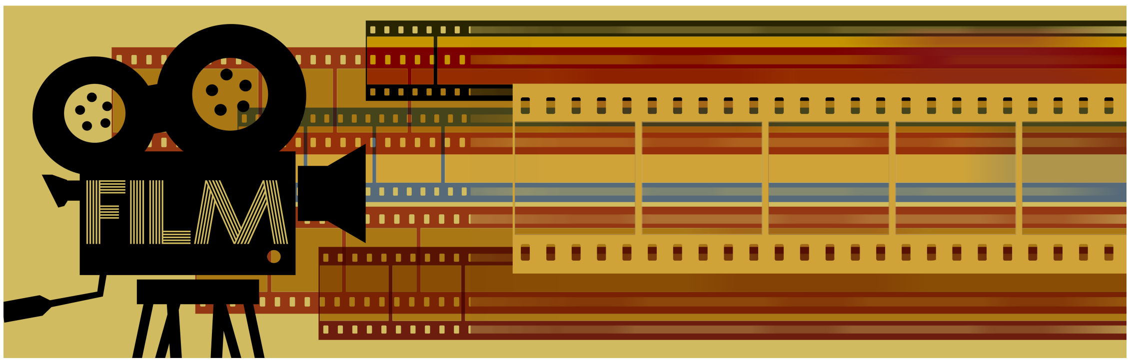
movie_studio_ROI_analysis

October 28, 2023

1 Movie Analysis



Credit : geralt from pixabay.com

2 Overview

The "Jelly Movie Studio Project" embarked on a mission to identify the critical factors associated with achieving the highest Return on Investment (ROI) in the film industry. This comprehensive analysis delved into various datasets, uncovering valuable insights that can guide the strategic decisions of the Jelly Movie Studio.

3 Business Problem

Identifying Movie Features Related to the Highest Return on Investment (ROI)

4 Data Understanding

4.1 Data Sets

1. IMDB Movie Database
    - (a) Ratings
    - (b) People - writers, directors
    - (c) Features information
2. Box Office Earnings from MOJO
3. Movie Budgets from the Numbers

4.2 Conditions

1. Domestic
2. Since 2010
3. Box Office Revenue
4. Top 10 Genres by Revenue

4.3 Data Limitations

1. Data Gaps
   - (a) Missing Multiple Years
   - (b) Sample Size Datasets
2. Financial Details
   - (a) Streaming Revenue
   - (b) Full Budgets
   - (c) Final Costs
3. Data Cohesion
   - Limited Number of Relevant Records When Combining Incomplete Information

5 Data Preparation

5.1 Import

[1]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline

from scipy import stats
import seaborn as sns
import statsmodels.api as sm
from sklearn import linear_model
import statsmodels.stats.power as smp
import sqlite3
import warnings
warnings.simplefilter("ignore")

import requests
from bs4 import BeautifulSoup
import zipfile
```

5.2 Master data set by Andrei and Namsoo

[2]:
```
# Unzip im.db.zip
# Set paths
zip_file_path = 'zippedData/im.db.zip'
extraction_path = 'zippedData/'
```

```
# Open the zipped file
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    # Extract all the contents to the specified directory
    zip_ref.extractall(extraction_path)
```

[3]:
```
# Assign csv files and im.db
movie_tmdb = pd.read_csv('zippedData/tmdb.movies.csv.gz')
movie_budget = pd.read_csv('zippedData/tn.movie_budgets.csv.gz')
theatre_weekend = pd.read_csv('zippedData/weekend_theaters_numbers.csv')
conn = sqlite3.connect('zippedData/im.db')
```

5.2.1 Clean database and tables

[4]:
```
# Assign database to dataframe
sql_query = pd.read_sql_query ('''
                               SELECT
                               mb.original_title,
                               mb.start_year,
                               mb.runtime_minutes,
                               mb.genres,
                               mr.averagerating,
                               mr.numvotes,
                               p.primary_name,
                               p.primary_profession
                               FROM movie_akas
                               JOIN movie_ratings mr
                               USING(movie_id)
                               JOIN movie_basics mb
                               USING(movie_id)
                               JOIN writers
                               USING(movie_id)
                               JOIN persons p
                               USING(person_id)
                               WHERE start_year >= 2010
                               AND region = 'US'
                               GROUP BY primary_title
                               ''', conn)

df = pd.DataFrame(sql_query, columns = [
    'original_title', 'start_year', 'runtime_minutes',
    'genres', 'averagerating', 'numvotes',
    'primary_name', 'primary_profession']
)
```

[5]:
```
# Clean original_title by lowering case
df.original_title = df.original_title.str.lower()
df.genres = df.genres.str.lower()
```

```
# Fillin missing values of runtime_minutes with mean
df.runtime_minutes = df.runtime_minutes.fillna(df.runtime_minutes.mean())

# Rename columns
df = df.rename(columns={'original_title': 'movie',
                        'start_year':'release_year'})

# Drop missing values
df = df.dropna(subset=['genres', 'primary_profession'])
```

[6]:
```
# Clean the movie_budget table
movie_budget.release_date = pd.to_datetime(movie_budget.release_date) #␣
 ↪Changing datatype for next filtering
movie_budget = movie_budget[movie_budget.release_date >= '2010-01-01'] #␣
 ↪Filtering target duration
movie_budget = movie_budget.drop(columns=['id',
                                          'worldwide_gross',
                                          'release_date'
                                          ]
                                          ) # Dropping unrelevant data
movie_budget.movie = movie_budget.movie.str.lower() # Lowring title case
```

[7]:
```
# Clean the theaters_numbers table
theatre_weekend = theatre_weekend.drop(columns=['year', 'rank']) # Dropping␣
 ↪unrelevant data
theatre_weekend = theatre_weekend.dropna(subset=['domestic_box_office',
                                                 'opening_weekend_box_office',
                                                 'movie',
                                                ]
                                                ) # Dropping missing values␣
 ↪which make noise in analysis
theatre_weekend.movie = theatre_weekend.movie.str.lower() # Lowring title case

# Fill-in missing values with a correct category
theatre_weekend.distributor = theatre_weekend.distributor.fillna("No␣
 ↪Distributor")
```

[8]:
```
# Clean the movie_tmdb table
movie_tmdb.release_date = pd.to_datetime(movie_tmdb.release_date) # Changing␣
 ↪datatype for next filtering
movie_tmdb = movie_tmdb[movie_tmdb.release_date >= '2010-01-01'] # Filtering␣
 ↪target duration

