Final Project Submission

- Student name: Group 5
- Members: Francis Ndirangu, Hanifa Chepchirchir, Brian Kipyegon, Ian Riua, Denis Kibor
- Student pace: full time
- Scheduled project review date/time: 2/5/2025
- Instructor name: Nikita Njoroge.
- Blog post URL:

Introduction

Our company wants to venture into the movie industry by launching a new movie studio. However, as a newcomer in this competitive space, we currently lack the insights and experience to determine what types of films will yield the greatest commercial success.

This project aims to bridge that gap.

By analyzing recent data from sources such as IMDb, The Movies and The Numbers, we will explore trends in genre, director success, language profitability, movie_popularity, and budget efficiency over the past years. Our goal is to identify which types of films — by genre, language, or production strategy — consistently perform well at the box office.

The findings from this analysis will be translated into actionable recommendations that the company can use to make data-driven decisions on what types of films to create, how to allocate production budgets, and who to collaborate with in the filmmaking process.

Objectives

Investigate the data to come up with insights on the types of movies, directors to hire, our new movie studio should make.

Do top-rated directors make more profits than lesser rated directors.

Do popular movies make more profit than less popular ones.

Do higher-rated movies make more profits than lower-rated movies.

Step 1: Importing the necessary Libraries.

To perform this analysis, we use a combination of powerful Python libraries.

In [1]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
import sqlite3
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from scipy import stats
from statsmodels.stats.weightstats import ttest_ind
```

Step 2: Loading Datasets and Merging

In this step we will load the datasets and extract information that we will use for our analysis.

1. IMDB Database

We will first look at the IMDb database. Because it's an database we will use sql to connect to it and the combine pandas and sql to create dataframes from the database.

In [2]:
```
# Connect sqlite3 to the database and fetch all tables
con = sqlite3.connect(r'zippedData\im.db\im.db')

cur = con.cursor()

cur.execute("SELECT name from sqlite_master").fetchall()
```

Out[2]:
```
[('movie_basics',),
 ('directors',),
 ('known_for',),
 ('movie_akas',),
 ('movie_ratings',),
 ('persons',),
 ('principals',),
 ('writers',)]
```

Combining Tables from Imdb

To create a single dataset with rich movie-level insights, we will join multiple tables from our IMDb database. This includes information about movies, their alternate titles, ratings, key personnel (like actors and directors), and their roles.

By joining `principals`, `movie_akas`, `movie_basics`, `movie_ratings`, and `persons` using shared IDs (`movie_id` and `person_id`), we will build a comprehensive dataframe that allows us to analyze movies from multiple perspectives.

In [ ]:
```
# Run a comprehensive SQL join to combine key movie-related tables:
# This unified query brings together movie titles, alternate names, ratings, and c
#Please note some tables had repeating columns.
#the tables merged below have all the different columns in the database.

combined = """SELECT *
                from principals
                JOIN movie_akas USING (movie_id)
                JOIN movie_basics USING (movie_id)
                JOIN movie_ratings USING (movie_id)
                JOIN persons USING (person_id);"""

combined_df = pd.read_sql_query(combined,con)
combined_df.head()
```

Out[ ]:

	movie_id	ordering	person_id	category	job	characters	ordering	title	region	lang
0	tt0323808	10	nm0059247	editor	None	None	1	May Day	GB	
1	tt0323808	10	nm0059247	editor	None	None	2	Cowboys for Christ	GB	
2	tt0323808	10	nm0059247	editor	None	None	3	The Wicker Tree	GB	
3	tt0323808	10	nm0059247	editor	None	None	4	The Wicker Tree	None	
4	tt0323808	10	nm0059247	editor	None	None	5	Плетеное дерево	RU	

5 rows × 24 columns

In [4]:
```
# Create a copy of the combined dataframe to preserve the original and inspect its

imdb_df = combined_df.copy()
imdb_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2422866 entries, 0 to 2422865
Data columns (total 24 columns):
 #   Column              Dtype
---  ------              -----
 0   movie_id            object
 1   ordering            int64
 2   person_id           object
 3   category            object
 4   job                 object
 5   characters          object
 6   ordering            int64
 7   title               object
 8   region              object
 9   language            object
 10  types               object
 11  attributes          object
 12  is_original_title   float64
 13  primary_title       object
 14  original_title      object
 15  start_year          int64
 16  runtime_minutes     float64
 17  genres              object
```

```
 18  averagerating       float64
 19  numvotes            int64
 20  primary_name        object
 21  birth_year          float64
 22  death_year          float64
 23  primary_profession  object
dtypes: float64(5), int64(4), object(15)
memory usage: 443.6+ MB
```

Checking for missing values

In the cell below we check the null value percentage for each column in the dataframe.

We will rank the null value percentage in descending order.

We want to remove the null values from the dataframe before we combine it with the other dataframes

In [5]:
```
#checking columns with null values.
imdb_df.isna().mean()[imdb_df.isna().mean() > 0].sort_values(ascending=False)
```

Out[5]:
```
death_year            0.982493
attributes            0.949760
language              0.855840
job                   0.725228
characters            0.606105
birth_year            0.564602
types                 0.402041
region                0.162510
runtime_minutes       0.039211
primary_profession    0.021650
genres                0.003738
dtype: float64
```

Steps to be taken regarding null values:

1. The columns with more than 60% null values will be dropped.

2. The null value rows of the remaining columns will be dropped.

In [6]:
```
#dropping the columns with 40%+ null values
imdb_df.drop(labels=['death_year','attributes','language','job','characters','birt

#checking columns with null values.
imdb_df.isna().mean()[imdb_df.isna().mean() > 0].sort_values(ascending=False)
```

Out[6]:
```
region                0.162510
runtime_minutes       0.039211
primary_profession    0.021650
genres                0.003738
dtype: float64
```

Now that we have confirmed the dropping of columns with more than 60% null values.

We will now drop the null value rows of the remaining columns.

In [7]:
```
columns_with_null_values = imdb_df.isna().mean()[imdb_df.isna().mean() > 0].index
columns_with_null_values

#I will drop all the rows with null values since they are only a small percentage
imdb_df.dropna(axis=0,subset=columns_with_null_values,inplace=True)
#checking columns with null values.
imdb_df.isna().mean()[imdb_df.isna().mean() > 0].sort_values(ascending=False)

#The output of the empty list is showing that there are no missing values in the
```

Out[7]:
```
Series([], dtype: float64)
```

In [8]:
```
# checking the info of the dataframe
imdb_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1909588 entries, 0 to 2422865
Data columns (total 17 columns):
 #   Column              Dtype
---  ------              -----
 0   movie_id            object
 1   ordering            int64
 2   person_id           object
 3   category            object
 4   ordering            int64
 5   title               object
 6   region              object
 7   is_original_title   float64
 8   primary_title       object
 9   original_title      object
 10  start_year          int64
 11  runtime_minutes     float64
 12  genres              object
 13  averagerating       float64
 14  numvotes            int64
 15  primary_name        object
 16  primary_profession  object
dtypes: float64(3), int64(4), object(10)
memory usage: 262.2+ MB
```

2. TMDb Movies Dataset

We will now load the TMDb (The Movie Database) dataset, which provides additional movie-related information such as popularity scores, average ratings, and original language. This data will help us explore audience preferences and language trends, and enrich our overall analysis of movie performance.

In [9]:
```
tmdb_movies_df = pd.read_csv(r'zippedData\tmdb.movies.csv\tmdb.movies.csv')
tmdb_movies_df.head()
```

Out[9]:

	Unnamed: 0	genre_ids	id	original_language	original_title	popularity	release_date	tit
0	0	[12, 14, 10751]	12444	en	Harry Potter and the Deathly Hallows: Part 1	33.533	2010-11-19	Har Pott and th Death Hallow Part

	Unnamed: 0	genre_ids	id	original_language	original_title	popularity	release_date	tit
1	1	[14, 12, 16, 10751]	10191	en	How to Train Your Dragon	28.734	2010-03-26	How t Tra Yo Dragc
2	2	[12, 28, 878]	10138	en	Iron Man 2	28.515	2010-05-07	Iron Ma
3	3	[16, 35, 10751]	862	en	Toy Story	28.005	1995-11-22	Toy Sto
4	4	[28, 878, 12]	27205	en	Inception	27.920	2010-07-16	Inceptic

3. The Numbers Dataset

To analyze the financial performance of movies, we load a dataset containing production budgets, domestic and international grosses, and release dates. This data comes from **The Numbers** and is essential for calculating net profits and evaluating the commercial success of films.

In [10]:
```
movie_budgets_df = pd.read_csv(r'zippedData\tn.movie_budgets.csv\tn.movie_budgets

movie_budgets_df.head()
```

Out[10]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross
0	1	Dec 18, 2009	Avatar	$425,000,000	$760,507,625	$2,776,345,279
1	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	$410,600,000	$241,063,875	$1,045,663,875
2	3	Jun 7, 2019	Dark Phoenix	$350,000,000	$42,762,350	$149,762,350
3	4	May 1, 2015	Avengers: Age of Ultron	$330,600,000	$459,005,868	$1,403,013,963
4	5	Dec 15, 2017	Star Wars Ep. VIII: The Last Jedi	$317,000,000	$620,181,382	$1,316,721,747

At this stage, we have successfully loaded all the required datasets from IMDb, TMDb and The Numbers. Through careful inspection of each dataset's structure, we now have comprehensive datasets without missing values and duplicates.

Merging IMDb and TMDb Datasets

After looking at the datasets that we have chosen for ou analysis, we noticed that imdb and tmdb datasets contains information about movies that's almost similar(they complement each other)

We decided merge the 2 datasets using the `title` column as the key. This will allow us to combine detailed movie information from IMDb (like genres, ratings, and contributors) with TMDB

In [11]:
```
imdb_tmdb = pd.merge(imdb_df,tmdb_movies_df,on='title')
imdb_tmdb.head()
```

Out[11]:

	movie_id	ordering	person_id	category	ordering	title	region	is_original_title	primary_t
0	tt0323808	10	nm0059247	editor	3	The Wicker Tree	GB	0.0	The Wi
1	tt0323808	1	nm3579312	actress	3	The Wicker Tree	GB	0.0	The Wi
2	tt0323808	2	nm2694680	actor	3	The Wicker Tree	GB	0.0	The Wi
3	tt0323808	3	nm0574615	actor	3	The Wicker Tree	GB	0.0	The Wi
4	tt0323808	4	nm0502652	actress	3	The Wicker Tree	GB	0.0	The Wi

5 rows × 26 columns

In [12]:
```
# checking the info of the dataframe
imdb_tmdb.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 289507 entries, 0 to 289506
Data columns (total 26 columns):
 #   Column              Non-Null Count   Dtype  
---  ------              --------------   -----  
 0   movie_id            289507 non-null  object 
 1   ordering            289507 non-null  int64  
 2   person_id           289507 non-null  object 
 3   category            289507 non-null  object 
 4   ordering            289507 non-null  int64  
 5   title               289507 non-null  object 
 6   region              289507 non-null  object 
 7   is_original_title   289507 non-null  float64
 8   primary_title       289507 non-null  object 
 9   original_title_x    289507 non-null  object 
 10  start_year          289507 non-null  int64  
 11  runtime_minutes     289507 non-null  float64
 12  genres              289507 non-null  object 
 13  averagerating       289507 non-null  float64
 14  numvotes            289507 non-null  int64  
 15  primary_name        289507 non-null  object 
 16  primary_profession  289507 non-null  object 
 17  Unnamed: 0          289507 non-null  int64  
 18  genre_ids           289507 non-null  object 
 19  id                  289507 non-null  int64  
 20  original_language   289507 non-null  object 
 21  original_title_y    289507 non-null  object 
 22  popularity          289507 non-null  float64
 23  release_date        289507 non-null  object 
 24  vote_average        289507 non-null  float64
 25  vote_count          289507 non-null  int64  
dtypes: float64(5), int64(7), object(14)
memory usage: 59.6+ MB
```

Adding Budget and Revenue Data

In this step, we will merge our combined IMDb-TMDb dataset with financial data from **The Numbers** dataset. We'll match records using the movie `title` from IMDb-TMDb and the `movie` column from the budgets dataset. This merge will give us access to key financial figures like production budgets, domestic gross, and worldwide gross — essential for analyzing box office performance.

In [13]:
```
# merge imdb_tmdb_merge to numbers

all_merge = pd.merge(imdb_tmdb,movie_budgets_df,left_on='title',right_on='movie')
all_merge.head()
```

Out[13]:

	movie_id	ordering	person_id	category	ordering	title	region	is_original_title	p
0	tt0475290	10	nm0005683	cinematographer	24	Hail, Caesar!	GB	0.0	
1	tt0475290	1	nm0000982	actor	24	Hail, Caesar!	GB	0.0	
2	tt0475290	2	nm0000123	actor	24	Hail, Caesar!	GB	0.0	
3	tt0475290	3	nm2403277	actor	24	Hail, Caesar!	GB	0.0	
4	tt0475290	4	nm0000146	actor	24	Hail, Caesar!	GB	0.0	

5 rows × 32 columns

In [14]:
```
# checking the info of the dataframe
all_merge.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 84687 entries, 0 to 84686
Data columns (total 32 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   movie_id            84687 non-null  object
 1   ordering            84687 non-null  int64
 2   person_id           84687 non-null  object
 3   category            84687 non-null  object
 4   ordering            84687 non-null  int64
 5   title               84687 non-null  object
 6   region              84687 non-null  object
 7   is_original_title   84687 non-null  float64
 8   primary_title       84687 non-null  object
 9   original_title_x    84687 non-null  object
 10  start_year          84687 non-null  int64
 11  runtime_minutes     84687 non-null  float64
 12  genres              84687 non-null  object
 13  averagerating       84687 non-null  float64
 14  numvotes            84687 non-null  int64
 15  primary_name        84687 non-null  object
 16  primary_profession  84687 non-null  object
 17  Unnamed: 0          84687 non-null  int64
 18  genre_ids           84687 non-null  object
 19  id_x                84687 non-null  int64
 20  original_language   84687 non-null  object
```

```
 21  original_title_y    84687 non-null  object
 22  popularity          84687 non-null  float64
 23  release_date_x      84687 non-null  object
 24  vote_average        84687 non-null  float64
 25  vote_count          84687 non-null  int64
 26  id_y                84687 non-null  int64
 27  release_date_y      84687 non-null  object
 28  movie               84687 non-null  object
 29  production_budget   84687 non-null  object
 30  domestic_gross      84687 non-null  object
 31  worldwide_gross     84687 non-null  object
dtypes: float64(5), int64(8), object(19)
memory usage: 21.3+ MB
```

In [15]:
```
#Confirming there are no columns with null values in the merged dataset.
all_merge.isna().any()[all_merge.isna().any() == True]

