Exploratory Data Analysis of Netflix Movies Dataset

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Import the Movie Dataset and check it structure

In [2]:
```
movie_df=pd.read_csv('C:/Users/raviy/Downloads/netflix_movies_dataset_with_nul
movie_df.head(3)
```

Out[2]:

	show_id	type	title	director	cast	country	date_added	release_year	rating
0	10000	Movie	Movie_0	Director_248	Actor_1000, Actor_1744	United Kingdom	01/07/2013	2018	8.8
1	10001	Movie	Movie_1	Director_362	Actor_269, Actor_1084	United Kingdom	12/17/2013	2004	8.6
2	10002	Movie	Movie_2	Director_441	Actor_871, Actor_1822	United Kingdom	12/31/2022	1997	5.6

Shape of the Dataset

In [3]:
```
print(f'The Total Number of Columns is {movie_df.shape[1]}')
print(f'The Total Number of Rows is {movie_df.shape[0]}')
```

```
The Total Number of Columns is 18
The Total Number of Rows is 16000
```

Check the Null Values

In [4]:
```
movie_df.isnull().sum()
```

Out[4]:
```
show_id            0
type               0
title              0
director         132
cast             204
country          466
date_added         0
release_year       0
rating             0
duration       16000
genres           107
language           0
description      132
popularity         0
vote_count         0
vote_average       0
budget             0
revenue            0
dtype: int64
```

Replace missing values in 'director', 'cast', 'country', and 'genres' columns

In [5]:
```
movie_df.fillna({'director':'Unknown','cast':'Unknown','country':'Unknown','ge
# Drop unnecessary columns 'duration' and 'description' from the dataframe
movie_df.drop(columns=['duration', 'description'], inplace=True, errors='ignor
```

after fill the value and drop value

In [6]:
```
movie_df.isnull().sum()
```

Out[6]:
```
show_id         0
type            0
title           0
director        0
cast            0
country         0
date_added      0
release_year    0
rating          0
genres          0
language        0
popularity      0
vote_count      0
vote_average    0
budget          0
revenue         0
dtype: int64
```

Check the number of duplicate rows in the dataset

In [7]:
```
movie_df.duplicated().sum()
```

Out[7]:
```
0
```

Check the descriptive statistics

In [8]:
```
movie_df.describe()
```

Out[8]:

	show_id	release_year	rating	popularity	vote_count	vote_average	
count	16000.000000	16000.000000	16000.000000	16000.000000	16000.000000	16000.000000	1.6
mean	17999.500000	2006.516437	5.524087	49.838811	25057.025313	5.508706	9.9
std	4618.946489	9.721427	2.587853	28.898558	14379.931396	2.585147	5.7
min	10000.000000	1990.000000	1.000000	0.000000	11.000000	1.000000	1.0
25%	13999.750000	1998.000000	3.300000	24.670000	12636.750000	3.300000	5.0
50%	17999.500000	2006.500000	5.600000	49.720000	25033.000000	5.500000	9.9
75%	21999.250000	2015.000000	7.700000	74.882500	37474.000000	7.800000	1.4
max	25999.000000	2023.000000	10.000000	100.000000	49987.000000	10.000000	1.9

Check the data type and non-null count of all columns in the dataset

In [9]:
```
movie_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16000 entries, 0 to 15999
Data columns (total 16 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   show_id       16000 non-null  int64  
 1   type          16000 non-null  object 
 2   title         16000 non-null  object 
 3   director      16000 non-null  object 
 4   cast          16000 non-null  object 
 5   country       16000 non-null  object 
 6   date_added    16000 non-null  object 
 7   release_year  16000 non-null  int64  
 8   rating        16000 non-null  float64
 9   genres        16000 non-null  object 
 10  language      16000 non-null  object 
 11  popularity    16000 non-null  float64
 12  vote_count    16000 non-null  int64  
 13  vote_average  16000 non-null  float64
 14  budget        16000 non-null  int64  
 15  revenue       16000 non-null  int64  
dtypes: float64(3), int64(5), object(8)
memory usage: 2.0+ MB
```

Convert the 'date_added' column from object type to datetime format

In [10]:
```
movie_df['date_added'] = pd.to_datetime(movie_df['date_added'])
```

After converting, check the data type of 'date_added' column

In [11]:
```
movie_df['date_added'].dtype
```

Out[11]:
```
dtype('<M8[ns]')
```

Check the number of unique values in each column

In [12]:
```
movie_df.nunique()
```

Out[12]:
```
show_id         16000
type                1
title           16000
director          501
cast            15677
country             5
date_added       4790
release_year       34
rating             91
genres              6
language            3
popularity       7970
vote_count      13719
vote_average       91
budget          16000
revenue         16000
dtype: int64
```

Add a new column 'profit' by subtracting 'budget' from 'revenue'

In [13]:
```
movie_df['profit'] = movie_df['revenue'] - movie_df['budget']
```

In [14]:
```
movie_df['profit']
```

Out[14]:
```
0        189065380
1        701352475
2        572585442
3        549357968
4         14980667
           ...
15995     81710348
15996    602858241
15997     29394271
15998    799320752
15999    347884698
Name: profit, Length: 16000, dtype: int64
```

Identifying Top Movies Based on Key Metrics

In [15]:
```
# Top 10 Movies by Revenue
top10_revenue = movie_df[['title', 'revenue']].sort_values(by='revenue', ascer
# Top 10 Movies by Budget
top10_budget = movie_df[['title', 'budget']].sort_values(by='budget', ascendir
# Top 10 Movies by Popularity
top10_popularity = movie_df[['title', 'popularity']].sort_values(by='popularit
# Top 10 Profitable Movies
top10_profitable = movie_df[['title', 'profit']].sort_values(by='profit',  asc
```

In [16]:
```
top10_revenue
top10_budget
```

Out[16]:

	title	budget
6806	Movie_6806	199999488
6747	Movie_6747	199997440
9479	Movie_9479	199989892
13668	Movie_13668	199984450
15515	Movie_15515	199973304
9105	Movie_9105	199953337
8922	Movie_8922	199899349
15364	Movie_15364	199897661
12931	Movie_12931	199896471
13816	Movie_13816	199878347

Top 10 Most Expensive Movies (by Budget)

In [17]:
```
# For Budget
top10_budget.reset_index(drop=True)
```

Out[17]:

	title	budget
0	Movie_6806	199999488
1	Movie_6747	199997440
2	Movie_9479	199989892
3	Movie_13668	199984450
4	Movie_15515	199973304
5	Movie_9105	199953337
6	Movie_8922	199899349
7	Movie_15364	199897661
8	Movie_12931	199896471
9	Movie_13816	199878347

Top 10 Movies with the Highest Revenue

In [18]:
```
# For Revenue
top10_revenue.reset_index(drop=True)
```

Out[18]:

	title	revenue
0	Movie_15725	999992343
1	Movie_874	999883507
2	Movie_13805	999846590
3	Movie_10547	999846396
4	Movie_15005	999810535
5	Movie_12809	999759645
6	Movie_5751	999733443
7	Movie_11348	999575346
8	Movie_12379	999490269
9	Movie_7235	999416756

Top 10 Most Popular Movies

In [19]:
```
# For Popular Movies
top10_popularity.reset_index(drop=True)
```

Out[19]:

	title	popularity
0	Movie_12662	100.00
1	Movie_15848	100.00
2	Movie_10308	99.99
3	Movie_2727	99.98
4	Movie_15113	99.98
5	Movie_261	99.97
6	Movie_5967	99.93
7	Movie_3950	99.92
8	Movie_14382	99.92
9	Movie_5984	99.92

Top 10 Profitabel Movies

In [20]:
```
# For Profitable Movies
top10_profitable.reset_index(drop=True)
```

Out[20]:

	title	profit
0	Movie_12379	997289341
1	Movie_9677	995328378
2	Movie_14882	994295929
3	Movie_10077	993049843
4	Movie_4815	992844406
5	Movie_15014	992240510
6	Movie_9101	991598552
7	Movie_6462	991188310
8	Movie_12959	990373901
9	Movie_1688	989399656