# Dropping unrelevant data
movie_tmdb = movie_tmdb.drop(columns=['Unnamed: 0',
                                      'genre_ids',
```

```
                                        'id',
                                        'original_language',
                                        'original_title'
                                        ]
                                        )

movie_tmdb.title = movie_tmdb.title.str.lower() # Lowring title case
movie_tmdb =movie_tmdb.rename(columns={'title':'movie'}) # Rename a column for␣
 ↪merging
```

5.2.2 Merging Data Sets and Making calculations and columns in order

[9]:
```
# Merging Data Sets using the movie column
merge_1 = pd.merge(movie_tmdb, df, on="movie")
merge_2 = pd.merge(merge_1,theatre_weekend, on="movie")
data = pd.merge(merge_2,movie_budget, on="movie")
```

[10]:
```
conn4 = sqlite3.connect('zippedData/master_andrei_sql.db') # Create a sql db␣
 ↪and Connect
data.to_sql('master_andrei_sql', conn4, if_exists='replace', index=False) #␣
 ↪Transform df to db

# Making calcumations and columns in order
master_crafted = pd.read_sql(
    """
    WITH modified AS
    (
    SELECT
        DISTINCT movie AS movie_name,
        CAST(REPLACE(REPLACE(domestic_gross, '$', ''), ',', '') AS FLOAT) AS␣
 ↪domestic_gross_numeric,
        CAST(REPLACE(REPLACE(production_budget, '$', ''), ',', '') AS FLOAT) AS␣
 ↪production_budget_numeric,
        CAST(REPLACE(REPLACE(opening_weekend_box_office, '$', ''), ',', '') AS␣
 ↪FLOAT) AS opening_weekend_boxoffice,
        *
    FROM master_andrei_sql
    ORDER BY movie
    )

    SELECT
        movie_name,
        genres,
        ROUND(domestic_gross_numeric/ production_budget_numeric*100, 2) AS␣
 ↪percentage_ROI_gross_budget,
        domestic_gross_numeric AS gross,
```

```
        production_budget_numeric AS budget,
        opening_weekend_boxoffice,
        (vote_average + averagerating)/2 AS average_rating,
        ROUND((vote_count + numvotes)/2, 0) AS number_vote,
        primary_name,
        primary_profession,
        release_date,
        release_year,
        runtime_minutes,
        distributor,
        max_theatre_count

    FROM modified
    ORDER BY release_date
    ;
    """
, conn4
)
```

5.2.3 Sanity Check of Master Data set and Create a csv master file

[11]:
```
# de-duplicate on movie_name and release_year
master_dedup = master_crafted.drop_duplicates(subset=['movie_name',␣
 ↪'release_year'])

master_dedup.to_csv('zippedData/master_dedup_v2.csv', index=False) # Create a␣
 ↪csv file for further analysis
```

6 Data Analysis

6.1 Analysis by Namsoo

6.1.1 Yearly or Monthly Trends of Number of Movies, Budget, Gross, and Opening Weekend Boxoffice

Data Preparation for Yearly or Monthly Trends

[12]:
```
master_dedup.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1290 entries, 0 to 1471
Data columns (total 15 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   movie_name                   1290 non-null   object
 1   genres                       1290 non-null   object
 2   percentage_ROI_gross_budget  1290 non-null   float64
 3   gross                        1290 non-null   float64
 4   budget                       1290 non-null   float64
```

```
 5   opening_weekend_boxoffice    1290 non-null   float64
 6   average_rating               1290 non-null   float64
 7   number_vote                  1290 non-null   float64
 8   primary_name                 1290 non-null   object
 9   primary_profession           1290 non-null   object
 10  release_date                 1290 non-null   object
 11  release_year                 1290 non-null   int64
 12  runtime_minutes              1290 non-null   float64
 13  distributor                  1290 non-null   object
 14  max_theatre_count            1290 non-null   object
dtypes: float64(7), int64(1), object(7)
memory usage: 161.2+ KB
```

[13]:
```python
# Transform master_dedup to a sql database
conn5 = sqlite3.connect('zippedData/master_1.db')
master_dedup.to_sql('master_1', conn5, if_exists='replace', index=False)
```

[13]: 1290

[14]:
```python
# Extract months from 'release_date' column by using STRFTIME
# SELECT AVG()
line_graph = pd.read_sql(
    """
    SELECT
        STRFTIME('%m', release_date) AS months,
        AVG(percentage_ROI_gross_budget) AS avg_ROI,
        AVG(gross) AS avg_gross,
        AVG(opening_weekend_boxoffice) AS avg_opening_gross,
        AVG(budget) AS avg_budget,
        COUNT(*) AS number_movie
    FROM master_1
    GROUP BY months
    ORDER BY months
    ;
    """
, conn5
)
```

[15]:
```python
# Extract years from 'release_date' column by using STRFTIME
# SELECT AVG()
line_graph_year = pd.read_sql(
    """
    SELECT
        STRFTIME('%Y', release_date) AS years,
        AVG(percentage_ROI_gross_budget) AS avg_ROI,
        AVG(gross) AS avg_gross,
        AVG(opening_weekend_boxoffice) AS avg_opening_gross,
        AVG(budget) AS avg_budget,
```

```
        COUNT(*) AS number_movie
    FROM master_1
    GROUP BY years
    ORDER BY years
    ;
    """
, conn5
)
line_graph_year
```

[15]:

```
   years      avg_ROI     avg_gross  avg_opening_gross    avg_budget  \
0   2010  200.000909  6.145796e+07       1.752015e+07  4.672205e+07
1   2011  183.815829  5.296555e+07       1.583070e+07  4.358524e+07
2   2012  273.894837  6.571783e+07       1.974067e+07  4.861235e+07
3   2013  199.444733  6.058434e+07       1.760928e+07  5.110985e+07
4   2014  201.276458  6.414625e+07       1.953153e+07  4.555257e+07
5   2015  357.010797  5.670907e+07       1.762712e+07  4.316051e+07
6   2016  214.776124  6.873269e+07       2.146894e+07  5.023245e+07
7   2017  199.713571  6.530314e+07       1.968499e+07  5.717149e+07
8   2018  220.936545  7.467405e+07       2.291051e+07  4.979727e+07
9   2019  148.690000  5.446788e+07       1.052933e+07  2.275000e+07