#The returned empty list shows that there are no columns with  null values.
```

Out[15]:
```
Series([], dtype: bool)
```

Conclusion.

At this stage, we have successfully loaded all the required datasets from IMDb, TMDb, Box Office Mojo, and The Numbers. Through careful merging and inspection of each dataset's structure, we now have a comprehensive dataset that combines:

- **Movie details** (titles, genres, release years)
- **Ratings and popularity metrics**
- **Contributor information** (directors, actors, and other key personnel)
- **Financial data** (production budgets, domestic and worldwide gross)

This unified dataset will serve as the foundation for our analysis, allowing us to explore trends in genres, profitability, and key performance drivers across the film industry.

Step 2: Data Cleaning

Before performing any analysis, it's important to clean the dataset to ensure accuracy and consistency. This step will involve:

- Removing or handling missing values (e.g., missing genres or financial figures)
- Fixing duplicates or inconsistent entries
- Standardizing column formats (e.g., dates, currencies)
- Creating new calculated fields such as profits

A clean and well-structured dataset will allow us to generate reliable insights in the next steps of the project.

In [16]:
```
# Checking for duplicates in our merged data

all_merge[all_merge.duplicated() == True]
```

Out[16]:

movie_id	ordering	person_id	category	ordering	title	region	is_original_title	primary_title

0 rows × 32 columns

Our data doesn't have any duplicated values, so we will move to the next step.

In [17]:
```
# checking if there's any column with missing values
all_merge.isna().any()[all_merge.isna().any() == True]
```

Out[17]:
```
Series([], dtype: bool)
```

The data does not also have missing values. So we move on

In [18]:
```
# rename our datframe to all_combined
all_combined = all_merge
```

In [19]:
```
# checking for the columns of our combined dataframe
all_combined.columns
```

Out[19]:
```
Index(['movie_id', 'ordering', 'person_id', 'category', 'ordering', 'title',
       'region', 'is_original_title', 'primary_title', 'original_title_x',
       'start_year', 'runtime_minutes', 'genres', 'averagerating', 'numvotes',
       'primary_name', 'primary_profession', 'Unnamed: 0', 'genre_ids', 'id_x',
       'original_language', 'original_title_y', 'popularity', 'release_date_x',
       'vote_average', 'vote_count', 'id_y', 'release_date_y', 'movie',
       'production_budget', 'domestic_gross', 'worldwide_gross'],
      dtype='object')
```

Cleaning Currency Columns

The financial columns in our dataset (`production_budget` , `domestic_gross` , and `worldwide_gross`) are stored as strings with dollar signs and commas, which prevents numerical analysis.

In this step, we remove the dollar signs (`$`) and commas (`,`) using regular expressions, and then convert the cleaned strings into numeric (`float`) format. This transformation will allow us to perform mathematical operations such as calculating profits.

In [20]:
```
# Clean currency columns by removing dollar signs and commas,
# then convert them from string to float for numerical analysis

all_combined['worldwide_gross'] = all_combined['worldwide_gross'].replace('[\$,]'
all_combined['domestic_gross'] = all_combined['domestic_gross'].replace('[\$,]',
all_combined['production_budget'] = all_combined['production_budget'].replace('[\
```

We can now create a profits column in our dataset in the step below

In [21]:
```
#creating a profit column worldwide_gross - production_budget
all_combined['profits'] = all_combined['worldwide_gross'] - all_combined['product
all_combined[['profits','worldwide_gross','production_budget']].head()
```

Out[21]:

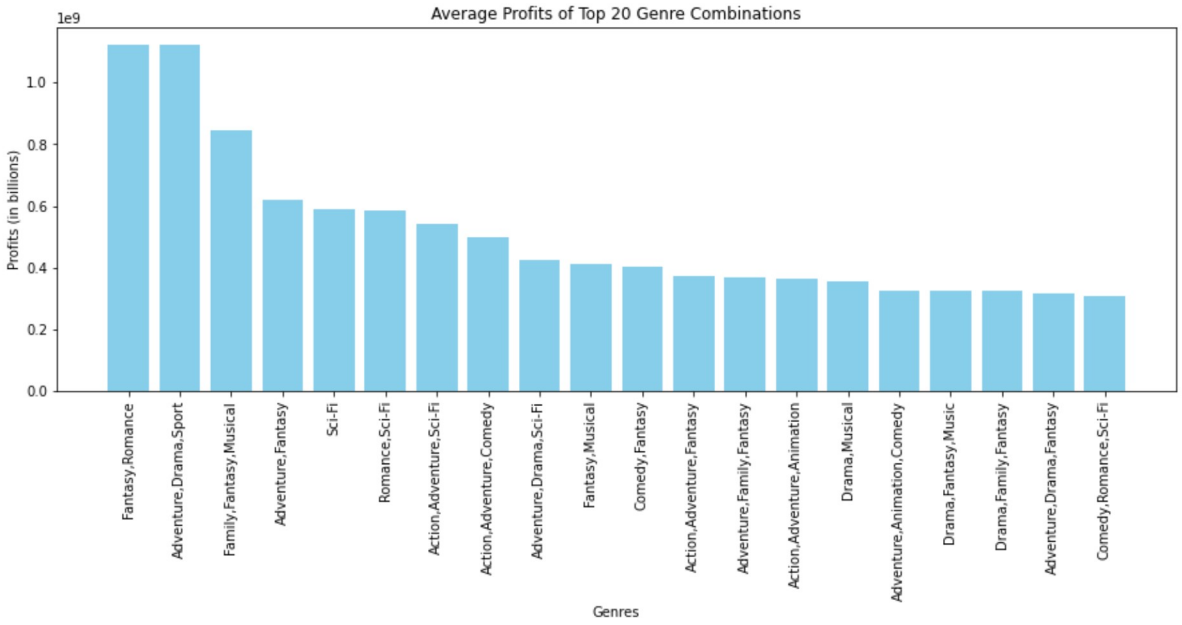
	profits	worldwide_gross	production_budget
0	42160680.0	64160680.0	22000000.0
1	42160680.0	64160680.0	22000000.0
2	42160680.0	64160680.0	22000000.0
3	42160680.0	64160680.0	22000000.0
4	42160680.0	64160680.0	22000000.0

Step 4. Exploratory Data anlysis

A graph of Average profits against top 20 genre combinations.

We are able to make this graph because majority of the movies in the dataset has more than one genre.

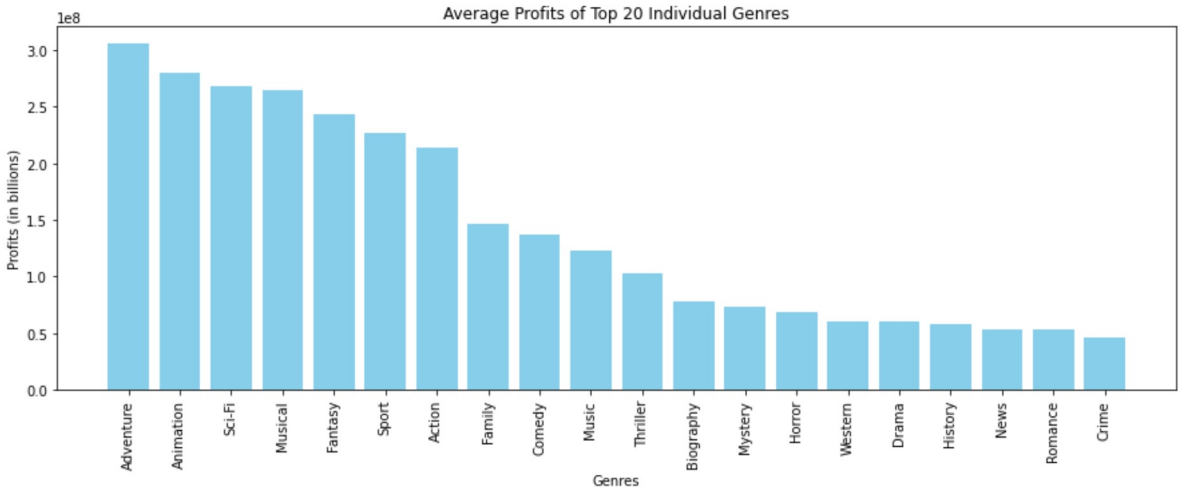
In [22]:
```
# Convert the columns to strings
all_combined['worldwide_gross'] = all_combined['worldwide_gross'].astype(str).str
all_combined['production_budget'] = all_combined['production_budget'].astype(str)

# Convert to float after cleaning
all_combined['worldwide_gross'] = pd.to_numeric(all_combined['worldwide_gross'], e
all_combined['production_budget'] = pd.to_numeric(all_combined['production_budget



# Calculate profits
all_combined['profits'] = all_combined['worldwide_gross'] - all_combined['product

# Plot graph of top 20 genre combinations against profits

result = all_combined.groupby('genres')['profits'].mean().sort_values(ascending=Fa
fig, ax = plt.subplots(figsize=(15, 5))
ax.bar(result.index, result.values, color='skyblue')
plt.title('Average Profits of Top 20 Genre Combinations')
plt.xlabel('Genres')
plt.ylabel('Profits (in billions)')
plt.xticks(rotation=90)
plt.show()
```



We now want to graph Top 20 Individual genres against profits

As previously mentioned majority of the movies in the dataset have more than one genre. We will use the `explode` method in order to seperate the genres of each movie. Meaning that each movie will have an record for each genre in its genre combination. This will allow us to look at each individual genre and rank them.

In [23]:
```
all_combined.columns = [f"{col}_{i}" if all_combined.columns.tolist().count(col) :
                        for i, col in enumerate(all_combined.columns)]

# Seperate genre combinations into individual genres using the explode method
all_combined_exploded = all_combined.assign(genres=all_combined['genres'].str.spli

#calculating the profits of each genre
result = all_combined_exploded.groupby('genres')['profits'].mean().sort_values(asc

# Plot the results
fig, ax = plt.subplots(figsize=(15, 5))
ax.bar(result.index, result.values, color='skyblue')
plt.title('Average Profits of Top 20 Individual Genres')
plt.xlabel('Genres')
plt.ylabel('Profits (in billions)')
plt.xticks(rotation=90)
plt.show()
```



Insight from the graph above:

We can see that Adventure, Animation and Sci-fi have the highest average profits. It is advisable for out movie studio to focus on these genres as they are likely to make the highest profits.

In the cell below we are seeing how many times each individual genre appears in the dataset

In [24]:
```
#looking at the count of individual genres in the dataset ordering them by descend
all_combined_exploded['genres'].value_counts()
```

Out[24]:
```
Drama          45209
Action         26953
Comedy         21396
Adventure      20720
Thriller       17569
Crime          12050
Sci-Fi         10643
Horror         10605
Romance         8636
Biography       7732
Mystery         7204
Animation       5547
Fantasy         5503
Family          3682
Music           2665
History         2619
```

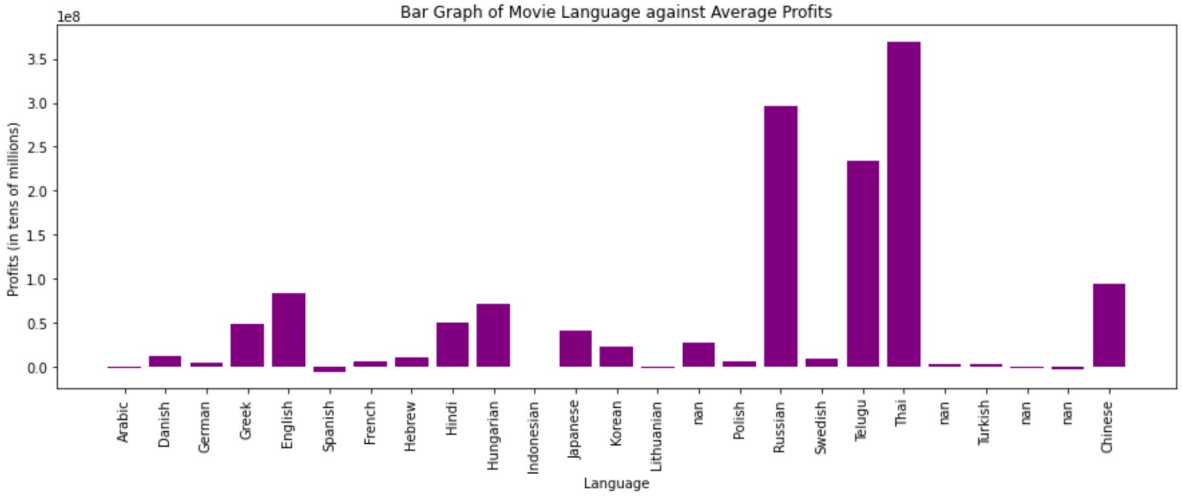
```
Documentary     1978
Sport           1368
War              840
Musical          434
Western          338
News              43
Name: genres, dtype: int64
```

We now want to make a graph of Language against average profits.

In order to make the x-axis easier to understand we will be adding the full name of each language.

We will then graph the language vs the average profits to see which languages make the most average profit.

In [25]:
```
language_mapping = {'en': 'English', 'es': 'Spanish', 'fr': 'French', 'de': 'Germa
                    'pt': 'Portuguese', 'ru': 'Russian', 'zh': 'Chinese', 'ja': '
                    'ar': 'Arabic', 'hi': 'Hindi', 'bn': 'Bengali', 'pa': 'Punjabi
                    'vi': 'Vietnamese', 'tr': 'Turkish', 'th': 'Thai', 'ms': 'Mala
                    'ta': 'Tamil', 'kn': 'Kannada', 'gu': 'Gujarati', 'te': 'Telug
                    'pl': 'Polish', 'uk': 'Ukrainian', 'sv': 'Swedish', 'no': 'Nor
                    'fi': 'Finnish', 'cs': 'Czech', 'hu': 'Hungarian', 'ro': 'Roma
                    'el': 'Greek', 'he': 'Hebrew', 'th': 'Thai', 'tr': 'Turkish',
                    'hr': 'Croatian', 'bs': 'Bosnian', 'sl': 'Slovenian', 'sq': 'A
                    'bg': 'Bulgarian', 'is': 'Icelandic', 'lv': 'Latvian', 'lt':
                    'eu': 'Basque', 'gl': 'Galician', 'ca': 'Catalan', 'az': 'Azer
                    'ky': 'Kyrgyz', 'uz': 'Uzbek', 'km': 'Khmer', 'lo': 'Lao', 'my
                    'si': 'Sinhalese', 'sw': 'Swahili', 'zu': 'Zulu', 'af': 'Afrik
                    'sq': 'Albanian', 'km': 'Khmer'}

#making a df with unique movies because the all_combined has more than one actor p
unique_movies = all_combined.drop_duplicates(subset='title')
lang_prof = unique_movies.groupby('original_language')['profits'].mean()
lang_prof.index = lang_prof.index.map(language_mapping)

x_positions = range(len(lang_prof))

fig, ax = plt.subplots(figsize=(15, 5))
bars = ax.bar(x_positions, lang_prof.values, color='purple')
ax.set_xticks(x_positions)
ax.set_xticklabels(lang_prof.index, rotation=90)
ax.set_title('Bar Graph of Movie Language against Average Profits')
ax.set_xlabel('Language')
ax.set_ylabel('Profits (in tens of millions)')

plt.show()
```



Insights from the table above:

From the table above we see that the Russian, Thai, Telugu, Chinese and English have the highest profits.

We expected english to have the highest mean profits since more countries speak english meaning that movies in english appear in cinemas in more countries. Its possible that there are many more english movies in the dataset than the other languages, which would make english look like it has a lower profit.

In the cell below we will check how many movies are in each language in the dataset.

In [26]:
```
#making a df with unique movies
unique_movies = all_combined.drop_duplicates(subset='title')
#adding the full language names to the original_language column
unique_movies['original_language'] = unique_movies['original_language'].map(langua



#making counting the number of movies per language and the mean profits of each la
#then sorting by the mean profits column
mean_profit_by_language = unique_movies.groupby('original_language').agg(lang_cour
                                                                          mean_p

#returning the table to see the languages and the number of movies per language
mean_profit_by_language.head(15)
```