Global Metrics Distribution

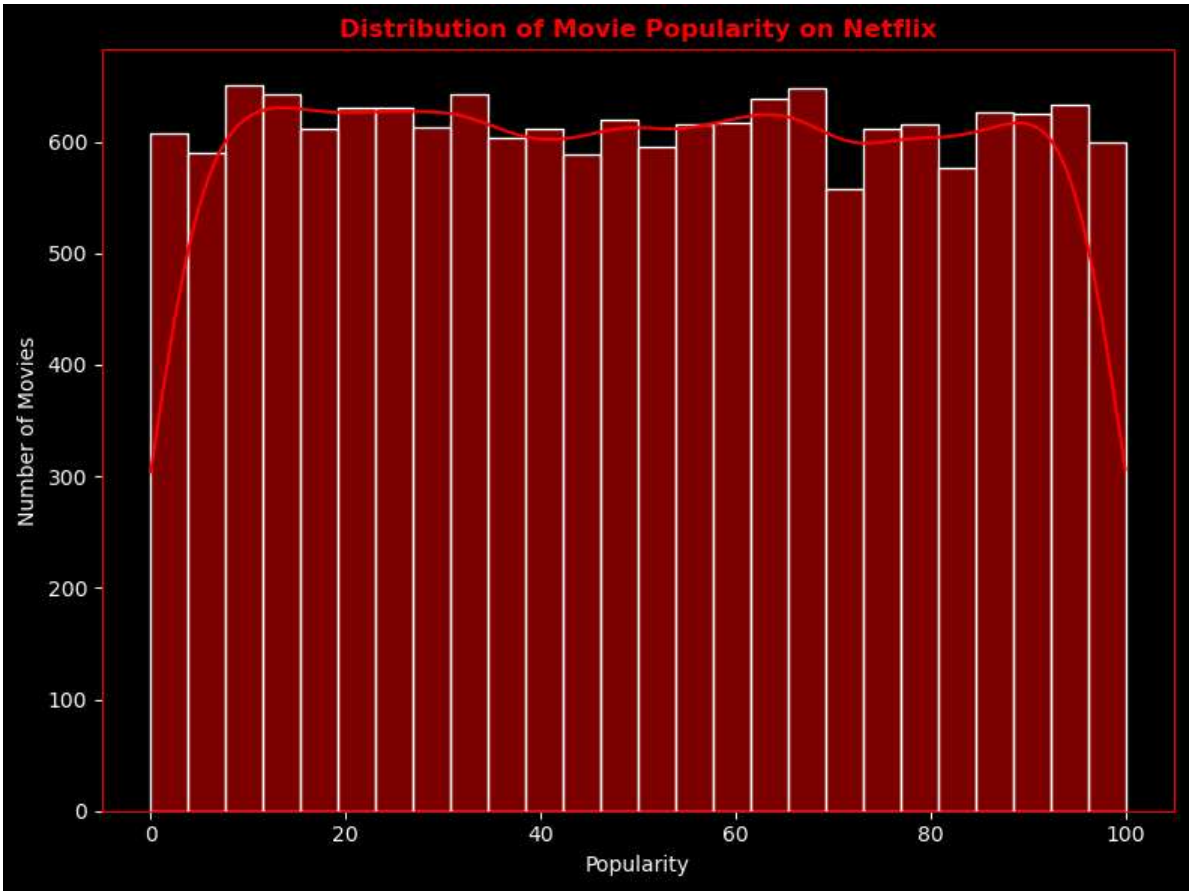
Set the Visual Theme for All Charts

In [21]:
```
# Global customization for all plots
plt.style.use('dark_background')
plt.rcParams.update({
'figure.facecolor': 'black', # Background of figure
'axes.facecolor': 'black', # Background of plot area
'axes.edgecolor': 'red', # Axis border color
'axes.labelcolor': 'white', # Axis label color
'xtick.color': 'white', # X-tick color
'ytick.color': 'white', # Y-tick color
'text.color': 'white', # Default text color
'axes.titleweight': 'bold', # Bold titles
'axes.titlecolor': 'red', # Title color (Netflix red)
'axes.prop_cycle': plt.cycler(color=['red', 'white', 'grey']),
'figure.figsize': (8, 6)# Line/bar colors
})
# Change to Y-Axis in Million
from matplotlib.ticker import FuncFormatter
def millions(x, pos):
    return f'{x/1_000_000:.1f}M'
```

Distribution of Movie Popularity Scores on Netflix

In [22]:
```
# Distribution of Moview Popularity
sns.histplot(x='popularity', data=movie_df[movie_df['popularity'] < 100], kde=
plt.xlabel('Popularity',)
plt.ylabel('Number of Movies')
plt.title('Distribution of Movie Popularity on Netflix')
plt.tight_layout()
plt.show()
```

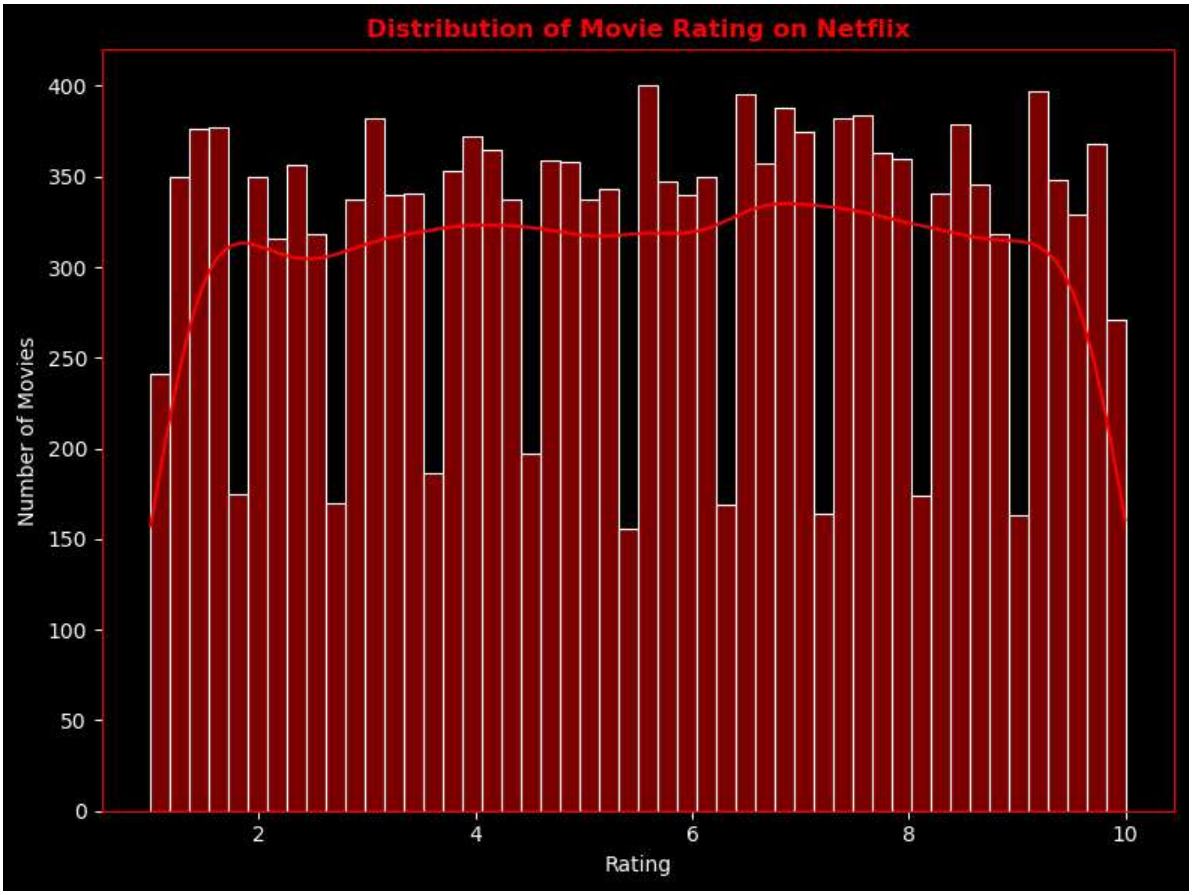
```
C:\Users\raviy\anaconda3\lib\site-packages\seaborn\_oldcore.py:1119: FutureW
arning: use_inf_as_na option is deprecated and will be removed in a future v
ersion. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



Distribution of Movie Ratings on Netflix

In [23]:
```
# Distribution of Movie Ratings
sns.histplot(x='rating', data=movie_df[movie_df['rating'] >0], kde=True, bins=
plt.xlabel('Rating')
plt.ylabel('Number of Movies')
plt.title('Distribution of Movie Rating on Netflix')
plt.tight_layout()
plt.show()
```

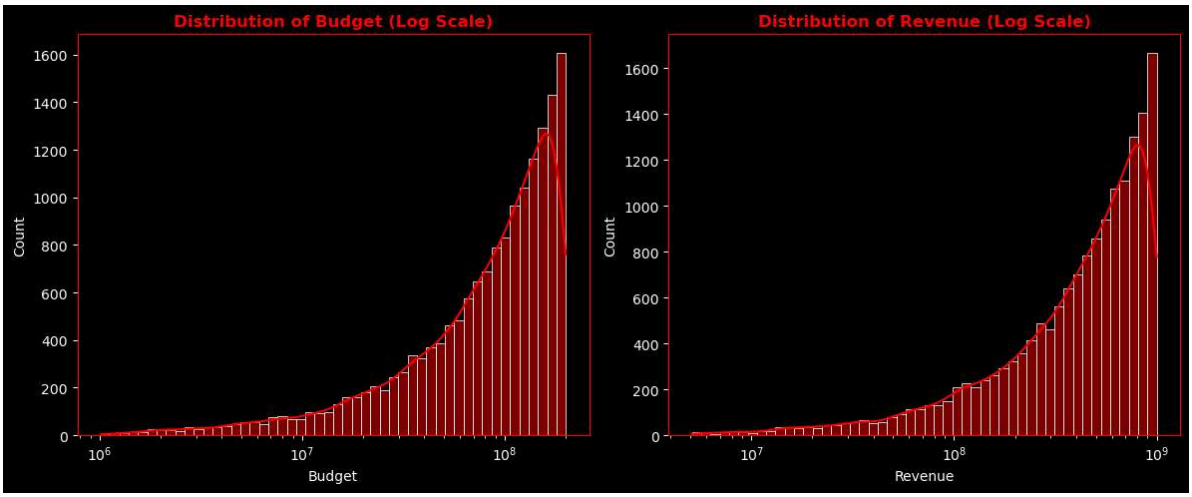
```
C:\Users\raviy\anaconda3\lib\site-packages\seaborn\_oldcore.py:1119: FutureW
arning: use_inf_as_na option is deprecated and will be removed in a future v
ersion. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



Budget vs Revenue Distributions on Netflix Movies (Log Scale)