   number_movie
0           165
1           187
2           153
3           150
4           144
5           138
6           129
7           112
8           110
9             2
```

[16]:

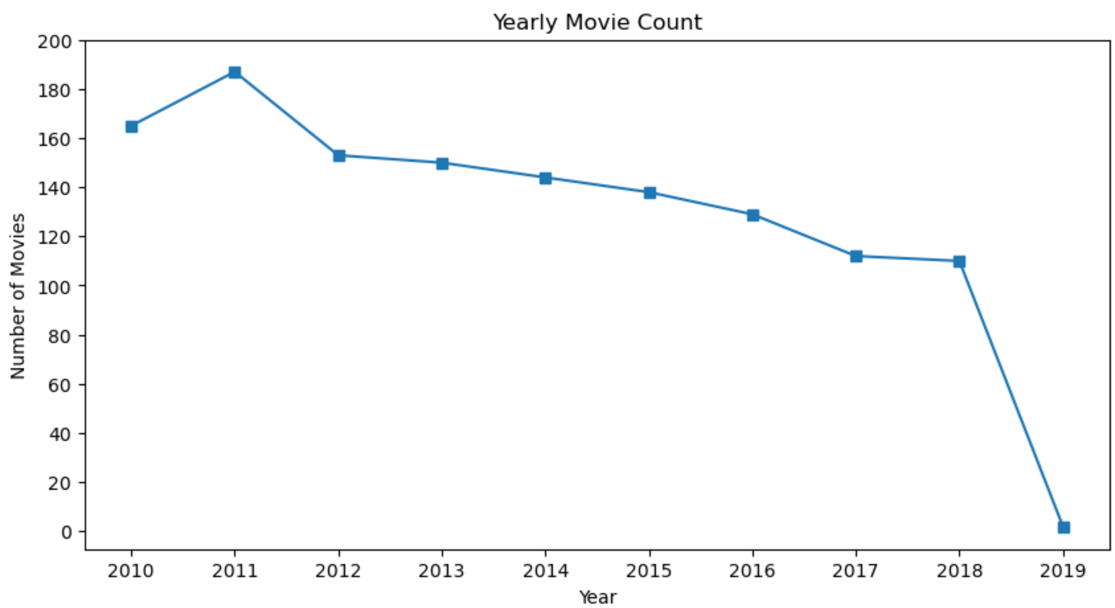
```
line_graph.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12 entries, 0 to 11
Data columns (total 6 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   months             12 non-null     object
 1   avg_ROI            12 non-null     float64
 2   avg_gross          12 non-null     float64
 3   avg_opening_gross  12 non-null     float64
 4   avg_budget         12 non-null     float64
 5   number_movie       12 non-null     int64
dtypes: float64(4), int64(1), object(1)
```

memory usage: 704.0+ bytes

Number of Movies by Year

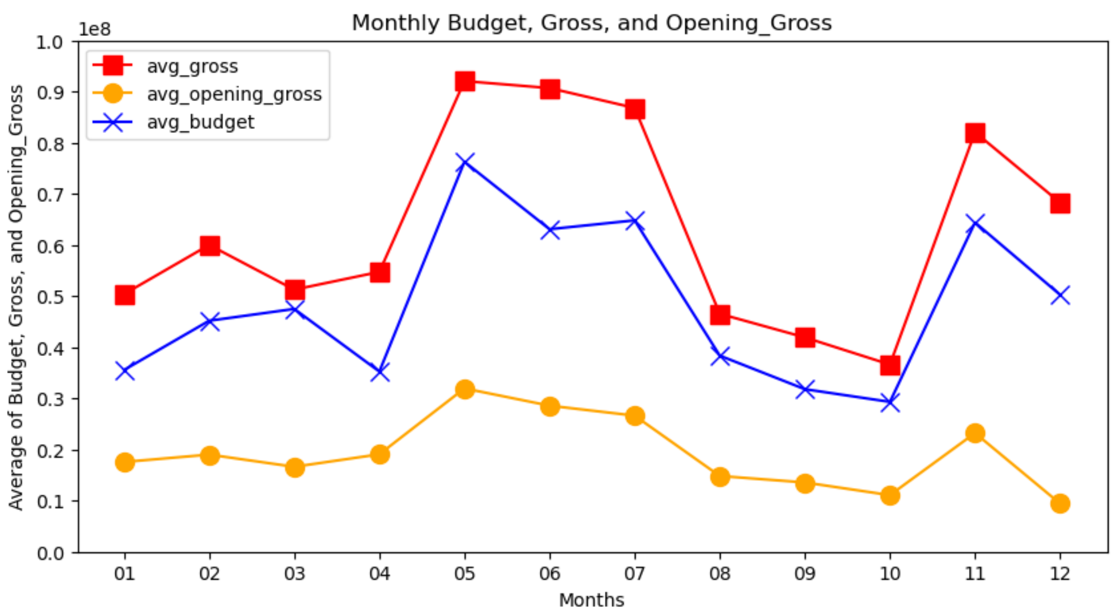
[17]:
```
# Line graph of number_movie by months
fig, ax = plt.subplots(figsize = (10, 5))

ax.plot(line_graph_year['years'], line_graph_year['number_movie'], marker='s',␣
↪linestyle='-')

ax.set_yticks(range(0, 201, 20))
ax.set_xlabel('Year')
ax.set_ylabel('Number of Movies')
ax.set_title('Yearly Movie Count')

plt.show();
```



Average of Budget, Gross, and Opening_Gross by Month

[18]:
```
# Line graph of gross, opening_gross, and budget
fig, ax = plt.subplots(figsize = (10, 5))

ax.plot(line_graph['months'], line_graph['avg_gross'], label = 'avg_gross',␣
↪marker='s', markersize=10, linestyle='-', color='red')

ax.plot(line_graph['months'], line_graph['avg_opening_gross'], label =␣
↪'avg_opening_gross', marker='o', markersize=10, linestyle='-',␣
↪color='orange')
```

```
ax.plot(line_graph['months'], line_graph['avg_budget'], label = 'avg_budget',␣
↪marker='x', markersize=10, linestyle='-', color='blue')

ax.set_yticks(range(0, 110000000, 10000000))
ax.set_xlabel('Months')
ax.set_ylabel('Average of Budget, Gross, and Opening_Gross')
ax.set_title('Monthly Budget, Gross, and Opening_Gross')

plt.legend(loc='upper left')

plt.show();
```



6.1.2 What Are the Most Profitable Genres?

Within our dataset, we have 205 different movie genres. To find out which genres are the most profitable, we will focus on genres that have more than 6 movies which is 75th percentile.

Calculating the number of movies in the 75th percentile

[19]:
```
number_movies_genres = pd.read_sql(
    """
    SELECT
        genres,
        COUNT(*) AS number_movie
    FROM master_1
    GROUP BY genres
```