```
<ipython-input-26-e0d392d2369b>:4: 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/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  unique_movies['original_language'] = unique_movies['original_language'].map(lang
uage_mapping)
```

Out[26]:

original_language	lang_count	mean_prof
English	1709	8.458770e+07
French	17	6.953255e+06
Spanish	9	-4.992453e+06

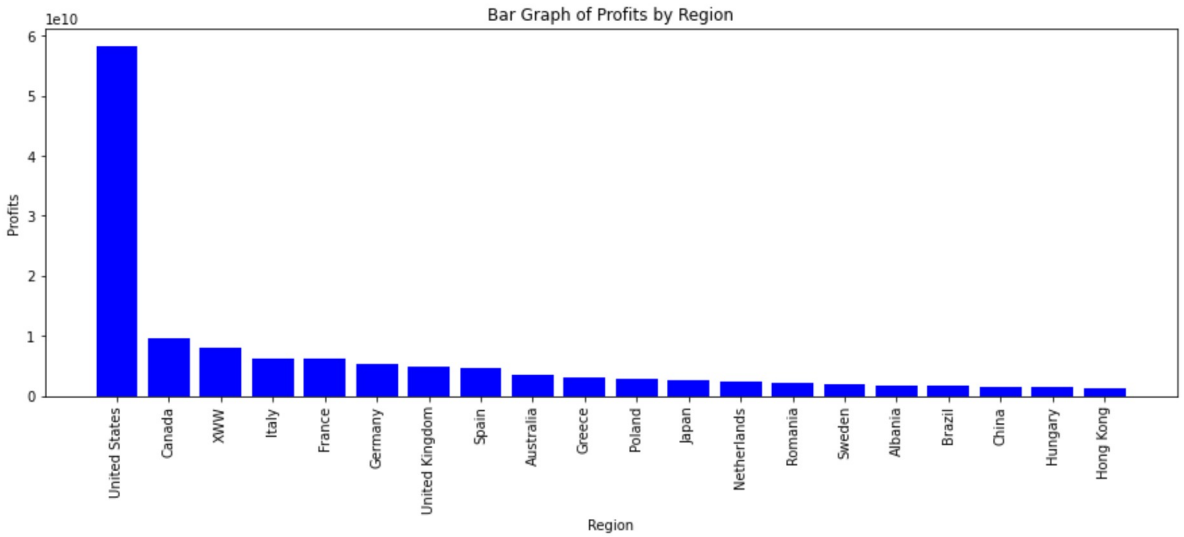
original_language	lang_count	mean_prof
Hindi	8	5.033031e+07
Chinese	6	9.408204e+07
German	4	4.476870e+06
Russian	4	2.961487e+08
Korean	3	2.377689e+07
Polish	2	6.251002e+06
Swedish	2	9.204893e+06
Japanese	2	4.227937e+07
Arabic	1	-1.159068e+06
Thai	1	3.696065e+08
Telugu	1	2.335029e+08
Indonesian	1	2.273160e+05

Insight from the table above:

The dataset has many more english movies than any other language. We can't draw conclusions about other movie languages from the dataset because the dataset is not balanced.

We want to graph of region against sum profits

In [27]:
```
#Getting the full names of each country in the region column
imdb_region_dict = {
    'US': 'United States', 'CA': 'Canada', 'GB': 'United Kingdom', 'AU': 'Australi
    'DE': 'Germany', 'FR': 'France', 'IT': 'Italy', 'ES': 'Spain', 'MX': 'Mexico',
    'JP': 'Japan', 'CN': 'China', 'KR': 'South Korea', 'RU': 'Russia', 'PL': 'Pola
    'AR': 'Argentina', 'NG': 'Nigeria', 'NL': 'Netherlands', 'SE': 'Sweden', 'DK'
    'FI': 'Finland', 'BE': 'Belgium', 'AT': 'Austria', 'CH': 'Switzerland', 'PT':
    'TR': 'Turkey', 'EG': 'Egypt', 'SG': 'Singapore', 'MY': 'Malaysia', 'ID': 'Ind
    'TH': 'Thailand', 'VN': 'Vietnam', 'KW': 'Kuwait', 'SA': 'Saudi Arabia', 'AE'
    'QA': 'Qatar', 'HK': 'Hong Kong', 'TW': 'Taiwan', 'IS': 'Iceland', 'LU': 'Luxe
    'PL': 'Poland', 'HR': 'Croatia', 'RS': 'Serbia', 'UA': 'Ukraine', 'RO': 'Roman
    'HR': 'Croatia', 'SK': 'Slovakia', 'SI': 'Slovenia', 'CZ': 'Czech Republic',
    'LV': 'Latvia', 'LT': 'Lithuania', 'UA': 'Ukraine', 'BY': 'Belarus', 'MD': 'Mo
    'GE': 'Georgia', 'AZ': 'Azerbaijan', 'KG': 'Kyrgyzstan', 'UZ': 'Uzbekistan',
    'TM': 'Turkmenistan', 'MN': 'Mongolia', 'LA': 'Laos', 'KH': 'Cambodia', 'NP':
    'PK': 'Pakistan', 'AF': 'Afghanistan', 'LK': 'Sri Lanka', 'MM': 'Myanmar', 'BD
    'AL': 'Albania', 'JM': 'Jamaica', 'CO': 'Colombia', 'CM': 'Cameroon'
}

#converting the region column to strings
all_combined['region'] = all_combined['region'].astype(str)

#making a unique movies df
unique_movies = all_combined.drop_duplicates(subset='title')

#grouping by country and sorting for highest sum profits
reg_prof = unique_movies.groupby('region')['profits'].sum().sort_values(ascending=

#mapping each country abbreviation to the country's full name
reg_prof.index = reg_prof.index.map(lambda x: imdb_region_dict.get(x, str(x)))

#plotting a bar graph for the data
fig, ax = plt.subplots(figsize=(15, 5))
ax.bar(reg_prof.index, reg_prof.values, color='blue')
ax.set_title('Bar Graph of Profits by Region')
ax.set_xlabel('Region')
ax.set_ylabel('Profits')
plt.xticks(rotation=90)
plt.show()
```



Insights from the graph above

We can see that by far the highest profits were made in the United States. This is probably

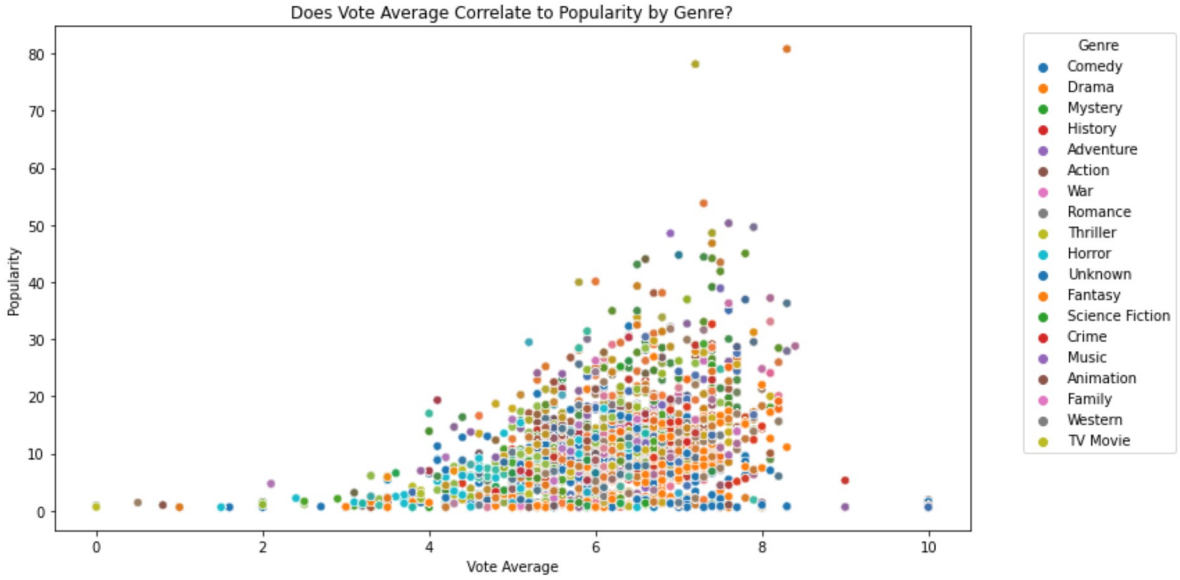
due to the fact that the datasets are imbalanced towards movies produced in America. We can see that the Canada, Italy, France, Germany, United Kindgom are following America, these are western countries where a large proportion of the population make movies. We can conclude that if we make movies in english we are likely to make money from western english speaking countries.

However it would be better to have a dataset with more movies in other languages.

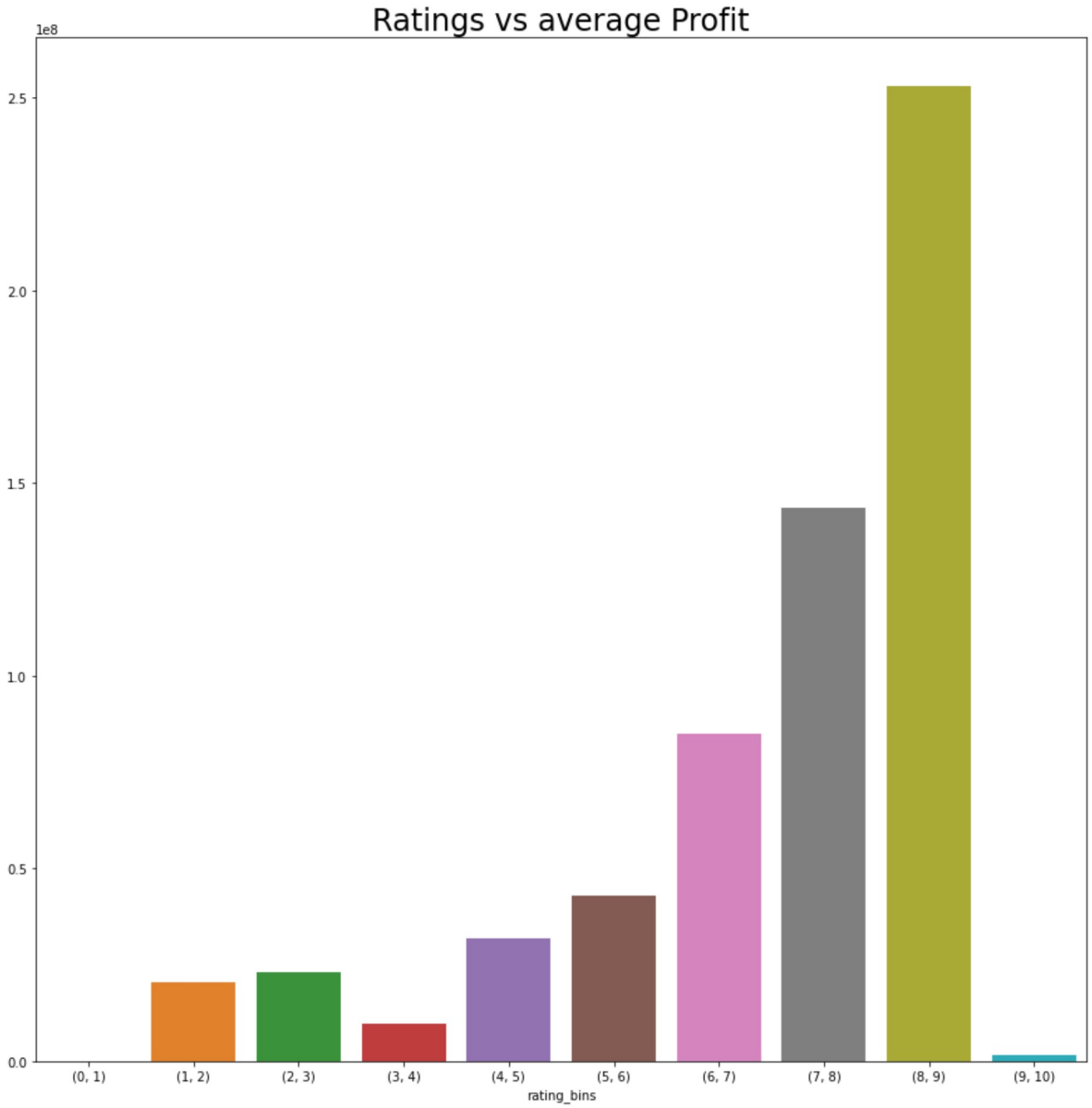
Next we will make a graph of Correlation Between Average Vote and Popularity

We will then make the each point correspond to the genre of the movie. The colors will help us see if the genre is connected to the average vote and popularity

In [28]:
```
genre_dict = {
    28: 'Action', 12: 'Adventure', 16: 'Animation', 35: 'Comedy',
    80: 'Crime', 18: 'Drama', 10751: 'Family', 14: 'Fantasy',
    36: 'History', 27: 'Horror', 10402: 'Music', 9648: 'Mystery',
    10749: 'Romance', 878: 'Science Fiction', 10770: 'TV Movie',
    53: 'Thriller', 10752: 'War', 37: 'Western'
}

# Cleaning and preparing the genre names
all_combined['genre_ids'] = all_combined['genre_ids'].apply(
    lambda x: [int(i) for i in str(x).strip('[]').split(', ') if i.isdigit()] if :
)
all_combined['genre_names'] = all_combined['genre_ids'].apply(
    lambda ids: [genre_dict.get(genre_id, 'Unknown') for genre_id in ids]
)

# Explode genres for independent rows
df = all_combined.explode('genre_names').dropna(subset=['genre_names', 'popularity

# Plot: vote_average vs popularity grouped by genres
plt.figure(figsize=(12, 6))
sns.scatterplot(data=df, x='vote_average', y='popularity', hue='genre_names', pale

plt.title('Does Vote Average Correlate to Popularity by Genre?')
plt.xlabel('Vote Average')
plt.ylabel('Popularity')
plt.legend(title='Genre', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
```



In [29]:
```
#group by averagerating vs profits per movie
#im gonna create bins for the ratings 1,2,3,4 to 10
rating_bins = [1,2,3,4,5,6,7,8,9,10]
all_combined['rating_bins'] = pd.cut(all_combined['averagerating'],bins=[0,1,2,3,4

#then I want to make sure the df is only showing unique movies
unique_movies = all_combined.drop_duplicates(subset='title')


#making a table with the index as the rating_bins and columns for the number of mo
ratings_profits_mean = unique_movies.groupby('rating_bins').agg(num_movies=('movie
                                                                ,mean_profit=(



#creating x_tick labels
rating_ranges=  [(0,1),(1,2),(2,3),(3,4),(4,5),(5,6),(6,7),(7,8),(8,9),(9,10)]

#plotting the graph
fig,ax = plt.subplots(figsize=(15,15))
sns.barplot(data=ratings_profits_mean,x=ratings_profits_mean.index,y=ratings_profi
ax.set_xticklabels(rating_ranges)
ax.set_title('Ratings vs average Profit',fontdict={'fontsize':24});
```



Insights from the graph above:

The general pattern that we see is as the ratings increase the mean profits also increase.

From the cell below we can see that the reason why the 9-10 rating movies have a low profit is because there is only one movie that had this rating. From the mean popularity column we can see that the 9-10 movie wasn't very popular. This could be due to the fact that the best movies aren't the most entertaining. But we can see that the 8-9 movie ratings are the most popular. We must make a movie that has a balance between being technically good while remaining entertaining to the average consumer.

In [30]:
```
ratings_profits_mean
```

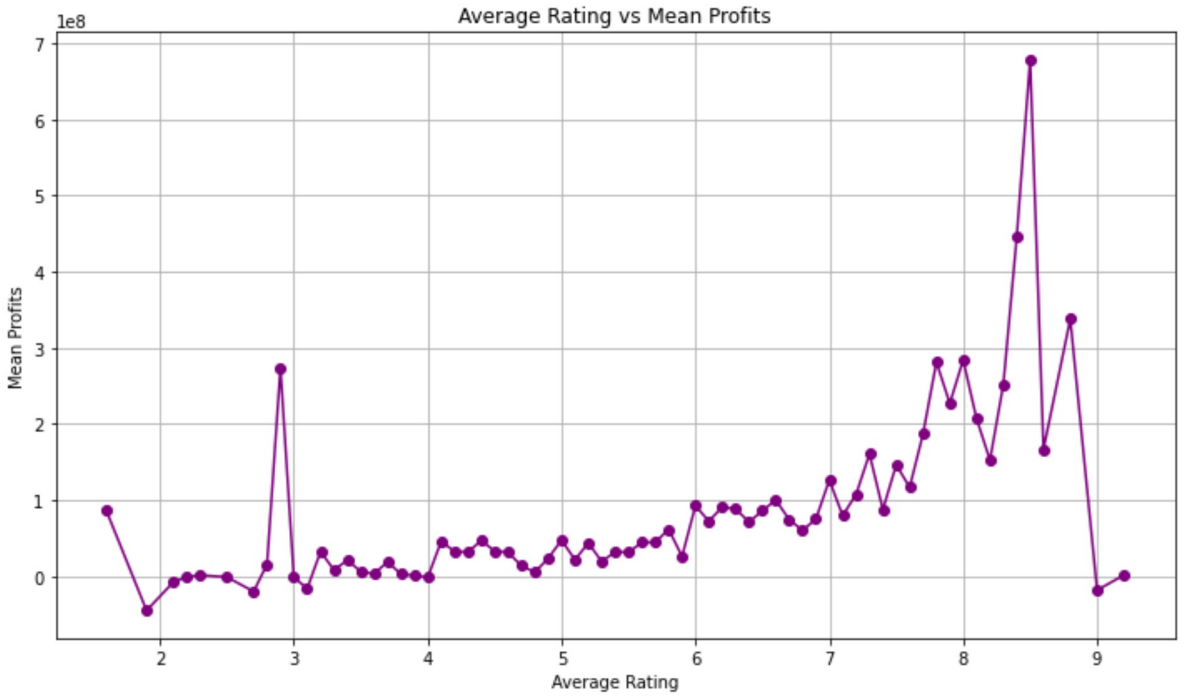
Out[30]:

rating_bins	num_movies	mean_profit	mean_popularity
(8, 9]	46	2.529563e+08	17.486326
(7, 8]	364	1.435457e+08	13.493940
(6, 7]	694	8.514308e+07	11.455536

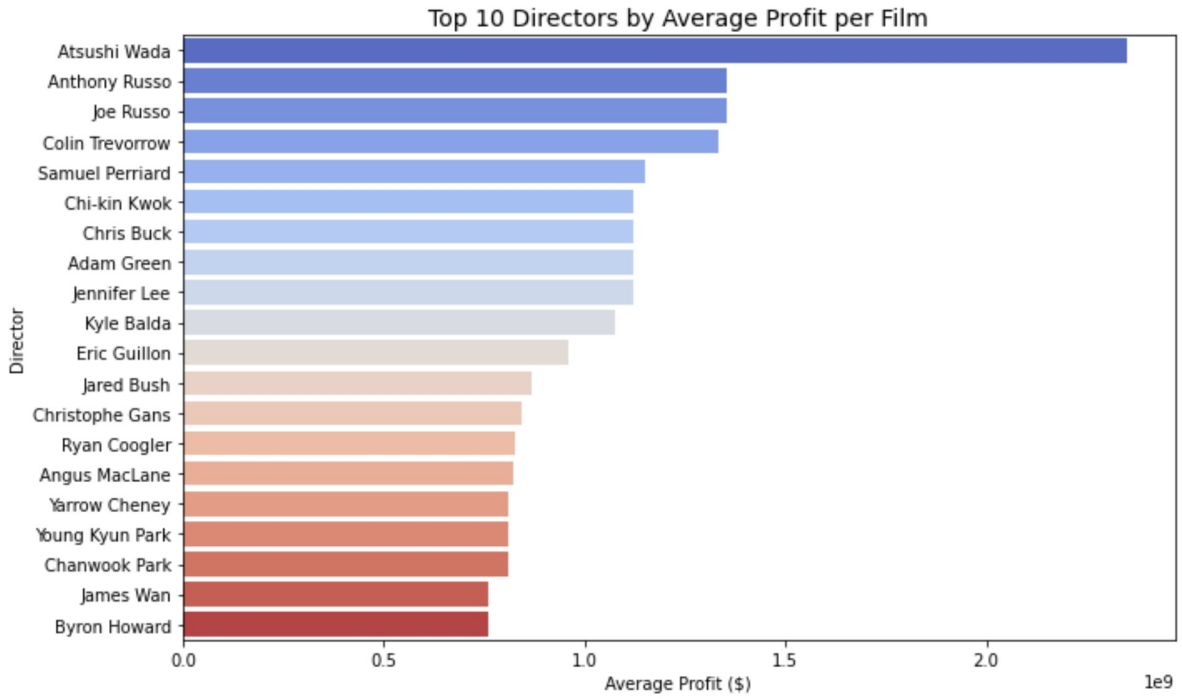
rating_bins	num_movies	mean_profit	mean_popularity
(5, 6]	434	4.286585e+07	9.227673
(4, 5]	166	3.199411e+07	7.056042
(2, 3]	11	2.295958e+07	2.690455
(1, 2]	2	2.055392e+07	6.653500
(3, 4]	63	9.696038e+06	5.370143
(9, 10]	1	1.495262e+06	5.577000
(0, 1]	0	NaN	NaN

The graph below is a line graph that shows the same information as the one above.

In [31]:
```
# Group by average_vote and compute mean profits
grouped = unique_movies.groupby('averagerating')['profits'].mean().reset_index()

