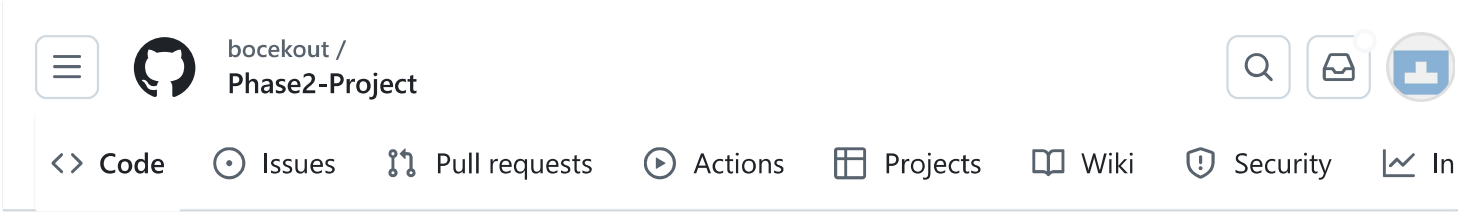


Phase2-Project / FINAL.ipynb

bocekout Almost final commit. One more thing to do. 903a458 · 4 hours ago

3358 lines (3358 loc) · 1.04 MB

Preview Code Blame Raw

A Guide to Box Office Success

Authored by Elif Surucu & Ricky Bocek

Project Overview

This analysis of movie data, sourced from Kaggle, The Numbers, and IMDb, investigates the financial success of movies by analyzing the risk-reward relationship between production budget and profitability, adjusted to the yearly Consumer Price Index. This analysis leverages historical movie data to identify patterns and key features, such as genres, directors, and number of principals, that correlate with higher profitability. By applying statistical techniques like ANOVA and linear regression, the project uncovers which genres and budget categories lead to the most successful movies. Additionally, the project offers recommendations for future movie production, focusing on maximizing ROI based on these identified success factors.

Business Understanding

The movie industry operates in a highly competitive environment where production companies aim to maximize profitability while minimizing financial risk. However, predicting the financial success of a movie is challenging due to various factors like genre, budget, cast, and market trends. This project seeks to address this uncertainty by identifying the key features that drive higher returns on investment (ROI) for movies. The goal is to provide movie studios and producers with actionable insights to optimize their budgets and make informed decisions about which genres, directors, and budget levels are more likely to result in profitable outcomes. By analyzing historical data, the

project can help studios focus their resources on the most promising projects, thereby improving financial performance in an unpredictable market.

Data Understanding

The project uses movie data sourced from Kaggle, The Numbers, and IMDb and historical Consumer Price Index tables from the US Bureau of Labor Statistics. Key data fields include genres, adjusted production budgets, adjusted domestic and worldwide gross earnings, and directors' information. The focus is on maximizing ROI to evaluate the financial success of movies, while accounting for risk as determined by production budget.

Key insights from the data include:

Genres like Horror, Family, and Comedy often deliver high ROI, especially for low-budget movies. Directors with strong track records in specific genres tend to have higher ROI. Data cleaning involved uniting disparate data sets, adjusting for inflation, handling missing values and normalizing multi-genre entries. This data provided a strong foundation for further analysis using ANOVA tests and linear regression models to identify key success factors for movies.

Source Details

The Kaggle dataset has detailed financial and production information for movies, which is up-to-date and suitable for analysis.

The Numbers is an online movie database service. The dataset from them is somewhat outdated but reasonably large. It incorporates few features but provides comprehensive budget and gross earnings data.

Consumer Price Index (CPI) Tables sourced from the US Bureau of Labor Statistics are used to adjust financial data for inflation, ensuring all monetary figures are standardized over time.

IMDb SQLite3 database is a rich and comprehensive source of movie features for analysis. However, it lacks financial metrics for evaluating movie success. It is structured as follows:

 ERD showing primary datasets "movie_basics" and "persons" with tables of relationships between movies and persons and some additional movie data tables

In [1]:

```
# Importing several standard modules in our Preparation and Analysis
import pandas as pd
import numpy as np
import sqlite3
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
```

```
import scipy.stats as stats
import statsmodels.api as sm
from statsmodels.stats.multicomp import pairwise_tukeyhsd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
import matplotlib.ticker as mtick
```

In [2]:

```
#Connecting to our IMDB Sqlite database file
con = sqlite3.connect('Data/im.db')
```

In [3]:

```
#Use Pandas to quickly validate EDA information
pd.read_sql('''

SELECT *
FROM sqlite_master

''', con)
```

Out[3]:

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

In [4]:

```
pd.read_sql('''

SELECT *
FROM movie_basics
LIMIT 15

''', con)
```

Out[4]:

	movie_id	primary_title	original_title	start_year	runtime_minutes	
0	tt0063540	Sunghursh	Sunghursh	2013	175.0	Actior

1	tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	114.0	Bio
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	C
4	tt0100275	The Wandering Soap Opera	La Telenovela Errante	2017	80.0	Comedy,D
5	tt0111414	A Thin Life	A Thin Life	2018	75.0	
6	tt0112502	Bigfoot	Bigfoot	2017	NaN	
7	tt0137204	Joe Finds Grace	Joe Finds Grace	2017	83.0	Adventure,Anim
8	tt0139613	O Silêncio	O Silêncio	2012	NaN	Docum
9	tt0144449	Nema aviona za Zagreb	Nema aviona za Zagreb	2012	82.0	
10	tt0146592	Pál Adrienn	Pál Adrienn	2010	136.0	
11	tt0154039	So Much for Justice!	Oda az igazság	2010	100.0	
12	tt0159369	Cooper and Hemingway: The True Gen	Cooper and Hemingway: The True Gen	2013	180.0	
13	tt0162942	Children of the Green Dragon	A zöld sárkány gyermekei	2010	89.0	
14	tt0170651	T.G.M. - osvoboditel	T.G.M. - osvoboditel	2018	60.0	

Data Preparation

In our project, data preparation was a key step to ensure accurate analysis.

We addressed missing values, removed duplicates, and ensured data consistency across all relevant fields. We merged datasets (movies, directors, and others) based on movie_id and person_id to consolidate essential information like gross earnings, genres, and director details. Handling Multi-Genres: We split multi-genre entries to analyze each genre's impact separately on metrics like ROI. We normalized ROI by considering production budgets and gross earnings adjusted for inflation. We created new features like approval index and normalized budgets to improve insights.We filtered directors based on relevant criteria such as age and status (alive) for modern analysis.

In [5]:
```
# Reading in financial data csvs
tn_data = pd.read_csv('Data/tn.movie_budgets.csv')
kaggle_data = pd.read_csv('Data/movie_statistic_dataset.csv')
inflation_data  = pd.read_csv('Data/US_CPI.csv')

# Reading in IMDB movie basics
imdb_data = pd.read_sql('''
                        SELECT *
                        FROM movie_basics
                        ''', con)

# We investigated whether data from Box Office Mojo could fill in missing data
bom_data = pd.read_csv('Data/bom.movie_gross.csv')
```

In [6]:
```
# Financial data in TN dataset was formatted as a string with $#,###,###
# converts monetary values in the dataset from strings to floats
tn_data['production_budget'] = tn_data['production_budget'].str.replace('$', '
tn_data['domestic_gross'] = tn_data['domestic_gross'].str.replace('$', '').str
tn_data['worldwide_gross'] = tn_data['worldwide_gross'].str.replace('$', '').s
```

In [7]:
```
tn_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5782 entries, 0 to 5781
Data columns (total 6 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   id                 5782 non-null   int64 
 1   release_date       5782 non-null   object
 2   movie              5782 non-null   object
 3   production_budget  5782 non-null   float64
 4   domestic_gross     5782 non-null   float64
 5   worldwide_gross    5782 non-null   float64
dtypes: float64(3), int64(1), object(2)
memory usage: 271.2+ KB
```

In [8]:
```
# Filtering tn_data by production budget
tn_data = tn_data[tn_data['production_budget'] != 0]
```

In [9]:
```
# Merging tn_data with bom_data
tn_and_bom = tn_data.merge(bom_data, left_on='movie', right_on='title', how='l
```

In [10]:
```
# Further filtering tn_and_bom based on domestic_gross values
tn_and_bom = tn_and_bom[(tn_and_bom['domestic_gross_x'] != 0) | (tn_and_bom['d
```

In [11]:
```
# the bom_data apparently only contains information for 4 movies that are not
# (3 when you notice one is a remake.)
# BOM data can be safely disregarded.
```

```
tn_and_bom[(tn_and_bom['domestic_gross_x']==0) & (tn_and_bom['worldwide_gross'
```

Out[11]:

	id	release_date	movie	production_budget	domestic_gross_x	worldwide_gross
4768	69	Sep 18, 1967	Point Blank	3000000.0	0.0	0.0
4865	66	Jan 19, 2016	Eden	2300000.0	0.0	0.0
5330	31	Dec 31, 2012	Trance	950000.0	0.0	0.0
5351	52	Dec 31, 2012	Snitch	850000.0	0.0	0.0

In [12]:

```
# filtering the dataset (tn_data) to remove any records where the domestic_gro
tn_data = tn_data.loc[(tn_data['domestic_gross'] != 0)|(tn_data['worldwide_gro
```

In [13]:

```
# combining data from the two different sources (kaggle_data and tn_data) to f
# which includes movie titles, production budgets, and gross earnings from bot
money_data = kaggle_data.merge(tn_data, left_on='movie_title', right_on='movie
```

In [14]:

```
# Shows that aligning on title was largely successful.
money_data.head()
```

Out[14]:

	movie_title	production_date	genres	runtime_minutes	director_nar
0	NaN	NaN	NaN	NaN	Na
1	10 Cloverfield Lane	2016-01-04	Drama,Horror,Mystery	103.0	D Trachtenbe
2	10 Days in a Madhouse	2015-11-11	Drama	111.0	Timothy Hin
3	10 Things I Hate About You	1999-03-31	Comedy,Drama,Romance	97.0	Gil Jung
4	NaN	NaN	NaN	NaN	Na