```
    ORDER BY number_movie DESC
    ;
    """
, conn5
)
```

[20]:
```
number_movies_genres.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 2 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   genres        205 non-null    object
 1   number_movie  205 non-null    int64
dtypes: int64(1), object(1)
memory usage: 3.3+ KB
```

[21]:
```
number_movies_genres['number_movie'].describe()
```

[21]:
```
count    205.000000
mean       6.292683
std       10.144650
min        1.000000
25%        1.000000
50%        3.000000
75%        6.000000
max       63.000000
Name: number_movie, dtype: float64
```

Horror releated genres are in top 4 ROI and average of ROI is 942%

[22]:
```
top_4_roi = pd.read_sql(
    """
   WITH gen_roi_num AS
   (
    SELECT
        genres,
        AVG(percentage_ROI_gross_budget) AS AVG_ROI,
        COUNT(*) AS Num_Movie
    FROM master_1
    GROUP BY genres
    ORDER BY AVG_ROI DESC
    )

    SELECT
        DENSE_RANK() OVER(ORDER BY AVG_ROI DESC) AS Rank_N,
        genres,
```

```
        AVG_ROI,
        Num_Movie


    FROM gen_roi_num
    WHERE Num_Movie >= 6 -- Filtering Justification : 6 is 75th percentile␣
 ↪value of number of movie for genres
    ORDER BY AVG_ROI DESC
    LIMIT 4
    ;
    """
, conn5
)

top_4_roi
```

[22]:

	Rank_N	genres	AVG_ROI	Num_Movie
0	1	horror,mystery,thriller	1902.296538	26
1	2	horror	735.743077	13
2	3	drama,horror,thriller	582.323333	6
3	4	horror,thriller	547.683043	23

[23]:
```
top_4_roi['AVG_ROI'].mean()
```

[23]: 942.0114980490524

6.1.3 What's the right amount of money to spend on making a movie?

Calculating the ROI in the 75th percentile 227 is 75th percentile of percentage of ROI

[24]:
```
AVG_ROI_genres = pd.read_sql(
    """
    WITH gen_ROI_movie_N AS
    (
    SELECT
        genres,
        AVG(percentage_ROI_gross_budget) AS AVG_ROI,
        COUNT(*) AS movie_N
    FROM master_1
    GROUP BY genres
    ORDER BY AVG_ROI DESC
    )
    SELECT
        genres,
        AVG_ROI,
        movie_N
    FROM gen_ROI_movie_N
    ORDER BY AVG_ROI DESC
```

```
    ;
    """
, conn5
)
```

[25]:
```
AVG_ROI_genres.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   genres   205 non-null    object
 1   AVG_ROI  205 non-null    float64
 2   movie_N  205 non-null    int64
dtypes: float64(1), int64(1), object(1)
memory usage: 4.9+ KB
```

[26]:
```
AVG_ROI_genres['AVG_ROI'].describe()
```

[26]:
```
count     205.000000
mean      187.176660
std       219.826937
min         0.020000
25%        68.140000
50%       116.207500
75%       227.346000
max      1902.296538
Name: AVG_ROI, dtype: float64
```

Range of budgets of horror related genres in top 25% of ROI $3M - $11M

[27]:
```
AVG_ROI_genres_2 = pd.read_sql(
    """
    WITH gen_ROI_movie_N AS
    (
    SELECT
        genres,
        AVG(percentage_ROI_gross_budget) AS AVG_ROI,
        COUNT(*) AS movie_N,
        AVG(budget) AS AVG_budget
    FROM master_1
    WHERE percentage_ROI_gross_budget >= 227  -- Filtering Justification : 227␣
 ↪is 75th percentil value of ROI
    GROUP BY genres
    ORDER BY AVG_ROI DESC
    )
    SELECT
```

```
        DENSE_RANK() OVER(ORDER BY AVG_ROI DESC) AS Rank_N,
        genres,
        AVG_ROI,
        movie_N,
        AVG_budget
    FROM gen_ROI_movie_N
    WHERE
        genres LIKE '%horror%'
        AND movie_N >= 3 -- Filtering Justification : 3 is median value of␣
↪number of movie for genres
    ORDER BY AVG_ROI DESC
    LIMIT 7
    ;
    """
, conn5
)
```

[28]:
```
AVG_ROI_genres_2
```

[28]:

	Rank_N	genres	AVG_ROI	movie_N	AVG_budget
0	1	horror,mystery,thriller	2318.192857	21	6.814286e+06
1	2	horror	2234.172500	4	1.150000e+07
2	3	horror,thriller	1634.200000	7	9.571429e+06
3	4	drama,horror,thriller	860.345000	4	7.200000e+06
4	5	action,horror,sci-fi	706.696667	3	1.066667e+07
5	6	drama,horror,mystery	696.377500	4	6.375000e+06
6	7	horror,mystery	687.617500	4	3.875000e+06

6.1.4 Which directors or writers are top ROI achievers in the horror genres?

[29]:
```
AVG_ROI_people = pd.read_sql(
    """
    WITH gen_ROI_movie_N AS
    (
    SELECT
        primary_name,
        primary_profession,
        AVG(percentage_ROI_gross_budget) AS AVG_ROI,
        COUNT(*) AS movie_N,
        AVG(budget) AS AVG_budget,
        genres
    FROM master_1
    GROUP BY primary_name
    ORDER BY AVG_ROI DESC
    )
    SELECT
        DENSE_RANK() OVER(ORDER BY AVG_ROI DESC) AS Rank_N,
```

```
        primary_name,
        primary_profession,
        AVG_ROI,
        movie_N,
        AVG_budget,
        genres
    FROM gen_ROI_movie_N
    WHERE
        genres LIKE '%horror%'
        AND movie_N >= 3 -- Filtering Justification : 3 is median value of␣
↪number of movie for genres
    ORDER BY AVG_ROI DESC
    LIMIT 5
    ;
    """
, conn5
)
```

[30]:
```
AVG_ROI_people
```

[30]:
```
   Rank_N        primary_name         primary_profession      AVG_ROI  movie_N  \
0       1      Leigh Whannell     actor,writer,producer  1559.100000        3
1       2  Christopher Landon  writer,producer,director  1331.526000        5
2       3      James DeMonaco  writer,producer,director  1067.297500        4
3       4      Andrew Gurland  producer,writer,director   897.003333        3
4       5    Scott Derrickson  writer,director,producer   673.770000        3