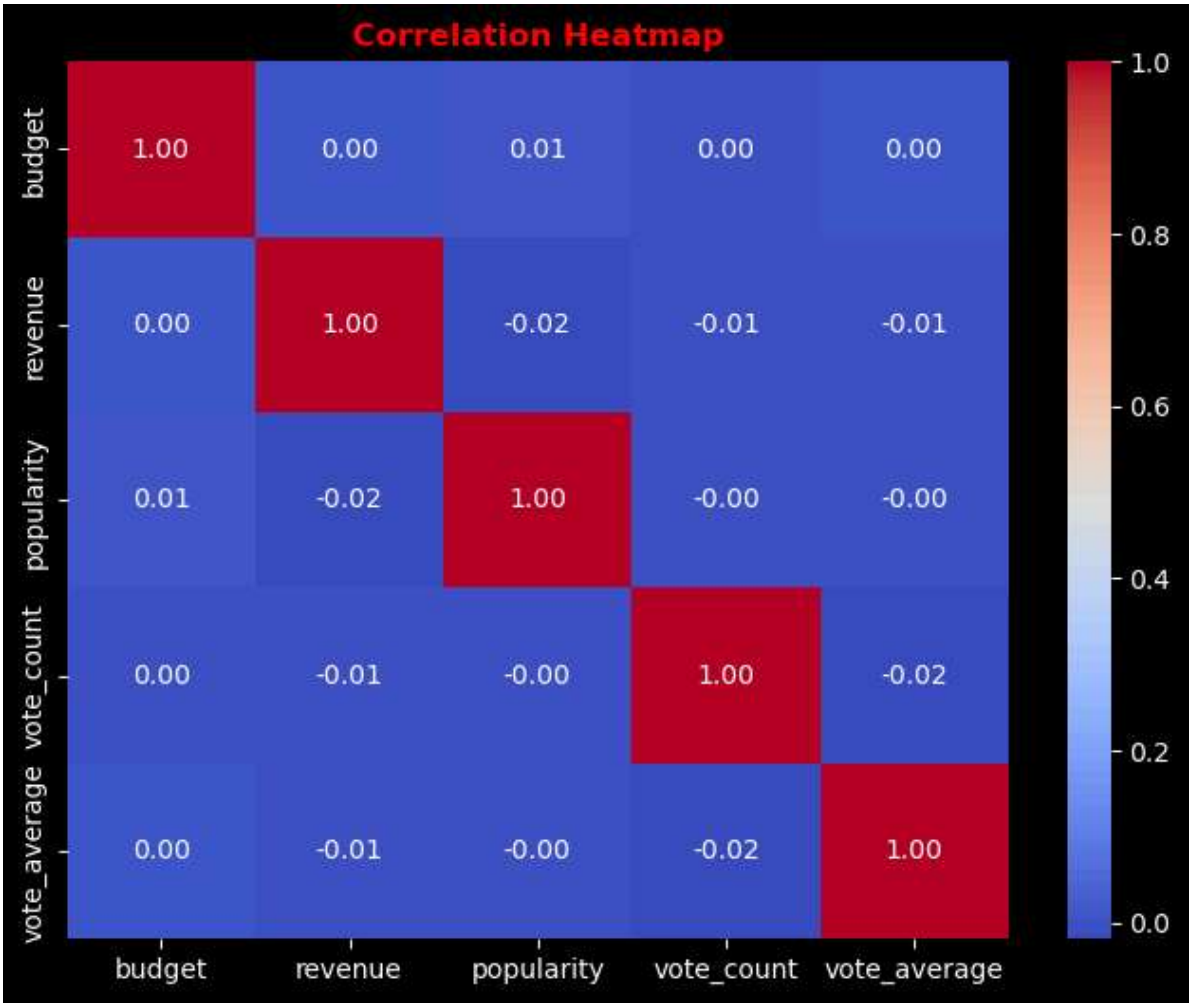
In [24]:
```
# Make sure budget & revenue > 0 for log scale
df_plot = movie_df[(movie_df['budget'] > 0) & (movie_df['revenue'] > 0)]
# Figure size
plt.figure(figsize=(12,5))
# Budget distribution
plt.subplot(1,2,1)
sns.histplot(df_plot['budget'], bins=50, log_scale=True,kde=True)
plt.title('Distribution of Budget (Log Scale)')
plt.xlabel('Budget')
plt.ylabel('Count')
# Revenue distribution
plt.subplot(1,2,2)
sns.histplot(df_plot['revenue'], bins=50, log_scale=True,kde=True)
plt.title('Distribution of Revenue (Log Scale)')
plt.xlabel('Revenue')
plt.ylabel('Count')
plt.tight_layout()
plt.show()
```

```
C:\Users\raviy\anaconda3\lib\site-packages\seaborn\_oldcore.py:1119: FutureW
arning: use_inf_as_na option is deprecated and will be removed in a future v
ersion. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\raviy\anaconda3\lib\site-packages\seaborn\_oldcore.py:1119: FutureW
arning: use_inf_as_na option is deprecated and will be removed in a future v
ersion. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



Relationship Between Budget, Revenue, Popularity and Ratings

In [25]:
```
# Select the relevant columns
corr_cols = ['budget', 'revenue', 'popularity', 'vote_count', 'vote_average']
corr = movie_df[corr_cols].corr() # correlation matrix
#Plot
sns.heatmap(corr, annot=True, fmt='.2f', cmap='coolwarm', cbar=True)
plt.title('Correlation Heatmap')
plt.show()
```



Genre Analysis

Splitting and Exploding the Genres Column

In [26]:
```
# Spilting & Exploding
movie_df['genres'] = movie_df['genres'].astype(str).str.split(',')
movie_df_exploded = movie_df.explode('genres')
movie_df_exploded['genres'] = movie_df_exploded['genres'].astype(str).str.stri
```

In [27]:
```
# Check the DataFrame after exploding to verify successful genre separation
movie_df_exploded['genres'].head()
```

Out[27]:
```
0    Action
1     Drama
2    Action
3    Comedy
4    Comedy
Name: genres, dtype: object
```

In [28]:
```
# Group by 'genres' to get count and average metrics for easier genre-wise ana
genres_group = (
movie_df_exploded.groupby('genres')
.agg(
number_of_movies = ('show_id','count'),
avg_rating = ('rating','mean'),
avg_popularity = ('popularity','mean'),
avg_vote_count = ('vote_count','mean'),
avg_budget = ('budget','mean'),
avg_revenue = ('revenue','mean'),
avg_profit = ('profit','mean')
).sort_values(by='number_of_movies', ascending=False).reset_index())
```

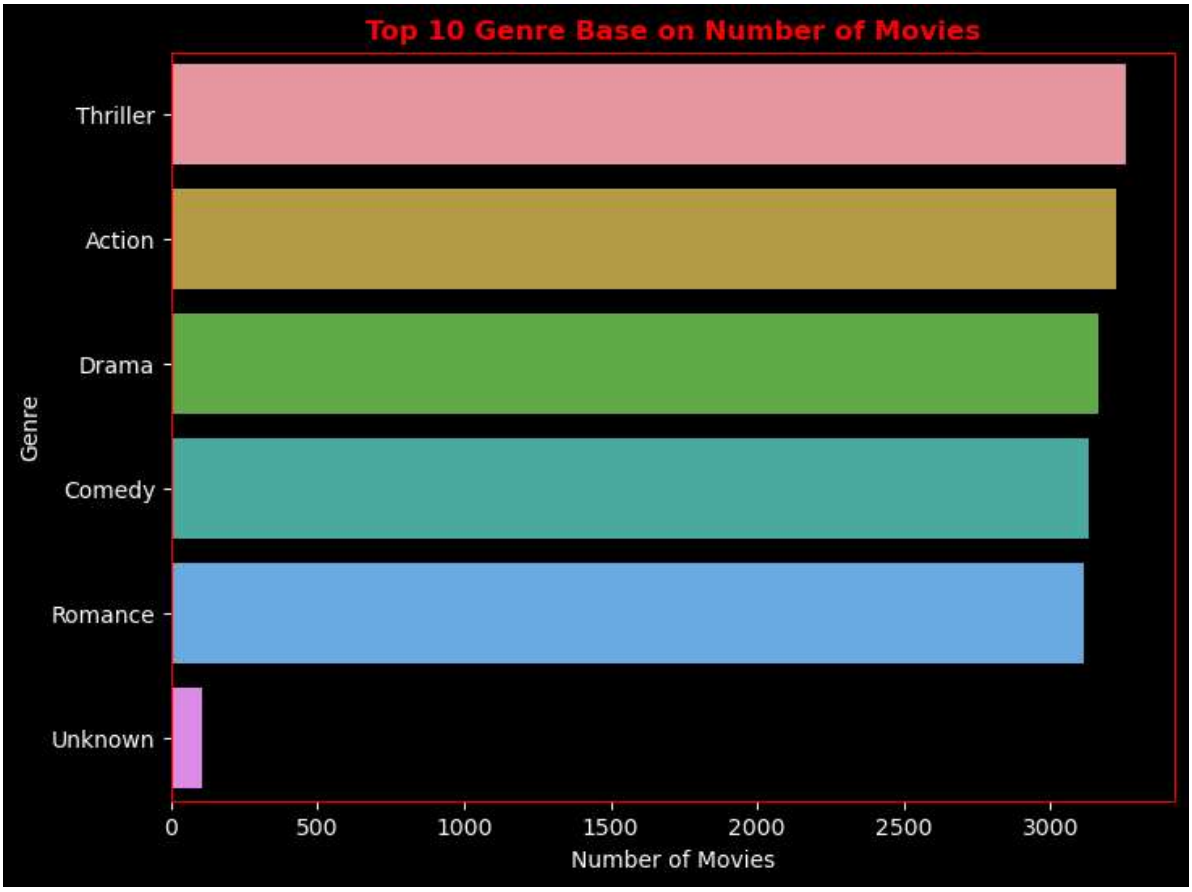
In [29]:
```
genres_group.head()
```

Out[29]:

	genres	number_of_movies	avg_rating	avg_popularity	avg_vote_count	avg_budget	avg
0	Thriller	3260	5.477178	50.425712	25244.595706	9.862793e+07	4.96