In [15]:

```
#Since the kaggle set was preserved (4527 entries) we can see that adding the
#But we also see that TN contributed 5434 entries and so there were also some
money_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5937 entries, 0 to 5936
Data columns (total 20 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   movie_title           4528 non-null   object 
 1   production_date       4528 non-null   object 
 2   genres                4528 non-null   object 
 3   runtime_minutes       4528 non-null   float64
 4   director_name         4528 non-null   object 
 5   director_professions  4528 non-null   object 
 6   director_birthYear    4528 non-null   object 
 7   director_deathYear    4528 non-null   object 
 8   movie_averageRating   4528 non-null   float64
 9   movie_numerOfVotes    4528 non-null   float64
 10  approval_Index        4528 non-null   float64
 11  Production budget $   4528 non-null   float64
 12  Domestic gross $      4528 non-null   float64
 13  Worldwide gross $     4528 non-null   float64
 14  id                    5616 non-null   float64
 15  release_date          5616 non-null   object 
 16  movie                 5616 non-null   object 
 17  production_budget     5616 non-null   float64
 18  domestic_gross        5616 non-null   float64
 19  worldwide_gross       5616 non-null   float64
dtypes: float64(11), object(9)
memory usage: 927.8+ KB
```

In [16]:

```
# Handling missing values and filling in the gaps in our merged dataset (money
# Any gaps in financial data from one dataset (Kaggle or The Numbers) are fill

money_data['Production budget $'].fillna(money_data['production_budget'], inpl
money_data['Domestic gross $'].fillna(money_data['domestic_gross'], inplace=Tr
money_data['Worldwide gross $'].fillna(money_data['worldwide_gross'], inplace=
money_data['movie_title'].fillna(money_data['movie'], inplace=True)
money_data['production_date'] = pd.to_datetime(money_data['production_date'],
money_data['release_date'] = pd.to_datetime(money_data['release_date'], errors
money_data['production_date'].fillna(money_data['release_date'], inplace=True)
money_data['year'] = money_data['production_date'].dt.year
print(money_data.head())
```

```
                movie_title production_date                genres  \
0       (500) Days of Summer      2009-07-17                   NaN   
1        10 Cloverfield Lane      2016-01-04  Drama,Horror,Mystery   
2      10 Days in a Madhouse      2015-11-11                 Drama   
3  10 Things I Hate About You      1999-03-31  Comedy,Drama,Romance   
4                10,000 B.C.      2008-03-07                   NaN   

   runtime_minutes     director_name             director_professions  \
0              NaN               NaN                              NaN   
1            103.0  Dan Trachtenberg  music_department,director,writer   
2            111.0     Timothy Hines    visual_effects,director,writer   
3             97.0        Gil Junger   producer,director,miscellaneous   
4              NaN               NaN                              NaN   

  director_birthYear director_deathYear  movie_averageRating  \
0                NaN               NaN                  NaN   
```

```
1                \N             alive                  7.2
2              1960             alive                  5.8
3              1954             alive                  7.3
4               NaN               NaN                  NaN

   movie_numerOfVotes  ...  Production budget $  Domestic gross $  \
0                 NaN  ...            7500000.0        32425665.0
1            333495.0  ...           15000000.0        72082999.0
2              2797.0  ...           12000000.0           14616.0
3            349513.0  ...           13000000.0        38177966.0
4                 NaN  ...          105000000.0        94784201.0

   Worldwide gross $    id  release_date                       movie  \
0         34439060.0  55.0    2009-07-17        (500) Days of Summer
1        108286422.0  54.0    2016-03-11         10 Cloverfield Lane
2            14616.0  48.0    2015-11-11       10 Days in a Madhouse
3         60413950.0  63.0    1999-03-31  10 Things I Hate About You
4        269065678.0  51.0    2008-03-07                10,000 B.C.

  production_budget  domestic_gross  worldwide_gross  year
0         7500000.0      32425665.0       34439060.0  2009
1         5000000.0      72082999.0      108286422.0  2016
2        12000000.0         14616.0          14616.0  2015
3        13000000.0      38177966.0       60413950.0  1999
4       105000000.0      94784201.0      269065678.0  2008

[5 rows x 21 columns]
```

```
C:\Users\erica\AppData\Local\Temp\ipykernel_31200\1593153321.py:4: FutureWarnin
g: A value is trying to be set on a copy of a DataFrame or Series through chaine
d assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work beca
use the intermediate object on which we are setting values always behaves as a c
opy.

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


  money_data['Production budget $'].fillna(money_data['production_budget'], inpl
ace=True)
C:\Users\erica\AppData\Local\Temp\ipykernel_31200\1593153321.py:5: FutureWarnin
g: A value is trying to be set on a copy of a DataFrame or Series through chaine
d assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work beca
use the intermediate object on which we are setting values always behaves as a c
opy.

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


  money_data['Domestic gross $'].fillna(money_data['domestic_gross'], inplace=Tr
ue)
C:\Users\erica\AppData\Local\Temp\ipykernel_31200\1593153321.py:6: FutureWarnin
g: A value is trying to be set on a copy of a DataFrame or Series through chaine
d assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work beca
use the intermediate object on which we are setting values always behaves as a c
```

```
opy.

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


  money_data['Worldwide gross $'].fillna(money_data['worldwide_gross'], inplace=
True)
C:\Users\erica\AppData\Local\Temp\ipykernel_31200\1593153321.py:7: FutureWarnin
g: A value is trying to be set on a copy of a DataFrame or Series through chaine
d assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work beca
use the intermediate object on which we are setting values always behaves as a c
opy.

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


  money_data['movie_title'].fillna(money_data['movie'], inplace=True)
C:\Users\erica\AppData\Local\Temp\ipykernel_31200\1593153321.py:10: FutureWarnin
g: A value is trying to be set on a copy of a DataFrame or Series through chaine
d assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work beca
use the intermediate object on which we are setting values always behaves as a c
opy.

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


  money_data['production_date'].fillna(money_data['release_date'], inplace=True)
```

In [17]:

```
#Seeing that our combined data is complete for date and financial data.
print(money_data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5937 entries, 0 to 5936
Data columns (total 21 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   movie_title           5937 non-null   object
 1   production_date       5937 non-null   datetime64[ns]
 2   genres                4528 non-null   object
 3   runtime_minutes       4528 non-null   float64
 4   director_name         4528 non-null   object
 5   director_professions  4528 non-null   object
 6   director_birthYear    4528 non-null   object
 7   director_deathYear    4528 non-null   object
 8   movie_averageRating   4528 non-null   float64
 9   movie_numerOfVotes    4528 non-null   float64
 10  approval_Index        4528 non-null   float64
 11  Production budget $   5937 non-null   float64
 12  Domestic gross $      5937 non-null   float64
 13  Worldwide gross $     5937 non-null   float64
 14  id                    5616 non-null   float64
 15  release date          5616 non-null   datetime64[ns]
```

```
 16  movie                 5616 non-null   object
 17  production_budget     5616 non-null   float64
 18  domestic_gross        5616 non-null   float64
 19  worldwide_gross       5616 non-null   float64
 20  year                  5937 non-null   int32
dtypes: datetime64[ns](2), float64(11), int32(1), object(7)
memory usage: 951.0+ KB
None
```

Next step: Derived financial data - costs and gross adjusted to year

Understanding that dollar amounts do not directly correlate to actual value, this adjustment will ensure that all financial metrics are comparable across different time periods.

Preparing to adjust the costs and gross revenue data for inflation by using the Consumer Price Index (CPI) from the inflation_data DataFrame. The output of inflation_data.head() shows the structure of your CPI dataset.

In [18]:
```
inflation_data.head()
```

Out[18]:

	Yearmon	CPI
0	01-01-1913	9.8
1	01-02-1913	9.8
2	01-03-1913	9.8
3	01-04-1913	9.8
4	01-05-1913	9.7

By selecting only December CPI values, we ensure that inflation adjustments are based on a consistent point in each year, reducing variability within the year and allowing that the year of production/release is not usually the primary year of earnings, so the latest CPI attached to the production year is more accurate.

In [19]:
```
#Convert Yearmon to datetime
inflation_data['Yearmon'] = pd.to_datetime(inflation_data['Yearmon'], errors='

# Filter the CPI values for December of each year.
# datetime incorrectly guessed 12 days in january per year rather than 1st day
CPI_by_year = inflation_data[inflation_data['Yearmon'].dt.day == 12]

#Create Year column
CPI_by_year['Year']=CPI_by_year['Yearmon'].dt.year

CPI_by_year.head()
```

```
C:\Users\erica\AppData\Local\Temp\ipykernel_31200\974917147.py:9: SettingWithCop
yWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  CPI_by_year['Year']=CPI_by_year['Yearmon'].dt.year
```

Out[19]:

	Yearmon	CPI	Year
11	1913-01-12	10.0	1913
23	1914-01-12	10.1	1914
35	1915-01-12	10.3	1915
47	1916-01-12	11.6	1916
59	1917-01-12	13.7	1917

In [20]:

```
# Base year for inflation adjustment (choose the latest year in your dataset)
base_year = CPI_by_year['Year'].max()
base_cpi = CPI_by_year[CPI_by_year['Year'] == base_year]['CPI'].values[0]

# Merge inflation data with movie data
money_data = pd.merge(money_data, CPI_by_year, how='left', left_on='year', rig

# Adjust values
money_data['adjusted_production_budget'] = (money_data['Production budget $']
money_data['adjusted_domestic_gross'] = (money_data['Domestic gross $'] * base
money_data['adjusted_worldwide_gross'] = (money_data['Worldwide gross $'] * ba

# Calculate ROI
money_data['ROI'] = (np.maximum(money_data['adjusted_worldwide_gross'], money_


# Drop unnecessary columns from the merge
money_data.drop(columns=['Year', 'CPI'], inplace=True)

# Preview adjusted data
print(money_data.head())
```

```
                movie_title production_date                genres  \
0       (500) Days of Summer      2009-07-17                   NaN
1        10 Cloverfield Lane      2016-01-04  Drama,Horror,Mystery
2      10 Days in a Madhouse      2015-11-11                 Drama
3  10 Things I Hate About You      1999-03-31  Comedy,Drama,Romance
4                10,000 B.C.      2008-03-07                   NaN

   runtime_minutes     director_name             director_professions  \
0              NaN               NaN                              NaN
1            103.0  Dan Trachtenberg  music_department,director,writer
2            111.0     Timothy Hines    visual_effects,director,writer
3             97.0        Gil Junger   producer,director,miscellaneous
4              NaN               NaN                              NaN

  director_birthYear director_deathYear  movie_averageRating  \
0                NaN                NaN                  NaN
1                 \N              alive                  7.2
2               1960              alive                  5.8
```

```
3              1954              alive                  7.3
4               NaN                NaN                  NaN

   movie_numerOfVotes  ...                        movie  production_budget  \
0                 NaN  ...          (500) Days of Summer          7500000.0
1            333495.0  ...           10 Cloverfield Lane          5000000.0
2              2797.0  ...         10 Days in a Madhouse         12000000.0
3            349513.0  ...  10 Things I Hate About You         13000000.0
4                 NaN  ...                  10,000 B.C.        105000000.0

   domestic_gross  worldwide_gross  year    Yearmon  \
0      32425665.0       34439060.0  2009 2009-01-12
1      72082999.0      108286422.0  2016 2016-01-12
2         14616.0          14616.0  2015 2015-01-12
3      38177966.0       60413950.0  1999 1999-01-12
4      94784201.0      269065678.0  2008 2008-01-12

  adjusted_production_budget  adjusted_domestic_gross  \
0               1.016073e+07             4.392913e+07
1               1.817655e+07             8.734800e+07
2               1.484291e+07             1.807867e+04
3               2.259822e+07             6.636569e+07
4               1.461214e+08             1.319047e+08

   adjusted_worldwide_gross       ROI
0              4.665681e+07  3.591875
1              1.312182e+08  6.219095
2              1.807867e+04 -0.998782
3              1.050190e+08  3.647227
4              3.744404e+08  1.562530

[5 rows x 26 columns]
```

