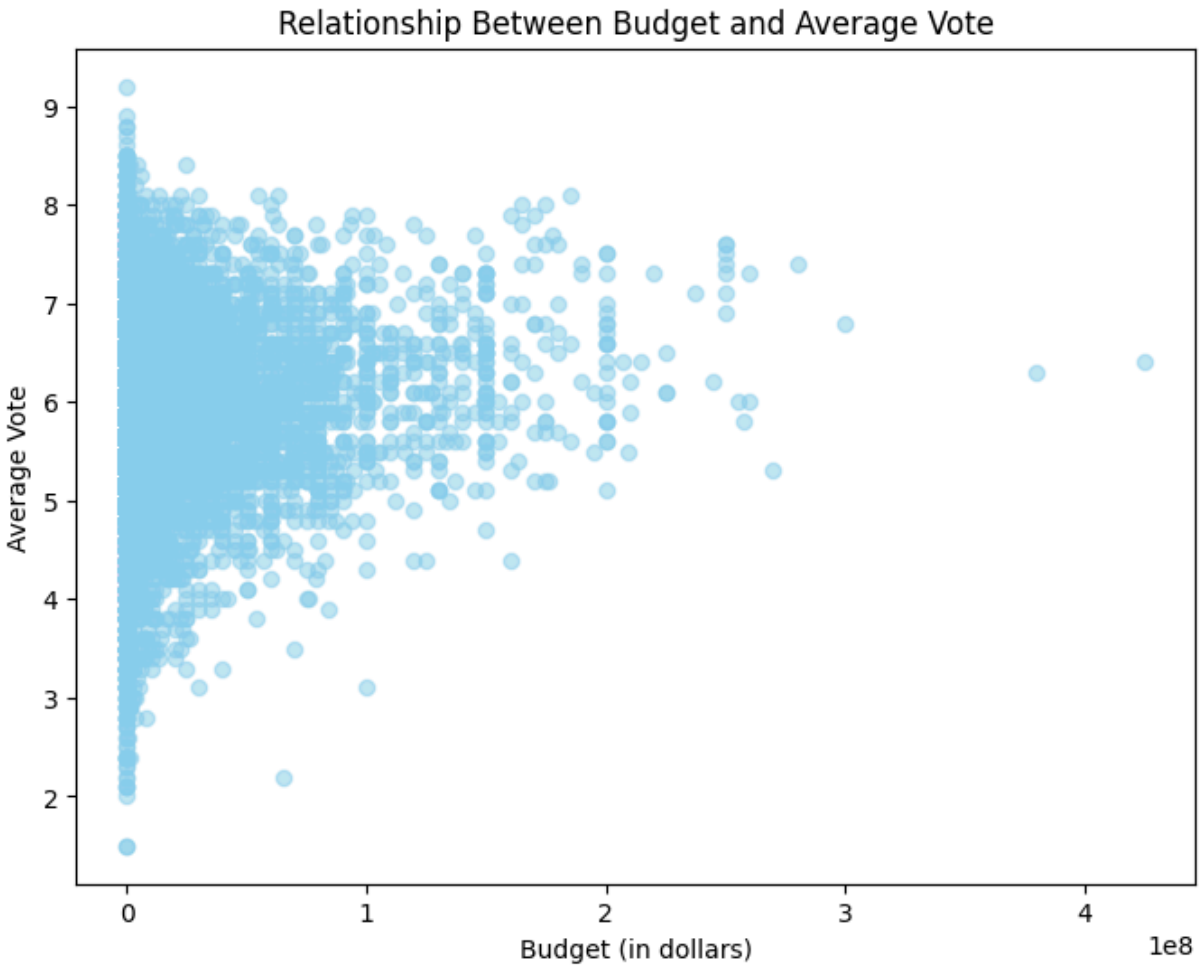
```
Correlation between budget and average vote is: 0.08106672575599086
```

The correlation value is close to zero, which suggests that there is no strong relationship between a movie's budget and its average viewer rating.

The Visualization (Plot)

In [138…
```
# Scatter plot of budget vs. average vote
plt.figure(figsize=(8, 6))
```

```
plt.scatter(df['budget'], df['vote_average'], alpha=0.5, color='skyblue')
plt.title('Relationship Between Budget and Average Vote')
plt.xlabel('Budget (in dollars)')
plt.ylabel('Average Vote')
plt.show()
```



Research Question 5: What Are the Most Common Release Years in the Dataset?

To find the years with the most movie releases, we counted the number of movies released each year and sorted the results in descending order.