# Convert to numpy arrays to avoid indexing issues (as a safeguard)
x = grouped['averagerating'].values
y = grouped['profits'].values

# Plotting
plt.figure(figsize=(10, 6))
plt.plot(x, y, marker='o', linestyle='-', color='purple')
plt.title('Average Rating vs Mean Profits')
plt.xlabel('Average Rating')
plt.ylabel('Mean Profits')
plt.grid(True)
plt.tight_layout()
plt.show()
```



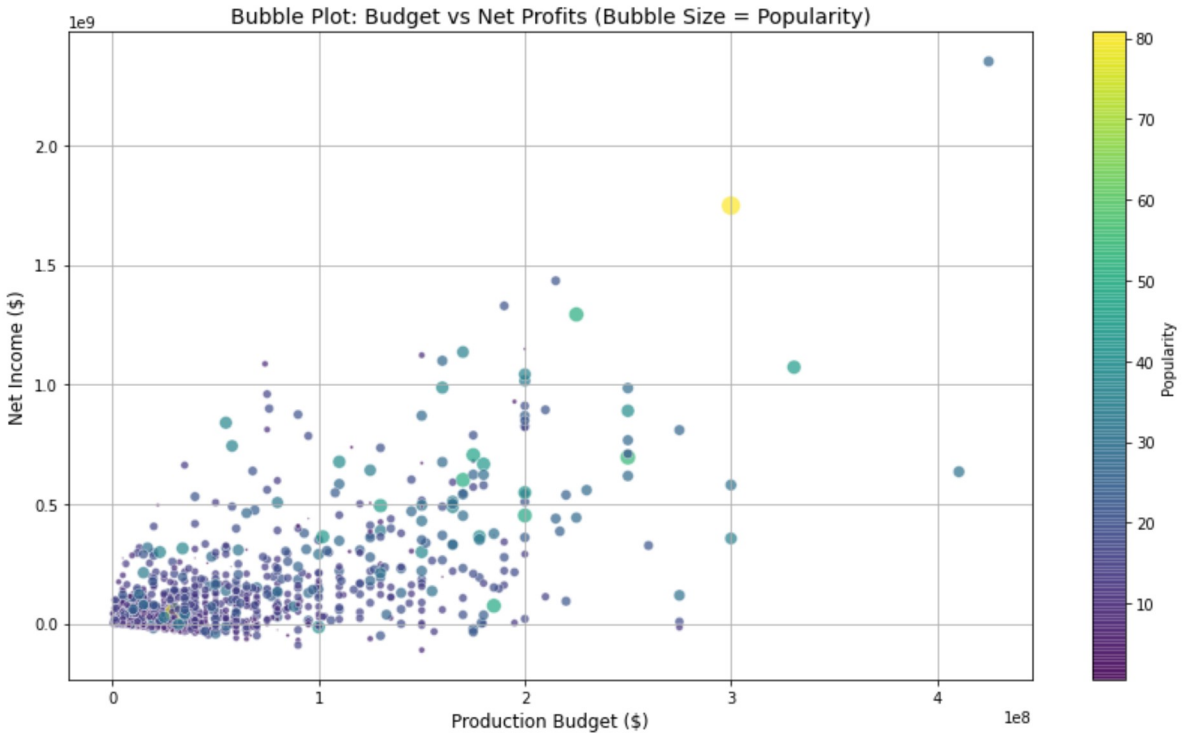
In [32]:
```
#Step 1: Drop rows where any of the key columns are missing
df_profit = all_combined.dropna(subset=['category', 'primary_name', 'profits'])

# Step 2: Filter for directors only
df_directors = all_combined[all_combined['category'] == 'director']

# Step 3: Group by director's name and calculate average profit
director_profit = df_directors.groupby('primary_name')['profits'].mean().reset_inc

# Step 4: Sort by average profit (highest first)
director_profit = director_profit.sort_values(by='profits', ascending=False)

# Step 5: Display top 20 most profitable directors
director_profit.head(20)

# Take top 10 profitable directors
top_profit_directors = director_profit.head(20)

plt.figure(figsize=(10, 6))
sns.barplot(data=top_profit_directors, x='profits', y='primary_name', palette='coo
plt.title('Top 10 Directors by Average Profit per Film', fontsize=14)
plt.xlabel('Average Profit ($)')
plt.ylabel('Director')
plt.tight_layout()
plt.show()
```



Bubble Plot: Budget vs Net Income (Unique Movies)

This bubble chart visualizes the relationship between a movie's **production budget** and its resulting **net income**, considering only **unique movies** to eliminate duplication from multiple categories like cast and crew. ter

In [33]:
```
# Step 1: Drop duplicates to isolate unique movies (based on title or movie_id)
unique_movies_df = all_combined.drop_duplicates(subset='movie')

# Step 2: Prepare DataFrame for plotting
plot_df = unique_movies_df[['production_budget', 'profits', 'popularity']].dropna(

# Step 3: Create bubble plot
plt.figure(figsize=(12, 7))
scatter = plt.scatter(
    plot_df['production_budget'],
    plot_df['profits'],
    s=plot_df['popularity'] * 2,  # Bubble size
    alpha=0.7,
    c=plot_df['popularity'],      # Color by popularity
    cmap='viridis',
    edgecolors='w',
    linewidth=0.5
)

plt.title('Bubble Plot: Budget vs Net Profits (Bubble Size = Popularity)', fontsi
plt.xlabel('Production Budget ($)', fontsize=12)
plt.ylabel('Net Income ($)', fontsize=12)
plt.colorbar(scatter, label='Popularity')
plt.grid(True)
plt.tight_layout()
plt.show()
```



Observations

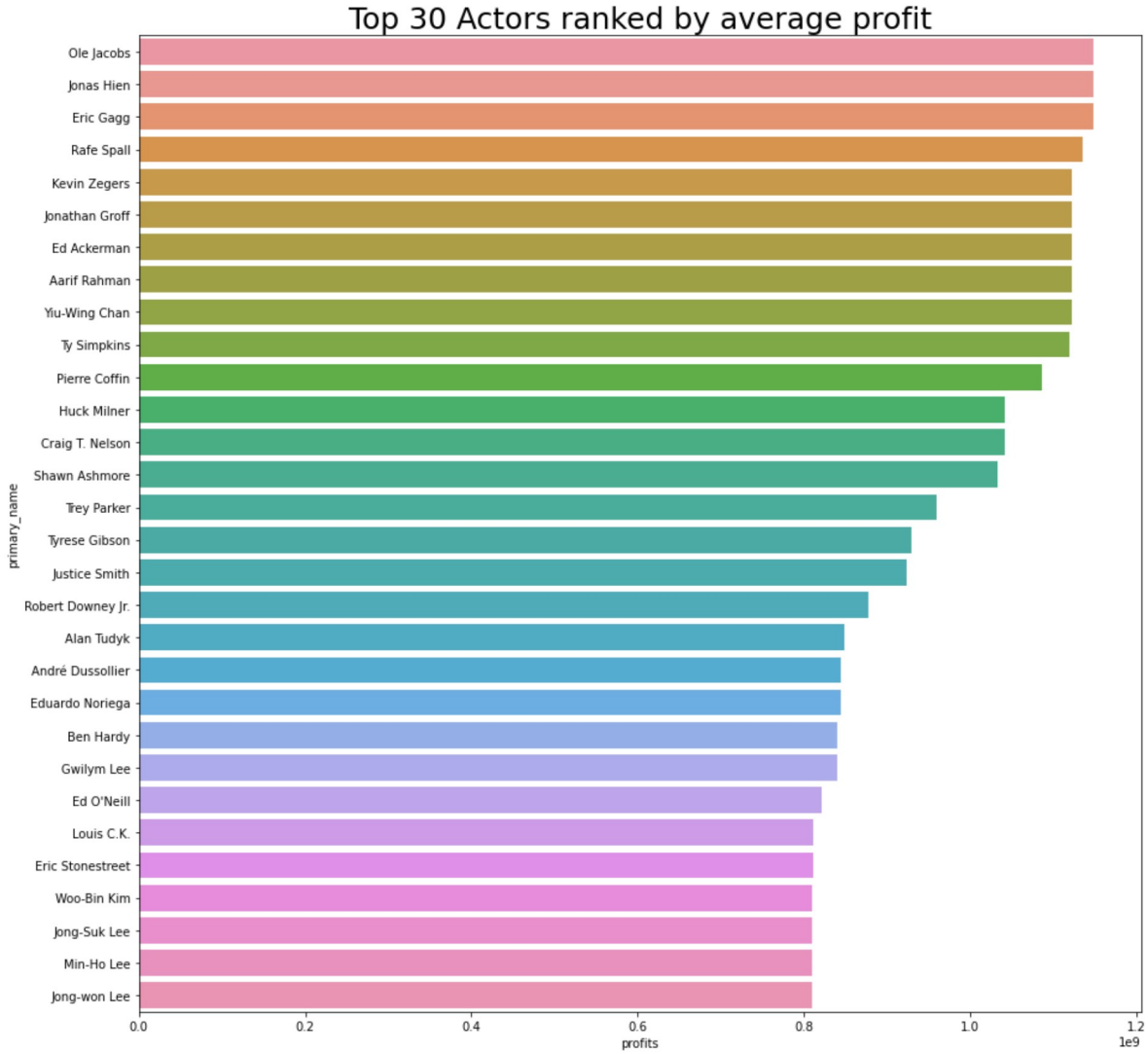
- Most movies with lower production budgets cluster around low-to-moderate net incomes.
- A few high-budget films yield exceptionally high net incomes, indicating big-budget blockbusters can pay off.
- Outliers with large, brightly colored bubbles highlight movies that were both popular and financially successful.
- Some movies with high budgets still performed poorly (bubbles close to the X-axis),

underscoring financial risk.

This plot helps spot financial winners and underperformers at a glance while factoring in audience appeal.

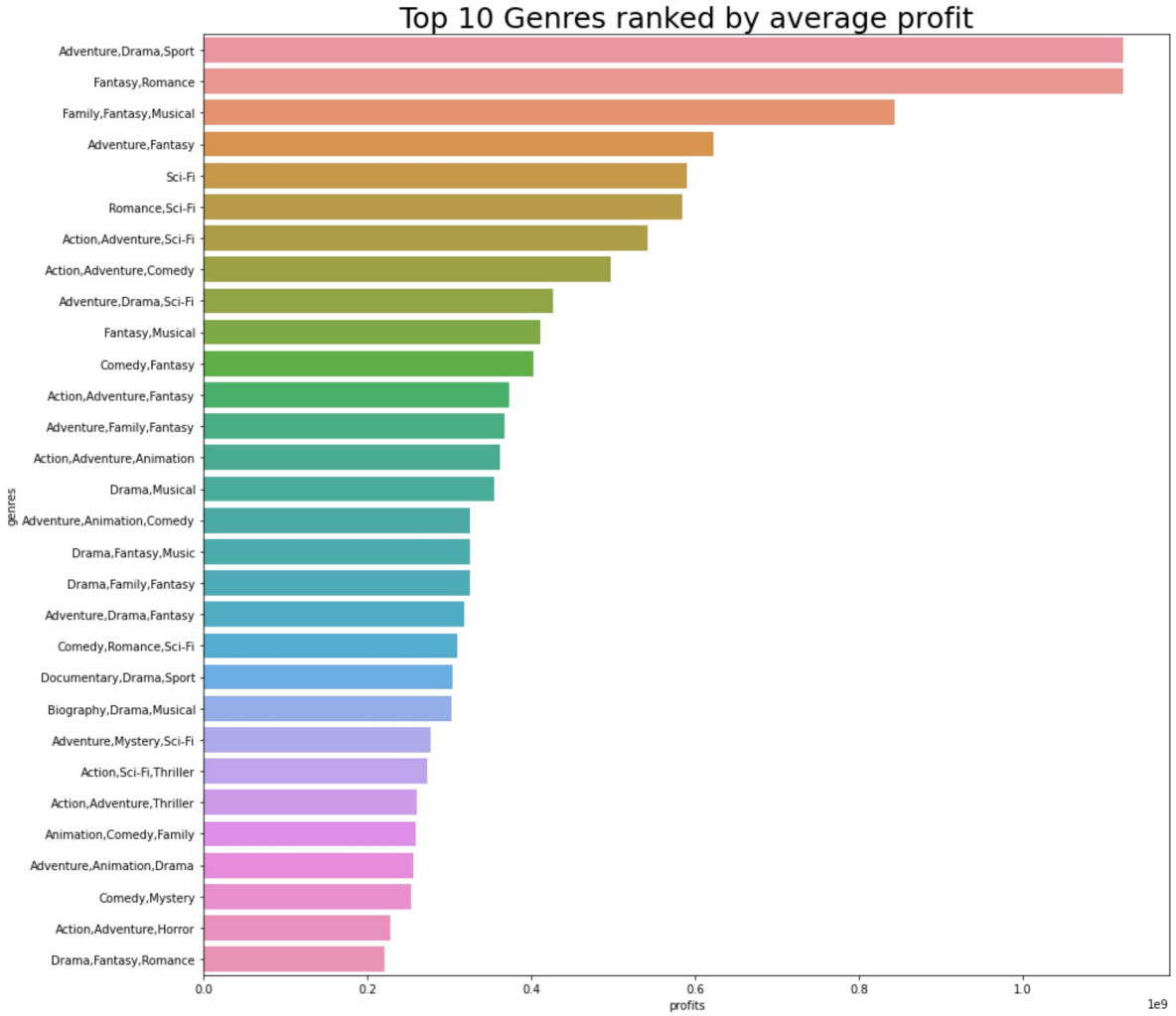
In [34]:
```
#average profits per director
directors_id_groupby = all_combined[all_combined['category'] == 'actor'].groupby(
top_20_directors_by_profit = directors_id_groupby.head(30)
top_20_directors_by_profit