In [21]:
```
print(imdb_data.head())
```

```
    movie_id                    primary_title              original_title  \
0  tt0063540                        Sunghursh                 Sunghursh
1  tt0066787  One Day Before the Rainy Season             Ashad Ka Ek Din
2  tt0069049       The Other Side of the Wind  The Other Side of the Wind
3  tt0069204                  Sabse Bada Sukh             Sabse Bada Sukh
4  tt0100275         The Wandering Soap Opera       La Telenovela Errante

   start_year  runtime_minutes                genres
0        2013            175.0    Action,Crime,Drama
1        2019            114.0       Biography,Drama
2        2018            122.0                 Drama
3        2018              NaN          Comedy,Drama
4        2017             80.0  Comedy,Drama,Fantasy
```

In [22]:
```
print(imdb_data.columns)
```

```
Index(['movie_id', 'primary_title', 'original_title', 'start_year',
       'runtime_minutes', 'genres'],
      dtype='object')
```

In [23]:
```
imdb_data['movie_title'] = imdb_data['primary_title'].str.strip()

# Merge the datasets using movie_title
combined_data = pd.merge(imdb_data, money_data, on='movie_title')
```

```
# Preview the merged dataset
print(combined_data.head())
```

```
   movie_id  primary_title original_title  start_year  runtime_minutes_x  \
0  tt0249516     Foodfight!     Foodfight!        2012               91.0
1  tt0293429  Mortal Kombat  Mortal Kombat        2021                NaN
2  tt0293429  Mortal Kombat  Mortal Kombat        2021                NaN
3  tt0326592  The Overnight  The Overnight        2010               88.0
4  tt0337692    On the Road    On the Road        2012              124.0

                  genres_x    movie_title production_date  \
0   Action,Animation,Comedy     Foodfight!      2012-12-31
1  Action,Adventure,Fantasy  Mortal Kombat      2021-04-08
2  Action,Adventure,Fantasy  Mortal Kombat      1995-08-18
3                      None  The Overnight      2015-06-19
4   Adventure,Drama,Romance    On the Road      2012-05-23

                  genres_y  runtime_minutes_y  ...          movie  \
0                      NaN                NaN  ...     Foodfight!
1  Action,Adventure,Fantasy              110.0  ...  Mortal Kombat
2  Action,Adventure,Fantasy              101.0  ...  Mortal Kombat
3            Comedy,Mystery               79.0  ...  The Overnight
4   Adventure,Drama,Romance              124.0  ...    On the Road

  production_budget domestic_gross worldwide_gross  year    Yearmon  \
0        45000000.0            0.0          73706.0  2012 2012-01-12
1        20000000.0     70433227.0     122133227.0  2021 2021-01-12
2        20000000.0     70433227.0     122133227.0  1995 1995-01-12
3          200000.0      1109808.0       1165996.0  2015 2015-01-12
4        25000000.0       720828.0       9313302.0  2012 2012-01-12

   adjusted_production_budget  adjusted_domestic_gross  \
0                5.733947e+07             0.000000e+00
1                5.771408e+07             4.441469e+07
2                3.811857e+07             1.342407e+08
3                2.473819e+05             1.372732e+06
4                3.185526e+07             9.184866e+05

   adjusted_worldwide_gross       ROI
0              9.391696e+04 -0.998362
1              8.762913e+07  0.518332
2              2.327772e+08  5.106661
3              1.442231e+06  4.829980
4              1.186711e+07 -0.627468

[5 rows x 32 columns]
```

In [24]:

```
# Drop unnecessary or duplicate columns
combined_data_cleaned = combined_data.drop(columns=['runtime_minutes_x', 'movi

# Preview the cleaned dataset
print(combined_data_cleaned.head())
```

```
   movie_id  primary_title original_title  start_year  \
0  tt0249516     Foodfight!     Foodfight!        2012
1  tt0293429  Mortal Kombat  Mortal Kombat        2021
2  tt0293429  Mortal Kombat  Mortal Kombat        2021
```

```
3  tt0326592  The Overnight  The Overnight        2010
4  tt0337692    On the Road    On the Road        2012

                   genres_x    movie_title  runtime_minutes_y  \
0   Action,Animation,Comedy     Foodfight!                NaN
1  Action,Adventure,Fantasy  Mortal Kombat              110.0
2  Action,Adventure,Fantasy  Mortal Kombat              101.0
3                      None  The Overnight               79.0
4   Adventure,Drama,Romance    On the Road              124.0

        director_name      director_professions director_birthYear  ...  \
0                 NaN                       NaN               NaN  ...
1       Simon McQuoid         director,producer                \N  ...
2  Paul W.S. Anderson  writer,director,producer              1965  ...
3       Patrick Brice     director,writer,actor              1983  ...
4       Walter Salles  director,producer,writer              1956  ...

  release_date  production_budget  domestic_gross  worldwide_gross  year  \
0   2012-12-31         45000000.0             0.0          73706.0  2012
1   1995-08-18         20000000.0      70433227.0      122133227.0  2021
2   1995-08-18         20000000.0      70433227.0      122133227.0  1995
3   2015-06-19           200000.0       1109808.0        1165996.0  2015
4   2013-03-22         25000000.0        720828.0        9313302.0  2012

     Yearmon  adjusted_production_budget  adjusted_domestic_gross  \
0 2012-01-12                5.733947e+07             0.000000e+00
1 2021-01-12                5.771408e+07             4.441469e+07
2 1995-01-12                3.811857e+07             1.342407e+08
3 2015-01-12                2.473819e+05             1.372732e+06
4 2012-01-12                3.185526e+07             9.184866e+05

  adjusted_worldwide_gross       ROI
0             9.391696e+04 -0.998362
1             8.762913e+07  0.518332
2             2.327772e+08  5.106661
3             1.442231e+06  4.829980
4             1.186711e+07 -0.627468

[5 rows x 28 columns]
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4341 entries, 0 to 4340
Data columns (total 28 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   movie_id                   4341 non-null   object
 1   primary_title              4341 non-null   object
 2   original_title             4340 non-null   object
 3   start_year                 4341 non-null   int64
 4   genres_x                   4265 non-null   object
 5   movie_title                4341 non-null   object
 6   runtime_minutes_y          3772 non-null   float64
 7   director_name              3772 non-null   object
 8   director_professions       3772 non-null   object
 9   director_birthYear         3772 non-null   object
 10  director_deathYear         3772 non-null   object
 11  movie_averageRating        3772 non-null   float64
 12  movie_numerOfVotes         3772 non-null   float64
```

```
 13  approval_Index               3772 non-null   float64
 14  Production budget $          4341 non-null   float64
 15  Domestic gross $             4341 non-null   float64
 16  Worldwide gross $            4341 non-null   float64
 17  id                           3995 non-null   float64
 18  release_date                 3995 non-null   datetime64[ns]
 19  production_budget            3995 non-null   float64
 20  domestic_gross               3995 non-null   float64
 21  worldwide_gross              3995 non-null   float64
 22  year                         4341 non-null   int32
 23  Yearmon                      4341 non-null   datetime64[ns]
 24  adjusted_production_budget   4341 non-null   float64
 25  adjusted_domestic_gross      4341 non-null   float64
 26  adjusted_worldwide_gross     4341 non-null   float64
 27  ROI                          4341 non-null   float64
dtypes: datetime64[ns](2), float64(15), int32(1), int64(1), object(9)
memory usage: 932.8+ KB
```

In [26]:
```
imdb_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 146144 entries, 0 to 146143
Data columns (total 7 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
 6   movie_title      146144 non-null  object
dtypes: float64(1), int64(1), object(5)
memory usage: 7.8+ MB
```

In [27]:
```
df_clean_movies = combined_data_cleaned[['movie_id','movie_title','genres_x','
                                         'adjusted_production_budget','adju
                                         'movie_averageRating','movie_numer

df_clean_movies.head()
```

Out[27]:

	movie_id	movie_title	genres_x	year	director_name	director_birtl
0	tt0249516	Foodfight!	Action,Animation,Comedy	2012	NaN	
1	tt0293429	Mortal Kombat	Action,Adventure,Fantasy	2021	Simon McQuoid	
2	tt0293429	Mortal Kombat	Action,Adventure,Fantasy	1995	Paul W.S. Anderson	
3	tt0326592	The Overnight	None	2015	Patrick Brice	
4	tt0337692	On the Road	Adventure,Drama,Romance	2012	Walter Salles	

In [28]:
```
df_clean_movies['director_birthYear'] = pd.to_numeric(df_clean_movies['directo
available_directors = df_clean_movies[
    (df_clean_movies['director_birthYear'] > 1952) |
    (df_clean_movies['director_birthYear'].isna()) &
    (df_clean_movies['director_deathYear']=='alive')
]
director_roi = available_directors.groupby('director_name')['ROI'].mean().sort

print(director_roi.head())
```

```
     director_name        ROI
0  Aneesh Chaganty  84.950053
1  Nathaniel Davis  68.609409
2     Tod Williams  58.170677
3     Damien Leone  50.165976
4   Bradley Parker  41.411721
```

```
C:\Users\erica\AppData\Local\Temp\ipykernel_31200\2640403936.py:1: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_clean_movies['director_birthYear'] = pd.to_numeric(df_clean_movies['directo
r_birthYear'], errors='coerce')
```