In [139…

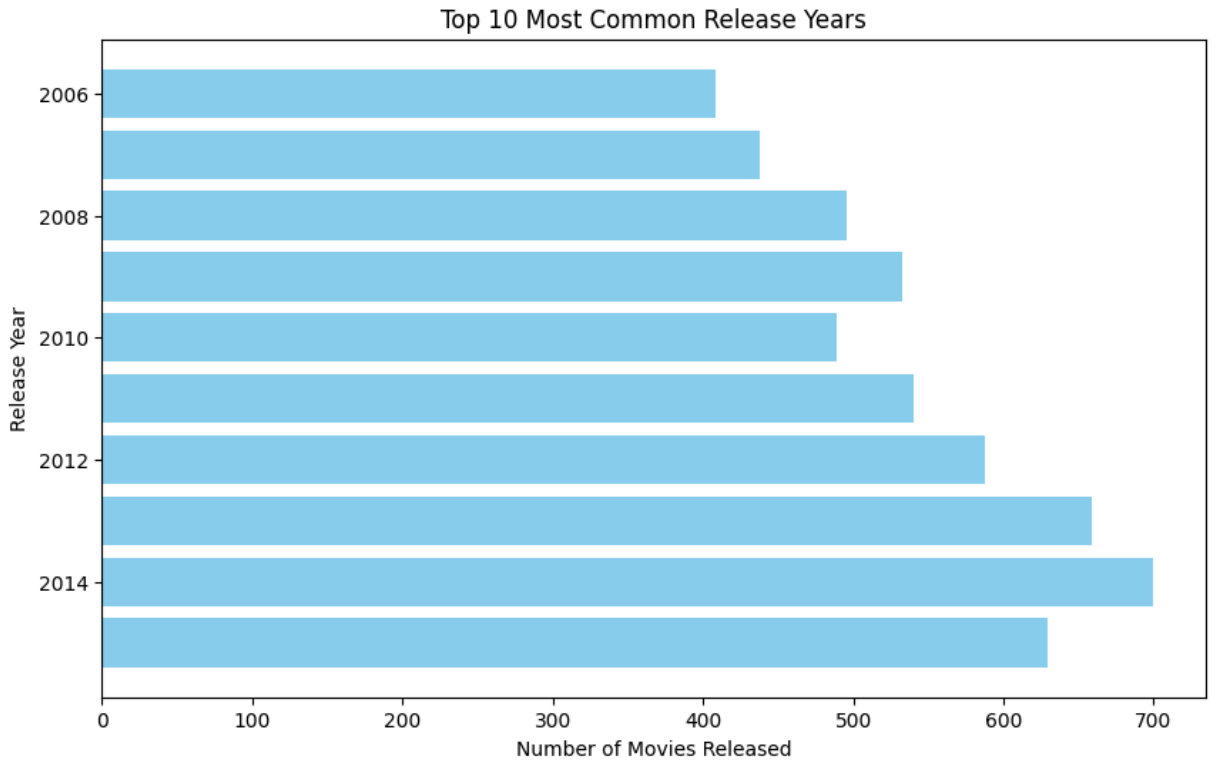
```
# Count the number of movies released each year
release_year_counts = df['release_year'].value_counts()

# Sort the counts in descending order to get the most common release years
release_year_counts = release_year_counts.sort_values(ascending=False)

# Display the top 10 most common release years
release_year_counts.head(10)
```

Out[139…
```
2014    700
2013    659
2015    629
2012    588
2011    540
2009    533
2008    496
2010    489
2007    438
2006    408
Name: release_year, dtype: int64
```

The table below shows the top 10 most common release years, highlighting the years with the highest number of movie releases. This gives us a quick view of the most active years in movie production within the dataset.

The Visualization (Plot)

In [140…
```
plot_top10(release_year_counts.head(10).reset_index(), 'index', 'release_year',
           'Top 10 Most Common Release Years', 'Number of Movies Released', 'Release
           color='skyblue', horizontal=True)
```



Research Question 6: Is There a Relationship Between Movie Popularity and Revenue?

before ploting,I display a small sample of the data to get a feel for the popularity and revenue values.

In [141…

```
# Display a sample table of popularity and adjusted revenue columns
popularity_revenue_sample = df[['popularity', 'revenue_adj']].sample(10)
popularity_revenue_sample
```