fig,ax = plt.subplots(figsize=(15,15))
sns.barplot(data = top_20_directors_by_profit,x='profits',y=top_20_directors_by_pr
ax.set_title('Top 30 Actors ranked by average profit',fontdict={'fontsize':25})
```

Out[34]:
```
Text(0.5, 1.0, 'Top 30 Actors ranked by average profit')
```

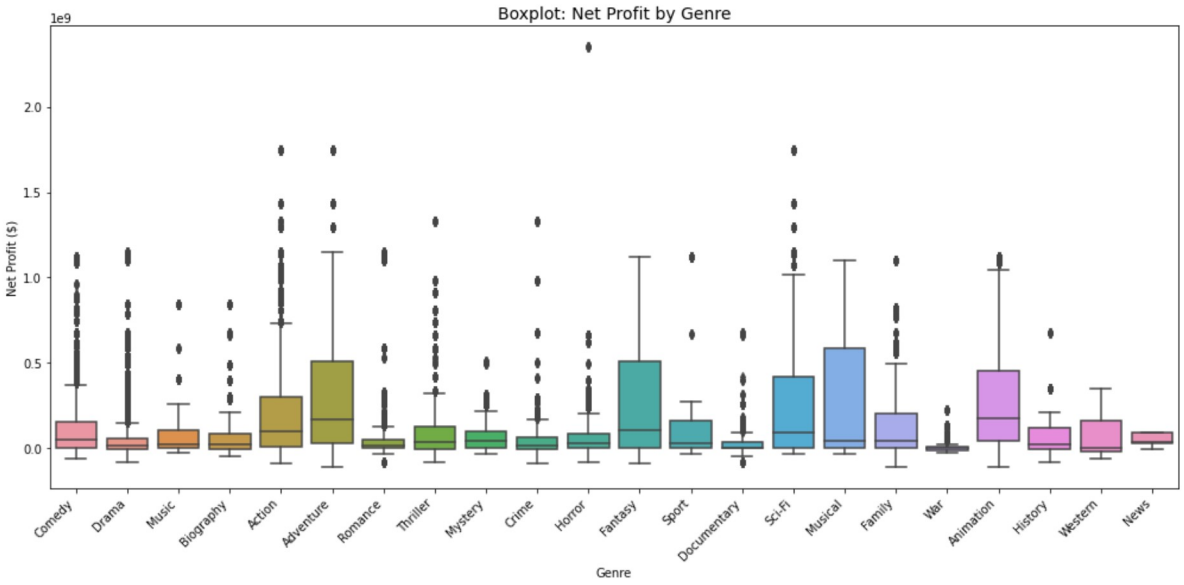


In [35]:
```
mean_genre_groupby = all_combined.groupby('genres').mean().sort_values('profits',a
top_30_genres_average_profits = mean_genre_groupby.head(30)

fig,ax = plt.subplots(figsize=(15,15))
sns.barplot(data = top_30_genres_average_profits,x='profits',y=top_30_genres_avera
ax.set_title('Top 10 Genres ranked by average profit',fontdict={'fontsize':25})
```

Out[35]:
```
Text(0.5, 1.0, 'Top 10 Genres ranked by average profit')
```



In [36]:
```
all_combined_exploded['genres'] = all_combined_exploded['genres'].str.strip()

# Drop rows with missing or zero net income
boxplot_df = all_combined_exploded[['genres', 'profits']].dropna()
boxplot_df = boxplot_df[boxplot_df['profits'] != 0]

# Plot boxplot
plt.figure(figsize=(14, 7))
sns.boxplot(data=boxplot_df, x='genres', y='profits')
plt.xticks(rotation=45, ha='right')
plt.title('Boxplot: Net Profit by Genre', fontsize=14)
plt.xlabel('Genre')
plt.ylabel('Net Profit ($)')
plt.tight_layout()
plt.show()
```

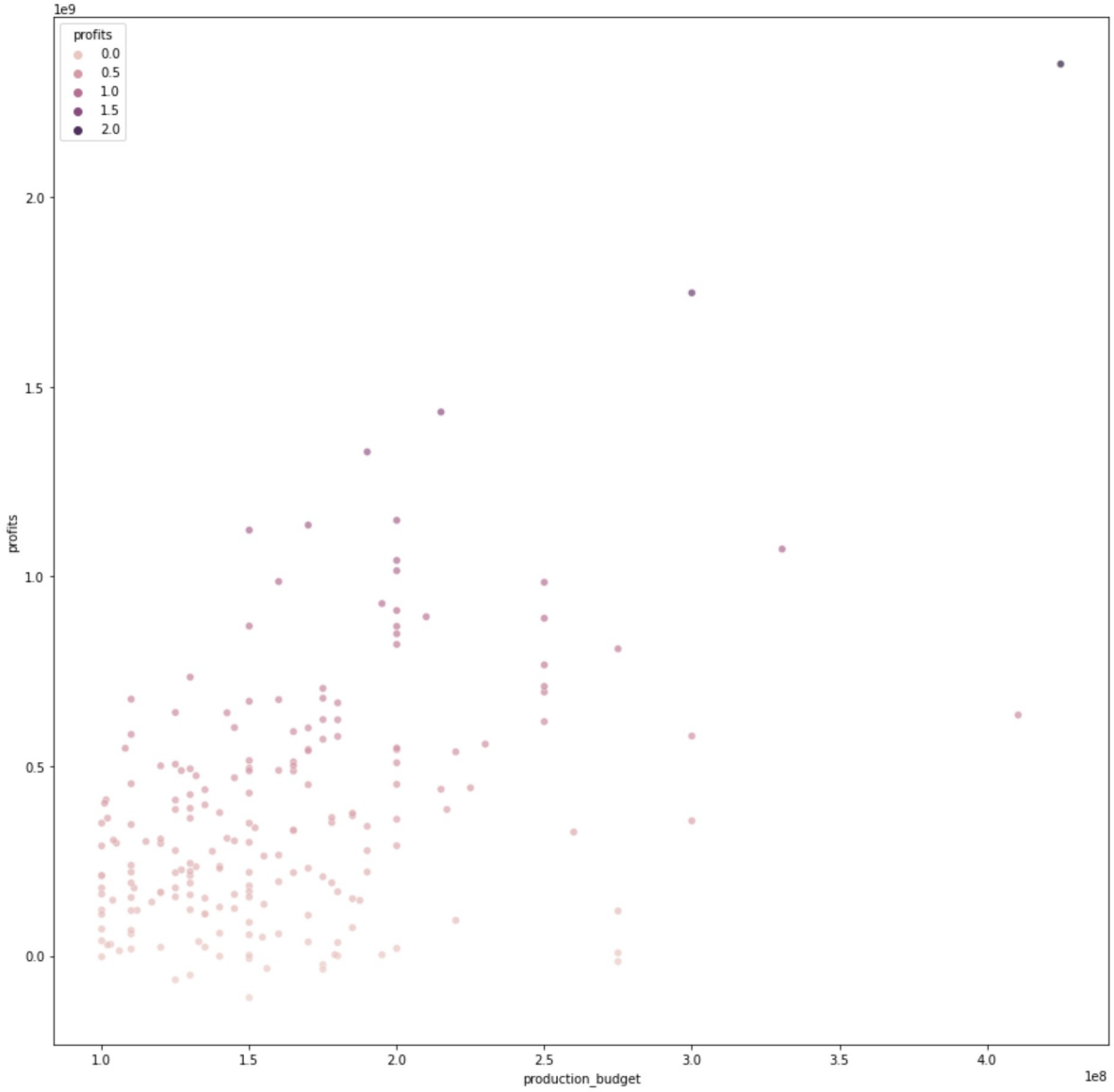


The boxplot above visualizes the distribution of net income across different movie genres. Each box represents the interquartile range (IQR), with the line inside indicating the median net income for that genre. Outliers are plotted as individual points.

This visualization helps identify which genres tend to generate higher profits. For example, **genres such as Adventure, Animation, Musical, Sci-Fi and Fantasy**show both higher medians and wider spreads, indicating they can be very lucrative but also variable. In contrast, genres like Biography, Romance, and War generally display lower net income and narrower distributions.

In [37]:
```
fig,ax = plt.subplots(figsize=(15,15))
movie_budgets_ranking  = all_combined.groupby('title').mean().sort_values('product
top_200_movie_budgets_ranking = movie_budgets_ranking.head(200)

sns.scatterplot(data=top_200_movie_budgets_ranking,x='production_budget',y='profit
```

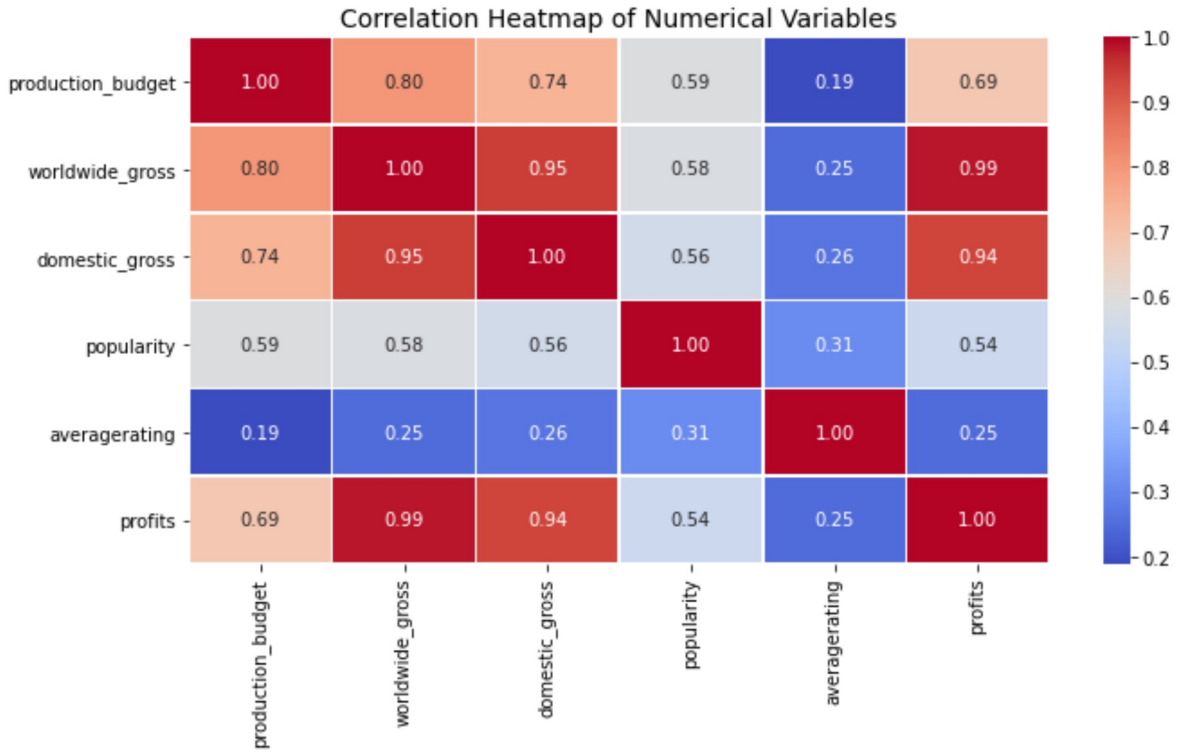
Out[37]:
```
<AxesSubplot:xlabel='production_budget', ylabel='profits'>
```



Correlation Heatmap: Numerical Variables.

This heatmap illustrates the relationships between key numerical features for unique movies in our dataset. Correlation values range from -1 (perfect negative correlation) to 1 (perfect positive correlation).

In [38]:
```
unique_movies_df = all_combined.drop_duplicates(subset='movie')

# Select numerical columns to check correlation
correlation_df = unique_movies_df[['production_budget', 'worldwide_gross', 'domest

# Compute correlation matrix
corr_matrix = correlation_df.corr()

# Plot heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title('Correlation Heatmap of Numerical Variables', fontsize=14)
plt.tight_layout()
plt.show()
```



Observations:

- **WorldWide Gross vs Net Income (0.99)**:
  As expected, there's a nearly perfect positive correlation. Higher worldwide revenue almost always results in higher profit.

- **Production Budget vs Worldwide Gross (0.80)**:
  A strong positive relationship exists — movies with larger budgets tend to earn more globally. However, high spending doesn't always guarantee higher profit.

- **Production Budget vs Net Income (0.69)**:
  Bigger budgets often lead to higher net income, although there's still some variability.

- **Popularity (0.54–0.59 with financials)**:
  Popularity shows moderate correlation with budget, revenue, and net income, suggesting that buzz and visibility contribute to a movie's financial performance.

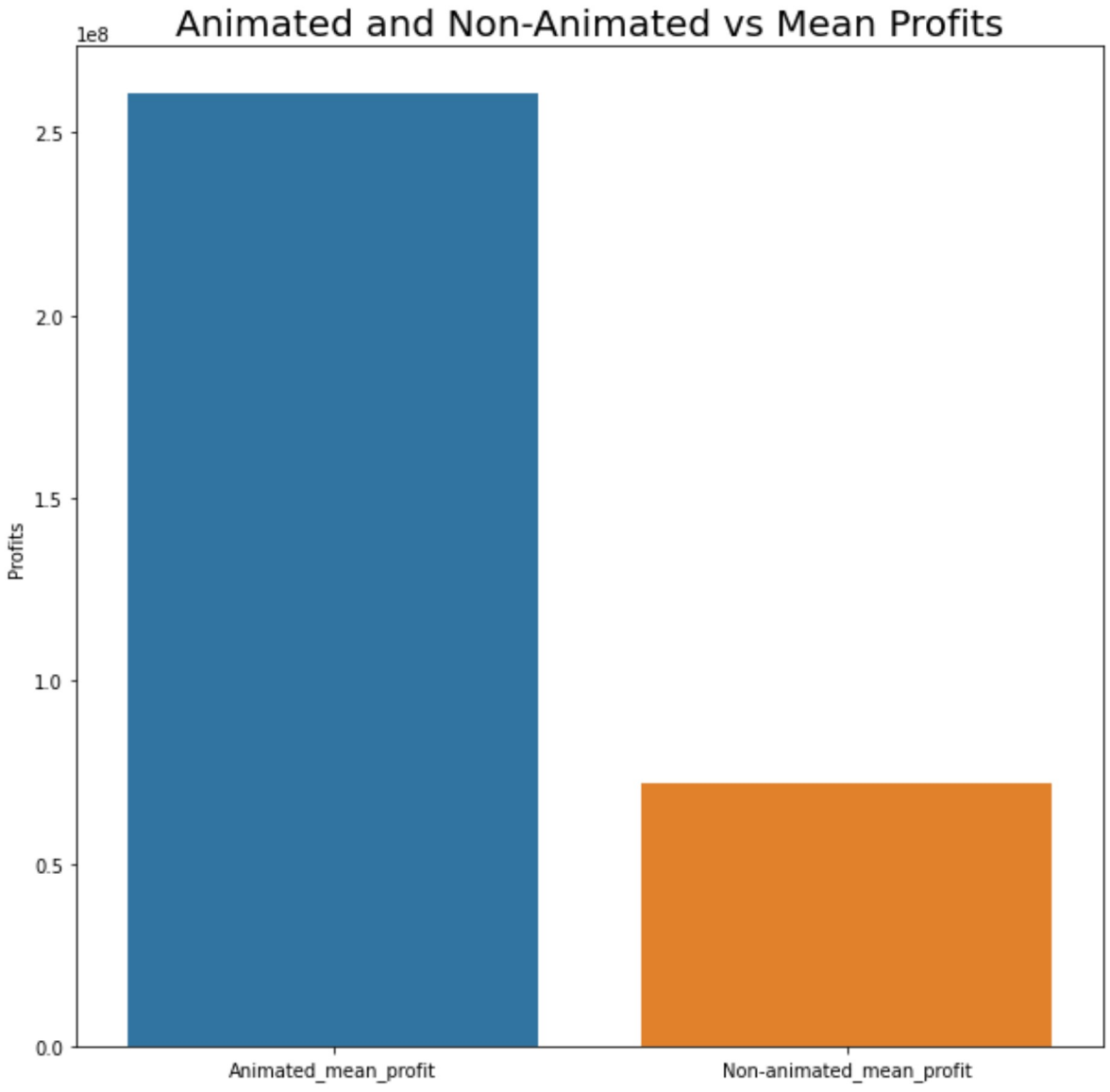
- **Average Rating (0.19–0.31 with other metrics)**:
  Ratings show weak correlation with financial outcomes. This indicates that a highly-rated film is not necessarily a top earner.

Add a visual showing animation vs real movies.

We will look at the genres column. Any row that has the string animation will be counted as that We will compare that to the rest of the dataset

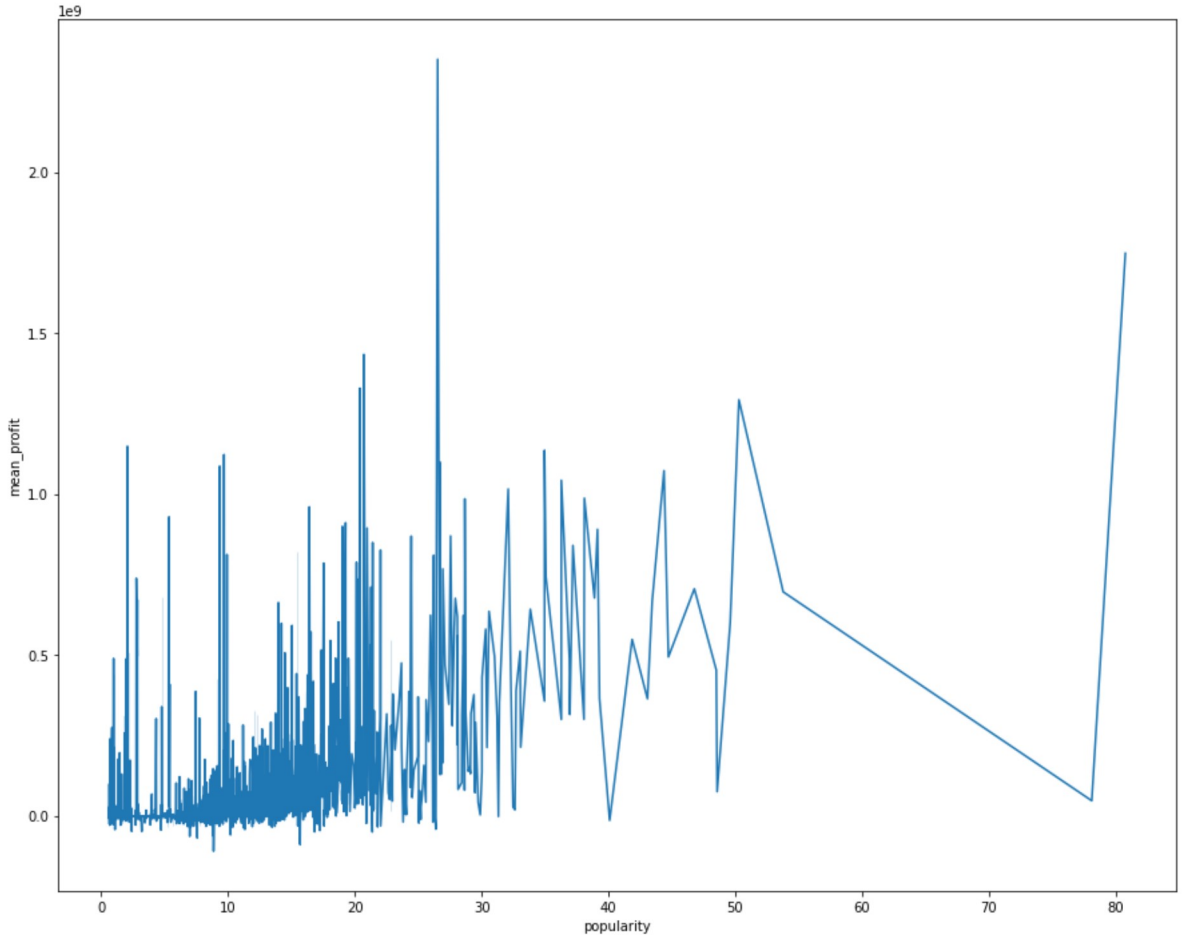
In [39]:
```
unique_movies  = all_combined.drop_duplicates(subset='title')

