LOAD THE LIBARAIES

```
#load the necessary library
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.multioutput import MultiOutputRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler

import ast
```

LOAD THE DATASET

```
movie = pd.read_csv('movie.csv')
```

```
movie.head()
```

	title	imdb_id	release_date	budget	revenue	tmdb_rating	vote_coun
0	Once Upon a Time... in Hollywood	tt7131622	2019-07-24	95000000	392105159	7.426	1423
1	Pain and Glory	tt8291806	2019-03-22	10769016	37359689	7.382	184
2	Taxi 5	tt7238392	2017-01-19	20390000	64497208	5.398	104
3	Wonder Park	tt6428676	2019-03-13	100000000	119559110	6.529	72
4	The King of Kings	tt7967302	2025-04-07	25200000	66465461	8.600	10

```
print(movie.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1373 entries, 0 to 1372
Data columns (total 16 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   title                   1373 non-null   object
 1   imdb_id                 1373 non-null   object
 2   release_date            1373 non-null   object
 3   budget                  1373 non-null   int64
 4   revenue                 1373 non-null   int64
 5   tmdb_rating             1373 non-null   float64
 6   vote_count              1373 non-null   int64
 7   runtimeMinutes          1373 non-null   int64
 8   imdb_rating             1373 non-null   float64
 9   avg_cast_rating         1370 non-null   float64
 10  director_rating         1371 non-null   float64
 11  writer_rating           1371 non-null   float64
 12  composer_rating         1321 non-null   float64
 13  cinematographer_rating  1312 non-null   float64
 14  editor_rating           1352 non-null   float64
 15  genre                   1373 non-null   object
dtypes: float64(8), int64(4), object(4)
memory usage: 171.8+ KB
None
```

```
movie.describe()
```

	budget	revenue	tmdb_rating	vote_count	runtimeMinutes	imdb_
count	1.373000e+03	1.373000e+03	1373.000000	1373.000000	1373.000000	1373
mean	4.205404e+07	1.038718e+08	6.684362	2058.007283	117.437728	6
std	5.921641e+07	2.474386e+08	0.848751	3077.486112	23.933306	1
min	1.000000e+00	0.000000e+00	0.000000	0.000000	53.000000	1
25%	7.000000e+06	1.120191e+06	6.255000	315.000000	100.000000	5
50%	2.000000e+07	1.816093e+07	6.764000	921.000000	112.000000	6
75%	5.000000e+07	8.246870e+07	7.200000	2513.000000	131.000000	7
max	5.839000e+08	2.799439e+09	10.000000	31060.000000	242.000000	8

DATA PREPROCESSING

HANDLING MISSING VALUES

Since only considering movies, removing documentary and biography

```
movie.dtypes
```

	0
title	object
imdb_id	object
release_date	object
budget	int64
revenue	int64
tmdb_rating	float64
vote_count	int64
runtimeMinutes	int64
imdb_rating	float64
avg_cast_rating	float64
director_rating	float64
writer_rating	float64
composer_rating	float64
cinematographer_rating	float64
editor_rating	float64
genre	object

dtype: object

```
#removing documentary and biography
movie = movie[~movie['genre'].str.contains('Documentary|Biography')]
```

```
movie.shape
```

```
(1238, 16)
```

```
# Shows rows where at least one column is null
null_rows = movie[movie.isnull().any(axis=1)]
null_rows
```

	title	imdb_id	release_date	budget	revenue	tmdb_rating	vote_
0	Once Upon a Time... in Hollywood	tt7131622	2019-07-24	95000000	392105159	7.426	
3	Wonder Park	tt6428676	2019-03-13	100000000	119559110	6.529	
37	Paws of Fury: The Legend of Hank	tt4428398	2022-07-14	45000000	42500000	6.654	
80	The Secret Life of Pets 2	tt5113040	2019-05-24	80000000	429434163	6.950	
86	The Dawn	tt7461372	2019-09-27	110000	0	6.045	
...	...	...	...	...	...	...	
1334	A Little Something Extra	tt30795948	2024-04-18	6400000	84058132	7.126	
1348	Jai Mata ji - lets Rock	tt35705898	2025-05-09	585317	0	0.000	
1349	Mahavatar Narsimha	tt34365591	2025-07-25	4700000	36000000	7.824	
1357	The Great Battle	tt6931414	2018-09-19	13305000	41509280	6.798	
1366	Oh, Ramona!	tt7200946	2019-02-14	2800000	1200582	5.769	