In [29]:
```
print(director_roi.head(50))
```

```
         director_name        ROI
0      Aneesh Chaganty  84.950053
1      Nathaniel Davis  68.609409
2         Tod Williams  58.170677
3         Damien Leone  50.165976
4       Bradley Parker  41.411721
5        Nitesh Tiwari  31.120996
6        Barry Jenkins  28.451563
7       William Lustig  27.571429
8    David F. Sandberg  23.777532
9   William Brent Bell  22.735758
10        Jordan Peele  21.215722
11      Stephen Daldry  20.850594
12         Jeff Wadlow  20.568114
13    John R. Leonetti  19.736494
14        Stiles White  19.660126
15    Oyefunke Fayoyin  18.735563
16      Michael Madsen  18.735563
17      John Krasinski  18.698628
18       Chuck Russell  18.534452
19        Daniel Stamm  18.496472
20     Takashi Shimizu  17.728112
21      James DeMonaco  17.222144
22        Adam Robitel  16.288277
23       Alex Kendrick  15.708095
24         Corin Hardy  15.517802
25       Stephen Susco  15.434588
26    Scott Derrickson  15.379035
27             Ti West  14.131549
28        Steve Barron  13.962963
```

28	Steve Barron	13.962963
29	Drake Doremus	13.913600
30	Adrienne Shelly	13.811455
31	Tatsuya Nagamine	13.440912
32	Sam Taylor-Johnson	13.269874
33	Leigh Whannell	13.178853
34	David Lowery	12.956411
35	Peter Farrelly	12.912827
36	Michael Chaves	12.681527
37	Justin Baldoni	12.222030
38	Josh Boone	12.131515
39	Todd Phillips	11.999259
40	Benh Zeitlin	11.925073
41	Christopher Landon	11.851851
42	Ketan Pendse	11.708808
43	Parker Finn	11.708808
44	Garth Davis	11.426492
45	M. Night Shyamalan	11.345449
46	Chris Renaud	10.804033
47	Joel Edgerton	10.398759
48	Michael Showalter	10.373735
49	James Cameron	10.336491

In [30]:
```
available_directors['genres_x'].fillna('Unknown', inplace=True)

director_stats = available_directors.groupby('director_name').agg({
    'genres_x': lambda x: ', '.join(set(x)),  # Get unique genres and join the
    'adjusted_production_budget': 'mean'  # Calculate the average adjusted pro
}).reset_index()
director_full_stats = pd.merge(director_roi, director_stats, on='director_name
```

```
C:\Users\erica\AppData\Local\Temp\ipykernel_31200\3379184613.py:1: FutureWarnin
g: A value is trying to be set on a copy of a DataFrame or Series through chaine
d assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work beca
use the intermediate object on which we are setting values always behaves as a c
opy.

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


  available_directors['genres_x'].fillna('Unknown', inplace=True)
C:\Users\erica\AppData\Local\Temp\ipykernel_31200\3379184613.py:1: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  available_directors['genres_x'].fillna('Unknown', inplace=True)
```

In [31]:
```
director_full_stats['budget_category'] = pd.cut(director_full_stats['adjusted_
                                                bins=[0, 5e6, 2e7, np.inf], #b
                                                labels=['Low Budget', 'Medium
budget_analysis = director_full_stats.groupby('budget_category')['ROI'].mean()
budget_analysis
```

```
C:\Users\erica\AppData\Local\Temp\ipykernel_31200\280541130.py:4: FutureWarning:
The default of observed=False is deprecated and will be changed to True in a fut
ure version of pandas. Pass observed=False to retain current behavior or observe
d=True to adopt the future default and silence this warning.
  budget_analysis = director_full_stats.groupby('budget_category')['ROI'].mean
().reset_index()
```

Out[31]:

	budget_category	ROI
0	Low Budget	4.307052
1	Medium Budget	3.121742
2	High Budget	1.828966

In [32]:

```
movie_counts = available_directors.groupby('director_name').size().reset_index

director_full_stats = pd.merge(director_full_stats, movie_counts, on='director

target_genres = ['Horror', 'Musical', 'Family']
filtered_data = director_full_stats[
    (director_full_stats['genres_x'].str.contains('|'.join(target_genres))) &
    (director_full_stats['budget_category'] != 'High Budget')
]


top_directors = filtered_data[['director_name', 'ROI', 'genres_x', 'adjusted_p
```

In [33]:

```
df_cleaned_exploded = df_clean_movies.copy()
df_cleaned_exploded['genres_x'] = df_cleaned_exploded['genres_x'].str.split(',
df_cleaned_exploded = df_cleaned_exploded.explode('genres_x')
df_cleaned_exploded['genres_x'].replace({None:'Unknown','None':'Unknown'},inpl
```

```
C:\Users\erica\AppData\Local\Temp\ipykernel_31200\1100441034.py:4: FutureWarnin
g: A value is trying to be set on a copy of a DataFrame or Series through chaine
d assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work beca
use the intermediate object on which we are setting values always behaves as a c
opy.

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


  df_cleaned_exploded['genres_x'].replace({None:'Unknown','None':'Unknown'},inpl
ace=True)
```

In [34]:

```
# Group by genres and calculate the mean ROI for each genre
avg_roi_by_genre = df_cleaned_exploded.groupby('genres_x')['ROI'].mean().reset

print(avg_roi_by_genre)
```

	genres_x	ROI
13	Musical	12.002641
11	Horror	9.717250
8	Family	7.405298
9	Fantasy	6.352203
2	Animation	6.129440
6	Documentary	6.099865
10	History	6.039342
14	Mystery	6.032886
19	Thriller	5.728399
3	Biography	4.923582
12	Music	4.365667
16	Romance	4.269131
21	War	3.899068
7	Drama	3.541800
4	Comedy	3.284435
18	Sport	2.719585
20	Unknown	2.695256
0	Action	2.655679
1	Adventure	2.540650
17	Sci-Fi	2.291887
5	Crime	1.522198
15	News	0.496427
22	Western	0.362860

In [35]:
```
genres = df_cleaned_exploded['genres_x'].unique()


roi_data = [df_cleaned_exploded[df_cleaned_exploded['genres_x'] == genre]['ROI
roi_data = [data for data in roi_data if len(data) > 0]  # Keep only non-empty
```

In [36]:
```
#I wonder if number of principals has any bearing on ROI
principals_count = pd.read_sql('''
                        SELECT movie_id, COUNT(person_id) AS principal_count
                        FROM principals
                        GROUP BY movie_id
                        ''',con)
principals_count.head()
```

Out[36]:

	movie_id	principal_count
0	tt0063540	10
1	tt0066787	7
2	tt0069049	10
3	tt0069204	10
4	tt0100275	10

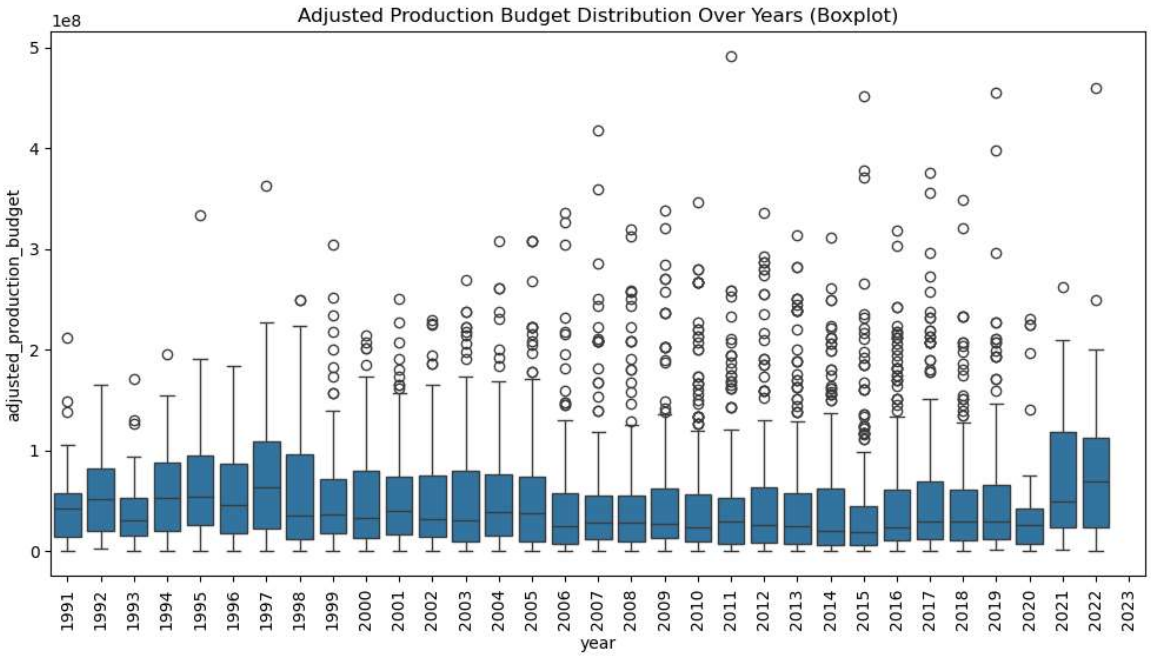
In [37]:
```
#Adding number of principals to our dataframe
df_clean_movies = df_clean_movies.merge(principals_count, on='movie_id',how='l
df_clean_movies['principal_count'] = df_clean_movies['principal_count'].fillna
```