Out[141…

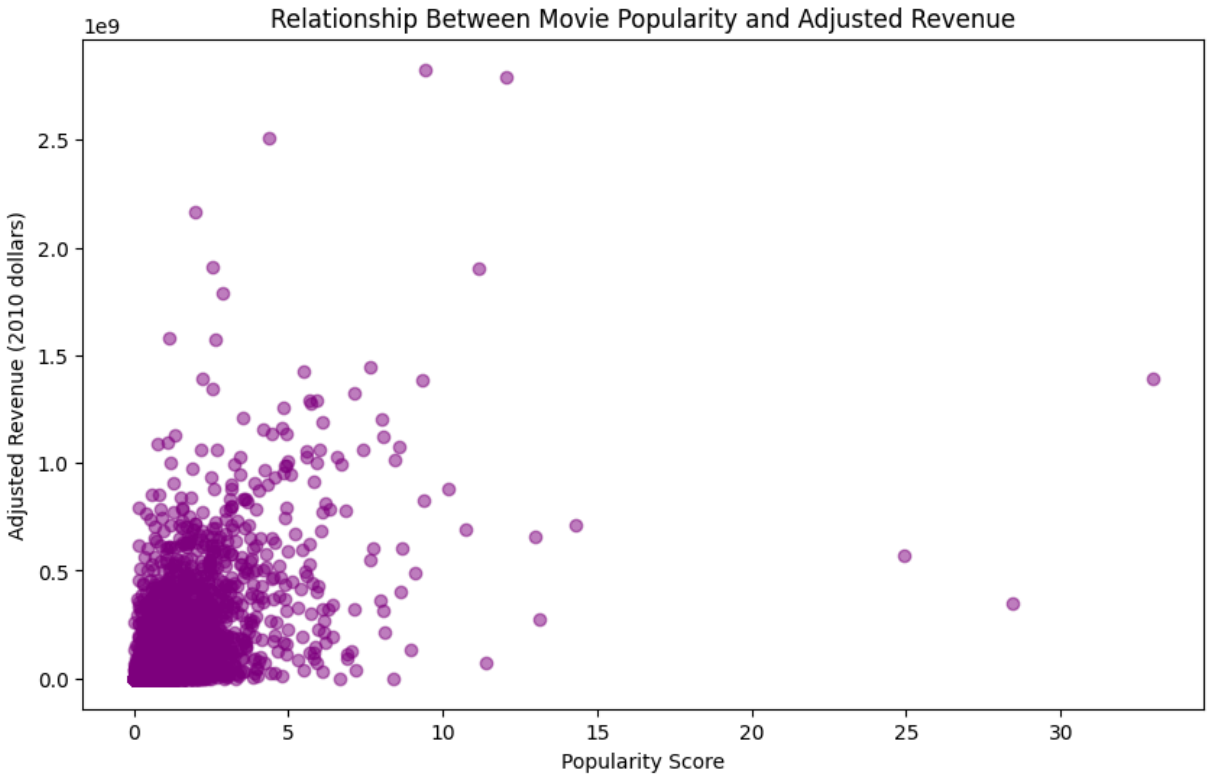
	popularity	revenue_adj
3253	0.176665	0.000000e+00
7822	0.010471	0.000000e+00
1890	0.086490	0.000000e+00
8923	0.391453	0.000000e+00
2200	0.319013	0.000000e+00
7284	0.321141	4.409595e+08
5833	0.238632	1.420833e+07
7907	0.633247	8.067867e+07
8879	0.035184	0.000000e+00
2292	0.200531	0.000000e+00

The Visualization (Plot)

Now, we'll create a scatter chart to see if there's a pattern between popularity and revenue.

In [142…

```
# Scatter plot of popularity vs. adjusted revenue
plt.figure(figsize=(10, 6))
plt.scatter(df['popularity'], df['revenue_adj'], alpha=0.5, color='purple')
plt.title('Relationship Between Movie Popularity and Adjusted Revenue')
plt.xlabel('Popularity Score')
plt.ylabel('Adjusted Revenue (2010 dollars)')
plt.show()
```



The scatter plot shows that most movies have low popularity and lower revenue (clustered in the bottom-left). Some highly popular movies earn much higher revenue, but there's no clear pattern—popularity doesn't strongly predict revenue. This suggests other factors likely influence a movie's earnings.

Research Question 7: How Has the Average Budget for Movies Changed Over the Years?

To explore how movie production budgets have changed over time, we calculated the average `budget` for each release year and plotted it on a line chart. This helps us see if studios have been spending more or less on making movies over the years.

In [143…

```
# Calculate the average budget for each release year
average_budget_by_year = df.groupby('release_year')['budget'].mean()
# Print the average budget by year
print("Average Movie Budget by Release Year:")
print(average_budget_by_year)
```