     AVG_budget                  genres
0  4.833333e+06  horror,mystery,thriller
1  6.600000e+06                   horror
2  8.750000e+06          horror,thriller
3  2.600000e+06    drama,horror,thriller
4  5.933333e+07  horror,mystery,thriller
```

6.2 Analysis by Kari

[31]:
```
# Reading a csv file to a dataframe
master_final_df = pd.read_csv('zippedData/master_dedup_v2.csv')
```

6.2.1 Two Sample T-Test: One-tailed

Null Hypothesis: The mean ROI of horror movies is less than or equal to that of non-horror movies.

Alternative Hypothesis: The mean ROI of horror movies is greater than that of non-horror movies.

Create Sample Groups

[32]:
```
# Split data into two groups (samples)
horror_movies = master_final_df[master_final_df['genres'].str.
↪contains('horror', case=False, regex=True)]
non_horror_movies = master_final_df[~master_final_df['genres'].str.
↪contains('horror', case=False, regex=True)]

# Sanity check
horror_movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 150 entries, 11 to 1275
Data columns (total 15 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   movie_name                   150 non-null    object
 1   genres                       150 non-null    object
 2   percentage_ROI_gross_budget  150 non-null    float64
 3   gross                        150 non-null    float64
 4   budget                       150 non-null    float64
 5   opening_weekend_boxoffice    150 non-null    float64
 6   average_rating               150 non-null    float64
 7   number_vote                  150 non-null    float64
 8   primary_name                 150 non-null    object
 9   primary_profession           150 non-null    object
 10  release_date                 150 non-null    object
 11  release_year                 150 non-null    int64
 12  runtime_minutes              150 non-null    float64
 13  distributor                  150 non-null    object
 14  max_theatre_count            150 non-null    object
dtypes: float64(7), int64(1), object(7)
memory usage: 18.8+ KB
```

[33]:
```
# Sanity check
non_horror_movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1140 entries, 0 to 1289
Data columns (total 15 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   movie_name                   1140 non-null   object
 1   genres                       1140 non-null   object
 2   percentage_ROI_gross_budget  1140 non-null   float64
 3   gross                        1140 non-null   float64
 4   budget                       1140 non-null   float64
 5   opening_weekend_boxoffice    1140 non-null   float64
 6   average_rating               1140 non-null   float64
 7   number_vote                  1140 non-null   float64
```

```
 8   primary_name                1140 non-null   object
 9   primary_profession          1140 non-null   object
 10  release_date                1140 non-null   object
 11  release_year                1140 non-null   int64
 12  runtime_minutes             1140 non-null   float64
 13  distributor                 1140 non-null   object
 14  max_theatre_count           1140 non-null   object
dtypes: float64(7), int64(1), object(7)
memory usage: 142.5+ KB
```

Interpretation of Two-Sample T-Test (One-Tailed) The result of this analysis indicates that there is a 95% chance that the mean ROI of horror movies is significantly greater than the mean ROI of non-horror movies.

[34]:
```
# Set the significance level
alpha = 0.05

# Perform a two-sample t-test to compare the mean 'ROI' of horror and␣
↪non-horror movies
t_statistic, p_value = stats.
↪ttest_ind(horror_movies['percentage_ROI_gross_budget'],␣
↪non_horror_movies['percentage_ROI_gross_budget'], equal_var=False)


# Determine whether to reject the null
if p_value < alpha and t_statistic < 0:
    result = 'Reject the null hypothesis'

elif p_value < alpha and t_statistic > 0:
    result = 'Reject the null hypothesis'
else:
    result = 'Fail to reject the null hypothesis'


# Display the results
print(f'T-statistic: {t_statistic}')
print(f'P-value: {p_value}')
print(f'Result: {result}')
```

```
T-statistic: 2.890674145719811
P-value: 0.004417071508529201
Result: Reject the null hypothesis
```

Power Analysis of Two-Sample T-Test

[35]:
```
# Determine effect size using Cohen's d for a two-sample test

horror_mean =  horror_movies['percentage_ROI_gross_budget'].mean()
```

```
non_horror_mean =  non_horror_movies['percentage_ROI_gross_budget'].mean()
horror_std =   horror_movies['percentage_ROI_gross_budget'].std()
non_horror_std =   horror_movies['percentage_ROI_gross_budget'].std()

horror_sample = 150
non_horror_sample = 1140

# Calculate pooled standard deviation
pooled_std = np.sqrt(((horror_sample - 1) * horror_std**2 + (non_horror_sample␣
↪- 1) * non_horror_std**2) / (horror_sample + non_horror_sample - 2))

# Calculate Cohen's d
effect_sizes = (horror_mean - non_horror_mean) / pooled_std

effect_sizes
```

[35]:

```
0.2362376885444173
```

[36]:

```
# Set the parameters for the power analysis
effect_size = .236238  # Set effect_size according to Cohen's d calculation
alpha = 0.05  # Set significance level
alternative = 'two-sided'

# Perform the power analysis with given sample sizes
analysis = smp.TTestIndPower()
result = analysis.solve_power(effect_size=effect_size, alpha=alpha,␣
↪nobs1=horror_sample, ratio=non_horror_sample/horror_sample,␣
↪alternative=alternative)