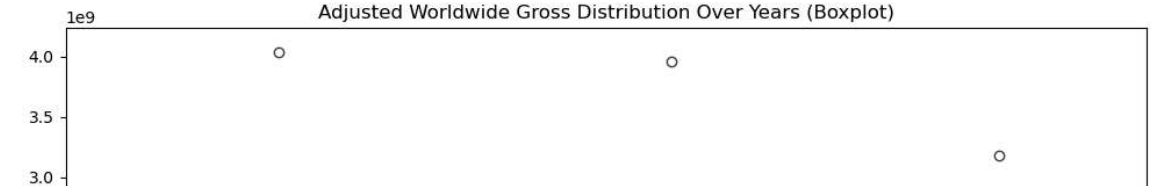
Data Analysis

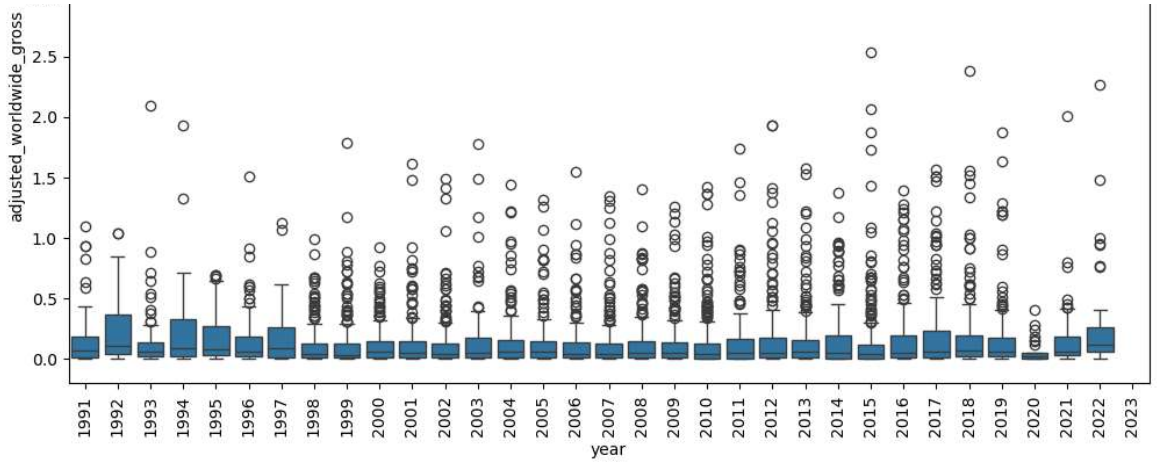
Overview:

We begin by exploring and visualizing the adjusted financial metrics to gain initial insights into movie performance. From there, we perform a more in-depth analysis by examining key features such as budget categories, genres, director performance, and principal counts to better understand their influence on ROI and profitability, leading to several actionable recommendations to maximize expected ROI while controlling risk.

In [38]:
```
#Visualizing recent budget and gross earnings (since 1990)

filtered_data = money_data[money_data['year'] > 1990]

# Distribution by years with boxplot
plt.figure(figsize=(12, 6))
sns.boxplot(x='year', y='adjusted_production_budget', data=filtered_data)
plt.xticks(rotation=90)  # X eksenindeki yılları net görmek için döndürme
plt.title('Adjusted Production Budget Distribution Over Years (Boxplot)')
plt.show()

# Likewise for Worldwide Gross
plt.figure(figsize=(12, 6))
sns.boxplot(x='year', y='adjusted_worldwide_gross', data=filtered_data)
plt.xticks(rotation=90)
plt.title('Adjusted Worldwide Gross Distribution Over Years (Boxplot)')
plt.show()
```

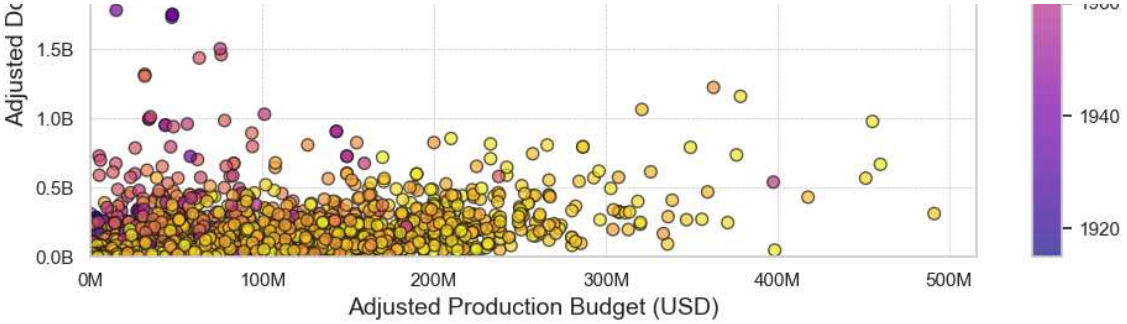






In [39]:

```
# Set plot style
sns.set_theme(style="whitegrid")

# Scatter plot for Adjusted Domestic Gross vs Adjusted Production Budget
plt.figure(figsize=(10, 6))

# Define the scatter plot with better visual parameters
scatter = plt.scatter(money_data['adjusted_production_budget'],
                      money_data['adjusted_domestic_gross'],
                      c=money_data['year'], cmap='plasma', s=50, alpha=0.7, ed

# Add a color bar to indicate the years
cbar = plt.colorbar(scatter)
cbar.set_label('Year')

# Set titles and labels
plt.title('Adjusted Domestic Gross vs Adjusted Production Budget (Colored by Y
plt.xlabel('Adjusted Production Budget (USD)', fontsize=14)
plt.ylabel('Adjusted Domestic Gross (USD)', fontsize=14)

plt.xlim(left= 0)
plt.ylim(bottom= 0)

plt.grid(True, which='both', linestyle='--', linewidth=0.5)
#Remove scientific notation

plt.gca().xaxis.set_major_formatter(mtick.FuncFormatter(lambda x, _: f'{x * 1e
plt.gca().yaxis.set_major_formatter(mtick.FuncFormatter(lambda y, _: f'{y * 1e
plt.tight_layout()
plt.show()
```

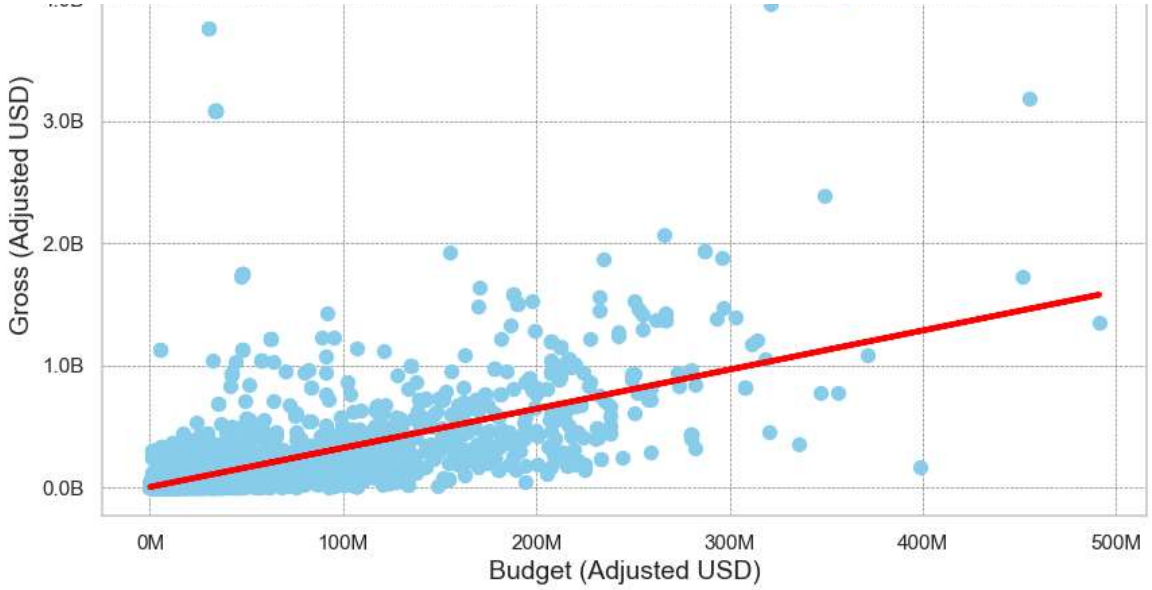


These exploratory visualizations of the data emphasize the significant variance in gross earnings and, consequently, in return on investment (ROI). The color scale indicates that more recent films (yellow and light orange) tend to have higher production budgets than older films (purple), reflecting expansion and capitalization of the industry.

In [40]:

```
# Select the independent (X) and dependent (y) variables
X = df_clean_movies['adjusted_production_budget'].dropna().values.reshape(-1,
y = np.maximum(df_clean_movies['adjusted_worldwide_gross'], df_clean_movies['a

# Fit the linear regression model
model = LinearRegression()
model.fit(X, y)

# Print the coefficients
print(f"Intercept: {model.intercept_}")
print(f"Slope: {model.coef_[0]}")

# Predict values using the model
y_pred = model.predict(X)
r2 = r2_score(y, y_pred)
print(f"R-squared: {r2}")

# Plot the regression line and data points
plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='skyblue', s=50, label='Data Points')
plt.plot(X, y_pred, color='red', linewidth=3, label='Regression Line')
plt.xlabel('Budget (Adjusted USD)', fontsize=14)
plt.ylabel('Gross (Adjusted USD)', fontsize=14)
plt.title('Linear Regression: Budget vs Gross', fontsize=16, fontweight='bold'

# Remove scientific notation from axes
plt.gca().xaxis.set_major_formatter(mtick.FuncFormatter(lambda x, _: '{:,.0f}M
plt.gca().yaxis.set_major_formatter(mtick.FuncFormatter(lambda y, _: '{:,.1f}B

plt.legend(loc='upper right', fontsize=12)
plt.grid(color='gray', linestyle='--', linewidth=0.5)
plt.show()
```

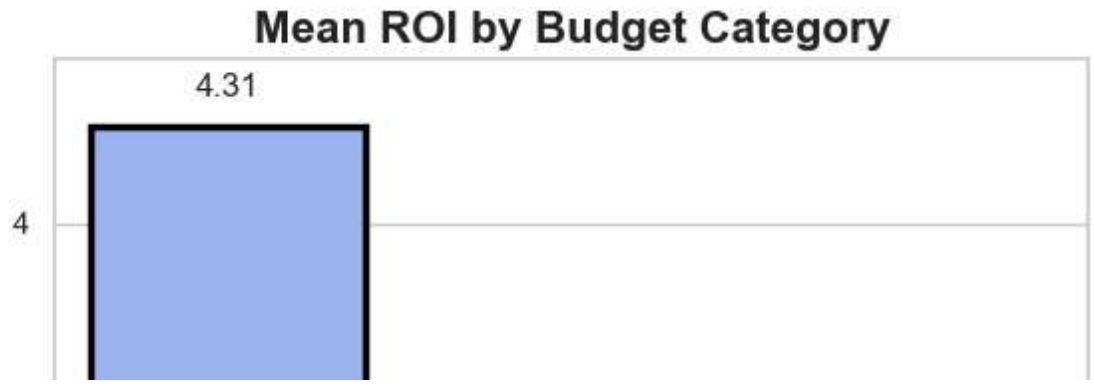
```
Intercept: 5144797.991259277
Slope: 3.2027515445383683
R-squared: 0.30375696764169346
```