animation_movies = unique_movies[unique_movies['genres'].str.contains('animation'
animation_movies_profits = animation_movies[['title','profits']]
animation_mean_profit = animation_movies_profits['profits'].mean()
# animation_movies['genres'].value_counts()
not_animated = unique_movies[unique_movies['genres'].str.contains('animation',case
not_animated_profits = not_animated[['title','profits']]
not_animated_mean_profit = not_animated_profits['profits'].mean()
# all_combined['genres'].str.contains('animation')

# df = pd.DataFrame(data= {'Animated_mean_profit':[animation_mean_profit],
#                     'Non-animated_mean_profit': [not_animated_mean_profit]}).tra

df = pd.DataFrame(data= {'Profits':[animation_mean_profit,not_animated_mean_profi
                  ,index=['Animated_mean_profit','Non-animated_mean_profit'])

# df

fig,ax = plt.subplots(figsize=(10,10))
sns.barplot(data=df,x=df.index,y=df['Profits'])
plt.title('Animated and Non-Animated vs Mean Profits',fontdict={'fontsize':20})
```

Out[39]: Text(0.5, 1.0, 'Animated and Non-Animated vs Mean Profits')



In [40]:
```
#popular movies vs Profits

unique_movies = all_combined.drop_duplicates(subset='title')

movies_ranked_profit = unique_movies.groupby('title').agg(mean_profit = ('profits
                                                          ,popularity=('popularity
                                                          ).sort_values('mean_pro

fig,ax= plt.subplots(figsize=(15,12))
sns.lineplot(data=movies_ranked_profit,y='mean_profit',x='popularity')
```

Out[40]: <AxesSubplot:xlabel='popularity', ylabel='mean_profit'>



The graph above ranked profit vs popularity of each movie

In [41]:
```
movies_ranked_profit.head(20)
```

Out[41]:

title	mean_profit	popularity
Avatar	2.351345e+09	26.526
Avengers: Infinity War	1.748134e+09	80.773
Jurassic World	1.433855e+09	20.709
Furious 7	1.328723e+09	20.396
The Avengers	1.292936e+09	50.289
Black Panther	1.148258e+09	2.058
Jurassic World: Fallen Kingdom	1.135773e+09	34.958

title	mean_profit	popularity
Frozen	1.122470e+09	9.678
Beauty and the Beast	1.099200e+09	26.701
Minions	1.086336e+09	9.338
Avengers: Age of Ultron	1.072414e+09	44.383
Incredibles 2	1.042521e+09	36.286
Iron Man 3	1.015392e+09	32.093
Aquaman	9.868946e+08	38.102
The Fate of the Furious	9.848463e+08	28.668
Despicable Me 3	9.597278e+08	16.407
Transformers: Dark of the Moon	9.287905e+08	5.339
Skyfall	9.105270e+08	19.270
Despicable Me 2	8.992168e+08	19.014
Transformers: Age of Extinction	8.940391e+08	20.961

In [42]:
```
# Monthly Profit Trends (Line Plot)
fig,ax = plt.subplots(figsize=(15,15))
all_combined['release_month'] = pd.to_datetime(all_combined['release_date_x']).dt
monthly_profit = all_combined.groupby('release_month')['profits'].mean()
# monthly_profit
months = ['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec']

sns.barplot(x=monthly_profit.index, y=monthly_profit.values)
ax.set_xticklabels(months)
plt.title(label='Mean Profits per Month',fontdict={'fontsize':25})
plt.xlabel(fontdict={'fontsize':15},xlabel='release month')
plt.ylabel(fontdict={'fontsize':15},ylabel='profits');
```

Step 5. Hypothesis Testing

In [43]:
```
def cohen_d(a, b):
    n1, n2 = len(a), len(b)
    pooled_std = np.sqrt(((n1-1)*np.var(a, ddof=1) + (n2-1)*np.var(b, ddof=1)) / (
    return (np.mean(a) - np.mean(b)) / pooled_std
```

Question 1: Do movies with top-rated directors (Group A) generate higher profits than those with lower-rated directors (Group B)? Null Hypothesis: Movies with top-rated directors (Group A) generate equal profit compared to those with lower-rated directors (Group B)? Alternative Hypothesis: Movies with top-rated directors (Group A) generate more profits compared to those with lower-rated directors (Group B)?

Groups:

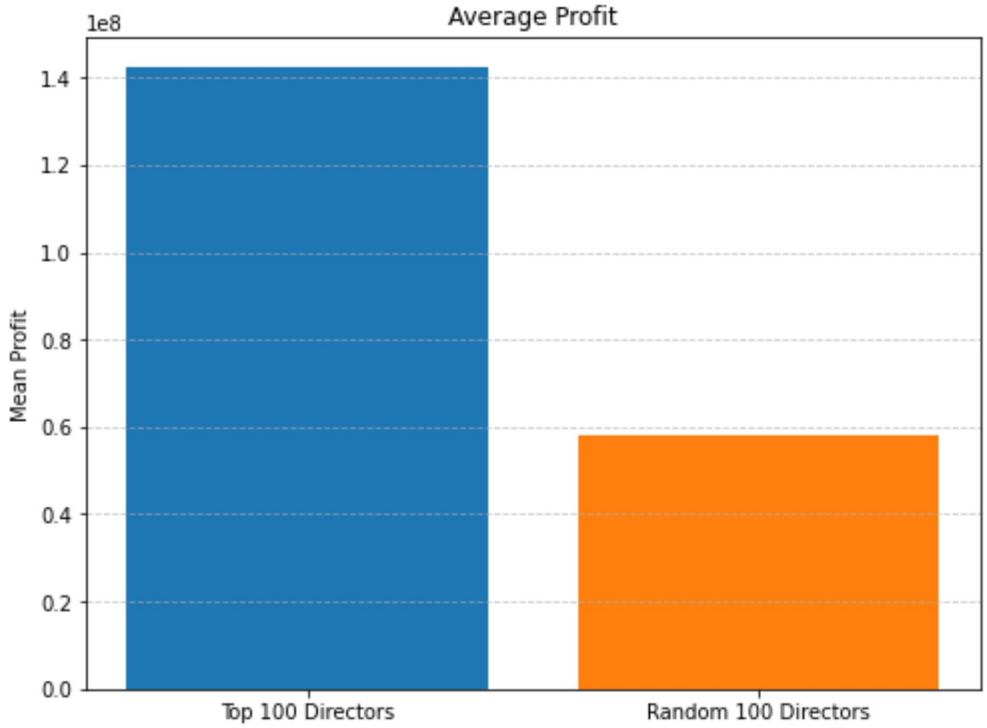
```
#####   Group A: Movies directed by directors in the top 10% by
averagerating or profits.

#####    Group B: Movies directed by others.

#### Metric: Compare mean profits or worldwide_gross.
```

Method: Use a two-sample t-test to check if the difference is statistically significant.

In [44]:
```
directors_top_100 = all_combined[all_combined['category'] == 'director'].groupby(
top_100_directors_average_rating_profits = directors_top_100['profits'].head(100)
# mean_profits_top_50_directors = top_50_directors_average_rating_profits.mean()

not_top_100_directors_average_rating_profits = directors_top_100.tail(834)
random_sample_100_not_top_100 = not_top_100_directors_average_rating_profits['prof


alpha = 0.05
t_statistic, p_value = stats.ttest_ind(top_100_directors_average_rating_profits,ra
print('t-stat: ',t_statistic)
print('p_value: ',p_value)

if p_value < alpha:
    print('Null Hypothesis is rejected. Results are statistically significant at a
    print('Movies with more popular directors do not generate equal profits')
else:
    print('Failed to reject null hypothesis. Results are not statistically signifi

effect_size = cohen_d(top_100_directors_average_rating_profits,random_sample_100_r
print('The effect size of the following was: ',effect_size)
print('A cohen d value of 0.44 indicates that the difference between having a top
```

```
t-stat:  2.2588207403497416
p_value:  0.02498449461007823
Null Hypothesis is rejected. Results are statistically significant at alpha = 0.05
Movies with more popular directors do not generate equal profits
The effect size of the following was:  0.319445492597224
A cohen d value of 0.44 indicates that the difference between having a top 100 dir
ector is noticable
 and is something the studio should look out for
```

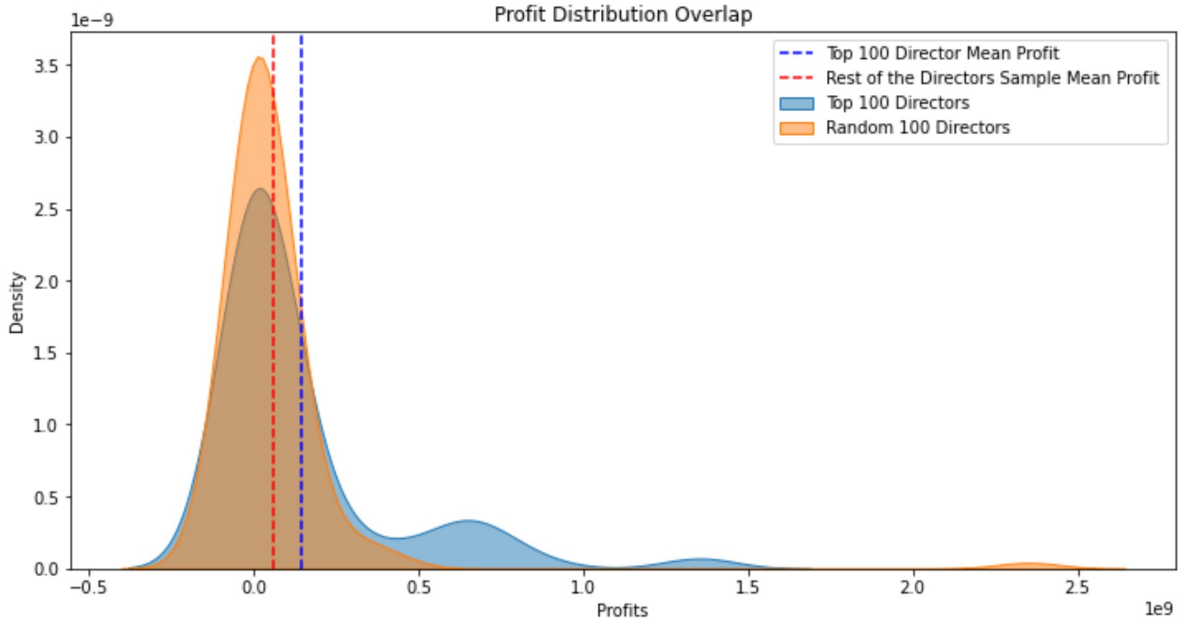
In [45]:
```
means = [top_100_directors_average_rating_profits.mean(),
         random_sample_100_not_top_100.mean()]
# errors = [top_100_directors_average_rating_profits.sem() * 1.96,  # 95% CI
#           random_sample_100_not_top_100.sem() * 1.96]

plt.figure(figsize=(8, 6))
plt.bar(['Top 100 Directors', 'Random 100 Directors'], means,
        capsize=10, color=['#1f77b4', '#ff7f0e'])
plt.ylabel('Mean Profit')
plt.title('Average Profit')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



Observation for the graph above and graph below

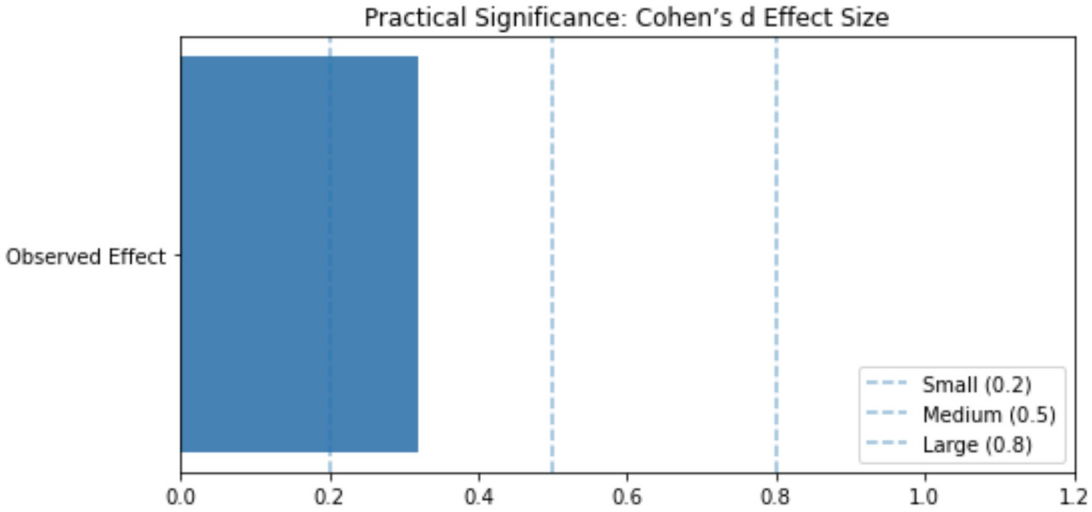
- Top directors' mean is higher: Prioritize collaborations with highly-rated directors.

