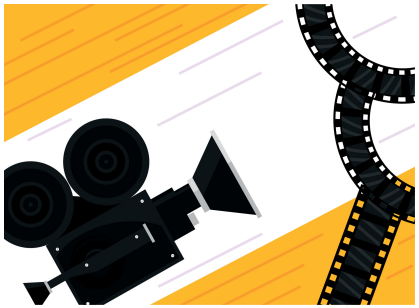
Film Industry Investment Analysis

Film Industry Vectors by Vecteezy (https://www.vecteezy.com/free-vector/film-industry)

Overview

This project analyzes which types of movies generate the most revenue at the box office. Descriptive analysis of movie data sets collected from Box Office Mojo, IMBD, Rotten Tomatoes, The Movie DB, and The Numbers, will highlight specific movie characteristics that increase ROI such as, genre selection, month of the film's release, and the length/runtime of the movie. This analysis can be used by the company to make decisions on these specific movie criteria and help the company's new movie studio operate with the lowest risk in their new business endeavor.

Business Problem



Film Industry Vectors by Vecteezy (https://www.vecteezy.com/free-vector/film-industry)

Analyzing several movie datasets can provide valuable insights that can lead to concrete business recommendations in various aspects of the film industry. When considering characteristics of a successful movie, the movie data was analyzed to identify movies with the greatest return on investment. Based on this analysis, which is explained below, are the following three recommendations:

1. Action and/or Adventure genre movies gross higher than other genres**
2. Most successful release months are May, June, November, and December**
3. The most profitable movies are 85-125 minutes in length**

It's important to note that any business recommendations derived from data analysis should be accompanied by careful consideration of the specific context, legal and regulatory requirements, and limitations of the dataset. Additionally, these recommendations should be subject to ongoing evaluation and refinement based on updated data and emerging industry practices.

Data Understanding

Some media databases provide public data that includes various data about historical and current films both domestically and internationally. Each database has a variety of variables some of which may match other databases. The quantity of data provided was excellent, however, the quality of these various datasets were weak.

Some files are compressed CSV (comma-separated values) or TSV (tab-separated values) files that can be opened using spreadsheet software or pd.read_csv, while the data from IMDb is located in a SQLite database.

I have been provided a diagram for the IMDb data and will have to explore the other data to see how they connect to each other and the IMDb data.

Overview of dataset:

1. **Independent Sources:**

- Internet Movie Database (IMDb)
- The Movie Database (TMDb)
- Box Office Mojo
- The Numbers

2. **About:** Selected film industry data reported between 1915-2020. Due to the differences in the independent databases, when choosing variables several considerations were made:

    - Data Relevance
    - Data Quantity
    - Data Quality

3. **Decision-based Variables used:**

    - Profit/Profit Margin
    - Average Domestic Gross
    - Movie Runtime
    - Movie Release Date
    - Genre of Movie

4. **Missing Values:** In the case of data quality, missing data were handled on a case by case basis. Some considerations made were the relative importance of the variable in a particular analysis, how much data was needed, and finally, the possibility of filling data with a central measure of tendency and the statistical implications.

Data Exploration

In [1]:

```
# Import the necessary packages.

# Native Python Packages
import os
from collections import Counter
from glob import glob
import functools

# Third Party Packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import FuncFormatter
import plotly.graph_objects as go
import plotly.express as px
import seaborn as sns
import locale
import inflect
import sqlite3
import cpi
%matplotlib inline

```

→ There are a number of .csv and .tsv files provided in the zipped folder, glob will be used to read them in by matching path names.

In [2]:

```
# Create a list of all csv files using glob
data_folder = './Data'
csv_files = glob(os.path.join(data_folder, '*.csv.gz'))
for i, file in enumerate(csv_files, 1):
    print(i, file, sep=': ')
```

```
1: ./Data/tmdb.movies.csv.gz
2: ./Data/tn.movie_budgets.csv.gz
3: ./Data/bom.movie_gross.csv.gz
```

→ Using the Name of the file as the key and and the Pandas Dataframe as the value, dictionaries will be used to store dataframes. All data can be previewed by iterrating over the dictionary this way.

In [3]:

```
# Create a dictionary of DataFrames
csv_dict = {}
for file in csv_files:
    filename = os.path.splitext(os.path.basename(file))[0].replace('.csv', '').replace('.', '_')  # Cle
    file_df = pd.read_csv(file)  # Create DataFrame
    csv_dict[filename] = file_df  # Insert DataFrame into dictionary
```

In [4]:

```
# Preview the data
for name, df in csv_dict.items():
    print(name)
    print('Total number of results:', len(df))
    display(df.head()) # Using display instead of print leads to neater formatting in Jupyter Notebook
```

```
tmdb_movies
Total number of results: 26517
```

	Unnamed: 0	genre_ids	id	original_language	original_title	popularity	release_date	title	vote_average	vote_count
0	0	[12, 14, 10751]	12444	en	Harry Potter and the Deathly Hallows: Part 1	33.533	2010-11-19	Harry Potter and the Deathly Hallows: Part 1	7.7	10788
1	1	[14, 12, 16, 10751]	10191	en	How to Train Your Dragon	28.734	2010-03-26	How to Train Your Dragon	7.7	7610
2	2	[12, 28, 878]	10138	en	Iron Man 2	28.515	2010-05-07	Iron Man 2	6.8	12368
3	3	[16, 35, 10751]	862	en	Toy Story	28.005	1995-11-22	Toy Story	7.9	10174
4	4	[28, 878, 12]	27205	en	Inception	27.920	2010-07-16	Inception	8.3	22186

```
tn_movie_budgets
Total number of results: 5782
```

	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

```
bom_movie_gross
Total number of results: 3387
```

	title	studio	domestic_gross	foreign_gross	year
0	Toy Story 3	BV	415000000.0	652000000	2010
1	Alice in Wonderland (2010)	BV	334200000.0	691300000	2010
2	Harry Potter and the Deathly Hallows Part 1	WB	296000000.0	664300000	2010
3	Inception	WB	292600000.0	535700000	2010
4	Shrek Forever After	P/DW	238700000.0	513900000	2010

→ Again, use the same process for the .tsv files.

In [5]:
```
# Create a list of all tsv files using glob
data_folder = './Data'
tsv_files = glob(os.path.join(data_folder, '*.tsv.gz'))
```

In [6]:
```
# Create a dictionary of DataFrames
tsv_dict = {}
for file in tsv_files:
    filename = os.path.splitext(os.path.basename(file))[0].replace('.tsv', '').replace('.', '_')  # Cle
    file_df = pd.read_table(file, delimiter = '\t', encoding = 'latin-1') # Create DataFrame
    tsv_dict[filename] = file_df  # Insert DataFrame into dictionary
```

In [7]:

```
# Preview the data
for name, df in tsv_dict.items():
    print(name)
    print('Total number of results:', len(df))
    display(df.head()) # Using display instead of print leads to neater formatting in Jupyter Notebook
```

```
rt_reviews
Total number of results: 54432
```

	id	review	rating	fresh	critic	top_critic	publisher	date
0	3	A distinctly gallows take on contemporary fina...	3/5	fresh	PJ Nabarro	0	Patrick Nabarro	November 10, 2018
1	3	It's an allegory in search of a meaning that n...	NaN	rotten	Annalee Newitz	0	io9.com	May 23, 2018
2	3	... life lived in a bubble in financial dealin...	NaN	fresh	Sean Axmaker	0	Stream on Demand	January 4, 2018
3	3	Continuing along a line introduced in last yea...	NaN	fresh	Daniel Kasman	0	MUBI	November 16, 2017
4	3	... a perverse twist on neorealism...	NaN	fresh	NaN	0	Cinema Scope	October 12, 2017

```
rt_movie_info
Total number of results: 1560
```

	id	synopsis	rating	genre	director	writer	theater_date	dvd_date	currency	box_office	runtime	stu
0	1	This gritty, fast-paced, and innovative police...	R	Action and Adventure|Classics|Drama	William Friedkin	Ernest Tidyman	Oct 9, 1971	Sep 25, 2001	NaN	NaN	104 minutes	
1	3	New York City, not-too-distant-future: Eric Pa...	R	Drama|Science Fiction and Fantasy	David Cronenberg	David Cronenberg|Don DeLillo	Aug 17, 2012	Jan 1, 2013	$	600,000	108 minutes	Entertainn
2	5	Illeana Douglas delivers a superb performance ...	R	Drama|Musical and Performing Arts	Allison Anders	Allison Anders	Sep 13, 1996	Apr 18, 2000	NaN	NaN	116 minutes	
3	6	Michael Douglas runs afoul of a treacherous su...	R	Drama|Mystery and Suspense	Barry Levinson	Paul Attanasio|Michael Crichton	Dec 9, 1994	Aug 27, 1997	NaN	NaN	128 minutes	
4	7	NaN	NR	Drama|Romance	Rodney Bennett	Giles Cooper	NaN	NaN	NaN	NaN	200 minutes	

→ There are a number of DataFrames. This analysis will focus on the DataFrames that likely lead to recommendations.

The first DataFrame that stands out is tn_movie_budgets. This table shows the production budget, domestic and worldwide gross. This financial information will be valuable in exploring the meaning of profit. It can also be used as a standard measure of success based on which films performed best in the box office.

Additional tables that will be explored include: bom_movie_gross (domestic and worldwide profit), and tmdb_movies (genre). movie_basics (genre, runtime) and will also be incorporating data from an SQL database.

Data Preparation

Data Cleaning

The request from the investors is to recommend movie characteristics that maximize their return on investment. Addressing missing values across these databases requires thoughtful consideration, as relevant data is presented in different databases and so there must be clean and standardized variables in order to connect the data across databases.

In [8]:
```
# Clean up 'bom_movie_gross'
movie_gross = csv_dict['bom_movie_gross']
# Get summary
movie_gross.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3387 entries, 0 to 3386
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   title           3387 non-null   object 
 1   studio          3382 non-null   object 
 2   domestic_gross  3359 non-null   float64
 3   foreign_gross   2037 non-null   object 
 4   year            3387 non-null   int64  
dtypes: float64(1), int64(1), object(3)
memory usage: 132.4+ KB
```

→ movie_gross will require that we convert the money related data types to intergers.

In [9]:

```
  '''
    calculates the ….. what (not how) - minimal implentation details, if  needed.
    -
    Input:
    df : Pandas dataframe, include the relavent column names
    -
    Output:
    summary : Pandas dataframe, details about new dataframe

    ###in line comments are what and how####(the one liners above the code)
    '''

# Define function to clean numbers
def numclean(df, col):
    '''
    Formats the float64 and the object values into integers. All
    NaN values within a specified column will be dropped from the data frame.
    -
    Input:
    df : Pandas dataframe, include the relavent column names
    col: A string that represents a column name
    -
    Output:
    df : Pandas dataframe, with integer and without Nan values
    '''

    # Convert to numeric, handling NaN values
    df[col] = pd.to_numeric(df[col], errors='coerce', downcast='integer')

    # Drop rows with NaN values in the specified column
    df = df.dropna(subset=[col])

    return df

```

In [10]:
```
# Run numclean function on the two columns
numclean(movie_gross,'foreign_gross')
numclean(movie_gross,'domestic_gross')

# Display dataframe with both columns cleaned
display(movie_gross.head())
```

	title	studio	domestic_gross	foreign_gross	year
0	Toy Story 3	BV	415000000.0	652000000.0	2010
1	Alice in Wonderland (2010)	BV	334200000.0	691300000.0	2010
2	Harry Potter and the Deathly Hallows Part 1	WB	296000000.0	664300000.0	2010
3	Inception	WB	292600000.0	535700000.0	2010
4	Shrek Forever After	P/DW	238700000.0	513900000.0	2010

In [11]:
```
# Overview the data for an potential anomalies
movie_gross.describe()
```

Out[11]:

	domestic_gross	foreign_gross	year
count	3.359000e+03	2.032000e+03	3387.000000
mean	2.874585e+07	7.505704e+07	2013.958075
std	6.698250e+07	1.375294e+08	2.478141
min	1.000000e+02	6.000000e+02	2010.000000
25%	1.200000e+05	3.775000e+06	2012.000000
50%	1.400000e+06	1.890000e+07	2014.000000
75%	2.790000e+07	7.505000e+07	2016.000000
max	9.367000e+08	9.605000e+08	2018.000000

→ There aren't any entries here that seem unusual. It's worth noting that the movies in this data base range in year from 2010-2018.

tn_movie_budgets will require that the:

1. '$' symbol

2. the comma seperator be removed
3. the number converted to an integer

It makes sense in this case to define a function and then apply it to the necessary columns to increase code efficiency.

In [12]:
```
#Clean up 'tn_movie_gross'
ww_gross = csv_dict['tn_movie_budgets']
display(ww_gross.head())
```

	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

In [13]:
```
# Convert release_date to a datetime object
ww_gross['release_date'] = pd.to_datetime(ww_gross['release_date'])
```

In [14]:
```
# Create year column
ww_gross['year'] = pd.DatetimeIndex(ww_gross['release_date']).year
```

In [15]:
```
# Define function to clean numbers
def numclean2(df,col):
    '''
    Formats the float64 and the object values into integers. All
    NaN values within a specified column will be dropped from the data frame.
    Removes $ symbol, removes ',' seperator and changes to int64 type
    -
    Input:
    df : Pandas dataframe, include the relavent column names
    col: A string that represents a column name
    -
    Output:
    df : Pandas dataframe, with integer and without Nan values
    '''

    if df[col].dtype != 'int64':
        df[col]=df[col].str.replace('$','').str.replace(',','').astype('int64')

    return df
```

In [16]:
```
# Run numclean2 function on the two columns
numclean2(ww_gross,'worldwide_gross')
numclean2(ww_gross,'domestic_gross')
numclean2(ww_gross,'production_budget')

# Display dataframe with all three columns cleaned
display(ww_gross.head())
```

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross	year
0	1	2009-12-18	Avatar	425000000	760507625	2776345279	2009
1	2	2011-05-20	Pirates of the Caribbean: On Stranger Tides	410600000	241063875	1045663875	2011
2	3	2019-06-07	Dark Phoenix	350000000	42762350	149762350	2019
3	4	2015-05-01	Avengers: Age of Ultron	330600000	459005868	1403013963	2015
4	5	2017-12-15	Star Wars Ep. VIII: The Last Jedi	317000000	620181382	1316721747	2017

In [17]:
```
# Overview the data for an anomalies
ww_gross.describe()
```

Out[17]:

	id	production_budget	domestic_gross	worldwide_gross	year
count	5782.000000	5.782000e+03	5.782000e+03	5.782000e+03	5782.000000
mean	50.372363	3.158776e+07	4.187333e+07	9.148746e+07	2003.967139
std	28.821076	4.181208e+07	6.824060e+07	1.747200e+08	12.724386
min	1.000000	1.100000e+03	0.000000e+00	0.000000e+00	1915.000000
25%	25.000000	5.000000e+06	1.429534e+06	4.125415e+06	2000.000000
50%	50.000000	1.700000e+07	1.722594e+07	2.798445e+07	2007.000000
75%	75.000000	4.000000e+07	5.234866e+07	9.764584e+07	2012.000000
max	100.000000	4.250000e+08	9.366622e+08	2.776345e+09	2020.000000

→ This data table shows entries where the movie gross is zero and that's unusual. Some movies may have staggered release dates, may not be released in the USA, or simply the data was never entered and so there may not be values avaiable for every movie title.

I will join these two data sets and then proceed to drop any data that still contains a zero value for the gross revenue.

Additionally, the range of years represented in this data table span from 1915-2020.

Profit and Profit Margin

The value of this venture relies on making a profit on the investment. Due to multiple revenue streams, it's challenging to understand whether or not a box office movie is ultimately profitable - and to whom. There is an abundance of information that takes careful consideration, in context, to interpret meaningfully.

For both a precise and accurate analysis, a concrete understanding of which variables the company would be owning would be needed. Due to the lack of detailed financial data available publically around movie profitability, this project will share a simplified analysis model based on publically available data.

In this analysis:

domestic profit is defined as follows: **profit = domestic gross revenue - production budget**

profit margin is defined to be: **profit margin = (domestic gross revenue - production budget) / domestic gross revenue *100**

*all values in this analysis have been adjusted for inflation for accurate comparison

Aggragating money data

In [18]:
```
# Create variables for dictionaries
rt_movie_info = tsv_dict['rt_movie_info']
tmdb_movies = csv_dict['tmdb_movies']
```

In [19]:
```
# Print columns for tmdb_movies data frame
print('Columns in tmdb_movies:')
print(tmdb_movies.columns)

# Print columns for tn_movie_budgets data frame
print('Columns in ww_gross:')
print(ww_gross.columns)

# Print columns for bom_movie_gross data frame
print('Columns in movie_gross:')
print(movie_gross.columns)
```

```
Columns in tmdb_movies:
Index(['Unnamed: 0', 'genre_ids', 'id', 'original_language', 'original_title',
       'popularity', 'release_date', 'title', 'vote_average', 'vote_count'],
      dtype='object')
Columns in ww_gross:
Index(['id', 'release_date', 'movie', 'production_budget', 'domestic_gross',
       'worldwide_gross', 'year'],
      dtype='object')
Columns in movie_gross:
Index(['title', 'studio', 'domestic_gross', 'foreign_gross', 'year'], dtype='object')
```

In [20]:
```
# Rename columns in ww_gross DataFrame
ww_gross = ww_gross.rename(columns={'movie': 'title'})
```

In [21]:
```
# How many entries of domestic gross are zero or null?
len(movie_gross[(movie_gross['domestic_gross'] == 0) | (movie_gross['domestic_gross'].isnull())])
```

Out[21]: 28

In [22]:
```
# How many entries of domestic gross are zero or null?
len(ww_gross[(ww_gross['domestic_gross'] == 0) | (ww_gross['domestic_gross'].isnull())])
```

Out[22]: 548

In [23]:
```
# How many entries of domestic gross are zero or null?
len(ww_gross[(ww_gross['worldwide_gross'] == 0) | (ww_gross['worldwide_gross'].isnull())])
```

Out[23]: 367

In [24]:
```
# How many entries of foreign gross are zero or null?
len(movie_gross[(movie_gross['foreign_gross'] == 0) | (movie_gross['foreign_gross'].isnull())])
```

Out[24]: 1355

In [25]:
```
# How many entries of production budget are zero or null?
len(ww_gross[(ww_gross['production_budget'] == 0) | (ww_gross['production_budget'].isnull())])
```

Out[25]: 0

→ Given the number of null or zero values between the movie_gross and ww_gross data sets, it turns out that the total number of usable data points reporting a worldwide gross is 5415 and the total number of usable data points reporting a domestic gross is 8565.

The majority of the foreign gross reports are zero or null values so this data will not be used.

Meta data could be used to verify whether all of the worldwide gross reports are in USD.

It makes sense to maximize the data points reporting revenue in order to advise stakeholders on the best way to increase ROI. All subsequent calculations will be based on available adjusted domestic revenue figures.

In [26]:
```
# Convert all movie title strings to lowercase
movie_gross['title'] = movie_gross['title'].str.lower()
ww_gross['title'] = ww_gross['title'].str.lower()
```

In [27]:
```
# Join the DataFrames using an 'outer' join to include all rows from both DataFrames
money_df = pd.merge(movie_gross, ww_gross, on=['title', 'year'], how='outer')
print('Total number of results:', len(money_df))
display(money_df.head())
```

```
Total number of results: 7914
```

	title	studio	domestic_gross_x	foreign_gross	year	id	release_date	production_budget	domestic_gross_y	worldwide_gross
0	toy story 3	BV	415000000.0	652000000.0	2010	47.0	2010-06-18	200000000.0	415004880.0	1.068880e+09
1	alice in wonderland (2010)	BV	334200000.0	691300000.0	2010	NaN	NaT	NaN	NaN	NaN
2	harry potter and the deathly hallows part 1	WB	296000000.0	664300000.0	2010	NaN	NaT	NaN	NaN	NaN
3	inception	WB	292600000.0	535700000.0	2010	38.0	2010-07-16	160000000.0	292576195.0	8.355246e+08
4	shrek forever after	P/DW	238700000.0	513900000.0	2010	27.0	2010-05-21	165000000.0	238736787.0	7.562447e+08

In [28]:
```
# Convert release_date to a datetime object
money_df['release_date'] = pd.to_datetime(money_df['release_date'])
```

In [29]:

```
# Create columns and average values from two different data sets

# Calculate the average for domestic gross
money_df['avg_domestic_gross'] = money_df[['domestic_gross_x', 'domestic_gross_y']].replace(0, np.nan).

# Drop unnecessary columns
money_df.drop(['domestic_gross_x', 'domestic_gross_y', 'studio', 'id'], axis=1, inplace=True)

# Drop rows with NaN in 'avg_domestic_gross'
money_df.dropna(subset=['avg_domestic_gross'], inplace=True)

print('Total number of results:', len(money_df))
display(money_df.head())
```

```
Total number of results: 7344
```

	title	foreign_gross	year	release_date	production_budget	worldwide_gross	avg_domestic_gross
0	toy story 3	652000000.0	2010	2010-06-18	200000000.0	1.068880e+09	415002440.0
1	alice in wonderland (2010)	691300000.0	2010	NaT	NaN	NaN	334200000.0
2	harry potter and the deathly hallows part 1	664300000.0	2010	NaT	NaN	NaN	296000000.0
3	inception	535700000.0	2010	2010-07-16	160000000.0	8.355246e+08	292588097.5
4	shrek forever after	513900000.0	2010	2010-05-21	165000000.0	7.562447e+08	238718393.5

In [30]:

```
# Accessing sorted duplicated movie titles
print('Total number of results:', len(money_df[money_df.duplicated(['title'], keep=False)]))
money_df[money_df.duplicated(['title'], keep=False)].sort_values(by=['title']).head()

```

```
Total number of results: 191
```

Out[30]:

	title	foreign_gross	year	release_date	production_budget	worldwide_gross	avg_domestic_gross
7752	20,000 leagues under the sea	NaN	1916	1916-12-24	200000.0	8000000.0	8000000.0
6574	20,000 leagues under the sea	NaN	1954	1954-12-23	5000000.0	28200000.0	28200000.0
4315	a monster calls	NaN	2017	2017-01-06	43000000.0	46414530.0	3740823.0
2435	a monster calls	43600000.0	2016	NaT	NaN	NaN	3700000.0
4531	a nightmare on elm street	NaN	2010	2010-04-30	35000000.0	117729621.0	63075011.0

In [31]:

```
# Sort the DataFrame by 'title' and 'release_date'
money_df.sort_values(by=['title', 'release_date'], inplace=True)

# Find duplicates within one year of each other for each title
mask = money_df.duplicated(subset=['title'], keep=False)
duplicates = money_df[mask]

# Create a mask to identify duplicates with a difference of 1 in years
mask_within_one_year = (duplicates['year'].diff().abs() == 1)

# Filter out duplicates where the difference between years is 1
duplicates_within_one_year = duplicates[mask_within_one_year]

# Keep the rows with the 'release_date' and drop the other duplicates
to_drop = duplicates_within_one_year[duplicates_within_one_year['release_date'].isnull()].index
money_df.drop(to_drop, inplace=True)

print('Total number of results:', len(money_df))
display(money_df.head())
```

```
Total number of results: 7333
```

	title	foreign_gross	year	release_date	production_budget	worldwide_gross	avg_domestic_gross
2093	'71	355000.0	2015	NaT	NaN	NaN	1300000.0
6309	(500) days of summer	NaN	2009	2009-07-17	7500000.0	34439060.0	32425665.0
1830	1,000 times good night	NaN	2014	NaT	NaN	NaN	53900.0
2395	10 cloverfield lane	38100000.0	2016	2016-03-11	5000000.0	108286422.0	72091499.5
5913	10 days in a madhouse	NaN	2015	2015-11-11	12000000.0	14616.0	14616.0

In [32]:
```
1 money_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7333 entries, 2093 to 5331
Data columns (total 7 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   title               7333 non-null   object
 1   foreign_gross       1994 non-null   float64
 2   year                7333 non-null   int64
 3   release_date        5238 non-null   datetime64[ns]
 4   production_budget   5238 non-null   float64
 5   worldwide_gross     5238 non-null   float64
 6   avg_domestic_gross  7333 non-null   float64
dtypes: datetime64[ns](1), float64(4), int64(1), object(1)
memory usage: 458.3+ KB
```

In [33]:

```
# Adjust $ columns for inflation using CPI
columns_to_adjust = ['worldwide_gross', 'avg_domestic_gross', 'foreign_gross', 'production_budget']

for column in columns_to_adjust:
    print (column)
    money_df[f'cpi_{column}'] = money_df.apply(lambda row: cpi.inflate(row[column], row['year']), axis=

print('Total number of results:', len(money_df))
display(money_df.head())
```

```
worldwide_gross
avg_domestic_gross
foreign_gross
production_budget
Total number of results: 7333
```

	title	foreign_gross	year	release_date	production_budget	worldwide_gross	avg_domestic_gross	cpi_worldwide_gross	cpi_avg_dome
2093	'71	355000.0	2015	NaT	NaN	NaN	1300000.0	NaN	1.6
6309	(500) days of summer	NaN	2009	2009-07-17	7500000.0	34439060.0	32425665.0	4.891301e+07	4.6
1830	1,000 times good night	NaN	2014	NaT	NaN	NaN	53900.0	NaN	6.9
2395	10 cloverfield lane	38100000.0	2016	2016-03-11	5000000.0	108286422.0	72091499.5	1.374755e+08	9.1
5913	10 days in a madhouse	NaN	2015	2015-11-11	12000000.0	14616.0	14616.0	1.878989e+04	1.8

In [34]:

```
# Use plain formatting to remove scientific notation
money_df['cpi_worldwide_gross'] = pd.to_numeric(money_df['cpi_worldwide_gross'], errors='coerce')
money_df['cpi_production_budget'] = pd.to_numeric(money_df['cpi_production_budget'], errors='coerce')
money_df['cpi_foreign_gross'] = pd.to_numeric(money_df['cpi_foreign_gross'], errors='coerce')
money_df['cpi_avg_domestic_gross'] = pd.to_numeric(money_df['cpi_avg_domestic_gross'], errors='coerce')

# Create domestic_profit column
money_df['domestic_profit'] = money_df.apply(lambda x: (x['cpi_avg_domestic_gross']) - (x['cpi_producti

# Get summary statistics for profit
money_df['domestic_profit'].describe().apply(lambda x: format(x, 'f'))
```

Out[34]:

```
count          5238.000000
mean       31169023.887285
std       144654769.502849
min      -366177833.700231
25%       -14361218.380512
50%          397569.143800
75%        39215148.715017
max      4269784084.168346
Name: domestic_profit, dtype: object
```

Profit and loss amounts vary greatly ranging from approximately (in US Dollars) 360 million in losses to 4 billion in profits. The median indicates a movie profit of approximately 400 thousand.

In [35]:

```
# Calculate the domestic profit margin for each movie and make a new column
money_df['domestic_profit_margin'] = (money_df['domestic_profit'] / money_df['cpi_avg_domestic_gross'])

print('Total number of results:', len(money_df))
display(money_df.head())
```

```
Total number of results: 7333
```

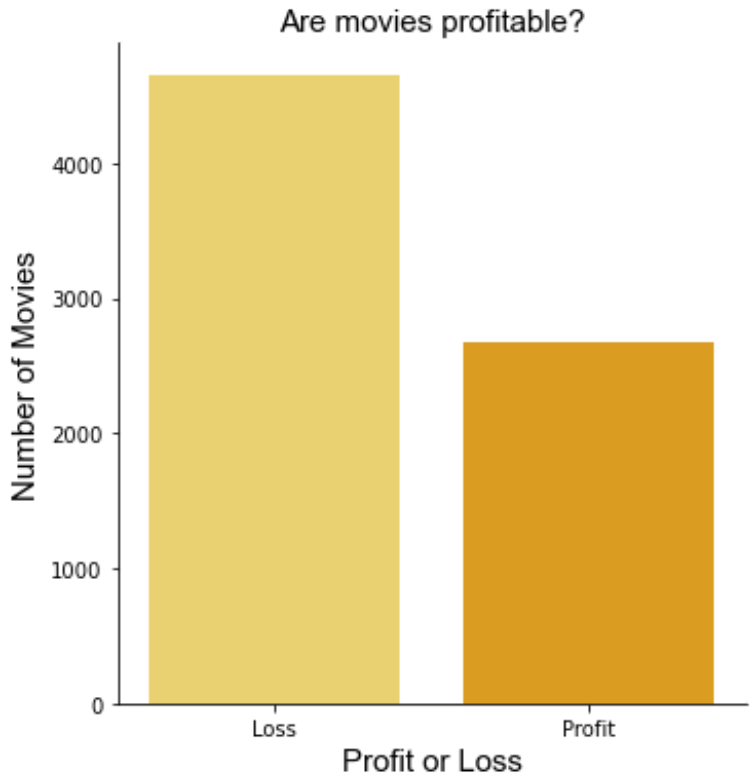
	title	foreign_gross	year	release_date	production_budget	worldwide_gross	avg_domestic_gross	cpi_worldwide_gross	cpi_avg_dome
2093	'71	355000.0	2015	NaT	NaN	NaN	1300000.0	NaN	1.6
6309	(500) days of summer	NaN	2009	2009-07-17	7500000.0	34439060.0	32425665.0	4.891301e+07	4.6
1830	1,000 times good night	NaN	2014	NaT	NaN	NaN	53900.0	NaN	6.9
2395	10 cloverfield lane	38100000.0	2016	2016-03-11	5000000.0	108286422.0	72091499.5	1.374755e+08	9.1
5913	10 days in a madhouse	NaN	2015	2015-11-11	12000000.0	14616.0	14616.0	1.878989e+04	1.8

→ Some of the percentages don't add to 100% so the difference may be attributed to documenting movie gross profits at different times during the movie's runtime. If the value is negative, it may also mean a profit loss.

The number of profitable movies can be visualized using domestic_profit/=profit margine, as defined above.

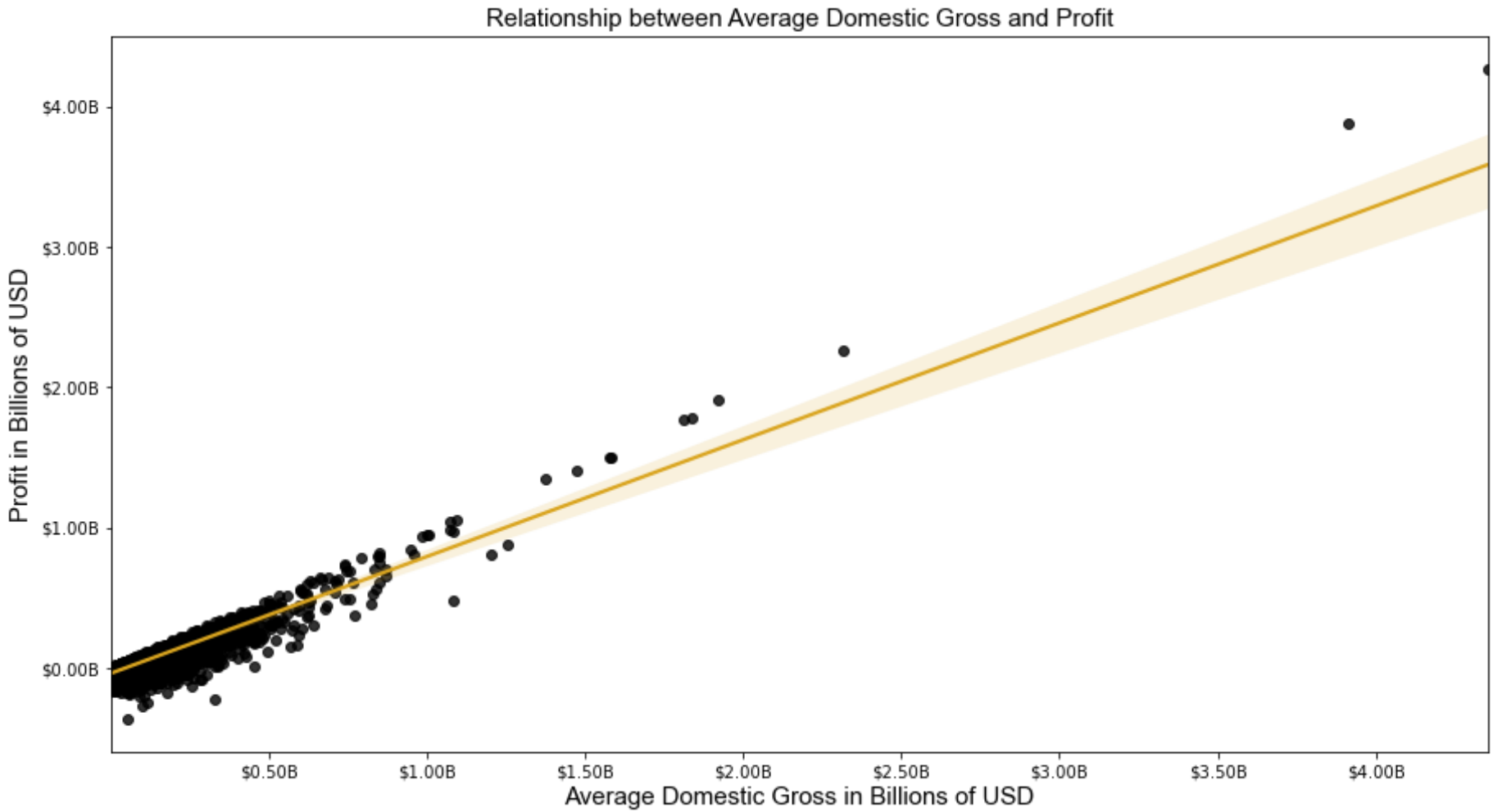
In [36]:

```
# Domestic_profit column has both positive and negative values
money_df['Profit or Loss'] = money_df['domestic_profit'].apply(lambda x: 'Profit' if x >= 0 else 'Loss

# Create plot
sns.catplot(x='Profit or Loss', kind='count', palette=['#FDDC5C','#F9A602'], data=money_df)
plt.xlabel('Profit or Loss', fontfamily = 'Arial', fontsize = 15)
plt.ylabel('Number of Movies', fontfamily = 'Arial', fontsize = 15)
plt.title('Are movies profitable?', fontfamily = 'Arial', fontsize = 15)
plt.show()
```



In [37]:

```python
# Plot relationship between profit and average domestic gross with confidence interval line
fig, ax = plt.subplots(figsize=(15, 8))

# Create plot
sns.regplot(
    x='cpi_avg_domestic_gross',
    y='domestic_profit',
    data=money_df,
    scatter_kws={"color": 'black'},
    line_kws={"color": '#DAA520'},
    ci=95,  # Adjust the confidence interval
)

plt.title('Relationship between Average Domestic Gross and Profit', fontfamily='Arial', fontsize=15)
plt.xlabel('Average Domestic Gross in Billions of USD', fontfamily='Arial', fontsize=15)
plt.ylabel('Profit in Billions of USD', fontfamily='Arial', fontsize=15)

# Define a custom formatter function to display values in billions without scientific notation
def billions_formatter(x, pos):
    """
    Custom formatter for formatting values in billions without scientific notation.

    Parameters:
    - x : Value to be formatted
    - pos : Position of the tick on the axis

    Returns:
    - formatted_string : Formatted string representing the value in billions
    """
    return f'${x / 1e9:.2f}B'

# Apply the custom formatter to the x-axis and y-axis
ax.xaxis.set_major_formatter(billions_formatter)
ax.yaxis.set_major_formatter(billions_formatter)

plt.show()


```



→ This plot shows a strong, positive, linear association between profit and average domestic gross with a few outliers.

This means that it can assumed (based on how profit has been previously defined), that top grossing films are also the most profitable.

Next, is an exploration into profit a little further to understand the difference between profit and profit margin and realize which one serves as a better measurement for understanding the costs associated with making movies. Since it's been shown that top grossing movies are the most profitable, the top 100 domestic profit movies will be considered in this analysis.

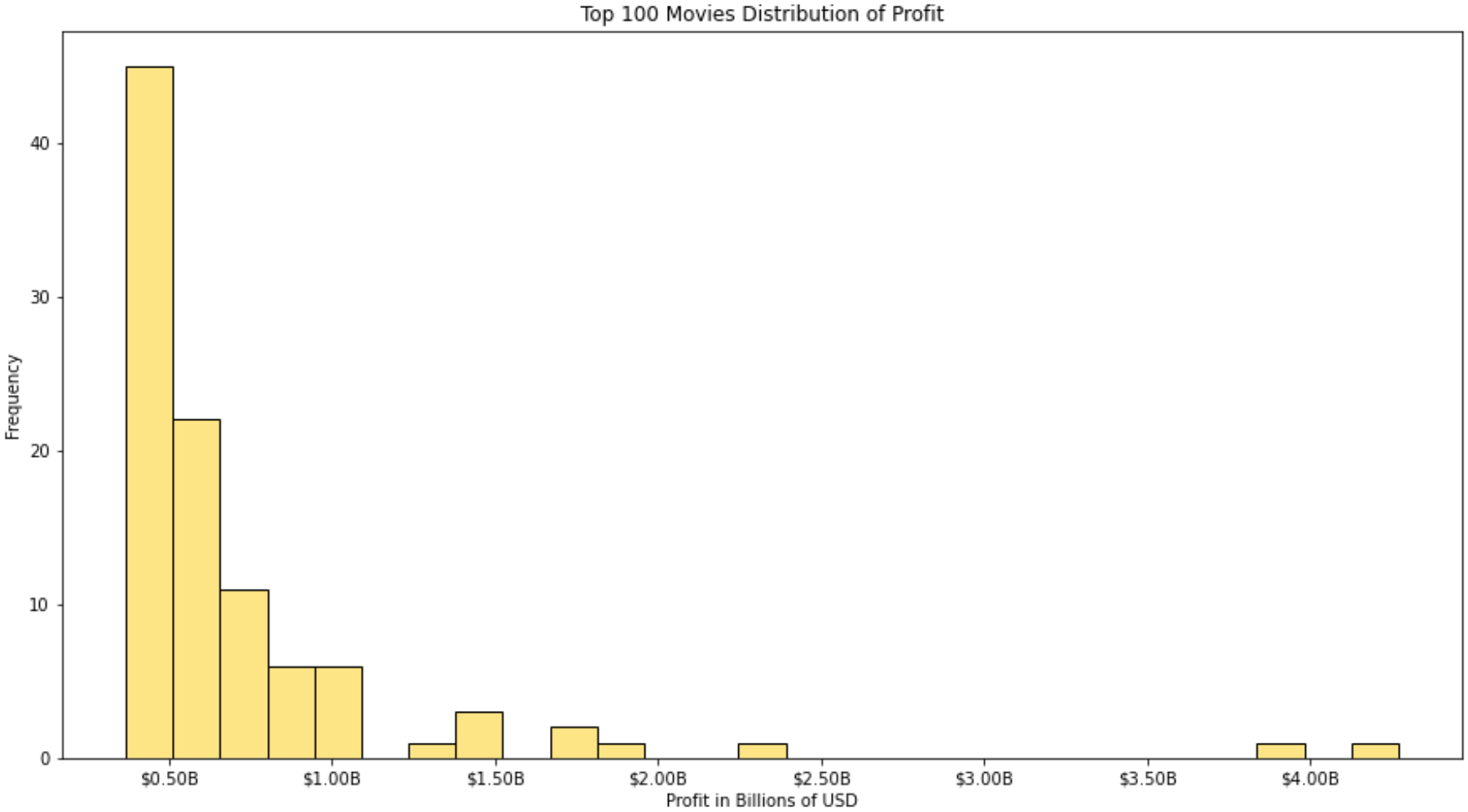
In [38]:
```
# Initial setup of top_profit as top 100 domestic profit movies
top_profit = money_df.sort_values('domestic_profit', ascending = False)[:100]
```

In [39]:
```
# Describe profit
# Use plain format to remove scientific notation - improves readability
top_profit['domestic_profit'].describe().apply(lambda x: format(x, 'f'))
```

Out[39]:
```
count           100.000000
mean      721823114.735430
std       602538100.653499
min       367434764.705882
25%       413208392.839609
50%       534583077.617109
75%       758235838.292039
max      4269784084.168346
Name: domestic_profit, dtype: object
```

In [40]:

```python
# Plot distribution of profit using histplot
fig, ax = plt.subplots(figsize=(15, 8))

sns.histplot(top_profit['domestic_profit'], color= '#FDDC5C', fill=True, edgecolor='black')
plt.title('Top 100 Movies Distribution of Profit')
plt.xlabel('Profit in Billions of USD')
plt.ylabel('Frequency')

# Define a custom formatter function to display values in billions without scientific notation
def billions_formatter(x, pos):
    """
    Custom formatter for formatting values in billions without scientific notation.

    Parameters:
    - x : Value to be formatted
    - pos : Position of the tick on the axis

    Returns:
    - formatted_string : Formatted string representing the value in billions
    """
    return f'${x / 1e9:.2f}B'

# Apply the custom formatter to the x-axis and y-axis
ax.xaxis.set_major_formatter(billions_formatter)

plt.show()
```



This graph shows that the median profit for top 100 domestic movies is approximately 500 million US dollars.

In [41]:
```
# Get summary statistics for profit margin
# Use plain formatting to remove scientific notation
money_df['domestic_profit_margin'].describe().apply(lambda x: format(x, 'f'))
```

Out[41]:
```
count        5238.000000
mean        -4459.698385
std         70484.653187
min      -3267873.856209
25%           -156.738678
50%              2.931248
75%             55.418350
max             99.944444
Name: domestic_profit_margin, dtype: object
```

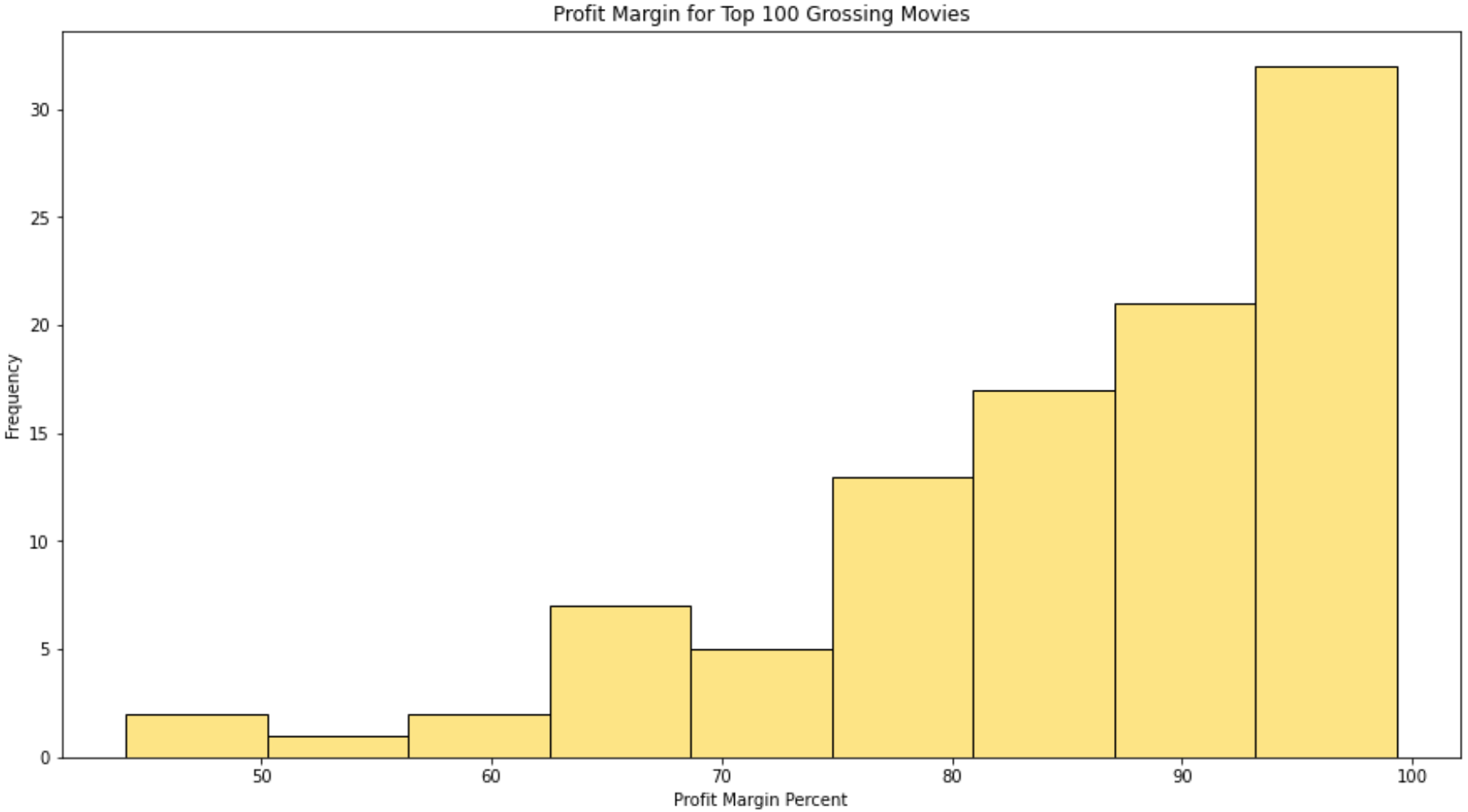
→ There is a significant outlier with a large negative value. Additionally, a considerable portion of these movies have a negative profit margin, indicating a loss. So again, the analysis will only consider the top 100 profitable movies.

In [42]:
```
# Set top_profit as top 100 movies
# Describe profit
top_profit['domestic_profit_margin'].describe().apply(lambda x: format(x, 'f'))
```

Out[42]:
```
count    100.000000
mean      85.139087
std       11.931149
min       44.116274
25%       79.593959
50%       88.051873
75%       94.760766
max       99.324348
Name: domestic_profit_margin, dtype: object
```

In [43]:

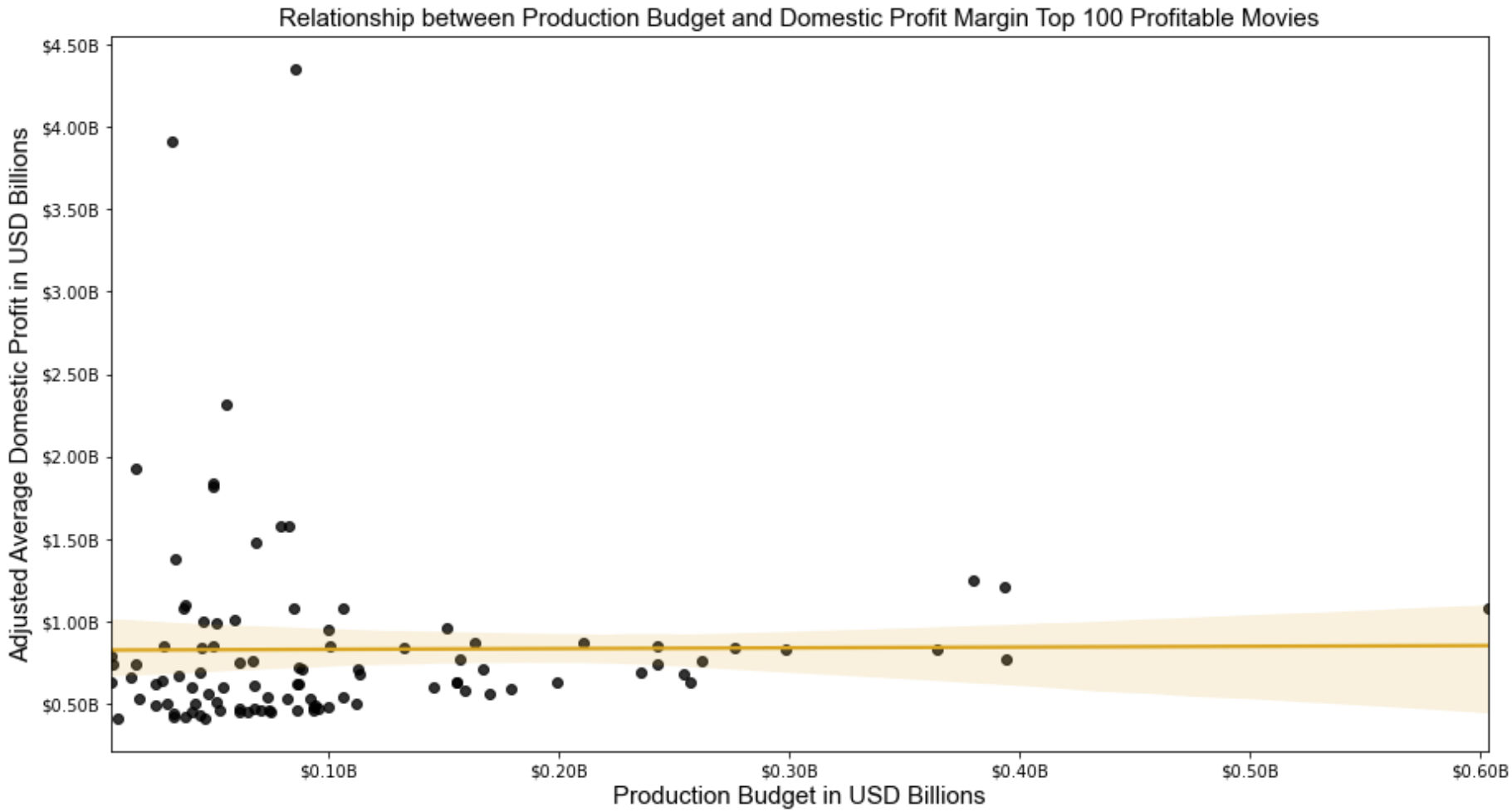
```
# Plot distribution of profit margin for top_profit using histplot
plt.figure(figsize = (15,8))
sns.histplot(top_profit['domestic_profit_margin'], color= '#FDDC5C', fill=True, edgecolor='black')
plt.title('Profit Margin for Top 100 Grossing Movies')
plt.xlabel('Profit Margin Percent')
plt.ylabel('Frequency')
plt.show()

```



The median profit margin for a top 100 all-time domestic grossing movies is approximately 88%

In [44]:

```python
# Create a scatter plot with confidence interval line
fig, ax = plt.subplots(figsize=(15, 8))

sns.regplot(
    x='cpi_production_budget',
    y='cpi_avg_domestic_gross',
    data=top_profit,
    scatter_kws={"color": 'black'},
    line_kws={"color": '#DAA520'},
    ci=95,  # Adjust the confidence interval
)

plt.title('Relationship between Production Budget and Domestic Profit Margin Top 100 Profitable Movies
plt.xlabel('Production Budget in USD Billions', fontfamily='Arial', fontsize=15)
plt.ylabel('Adjusted Average Domestic Profit in USD Billions', fontfamily='Arial', fontsize=15)

# Define a custom formatter function to display values in billions without scientific notation
def billions_formatter(x, pos):
    """
    Custom formatter for formatting values in billions without scientific notation.

    Parameters:
    - x : Value to be formatted
    - pos : Position of the tick on the axis

    Returns:
    - formatted_string : Formatted string representing the value in billions
    """
    return f'${x / 1e9:.2f}B'

# Apply the custom formatter to both x-axis and y-axis
x_formatter = FuncFormatter(billions_formatter)
ax.xaxis.set_major_formatter(x_formatter)
ax.yaxis.set_major_formatter(billions_formatter)

plt.show()

```



In [45]:
```
# Set top_profit as top 100 movies
# Describe profit
top_profit['cpi_production_budget'].describe().apply(lambda x: format(x, 'f'))
```

Out[45]:
```
count          100.000000
mean     106905490.695924
std      101674087.339997
min        5332285.000000
25%       44218569.249904
50%       74142873.566314
75%      136047153.909770
max      603617790.870572
Name: cpi_production_budget, dtype: object
```

This plot between production budget and adjusted average domestic gross of the top 100 grossing movies, does not show a clear correlation. It would seem that movies have seen success with small budgets and others depsite robust budgets, failed to perform. It appears that budgets between 44 and 136 million seem to be reasonable.

In [46]:
```
# Get top profit movies in descending order
print('Total number of results:', len(top_profit))
display(top_profit.head())
```

```
Total number of results: 100
```

	title	foreign_gross	year	release_date	production_budget	worldwide_gross	avg_domestic_gross	cpi_worldwide_gross	cpi_avg_dome
6814	gone with the wind	NaN	1939	1939-12-15	3900000.0	390525192.0	198680470.0	8.560706e+09	4.35
7293	snow white and the seven dwarfs	NaN	1937	1937-12-21	1488000.0	184925486.0	184925486.0	3.912998e+09	3.91
5928	star wars ep. iv: a new hope	NaN	1977	1977-05-25	11000000.0	786598007.0	460998007.0	3.955082e+09	2.31
7497	bambi	NaN	1942	1942-08-13	858000.0	268000000.0	102797000.0	5.009824e+09	1.92
7073	pinocchio	NaN	1940	1940-02-09	2289247.0	84300000.0	84300000.0	1.834741e+09	1.83

In [47]:
```
# Get top profit margin movies
top_margin = top_profit.sort_values('domestic_profit_margin', ascending = False)
print('Total number of results:', len(top_margin))
display(top_margin.head())
```

```
Total number of results: 100
```

	title	foreign_gross	year	release_date	production_budget	worldwide_gross	avg_domestic_gross	cpi_worldwide_gross	cpi_avg_domes
7521	american graffiti	NaN	1973	1973-08-11	777000.0	140000000.0	115000000.0	9.607721e+08	7.89
7293	snow white and the seven dwarfs	NaN	1937	1937-12-21	1488000.0	184925486.0	184925486.0	3.912998e+09	3.91
7507	billy jack	NaN	1971	1971-01-01	800000.0	98000000.0	98000000.0	7.373036e+08	7.37
7497	bambi	NaN	1942	1942-08-13	858000.0	268000000.0	102797000.0	5.009824e+09	1.92
7384	rocky	NaN	1976	1976-11-21	1000000.0	225000000.0	117235147.0	1.204885e+09	6.27

The movies with the all-time highest profit differ from those with the highest all-time profit margin.

Profit and Profit Margin Analysis Summary

Production budgets for the top 100 profit movies varies greatly ranging from around 5 million to around 600 million. This shows that increasing the movie production budget does not necessarily increase profits.

A further investigation of the aspects included in the production budget could reveal how to best allocate the production budget.

Aggragating additional release date and title names

In [48]:
```
1 tmdb_movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26517 entries, 0 to 26516
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Unnamed: 0         26517 non-null  int64  
 1   genre_ids          26517 non-null  object 
 2   id                 26517 non-null  int64  
 3   original_language  26517 non-null  object 
 4   original_title     26517 non-null  object 
 5   popularity         26517 non-null  float64
 6   release_date       26517 non-null  object 
 7   title              26517 non-null  object 
 8   vote_average       26517 non-null  float64
 9   vote_count         26517 non-null  int64  
dtypes: float64(2), int64(3), object(5)
memory usage: 2.0+ MB
```

In [49]:

```
# Change the movie titles to lowercase
tmdb_movies['original_title'] = tmdb_movies['original_title'].str.lower()
tmdb_movies['title'] = tmdb_movies['title'].str.lower()

# Create year column
tmdb_movies['year'] = pd.DatetimeIndex(tmdb_movies['release_date']).year

print('Total number of results:', len(tmdb_movies))
display(tmdb_movies.head())
```

```
Total number of results: 26517
```

	Unnamed: 0	genre_ids	id	original_language	original_title	popularity	release_date	title	vote_average	vote_count	year
0	0	[12, 14, 10751]	12444	en	harry potter and the deathly hallows: part 1	33.533	2010-11-19	harry potter and the deathly hallows: part 1	7.7	10788	2010
1	1	[14, 12, 16, 10751]	10191	en	how to train your dragon	28.734	2010-03-26	how to train your dragon	7.7	7610	2010
2	2	[12, 28, 878]	10138	en	iron man 2	28.515	2010-05-07	iron man 2	6.8	12368	2010
3	3	[16, 35, 10751]	862	en	toy story	28.005	1995-11-22	toy story	7.9	10174	1995
4	4	[28, 878, 12]	27205	en	inception	27.920	2010-07-16	inception	8.3	22186	2010

In [50]:

```
# Merge money_df and tmdb_movies dataframes
many_titles_df = pd.merge(money_df, tmdb_movies, on=['title', 'year'], how='outer')
print('Total number of results:', len(many_titles_df))
display(many_titles_df.head())
```

```
Total number of results: 31373
```

	title	foreign_gross	year	release_date_x	production_budget	worldwide_gross	avg_domestic_gross	cpi_worldwide_gross	cpi_avg_domes
0	'71	355000.0	2015	NaT	NaN	NaN	1300000.0	NaN	1.67
1	(500) days of summer	NaN	2009	2009-07-17	7500000.0	34439060.0	32425665.0	4.891301e+07	4.60
2	1,000 times good night	NaN	2014	NaT	NaN	NaN	53900.0	NaN	6.93
3	10 cloverfield lane	38100000.0	2016	2016-03-11	5000000.0	108286422.0	72091499.5	1.374755e+08	9.15

In [51]:

```
# Fill NaN or 0.0 cells in column release_date_y with value from release_date_x
many_titles_df['release_date_y'].fillna(many_titles_df['release_date_x'], inplace=True)

# Drop the 'release_date_x' column
many_titles_df.drop(columns=['release_date_x'], inplace=True)

# Rename 'release_date_y' to 'release_date'
many_titles_df.rename(columns={'release_date_y': 'release_date'}, inplace=True)

# Drop the 'id and Unnamed columns' columns
many_titles_df.drop(columns=['id', 'Unnamed: 0'], inplace=True)

# Drop duplicates
many_titles_df.drop_duplicates()

print('Total number of results:', len(many_titles_df))
display(many_titles_df.head())
```

```
Total number of results: 31373
```

	title	foreign_gross	year	production_budget	worldwide_gross	avg_domestic_gross	cpi_worldwide_gross	cpi_avg_domestic_gross	cpi_fo
0	'71	355000.0	2015	NaN	NaN	1300000.0	NaN	1.671241e+06	4
1	(500) days of summer	NaN	2009	7500000.0	34439060.0	32425665.0	4.891301e+07	4.605343e+07	
2	1,000 times good night	NaN	2014	NaN	NaN	53900.0	NaN	6.937448e+04	
3	10 cloverfield lane	38100000.0	2016	5000000.0	108286422.0	72091499.5	1.374755e+08	9.152410e+07	4
4	10 days in a madhouse	NaN	2015	12000000.0	14616.0	14616.0	1.878989e+04	1.878989e+04	

In [52]:

```
# Replace NaN values in 'original_title' with values from 'title'
many_titles_df['original_title'].fillna(many_titles_df['title'], inplace=True)

# Replace NaN values in 'title' with values from 'original_title'
many_titles_df['title'].fillna(many_titles_df['original_title'], inplace=True)

```

In [53]:

```
# Find and count duplicates between the two columns
duplicate_count = many_titles_df[many_titles_df.duplicated(subset=['original_title', 'title'], keep=Fal

print(f"Number of duplicate titles between 'original_title' and 'title': {duplicate_count}")

```

```
Number of duplicate titles between 'original_title' and 'title': 3924
```

In [54]:

```python
# Create a boolean mask to identify rows where values in the two columns differ
mask = many_titles_df['original_title'] != many_titles_df['title']

# Extract rows with differences in the specified columns
differing_rows = many_titles_df[mask][['original_title', 'title']]

# Print the differing rows
print("Rows with differences in 'original_title' and 'title':")
display(differing_rows)

# Print the length of the list
f"Number of differing rows: {len(differing_rows)}"
```

```
Rows with differences in 'original_title' and 'title':
```

	original_title	title
18	十三人の刺客	13 assassins
21	elser	13 minutes
29	辛亥革命	1911
36	टू स्टेट्स	2 states
103	7 cajas	7 boxes
...	...	...
31277	czarne lusterko: 69.90	little black mirror: 69.90
31292	el verano del león eléctrico	the summer of the electric lion
31301	contes de juillet	july tales
31353	la última virgen	the last virgin
31365	dreamaway	dream away

2532 rows × 2 columns

Out[54]:

```
'Number of differing rows: 2532'
```

In [55]:
```
# Drop the 'original_title' column
many_titles_df.drop(columns=['original_title'], inplace=True)
```

→ Code blocks 54-56 discover that the differences between the title names can be attributed to being documneted in different languages.

Extract and aggragate SQLite Tables to Pandas dataframe

In [56]:
```
# Connect to SQL Database
conn = sqlite3.connect('Data/im.db')

# Open SQL Table
table_names = ['movie_basics']

# Create an empty dictionary to store DataFrames
SQL_dataframes = {}

# Loop through table names and read data into DataFrames
for table in table_names:
    sql_query = f'SELECT * FROM {table}'
    SQL_dataframes[table] = pd.read_sql(sql_query, conn)

```

In [57]:

```
# Preview the data from SQL DataFrames
for name, df in SQL_dataframes.items():
    print(name)
    print('Total number of results:', len(df))
    display(df.head())  # Using display instead of print leads to neater formatting in Jupyter Notebook
    print(df.info())
    print('------------------ \n')

```

```
movie_basics
Total number of results: 146144
```

	movie_id	primary_title	original_title	start_year	runtime_minutes	genres
0	tt0063540	Sunghursh	Sunghursh	2013	175.0	Action,Crime,Drama
1	tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	114.0	Biography,Drama
2	tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018	122.0	Drama
3	tt0069204	Sabse Bada Sukh	Sabse Bada Sukh	2018	NaN	Comedy,Drama
4	tt0100275	The Wandering Soap Opera	La Telenovela Errante	2017	80.0	Comedy,Drama,Fantasy

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 146144 entries, 0 to 146143
Data columns (total 6 columns):
 #   Column           Non-Null Count   Dtype  
---  ------           --------------   -----  
 0   movie_id         146144 non-null  object 
 1   primary_title    146144 non-null  object 
 2   original_title   146123 non-null  object 
 3   start_year       146144 non-null  int64  
 4   runtime_minutes  114405 non-null  float64
 5   genres           140736 non-null  object 
dtypes: float64(1), int64(1), object(4)
memory usage: 6.7+ MB
None
------------------
```

In [58]:

```
# Access the 'movie_basics' DataFrame from the SQL_dataframes dictionary
movie_basics_df = SQL_dataframes['movie_basics']

# Create a copy of the DataFrame to avoid altering the original
movie_basics_ga = movie_basics_df.copy()

# Merge the DataFrames on 'primary_title' and 'start_year' in 'movie_basics_ga' with 'title' and 'year'
main_df = pd.merge(
    movie_basics_ga,
    many_titles_df,
    left_on=['primary_title', 'start_year'],
    right_on=['title', 'year'],
    how='outer'
).drop_duplicates()



print('Total number of results:', len(main_df))
display(main_df.head())
```

```
Total number of results: 176449
```

	movie_id	primary_title	original_title	start_year	runtime_minutes	genres	title	foreign_gross	year	production_budget	...	cpi
0	tt0063540	Sunghursh	Sunghursh	2013.0	175.0	Action,Crime,Drama	NaN	NaN	NaN	NaN	...	
1	tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019.0	114.0	Biography,Drama	NaN	NaN	NaN	NaN	...	
2	tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018.0	122.0	Drama	NaN	NaN	NaN	NaN	...	
3	tt0069204	Sabse Bada Sukh	Sabse Bada Sukh	2018.0	NaN	Comedy,Drama	NaN	NaN	NaN	NaN	...	
4	tt0100275	The Wandering Soap Opera	La Telenovela Errante	2017.0	80.0	Comedy,Drama,Fantasy	NaN	NaN	NaN	NaN	...	

5 rows × 25 columns

In [59]:

```
# Convert strings to lowercase in 'primary_title_y' and 'title' columns
main_df['primary_title'] = main_df['primary_title'].str.lower()

# Convert strings to lowercase in 'original_title' and 'title' columns
main_df['original_title'] = main_df['original_title'].str.lower()

# Find and count duplicates between the two columns
duplicate_count = main_df[main_df.duplicated(subset=['primary_title', 'title'], keep=False)].shape[0]

print(f"Number of duplicate titles between 'primary_title' and 'title': {duplicate_count}")
```

```
Number of duplicate titles between 'primary_title' and 'title': 19165
```

In [60]:

```
# Fill NaN or 0.0 cells in column primary_title with value from title
main_df['primary_title'].fillna(main_df['title'], inplace=True)

# Fill NaN or 0.0 cells in column primary_title with value from original_title
main_df['primary_title'].fillna(main_df['original_title'], inplace=True)

# Fill NaN or 0.0 cells in column start_year with value from year
main_df['start_year'].fillna(main_df['year'], inplace=True)

# Drop the 'year' column
main_df.drop(columns=['year'], inplace=True)

# Rename 'start_year' to 'years'
main_df.rename(columns={'start_year': 'release_year'}, inplace=True)

# Drop the 'title' column
main_df.drop(columns=['title','original_title'], inplace=True)

# Rename 'primary_title' to 'title'
main_df.rename(columns={'primary_title': 'title'}, inplace=True)

print('Total number of results:', len(main_df))
display(main_df.head())
```

```
Total number of results: 176449
```

	movie_id	title	release_year	runtime_minutes	genres	foreign_gross	production_budget	worldwide_gross	avg_domestic_
0	tt0063540	sunghursh	2013.0	175.0	Action,Crime,Drama	NaN	NaN	NaN	
1	tt0066787	one day before the rainy season	2019.0	114.0	Biography,Drama	NaN	NaN	NaN	
2	tt0069049	the other side of the wind	2018.0	122.0	Drama	NaN	NaN	NaN	
3	tt0069204	sabse bada sukh	2018.0	NaN	Comedy,Drama	NaN	NaN	NaN	
4	tt0100275	the wandering soap opera	2017.0	80.0	Comedy,Drama,Fantasy	NaN	NaN	NaN	

5 rows × 22 columns

In [61]:

```
# Check for duplicates in the 'title' column
title_year_duplicates = main_df.duplicated(subset=['title','release_year'], keep=False)

# Display a sample of rows with duplicate titles (view only)
sample_duplicates = main_df[title_year_duplicates].sort_values(by='title')

print('Total number of results:', len(sample_duplicates))
display(sample_duplicates.head())
```

```
Total number of results: 29180
```

	movie_id	title	release_year	runtime_minutes	genres	foreign_gross	production_budget	worldwide_gross	avg_domestic_g
168121	NaN	#allmymovies	2015.0	NaN	NaN	NaN	NaN	NaN	
128223	tt7853996	#allmymovies	2015.0	NaN	Documentary	NaN	NaN	NaN	
173604	NaN	#captured	2017.0	NaN	NaN	NaN	NaN	NaN	
115921	tt6856592	#captured	2017.0	81.0	Thriller	NaN	NaN	NaN	
170488	NaN	#followfriday	2016.0	NaN	NaN	NaN	NaN	NaN	

5 rows × 22 columns

In [62]:

```
# Split out main_df into duplicates and unique
# Make 'unique' title df
main_df_unique = main_df[~title_year_duplicates].copy()

# Handle duplicates and concat them back to the above unique df
# Make 'duplicate' title df
main_df_dups = main_df[title_year_duplicates].copy()

print('Total number of results:', len(main_df_dups))
display(main_df_dups.head())
```

```
Total number of results: 29180
```

	movie_id	title	release_year	runtime_minutes	genres	foreign_gross	production_budget	worldwide_gross	avg_domesti
2	tt0069049	the other side of the wind	2018.0	122.0	Drama	NaN	NaN	NaN	
17	tt0192528	heaven & hell	2018.0	104.0	Drama	NaN	NaN	NaN	
21	tt0250404	godfather	2012.0	NaN	Crime,Drama	NaN	NaN	NaN	
22	tt3270104	godfather	2012.0	145.0	Drama	NaN	NaN	NaN	
24	tt0255820	return to babylon	2013.0	75.0	Biography,Comedy,Drama	NaN	NaN	NaN	

5 rows × 22 columns

In [63]:

```
# Get value counts
title_counts = main_df_dups['title'].value_counts()

# Filter titles that appear more than twice
titles_more_than_twice = title_counts[title_counts > 2].index.tolist()

# Display the titles
print("Titles listed more than twice:")
print(len(titles_more_than_twice))
```

```
Titles listed more than twice:
805
```

→ It looks like some of the films may have multiple release years, which is a factor to consider in removing duplicates.

In [64]:

```python
# Define a custom function to choose the longer of the two genres
def choose_longer_genre(series):
    '''
    All NaN values within genres will be filled from duplicate title entries in the data frame.
    The longer string value of the two genres is chosen when there are different values between rows.
    -
    Input:
    series : Pandas Series, represents the 'genres' column
    -
    Output:
    value : Chosen genre value based on conditions
    '''

    if series.isnull().any():
        # If one of the values is missing, choose the non-null value
        return next((value for value in series.dropna()), None)
    else:
        # If both values are present, choose the longer one
        return max(series, key=len)

# Apply the custom function to the 'genres' column
main_df_dups['genres'] = main_df_dups.groupby(['title', 'release_year'])['genres'].transform(choose_lor

# Columns to fill NaN values from the next row
cols_to_fill = ['runtime_minutes', 'release_date', 'movie_id', 'cpi_avg_domestic_gross']

# Forward fill NaN values from the next row
main_df_dups[cols_to_fill] = main_df_dups.groupby(['title', 'release_year'])[cols_to_fill].transform(la

print('Total number of results:', len(main_df_dups))
display(main_df_dups.head())

```

```
Total number of results: 29180
```

	movie_id	title	release_year	runtime_minutes	genres	foreign_gross	production_budget	worldwide_gross	avg_domesti
2	tt0069049	the other side of the wind	2018.0	122.0	Drama	NaN	NaN	NaN	
17	tt0192528	heaven & hell	2018.0	104.0	Drama	NaN	NaN	NaN	
21	tt0250404	godfather	2012.0	145.0	Crime,Drama	NaN	NaN	NaN	
22	tt3270104	godfather	2012.0	145.0	Crime,Drama	NaN	NaN	NaN	
24	tt0255820	return to babylon	2013.0	75.0	Biography,Comedy,Drama	NaN	NaN	NaN	

5 rows × 22 columns

In [65]:
```
# Drop duplicate rows and keep the first occurrence
main_df_dups_dropped = main_df_dups.drop_duplicates(subset=['title', 'release_year'], keep='first')

print('Total number of results:', len(main_df_dups_dropped))
display(main_df_dups_dropped.head())
```

```
Total number of results: 14252
```

	movie_id	title	release_year	runtime_minutes	genres	foreign_gross	production_budget	worldwide_gross	avg_domesti
2	tt0069049	the other side of the wind	2018.0	122.0	Drama	NaN	NaN	NaN	
17	tt0192528	heaven & hell	2018.0	104.0	Drama	NaN	NaN	NaN	
21	tt0250404	godfather	2012.0	145.0	Crime,Drama	NaN	NaN	NaN	
24	tt0255820	return to babylon	2013.0	75.0	Biography,Comedy,Drama	NaN	NaN	NaN	
39	tt0315642	wazir	2016.0	103.0	Action,Crime,Drama	NaN	NaN	NaN	

5 rows × 22 columns

In [66]:

```
# Rebuilding the main_df, but with the duplicates removed
# Main_df_unique + main_df_dups_dropped
main_df_clean = main_df_unique.append(main_df_dups_dropped, ignore_index=True)

print('Total number of results:', len(main_df_clean))
display(main_df_clean.head())
```

```
Total number of results: 161521
```

	movie_id	title	release_year	runtime_minutes	genres	foreign_gross	production_budget	worldwide_gross	avg_domestic_
0	tt0063540	sunghursh	2013.0	175.0	Action,Crime,Drama	NaN	NaN	NaN	
1	tt0066787	one day before the rainy season	2019.0	114.0	Biography,Drama	NaN	NaN	NaN	
2	tt0069204	sabse bada sukh	2018.0	NaN	Comedy,Drama	NaN	NaN	NaN	
3	tt0100275	the wandering soap opera	2017.0	80.0	Comedy,Drama,Fantasy	NaN	NaN	NaN	
4	tt0111414	a thin life	2018.0	75.0	Comedy	NaN	NaN	NaN	

5 rows × 22 columns

Genre Analysis

The genre associated with a movie may greatly influence profitability. By analyzing the adjusted average domestic gross for each individual genre, it may shed light on a factor that contributes to a film's success at the box office.

In [67]:

```
# Create a copy of the DataFrame to avoid altering the original
genre_analysis_explode = main_df_clean.copy()

# Split the genres into separate rows and create a new 'genre' column
genre_analysis_explode['genre'] = genre_analysis_explode['genres'].str.split(',')

# Explode the 'genre' column to create new rows for each genre
genre_analysis_explode = genre_analysis_explode.explode('genre')

# Display the updated DataFrame
print('Total number of results:', len(genre_analysis_explode))
display(genre_analysis_explode.head())

```

```
Total number of results: 249658
```

	movie_id	title	release_year	runtime_minutes	genres	foreign_gross	production_budget	worldwide_gross	avg_domestic_gros
0	tt0063540	sunghursh	2013.0	175.0	Action,Crime,Drama	NaN	NaN	NaN	Na
0	tt0063540	sunghursh	2013.0	175.0	Action,Crime,Drama	NaN	NaN	NaN	Na
0	tt0063540	sunghursh	2013.0	175.0	Action,Crime,Drama	NaN	NaN	NaN	Na
1	tt0066787	one day before the rainy season	2019.0	114.0	Biography,Drama	NaN	NaN	NaN	Na
1	tt0066787	one day before the rainy season	2019.0	114.0	Biography,Drama	NaN	NaN	NaN	Na

In [68]:

```
# Count the occurrences of each genre and convert to DataFrame
genre_counts = genre_analysis_explode['genre'].value_counts().reset_index()
genre_counts.columns = ['genre', 'count']

# Get the total sum of value counts
total_count = genre_counts['count'].sum()

# Calculate the percentage of each genre
genre_percentages = (genre_counts['count'] / total_count) * 100

# Create a DataFrame to display the results
percent_genre = pd.DataFrame({
    'Genre': genre_counts['genre'],
    'Count': genre_counts['count'],
    'Percentage': genre_percentages
})

# Display the results
print('Total number of results:', total_count)
display(percent_genre)

```

```
Total number of results: 226988
```

	Genre	Count	Percentage
0	Documentary	51191	22.552294
1	Drama	48979	21.577793
2	Comedy	25120	11.066664
3	Thriller	11701	5.154898
4	Horror	10655	4.694081
5	Action	10222	4.503322
6	Romance	9274	4.085679
7	Biography	8687	3.827075
8	Crime	6667	2.937160
9	Adventure	6426	2.830987

In [69]:

```python
# Define your color codes for each genre
color_codes = {
    'Documentary': '#FCC252',
    'Drama': '#FCC252',
    'Comedy': '#FCC252',
    'Thriller': '#C0C1C1',
    'Horror': '#C0C1C1',
    'Action': '#C0C1C1',
    'Romance': '#C0C1C1',
    'Biography': '#C0C1C1',
    'Crime': '#C0C1C1',
    'Adventure': '#DC2026',
    'History': '#DC2026',
    'Family': '#DC2026',
    'Mystery': '#DC2026',
    'Music': '#DC2026',
    'Fantasy': '#DC2026',
    'Sci-Fi': '#DC2026',
    'Animation': '#DC2026',
    'Sport': '#DC2026',
    'News': '#DC2026',
    'Musical': '#DC2026',
    'War': '#DC2026',
    'Western': '#DC2026',
    'Reality-TV': '#DC2026',
    'Talk-Show': '#DC2026',
    'Adult': '#DC2026',
    'Short': '#DC2026',
    'Game-Show': '#DC2026',
}

# Create a new column 'color' in the dataframe based on genre
genre_counts['color'] = genre_counts['genre'].apply(lambda x: color_codes.get(x, '#FCC252'))

# Define treemap trace
treemap_trace = go.Treemap(
    labels=genre_counts['genre'],
    parents=[''] * len(genre_counts['genre']),
    values=genre_counts['count'],
    marker=dict(
        colors=genre_counts['color'],
        line=dict(
            color=['#000000' if genre in ['Action', 'Adventure', 'Comedy'] else 'rgba(0,0,0,0)' for ge
            width=2
```

```
        )
    )

)

# Define symbols for specific genres
symbols = ['*' if genre in ['Action', 'Adventure', 'Comedy'] else '' for genre in genre_counts['genre']

# Add annotations for symbols
annotations = [dict(text=symbol, x=genre, y=1, font=dict(size=12)) for genre, symbol in zip(genre_count

# Create layout
layout = go.Layout(
    title='Tree Map: Movie Production Count by Genre',
    annotations=annotations,
    height=800,
    width=1000,
    font=dict(
        family="Times New Roman, serif",
        size=14,
        color="black"
    )
)

# Create figure
fig = go.Figure(data=[treemap_trace], layout=layout)

# Show plot
fig.show()

```

Tree Map: Movie Production Count by Genre

In a Tree Map the size of the box correlates to the number of movies produced. The top three most profitable genres of Action, Adventure, and Comedy are outlined in black. More commonly made movies are not always correlated to the top grossing genres. This could be due to the overall cost of making Action, Adventure, and Comedy movies which could be determined upon further investigation.

In [70]:

```
# Group by 'genre' and sum the 'cpi_avg_domestic_gross' column
genre_summary = genre_analysis_explode.groupby('genre')['cpi_avg_domestic_gross'].sum().reset_index()

# Sort the summary by 'cpi_avg_domestic_gross' in descending order
genre_summary = genre_summary.sort_values('cpi_avg_domestic_gross', ascending=False)

# Display the summary
display(genre_summary)
```

	genre	cpi_avg_domestic_gross
2	Adventure	6.058650e+10
0	Action	5.454206e+10
5	Comedy	4.331613e+10
8	Drama	3.406927e+10
20	Sci-Fi	2.179418e+10
3	Animation	1.899803e+10
24	Thriller	1.671366e+10
10	Fantasy	1.451499e+10
6	Crime	1.235779e+10
9	Family	9.881030e+09
19	Romance	9.533938e+09
13	Horror	7.831989e+09
4	Biography	7.396756e+09
16	Mystery	6.891842e+09
14	Music	2.541608e+09
12	History	2.417894e+09
22	Sport	1.536252e+09
15	Musical	1.111355e+09
7	Documentary	8.277858e+08
26	Western	8.044687e+08
25	War	5.801241e+08
17	News	1.726527e+04
1	Adult	0.000000e+00
18	Reality-TV	0.000000e+00
11	Game-Show	0.000000e+00
21	Short	0.000000e+00

	genre	cpi_avg_domestic_gross
23	Talk-Show	0.000000e+00

In [71]:

```
# View numeric and word format of the domestic gross values by genre

# Set the locale for formatting numbers
locale.setlocale(locale.LC_ALL, 'en_US.UTF-8')

# Function to format numbers in both numeric and word format
def format_numbers(num):
    '''
    Function to format numbers in both numeric and word format.

    Input:
    num : Numeric value to be formatted

    Output:
    num_formatted : Formatted numeric value with grouping
    num_words : Numeric value converted to words
    '''

    # Format number in numeric format
    num_formatted = locale.format_string('%.0f', num, grouping=True)

    # Convert number to words
    p = inflect.engine()
    num_words = p.number_to_words(int(num)).replace('-', ' ')

    return num_formatted, num_words

# Apply the function to the 'cpi_avg_domestic_gross' column
genre_summary['numeric_format'], genre_summary['word_format'] = zip(*genre_summary['cpi_avg_domestic_gr

# Display the updated summary without genres that report zero gross
display(genre_summary[genre_summary['cpi_avg_domestic_gross'] != 0])

```

	genre	cpi_avg_domestic_gross	numeric_format	word_format
2	Adventure	6.058650e+10	60,586,497,706	sixty billion, five hundred and eighty six mil...
0	Action	5.454206e+10	54,542,059,389	fifty four billion, five hundred and forty two...
5	Comedy	4.331613e+10	43,316,127,595	forty three billion, three hundred and sixteen...
8	Drama	3.406927e+10	34,069,269,087	thirty four billion, sixty nine million, two h...
20	Sci-Fi	2.179418e+10	21,794,184,801	twenty one billion, seven hundred and ninety f...
3	Animation	1.899803e+10	18,998,033,334	eighteen billion, nine hundred and ninety eigh...
24	Thriller	1.671366e+10	16,713,659,373	sixteen billion, seven hundred and thirteen mi...
10	Fantasy	1.451499e+10	14,514,988,614	fourteen billion, five hundred and fourteen mi...
6	Crime	1.235779e+10	12,357,785,265	twelve billion, three hundred and fifty seven ...
9	Family	9.881030e+09	9,881,029,880	nine billion, eight hundred and eighty one mil...
19	Romance	9.533938e+09	9,533,937,567	nine billion, five hundred and thirty three mi...
13	Horror	7.831989e+09	7,831,988,544	seven billion, eight hundred and thirty one mi...
4	Biography	7.396756e+09	7,396,755,647	seven billion, three hundred and ninety six mi...
16	Mystery	6.891842e+09	6,891,842,100	six billion, eight hundred and ninety one mill...
14	Music	2.541608e+09	2,541,608,385	two billion, five hundred and forty one millio...
12	History	2.417894e+09	2,417,893,750	two billion, four hundred and seventeen millio...
22	Sport	1.536252e+09	1,536,252,159	one billion, five hundred and thirty six milli...
15	Musical	1.111355e+09	1,111,355,064	one billion, one hundred and eleven million, t...
7	Documentary	8.277858e+08	827,785,784	eight hundred and twenty seven million, seven ...
26	Western	8.044687e+08	804,468,702	eight hundred and four million, four hundred a...
25	War	5.801241e+08	580,124,132	five hundred and eighty million, one hundred a...
17	News	1.726527e+04	17,265	seventeen thousand, two hundred and sixty five

In [72]:

```
# Calculate the total cpi_avg_domestic_gross
total_cpi_avg_domestic_gross = genre_summary['cpi_avg_domestic_gross'].sum()

# Calculate the percentage contribution for each genre
genre_summary['percentage_contribution'] = (genre_summary['cpi_avg_domestic_gross'] / total_cpi_avg_dor

# Display the table
percentage_table = genre_summary[['genre', 'percentage_contribution']].sort_values(by='percentage_contr

# Display the updated DataFrame
print('Total number of results:', len(genre_analysis_explode))
display(percentage_table)
```

```
Total number of results: 249658
```

	genre	percentage_contribution
2	Adventure	18.457556
0	Action	16.616130
5	Comedy	13.196172
8	Drama	10.379135
20	Sci-Fi	6.639555
3	Animation	5.787713
24	Thriller	5.091783
10	Fantasy	4.421962
6	Crime	3.764775
9	Family	3.010236
19	Romance	2.904495
13	Horror	2.386000
4	Biography	2.253407
16	Mystery	2.099586
14	Music	0.774296
12	History	0.736607
22	Sport	0.468016
15	Musical	0.338572
7	Documentary	0.252183
26	Western	0.245080
25	War	0.176734
17	News	0.000005
1	Adult	0.000000
18	Reality-TV	0.000000
11	Game-Show	0.000000
21	Short	0.000000

	genre	percentage_contribution
23	Talk-Show	0.000000

It is useful to view the genre by percentages that they contribute to that total to get a sense for how much each genre contributes to the overall domestic gross revenue. For example, Adventure movies account for almost 20% of all reported domestic revenue meaning that they contribute substancially to the overall total domestic gross revenue.

→ In this next section, let's explore how much the top 100 most profitable movies of all time contribute to the total domestic gross revenue by genre.

In [73]:
```
print('Total number of results:', len(top_profit))
display(top_profit.head())
```

```
Total number of results: 100
```

	title	foreign_gross	year	release_date	production_budget	worldwide_gross	avg_domestic_gross	cpi_worldwide_gross	cpi_avg_dome
6814	gone with the wind	NaN	1939	1939-12-15	3900000.0	390525192.0	198680470.0	8.560706e+09	4.35
7293	snow white and the seven dwarfs	NaN	1937	1937-12-21	1488000.0	184925486.0	184925486.0	3.912998e+09	3.91
5928	star wars ep. iv: a new hope	NaN	1977	1977-05-25	11000000.0	786598007.0	460998007.0	3.955082e+09	2.31
7497	bambi	NaN	1942	1942-08-13	858000.0	268000000.0	102797000.0	5.009824e+09	1.92
7073	pinocchio	NaN	1940	1940-02-09	2289247.0	84300000.0	84300000.0	1.834741e+09	1.83

In [74]:
```
# Inner merge 'top_profit' with 'genre_analysis_explode' on 'title', 'year', and 'genre'
top_profit_genre = pd.merge(top_profit, genre_analysis_explode[['title', 'release_year', 'genre']], le

# Display the resulting DataFrame
print('Total number of results:', len(top_profit_genre))
display(top_profit_genre.head())

```

```
Total number of results: 128
```

	title	foreign_gross	year	release_date	production_budget	worldwide_gross	avg_domestic_gross	cpi_worldwide_gross	cpi_avg_domestic
0	gone with the wind	NaN	1939	1939-12-15	3900000.0	390525192.0	198680470.0	8.560706e+09	4.3552
1	snow white and the seven dwarfs	NaN	1937	1937-12-21	1488000.0	184925486.0	184925486.0	3.912998e+09	3.9129
2	star wars ep. iv: a new hope	NaN	1977	1977-05-25	11000000.0	786598007.0	460998007.0	3.955082e+09	2.3179
3	bambi	NaN	1942	1942-08-13	858000.0	268000000.0	102797000.0	5.009824e+09	1.9216
4	pinocchio	NaN	1940	1940-02-09	2289247.0	84300000.0	84300000.0	1.834741e+09	1.8347

In [75]:
```
# Why are there more than 100 entries in the top_profit_genre df?
# Consider unuique titles
unique_titles_count = top_profit_genre['title'].nunique()
unique_titles_count
```

Out[75]: 99

In [76]:

```
# Consider duplicate titles
duplicate_titles = top_profit[top_profit['title'].duplicated(keep=False)]
display(duplicate_titles)
```

	title	foreign_gross	year	release_date	production_budget	worldwide_gross	avg_domestic_gross	cpi_worldwide_gross	cpi_avg_domesti
5165	beauty and the beast	NaN	1991	1991-11-13	20000000.0	6.084311e+08	376057266.0	1.361161e+09	8.4130
3439	beauty and the beast	NaN	2017	2017-03-17	160000000.0	1.259200e+09	504014165.0	1.565277e+09	6.2652

This shows 99 unique titles in the top_profit_genre dataframe because some titles such as Beauty and the Beast (release years: 1991 and 2017) were top profit movies in multiple release years. This and the exploded genre count account for the 128 values shown in the top_profit_genre dataframe.

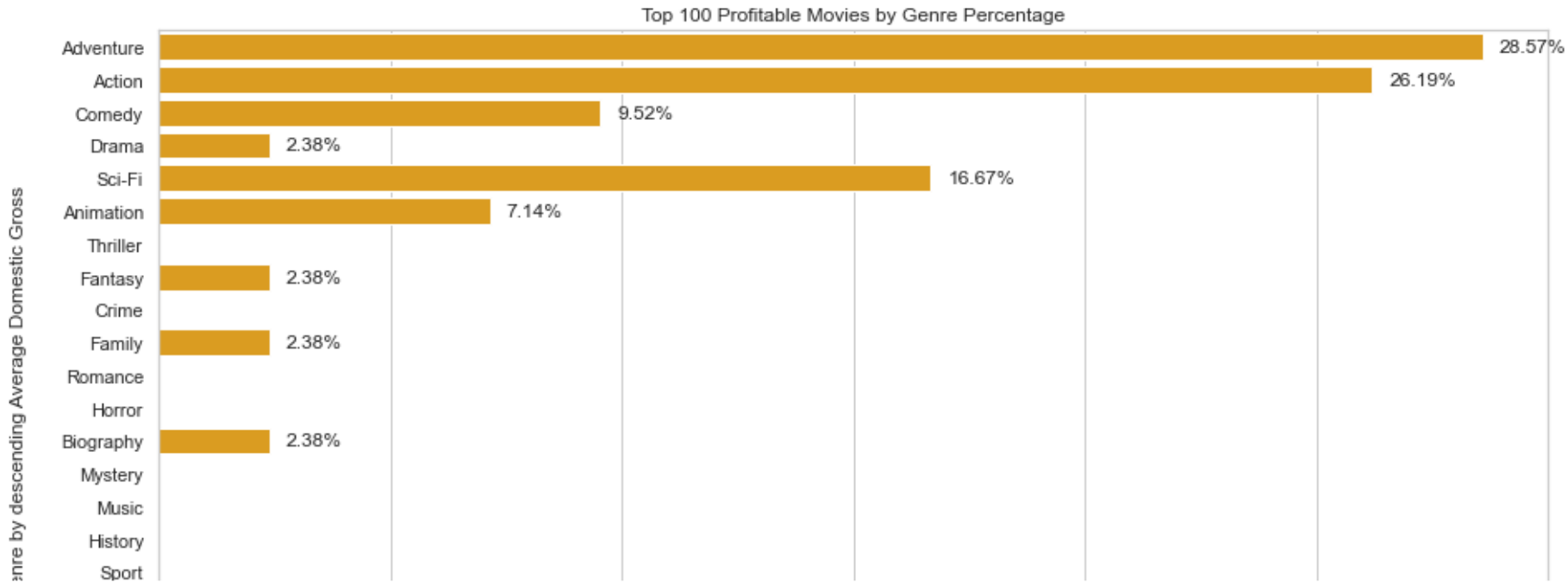
In [77]:

```
# Calculate the total number of movies
total_movies = top_profit_genre.shape[0]

# Calculate the percentage of movies for each genre
genre_percentage = top_profit_genre['genre'].value_counts(normalize=True) * 100

# Display the resulting table
print('Total number of results:', len(genre_percentage))
display(genre_percentage)
```

```
Total number of results: 10

Adventure    28.571429
Action       26.190476
Sci-Fi       16.666667
Comedy        9.523810
Animation     7.142857
Drama         2.380952
Family        2.380952
Biography     2.380952
Fantasy       2.380952
Musical       2.380952
Name: genre, dtype: float64
```

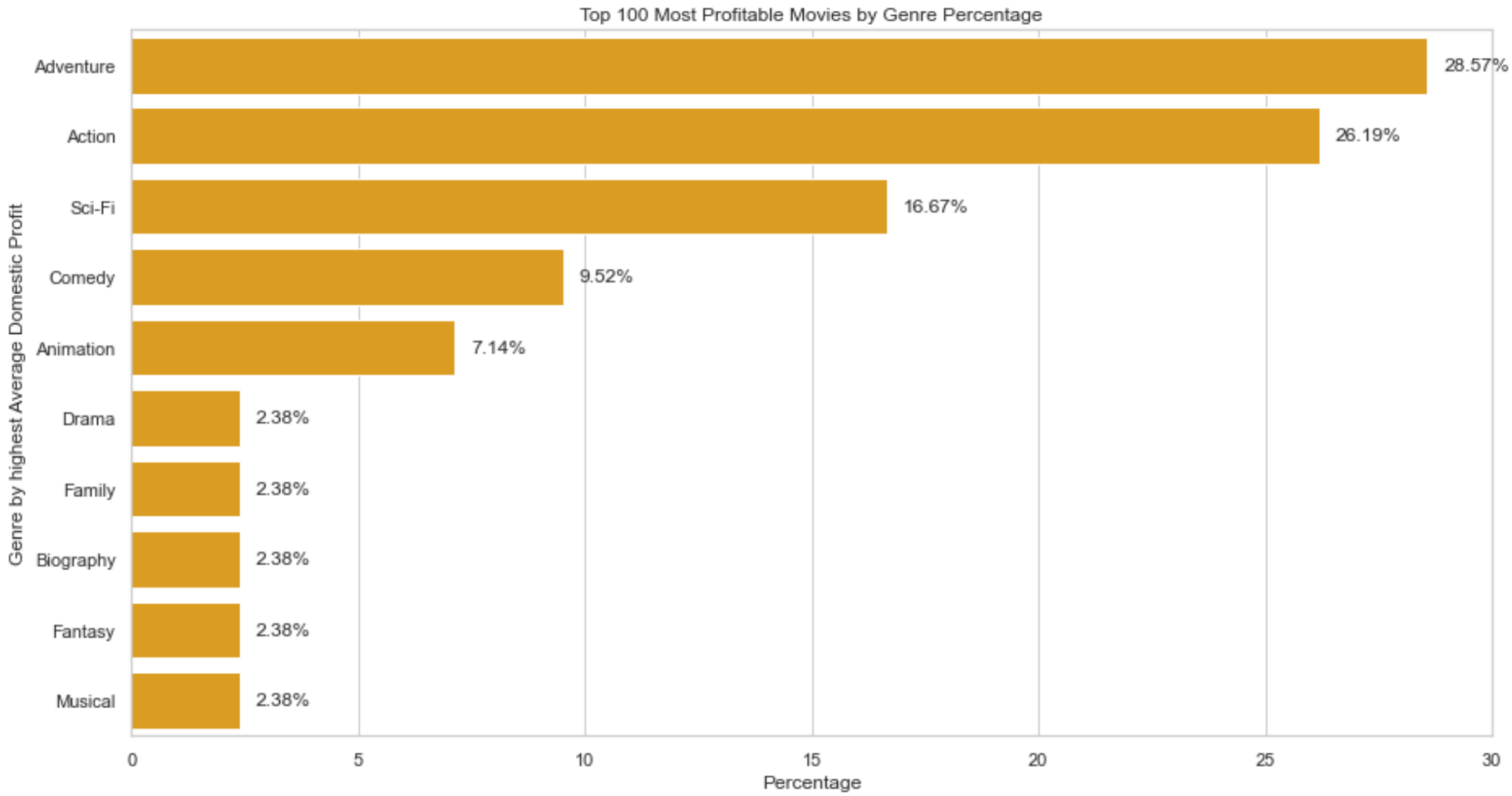
In [78]:

```
# Convert the genre_percentage Series to a DataFrame
genre_percentage_df = genre_percentage.reset_index()
genre_percentage_df.columns = ['genre', 'percentage']


# Define the order of genres
genre_order = ['Adventure', 'Action', 'Comedy', 'Drama', 'Sci-Fi', 'Animation', 'Thriller', 'Fantasy',
               'Family', 'Romance', 'Horror', 'Biography', 'Mystery', 'Music', 'History', 'Sport', 'Mus
               'Documentary', 'Western', 'War', 'News']

# Set the style
sns.set(style="whitegrid")

# Create a horizontal bar graph with specified order
plt.figure(figsize=(15, 8))
bar_plot = sns.barplot(x='percentage', y='genre', data=genre_percentage_df, order=genre_order, color='#

# Add percentages on the right of each bar
for p in bar_plot.patches:
    bar_plot.annotate(f'{p.get_width():.2f}%', (p.get_width(), p.get_y() + p.get_height() / 2.),
                      ha='left', va='center', xytext=(10, 0), textcoords='offset points')

# Set plot title and labels
plt.title('Top 100 Profitable Movies by Genre Percentage')
plt.xlabel('Percentage')
plt.ylabel('Genre by descending Average Domestic Gross')

# Show the plot
plt.show()
```


Top 100 Profitable Movies by Genre Percentage
nre by descending Average Domestic Gross
Adventure
Action
Comedy
Drama
Sci-Fi
Animation
Thriller
Fantasy
Crime
Family
Romance
Horror
Biography
Mystery
Music
History
Sport
28.57%
26.19%
9.52%
2.38%
16.67%
7.14%
2.38%
2.38%
2.38%

In [79]:

```
# Set the style
sns.set(style="whitegrid")

# Create a horizontal bar graph
plt.figure(figsize=(15, 8))
bar_plot = sns.barplot(x='percentage', y='genre', data=genre_percentage_df, color='#F9A602')

# Add percentages on the right of each bar
for p in bar_plot.patches:
    bar_plot.annotate(f'{p.get_width():.2f}%', (p.get_width(), p.get_y() + p.get_height() / 2.),
                      ha='left', va='center', xytext=(10, 0), textcoords='offset points')

# Set plot title and labels
plt.title('Top 100 Most Profitable Movies by Genre Percentage')
plt.xlabel('Percentage')
plt.ylabel('Genre by highest Average Domestic Profit')

# Show the plot
plt.show()
```



About 64% of top 100 most profitable movies of all time fall into the top three, all-time grossing genres of Action, Adventure, and Comedy.

In [80]:

```
# Define the custom colors
background_color = '#FFFFFF'  # White background
grid_line_color = '#d3d3d3'
bubble_color = '#FCC252'
text_color = '#000000'
outline_color = '#eaeaea'

# Creating the bubble plot using Plotly with custom colors
fig = px.scatter(genre_summary, x='genre', y='cpi_avg_domestic_gross', size='cpi_avg_domestic_gross',
                 hover_name='genre', size_max=50, color='genre', color_discrete_sequence=[bubble_color]

# Set plot characteristics
fig.update_layout(title='Bubble Plot: Total Average Domestic Gross by Genre',
                  xaxis_title='Genre',
                  yaxis_title='Total Average Domestic Gross',
                  plot_bgcolor=background_color,
                  paper_bgcolor=background_color,
                  xaxis=dict(showgrid=True, gridcolor=grid_line_color, linecolor=outline_color),
                  yaxis=dict(showgrid=True, gridcolor=grid_line_color, linecolor=outline_color),
                  font=dict(family="Times New Roman",color=text_color),
                  height=800,
                  width=1000)

# Add borders around the plot
fig.update_xaxes(showline=True, linewidth=1, linecolor=outline_color, mirror=True)
fig.update_yaxes(showline=True, linewidth=1, linecolor=outline_color, mirror=True)

# Show plot
fig.show()

```

Bubble Plot: Total Average Domestic Gross by Genre

Genre

Genre Analysis Summary

The analysis of the most profitable movie genres reveals trends into the domestic gross earnings. Adventure emerges as the most financially successful genre, followed by Action and Comedy. Some genres, such as Talk-Show, Short, Reality-TV, Adult, and Game-Show, show zero values in the adjusted average domestic gross. This can be attributed to the integration of movie genre data from one dataset and financial data from another, leading to instances of missing financial information for specific genres.

Considering the top 100 most profitable movies, 64% of titles fall into the the overall top grossing genres of Action, Adventure, and Comedy.

Additionally, it's essential to recognize the potential impact of mixed-genre films on the dataset. Some movies may belong to more than one genre, introducing complexity and potential ambiguity in attributing financial success solely to a specific genre.

Considering these limitations, future analyses could explore the correlation between mixed-genre films and financial performance.

Release Date Analysis

The month that a movie is released may greatly influence profitability so when is the most profitable time to release a movie? By investigating the release month of a movie and referencing its adjusted average domestic gross, it can be determined if there is a more profitable time to release movies.

In [81]:

```
# Create a copy of the DataFrame to avoid altering the original
release_date_analysis = main_df_clean.copy()

# Select specific columns to create a new DataFrame
selected_columns = ['movie_id', 'release_year', 'title', 'release_date', 'cpi_avg_domestic_gross']

# Create a new DataFrame with only the selected columns
release_date_summary = release_date_analysis[selected_columns].copy()

# Create a release_month column
release_date_summary['release_month']=pd.DatetimeIndex(release_date_summary['release_date']).month

# Display the new DataFrame
print('Total number of results:', len(release_date_summary))
display(release_date_summary.head())
```

```
Total number of results: 161521
```

	movie_id	release_year	title	release_date	cpi_avg_domestic_gross	release_month
0	tt0063540	2013.0	sunghursh	NaN	NaN	NaN
1	tt0066787	2019.0	one day before the rainy season	NaN	NaN	NaN
2	tt0069204	2018.0	sabse bada sukh	NaN	NaN	NaN
3	tt0100275	2017.0	the wandering soap opera	NaN	NaN	NaN
4	tt0111414	2018.0	a thin life	NaN	NaN	NaN

This function will useful to create subsequent multiple visualizations on this data. Visualizing the number of movies released each month and the adjusted average domestic gross for each month will provide insight on the best range of dates to release a movie.

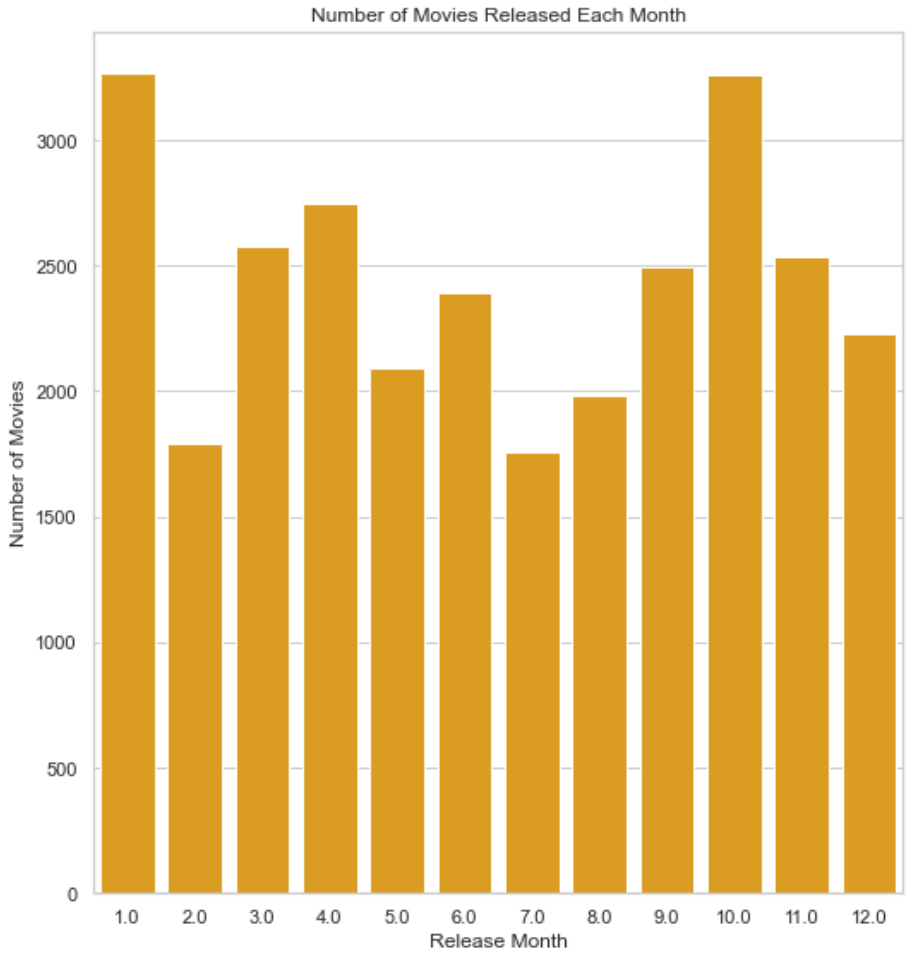
In [82]:

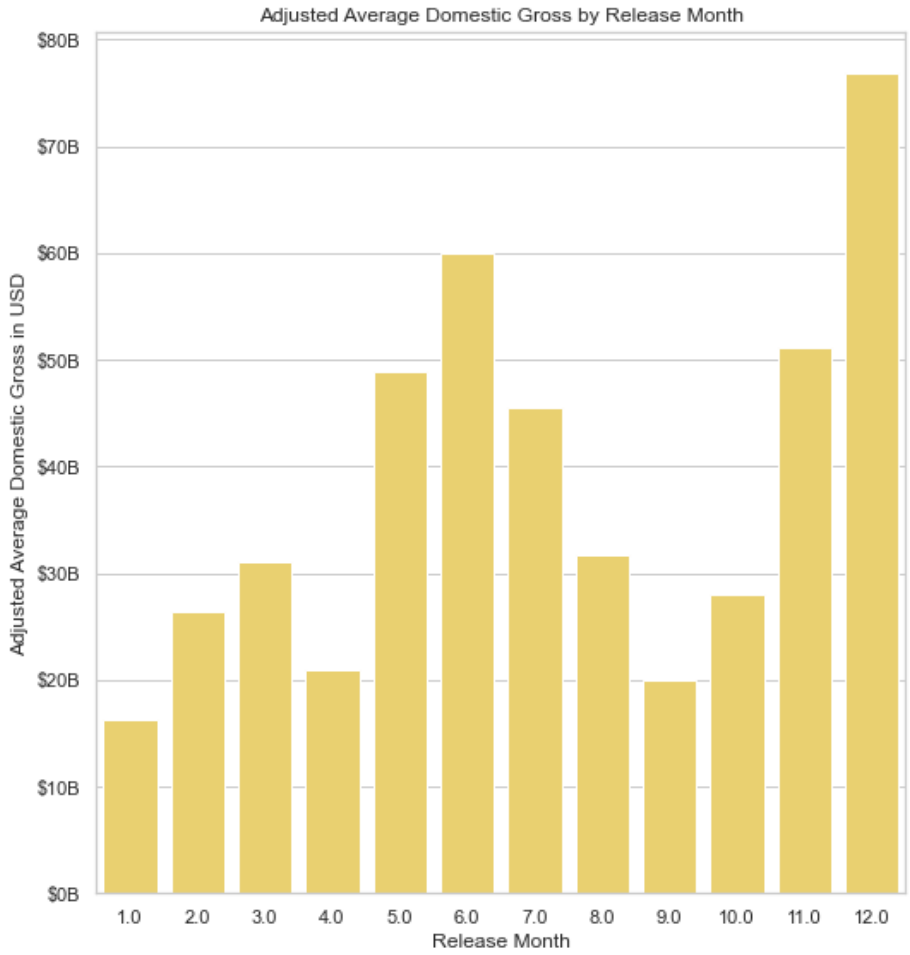
```
def billions_formatter(x, pos):
    '''
    Function to format y-axis labels to display in billions.

    Input:
    x : Value to be formatted
    pos : Position of the tick on the axis

    Output:
    formatted_string : Formatted string representing the value in billions
    '''
    return f'${x/1e9:.0f}B'

def release_month_visual(df):
    '''
    Visualizes movie release patterns based on months.

    Input:
    df : Pandas dataframe containing information about movies

    Output:
    None (Displays a matplotlib figure)
    '''
# Create a figure with 1 row and 2 columns
    fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(15, 8))
# Create a count plot for the number of movies released each month
    sns.countplot(x='release_month', color='#F9A602', data=df, ax=ax1)
    ax1.set_xlabel('Release Month')
    ax1.set_ylabel('Number of Movies')
    ax1.set_title('Number of Movies Released Each Month')
# Create a bar plot for the avg domestic gross by release month
    sns.barplot(x='release_month', y='cpi_avg_domestic_gross', estimator=sum, ci=None, color='#FDDC5C',
    ax2.set_xlabel('Release Month')
    ax2.set_ylabel('Adjusted Average Domestic Gross in USD')
    ax2.set_title('Adjusted Average Domestic Gross by Release Month')

    # Adjusting the formatting of the y-axis for the second plot
    formatter = FuncFormatter(billions_formatter)
    ax2.yaxis.set_major_formatter(formatter)

    plt.tight_layout()
# Show the plot
```

```
43      plt.show()
```

In [83]:

```
1  # Run the function on release_date_summary
2  release_month_visual(release_date_summary)
```





It is shown here that the most popular months for releasing movies are January, March, April, October, and November. The months where the average domestic gross is highest are May, June, July, November, and December.

In [84]:
```
# View top_profit dataframe
print('Total number of results:', len(top_profit))
display(top_profit.head())
```

```
Total number of results: 100
```

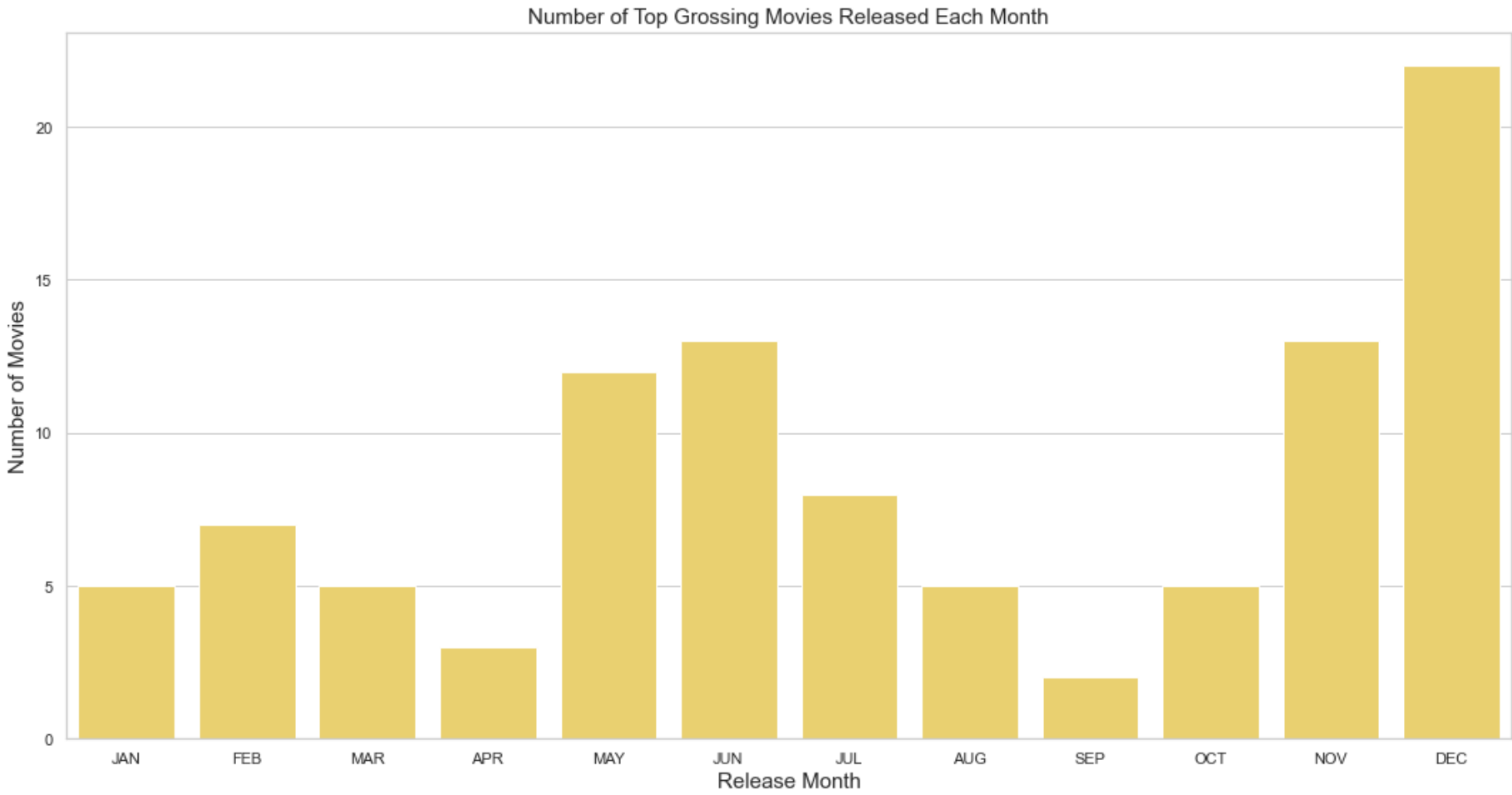
	title	foreign_gross	year	release_date	production_budget	worldwide_gross	avg_domestic_gross	cpi_worldwide_gross	cpi_avg_dome
6814	gone with the wind	NaN	1939	1939-12-15	3900000.0	390525192.0	198680470.0	8.560706e+09	4.3
7293	snow white and the seven dwarfs	NaN	1937	1937-12-21	1488000.0	184925486.0	184925486.0	3.912998e+09	3.9
5928	star wars ep. iv: a new hope	NaN	1977	1977-05-25	11000000.0	786598007.0	460998007.0	3.955082e+09	2.3
7497	bambi	NaN	1942	1942-08-13	858000.0	268000000.0	102797000.0	5.009824e+09	1.9
7073	pinocchio	NaN	1940	1940-02-09	2289247.0	84300000.0	84300000.0	1.834741e+09	1.8

In [85]:
```
# Create a release_month column
top_profit['release_month']=pd.DatetimeIndex(top_profit['release_date']).month
```

In [86]:

```python
def release_month_visual_tg(df):
    '''
    Visualizes the number of top-grossing movies released each month.

    Input:
    df : Pandas dataframe containing information about movies

    Output:
    None (Displays a matplotlib figure)
    '''
    # Create a Counter of months from the available data
    months_counter = Counter(df['release_month'])
    # Define all months you want to display
    all_months = ['JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL', 'AUG', 'SEP', 'OCT', 'NOV', 'DEC']

    # Create counts for all months, filling missing months with zero counts
    counts_per_month = {month: months_counter.get(index + 1, 0) for index, month in enumerate(all_month
    plt.figure(figsize=(15, 8))
    ax = sns.barplot(x=list(counts_per_month.keys()), y=list(counts_per_month.values()),
                     color='#FDDC5C', label='Number of Movies', ci=None)

    # Set positions and labels for x-axis ticks
    ax.set_xticks(range(len(all_months)))
    ax.set_xticklabels(all_months)

    ax.set_xlabel('Release Month', fontfamily='Arial', fontsize=15)
    ax.set_ylabel('Number of Movies', fontfamily='Arial', fontsize=15)
    ax.set_title('Number of Top Grossing Movies Released Each Month', fontfamily='Arial', fontsize=15)


    plt.tight_layout()

    # Show the plot
    plt.show()
```

In [87]:
```
# Run the function on top_profit
release_month_visual_tg(top_profit)
```



This visual somewhat agrees with what was shown previously in that May, June, July, and November/December are the best months to release movies.

In [88]:
```
# Format 'release_date' in the original DataFrame
release_date_summary['release_date'] = pd.to_datetime(release_date_summary['release_date'], errors='coe

# Create a new column 'day_of_week' in the original DataFrame
release_date_summary['day_of_week'] = release_date_summary['release_date'].dt.day_name()
```

In [89]:
```
# Count occurrences of each day of the week
day_counts = release_date_summary['day_of_week'].value_counts()

# Find the most popular day
most_popular_day = day_counts.idxmax()

# Display the results
print("Counts of each day of the week:")
print(day_counts)

print("\nThe most popular day to release a movie is:", most_popular_day)

```

```
Counts of each day of the week:
Friday       11585
Tuesday       4366
Saturday      3483
Thursday      2664
Wednesday     2589
Sunday        2495
Monday        1949
Name: day_of_week, dtype: int64

The most popular day to release a movie is: Friday
```

This list shows that the most popular day of the week to release a movie is Friday.

Release Date Summary

This analysis highlights a relationship between the release date and the success of top-grossing movies. Caution must be used here because additional factors may influence a movie's overall profit such as the economy, viewing platform trends, distribution strategy (wide vs limited release), advertising budget, any award recognition, and the movie's enduring appeal over time.

A comprehensive analysis that incorporates specific factors expressedly communicated by the film company as priorities based on the company staff strengths is essential for a nuanced understanding of the dynamics influencing the financial success of its movies is needed to

Runtime Analysis

Runtime can impact the type of audience that views the movie. Kid and family films may tend to be shorter while action movies may be longer. Depending on the intended audience, runtime may impact the financial success of the film.

In [90]:
```
# Create a copy of the DataFrame to avoid altering the original
runtime_analysis = main_df_clean.copy()

# Select specific columns to create a new DataFrame
selected_columns = ['title', 'runtime_minutes', 'cpi_avg_domestic_gross', 'genres', 'release_year']

# Create a new DataFrame with only the selected columns
runtime_summary = runtime_analysis[selected_columns].copy()

# Display the new DataFrame
runtime_summary.head()
```

Out[90]:

	title	runtime_minutes	cpi_avg_domestic_gross	genres	release_year
0	sunghursh	175.0	NaN	Action,Crime,Drama	2013.0
1	one day before the rainy season	114.0	NaN	Biography,Drama	2019.0
2	sabse bada sukh	NaN	NaN	Comedy,Drama	2018.0
3	the wandering soap opera	80.0	NaN	Comedy,Drama,Fantasy	2017.0
4	a thin life	75.0	NaN	Comedy	2018.0

In [91]:
```
# Get statistics for movie runtime
runtime_summary[['title', 'runtime_minutes']].describe()
```

Out[91]:

	runtime_minutes
count	113046.000000
mean	86.221193
std	167.322817
min	1.000000
25%	70.000000
50%	87.000000
75%	99.000000
max	51420.000000

→ There appears to be an outlier because the standard deviation is huge. There potentially aren't too many outliers because the mean and median are close together, so let's take a closer look.

In [92]:
```
# Order the runtime minutes by longest to shortest
print('Total number of results:', len(runtime_summary.sort_values(by='runtime_minutes', ascending=False
display((runtime_summary.sort_values(by='runtime_minutes', ascending=False)).head())
```

```
Total number of results: 161521
```

	title	runtime_minutes	cpi_avg_domestic_gross	genres	release_year
116541	logistics	51420.0	NaN	Documentary	2012.0
37951	modern times forever	14400.0	NaN	Documentary	2011.0
108172	nari	6017.0	NaN	Documentary	2017.0
75288	hunger!	6000.0	NaN	Documentary,Drama	2015.0
76594	london ec1	5460.0	NaN	Comedy,Drama,Mystery	2015.0

In [93]:
```
# Make a variable to take a closer look at the outliers
sorted_runtime = runtime_summary.sort_values(by='runtime_minutes', ascending=False)

# Run a conditional statement to view some of the outliers
print(len(sorted_runtime[sorted_runtime['runtime_minutes']>180]))
display((sorted_runtime[sorted_runtime['runtime_minutes']>180]).head())
```

532

	title	runtime_minutes	cpi_avg_domestic_gross	genres	release_year
116541	logistics	51420.0	NaN	Documentary	2012.0
37951	modern times forever	14400.0	NaN	Documentary	2011.0
108172	nari	6017.0	NaN	Documentary	2017.0
75288	hunger!	6000.0	NaN	Documentary,Drama	2015.0
76594	london ec1	5460.0	NaN	Comedy,Drama,Mystery	2015.0

→ After some internet fact checking, some of the runtime_minutes reported are false. A movie of 300 minutes would mean five hours which is a tremendous underataking for production. So considering that the mean statistics are skewed by these outliers, using the 95% quantile seems like a reasonable cut off for reliably reported film runtimes.

Additionally, according to the Academy of Motion Picture Arts and Sciences, the American Film Institute and the British Film Institute, a feature film runs for more than 40 minutes.

In [94]:
```
# Calculate the 5th and 95th percentiles
percentile_5 = runtime_summary['runtime_minutes'].quantile(0.05)
percentile_95 = runtime_summary['runtime_minutes'].quantile(0.95)

# Filter the DataFrame based on the percentiles
runtime_summary_inner_quantile = runtime_summary[((runtime_summary['runtime_minutes'] >= percentile_5)

```

In [95]:
```
1 runtime_summary_inner_quantile.describe()
```

Out[95]:

	runtime_minutes	cpi_avg_domestic_gross	release_year
count	101902.000000	7.014000e+03	150377.000000
mean	84.811927	6.322840e+07	2014.144763
std	19.605914	1.419309e+08	4.222397
min	44.000000	1.307975e+02	1915.000000
25%	72.000000	6.205125e+05	2012.000000
50%	87.000000	1.768768e+07	2015.000000
75%	97.000000	7.225980e+07	2017.000000
max	130.000000	4.355276e+09	2115.000000

→ Looking at the inner quantile, these values make much more sense.

Comparing the two measurements of spread, the standard deviation and the inter quartile range, before we removed the outliers the inter quartile range was 29 to a standard deviation of 169. After removing the outliers the inter quartile range is 26 to a standard deviation of 20.

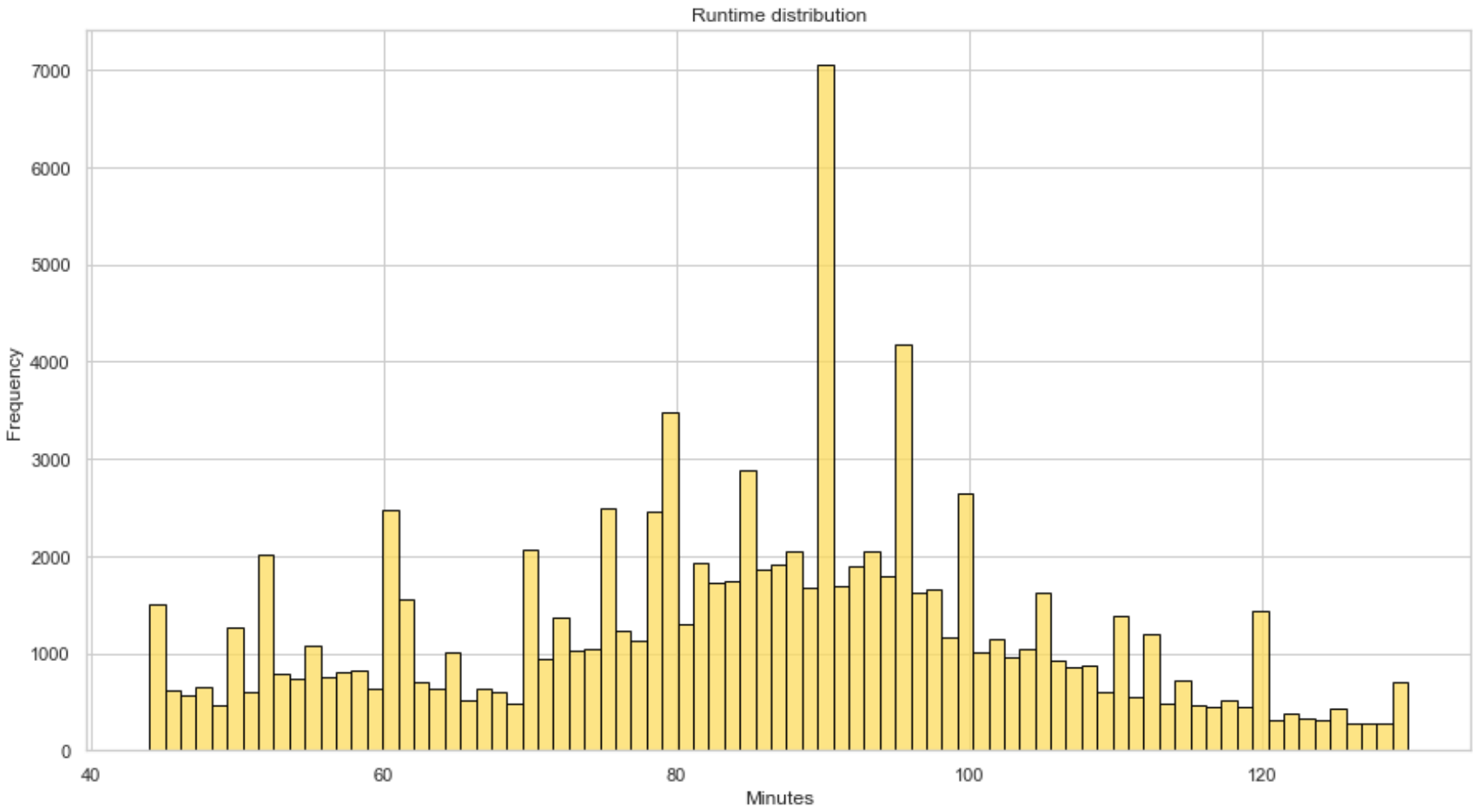
This is excellent because for a perfect normal distribution the inter quartile and the standard deviation have a ratio of 1.35.

The inter quartile range is 26 to a standard deviation of 20 and the ratio here is 1.3. So after the removal of the outliers this data can confidently be considered normally distributed.

In [96]:

```python
# Define function to plot

def runtime_visual(df, title, plot_range=None, bar_color='#FDDC5C', edge_color='black'):
    '''
    Creates a Seaborn histplot of the 'runtime_minutes' column within a specified range.

    Input:
    df : Pandas dataframe containing information about movies
    title: A string representing the title of the plot
    plot_range: Optional tuple specifying the range of runtimes to plot (e.g., (min_runtime, max_runtim

    Output:
    None (Displays a matplotlib figure)
    '''
    plt.figure(figsize=(15, 8))

    if plot_range:
        # Filter data within the specified range
        df_filtered = df[(df['runtime_minutes'] >= plot_range[0]) & (df['runtime_minutes'] <= plot_rang
        sns.histplot(df_filtered['runtime_minutes'], kde=False, color=bar_color, edgecolor=edge_color)
        plt.title(f"{title} (Within {plot_range[0]} to {plot_range[1]} Minutes)")
    else:
        sns.histplot(df['runtime_minutes'], kde=False, color=bar_color, edgecolor=edge_color)
        plt.title(title)

    plt.xlabel('Minutes')
    plt.ylabel('Frequency')
    return plt.show()

```

In [97]:
```
# Call runtime_visual function on movies DataFrame
runtime_visual(runtime_summary_inner_quantile, 'Runtime distribution')

```



The median is approximately 87 minutes.

In [98]:
```
# Get number of missing values per column
runtime_summary_inner_quantile.isna().sum()
```

Out[98]:
```
title                         0
runtime_minutes           48475
cpi_avg_domestic_gross   143363
genres                    22519
release_year                  0
dtype: int64
```

In [99]:
```
# Percent of missing values for runtime
(runtime_summary_inner_quantile['runtime_minutes'].isna().sum()/len(runtime_summary_inner_quantile)*100
```

Out[99]:
```
32.24
```

→ This is significant because runtime is essential in this part of the analysis. The missing values will have to be filled.

In [100]:
```
# Consider mean
runtime_summary_inner_quantile['runtime_minutes'].mean().round(2)
```

Out[100]:
```
84.81
```

In [101]:
```
# Consider median
runtime_summary_inner_quantile['runtime_minutes'].median().round(2)
```

Out[101]:
```
87.0
```

→ It is good that both values are about the same. Missing values will be filled with the median of 87 minutes because it is a better option as it is less sensitive to outliers.

In [102]:
```
# Make a copy to convert from a view to a data frame
runtime_summary_inner_quantile = runtime_summary_inner_quantile.copy()

# Fill missing values with median
runtime_summary_inner_quantile['runtime_minutes'] = runtime_summary_inner_quantile['runtime_minutes'].
```

Which runtime range receives the highest mean revenue?

Runtime will be classified into different ranges based on the output from the describe method as shown here:

Runtime(in minutes)	Range
45-65	Short
65-85	Medium
85-105	Long
105-125	Very Long

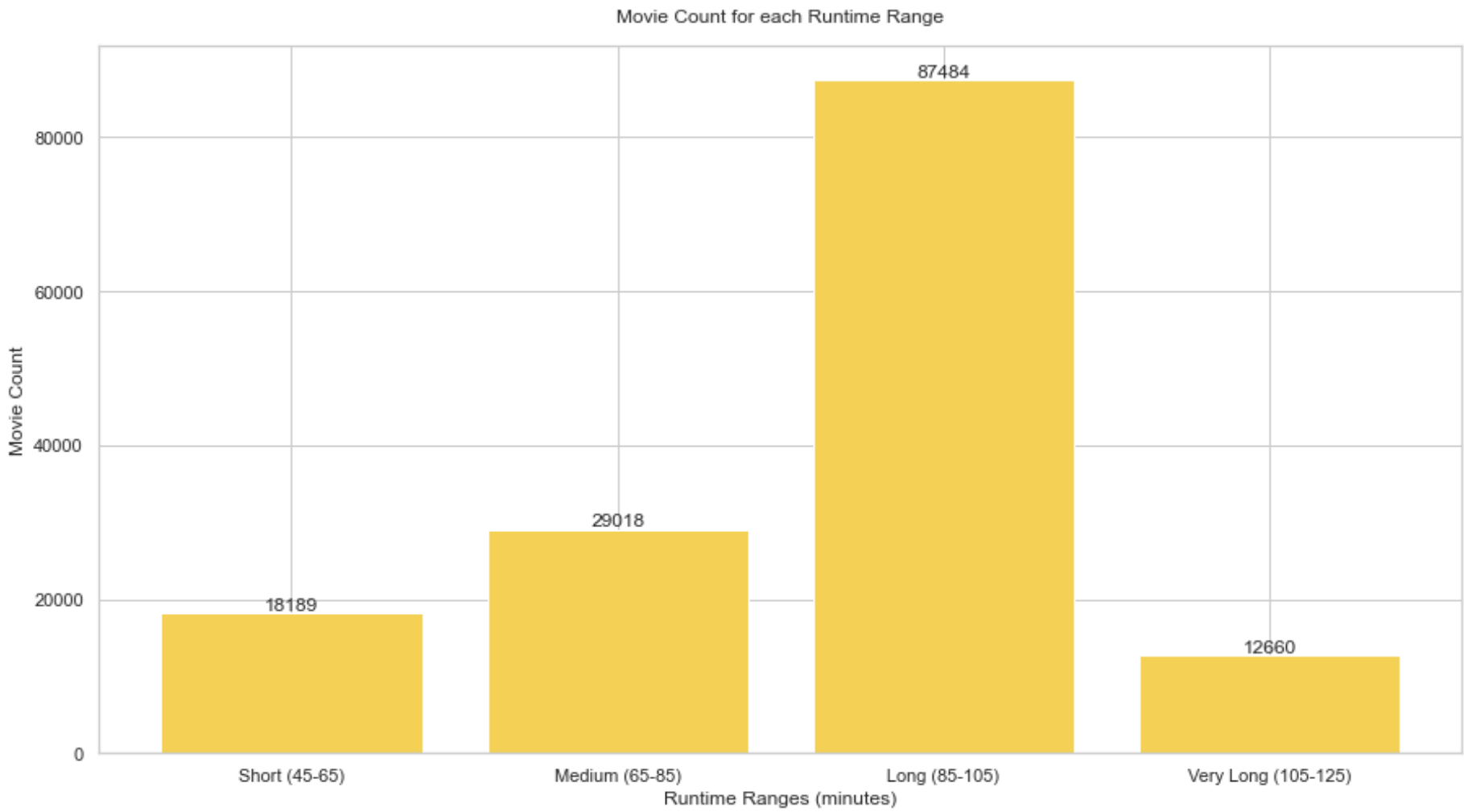
In [103]:
```
# Use the Pandas cut method to map the Runtime into the above ranges

# Define cutoff for each bin
runtime_bin_edges = [45, 65, 85, 105, 125]

# Name each bin
runtime_bin_names = ['Short (45-65)', 'Medium (65-85)', 'Long (85-105)', 'Very Long (105-125)']

# Make column for different ranges into which the values of the 'runtime_minutes' column are divided
runtime_summary_inner_quantile['runtime_range'] = pd.cut(runtime_summary_inner_quantile.runtime_minutes

#Display df
print(len(runtime_summary_inner_quantile))
display(runtime_summary_inner_quantile.head())

```

```
150377
```

	title	runtime_minutes	cpi_avg_domestic_gross	genres	release_year	runtime_range
1	one day before the rainy season	114.0	NaN	Biography,Drama	2019.0	Very Long (105-125)
2	sabse bada sukh	87.0	NaN	Comedy,Drama	2018.0	Long (85-105)
3	the wandering soap opera	80.0	NaN	Comedy,Drama,Fantasy	2017.0	Medium (65-85)
4	a thin life	75.0	NaN	Comedy	2018.0	Medium (65-85)
5	bigfoot	87.0	NaN	Horror,Thriller	2017.0	Long (85-105)

In [104]:

```python
def runtime_bin_visual(df):
    '''
    Creates a bar plot of movie counts for each runtime range based on the 'runtime_range' column.

    Input:
    df : Pandas dataframe containing information about movies with a 'runtime_range' column

    Output:
    None (Displays a matplotlib figure)
    '''
    # Create a Counter of runtime ranges from the available data
    bins_counter = Counter(df['runtime_range'])

    # Define all bins to display
    all_bins = ['Short (45-65)', 'Medium (65-85)', 'Long (85-105)', 'Very Long (105-125)']

    # Create counts for all bins, filling missing bins with zero counts
    counts_per_bin = {bins: bins_counter.get(bins, 0) for bins in all_bins}

    # Plot the count of different runtime ranges
    plt.figure(figsize=(15, 8))
    ax = plt.bar(counts_per_bin.keys(), counts_per_bin.values(), color='#F4D054')

    plt.ylabel('Movie Count')
    plt.xlabel('Runtime Ranges (minutes)')
    plt.title('Movie Count for each Runtime Range', y=1.02)

    # Show the counts on top of the bars
    for i, v in enumerate(counts_per_bin.values()):
        plt.text(i, v + 0.1, str(v), ha='center', va='bottom')


    plt.show()

# View dataframe
runtime_bin_visual(runtime_summary_inner_quantile)

```



→ Let's see if this pattern follows for the top grossing movies.

In [105]:

```
# Inner merge 'top_profit' with 'runtime_summary_inner_quantile' on 'title' and 'year'
top_profit_runtime = pd.merge(top_profit, runtime_summary_inner_quantile[['title', 'release_year']], le

# Display the resulting DataFrame
print(len((top_profit_runtime)))
display((top_profit_runtime).head())

```

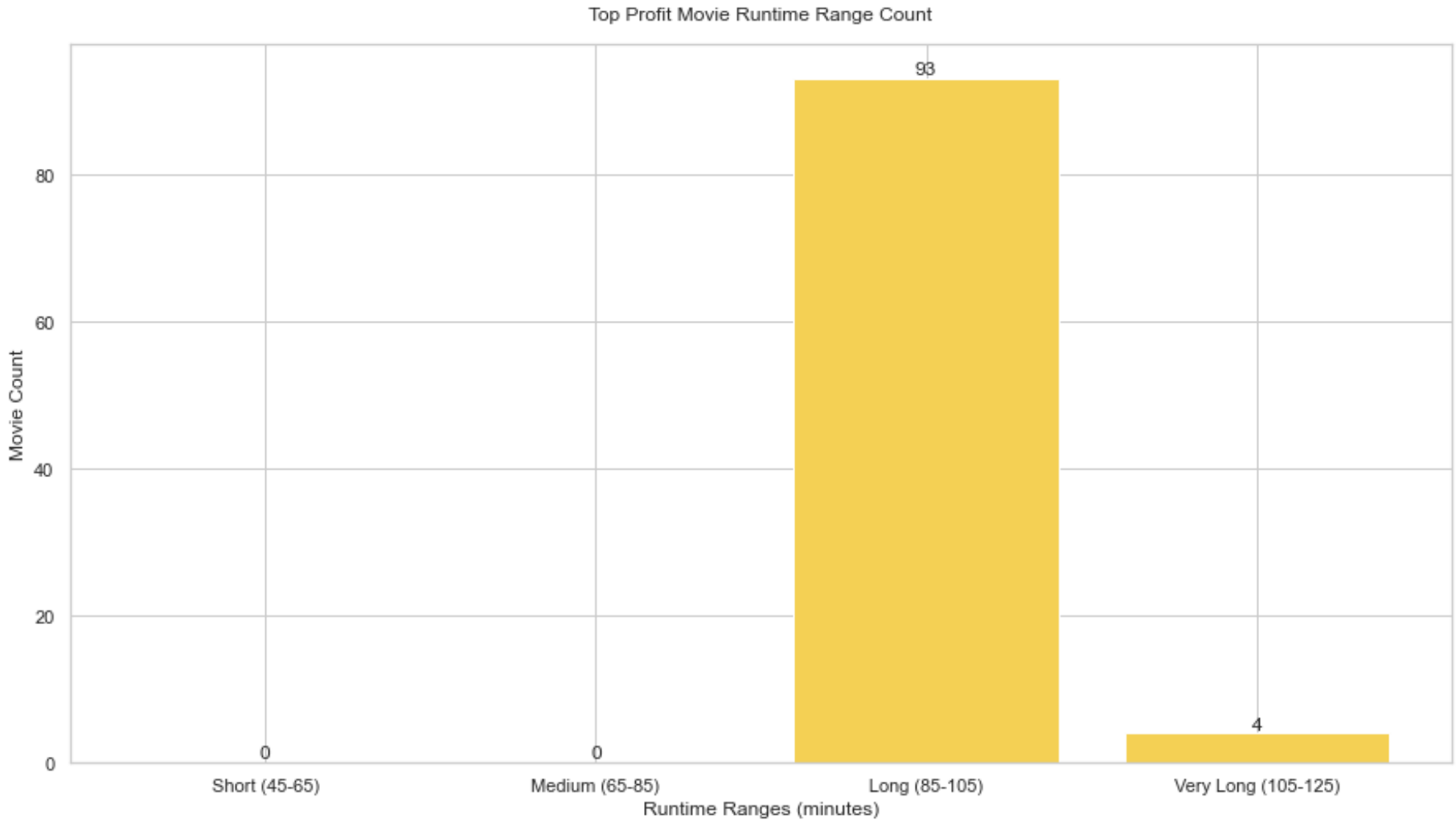
93

	title	foreign_gross	year	release_date	production_budget	worldwide_gross	avg_domestic_gross	cpi_worldwide_gross	cpi_avg_domestic
0	gone with the wind	NaN	1939	1939-12-15	3900000.0	390525192.0	198680470.0	8.560706e+09	4.3552
1	snow white and the seven dwarfs	NaN	1937	1937-12-21	1488000.0	184925486.0	184925486.0	3.912998e+09	3.9129
2	star wars ep. iv: a new hope	NaN	1977	1977-05-25	11000000.0	786598007.0	460998007.0	3.955082e+09	2.3179
3	bambi	NaN	1942	1942-08-13	858000.0	268000000.0	102797000.0	5.009824e+09	1.9216
4	pinocchio	NaN	1940	1940-02-09	2289247.0	84300000.0	84300000.0	1.834741e+09	1.8347

→ The top_profit_runtime is only showing 93/100 entries because some of the entries do not contain runtimes.

In [106]:

```python
# Set top_profit_runtime as top 100 movies
top_profit_runtime = runtime_summary_inner_quantile.sort_values('cpi_avg_domestic_gross', ascending = F

def runtime_bin_visual_tg(df):
    '''
    Creates a bar plot of movie counts for each runtime range based on the 'runtime_range' column.

    Input:
    df : Pandas dataframe containing information about top-profit movies with a 'runtime_range' column

    Output:
    None (Displays a matplotlib figure)
    '''
    # Create a Counter of runtime ranges from the available data
    bins_counter = Counter(df['runtime_range'])

    # Define all bins to display
    all_bins = ['Short (45-65)', 'Medium (65-85)', 'Long (85-105)', 'Very Long (105-125)']

    # Create counts for all bins, filling missing bins with zero counts
    counts_per_bin = {bins: bins_counter.get(bins, 0) for bins in all_bins}

    # Plot the count of different runtime ranges
    plt.figure(figsize=(15, 8))
    ax = plt.bar(counts_per_bin.keys(), counts_per_bin.values(), color='#F4D054')

    plt.ylabel('Movie Count')
    plt.xlabel('Runtime Ranges (minutes)')
    plt.title('Top Profit Movie Runtime Range Count', y=1.02)

    # Show the counts on top of the bars
    for i, v in enumerate(counts_per_bin.values()):
        plt.text(i, v + 0.1, str(v), ha='center', va='bottom')

    plt.show()

# View dataframe
runtime_bin_visual_tg(top_profit_runtime)

```


Top Profit Movie Runtime Range Count
Movie Count
0
20
40
60
80
0
0
93
4
Short (45-65)
Medium (65-85)
Long (85-105)
Very Long (105-125)
Runtime Ranges (minutes)

In [107]:

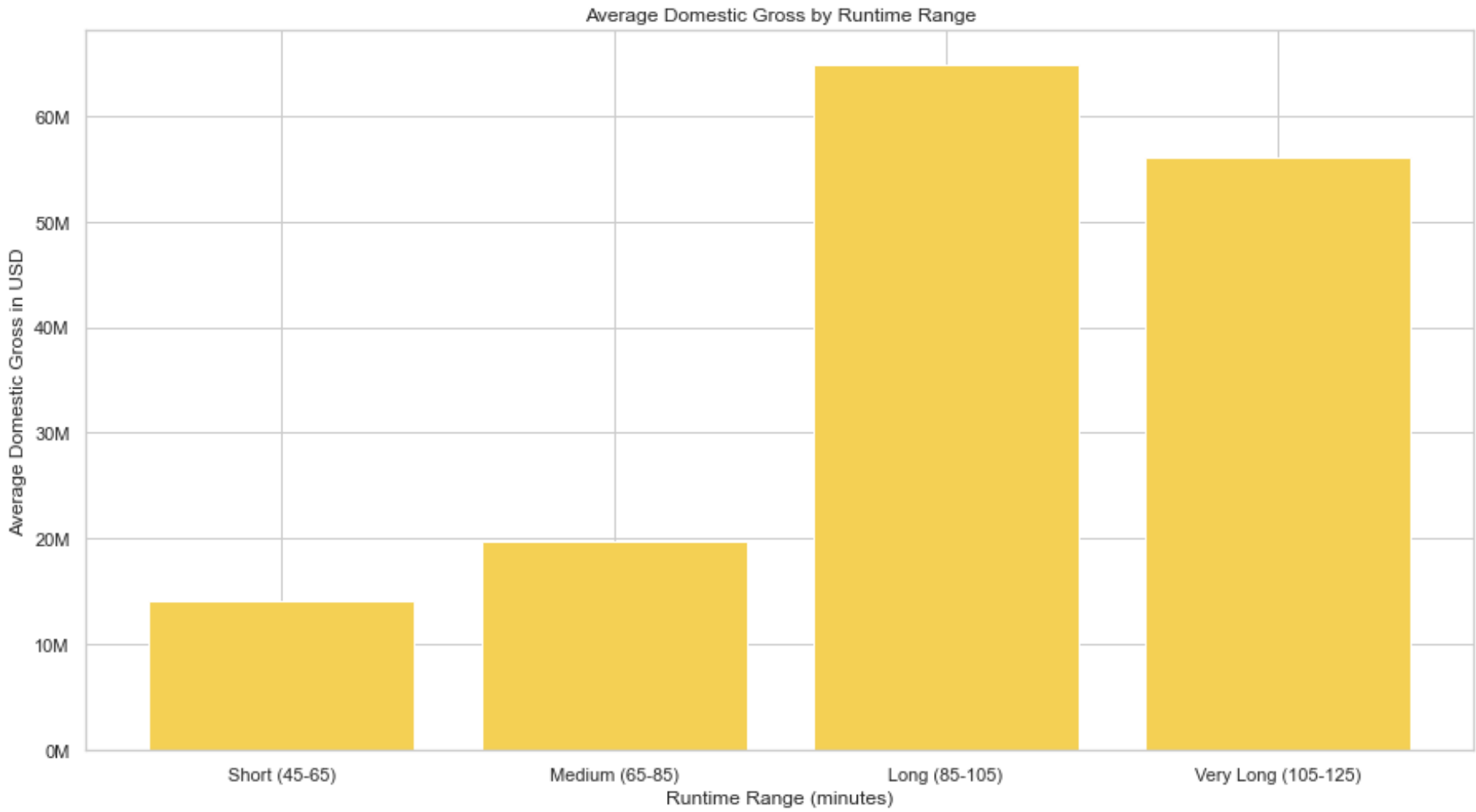
```
# Find out which runtime range receives the highest mean domestic gross
runtime_stats = runtime_summary_inner_quantile.groupby('runtime_range')['cpi_avg_domestic_gross'].mean(
print('Total number of results:', len(runtime_stats))
display(runtime_stats)
```

```
Total number of results: 4

runtime_range
Short (45-65)          1.412140e+07
Medium (65-85)         1.978483e+07
Long (85-105)          6.479307e+07
Very Long (105-125)    5.601178e+07
Name: cpi_avg_domestic_gross, dtype: float64
```

These bin categories can be altered to suit the stakeholders budget because longer movies may incur larger production budgets. These initial bin category settings are a conservative estimate with respect to what is considered a long movie. This analysis, priortizes bins with an equal width of 20 minute windows. For example, a deeper dive may include eight, 10 minute wide bins.

In [108]:

```python
def millions_formatter(x, pos):
    """
    Custom formatter for formatting y-axis values in millions.

    Input:
    - x (float): Value to be formatted
    - pos (int): Position of the tick on the axis

    Output:
    - str: Formatted string representing the value in millions
    """
    return f'{x / 1e6:.0f}M'

# Recreate the DataFrame from the Series to use it in Matplotlib's bar plot
runtime_stats = runtime_stats.reset_index()

# Create the plot
plt.figure(figsize=(15, 8))
plt.bar(runtime_stats['runtime_range'], runtime_stats['cpi_avg_domestic_gross'], color='#F4D054')
plt.xlabel('Runtime Range (minutes)')
plt.ylabel('Average Domestic Gross in USD')
plt.title('Average Domestic Gross by Runtime Range')

# Apply the millions formatter to the y-axis
formatter = FuncFormatter(millions_formatter)
plt.gca().yaxis.set_major_formatter(formatter)

# Show the plot
plt.show()

```



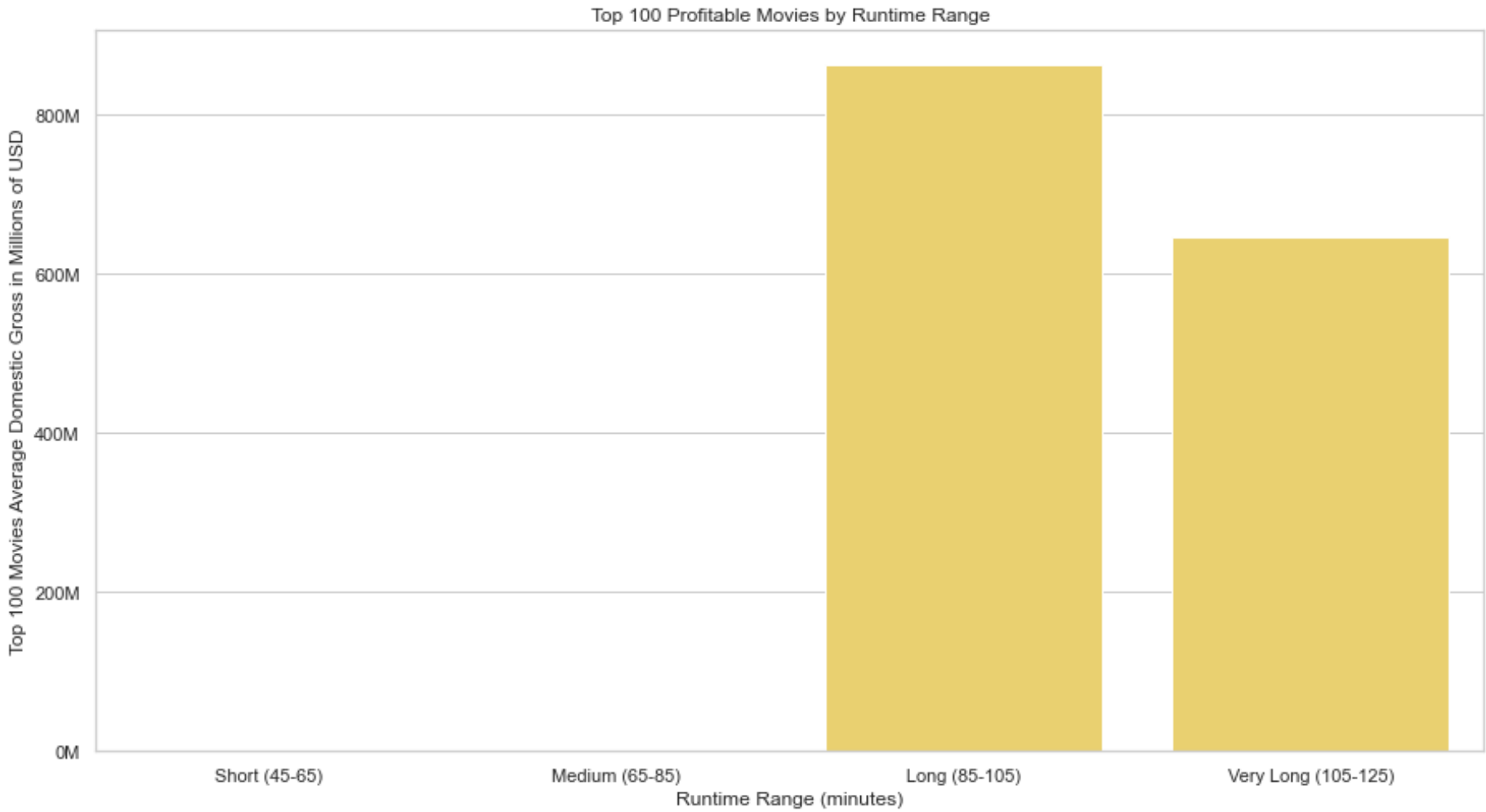
This shows a clearly consistent result where long and very long movies gross higher than short and medium length movies. Again, let's now compare this to the top 100 grossing movies.

In [109]:

```
# Find out which runtime range receives the highest mean gross
runtime_stats2 = top_profit_runtime.groupby('runtime_range')['cpi_avg_domestic_gross'].mean()
print('Total number of results:', len(runtime_stats2))
display(runtime_stats2)
```

```
Total number of results: 4

runtime_range
Short (45-65)                  NaN
Medium (65-85)                 NaN
Long (85-105)          8.625266e+08
Very Long (105-125)    6.470557e+08
Name: cpi_avg_domestic_gross, dtype: float64
```

In [110]:
```
# Create the DataFrame from the Series to use it in a barplot
runtime_stats2 = runtime_stats2.reset_index()

# Create the plot
plt.figure(figsize=(15, 8))
sns.barplot(x='runtime_range', y='cpi_avg_domestic_gross', estimator= sum, ci=None, color='#FDDC5C', data=
plt.xlabel('Runtime Range (minutes)')
plt.ylabel('Top 100 Movies Average Domestic Gross in Millions of USD')
plt.title('Top 100 Profitable Movies by Runtime Range')

# Define a custom formatter function to display values in millions
def millions_formatter(x, pos):
    '''
    Custom formatter for formatting y-axis values in millions.

    Input:
    x : Value to be formatted
    pos : Position of the tick on the axis

    Output:
    formatted_string : Formatted string representing the value in millions
    '''
    return f'{x / 1e6:.0f}M'

# Apply the millions formatter to the y-axis
formatter = FuncFormatter(millions_formatter)
plt.gca().yaxis.set_major_formatter(formatter)

# Show the plot
plt.show()
```



As expected, the data supports that the top 100 grossing movies in the long to very long range are likely to have a higher average domestic gross, there for a higher return on investment.

In [111]:
```
# Close the database connection
conn.close()
```

Summary

This project aims to identify factors influencing box office revenue for movies by analyzing datasets from Box Office Mojo, IMDB, Rotten Tomatoes, The Movie DB, and The Numbers. The analysis focuses on genres, release months, runtime, and their impact on Return on Investment (ROI).

Recommendations:

The data shows that The Film Company should consider the following to miximize profits:

1. Release movies in May/June or November/December.
2. Release action or adventure.
3. Produce a movie with a runtime between 85-125 minutes.
4. Plan for a budget of 44-136 million.

Profitability Trends:

- A strong, positive correlation exists between profit and average domestic gross, indicating that top-grossing films tend to be the most profitable.
  - Caution: this is a simplified approach that does not account for other costs.
- Profit margins for the top 100 grossing movies vary, with a median profit margin of approximately 88%.
- The median profit for a top 100 grossing movie is 500 million.
- The relationship between budget and adjusted average domestic gross of the top 100 grossing movies, does not show a clear correlation. It would seem that movies have seen success with small budgets and others depsite robust budgets, failed to perform.
- Budgets between 44 and 136 million seem to be reasonable for a top 100 grossing movie.

Genre Impact:

- Adventure, Action, and Comedy genres emerge as financially successful, while certain genres like Talk-Show, Short, Reality-TV, Adult, and Game-Show show zero CPI-adjusted average domestic gross, possibly due to missing financial data.
- Adventure, Action, and Comedy account for almost 64% of all adjusted domestic profits

Release Month Influence:

1. Movies released in May, June, July, and November/December tend to be more profitable, emphasizing the importance of strategic release timing.
2. Caution is advised, recognizing external factors like the economy, distribution strategy, advertising budget, and movie's enduring appeal.

Runtime Considerations:

1. After addressing outliers, the runtime data appears normally distributed, with a mean of approximately 87 minutes.

Runtime and Revenue:

1. Classifying movies into runtime ranges (Short, Medium, Long, Very Long) reveals a consistent trend where long and very long movies tend to have higher average domestic gross.

Limitations and Further Considerations:

1. Potential limitations include data gaps, advertising influence, streaming releases, and limited theater releases.
2. Adapting bin categories for runtime based on stakeholder budget considerations is suggested for a more tailored analysis.

Next Steps:

While providing valuable insights, the analysis acknowledges the need for further investigation to address nuances and ensure a comprehensive understanding of movie dynamics influencing financial success.

1. In-Depth Genre Analysis:

Conduct a detailed investigation into genres with zero CPI-adjusted average domestic gross. Verify the data sources and integrity, explore reasons for the absence of financial information, and determine if these genres are genuinely unprofitable or if there are data gaps.

2. Mixed-Genre Film Exploration:

Devote a segment of the analysis to understanding the influence of mixed-genre films on financial performance. This could involve categorizing movies with multiple genres and examining how these combinations contribute to box office revenue. Insights gained could refine genre-specific recommendations.

3. External Factors Impacting Release Timing:

Extend the analysis to consider external factors influencing release timing, beyond month-based trends. Factors like major ecomonmic events, holidays, or cultural phenomena may impact movie profitability. Identifying and understanding these factors can provide more nuanced recommendations for optimal release schedules.

4. Consumer Preferences and Genre Evolution:

Investigate evolving consumer preferences in movie genres over time. Analyze historical trends to identify shifts in audience interests. Understanding how genres have evolved can inform strategic decisions about genre selection/combination and potential shifts in the future.

5. Comprehensive ROI Forecasting Model:

Develop a comprehensive model that incorporates specific factors prioritized by the film company based on their staff strengths and goals. This could involve collaboration with stakeholders to identify key performance indicators (KPIs) and create a forecasting model tailored to the company's unique priorities.

These next steps aim to address identified inconsistencies, enhance the robustness of the analysis, and provide more actionable insights for the film company's decision-making processes.

References

1. https://pypi.org/project/cpi/#description (https://pypi.org/project/cpi/#description) (CPI)
2. https://sqlite.org/forum/forumpost/2ca63507ad (https://sqlite.org/forum/forumpost/2ca63507ad)(PRAGMA table)
3. https://stackoverflow.com/questions/24258878/how-to-split-comma-separated-values/32051164#32051164 (https://stackoverflow.com/questions/24258878/how-to-split-comma-separated-values/32051164#32051164) (SQL split comma separated values)
4. https://www.youtube.com/watch?v=Ohj-CqALrwk (https://www.youtube.com/watch?v=Ohj-CqALrwk) (SQL database backend)
5. https://www.youtube.com/watch?v=E-BEOD0EPDA (https://www.youtube.com/watch?v=E-BEOD0EPDA) (dataframe -> SQL migration)
6. https://www.sqlitetutorial.net/sqlite-import-csv/ (https://www.sqlitetutorial.net/sqlite-import-csv/) (import CSV to SQL)
7. https://www.youtube.com/watch?v=YyUknBHcZB8 (https://www.youtube.com/watch?v=YyUknBHcZB8) (pandas -> SQL)
8. https://stackoverflow.com/questions/13643558/should-glob-glob-be-preferred-over-os-listdir-or-the-other-way-around (https://stackoverflow.com/questions/13643558/should-glob-glob-be-preferred-over-os-listdir-or-the-other-way-around) (glob)
9. https://help.highbond.com/helpdocs/analytics/141/scripting-guide/en-us/Content/lang_ref/functions/r_exclude.htm (https://help.highbond.com/helpdocs/analytics/141/scripting-guide/en-us/Content/lang_ref/functions/r_exclude.htm)(exclude function)
10. https://www.geeksforgeeks.org/difference-between-list-and-dictionary-in-python/ (https://www.geeksforgeeks.org/difference-between-list-and-dictionary-in-python/)(list and dictionary background)
11. https://www.youtube.com/watch?v=tATFQUx0Zx0 (https://www.youtube.com/watch?v=tATFQUx0Zx0) (more glob)
12. https://developer.imdb.com/non-commercial-datasets/ (https://developer.imdb.com/non-commercial-datasets/) (IMBd data)
13. https://www.youtube.com/watch?v=THHwJcKfGLQ (https://www.youtube.com/watch?v=THHwJcKfGLQ) (tsv -> csv)
14. https://www.pythonpool.com/python-scientific-notation/ (https://www.pythonpool.com/python-scientific-notation/)(number to words coverter)
15. https://www.youtube.com/watch?v=inmMGhYU3uU (https://www.youtube.com/watch?v=inmMGhYU3uU) (reset index)
16. https://www.youtube.com/watch?v=xOoCd5VBzXw (https://www.youtube.com/watch?v=xOoCd5VBzXw)(project set up)
17. https://www.themoviedb.org/talk/5daf6eb0ae36680011d7e6ee (https://www.themoviedb.org/talk/5daf6eb0ae36680011d7e6ee)(movie database support)
18. https://learnsql.com/blog/learn-and-practice-sql-joins/ (https://learnsql.com/blog/learn-and-practice-sql-joins/)(SQL joins)
19. https://xkcd.com/color/rgb/ (https://xkcd.com/color/rgb/)(xkcd color codes)

20. https://github.com/kimfetti/Videos/blob/master/Seaborn/26_palettes.ipynb (https://github.com/kimfetti/Videos/blob/master/Seaborn/26_palettes.ipynb)(seaborn color codes)
21. https://www.youtube.com/watch?v=Eg0NJcUWLRM (https://www.youtube.com/watch?v=Eg0NJcUWLRM)(all color codes)
22. https://seaborn.pydata.org/tutorial/color_palettes.html (https://seaborn.pydata.org/tutorial/color_palettes.html) (seaborn pallets)
23. https://gizmodo.com/how-much-money-does-a-movie-need-to-make-to-be-profitab-5747305 (https://gizmodo.com/how-much-money-does-a-movie-need-to-make-to-be-profitab-5747305) (movie article)
24. https://stackoverflow.com/questions/22676081/what-is-the-difference-between-join-and-merge-in-pandas (https://stackoverflow.com/questions/22676081/what-is-the-difference-between-join-and-merge-in-pandas) (join and merge differences PANDAS)
25. https://smallbusiness.chron.com/difference-between-profit-profit-margin-1595.html (https://smallbusiness.chron.com/difference-between-profit-profit-margin-1595.html) (profit vs profit margin)
26. https://learngitbranching.js.org/ (https://learngitbranching.js.org/) (visualizing git)
27. https://flowingdata.com/2009/11/25/9-ways-to-visualize-proportions-a-guide/ (https://flowingdata.com/2009/11/25/9-ways-to-visualize-proportions-a-guide/)(visualization ideas)