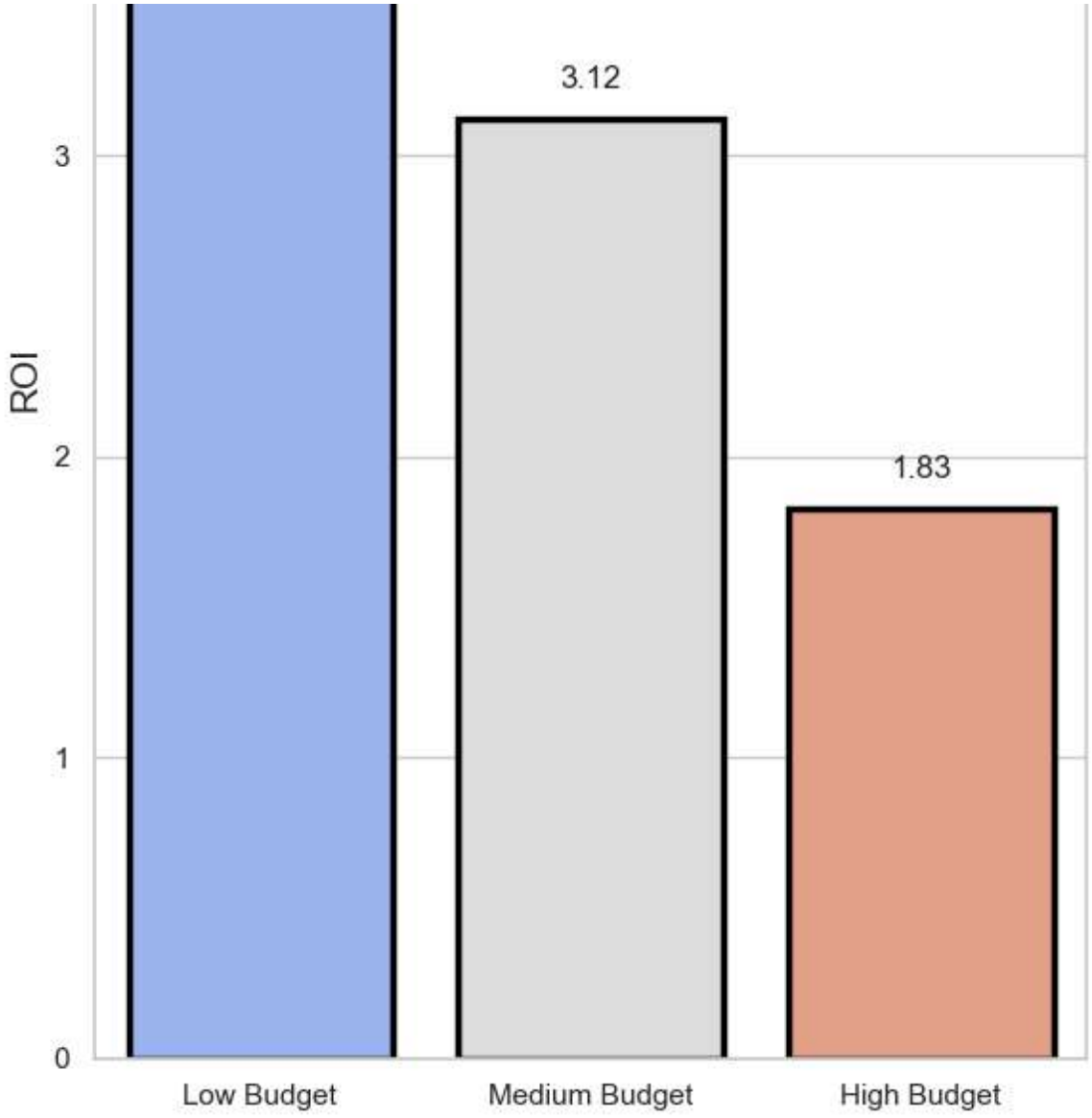


In [41]:

```
# Create the bar plot
plt.figure(figsize=(6, 8))
sns.barplot(x='budget_category', y='ROI', data=budget_analysis, palette='coolw

# Add labels and title
plt.title('Mean ROI by Budget Category', fontsize=16, weight='bold')
plt.ylabel('ROI', fontsize=14)
plt.xlabel('', fontsize=14)

# Add data labels on top of bars
for i in range(len(budget_analysis)):
    plt.text(i, budget_analysis['ROI'][i] + 0.1, round(budget_analysis['ROI'][
             ha='center', fontsize=12)

# Final adjustments to layout and display
plt.tight_layout()
plt.show()
```

```
C:\Users\erica\AppData\Local\Temp\ipykernel_31200\3727254661.py:3: FutureWarnin
g:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same eff
ect.

  sns.barplot(x='budget_category', y='ROI', data=budget_analysis, palette='coolw
arm', linewidth=2.5, edgecolor='black')
```





In [42]:
```
budget_analysis
```

Out[42]:

	budget_category	ROI
0	Low Budget	4.307052
1	Medium Budget	3.121742
2	High Budget	1.828966

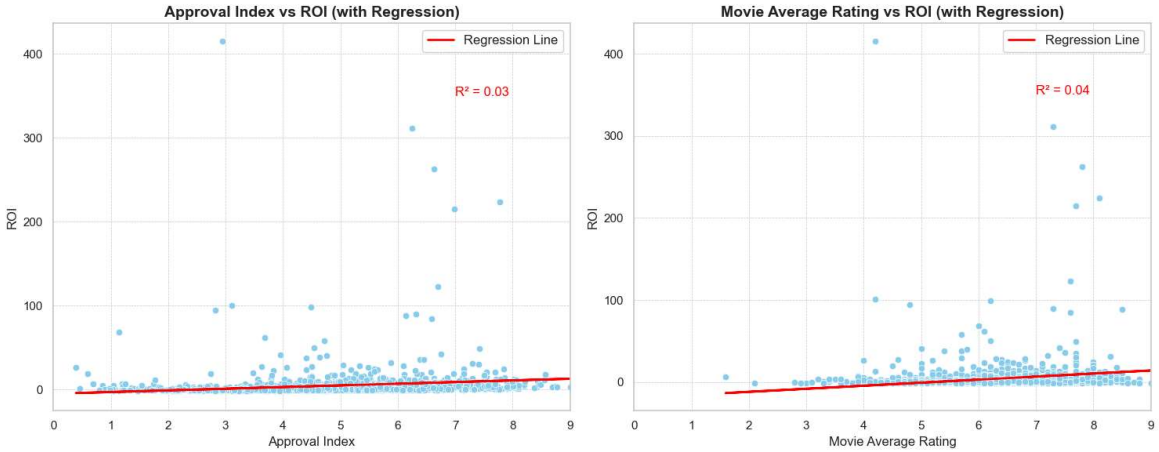
Given the noisy data, the first logical question is: "What is the relationship between budget and gross revenue?" From the model, we observe that for each budget dollar spent, there is an expected gross earning of $3.19, with budget explaining 30% of the variance. However, when analyzing budget categories, we notice that high-budget films have a diminished ROI compared to low-to-medium budget films. So increasing the budget within reasonable limits would be an effective strategy for maximizing revenue, but managing risk exposure is key - it's more worthwhile to be able to make 2 mid-budget films than to overreach for a higher budget film.

In [43]:

```
df_filtered = df_clean_movies[['approval_Index', 'ROI']].dropna()
X_approval = df_filtered['approval_Index'].values.reshape(-1, 1)
y = df_filtered['ROI'].values

# Linear regression model for Approval Index
model_approval = LinearRegression()
model_approval.fit(X_approval, y)
r2_approval = model_approval.score(X_approval, y)

# Filter data for Movie Average Rating vs ROI
df_filtered_rating = df_clean_movies[['movie_averageRating', 'ROI']].dropna()
X_rating = df_filtered_rating['movie_averageRating'].values.reshape(-1, 1)
y_rating = df_filtered_rating['ROI'].values

# Linear regression model for Movie Average Rating
model_rating = LinearRegression()
model_rating.fit(X_rating, y_rating)
r2_rating = model_rating.score(X_rating, y_rating)

# Print intercepts, coefficients, and R-squared values
print(f"Approval Index: Intercept = {model_approval.intercept_}, Coeff = {mode
print(f"Movie Average Rating: Intercept = {model_rating.intercept_}, Coeff = {

# Visualize the regression lines
plt.figure(figsize=(15, 6))

# Define the common x-axis limits for both plots
x_limits = (0, 9)  # Adjust these limits according to your data

# Subplot for Approval Index vs ROI
plt.subplot(1, 2, 1)
sns.scatterplot(data=df_clean_movies, x='approval_Index', y='ROI', color='skyb
plt.plot(X_approval, model_approval.predict(X_approval), color='red', label='R
plt.title('Approval Index vs ROI (with Regression)', fontsize=14, weight='bold
plt.xlabel('Approval Index', fontsize=12)
plt.ylabel('ROI', fontsize=12)
plt.grid(True, linestyle='--', linewidth=0.5)
plt.text(7, 350, f'R² = {r2_approval:.2f}', fontsize=12, color='red')
plt.xlim(x_limits)  # Set the same x-axis limits for consistency
plt.legend(loc='upper right', fontsize=12)

# Subplot for Movie Average Rating vs ROI
plt.subplot(1, 2, 2)
sns.scatterplot(data=df_clean_movies, x='movie_averageRating', y='ROI', color=
plt.plot(X_rating, model_rating.predict(X_rating), color='red', label='Regress
plt.title('Movie Average Rating vs ROI (with Regression)', fontsize=14, weight
plt.xlabel('Movie Average Rating', fontsize=12)
plt.ylabel('ROI', fontsize=12)
plt.grid(True, linestyle='--', linewidth=0.5)
plt.text(7, 350, f'R² = {r2_rating:.2f}', fontsize=12, color='red')
plt.xlim(x_limits)  # Set the same x-axis limits for consistency
plt.legend(loc='upper right', fontsize=12)

# Tight layout to avoid overlapping of subplots
plt.tight_layout()
plt.show()
```

```
Approval Index: Intercept = -4.7835400463077, Coeff = 1.982211505801775, R-squar
```

```
ed = 0.025831282002065747
Movie Average Rating: Intercept = -19.12960998077648, Coeff = 3.714676342737778,
R-squared = 0.04032053004430103
```



Both the approval index and movie average rating have very weak positive correlations with ROI, with R-squared values of 2.6% for Approval and 4% for Ratings. The relationships are small but present, as indicated by the positive slopes of both regression lines. However, the high dispersion of the data (especially in the ROI outliers) suggests other factors may have a stronger influence on ROI. In general it is recommended to ignore ratings as an avenue to ROI.