1	Action	3226	5.558617	49.612207	25016.506510	1.016876e+08	5.03
2	Drama	3163	5.482042	49.595447	24916.877964	9.956803e+07	4.97
3	Comedy	3130	5.634888	50.257674	24607.650160	1.004723e+08	5.02
4	Romance	3114	5.466249	49.399753	25449.135196	9.971911e+07	5.08

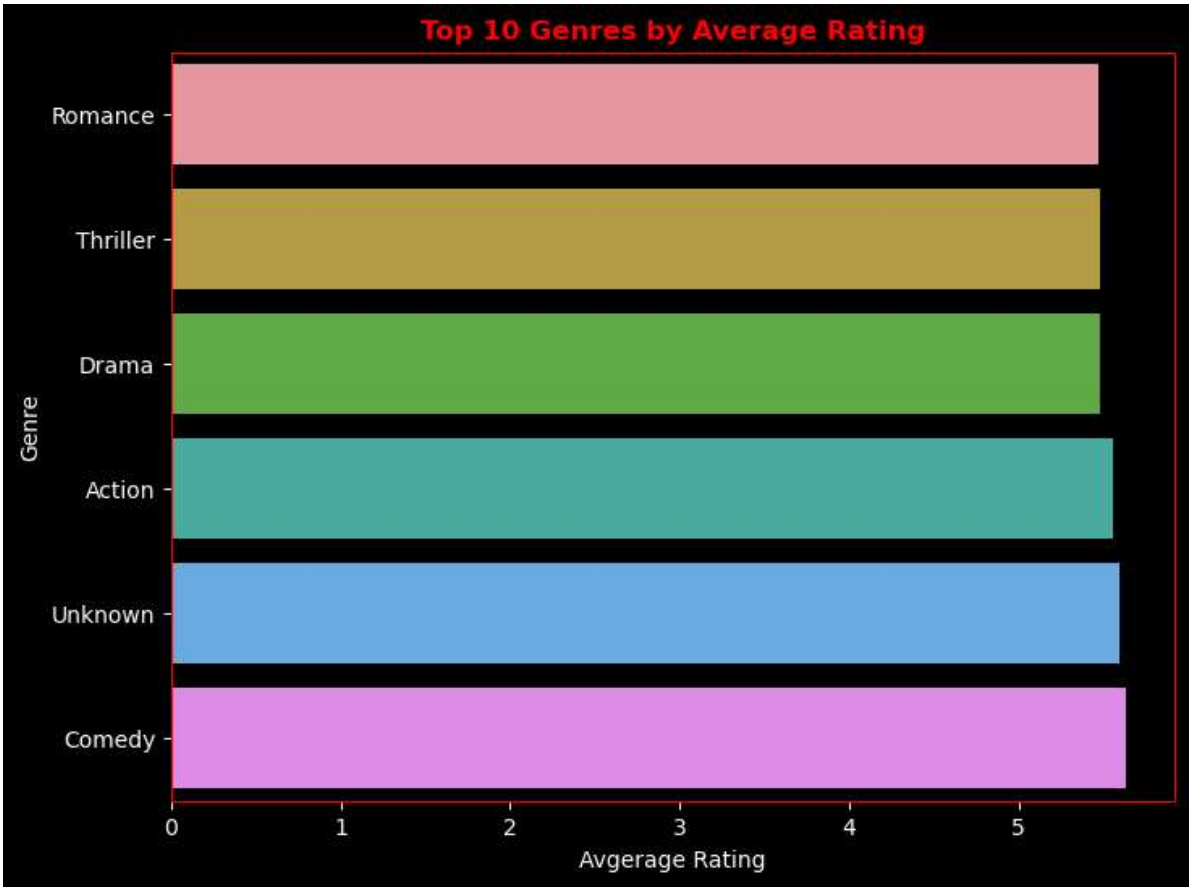
Top 10 Genre Base on Number of Movies

In [30]:
```
# Plot top 10 genres based on number of movies
sns.barplot(x='number_of_movies', y='genres', data=genres_group.head(10))
plt.xlabel("Number of Movies")
plt.ylabel("Genre")
plt.title("Top 10 Genre Base on Number of Movies")
plt.show()
```



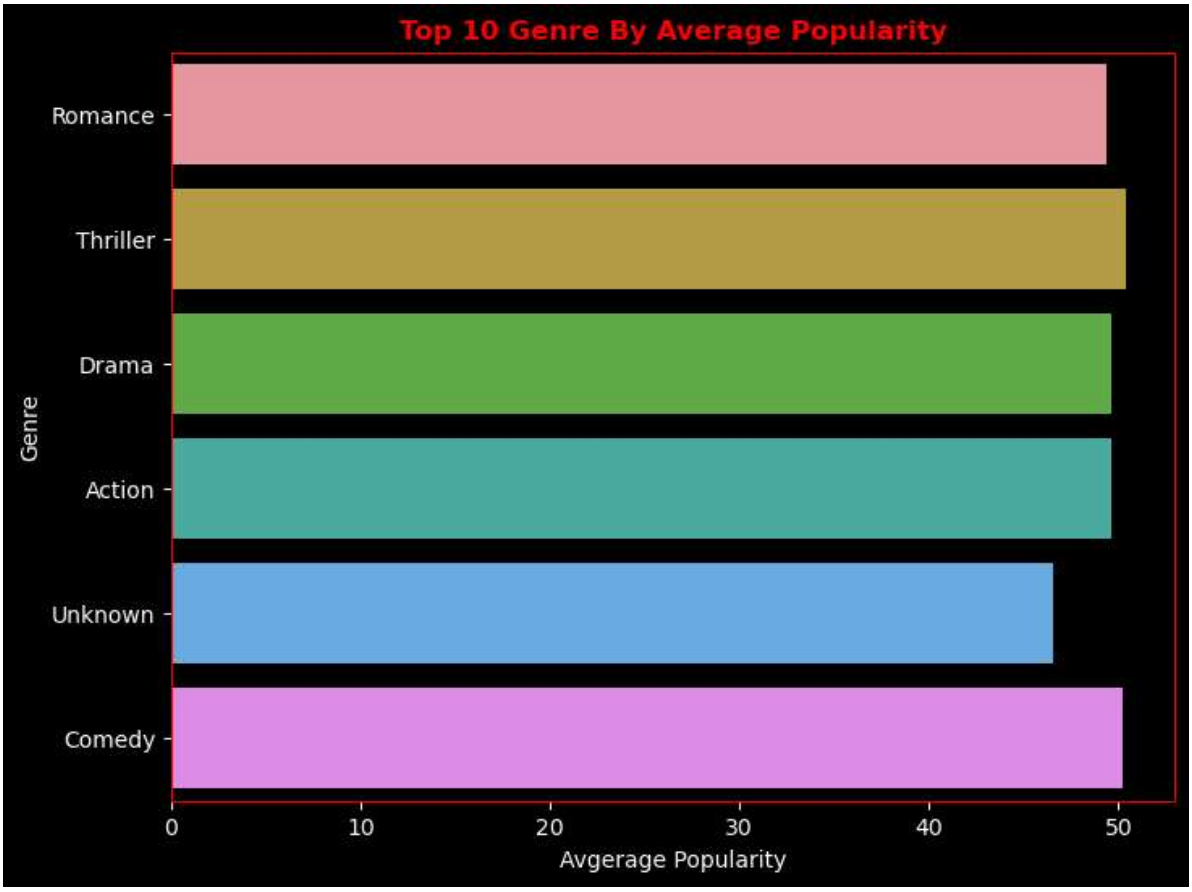
Top 10 Genres by Average Rating

In [31]:
```
# Plot top 10 genres based on average rating
sns.barplot(x='avg_rating', y='genres', data=genres_group.sort_values(by='avg_
plt.xlabel("Avgerage Rating")
plt.ylabel("Genre")
plt.title("Top 10 Genres by Average Rating")
plt.show()
```



Top 10 Genres by Average Popularity

In [32]:
```
# Plot top 10 genres based on Average Popularity
sns.barplot(x='avg_popularity', y='genres', data=genres_group.sort_values(by='
plt.xlabel("Avgerage Popularity")
plt.ylabel("Genre")
plt.title("Top 10 Genre By Average Popularity")
plt.show()
```



Country Analysis

In [33]:
```
# Country-wise explode with primary country and equal profit distribution
# Split country column into lists
movie_df['country'] = movie_df['country'].astype(str).str.split(',')
# Create primary country (take the first one)
movie_df['primary_country'] = movie_df['country'].apply(
lambda x: x[0].strip() if isinstance(x, list) else x)
# Explode so each country gets its own row
movie_df_exploded = movie_df.explode('country')
# Remove extra spaces from country names
movie_df_exploded['country'] = movie_df_exploded['country'].str.strip()
# Divide profit equally among all countries for each movie
movie_df_exploded['profit_divided'] = (
movie_df_exploded['profit'] /
movie_df_exploded.groupby('show_id')['country'].transform('count'))
```

In [34]:
```
# Check the DataFrame after exploding and add primary columhn
movie_df_exploded.iloc[:,[5,17]].head()
```

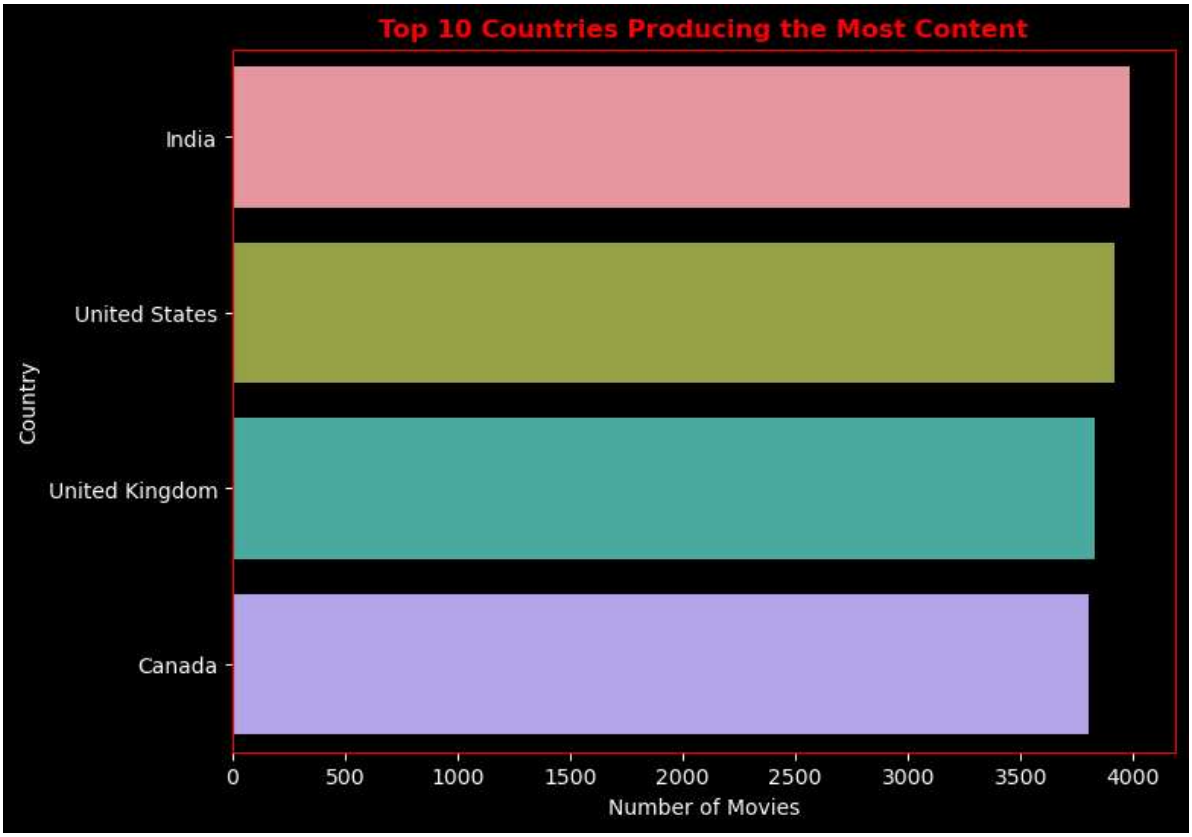
Out[34]:

	country	primary_country
0	United Kingdom	United Kingdom
1	United Kingdom	United Kingdom
2	United Kingdom	United Kingdom
3	United Kingdom	United Kingdom
4	India	India

In [35]:
```
# Group by 'country' to get count and average metrics for easier country-wise
country_group = (
movie_df_exploded[movie_df_exploded['primary_country']!='Unknown'].groupby('cc
.agg(
number_of_movies = ('show_id','count'),
avg_rating = ('rating','mean'),
avg_popularity = ('popularity','mean'),
avg_vote_count = ('vote_count','mean'),