print(f'Statistical Power: {result}')
```

```
Statistical Power: 0.7757482834707635
```

Interpretation of Power Analysis A statistical power of 0.7757 means that there is a 77.57% chance that if there is a real difference or effect between the horror ROI and non_horror ROI, our test will detect it. This is a reasonably good statistical power and we are slightly reducing the risk of a Type II error (false-negative).

6.2.2 Linear Regression Model: budget & gross (revenue)

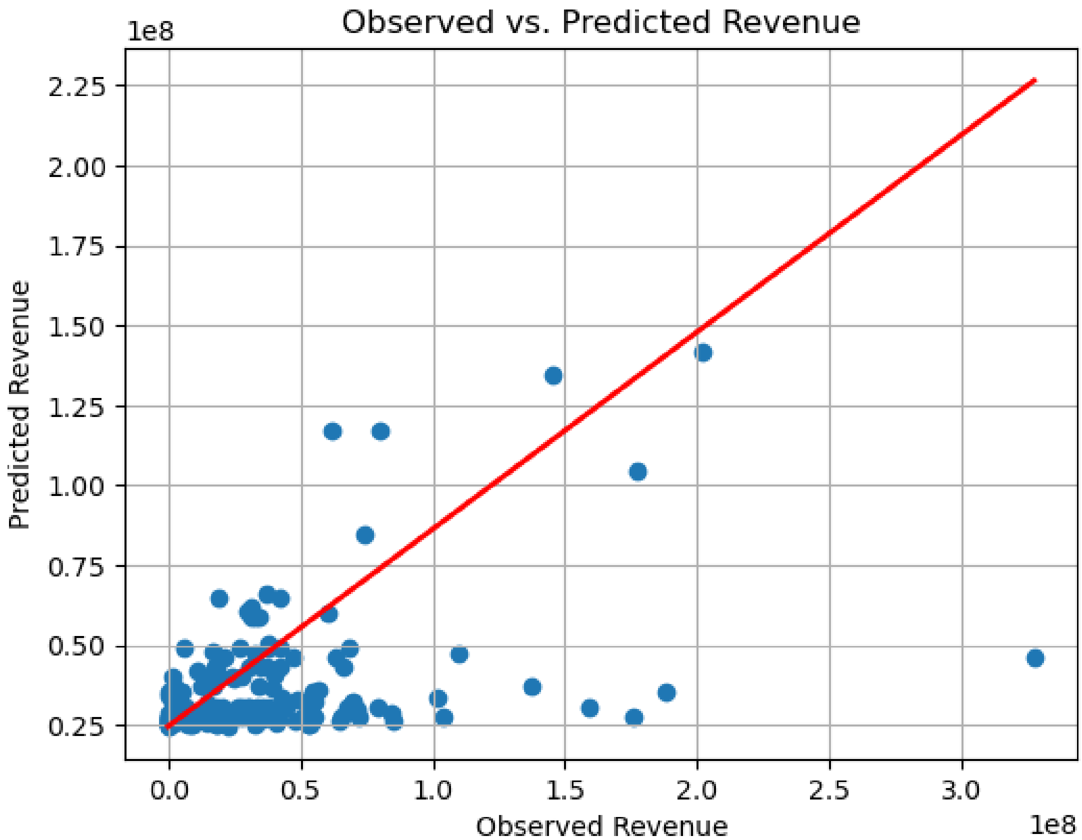
Independent Variable **Null Hypothesis:** The variable has no effect on revenue, and the coefficient is equal to zero.

Alternative Hypothesis: The variable has an effect on revenue, and the coefficient is not equal to zero.

Dependent Variable **Null Hypothesis:** The intercept represents a baseline revenue when the independent variable is zero.

Alternative Hypothesis: The intercept does not represent a meaningful baseline revenue when the independent variable is zero.

[37]:
```
# Add constant intercept to variables
X = sm.add_constant(horror_movies[['budget']])

# Dependent variable
y = horror_movies['gross']

# Apply fit test to model
model = sm.OLS(y, X).fit()

# Predicted values from model
predicted_values = model.predict(X)

# Create a scatter plot of observed vs. predicted ROI values
plt.scatter(y, predicted_values)
plt.xlabel('Observed Revenue')
plt.ylabel('Predicted Revenue')
plt.title('Observed vs. Predicted Revenue')
plt.grid(True)

# Calculate the minimum and maximum 'gross' values
min_gross = min(y)
max_gross = max(y)

# Create x values for the trendline (from min to max 'gross')
x_trendline = np.linspace(min_gross, max_gross, 100)

# Calculate corresponding predicted values for the trendline
y_trendline = model.predict(sm.add_constant(x_trendline))

# Add a regression line
plt.plot(x_trendline, y_trendline, color='red', linewidth=2, label='Trendline')


plt.show()



# Regression summary
print(model.summary())
```




```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  gross   R-squared:                       0.179
Model:                            OLS   Adj. R-squared:                  0.174
Method:                 Least Squares   F-statistic:                     32.29
Date:                Sat, 28 Oct 2023   Prob (F-statistic):           6.83e-08
Time:                        20:35:01   Log-Likelihood:                -2843.0
No. Observations:                 150   AIC:                             5690.
Df Residuals:                     148   BIC:                             5696.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       2.456e+07   4.07e+06      6.039      0.000    1.65e+07    3.26e+07
budget         0.6167      0.109      5.682      0.000       0.402       0.831
==============================================================================
Omnibus:                      132.231   Durbin-Watson:                   1.876
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1701.963
Skew:                           3.180   Prob(JB):                         0.00
```

```
Kurtosis:                      18.227   Cond. No.                     4.50e+07
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 4.5e+07. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Interpretation The low R-squared value suggests that budget explains only a small portion (17.9%) of the variance in revenue for horror movies.

However, the results indicate that horror movie budgets are a statistically significant predictor of horror movie revenue because of its low p-value for the F-statistic and coefficient.

Therefore, although budget is statistically significant in this model, there are other variables that can influence horror movie revenue.

6.2.3 Multiple Linear Regression Model : budget, number_vote, average_rating for ROI

Independent Variables **Null Hypothesis:** The variables have no effect on ROI, and their coefficients are equal to zero.