In [46]:
```
plt.figure(figsize=(12, 6))
sns.kdeplot(top_100_directors_average_rating_profits, label='Top 100 Directors', 
sns.kdeplot(random_sample_100_not_top_100, label='Random 100 Directors', fill=True
plt.axvline(top_100_directors_average_rating_profits.mean(), color='blue', linesty
plt.axvline(random_sample_100_not_top_100.mean(), color='red', linestyle='--',labe
plt.xlabel('Profits')
plt.title('Profit Distribution Overlap')
plt.legend()
plt.show()
```



Effect Size Visualization

In [47]:
```
# Cohen's d reference lines
effect_ref = {'Small': 0.2, 'Medium': 0.5, 'Large': 0.8}

plt.figure(figsize=(8, 4))
plt.barh(['Observed Effect'], [effect_size], color='steelblue')
for label, val in effect_ref.items():
    plt.axvline(val, linestyle='--', alpha=0.5, label=f'{label} ({val})')
plt.title('Practical Significance: Cohen's d Effect Size')
plt.xlim(0, 1.2)
plt.legend(loc='lower right')
plt.show()
```



Insight:

```
- A Cohen d effect size of 0.32 demonstrates that hiring a top
rated director has a small effect on the profits of a movie.
```

Hypothesis Test 2:

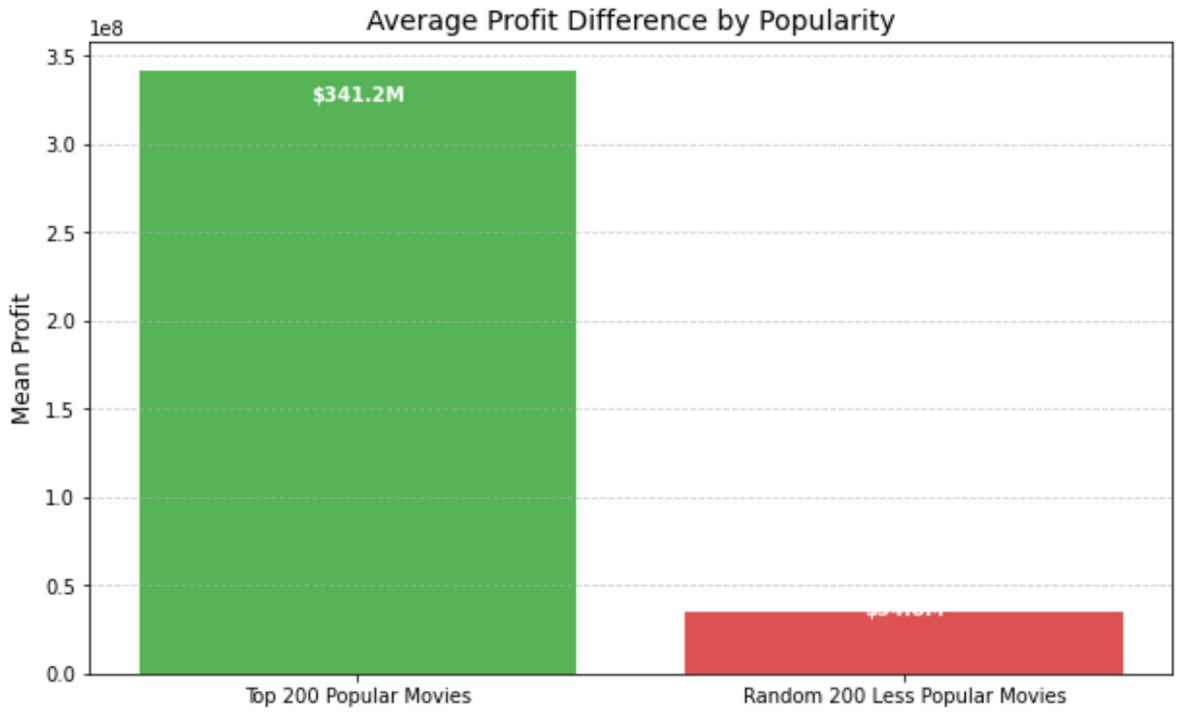
Null Hypothesis: The more popular movies make the don't make more profits as the less popular movies.

Alternative Hypothesis: The more popular movies make more profits than the less popular movies.

In [48]:
```
#the popularity column is for movie popularity.
#change the hypothesis test to check for movie popularity

#organize actors by popularity
#null hypothesis: more popular movies dont make more profits
#alternate: more popular movies make more profits.
movies_ranked_popularity = all_combined.groupby('title').mean().sort_values('popul
top_200_popular_movies = movies_ranked_popularity['profits'].head(200)
not_top_200_movies = movies_ranked_popularity.tail(1410)
random_sample_200_movies_not_top_200 = not_top_200_movies['profits'].sample(n=200,

alpha = 0.05
t_statistic, p_value = stats.ttest_ind(top_200_popular_movies,random_sample_200_mo
print('t-stat: ',t_statistic)
print('The positive t-statistic means that top rated movies make more profits than
print('p_value: ',p_value)

if p_value < alpha:
    print('Null Hypothesis is rejected. Results are statistically significant at a
    print('More popular movies do not make the same profits as less popular movies
else:
    print('Failed to reject null hypothesis. Results are not statistically signifi


effect_size = cohen_d(top_200_popular_movies,random_sample_200_movies_not_top_200)
print('The effect size of the following was: ',effect_size)
print('A cohen d value of 1.25 indicates that the difference between having a movi
'       \nthe studio should invest a large sum of the production budget into the a
```

```
t-stat:  12.256194189539704
The positive t-statistic means that top rated movies make more profits than lower
rated movies.
p_value:  1.605553663337206e-29
Null Hypothesis is rejected. Results are statistically significant at alpha = 0.0
5.
More popular movies do not make the same profits as less popular movies
The effect size of the following was:  1.2256194189539704
A cohen d value of 1.25 indicates that the difference between having a movie popul
arity is highly impactful when it comes to making profits
the studio should invest a large sum of the production budget into the advertising
of the movie and the hiring of popular actors.
```

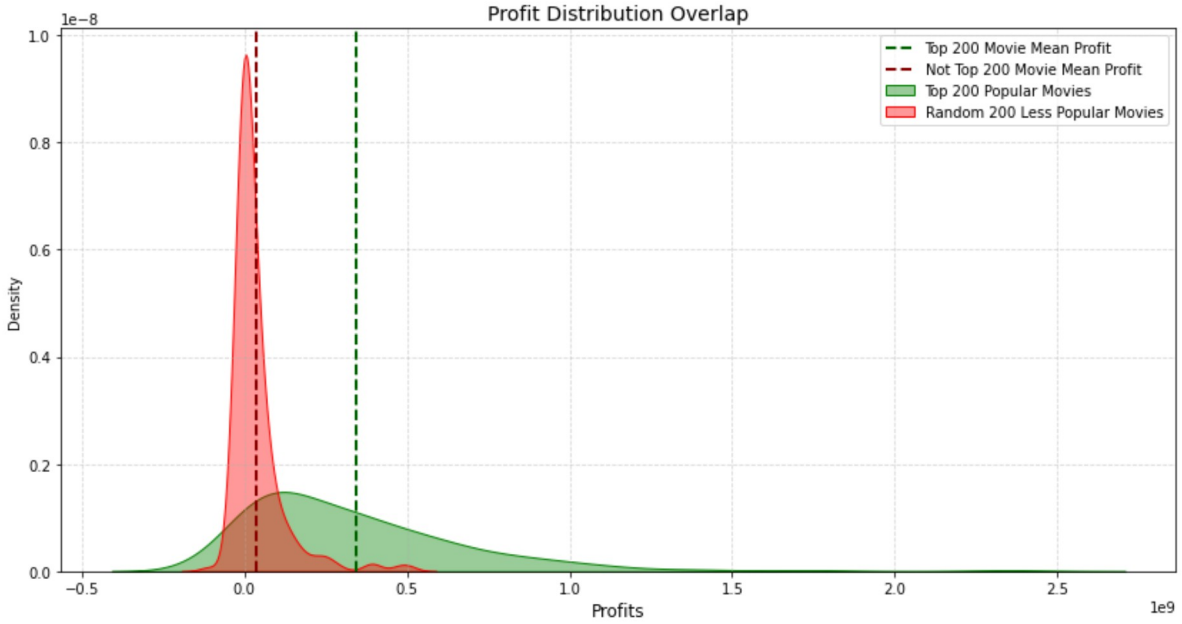
In [49]:
```
means = [top_200_popular_movies.mean(), random_sample_200_movies_not_top_200.mean(

plt.figure(figsize=(10, 6))
bars = plt.bar(['Top 200 Popular Movies', 'Random 200 Less Popular Movies'], means
plt.ylabel('Mean Profit', fontsize=12)
plt.title('Average Profit Difference by Popularity', fontsize=14)
plt.grid(axis='y', linestyle='--', alpha=0.6)

# Add value labels
for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2., height*0.95,
             f'${height/1e6:.1f}M', ha='center', color='white', fontweight='bold')

plt.show()
```



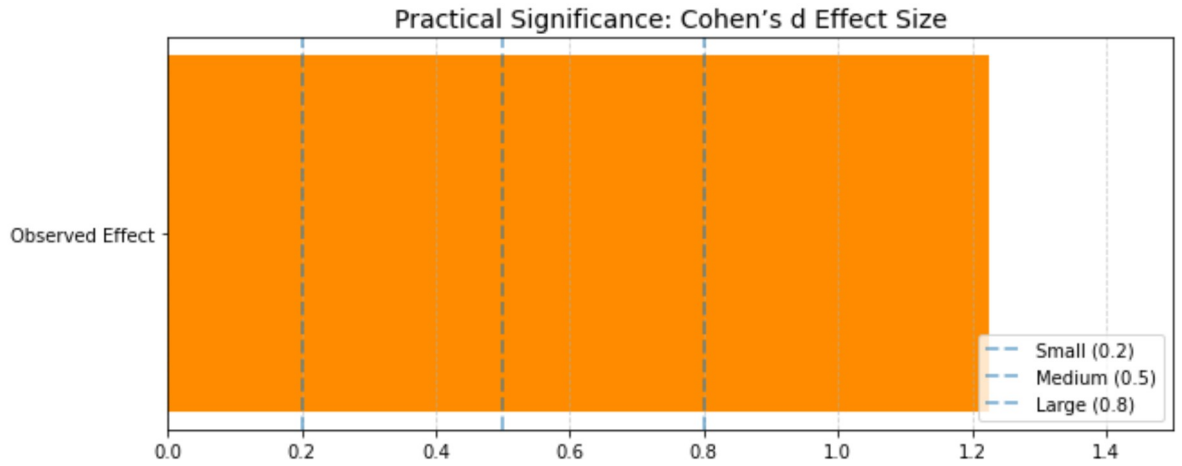
Insights from the graph above and the graph below:

- There is a significant difference in the profits between a very popular movie and a movie that isn't popular.
- The studio should heavily focus on promoting the movie through social media.
- Hiring a famous actor/director will help the movie make profits.

In [50]:
```
plt.figure(figsize=(14, 7))
sns.kdeplot(top_200_popular_movies, label='Top 200 Popular Movies', fill=True, co
sns.kdeplot(random_sample_200_movies_not_top_200, label='Random 200 Less Popular
            fill=True, color='red', alpha=0.4)
plt.axvline(top_200_popular_movies.mean(), color='darkgreen', linestyle='--', lin
plt.axvline(random_sample_200_movies_not_top_200.mean(), color='darkred', linesty
plt.xlabel('Profits', fontsize=12)
plt.title('Profit Distribution Overlap', fontsize=14)
plt.legend()
plt.grid(linestyle='--', alpha=0.5)
plt.show()
```



In [51]:
```
# Cohen's d reference lines
effect_ref = {'Small': 0.2, 'Medium': 0.5, 'Large': 0.8}

plt.figure(figsize=(10, 4))
plt.barh(['Observed Effect'], [effect_size], color='darkorange', height=0.5)
for label, val in effect_ref.items():
    plt.axvline(val, linestyle='--', alpha=0.5, label=f'{label} ({val})', linewidt
plt.title('Practical Significance: Cohen's d Effect Size', fontsize=14)
plt.xlim(0, 1.5)
plt.legend(loc='lower right')
plt.grid(axis='x', linestyle='--', alpha=0.6)
plt.show()
```



Insights:

- A cohen d effect size of 1.22 shows there is an extremely large effect from having a popular movie.
- The number one focus should be on making the movie popular through advertising and hiring a famous cast.

Hypothesis Test 3:

Null Hypothesis: Movies with higher ratings dont make more profits than lower rated movies.

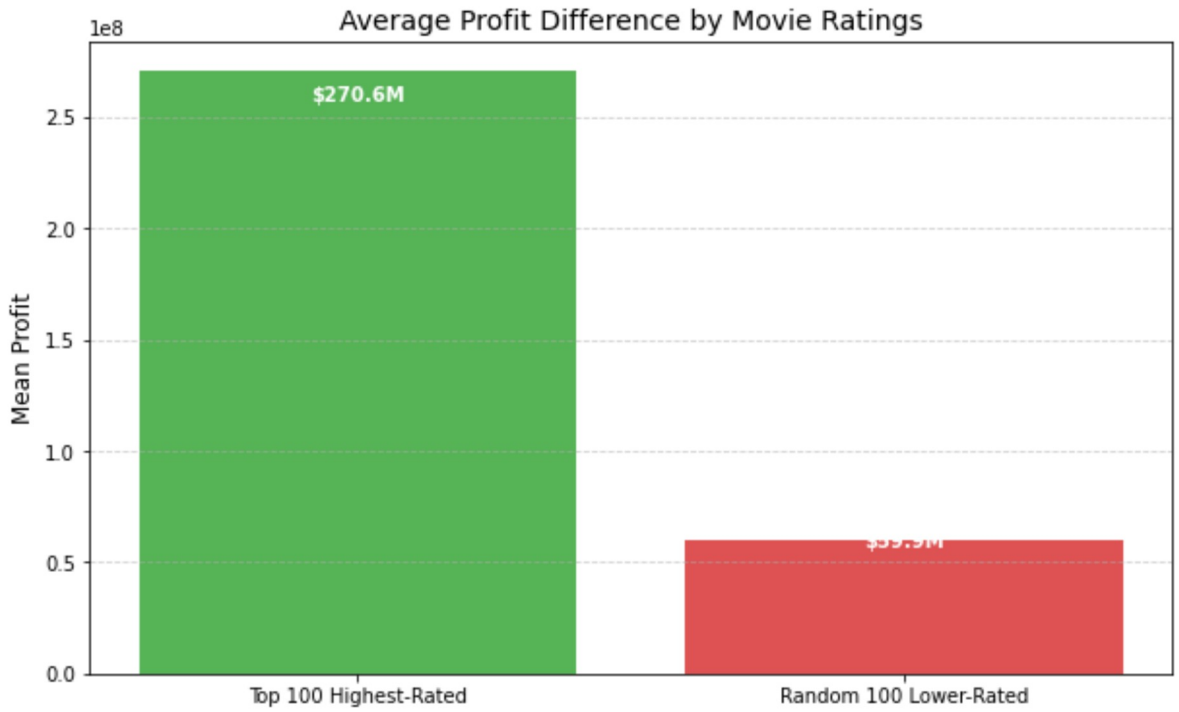
Alternative Hypothesis: Movies with higher ratings make more profits than lower rated

movies.