avg_budget = ('budget','mean'),
avg_revenue = ('revenue','mean'),
avg_profit = ('profit','mean')
).reset_index().sort_values(by='number_of_movies', ascending=False))
# Filter countries with at least 500 movies (to avoid bias from countries with
filtered_countries = country_group[country_group['number_of_movies'] >= 500]
```

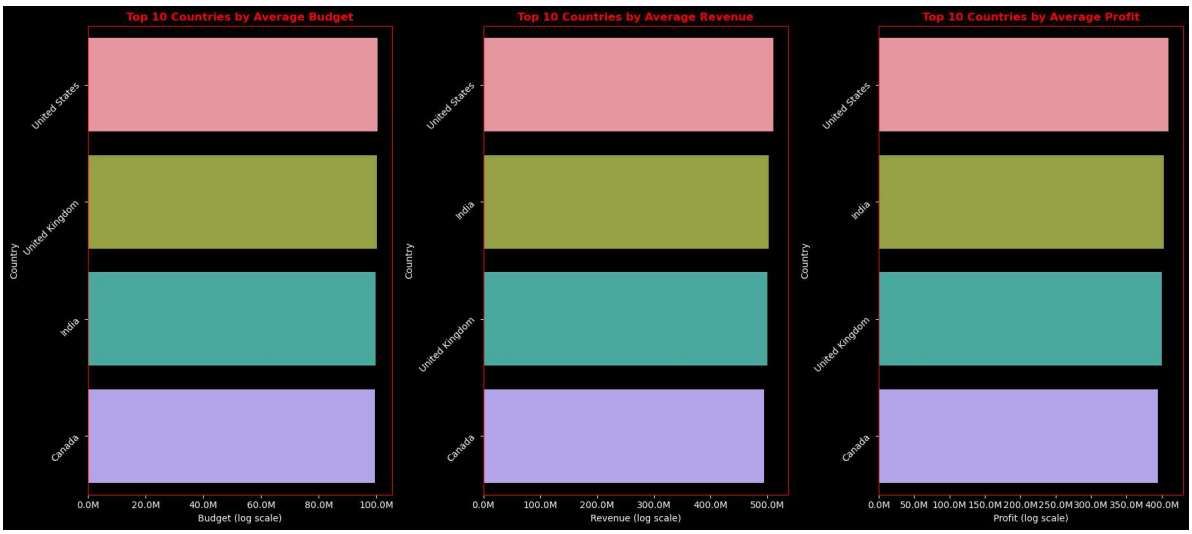
Top Producing Countries

In [38]:
```
sns.barplot(x='number_of_movies', y='country', data=filtered_countries.head(10
plt.xlabel("Number of Movies")
plt.ylabel("Country")
plt.title("Top 10 Countries Producing the Most Content")
plt.show()
```



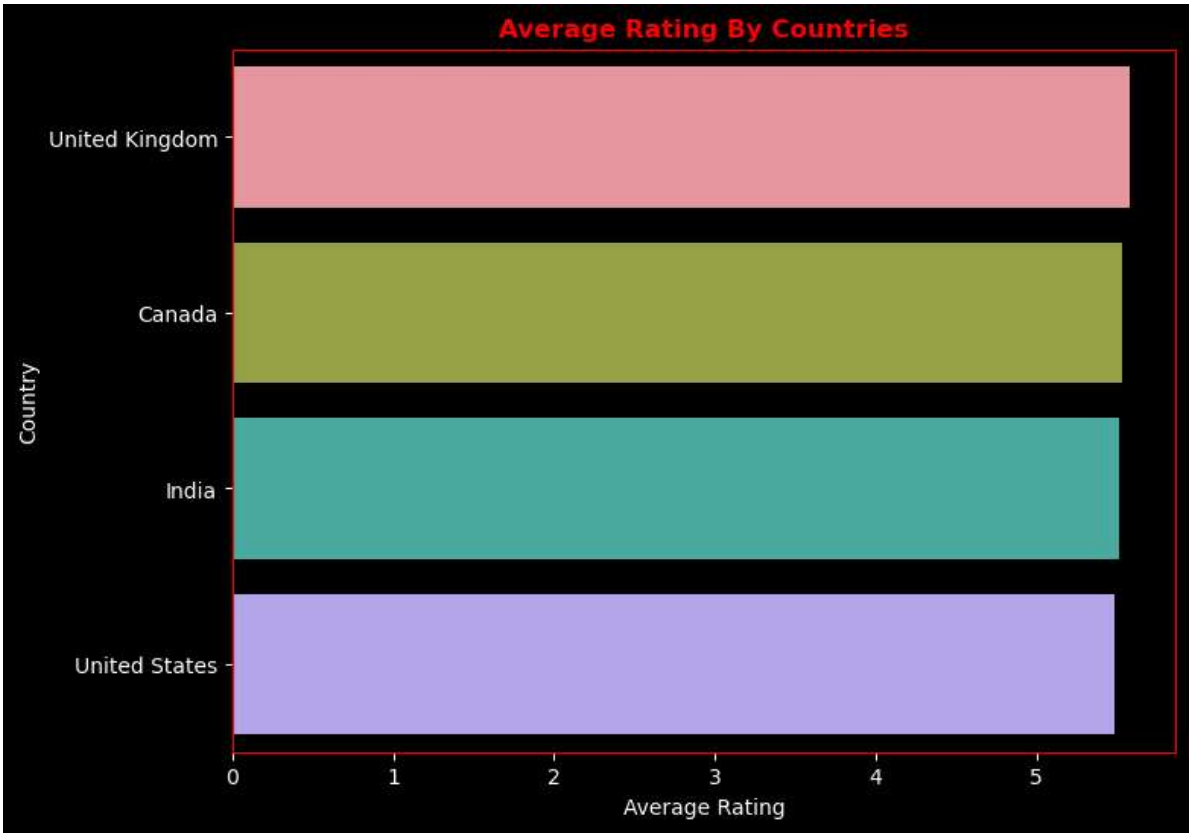
Compare Budget, Revenue and Profit by Countries

In [41]:
```
# Create side-by-side subplots to compare average revenue and average budget b
fig, axes = plt.subplots(1, 3, figsize=(18, 8))
for ax in axes:
    plt.setp(ax.get_yticklabels(), rotation=45)
# Change to Y-Axis in Million
from matplotlib.ticker import FuncFormatter
def millions(x, pos):
    return f'{x/1_000_000:.1f}M'
# Budget Plot
sns.barplot(
x='avg_budget',
y='country',
data=filtered_countries.sort_values(by='avg_budget', ascending=False).head(10)
ax=axes[0]
)
axes[0].set_title("Top 10 Countries by Average Budget")
axes[0].set_xlabel("Budget (log scale)")
axes[0].set_ylabel("Country")
axes[0].xaxis.set_major_formatter(FuncFormatter(millions))
# Revenue Plot
sns.barplot(x='avg_revenue',
y='country',
data=filtered_countries.sort_values(by='avg_revenue', ascending=False).head(10
ax=axes[1]
)
axes[1].set_title("Top 10 Countries by Average Revenue")
axes[1].set_xlabel("Revenue (log scale)")
axes[1].set_ylabel("Country")
axes[1].xaxis.set_major_formatter(FuncFormatter(millions))
# Profit Plot
sns.barplot(
x='avg_profit',
y='country',
data=filtered_countries.sort_values(by='avg_profit', ascending=False).head(10)
ax=axes[2]
)
axes[2].set_title("Top 10 Countries by Average Profit")
axes[2].set_xlabel("Profit (log scale)")
axes[2].set_ylabel("Country")
axes[2].xaxis.set_major_formatter(FuncFormatter(millions))
plt.tight_layout()
plt.show()
```



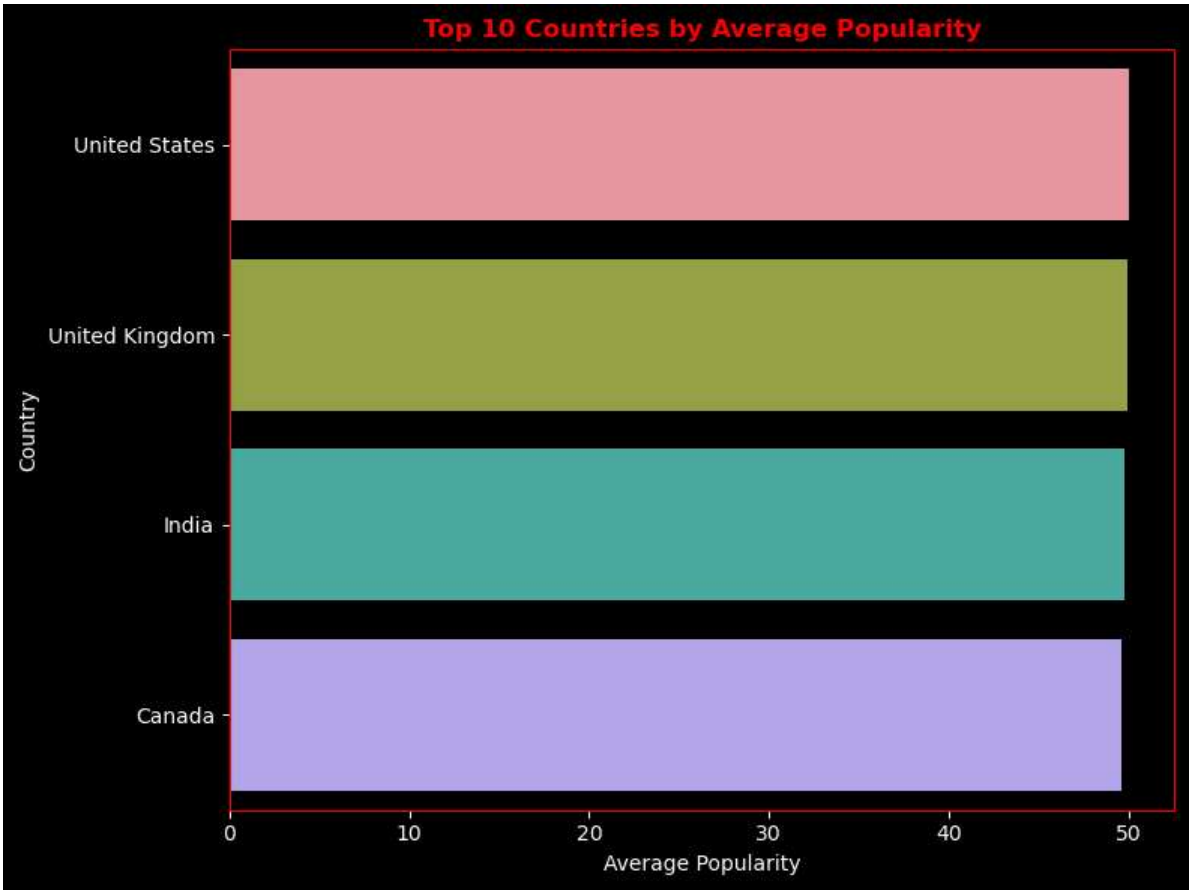
Top Average Rating Countries

In [42]:
```
# Average Rating Countries
sns.barplot(x='avg_rating',y='country',data=filtered_countries.sort_values(by=
plt.xlabel('Average Rating')
plt.ylabel('Country')
plt.title('Average Rating By Countries')
plt.show()
```



Top Average Popularity Countries

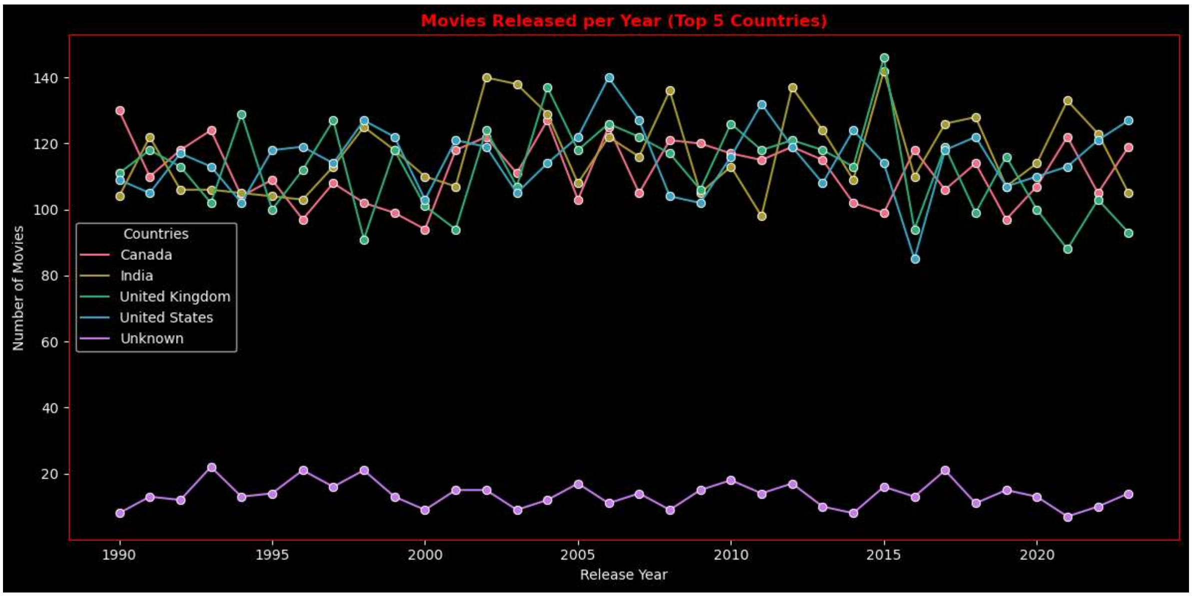
In [43]:
```
# Average Popularity Countries
sns.barplot(
    x='avg_popularity',
    y='country',
    data=filtered_countries.sort_values(
        by='avg_popularity', ascending=False
    ).head(10)
)