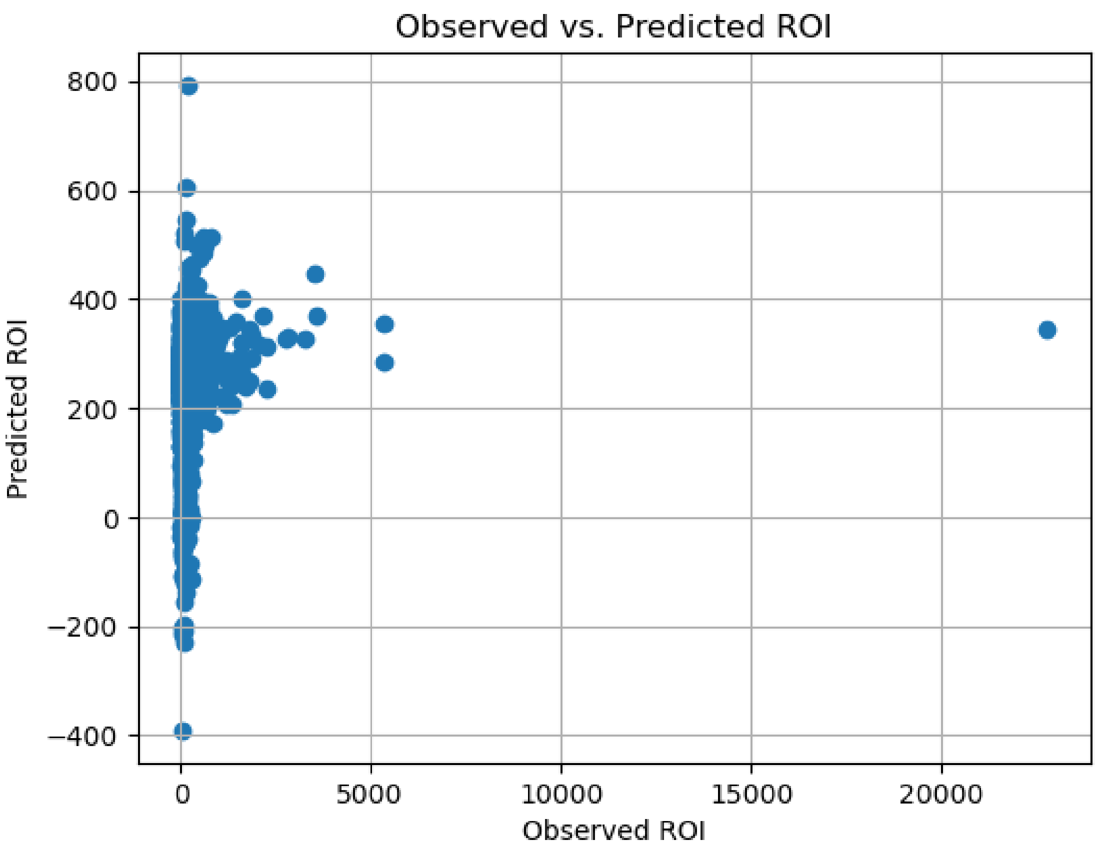
Alternative Hypothesis: The variables have an effect on ROI, and their coefficients are not equal to zero.

Dependent Variable **Null Hypothesis:** The intercept represents a baseline ROI when all independent variables are zero.

Alternative Hypothesis: The intercept does not represent a meaningful baseline ROI when all independent variables are zero.

[38]:
```python
# Add constant intercept to variables
X = sm.add_constant(master_final_df[['budget', 'average_rating',␣
 ↪'number_vote']])

# Dependent variable
y = master_final_df['percentage_ROI_gross_budget']

# Apply fit test to model
model = sm.OLS(y, X).fit()

# Predicted values from model
predicted_values = model.predict(X)

# Create a scatter plot of observed vs. predicted ROI values
plt.scatter(y, predicted_values)
plt.xlabel('Observed ROI')
plt.ylabel('Predicted ROI')
```

```python
plt.title('Observed vs. Predicted ROI')
plt.grid(True)
plt.show()


# Regression summary
print(model.summary())
```




```
                                OLS Regression Results
================================================================================
=======
Dep. Variable:     percentage_ROI_gross_budget   R-squared:
0.020
Model:                                     OLS   Adj. R-squared:
0.018
Method:                          Least Squares   F-statistic:
8.800
Date:                         Sat, 28 Oct 2023   Prob (F-statistic):
8.90e-06
Time:                                 20:35:01   Log-Likelihood:
```

```
-10336.
No. Observations:                        1290   AIC:
2.068e+04
Df Residuals:                            1286   BIC:
2.070e+04
Df Model:                                   3
Covariance Type:                    nonrobust
================================================================================
==
                    coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
--
const            518.0879    158.837      3.262      0.001     206.479
829.697
budget         -2.149e-06   4.25e-07     -5.053      0.000   -2.98e-06
-1.31e-06
average_rating   -40.4217     25.925     -1.559      0.119     -91.282
10.439
number_vote        0.0010      0.000      3.244      0.001       0.000
0.002
==============================================================================
Omnibus:                     3102.785   Durbin-Watson:                   1.926
Prob(Omnibus):                  0.000   Jarque-Bera (JB):         25716739.535
Skew:                          23.305   Prob(JB):                         0.00
Kurtosis:                     693.129   Cond. No.                     5.87e+08
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 5.87e+08. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Interpretation The low R-squared value suggests that this model may not be a good fit for explaining ROI. Additional factors or variables may be needed to improve its performance.

However, the results indicate that 'budget' and 'number_vote' are statistically significant predictors of ROI because they have very low p-values, while 'average_rating' may not be relevant in explaining variations in ROI.

6.2.4 Multiple Linear Regression Model: budget, genres, primary_name

[39]:
```
from sklearn.preprocessing import LabelEncoder

# Create separate LabelEncoder instances for 'Genre' and 'Director'
label_encoder_genres = LabelEncoder()
label_encoder_primary_name = LabelEncoder()
```

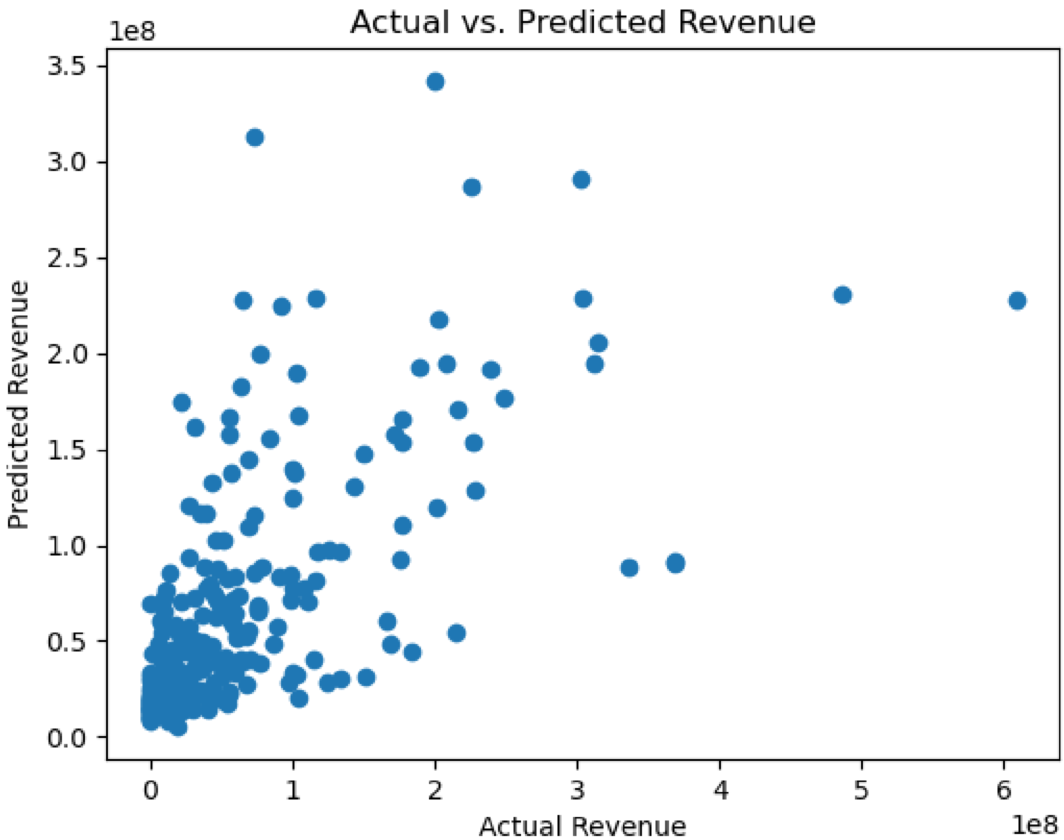
```python
# Apply label encoding to 'Genre' column
master_final_df['genres_encoded'] = label_encoder_genres.
 ↪fit_transform(master_final_df['genres'])

# Apply label encoding to 'Director' column
master_final_df['primary_name_encoded'] = label_encoder_primary_name.
 ↪fit_transform(master_final_df['primary_name'])

# Resulting DataFrame with label encoding for both columns
master_final_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1290 entries, 0 to 1289
Data columns (total 17 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   movie_name                  1290 non-null   object
 1   genres                      1290 non-null   object
 2   percentage_ROI_gross_budget 1290 non-null   float64
 3   gross                       1290 non-null   float64
 4   budget                      1290 non-null   float64
 5   opening_weekend_boxoffice   1290 non-null   float64
 6   average_rating              1290 non-null   float64
 7   number_vote                 1290 non-null   float64
 8   primary_name                1290 non-null   object
 9   primary_profession          1290 non-null   object
 10  release_date                1290 non-null   object
 11  release_year                1290 non-null   int64
 12  runtime_minutes             1290 non-null   float64
 13  distributor                 1290 non-null   object
 14  max_theatre_count           1290 non-null   object
 15  genres_encoded              1290 non-null   int64
 16  primary_name_encoded        1290 non-null   int64
dtypes: float64(7), int64(3), object(7)
memory usage: 171.5+ KB
```

[40]:
```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Load and preprocess the dataset (replace 'df' with your dataset)
# Ensure that 'budget' and 'revenue' are numerical columns.
# Encode categorical variables like genres and directors.

# Split the data into training and testing sets
```

```
x = master_final_df[['budget', 'genres_encoded', 'primary_name_encoded']]  #␣
↪Features
y = master_final_df['gross']  # Target variable
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,␣
↪random_state=42)

# Create and train a linear regression model
model = LinearRegression()
model.fit(x_train, y_train)

# Make predictions on the test set
y_pred = model.predict(x_test)

# Evaluate the model
mae = mean_squared_error(y_test, y_pred)
r_squared = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mae:.2f}")
print(f"R-squared: {r_squared:.2f}")

# Visualize predictions vs. actual values
plt.scatter(y_test, y_pred)
plt.xlabel("Actual Revenue")
plt.ylabel("Predicted Revenue")
plt.title("Actual vs. Predicted Revenue")
plt.show()
```

```
Mean Squared Error: 4065359596403251.50
R-squared: 0.42
```



6.3 Key Findings

Profitable Genres: In a dataset featuring 205 specific movie genres, our analysis focused on genres with at least six movies to assess Return on Investment (ROI). Notably, horror-related genres emerged as the top performers, boasting an impressive average ROI of 942%. This ROI percentage was calculated by dividing gross earnings by the budget and multiplying the result by 100.

Optimal Budget Range: For horror-related genres, the ideal budget range to achieve the 25th percentile ROI fell between $3 million and $11 million.

Director and Writer Recommendations: We recommend Christopher Landon, an exemplary talent in the horror genre, who has excelled as a writer, producer, and director within the domestic market.

Statistical Significance: A two-sample t-test provided a 95% confidence that the mean ROI of horror movies significantly outperformed that of non-horror movies, supported by a low p-value of 0.0044. Additionally, the power analysis demonstrated a robust statistical power of 77.57%.

Budget and Revenue Relationship: While a linear regression model indicated that budget explains only a modest portion (17.9%) of the revenue variance for horror movies, it remained a statistically significant predictor. This was evident from its low p-value and coefficient for the

F-statistic. This underscores the significant role of budget, though it also suggests the presence of other contributing variables.

Multiple Regression Model: In our attempt to employ a multiple regression model that incorporated budget, the number of votes, and average rating, we found that the R-squared value was relatively low. However, our analysis identified ‘budget’ and ‘number of votes’ as statistically significant predictors of ROI due to their exceptionally low p-values. In contrast, ‘average rating’ appeared to have less relevance in explaining variations in ROI.

7 Conclusions

The “Jelly Movie Studio Project” has successfully unveiled invaluable insights for guiding the strategic decisions of Jelly Movie Studio. It emphasized the prominence of horror-related genres, budget considerations, and the endorsement of talents like Christopher Landon. While budget emerged as a critical predictor, the intricate nature of the film industry implies the involvement of multifaceted factors influencing ROI. This holistic analysis lays a robust foundation for informed decision-making within the dynamic world of cinema.

8 Next Steps

Streaming Analysis: Evaluate the revenue potential and audience reach on streaming platforms before finalizing your distribution strategy.

International Market: Expand your analysis to include foreign horror films, as many influential movies in this genre originate from international sources.

Sequel Potential: Explore the possibility of creating films with sequel potential to tap into existing built-in audiences.

9 Source

IMDb The Numbers Box Office Mojo by IMDbPro Movie Budgets Top 10 Genres by Revenue Industry ROI Standards

10 About Us

Kari Primiano - Github Lead - Email: kkprim@gmail.com - github.com/kkprim

Namsoo Lee - Tech Lead - Email: likej1218@gmail.com - github.com/likej82

Andrei Hushcha - Presentation Lead - Email: andrew.hushcha@gmail.com - github.com/andreihushcha