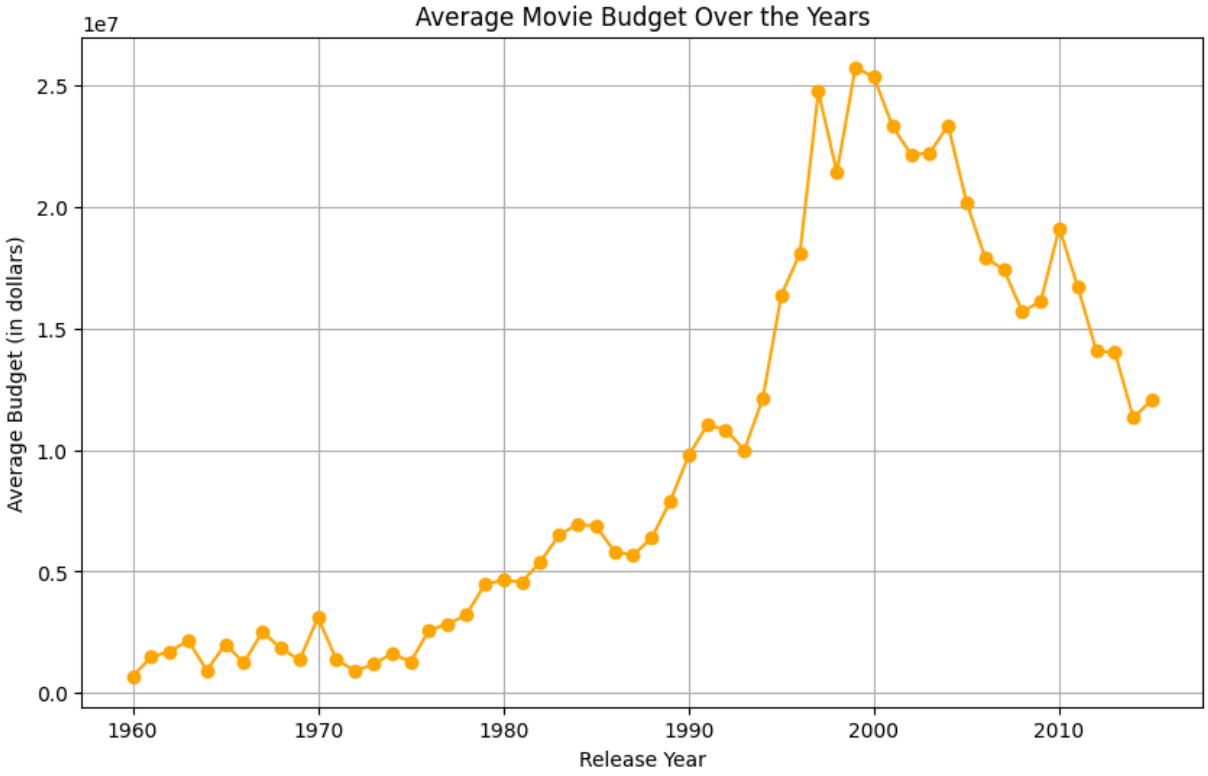
Average Movie Budget by Release Year:
release_year
1960   6.892796e+05
1961   1.488290e+06
1962   1.710066e+06
1963   2.156809e+06
1964   9.400753e+05
1965   2.005860e+06
1966   1.251191e+06
1967   2.516305e+06
1968   1.844590e+06
1969   1.359003e+06
1970   3.096755e+06
1971   1.381764e+06
1972   9.069813e+05
1973   1.185287e+06
1974   1.637660e+06
1975   1.279068e+06
1976   2.598936e+06
1977   2.834737e+06
1978   3.215339e+06
1979   4.470421e+06
1980   4.647436e+06
1981   4.558022e+06
1982   5.404877e+06
1983   6.488843e+06
1984   6.944876e+06
1985   6.868997e+06
1986   5.822592e+06
1987   5.675646e+06
1988   6.381710e+06
1989   7.880703e+06
1990   9.772137e+06
1991   1.102431e+07
1992   1.084034e+07
1993   9.997914e+06
1994   1.211526e+07
1995   1.637648e+07
1996   1.807374e+07
1997   2.474524e+07
1998   2.142695e+07
1999   2.573766e+07
2000   2.534229e+07
2001   2.331382e+07
2002   2.216030e+07
2003   2.220590e+07
2004   2.335616e+07
2005   2.017386e+07
2006   1.790732e+07
2007   1.743281e+07
2008   1.568803e+07
2009   1.612399e+07
2010   1.913088e+07
2011   1.670028e+07
2012   1.407157e+07
2013   1.401523e+07
2014   1.131999e+07

```
2015    1.207718e+07
Name: budget, dtype: float64
```

The Visualization (Plot)

In [144…

```python
# Plot the average movie budget over the years as a line chart
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
average_budget_by_year.plot(kind='line', color='orange', marker='o')
plt.title('Average Movie Budget Over the Years')
plt.xlabel('Release Year')
plt.ylabel('Average Budget (in dollars)')
plt.grid(True)
plt.show()
```



Research Question 8: Which Movies Have the Highest Revenue, Adjusted for Inflation?

To find the movies with the highest revenue adjusted for inflation, we used the `revenue_adj` column. This column provides the revenue adjusted to 2010 dollars, allowing for a fair comparison across movies released in different years.