plt.xlabel('Average Popularity')
plt.ylabel('Country')
plt.title('Top 10 Countries by Average Popularity')
plt.tight_layout()
plt.show()
```



Movie Release Trend of Top 5 Countries

In [47]:
```
# Top 5 Countries by number of movies
top5_countries = (
    movie_df_exploded
    .groupby('primary_country')['show_id']
    .count()
    .sort_values(ascending=False)
    .head(5)
    .index
)

# Filter dataframe for top 5 countries
df_top5 = movie_df_exploded[
    movie_df_exploded['primary_country'].isin(top5_countries)
]

# Group by release year and country
trend_df = (
    df_top5
    .groupby(['release_year', 'primary_country'], as_index=False)
    .agg(number_of_movies=('show_id', 'count'))
)

# Plot
plt.figure(figsize=(12, 6))
sns.lineplot(
    x='release_year',
    y='number_of_movies',
    hue='primary_country',
    data=trend_df,
    marker='o'
)

plt.title('Movies Released per Year (Top 5 Countries)')
plt.xlabel('Release Year')
plt.ylabel('Number of Movies')
plt.legend(title='Countries')
plt.tight_layout()
plt.show()
```

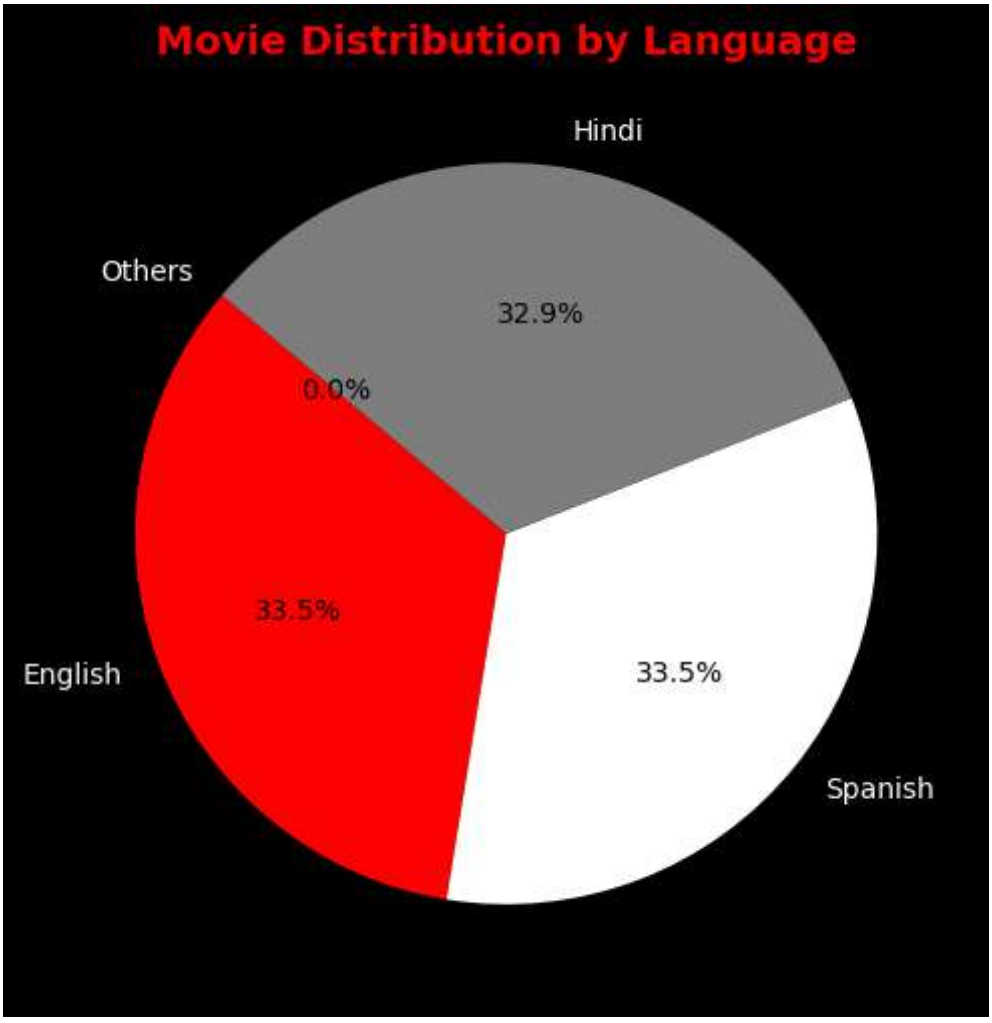
```
C:\Users\raviy\anaconda3\lib\site-packages\seaborn\_oldcore.py:1119: FutureW
arning: use_inf_as_na option is deprecated and will be removed in a future v
ersion. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\raviy\anaconda3\lib\site-packages\seaborn\_oldcore.py:1119: FutureW
arning: use_inf_as_na option is deprecated and will be removed in a future v
ersion. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\raviy\anaconda3\lib\site-packages\seaborn\_oldcore.py:1075: FutureW
arning: When grouping with a length-1 list-like, you will need to pass a len
gth-1 tuple to get_group in a future version of pandas. Pass `(name,)` inste
ad of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
C:\Users\raviy\anaconda3\lib\site-packages\seaborn\_oldcore.py:1075: FutureW
arning: When grouping with a length-1 list-like, you will need to pass a len
gth-1 tuple to get_group in a future version of pandas. Pass `(name,)` inste
ad of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
C:\Users\raviy\anaconda3\lib\site-packages\seaborn\_oldcore.py:1075: FutureW
arning: When grouping with a length-1 list-like, you will need to pass a len
gth-1 tuple to get_group in a future version of pandas. Pass `(name,)` inste