108 rows × 16 columns

```
#removing the missing values
df = movie.dropna()
```

```
df.shape
```

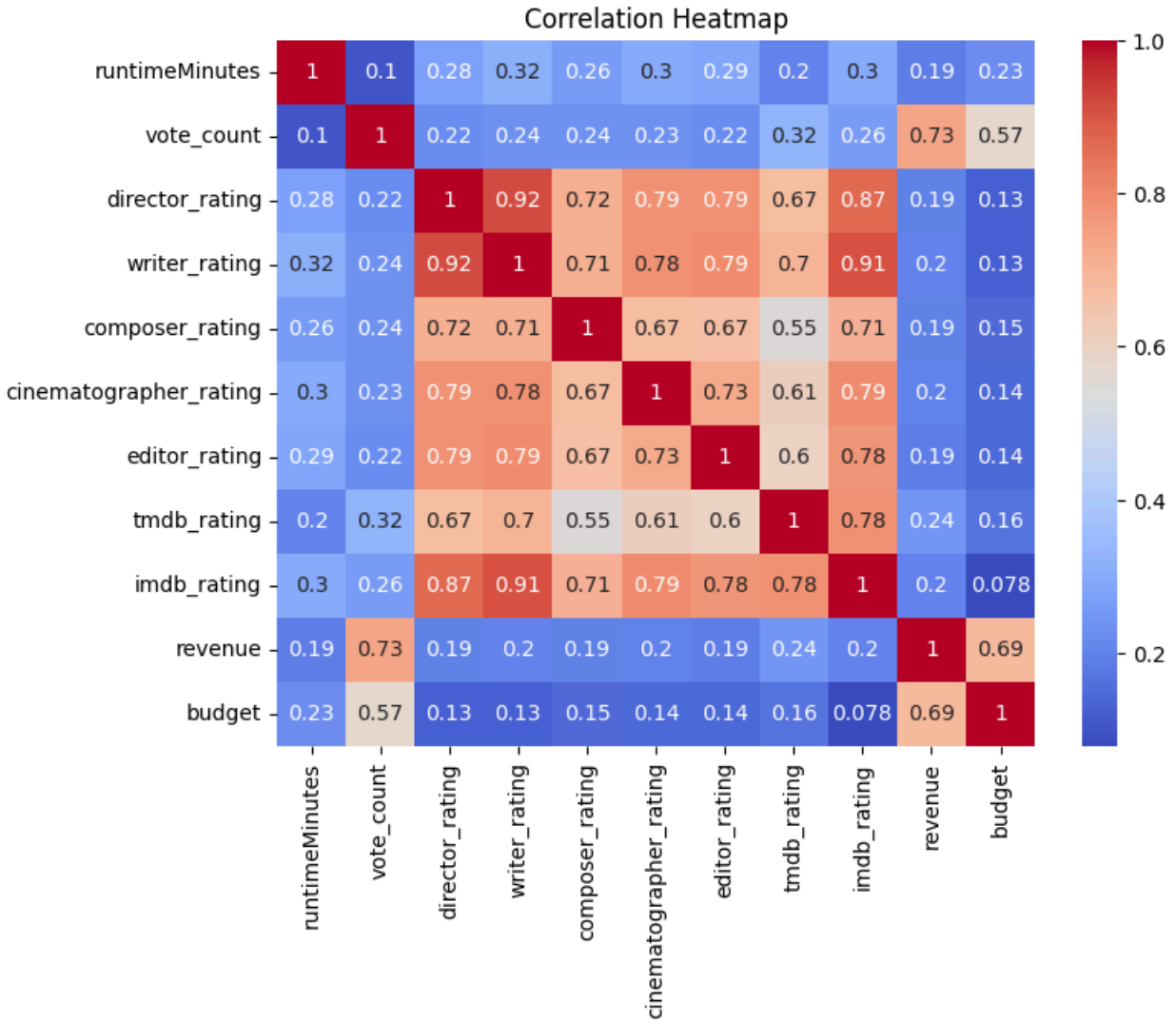
```
(1130, 16)
```

LINEAR REGRESSION FOR ANALYSING RATINGS

```
#define the features
df_x = df[['runtimeMinutes', 'vote_count', 'avg_cast_rating', 'director_rating',

#define the target
df_y = df[['tmdb_rating', 'imdb_rating']]
```

```
data = df[['runtimeMinutes', 'vote_count', 'director_rating', 'writer_rating', '
```

```
corr = data.corr()

plt.figure(figsize=(8,6))
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap")
plt.show()
```




```
# Scale features
scaler = StandardScaler()
x_scaled = scaler.fit_transform(df_x)
```

MODELLING THE ALGORITHM (LINEAR REGRESSION)

```
#split the data into training and testing sets (80% training and 20% testing)
x_train, x_test, y_train, y_test = train_test_split(x_scaled, df_y, test_size=0.
```

```
model = MultiOutputRegressor(LinearRegression())
#model
model.fit(x_train, y_train)
```

```
#make predictions
y_pred = model.predict(x_test)
```

```
#evaluate the model
mse = mean_squared_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)
```

```
#printing the values
print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
```

```
Mean Squared Error: 0.24710027427096906
R-squared: 0.6895618893579851
```

```
for i, estimator in enumerate(model.estimators_):
    print(f"Target {df_y.columns[i]} coefficients:")
    for feature, coef in zip(df_x.columns, estimator.coef_):
        print(f"  {feature}: {coef:.3f}")
    print()
```

```
Target tmdb_rating coefficients:
  runtimeMinutes: -0.029
  vote_count: 0.122
  avg_cast_rating: 0.389
  director_rating: 0.050
  writer_rating: 0.169
  composer_rating: -0.005
  cinematographer_rating: 0.007
  editor_rating: 0.003
  revenue: -0.009
  budget: -0.004

Target imdb_rating coefficients:
  runtimeMinutes: 0.004
  vote_count: 0.049
  avg_cast_rating: 0.627
  director_rating: 0.044
  writer_rating: 0.280
  composer_rating: 0.003
  cinematographer_rating: 0.039
  editor_rating: 0.026
  revenue: 0.030
  budget: -0.089
```

This output correlates with the heatmap

VISUALIZING

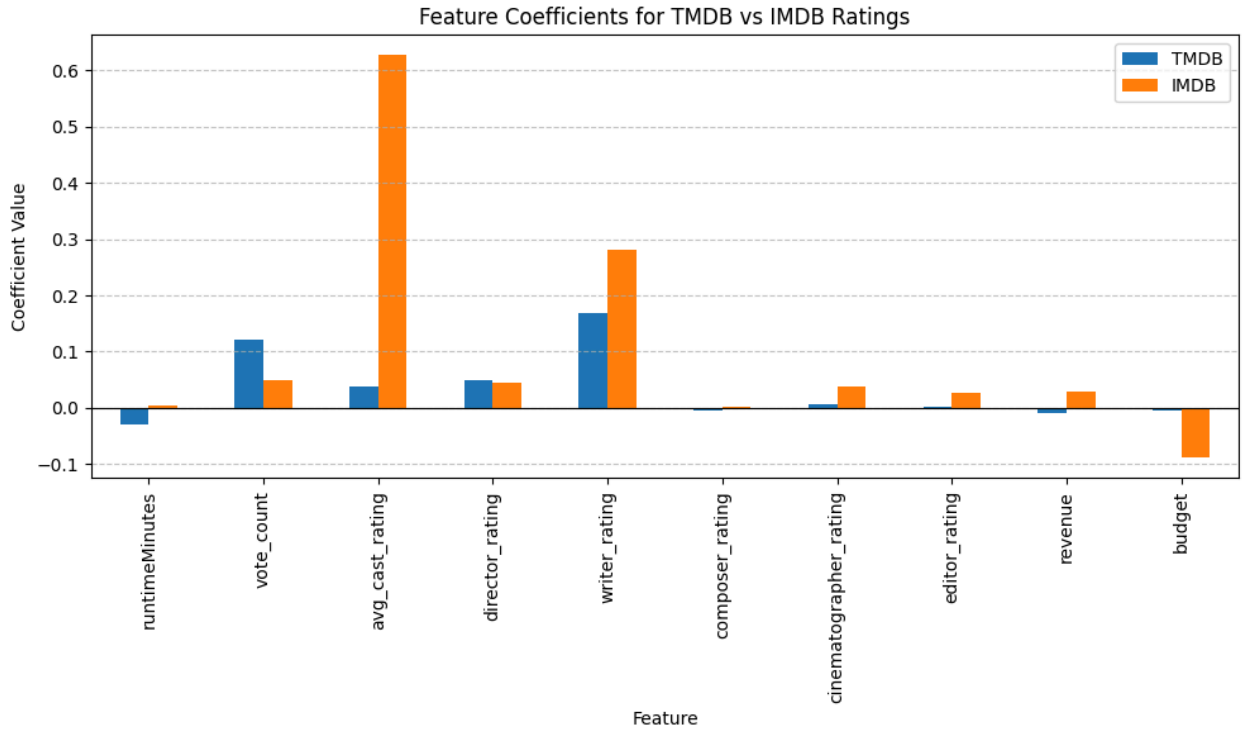
```
import pandas as pd
import matplotlib.pyplot as plt
```

```
import seaborn as sns

# Combine your coefficients into a dictionary
coef_data = {
    'Feature': ['runtimeMinutes', 'vote_count', 'avg_cast_rating', 'director_rat
                'composer_rating', 'cinematographer_rating', 'editor_rating', '
    'TMDB': [-0.029, 0.122, 0.0389, 0.050, 0.169, -0.005, 0.007, 0.003, -0.009,
    'IMDB': [0.004, 0.049, 0.627, 0.044, 0.280, 0.003, 0.039, 0.026, 0.030, -0.0
}

coef_df = pd.DataFrame(coef_data)
coef_df.set_index('Feature', inplace=True)
coef_df
```

Feature	TMDB	IMDB
runtimeMinutes	-0.0290	0.004
vote_count	0.1220	0.049
avg_cast_rating	0.0389	0.627
director_rating	0.0500	0.044
writer_rating	0.1690	0.280
composer_rating	-0.0050	0.003
cinematographer_rating	0.0070	0.039
editor_rating	0.0030	0.026
revenue	-0.0090	0.030
budget	-0.0040	-0.089

```
plt.figure(figsize=(10,6))
coef_df.plot(kind='bar', figsize=(10,6))
plt.title('Feature Coefficients for TMDB vs IMDB Ratings')
plt.xlabel('Feature')
plt.ylabel('Coefficient Value')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.axhline(0, color='black', linewidth=0.8)
plt.tight_layout()
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```



Correlation analysis revealed that vote count — a proxy for audience exposure — is the second strongest correlate of revenue after budget. This suggests that marketing reach and audience engagement play critical roles in driving financial success. However, regression results indicate that these same variables exhibit weak or even negative relationships with ratings, reinforcing a distinction between commercial popularity and perceived quality.

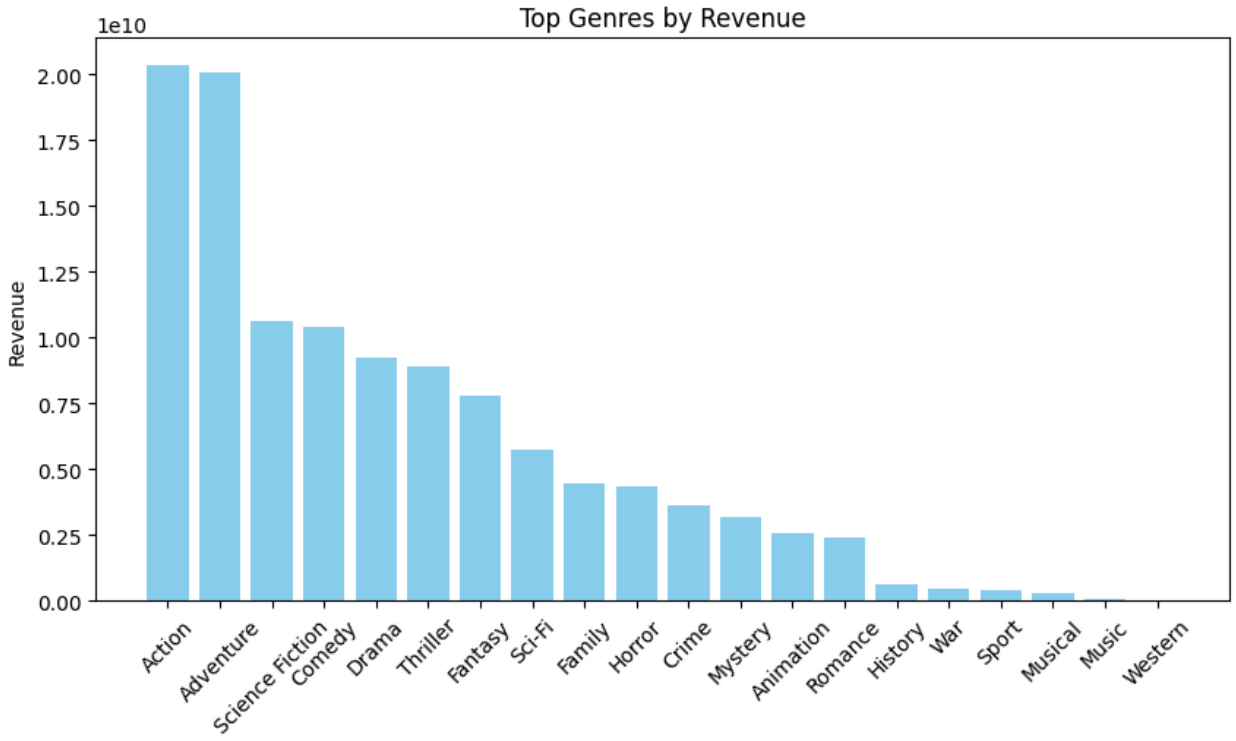
GENRE

```
# Split genres into lists
df['genre_list'] = df['genre'].str.split(', ')
df['num_genres'] = df['genre_list'].apply(len)

# Weighted revenue and rating
df['weighted_revenue'] = df['revenue'] / df['num_genres']
df['weighted_rating'] = df['imdb_rating'] / df['num_genres']
```

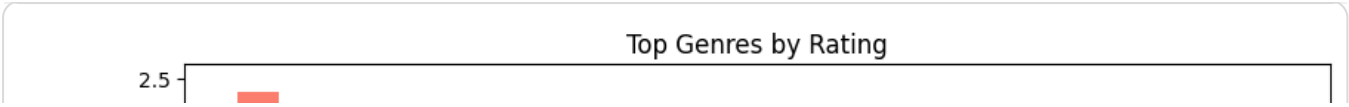
```
# Explode genre list directly (no ast.literal_eval needed)
df_exploded = df.explode('genre_list')

# Now group by genre
genre_stats = df_exploded.groupby('genre_list').agg({
    'weighted_revenue': 'sum',
    'weighted_rating': 'mean',
    'title': 'count'
}).rename(columns={'title': 'count'}).reset_index()

print(genre_stats)
```

```
         genre_list  weighted_revenue  weighted_rating  count
0            Action      2.037184e+10         1.726200    494
1         Adventure      2.008166e+10         1.658733    320
2         Animation      2.587620e+09         1.556088     49
3            Comedy      1.040809e+10         2.216503    382
4             Crime      3.617428e+09         1.773132    244
5             Drama      9.254728e+09         2.299779    590
6            Family      4.457768e+09         1.608559     96
7           Fantasy      7.790968e+09         1.640941    162
8           History      6.391790e+08         2.135106     63
9            Horror      4.372098e+09         1.863863    227
10            Music      1.032999e+08         2.052833     20
11          Musical      3.002503e+08         1.855556     15
12          Mystery      3.214570e+09         1.683930    193
13          Romance      2.440732e+09         2.238297    139
14           Sci-Fi      5.764069e+09         1.446505    103
15  Science Fiction      1.064165e+10         1.444155    188
16            Sport      4.297212e+08         2.441667     14
17         Thriller      8.897390e+09         1.784950    444
18         Tv Movie      0.000000e+00         1.080000      1
19              War      4.970583e+08         1.804550     37
20          Western      5.280927e+07         2.121667     13
/tmp/ipython-input-2851404681.py:2: SettingWithCopyWarning:
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/stable
  df['genre_list'] = df['genre'].str.split(', ')
/tmp/ipython-input-2851404681.py:3: SettingWithCopyWarning:
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/stable
  df['num_genres'] = df['genre_list'].apply(len)
/tmp/ipython-input-2851404681.py:6: SettingWithCopyWarning:
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/stable
  df['weighted_revenue'] = df['revenue'] / df['num_genres']
/tmp/ipython-input-2851404681.py:7: SettingWithCopyWarning:
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/stable
  df['weighted_rating'] = df['imdb_rating'] / df['num_genres']
```

```
# Top 10 by revenue
top_revenue = genre_stats.nlargest(20, 'weighted_revenue')

plt.figure(figsize=(10,5))
plt.bar(top_revenue['genre_list'], top_revenue['weighted_revenue'], color='skybl
plt.xticks(rotation=45)
plt.ylabel('Revenue')
plt.title('Top Genres by Revenue')
plt.show()
```




```
# Top 10 by rating after filtering
top_rating = genre_stats.nlargest(20, 'weighted_rating')

plt.figure(figsize=(10,5))
plt.bar(top_rating['genre_list'], top_rating['weighted_rating'], color='salmon')
plt.xticks(rotation=45)
plt.ylabel('Average Rating')
plt.title('Top Genres by Rating')
plt.show()
```


Top Genres by Rating
2.5