In [145…

```python
top_revenue_movies = df.sort_values(by='revenue_adj', ascending=False).head(10)
print("Top 10 Movies by Adjusted Revenue:")
print(top_revenue_movies[['original_title', 'revenue_adj']])
```

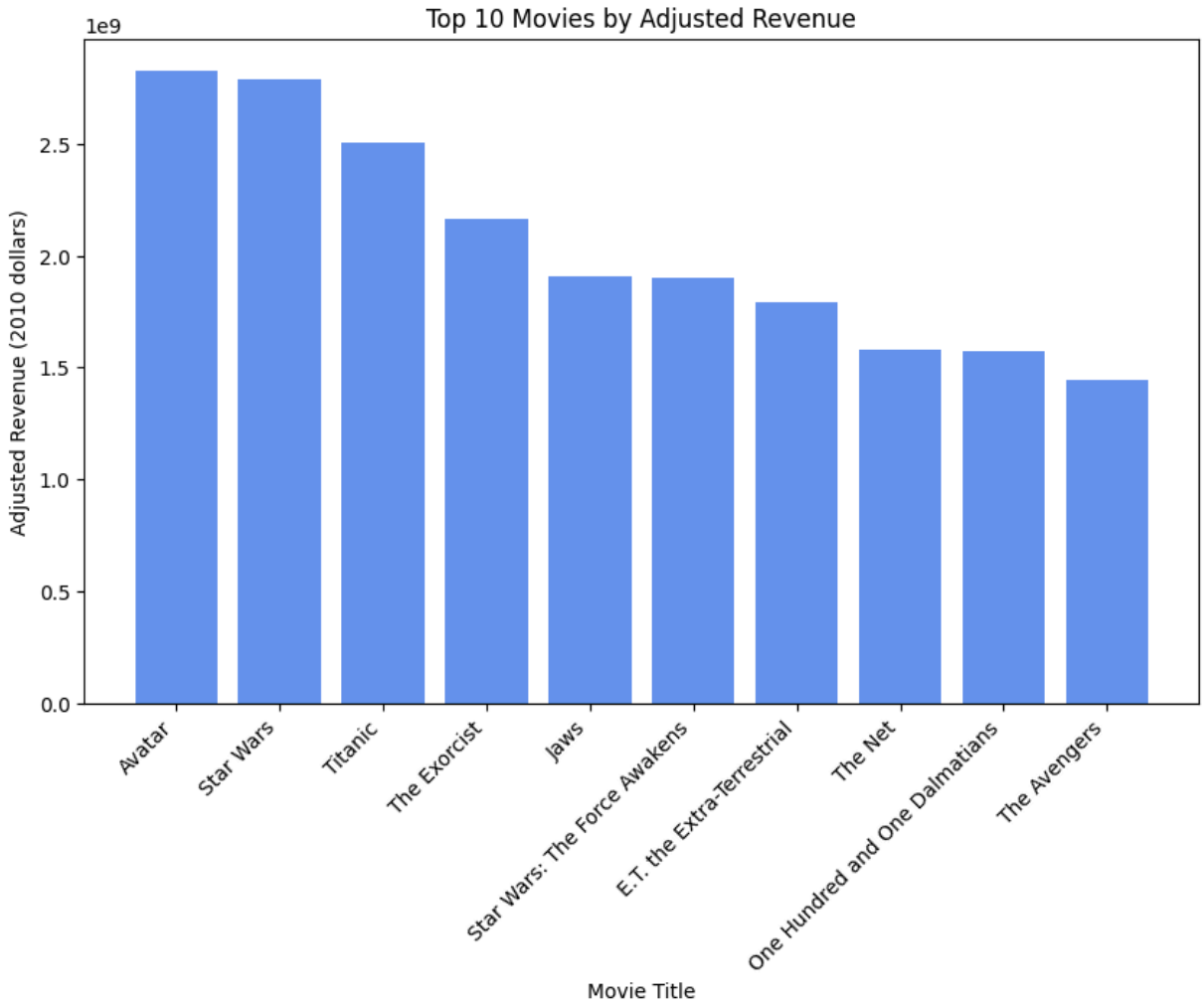
```
Top 10 Movies by Adjusted Revenue:
                      original_title   revenue_adj
1386                          Avatar  2.827124e+09
1329                       Star Wars  2.789712e+09
5231                         Titanic  2.506406e+09
10594                    The Exorcist  2.167325e+09
9806                             Jaws  1.907006e+09
3        Star Wars: The Force Awakens  1.902723e+09
8889       E.T. the Extra-Terrestrial  1.791694e+09
8094                           The Net  1.583050e+09
10110  One Hundred and One Dalmatians  1.574815e+09
4361                     The Avengers  1.443191e+09
```

The table below lists the top 10 movies with the highest adjusted revenue, showing which movies achieved the most financial success.