In [44]:
```
# Run the ANOVA test on genre
genre_anova_result = stats.f_oneway(*roi_data)

# Display the results
print(f'ANOVA F-statistic: {genre_anova_result.statistic}, p-value: {genre_ano
```

```
ANOVA F-statistic: 5.5543155168418075, p-value: 9.317557608280883e-16
```

Interpretation: Since the p-value is less than 0.05, we can reject the null hypothesis. This means that there are significant differences in the average ROI among the genres.

A Tukey HSD test can help us with post-hoc analysis of these differences.

In [45]:
```
df_cleaned_exploded = df_cleaned_exploded.dropna(subset=['ROI', 'genres_x'])
```

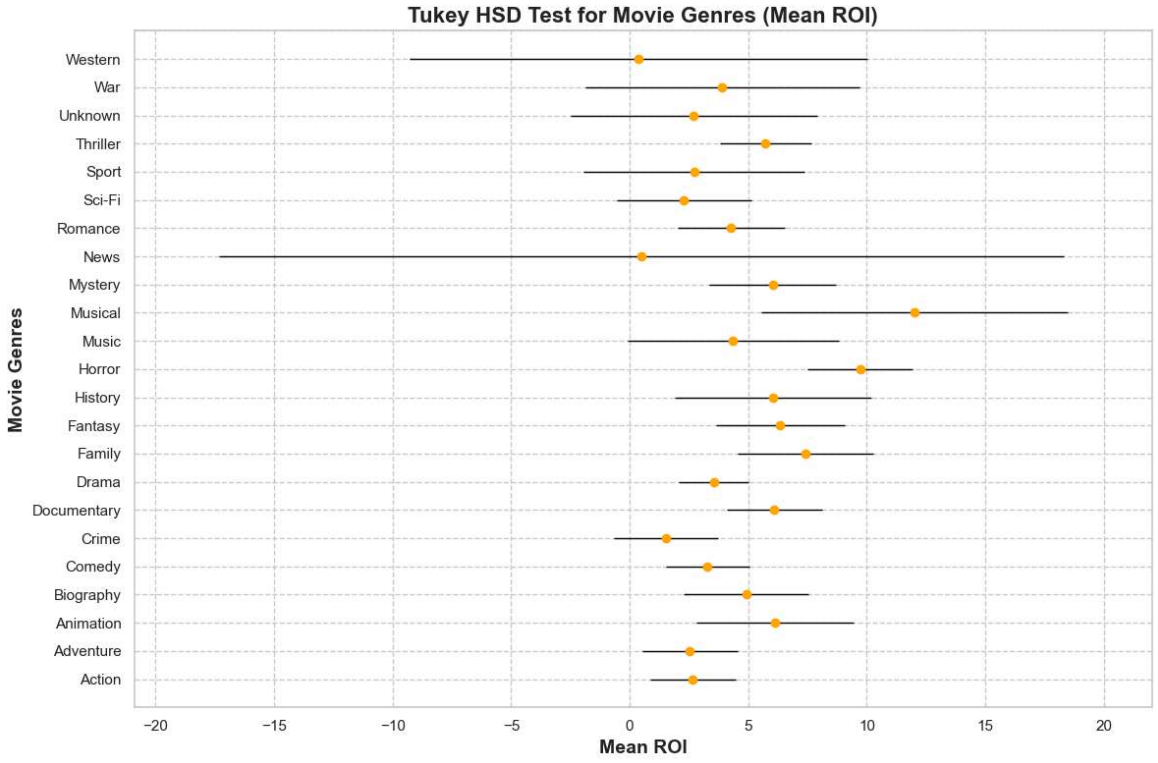
In [46]:
```
genre_tk_hsd = pairwise_tukeyhsd(df_cleaned_exploded['ROI'], df_cleaned_explod
```

```
c:\Users\erica\anaconda3\Lib\site-packages\scipy\integrate\_quadpack_py.py:1272:
IntegrationWarning: The integral is probably divergent, or slowly convergent.
  quad_r = quad(f, low, high, args=args, full_output=self.full_output,
```

In [47]:
```
genre_tk_hsd = pairwise_tukeyhsd(df_cleaned_exploded['ROI'], df_cleaned_explod


plt.figure(figsize=(12, 8))
```

```
genre_tk_hsd.plot_simultaneous(figsize=(12, 8))


lines = plt.gca().lines
for line in lines:
    line.set_color('orange')
    line.set_linewidth(2)


points = plt.gca().collections[0]
points.set_edgecolor('black')
points.set_linewidth(1)


plt.title('Tukey HSD Test for Movie Genres (Mean ROI)', fontsize=16, weight='b
plt.xlabel('Mean ROI', fontsize=14, weight='bold')  # Adjusted x label
plt.ylabel('Movie Genres', fontsize=14, weight='bold')

plt.grid(True, linestyle='--', linewidth=1)
plt.tight_layout()


plt.show()
```

```
<Figure size 1200x800 with 0 Axes>
```

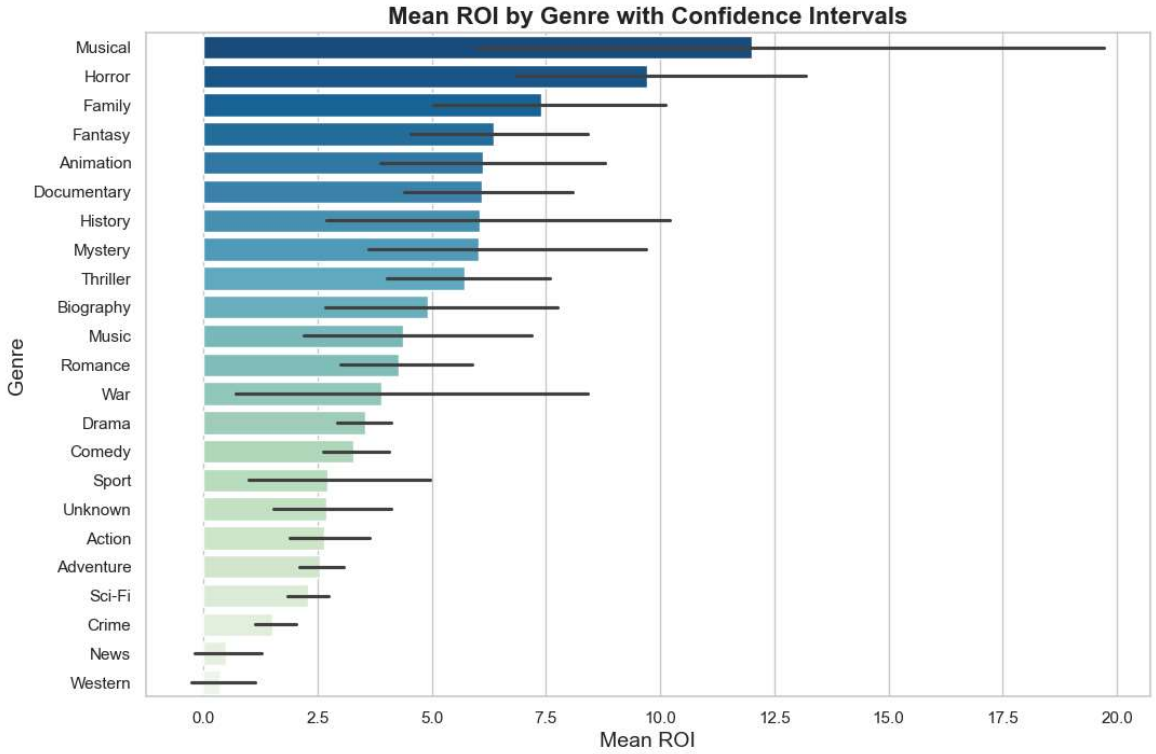


In [48]:

```
plt.figure(figsize=(12, 8))
sns.barplot(data=df_cleaned_exploded,x='ROI',y='genres_x',order=avg_roi_by_gen

plt.title('Mean ROI by Genre with Confidence Intervals', fontsize=16, weight='
plt.ylabel('Genre', fontsize=14)
plt.xlabel('Mean ROI', fontsize=14)
```

```
C:\Users\erica\AppData\Local\Temp\ipykernel_31200\1749268008.py:2: FutureWarnin
g:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same eff
ect.

  sns.barplot(data=df_cleaned_exploded,x='ROI',y='genres_x',order=avg_roi_by_gen
re['genres_x'], orient='h', palette='GnBu_r')
```

Out[48]: Text(0.5, 0, 'Mean ROI')



ANOVA test across genres allowed us to determine that there WAS a significant difference in mean ROI between genres. Following up with a post-hoc Tukey HSD test allowed us to describe which categories performed significantly better or worse than others. This bar plot now illustrates best the mean ROI differences between various movie genres, with error bars to show the overall confidence interval. This helps pinpoint which genres tend to perform better or worse in terms of ROI. For example, our data shows a good degree of confidence in the mean ROI for horror films, our second-highest performing category, but some high-performing genres, like musicals, are not as well-represented in the data, so they exhibit wider confidence intervals. Even so, musicals still rank among the genres with the strongest average ROIs and may still also be a good option. Family represents the third tier with smallest confidence interval of the top three and strong mean ROI as another emminently reasonable choice depending on other production factors.