In [52]:
```
# all_combined[['title','runtime_minutes']].groupby('title').mean().sort_values('r

#Question 3: Movies with higher ratings make more profit?

#Null: Movies with higher ratings don't make higher profits?
#alternate: Movies with higher ratings make more profits?


highest_rated_movies = all_combined.groupby('title').mean().sort_values('averagera
top_100_rated_movies_profits = highest_rated_movies['profits'].head(100)

not_top_100_movies = highest_rated_movies.tail(894)
sample_not_top_100_movies_profits = not_top_100_movies['profits'].sample(n=100,ran

alpha = 0.05
t_statistic, p_value = stats.ttest_ind(top_100_rated_movies_profits,sample_not_top
print('t-stat: ',t_statistic)
print('p_value: ',p_value)

if p_value < alpha:
    print('Null Hypothesis is rejected. Results are statistically significant at a
    print('More highest-rated movies do not make the same profits as lower-rated r
else:
    print('Failed to reject null hypothesis. Results are not statistically signifi


effect_size = cohen_d(top_100_rated_movies_profits,sample_not_top_100_movies_profi
print('The effect size of the following was: ',effect_size)
```

```
t-stat:  5.198120400069214
p_value:  4.994850179731832e-07
Null Hypothesis is rejected. Results are statistically significant at alpha = 0.0
5.
More highest-rated movies do not make the same profits as lower-rated movies
The effect size of the following was:  0.7351252368626141
```

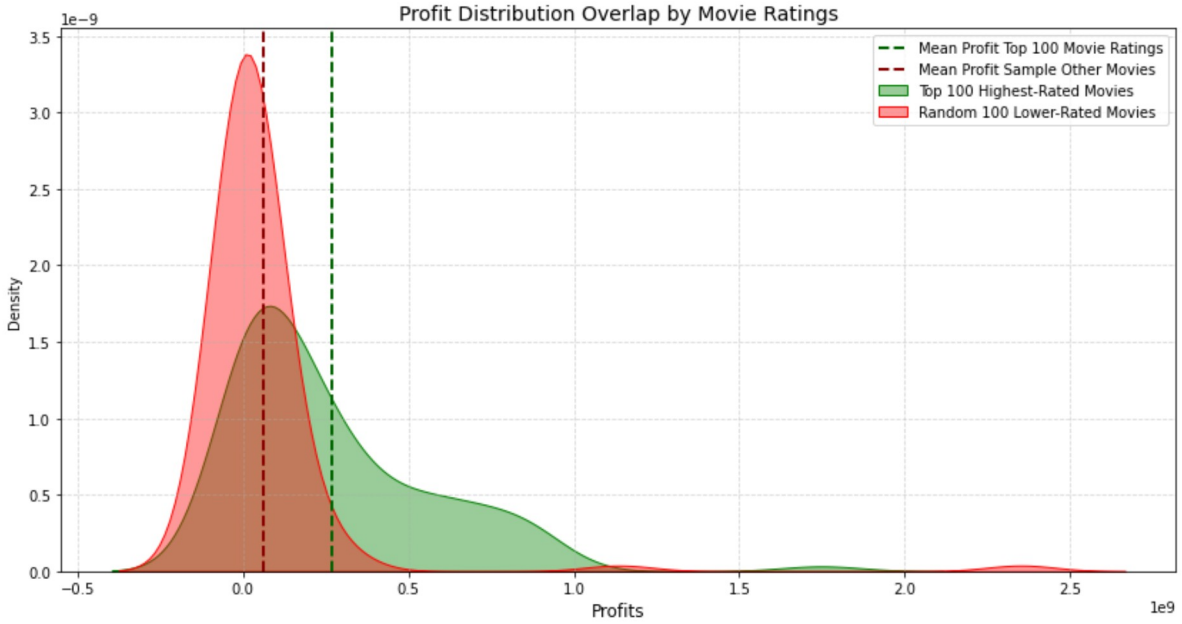
In [53]:
```
means = [top_100_rated_movies_profits.mean(), sample_not_top_100_movies_profits.me

plt.figure(figsize=(10, 6))
bars = plt.bar(['Top 100 Highest-Rated', 'Random 100 Lower-Rated'], means, capsize
plt.ylabel('Mean Profit', fontsize=12)
plt.title('Average Profit Difference by Movie Ratings', fontsize=14)
plt.grid(axis='y', linestyle='--', alpha=0.6)

# Add value labels
for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2., height*0.95,
             f'${height/1e6:.1f}M', ha='center', color='white', fontweight='bold')

plt.show()
```



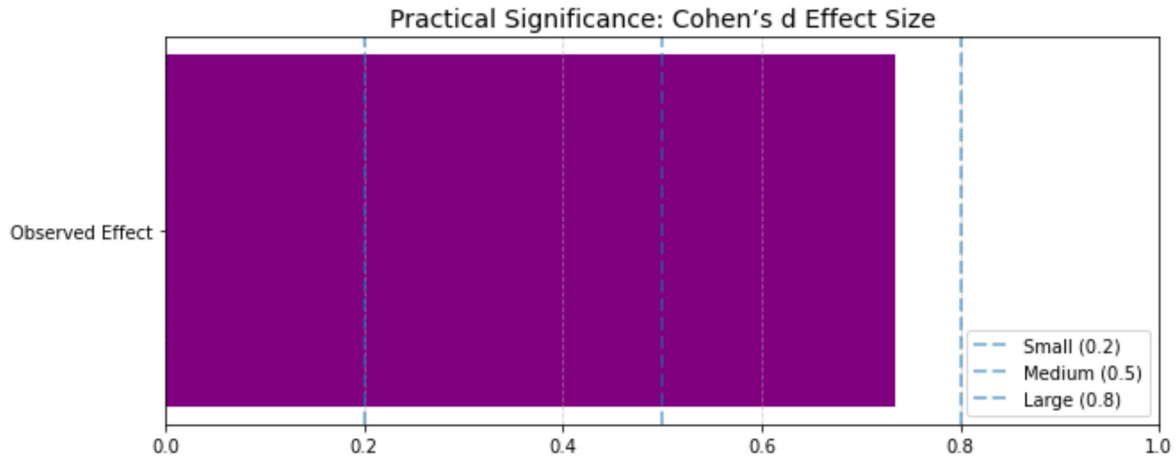
Insights:

- We can see that having a highly rated movie corresponds with a high mean profit.
- The studio should focus on hiring a good director to give movies the highest possible chance of being high quality

In [54]:
```
plt.figure(figsize=(14, 7))
sns.kdeplot(top_100_rated_movies_profits, label='Top 100 Highest-Rated Movies',
            fill=True, color='green', alpha=0.4)
sns.kdeplot(sample_not_top_100_movies_profits, label='Random 100 Lower-Rated Movie
            fill=True, color='red', alpha=0.4)
plt.axvline(top_100_rated_movies_profits.mean(), color='darkgreen', linestyle='--
plt.axvline(sample_not_top_100_movies_profits.mean(), color='darkred', linestyle=
plt.xlabel('Profits', fontsize=12)
plt.title('Profit Distribution Overlap by Movie Ratings', fontsize=14)
plt.legend()
plt.grid(linestyle='--', alpha=0.5)
plt.show()
```



In [55]:
```
# Cohen's d reference lines
effect_ref = {'Small': 0.2, 'Medium': 0.5, 'Large': 0.8}

plt.figure(figsize=(10, 4))
plt.barh(['Observed Effect'], [effect_size], color='purple', height=0.5)
for label, val in effect_ref.items():
    plt.axvline(val, linestyle='--', alpha=0.5, label=f'{label} ({val})', linewid
plt.title('Practical Significance: Cohen's d Effect Size', fontsize=14)
plt.xlim(0, 1.0)
plt.legend(loc='lower right')
plt.grid(axis='x', linestyle='--', alpha=0.6)
plt.show()
```



Insights

- A Cohen's d effect size of 0.73 means that high ratings have an almost large effect on the mean profits a movie generates.
- Good directors and good actors should be hired to give a movie a good chance of making more mean profits.

- The number one factor that will determine the success of the movie is the popularity of the film. Meaning the studio should allocate a lot of funds into promoting the movie. Actors with a high presence on social media should be hired since they can also play a large role in promotion.
- The second priority should be on ensuring the quality of the movie is as high as possible, since the highest-rated movies make substantially more profits. The studio should hire directors with the highest average ratings and actors who have won acting awards.
- The third priority should be hiring a highly rated director. While this has a smaller effect, it still contributes positively to profitability. Actor quality may have a stronger influence on the movie's success.
- From the EDA above, we noticed that movies released in the summer months (May, June, July) have higher mean profits. The studio should target these months for releases.
- The genres with the highest mean profits are Adventure, Animation, Sci-fi, Musical, Fantasy, Sport, and Action. The studio should focus on these genres.
- Animated movies have higher profits than non-animated ones. The studio should consider producing animated films.
- The studio should produce movies in English to maximize profits from the US and other English-speaking countries. The dataset lacked enough balance to draw conclusions about non-English films.

- Investigate how to make movies more popular, as this has the largest impact on profits. Begin by collecting data on movie advertising budgets and test whether there's a strong link between advertising and popularity using an A/B test.
- Study the impact of hiring famous actors on movie popularity. This could involve scraping actor social media stats and testing whether popularity correlates with increased profits.

- Emphasize hiring skilled actors and directors. Although the effect of highly rated directors was small (Cohen's d = 0.31), it still matters. Additional data can help assess the value of top-tier actors.
- Similarly, examine if hiring good screenwriters or adapting from high-quality source

material correlates with higher ratings and profits.

Recommendations Based on Hypothesis Testing

Based on statistical analysis and hypothesis testing, the following three data-driven recommendations are proposed to guide strategic decisions for our new movie studio:

---

1. Hire Top-Rated Directors

Insight: Movies directed by top-rated directors generate significantly higher profits compared to those directed by lesser-rated ones.

- *Statistical Significance*: Null hypothesis rejected ($p < 0.05$)
- *Effect Size (Cohen's d)*: 0.44 (moderate impact)

Recommendation:

> Prioritize hiring directors with a strong track record and high average ratings. Their experience and reputation positively influence box office performance.

---

2. Invest Heavily in Marketing and Popularity

Insight: Popular movies (measured by factors like search interest and social buzz—consistently) outperform less popular ones in terms of profit.

- *Statistical Significance*: Null hypothesis rejected ($p < 0.05$)
- *Effect Size (Cohen's d)*: 1.25 (very large impact)

Recommendation:

> Allocate a significant portion of the production budget toward building movie popularity. This includes digital marketing, influencer partnerships, viral campaigns, and casting popular actors to drive pre-release buzz and audience demand.

---

3. Focus on Producing High-Quality, Highly-Rated Films

Insight: Movies with higher viewer ratings tend to make more profits than lower-rated ones.

- *Statistical Significance*: Null hypothesis rejected ($p < 0.05$)

Recommendation:

> Emphasize strong storytelling, direction, acting, and production value to boost movie ratings. High-quality films are more likely to attract positive word-of-mouth and sustained revenue.

These recommendations provide a data-driven foundation to help the studio maximize its

chances of commercial success in the competitive film industry.

Modeling Section

Below we will import the libraries for modeling

In [85]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
```

My target variable is profits so I want to drop it from the training data. The remaining columns in X will be the independent variables that we will be using to train the model.

We will use multi-linear regression to make a model with multiple independent variables

We created a dataseries named y which is our target column.

In [ ]:
```
X = all_combined.drop(columns='profits')

y = all_combined['profits']
```

In [62]:
```
X.columns
```

Out[62]:
```
Index(['movie_id', 'ordering_1', 'person_id', 'category', 'ordering_4',
       'title', 'region', 'is_original_title', 'primary_title',
       'original_title_x', 'start_year', 'runtime_minutes', 'genres',
       'averagerating', 'numvotes', 'primary_name', 'primary_profession',
       'Unnamed: 0', 'genre_ids', 'id_x', 'original_language',
       'original_title_y', 'popularity', 'release_date_x', 'vote_average',
       'vote_count', 'id_y', 'release_date_y', 'movie', 'production_budget',
       'domestic_gross', 'worldwide_gross', 'genre_names', 'rating_bins',
       'release_month'],
      dtype='object')
```

In [ ]:
```
X = all_combined.drop_duplicates(subset='title')
```

We realised that the movie studio is only in charge of the run-time minutes, the genre of the film, popularity (to a certain degree by hiring popular actors/directors and spending money on advertising campaigns) and the production budget.

We will only keep these variables that can be controlled because we want to find out how well we can predict the profits made by using these features.

In [ ]:
```
X = X[['averagerating','runtime_minutes','genres',
       'popularity','production_budget']]

#we are encoding the categorical variable genres using the pd.get_dummies method
Dummy_columns = pd.get_dummies(X,columns=['genres'])
Dummy_columns.head()
```

Out[ ]:

	averagerating	runtime_minutes	popularity	production_budget	genres_Action	genres_Action,A
0	6.3	106.0	12.312	22000000.0	0	
1	6.3	106.0	12.312	22000000.0	0	

	averagerating	runtime_minutes	popularity	production_budget	genres_Action	genres_Action,A
2	6.3	106.0	12.312	22000000.0	0	
3	6.3	106.0	12.312	22000000.0	0	
4	6.3	106.0	12.312	22000000.0	0	

5 rows × 299 columns

In [79]:
```
Dummy_columns.shape
```

Out[79]:
```
(84687, 299)
```

In [80]:
```
y.shape
```

Out[80]:
```
(84687,)
```

We are splitting our randomly data into a training set and a testing set, we use a random_state to ensure the split remains the same when the code is run multiple times.

We want to reserve 20% of the data for testing.

In [81]:
```
X_train,X_test,y_train,y_test = train_test_split(Dummy_columns,y,test_size=0.2,ran
```

We then fit the model to our training data

In [82]:
```
model = LinearRegression()

model.fit(X_train,y_train)
```

Out[82]:
```
LinearRegression()
```

We are now using our testing data to look at our prediction accuracy

In [83]:
```
y_pred = model.predict(X_test)
```

We are using the R-squared, Mean Absolute error and Root Mean Squared Error as our gauges for model accuracy.

In [ ]:
```
print(f'Our R² Score: {r2_score(y_test, y_pred):.2f}')
print(f'MAE: ${mean_absolute_error(y_test, y_pred):,.0f}')
print(f'RMSE: ${np.sqrt(mean_squared_error(y_test, y_pred)):,.0f}')
```

```
R² Score: 0.64
MAE: $84,321,105
RMSE: $139,454,535
```

Intepretation

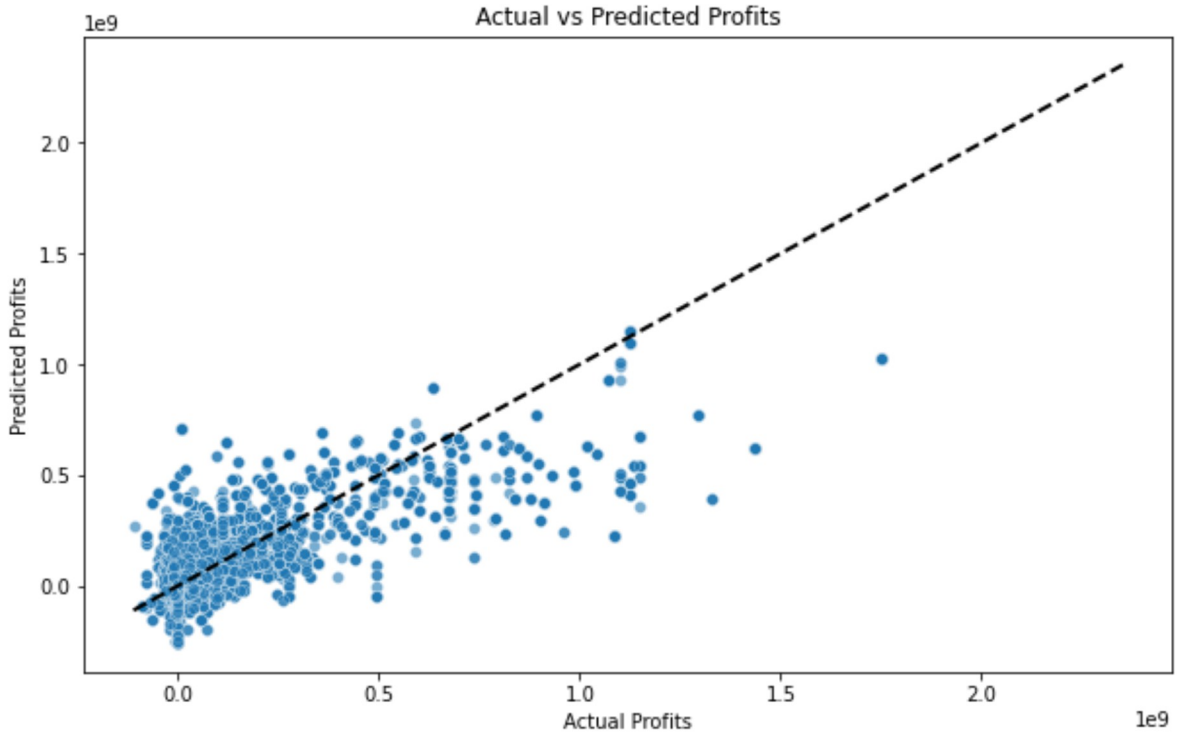
```
- The R-squared value of 0.64 means that 64% of the variation in
the profits has been accounted for by out model.
- The mean absolute error of $84,321,105 means that the model's
predictions were $84,321,105 away from the actual values.
- The root mean square error of $139,454,535 means that some of our
model prediction's were innaccurate, these are accounted for in the
root mean square error measurement. When we graph our model
predictions vs actual values we will see that some model line is
```

```
quite far from certain values.
```

We will now visualize our actual data vs the regression line that is fitting our data.

This will allow us to visually see the accuracy of the data

In [87]:
```
# Visualization 1: Actual vs Predicted
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_test, y=y_pred, alpha=0.6)
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=2)  # Perfect predictio
plt.xlabel('Actual Profits')
plt.ylabel('Predicted Profits')
plt.title('Actual vs Predicted Profits')
plt.show()
```



Insights from graph

```
- We can see that the regression line is reasonable accurate for
some values.
- Some values are quite far from the regression line. This might
mean that we need a higher order regression line because the line
that best approximates the data might be a curved line instead of a
straight one
```