The Visualization (Plot)

In [146…

```
plot_top10(top_revenue_movies.set_index('original_title').reset_index(), 'original_t
           'Top 10 Movies by Adjusted Revenue', 'Movie Title', 'Adjusted Revenue (20
           color='cornflowerblue')
```



Research Question 9: What is the Average Runtime for Movies in Each Genre?

I calculated the average `runtime` for each `genreTo` to see if some genres tend to have longer or shorter movies. This can reveal whether genres like drama or action are generally longer, while genres like comedy might be shorter.

In [147…

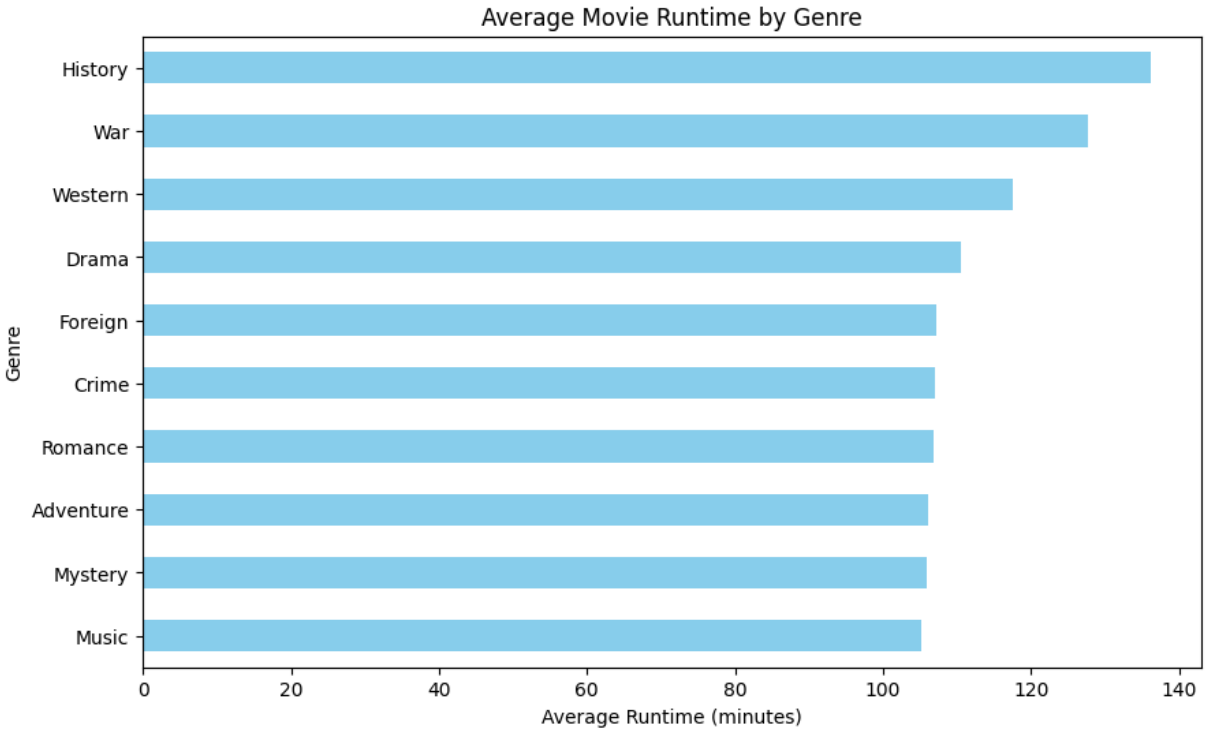
```
# Split genres and calculate average runtime for each genre
df_genres = df.dropna(subset=['genres'])  # Remove rows with missing genres
df_genres['genres'] = df_genres['genres'].str.split('|')
df_genres = df_genres.explode('genres')

# Calculate the average runtime by genre
average_runtime_by_genre = df_genres.groupby('genres')['runtime'].mean().sort_values
print("Average Runtime by Genre:")
print(average_runtime_by_genre.head(10))
```

```
Average Runtime by Genre:
genres
History      136.206587
War          127.625926
Western      117.575758
Drama        110.478151
Foreign      107.228723
Crime        106.917282
Romance      106.891355
Adventure    106.173351
Mystery      105.928395
Music        105.137255
Name: runtime, dtype: float64
```

The Visualization (Plot)

In [148…

```
import matplotlib.pyplot as plt

# Plot the average runtime by genre as a horizontal bar plot
plt.figure(figsize=(10, 6))
average_runtime_by_genre.head(10).plot(kind='barh', color='skyblue')
plt.title('Average Movie Runtime by Genre')
plt.xlabel('Average Runtime (minutes)')
plt.ylabel('Genre')
plt.gca().invert_yaxis()  # Show the longest average runtime at the top
plt.show()
```



Research Question 10: Which Movies Had the Highest Popularity Scores Each Year?

To find out which movies were the most popular each year, we looked at the `popularity` score and selected the top movie for each release year. This gives us a look at the movies that captured the most audience attention annually.

In [149…

```
# Find the most popular movie for each year by sorting by popularity within each yea
most_popular_movies_by_year = df.loc[df.groupby('release_year')['popularity'].idxmax

# Sort by release year in ascending order for a clean display
most_popular_movies_by_year = most_popular_movies_by_year.sort_values(by='release_ye

# Print the table with headers for readability
print("Most Popular Movies by Year:\n")
print(most_popular_movies_by_year.to_string(index=False))
```

Most Popular Movies by Year:

release_year	original_title	popularity
1960	Psycho	2.610362
1961	One Hundred and One Dalmatians	2.631987
1962	Dr. No	3.170651
1963	From Russia With Love	2.508235
1964	Goldfinger	3.153791
1965	Thunderball	1.910465
1966	How the Grinch Stole Christmas!	1.227582
1967	The Jungle Book	2.550704
1968	2001: A Space Odyssey	3.309196
1969	On Her Majesty's Secret Service	1.778746
1970	The Aristocats	1.936962
1971	A Clockwork Orange	3.072555
1972	The Godfather	5.738034
1973	Robin Hood	2.272486
1974	The Godfather: Part II	3.264571
1975	One Flew Over the Cuckoo's Nest	3.258151
1976	Taxi Driver	2.582657
1977	Star Wars	12.037933
1978	Grease	1.697618
1979	Alien	4.935897
1980	The Empire Strikes Back	5.488441
1981	Raiders of the Lost Ark	4.578300
1982	Blade Runner	4.215642
1983	Return of the Jedi	4.828854
1984	The Terminator	4.831966
1985	Back to the Future	6.095293
1986	Aliens	2.485419
1987	Predator	3.474728
1988	Die Hard	3.777441
1989	The Little Mermaid	4.143585
1990	Total Recall	2.679627
1991	Beauty and the Beast	3.852269
1992	Reservoir Dogs	4.586426
1993	Groundhog Day	2.571339
1994	Pulp Fiction	8.093754
1995	Se7en	4.765359
1996	Independence Day	4.480733
1997	Eddie Izzard: Glorious	6.668990
1998	The Truman Show	4.180540
1999	Fight Club	8.947905
2000	Gladiator	4.271452
2001	The Lord of the Rings: The Fellowship of the Ring	8.575419
2002	The Lord of the Rings: The Two Towers	8.095275
2003	The Lord of the Rings: The Return of the King	7.122455
2004	Harry Potter and the Prisoner of Azkaban	5.827781
2005	Harry Potter and the Goblet of Fire	5.939927
2006	Underworld: Evolution	5.838503
2007	Pirates of the Caribbean: At World's End	4.965391
2008	The Dark Knight	8.466668
2009	Avatar	9.432768
2010	Inception	9.363643
2011	Underworld: Endless War	8.411577
2012	The Avengers	7.637767
2013	Frozen	6.112766

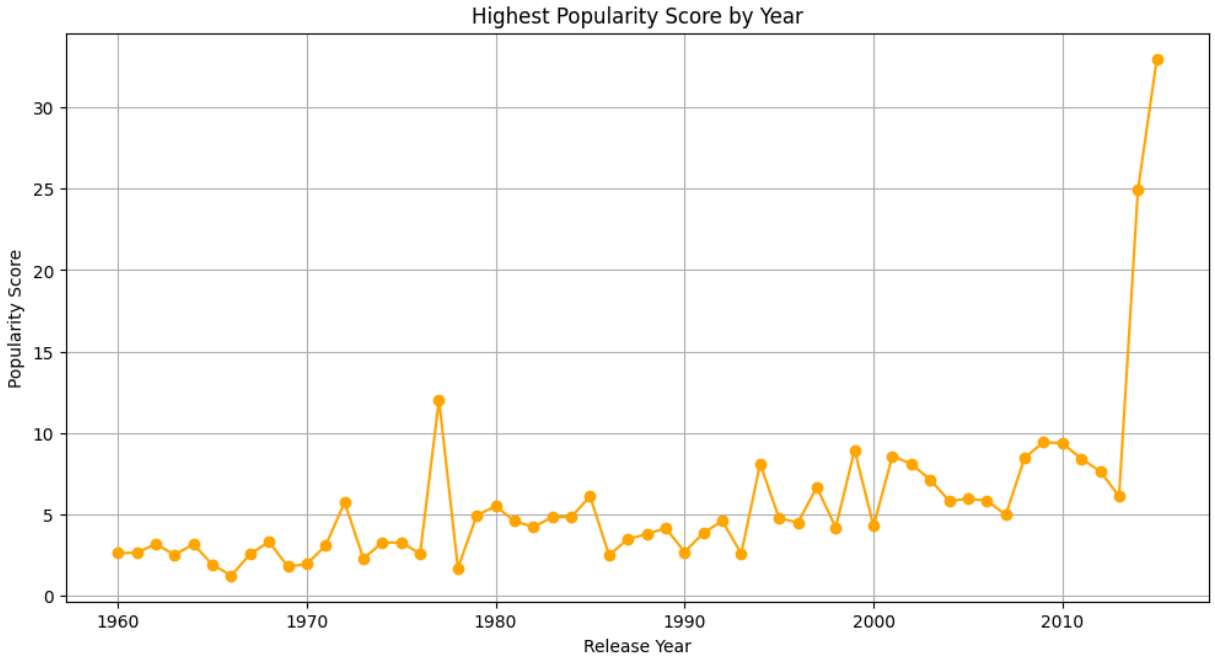
```
2014                                  Interstellar   24.949134
2015                                Jurassic World   32.985763
```

The table above shows the most popular movie from each year, based on its popularity score.

The Visualization (Plot)

In [150…

```
import matplotlib.pyplot as plt

# Plot the most popular movie's popularity score by year
plt.figure(figsize=(12, 6))
plt.plot(most_popular_movies_by_year['release_year'], most_popular_movies_by_year['p
plt.title('Highest Popularity Score by Year')
plt.xlabel('Release Year')
plt.ylabel('Popularity Score')
plt.grid(True)
plt.show()
```



Conclusions

Finaly, my analysis of the TMDB 5000 Movie Dataset shows some clear trends in movies and audience preferences. Big-budget movies, like **Pirates of the Caribbean** and **The Avengers**, often make the most money, proving that investing in large productions usually pays off. Popular movies like **Inception**, **Avatar**, and **The Dark Knight** attract more audience votes, likely due to their unique stories or big fan bases. Some genres, like documentaries and historical movies, get higher ratings, suggesting that viewers appreciate films that are informative or emotional.

Even though popular movies tend to earn more, there isn't a strong link between popularity and revenue, meaning other factors like marketing and franchise loyalty also play a role. We see that more movies are made each year, especially in recent years, probably due to higher

demand and better production technology. Movie budgets have also grown over time, reflecting higher costs for effects, actors, and complex scenes. Every year, certain movies like **Star Wars** or **Jurassic World** stand out as the most popular, shaping the year in film.

Overall, this analysis helps us understand what makes a movie successful and what audiences enjoy. It also gives insight into how the movie industry has changed over time.

Summary of Findings

my analysis of the TMDB 5000 Movie Dataset shows several interesting trends:

1. **Big Budgets Lead to High Revenue**: Movies with big budgets, like *Pirates of the Caribbean** and **The Avengers**, often make the most money. This suggests that investing a lot in blockbuster films usually pays off.

2. **Popular Movies and Unique Stories**: Movies with unique stories, like **Inception**, **Avatar**, and *The Dark Knight*, get the most audience votes, showing that these types of films attract strong interest and engagement.

3. **Higher Ratings for Certain Genres**: Audiences seem to enjoy documentaries and historical movies, which have higher ratings on average. This may mean that viewers appreciate movies that are informative or emotionally moving.

4. **Popularity and Revenue Link**: Popular movies tend to make more money, but the link isn't very strong. This suggests that factors like good marketing or being part of a well-known series also play a big role in revenue.

5. **More Movies Over Time**: The number of movies made each year has been going up, especially in recent years. This increase is likely due to higher demand and advances in production technology.

6. **Growing Budgets**: Over time, movie budgets have increased, which likely reflects the rising costs of special effects, high-profile actors, and complex productions.

7. **Yearly Standout Movies**: Each year has standout movies that capture a lot of attention, like **Star Wars** in 1977 and **Jurassic World** in 2015, which shape the movie landscape for that year.

In summary, our findings help us understand what makes movies successful, showing how factors like budget, genre, and popularity influence what audiences enjoy. This analysis also shows how the film industry has grown and adapted over time.

Limitations

While this analysis offers useful insights, there are some limitations to keep in mind:

1. **Missing Data**: Some columns have missing values, which could slightly impact the accuracy of certain results.
2. **Outdated Financial Adjustments**: Budgets and revenues are adjusted only up to 2010, so the financial data may not fully reflect recent economic changes.
3. **Multiple Genres**: Movies often have more than one genre, which can make it harder to analyze trends specific to a single genre.
4. **Undefined Popularity Metric**: The dataset does not clearly explain how popularity scores are calculated, limiting our understanding of this metric.
5. **Data Up to 2015 Only**: The dataset includes movies only through 2015, so it might not reflect recent trends in the film industry.
6. **Lack of External Factors**: Factors like marketing, global events, or cultural shifts, which can affect a movie's success, are not included in this data.

These limitations mean that while our findings are insightful, they should be viewed with caution, especially when considering current trends.