In [49]:

```
# ANOVA test setup for count of principals in a movie
principal_count_roi = [df_clean_movies[df_clean_movies['principal_count'] == c

# Run the ANOVA test
pcr_anova_result = stats.f_oneway(*principal_count_roi)
```

```
# Display the results
print(f'ANOVA F-statistic: {pcr_anova_result.statistic}, p-value: {pcr_anova_r
```

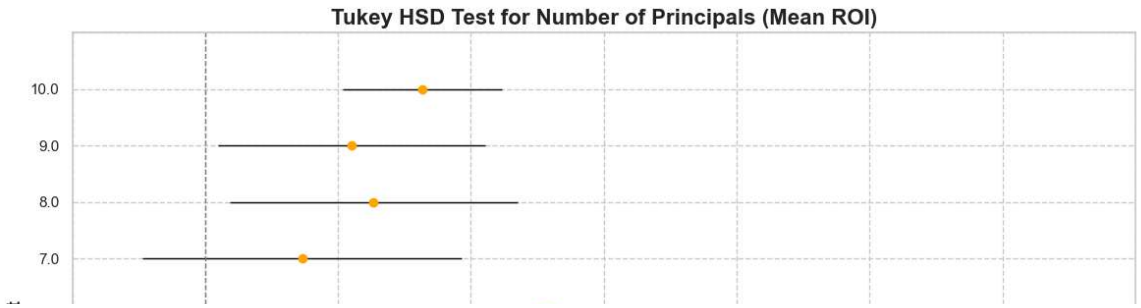
```
ANOVA F-statistic: 3.707585063663019, p-value: 0.0001200232505182961
```

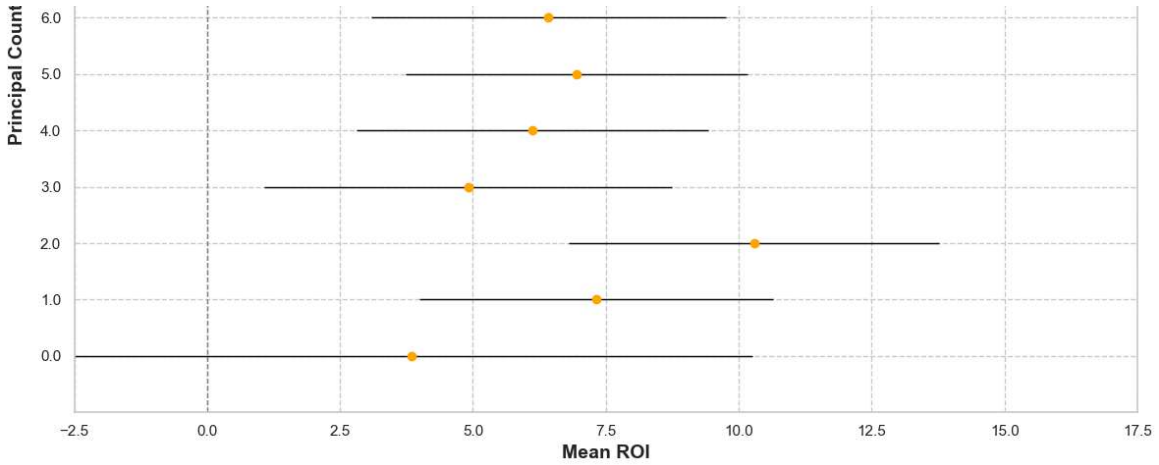
Results are significant for p<0.5 so we follow up with a tukey test to determine which numbers of principals perform well.

In [50]:

```
df_clean_filtered = df_clean_movies.replace([np.inf, -np.inf], np.nan).dropna(
# Perform the Tukey HSD test
pcr_tk_hsd = pairwise_tukeyhsd(df_clean_filtered['ROI'], df_clean_filtered['pr


plt.figure(figsize=(12, 8))

# Plot the Tukey HSD results
pcr_tk_hsd.plot_simultaneous(figsize=(12, 8))


lines = plt.gca().lines
for line in lines:
    line.set_color('orange')
    line.set_linewidth(2)


points = plt.gca().collections[0]
points.set_edgecolor('black')
points.set_linewidth(1)


plt.axvline(0, color='grey', linestyle='--', linewidth=1)

plt.xlim(-2.5, 17.5)
plt.title('Tukey HSD Test for Number of Principals (Mean ROI)', fontsize=16, w
plt.xlabel('Mean ROI', fontsize=14, weight='bold')
plt.ylabel('Principal Count', fontsize=14, weight='bold')


plt.grid(True, linestyle='--', linewidth=1)


plt.tight_layout()


plt.show()
```

```
<Figure size 1200x800 with 0 Axes>
```

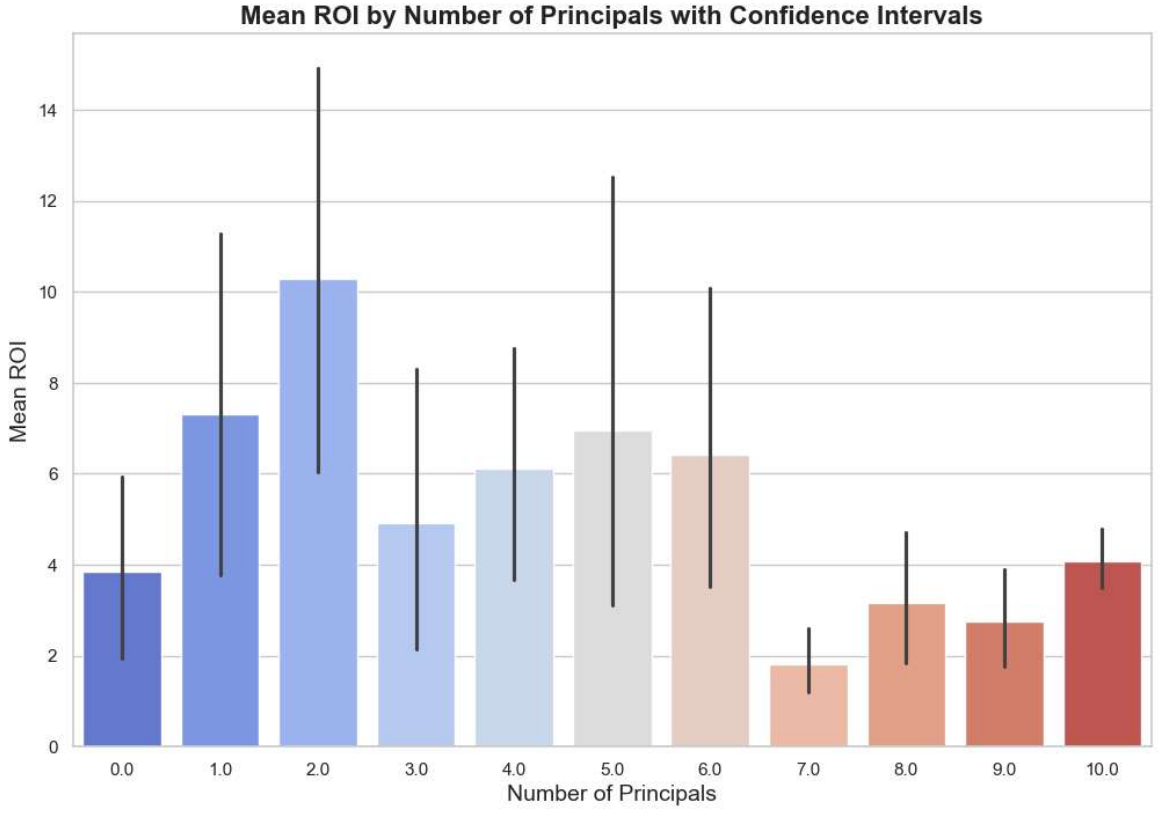




In [51]:
```
plt.figure(figsize=(12, 8))
sns.barplot(data=df_clean_filtered,x='principal_count',y='ROI', orient='v', pa

plt.title('Mean ROI by Number of Principals with Confidence Intervals', fontsi
plt.ylabel('Mean ROI', fontsize=14)
plt.xlabel('Number of Principals', fontsize=14)
```

```
C:\Users\erica\AppData\Local\Temp\ipykernel_31200\2959871837.py:2: FutureWarnin
g:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same eff
ect.

  sns.barplot(data=df_clean_filtered,x='principal_count',y='ROI', orient='v', pa
lette='coolwarm')
```

Out[51]: Text(0.5, 0, 'Number of Principals')



This visualization helps identify which specific groups of principal counts perform better or worse in terms of ROI. The confidence intervals allow us to assess which principal counts significantly differ from each other in terms of financial performance. Specifically we see that 2 principals has signficiantly higher mean ROI than 7-10 principlas with 95% confidence, and has the highest mean ROI overall, so it seems likely that a "dynamic duo" can boost a film's appeal, while a large principal cast is more risky.

In [52]:
```
top_directors.head(20)
```

Out[52]:

	director_name	ROI	genres_x	adjusted_production_budget	budg
1	Nathaniel Davis	68.609409	Horror	7.413009e+04	
2	Tod Williams	58.170677	Horror	4.004398e+06	
3	Damien Leone	50.165976	Horror	2.500000e+05	
4	Bradley Parker	41.411721	Horror,Mystery,Thriller	1.274210e+06	
6	Barry Jenkins	28.451563	Drama,Romance, Horror, Drama	5.869757e+06	Me
7	William Lustig	27.571429	Horror,Thriller, Action,Horror,Thriller	1.186512e+06	
9	William Brent Bell	22.735758	Documentary, Horror,Mystery,Thriller, Horror, ...	1.314730e+07	Me
10	Jordan Peele	21.215722	Horror,Mystery,Thriller, Comedy, Drama	1.856064e+07	Me
12	Jeff Wadlow	20.568114	Horror,Thriller, Action,Comedy,Crime, Drama,Ro...	8.433343e+06	Me
13	John R. Leonetti	19.736494	Drama,Fantasy,Horror, Horror,Mystery,Thriller	1.116972e+07	Me
14	Stiles White	19.660126	Horror,Mystery,Thriller	6.229665e+06	Me
15	Oyefunke Fayoyin	18.735563	Thriller, Horror,Mystery,Thriller, Documentary...	6.184547e+06	Me
16	Michael Madsen	18.735563	Thriller, Horror,Mystery,Thriller, Documentary...	6.184547e+06	Me
17	John Krasinski	18.698628	Documentary, Drama,Horror,Sci-Fi	1.979644e+07	Me
19	Daniel Stamm	18.496472	Horror,Thriller, Drama,Horror,Thriller	3.693185e+06	
21	James	17.222144	Horror,Thriller,	9.032390e+06	Me

21	DeMonaco	17.222144	Action,Horror,Sci-Fi	9.052590e+06	Me
22	Adam Robitel	16.288277	Horror,Mystery,Thriller, Horror,Thriller, Horr...	1.059599e+07	Me
25	Stephen Susco	15.434588	Crime,Horror,Mystery	1.164497e+06	
27	Ti West	14.131549	Action,Comedy, Drama,Horror,Thriller	1.000000e+06	
33	Leigh Whannell	13.178853	Horror,Thriller, Action,Drama,Horror, Sci-Fi, ...	6.114765e+06	Me

Here we have a list of our top 25 available and affordable high-performing directors (measured by mean ROI) who work in Horror and/or Drama, with helpful annotations on budget category and the number of movies they directed that we have data for. Given the small sample sizes, statistical comparisons like ANOVA or Tukey's test may not provide significant insights. However, this list can serve as a reliable starting point for determining which directors to approach. With confidence in their track record, directors like Jordan Peele, Jeff Wadlow, or William Brent Bell would be ideal partners for any project.