ad of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
C:\Users\raviy\anaconda3\lib\site-packages\seaborn\_oldcore.py:1075: FutureW
arning: When grouping with a length-1 list-like, you will need to pass a len
gth-1 tuple to get_group in a future version of pandas. Pass `(name,)` inste
ad of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
C:\Users\raviy\anaconda3\lib\site-packages\seaborn\_oldcore.py:1075: FutureW
arning: When grouping with a length-1 list-like, you will need to pass a len
gth-1 tuple to get_group in a future version of pandas. Pass `(name,)` inste
ad of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
```



Language Analysis

Language-wise Movie Distribution

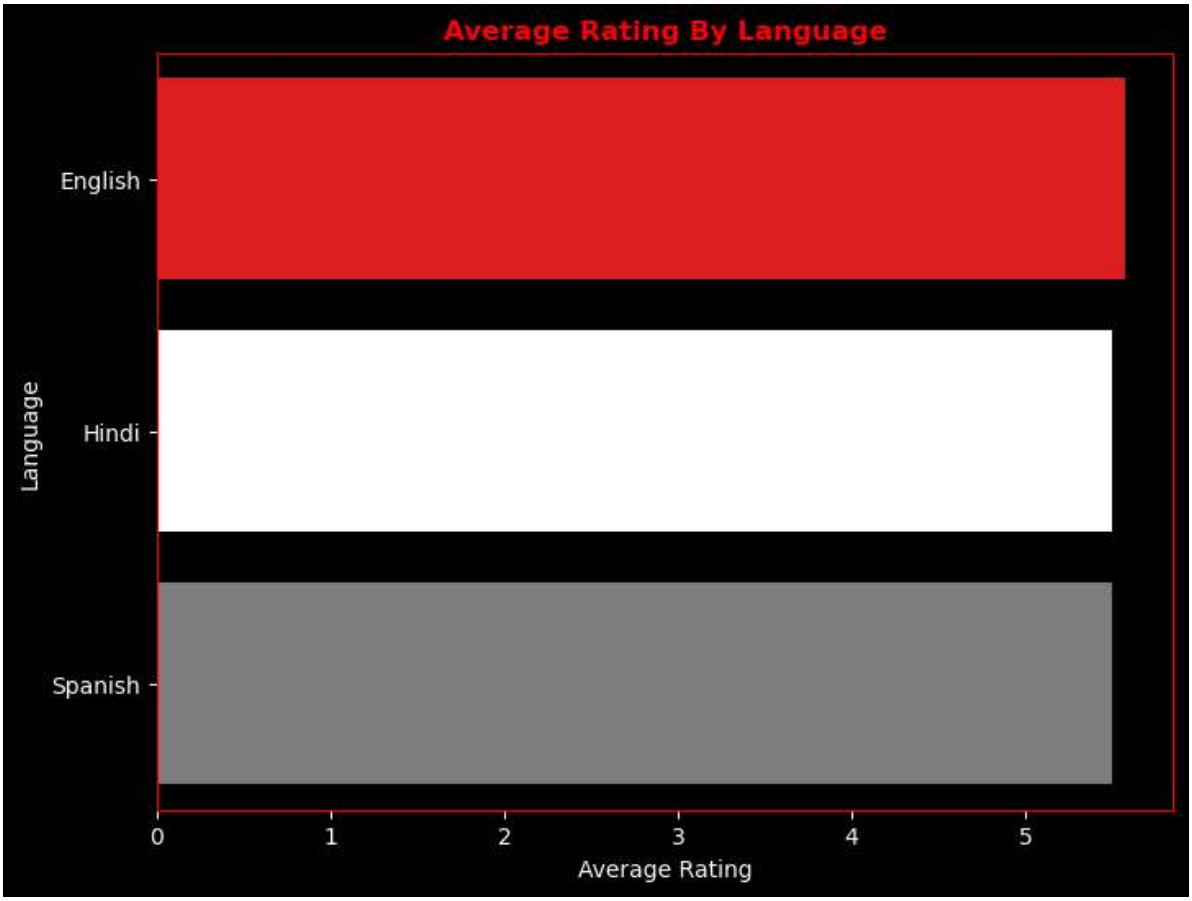
In [49]:
```
# Top 5 languages by number of movies
top_languages = movie_df['language'].value_counts().head(5)
# Calculate 'Others' as sum of remaining languages
others_count = movie_df['language'].value_counts().iloc[5:].sum()
top_languages['Others'] = others_count
# Pie chart
plt.figure(figsize=(8,6))
wedges, texts, autotexts = plt.pie(
top_languages,
labels=top_languages.index,
autopct='%1.1f%%',
startangle=140
)
# Set language labels to white
for text in texts:
    text.set_color('white')
# Set percentage text to black
for autotext in autotexts:
    autotext.set_color('black')
plt.title("Movie Distribution by Language", fontsize=14, fontweight='bold', cc
plt.show()
```



In [50]:
```
# Group by 'Languate' to get count and average metrics for easier language-wis
language_group = (
movie_df_exploded.groupby('language')
.agg(
number_of_movies = ('show_id','count'),
avg_rating = ('rating','mean'),
avg_popularity = ('popularity','mean'),
avg_vote_count = ('vote_count','mean'),
avg_budget = ('budget','mean'),
avg_revenue = ('revenue','mean'),
avg_profit = ('profit','mean')
).reset_index().sort_values(by='number_of_movies', ascending=False))
language_filtered=language_group[language_group['number_of_movies']>=100]
```

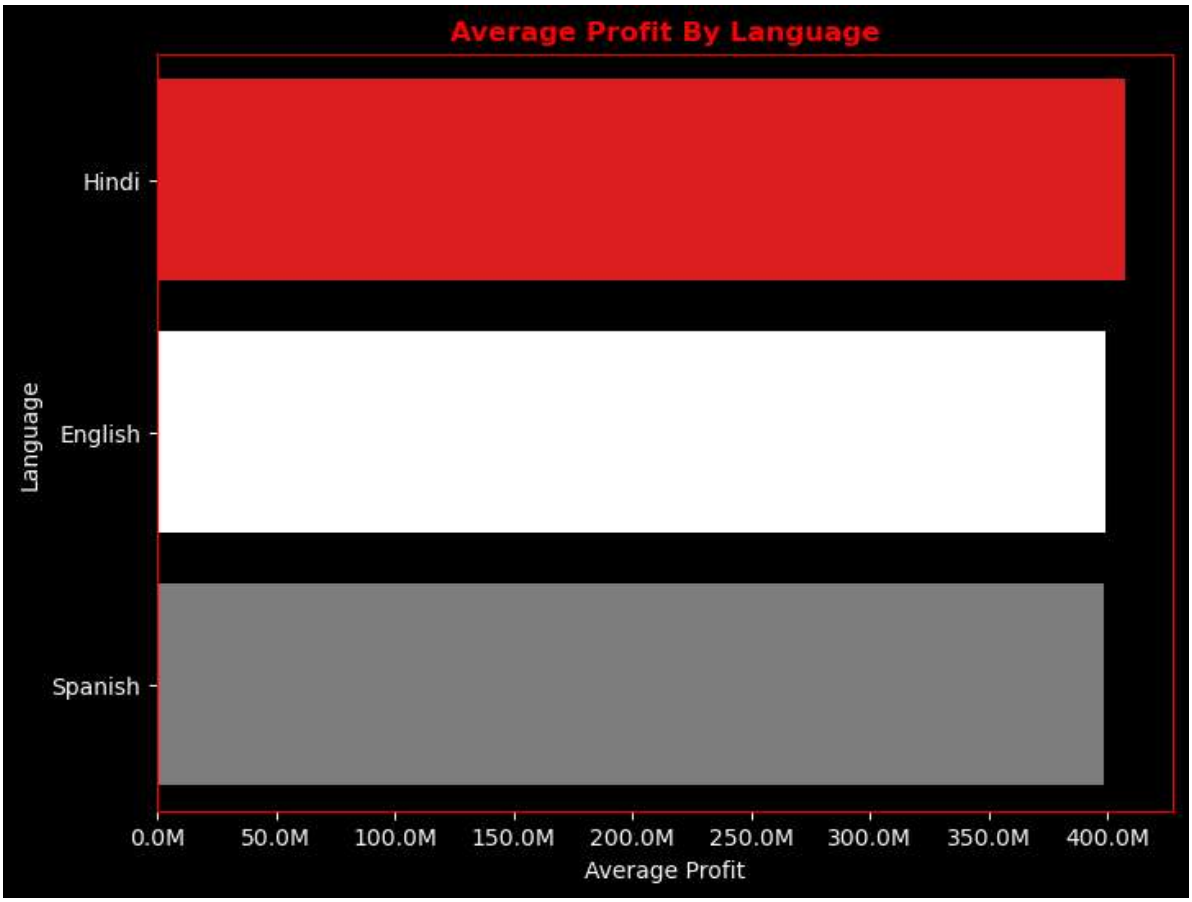
Top Average Rating Languages

In [53]:
```
# Plot Average Rating languages
sns.barplot(x='avg_rating',y='language',data=language_filtered.sort_values(by=
plt.xlabel('Average Rating')
plt.ylabel('Language')
plt.title('Average Rating By Language')
plt.show()
```



Top Average Porfit Languages

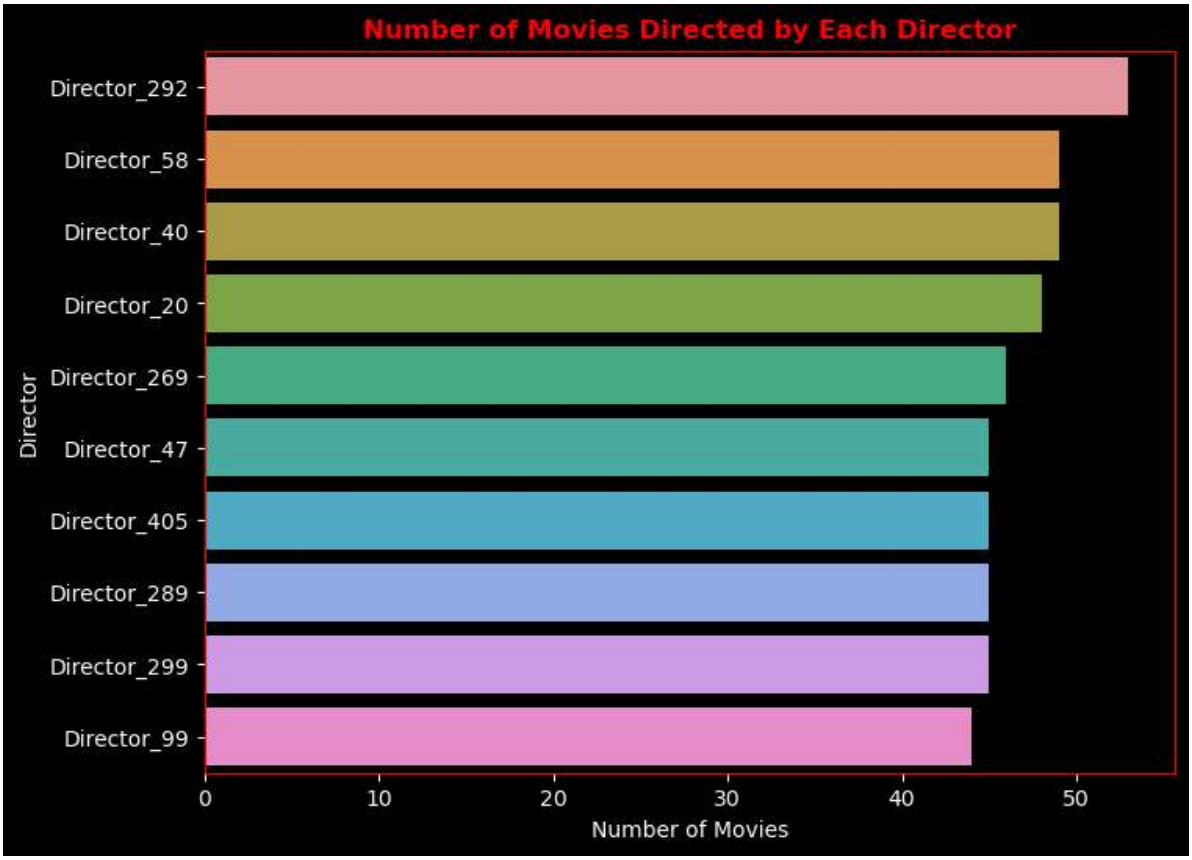
In [55]:
```
# Plot of Average Profit Language
sns.barplot(x='avg_profit',y='language',data=language_filtered.sort_values(by=
plt.xlabel('Average Profit')
plt.ylabel('Language')
plt.title('Average Profit By Language')
ax = plt.gca()
ax.xaxis.set_major_formatter(FuncFormatter(millions))
plt.show()
```



Director Analysis

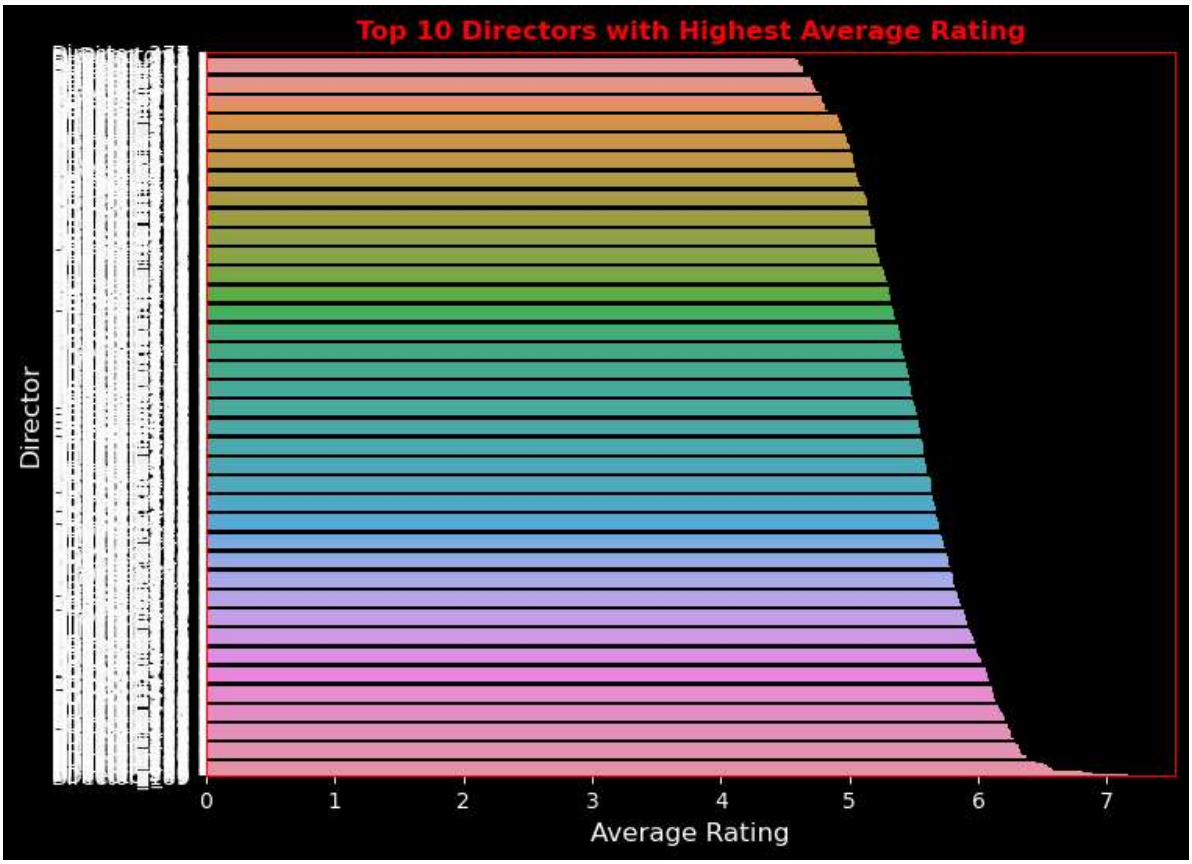
In [56]:
```
# Director-wise grouping
director_group = (
movie_df[movie_df['director']!='Unknown'].groupby('director')
.agg(
number_of_movies = ('show_id','count'),
avg_rating = ('rating','mean'),
avg_popularity = ('popularity','mean'),
avg_vote_count = ('vote_count','mean'),
avg_budget = ('budget','mean'),
avg_revenue = ('revenue','mean'),
avg_profit = ('profit','mean')
).reset_index().sort_values(by='number_of_movies', ascending=False))
```

In [57]:
```
# Plot the top 10 directors with the highest number of movies using a horizont
sns.barplot(x='number_of_movies', y='director', data=director_group.head(10))
plt.title('Number of Movies Directed by Each Director')
plt.xlabel('Number of Movies')
plt.ylabel('Director')
plt.show()
```



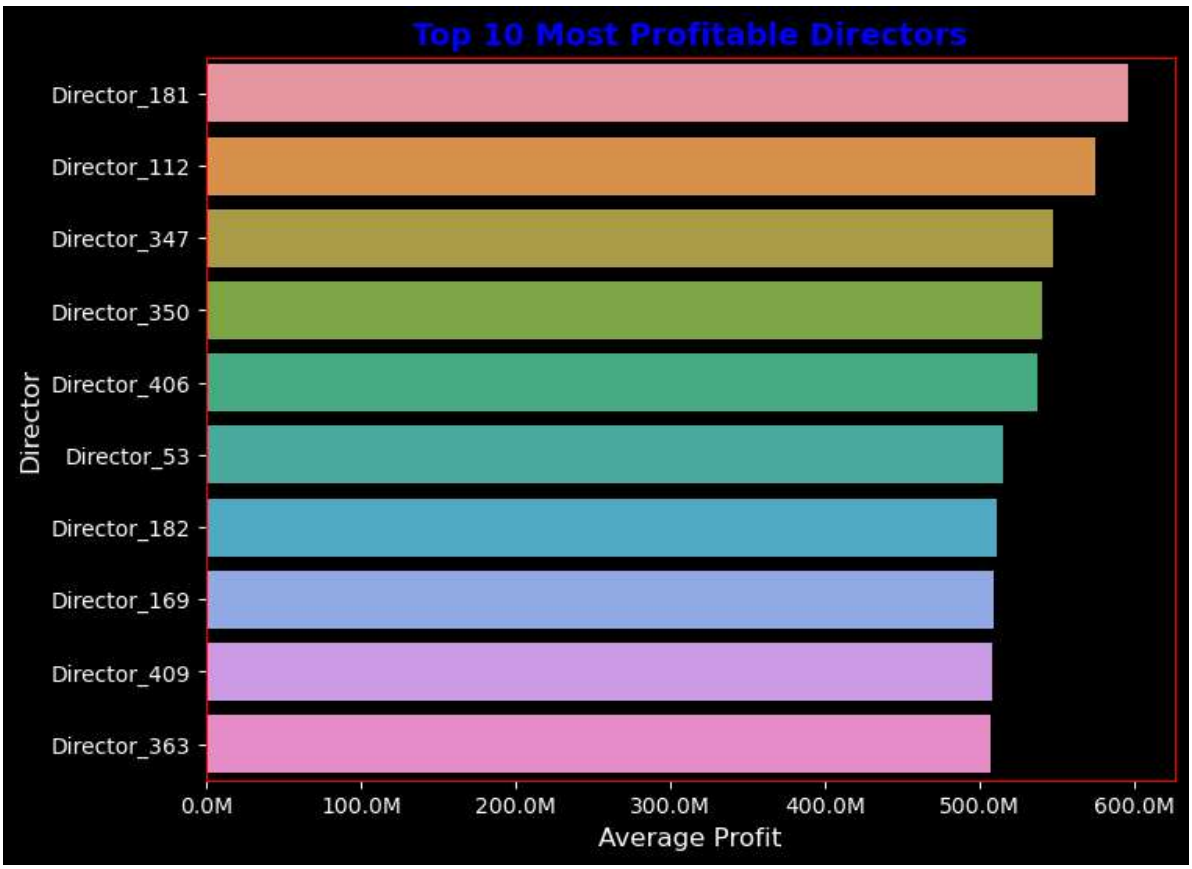
Directors with the best average ratings

In [59]:
```
# Keep only directors with more than 5 movies to ensure reliable averages
filtered=director_group[director_group['number_of_movies']>5]
# Plot Average Rating by Director
sns.barplot(x='avg_rating',y='director',data=filtered.sort_values(by='avg_rati
plt.title('Top 10 Directors with Highest Average Rating')
plt.xlabel('Average Rating', fontsize=12)
plt.ylabel('Director', fontsize=12)
plt.show()
```



Top Profitable Directors

In [61]:
```
# Select the top 10 directors with the highest average profit
top_profitable = filtered.sort_values(by='avg_profit', ascending=False).head(1
# Bar plot of the top 10 most profitable directors (log scale for better visib
sns.barplot(x='avg_profit', y='director', data=top_profitable)
plt.title("Top 10 Most Profitable Directors", fontsize=14, fontweight='bold',
plt.xlabel("Average Profit", fontsize=12)
plt.ylabel("Director", fontsize=12)
ax = plt.gca()
ax.xaxis.set_major_formatter(FuncFormatter(millions))
plt.show()
```



In [ ]: