Microsoft New Movie Studio

Author: Albane Colmenares

Overview

This notebook examines various movie databases that encompass data from thousands of movies. It focuses on movies released in the past 10 years to ensure the results would remain relevant, and to understand how what metrics generate profit rapidly.

The analysis was centered on 3 main points:

- genre
- runtime and rating
- actors

By conducting a study of these data points, the aim is to provide valuable insights to help Microsoft's new movie studio focus on the main indicators of a movie's success.

It was found that Action movies generated the highest profit from 2010 to 2018 with over $8 billion generated.

No correlation was found between a movie's runtime and profit. In addition, most profitable movies tend to be much longer than the majority of movies so as long as a movie has the right recipe, runtime does not matter as much.
As for how long a movie should be: while movies tend to have a higher rating when on a shorter range, no correlation was found between a movie's runtime and profit. In addition, most profitable movies tend to be much longer than the majority of movies so as long as a movie has the right recipe, runtime does not matter as much.
Finally, some actors were identified as generating the most profit in recent years - and having them

Business problem

The project's goal is to provide Microsoft's head of new movie studio with 3 recommendations for the new studio they are creating. The actionable insights are based on data on existing movies' performances by understanding which movies are doing best at the box office. Datasets are from Box Office Mojo, IMDb, Rotten Tomatoes, The Movie Database and The Numbers.

From this data, Pearson's correlation was calculated to review the relationship between a movie's runtime and its profit. Median of both runtime and ratings was calculated to understand the relative preference. Groupby was used to review the sum of profit by genre and by actor.

Data Understanding

The data comes from main movies' data collection websites: Box Office Mojo, IMDb, Rotten Tomatoes, The Movie Database, and The Numbers.
The data represents all movies' key metrics of performance and descriptions:

Basic movies' descriptions:

- their title
- the studio that created the movie
- in what year the movie first went out in theaters
- how long the movie runs for
- what genres it belongs to
- in which language it is
- if the movie and the movie's title was translated

The persons involved in the movie's creation:

- actors
- directors
- producers
- writers

How movies were received:

- ratings
- by the general public
- by journalists

Movies' key performance metrics:

- worldwide profit calculated from:
  - domestic and foreign box office
  - minus budget invested to make the movie
- movie ratings
- runtime

The target variable used to measure movies' performances was profit.

The sum of profit was calculated by the various dimensions selected to measure a movie's success.

Data Understanding

The next lines of codes will open all data sources to understand what information is accessible, will transform numbers' data into integers by stripping any characters that prevent from reading them as integers.

Standard packages to read and process data in jupyter are imported.

In [1]:
```
# Import standard packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sqlite3

%matplotlib inline
```

1. Box Office Mojo

All movies that don't have domestic revenue have foreign revenues so they were distributed oversees.
Null rows in domestic gross can be replaced by 0, and so can null rows in foreign gross.

In [2]:
```
# Loading bom.movie_gross and storing data into df_bom
df_bom = pd.read_csv("data/bom.movie_gross.csv.gz", compression="gzip")
```

In [3]:
```
# Inspect overall shape of the dataframe
df_bom.shape
```

Out[3]:
```
(3387, 5)
```

In [4]:
```
# Inspect overall info of the dataframe
df_bom.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
```

In [5]:
```
# Convert foreign_gross column as float
df_bom["foreign_gross"] = df_bom["foreign_gross"].str.replace(",","").astyp

# Filling na values with 0 on both columns:
df_bom.update(df_bom[["domestic_gross", "foreign_gross"]].fillna(0))
```

2. IMDb

The IMDb dataset is made of 7 tables, each detailing information about movies.

The used tables will be movie_ratings, movie_basics and known_for. movie_basics includes the movies' genres and movie ids. Movie ids will be used to merge to movie_ratings to have an understanding of the preferred runtime to be able to better understand those that generate the most profit.

Movie ids is also used to merge to principals, to identify actors who starred in the most profitable movies.

In [6]:
```
# Loading and inspecting available datasets
import zipfile
with zipfile.ZipFile("data/im.db.zip", "r") as zip_ref:
    zip_ref.extractall("data")
```

In [7]:
```
# Creating connection to database
conn = sqlite3.connect("data/im.db")

# Creating a cursor
cur = conn.cursor()
```

In [8]:
```
# Opening df_imdb database
df_imdb = pd.read_sql("""
                SELECT *
                FROM sqlite_master
                ;
""", con=conn)
df_imdb
```

Out[8]:

	type	name	tbl_name	rootpage	sql
0	table	movie_basics	movie_basics	2	CREATE TABLE "movie_basics" (\n"movie_id" TEXT...
1	table	directors	directors	3	CREATE TABLE "directors" (\n"movie_id" TEXT,\n...
2	table	known_for	known_for	4	CREATE TABLE "known_for" (\n"person_id" TEXT,\...
3	table	movie_akas	movie_akas	5	CREATE TABLE "movie_akas" (\n"movie_id" TEXT,\...
4	table	movie_ratings	movie_ratings	6	CREATE TABLE "movie_ratings" (\n"movie_id" TEX...
5	table	persons	persons	7	CREATE TABLE "persons" (\n"person_id" TEXT,\n ...
6	table	principals	principals	8	CREATE TABLE "principals" (\n"movie_id" TEXT,\...
7	table	writers	writers	9	CREATE TABLE "writers" (\n"movie_id" TEXT,\n ...

Now opening and inspecting each table that are contained in df_imdb into a dataframe.

In [9]:
```
# Opening and storing movie_basics table
movie_basics = pd.read_sql(
"""
SELECT *
FROM movie_basics
;
"""
, con=conn)

movie_basics.head(3)
```

Out[9]:

	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

In [10]:
```
movie_basics.info()
```

```
<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
```

The raw file contains text initially written in non-English language which causes columns to generate unreadable characters (non-ascii).

A new column "clean_primary" is created to clean the "primary_title" column. It is first encoded as ascii and then decoded again to be read as a string and to replace the characters.

In [11]:
```
# Data cleaning movie_basics
movie_basics["clean_primary"] = movie_basics["primary_title"].str.normalize
```

In [12]:
```
# Opening and storing directors table
directors_df = pd.read_sql(
"""
SELECT *
FROM directors
;
"""
, con=conn)

directors_df.head(3)
```

Out[12]:

	movie_id	person_id
0	tt0285252	nm0899854
1	tt0462036	nm1940585
2	tt0835418	nm0151540

In [13]:
```
# Opening and storing known_for table
known_for_df = pd.read_sql(
"""
SELECT *
FROM known_for
;
"""
, con=conn)

known_for_df.head(3)
```

Out[13]:

	person_id	movie_id
0	nm0061671	tt0837562
1	nm0061671	tt2398241
2	nm0061671	tt0844471

In [14]:
```
# Opening and storing movie_akas table

movie_akas_df = pd.read_sql(
"""
SELECT *
FROM movie_akas
;
"""
, con=conn)

movie_akas_df.head(3)
```

Out[14]:

	movie_id	ordering	title	region	language	types	attributes	is_original_title
0	tt0369610	10	Джурасик свят	BG	bg	None	None	0.0
1	tt0369610	11	Jurashikku warudo	JP	None	imdbDisplay	None	0.0
2	tt0369610	12	Jurassic World: O Mundo dos Dinossauros	BR	None	imdbDisplay	None	0.0

In [15]:
```
# Opening and storing movie_ratings table
movie_ratings_df = pd.read_sql(
"""
SELECT *
FROM movie_ratings
;
"""
, con=conn)

movie_ratings_df.head(3)
```

Out[15]:

	movie_id	averagerating	numvotes
0	tt10356526	8.3	31
1	tt10384606	8.9	559
2	tt1042974	6.4	20

In [16]:
```
# Opening and storing persons table
persons_df = pd.read_sql(
"""
SELECT *
FROM persons
;
"""
, con=conn)

persons_df.head(3)
```

Out[16]:

	person_id	primary_name	birth_year	death_year	primary_profession
0	nm0061671	Mary Ellen Bauder	NaN	NaN	miscellaneous,production_manager,producer
1	nm0061865	Joseph Bauer	NaN	NaN	composer,music_department,sound_department
2	nm0062070	Bruce Baum	NaN	NaN	miscellaneous,actor,writer

In [17]:
```
# Opening and storing principals table

principals_df = pd.read_sql(
"""
SELECT *
FROM principals
;
"""
, con=conn)

principals_df.head(3)
```

Out[17]:

	movie_id	ordering	person_id	category	job	characters
0	tt0111414	1	nm0246005	actor	None	["The Man"]
1	tt0111414	2	nm0398271	director	None	None
2	tt0111414	3	nm3739909	producer	producer	None

In [18]:
```
# Opening and storing writers table

writers_df = pd.read_sql(
"""
SELECT *
FROM writers
;
"""
, con=conn)

writers_df.head(3)
```

Out[18]:

	movie_id	person_id
0	tt0285252	nm0899854
1	tt0438973	nm0175726
2	tt0438973	nm1802864

3.a. Rotten Tomatoes - Movie Info

In [19]:
```
# Inspecting rt.movie_info file
# Loading rt.movie_info and storing data into df_rt

df_rt = pd.read_csv("data/rt.movie_info.tsv.gz", compression="gzip", sep="\

df_rt.head()
```

Out[19]:

	id	synopsis	rating	genre	director	writer	theater_date	dvd_c
0	1	This gritty, fast-paced, and innovative police...	R	Action and Adventure|Classics|Drama	William Friedkin	Ernest Tidyman	Oct 9, 1971	Sep 2
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	Ja 2
2	5	Illeana Douglas delivers a superb performance ...	R	Drama|Musical and Performing Arts	Allison Anders	Allison Anders	Sep 13, 1996	Apr 2
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 1
4	7	NaN	NR	Drama|Romance	Rodney Bennett	Giles Cooper	NaN	I

In [20]:
```
# Inspect overall shape and info of the dataframe
df_rt.shape
df_rt.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1560 entries, 0 to 1559
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   id            1560 non-null   int64 
 1   synopsis      1498 non-null   object
 2   rating        1557 non-null   object
 3   genre         1552 non-null   object
 4   director      1361 non-null   object
 5   writer        1111 non-null   object
 6   theater_date  1201 non-null   object
 7   dvd_date      1201 non-null   object
 8   currency      340 non-null    object
 9   box_office    340 non-null    object
 10  runtime       1530 non-null   object
 11  studio        494 non-null    object
dtypes: int64(1), object(11)
memory usage: 146.4+ KB
```

3.b. Rotten Tomatoes - Reviews

The Rotten Tomatoes dataset does not contain any movie title or movie id, so is not used in this analysis

In [21]:
```
# Inspecting rt.reviews file
# Loading rt.reviews and storing data into df_reviews

df_reviews = pd.read_csv("data/rt.reviews.tsv.gz", compression="gzip", sep=
df_reviews.head()
```

Out[21]:

	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

In [22]:
```
# Inspect overall shape and info of the dataframe
df_reviews.shape
df_reviews.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 54432 entries, 0 to 54431
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   id          54432 non-null  int64 
 1   review      48869 non-null  object
 2   rating      40915 non-null  object
 3   fresh       54432 non-null  object
 4   critic      51710 non-null  object
 5   top_critic  54432 non-null  int64 
 6   publisher   54123 non-null  object
 7   date        54432 non-null  object
dtypes: int64(2), object(6)
memory usage: 3.3+ MB
```

4. The Movie Database

The Movie Database was not used in this particular analysis as The Numbers' dataset contained all information, including budget and worldwide box offife under the same dataset.

In [23]:
```
# Loading tmdb.movies and storing data into df_tmdb
# Dropping the unnamed column upon opening
df_tmdb = pd.read_csv("data/tmdb.movies.csv.gz", compression="gzip", index_
df_tmdb.head()
```

Out[23]:

	genre_ids	id	original_language	original_title	popularity	release_date	title	vote_average
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
1	[14, 12, 16, 10751]	10191	en	How to Train Your Dragon	28.734	2010-03-26	How to Train Your Dragon	7.7
2	[12, 28, 878]	10138	en	Iron Man 2	28.515	2010-05-07	Iron Man 2	6.8
3	[16, 35, 10751]	862	en	Toy Story	28.005	1995-11-22	Toy Story	7.9
4	[28, 878, 12]	27205	en	Inception	27.920	2010-07-16	Inception	8.3

In [24]:
```
# Inspect overall shape and info of the dataframe
df_tmdb.shape
df_tmdb.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 26517 entries, 0 to 26516
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   genre_ids          26517 non-null  object 
 1   id                 26517 non-null  int64  
 2   original_language  26517 non-null  object 
 3   original_title     26517 non-null  object 
 4   popularity         26517 non-null  float64
 5   release_date       26517 non-null  object 
 6   title              26517 non-null  object 
 7   vote_average       26517 non-null  float64
 8   vote_count         26517 non-null  int64  
dtypes: float64(2), int64(2), object(5)
memory usage: 2.0+ MB
```

5. The Numbers

The Numbers" dataset contains both box office and budget numbers per movie so the analysis will start from this dataset.

The file is compressed with gzip. The raw file contains text initially written in non-English language which causes columns to generate unreadable characters (non-ascii).

A new column "clean_movie" is created to clean the "movie" column. It is first encoded as ascii and then decoded again to be read as a string and to replace the characters.

The column "start_year" will be used to merge the IMDb dataframe with The Numbers" dataframe. "start_year""s data type is integer so the "year" column will be converted from string to integer in The Number"s dataframe.

In [25]:
```
# Inspecting tn.movie_budgets file
# Loading tn.movie_budgets and storing data into df_tn
df_tn = pd.read_csv("data/tn.movie_budgets.csv.gz", compression="gzip")
```

In [26]:
```
df_tn["clean_movie"] = df_tn["movie"].str.encode("ascii", errors="replace")
```

In [27]:
```
df_tn.shape
df_tn.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5782 entries, 0 to 5781
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   id                 5782 non-null   int64 
 1   release_date       5782 non-null   object
 2   movie              5782 non-null   object
 3   production_budget  5782 non-null   object
 4   domestic_gross     5782 non-null   object
 5   worldwide_gross    5782 non-null   object
 6   clean_movie        5782 non-null   object
dtypes: int64(1), object(6)
memory usage: 316.3+ KB
```

All number columns are turned as integers to be able to make calculations from them.

In [28]:
```
# Make all number columns as integers
columns_to_integers = ["production_budget" , "domestic_gross", "worldwide_g

for column in columns_to_integers:
    df_tn[column] = df_tn[column].astype(str).str.replace(",", '').str.repl

df_tn.head()
```

Out[28]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross	clean_movie
0	1	Dec 18, 2009	Avatar	425000000	760507625	2776345279	Avatar
1	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	410600000	241063875	1045663875	Pirates of the Caribbean: On Stranger Tides
2	3	Jun 7, 2019	Dark Phoenix	350000000	42762350	149762350	Dark Phoenix
3	4	May 1, 2015	Avengers: Age of Ultron	330600000	459005868	1403013963	Avengers: Age of Ultron
4	5	Dec 15, 2017	Star Wars Ep. VIII: The Last Jedi	317000000	620181382	1316721747	Star Wars Ep. VIII: The Last Jedi

Data Preparation

The Numbers dataset, as well as IMDb 3 tables are used to conduct the analysis.

1. Numeric columns

Two columns were created in The Numbers' Dataset.

- worldwide_profit column is created by subtracting budget to worldwide_gross
  - Profit is calculated by decucting costs from revenues
  - worldwide_profit is divided by one million to facilitate reading
- year: using the year in the date of release_date column

1. a. Worldwide profit

In [29]:
```
# Create worldwide_profit column divided by 1,000,000 for easier read. The
df_tn["worldwide_profit"] = (df_tn["worldwide_gross"] - df_tn["production_b

df_tn["worldwide_profit"]
```

Out[29]:
```
0       2351.345279
1        635.063875
2       -200.237650
3       1072.413963
4        999.721747
            ...
5777      -0.007000
5778       0.234495
5779      -0.003662
5780      -0.001400
5781       0.179941
Name: worldwide_profit, Length: 5782, dtype: float64
```

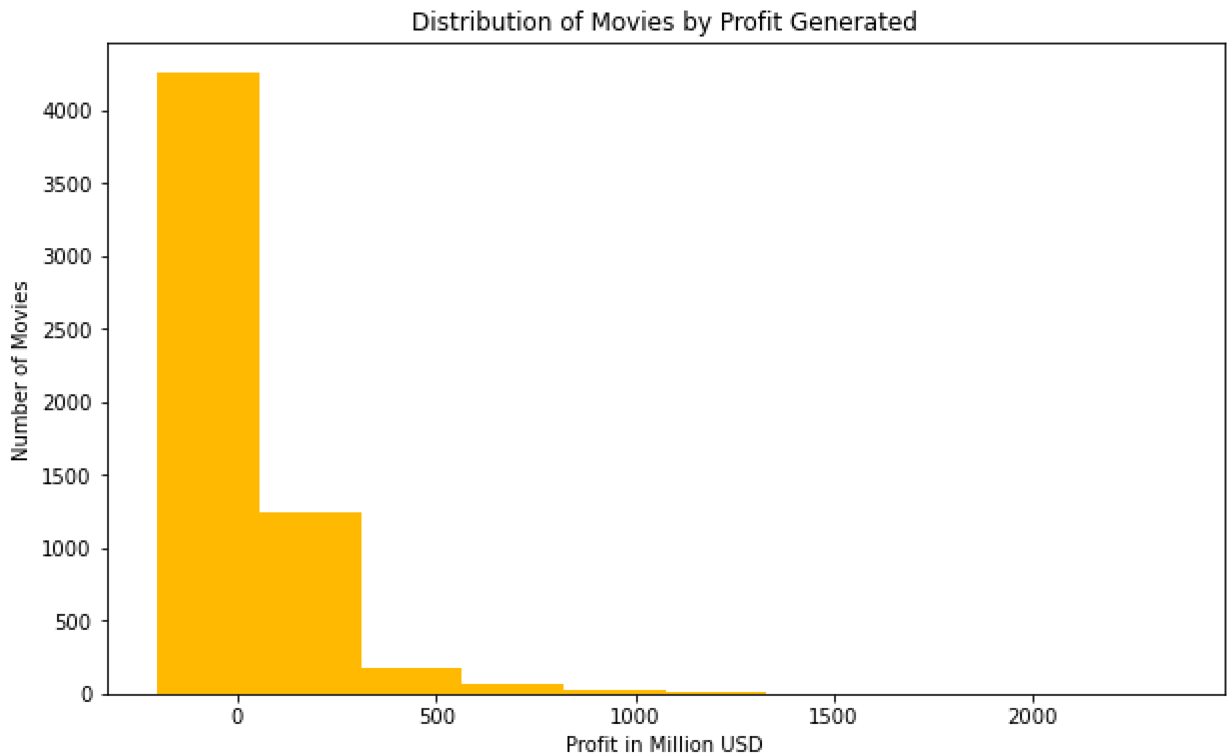
1. b. Year

In [30]:
```
# Create year column
df_tn["year"] = df_tn["release_date"].str[-4:].astype(int)
```

A histogram of all movies' profit was drawn to understand what profit threshold determines whether a movie is considered successful, from its results.
The histogram is highly skewed to the left, indicating most movies either don't make profit or make up to $300 million.

The number of movies reduces drastically from the $500 million profit mark, making it the threshold to define top performer movies to base the analysis on.

In [31]:
```
# See distribution through histograms to determine which movies to focus on
fig = plt.subplots(figsize=(10,6))

x = df_tn["worldwide_profit"]
num_bins = 10
plt.hist(x, num_bins, color="#FFB900")
plt.title("Distribution of Movies by Profit Generated")
plt.xlabel("Profit in Million USD")
plt.ylabel("Number of Movies")


plt.show()
```



2. Creating the Dataframe for Most Profitable Movies

2.a. df_top_tn

The dataframe df_top_tn is now created to filter only on top performing movies since 2010.

In [32]:
```
df_top_tn = df_tn[(df_tn["worldwide_profit"] > 0.5) & (df_tn["year"] > 2009
# len(df_top_tn)
df_top_tn.head()
```

Out[32]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross	clean_movie
1	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	410600000	241063875	1045663875	Pirates of the Caribbean: On Stranger Tides
3	4	May 1, 2015	Avengers: Age of Ultron	330600000	459005868	1403013963	Avengers: Age of Ultron
4	5	Dec 15, 2017	Star Wars Ep. VIII: The Last Jedi	317000000	620181382	1316721747	Star Wars Ep. VIII: The Last Jedi
5	6	Dec 18, 2015	Star Wars Ep. VII: The Force Awakens	306000000	936662225	2053311220	Star Wars Ep. VII: The Force Awakens
6	7	Apr 27, 2018	Avengers: Infinity War	300000000	678815482	2048134200	Avengers: Infinity War

2.b. movie_title_episode columns

The column movie_title_episode is created in The Numbers' top performing dataframe "df_top_tn" to be used as the main column to merge this dataframe with the from IMDb.

Two columns are created:

1. movie_title
2. movie_episode

Separating the title of the movie from the episode when it was made in sequels. Both are then combined again into the column movie_title_episode ensuring the punctuation is stripped.

When movie titles could not be stripped by a general rule, they were individually modified.

The same columns are created, the same way in "movie_basics"

2.b.1. movie_episode column

In [33]:
```
1. # Creating movie_episode column
# Under movie_episode, keeping only what comes after ":", replace the na va

df_top_tn["movie_episode"] = df_top_tn["clean_movie"].map(lambda x: x.split

# Replace roman numbers for part 1 and 2 for Harry Potter and the Deathly H

df_top_tn["movie_episode"] = df_top_tn["movie_episode"].str.replace("part i
# df_top_tn[["movie_episode_lambda", "movie_episode"]]

# Remove comas for Twilight:
df_top_tn["movie_episode"] = df_top_tn["movie_episode"].str.replace(",", ""

# Remove hyphen for Hunger Games:
df_top_tn["movie_episode"] = df_top_tn["movie_episode"].str.replace(" -", "

# Create space between mission impossible and rest of the movie title, remo
df_top_tn["movie_episode"] = df_top_tn["movie_episode"].str.replace("imposs

# Spiderman contains 3D, to remove
df_top_tn["movie_episode"] = df_top_tn["movie_episode"].str.replace(" 3d",

# Used only to verify that a specific movie is changed as expected
df_top_tn[df_top_tn["clean_movie"].str.contains("(?i)spider")]
```

```
<ipython-input-33-16bfe02c1a0e>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  df_top_tn["movie_episode"] = df_top_tn["clean_movie"].map(lambda x: x.s
plit(r":")).str[1].fillna("").str.lower()
<ipython-input-33-16bfe02c1a0e>:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  df_top_tn["movie_episode"] = df_top_tn["movie_episode"].str.replace("pa
rt ii", "part 2", regex=False).str.replace("part i", "part 1", regex=Fals
e)
<ipython-input-33-16bfe02c1a0e>:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  df_top_tn["movie_episode"] = df_top_tn["movie_episode"].str.replace
(",", "", regex=False)
<ipython-input-33-16bfe02c1a0e>:15: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  df_top_tn["movie_episode"] = df_top_tn["movie_episode"].str.replace(" -
", "", regex=False)
<ipython-input-33-16bfe02c1a0e>:18: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  df_top_tn["movie_episode"] = df_top_tn["movie_episode"].str.replace("im
possible", "impossible ", regex=False)
<ipython-input-33-16bfe02c1a0e>:21: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
```

```
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  df_top_tn["movie_episode"] = df_top_tn["movie_episode"].str.replace(" 3
d", "", regex=False)
```

Out[33]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross	clean_mov
30	31	Jul 3, 2012	The Amazing Spider-Man	220000000	262030663	757890267	The Amazi Spider-M
55	56	May 2, 2014	The Amazing Spider-Man 2	200000000	202853933	708996336	The Amazi Spider-M
98	99	Jul 7, 2017	Spider-Man: Homecoming	175000000	334201140	880166350	Spider-Ma Homecomi
443	44	Dec 14, 2018	Spider-Man: Into The Spider-Verse 3D	90000000	190173195	375381768	Spider-Ma Into T Spider-Ve

2.b.2. movie_title and movie_episode column

In [34]:
```
# Creating column movie_title if there were more than 1 movie
df_top_tn["movie_title"] = df_top_tn["clean_movie"].str.split("Ep\.").str[0
# Leaving the "'"
# df_top_tn["movie_title"] = df_top_tn["clean_movie"].str.split("Ep\.").str


# Creating column combining the movie title and the episode to have the sam
df_top_tn["movie_title_episode"] = df_top_tn["movie_title"].str.replace("."


# Changing "???" and to space,  and "??" to "e" and changing apostrophe to
df_top_tn["movie_title_episode"] = df_top_tn["movie_title_episode"].str.rep

# Extract Dr Seuss and Doctor Seuss, removing extra space on The Lorax and
df_top_tn["movie_title_episode"] = df_top_tn["movie_title_episode"].str.rep

# Mission impossible extra space
df_top_tn["movie_title_episode"] = df_top_tn["movie_title_episode"].str.rep

# Monster Hunt
df_top_tn["movie_title_episode"] = df_top_tn["movie_title_episode"].str.rep

# The Conjuring
df_top_tn["movie_title_episode"] = df_top_tn["movie_title_episode"].str.rep

# The Hangover Part III
df_top_tn["movie_title_episode"] = df_top_tn["movie_title_episode"].str.rep

# We're the millers
# df_top_tn["movie_title_episode"] = df_top_tn["movie_title_episode"].str.r

# John Wick 3 Parabellum
df_top_tn["movie_title_episode"] = df_top_tn["movie_title_episode"].str.rep

# Changing Disney Planes to Planes
df_top_tn["movie_title_episode"] = df_top_tn["movie_title_episode"].str.rep

# Changing Disney Planes to Planes
df_top_tn["movie_title_episode"] = df_top_tn["movie_title_episode"].str.rep

# # Changing Disney Planes to Planes
# df_top_tn["movie_title_episode"] = df_top_tn["movie_title_episode"].str.r

# Used only to verify that a specific movie is changed as expected
df_top_tn[df_top_tn["clean_movie"].str.contains("(?i)millers")]
```

```
<ipython-input-34-59ec59bf2ed1>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  df_top_tn["movie_title"] = df_top_tn["clean_movie"].str.split("Ep\.").s
tr[0].str.split("\bPart\b").str[0].str.split(r"[:)(!]").str[0].str.strip
().str.lower()
<ipython-input-34-59ec59bf2ed1>:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
```

In [35]:
```
# This cell is used only for troubleshooting. For troubleshoot: replace spa
df_top_tn[df_top_tn["movie_title_episode"].str.contains("(?i)insurgent", na
```

Out[35]:
```
321    insurgent
Name: movie_title_episode, dtype: object
```

Now applying the same process in the dataframe movie_basics to allow to merge the two dataframes on the column "movie_title_episode".

In [36]:
```
# Same process in movie_basics

# Creating column movie_episode if there were more than 1 movie
movie_basics["movie_episode"] = movie_basics["clean_primary"].str.split(r":

# Creating column movie_title if there were more than 1 movie
# movie_basics["movie_title"] = movie_basics["clean_primary"].str.split("\b
movie_basics["movie_title"] = movie_basics["clean_primary"].str.split("\bPa

# Creating column combining the movie title and the episode to have the sam
movie_basics["movie_title_episode"] = movie_basics["movie_title"] +  movie_
# movie_basics["movie_title_episode"] = movie_basics["movie_title"] +  movi

# Change apostrophe to space
movie_basics["movie_title_episode"] = movie_basics["movie_title_episode"].s

# Turn Eight to numeric in Ocean's 8
movie_basics["movie_title_episode"] = movie_basics["movie_title_episode"].s

# Correcting Maze Runner
movie_basics["movie_title_episode"] = movie_basics["movie_title_episode"].s

# Changing Shazam
movie_basics["movie_title_episode"] = movie_basics["movie_title_episode"].s

# Changing Jackass Bad Grandpa
movie_basics["movie_title_episode"] = movie_basics["movie_title_episode"].s

# Changing Prince of Persia
movie_basics["movie_title_episode"] = movie_basics["movie_title_episode"].s


# For verification
# movie_basics[movie_basics["original_title"].str.contains("(?i)sands of ti
```

In [37]:
```
# For troubleshoot: replace spaces with "_"
movie_basics[movie_basics["movie_title_episode"].str.contains("(?i)hansel",
```

Out[37]:
```
7512                 hansel_and_gretel_witch_hunters
27509                    hansel_and_gretel_get_baked
38504      hansel_and_gretel_warriors_of_witchcraft
70676                                hansel_et_gretel
72618                                hansel_vs_gretel
120097                              hansel_und_gretel
140254                              gretel_and_hansel
Name: movie_title_episode, dtype: object
```

3. Merging Both Dataframes Into top_movies

Now that the column movie_title_episode was created in both dataframes, most profitable movies from The Numbers and Basics from IMDb database. The two dataframes are also merged on year, to avoid duplication if two movies are named the same, for example Les Miserables.

In [38]:
```
top_movies = pd.merge(df_top_tn, movie_basics, how="left",
                      left_on=["movie_title_episode", "year"],
                      right_on=["movie_title_episode", "start_year"])
```

In [39]:
```
top_movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1321 entries, 0 to 1320
Data columns (total 21 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   id                   1321 non-null   int64  
 1   release_date         1321 non-null   object 
 2   movie                1321 non-null   object 
 3   production_budget    1321 non-null   int64  
 4   domestic_gross       1321 non-null   int64  
 5   worldwide_gross      1321 non-null   int64  
 6   clean_movie          1321 non-null   object 
 7   worldwide_profit     1321 non-null   float64
 8   year                 1321 non-null   int64  
 9   movie_episode_x      1321 non-null   object 
 10  movie_title_x        1321 non-null   object 
 11  movie_title_episode  1321 non-null   object 
 12  movie_id             1135 non-null   object 
 13  primary_title        1135 non-null   object 
 14  original_title       1135 non-null   object 
 15  start_year           1135 non-null   float64
 16  runtime_minutes      1131 non-null   float64
 17  genres               1134 non-null   object 
 18  clean_primary        1135 non-null   object 
 19  movie_episode_y      1135 non-null   object 
 20  movie_title_y        1135 non-null   object 
dtypes: float64(3), int64(5), object(13)
memory usage: 227.0+ KB
```

The null rows for movie_id all represent movies that generated profits lower than $500 million and the sample already contains more than 1,100 rows so is large enough to be representatitve. Null values will be dropped.

In [40]:
```
top_profit_movies = top_movies.dropna(subset=["movie_id"])
```

In [41]:
```
top_profit_movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1135 entries, 0 to 1320
Data columns (total 21 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   id                   1135 non-null   int64  
 1   release_date         1135 non-null   object 
 2   movie                1135 non-null   object 
 3   production_budget    1135 non-null   int64  
 4   domestic_gross       1135 non-null   int64  
 5   worldwide_gross      1135 non-null   int64  
 6   clean_movie          1135 non-null   object 
 7   worldwide_profit     1135 non-null   float64
 8   year                 1135 non-null   int64  
 9   movie_episode_x      1135 non-null   object 
 10  movie_title_x        1135 non-null   object 
 11  movie_title_episode  1135 non-null   object 
 12  movie_id             1135 non-null   object 
 13  primary_title        1135 non-null   object 
 14  original_title       1135 non-null   object 
 15  start_year           1135 non-null   float64
 16  runtime_minutes      1131 non-null   float64
 17  genres               1134 non-null   object 
 18  clean_primary        1135 non-null   object 
 19  movie_episode_y      1135 non-null   object 
 20  movie_title_y        1135 non-null   object 
dtypes: float64(3), int64(5), object(13)
memory usage: 195.1+ KB
```

In [42]:
```
top_profit_movies.head()
```

Out[42]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross	clean_movie	v
0	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	410600000	241063875	1045663875	Pirates of the Caribbean: On Stranger Tides	
1	4	May 1, 2015	Avengers: Age of Ultron	330600000	459005868	1403013963	Avengers: Age of Ultron	
2	5	Dec 15, 2017	Star Wars Ep. VIII: The Last Jedi	317000000	620181382	1316721747	Star Wars Ep. VIII: The Last Jedi	
3	6	Dec 18, 2015	Star Wars Ep. VII: The Force Awakens	306000000	936662225	2053311220	Star Wars Ep. VII: The Force Awakens	
4	7	Apr 27, 2018	Avengers: Infinity War	300000000	678815482	2048134200	Avengers: Infinity War	

5 rows × 21 columns

Unnecessary or duplicated columns are dropped. Only the genre, the cleaned version of the title is necessary, along with the year

In [43]:
```
# Drop unnecessary columns
top_profit_movies = top_profit_movies.drop(columns=["release_date", "id","m
```

In [44]:
```
top_profit_movies.head()
```

Out[44]:

	production_budget	domestic_gross	worldwide_gross	clean_movie	worldwide_profit	year	movie_
0	410600000	241063875	1045663875	Pirates of the Caribbean: On Stranger Tides	635.063875	2011	
1	330600000	459005868	1403013963	Avengers: Age of Ultron	1072.413963	2015	a
2	317000000	620181382	1316721747	Star Wars Ep. VIII: The Last Jedi	999.721747	2017	sta
3	306000000	936662225	2053311220	Star Wars Ep. VII: The Force Awakens	1747.311220	2015	star
4	300000000	678815482	2048134200	Avengers: Infinity War	1748.134200	2018	aveng

Data Modeling

1. Evaluating most profitable genres

Multiple genres are attributed to genres but only the first one, the main genre, will be used to determine the category of a movie.

To do so, the column genre will be turned as a list so the column "main_genre" can be created with the first item of the list.

The sum of profit by genre is calculated using groupby, and summarized among the top 10 genres.

Action genre generated the most profit between 2010 and 2018, producing over 82 billion dollars in profit. Adventure and Comedy movies get the second and third place, reaching respectively 37 and 15 billion dollar profit over these years.

Focusing on these three genres - primarily Action movies, will be the safest choice for Microsoft to ensure higher profits can be generated.

In [45]:
```
top_profit_movies["genres"] = top_profit_movies["genres"].str.split(",")
top_profit_movies["main_genre"] = top_profit_movies["genres"].str[0]

top_profit_movies.head()
```

Out[45]:

	production_budget	domestic_gross	worldwide_gross	clean_movie	worldwide_profit	year	movie_t
0	410600000	241063875	1045663875	Pirates of the Caribbean: On Stranger Tides	635.063875	2011	s
1	330600000	459005868	1403013963	Avengers: Age of Ultron	1072.413963	2015	ave
2	317000000	620181382	1316721747	Star Wars Ep. VIII: The Last Jedi	999.721747	2017	star
3	306000000	936662225	2053311220	Star Wars Ep. VII: The Force Awakens	1747.311220	2015	star v
4	300000000	678815482	2048134200	Avengers: Infinity War	1748.134200	2018	avenge

In [46]:
```
top_10_genres = top_profit_movies.groupby(["main_genre"])[["worldwide_gross
top_10_genres
```

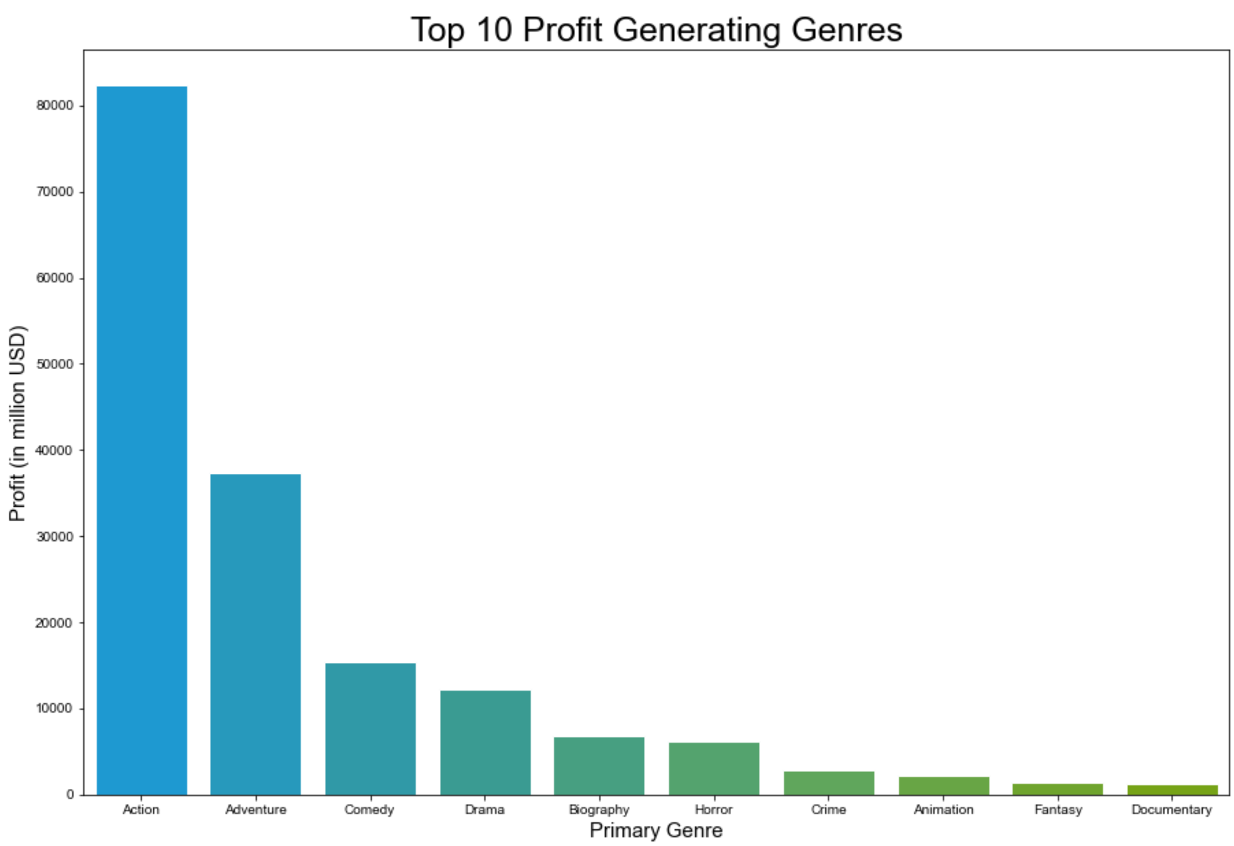
Out[46]:

	main_genre	worldwide_gross	worldwide_profit
0	Action	116773469805	82337.079805
1	Adventure	50645833404	37273.233404
2	Comedy	21769929094	15392.479094
3	Drama	16074553460	12183.230810
4	Biography	8864816505	6758.046505
5	Horror	7185039446	6223.939446
6	Crime	4149855259	2826.155259
7	Animation	2618078686	2195.078686
8	Fantasy	1777512123	1398.512123
9	Documentary	1799732034	1220.407034

In [47]:

```
# Displaying sum of profit by genres
fig1, ax1 = plt.subplots(figsize=(15, 10))

x = top_10_genres["main_genre"]
y = top_10_genres["worldwide_profit"]

sns.set(style="whitegrid", color_codes=True, font_scale=2)

sns.barplot(x, y, palette="blend:#00A4EF,#7FBA00")

ax1.set_title("Top 10 Profit Generating Genres", fontsize=25)
ax1.set_xlabel("Primary Genre", fontsize=15)
ax1.set_ylabel("Profit (in million USD)", fontsize=15)

plt.savefig("images/profit_genres.png")
```

```
/Users/albane/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/se
aborn/_decorators.py:36: FutureWarning: Pass the following variables as k
eyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword w
ill result in an error or misinterpretation.
  warnings.warn(
```



2. Defining How Long a Movie Should Run For

The relationship between profit, preferrence and runtime is evaluated by merging the top_profit_movies dataframe created and the movie_ratings_df from IMDb.

Both dataframes are merged on the movie_id column and unnecessary, duplicate columns are dropped. In this case, genre is not evaluated so is dropped as well.

Most movies run higher than 75 minutes so shorter movies are considered outliers and dropped.

While the median movie lasts 107 minutes, and average rating is at 6.5, the most profitable movies may vary between short and longer times - the most profitable movies even seem to last much longer: around 140 minutes. The low result of Pearson"s correlation: 0.30, confirms there is no correlation between how long a movie is and how profitable it can be.

In [48]:
```
profit_and_ratings = top_profit_movies.merge(movie_ratings_df,
                                              how="inner",
                                             left_on="movie_id",
                                             right_on="movie_id")
```

In [49]:
```
profit_and_ratings = profit_and_ratings.drop(columns=["clean_movie", "movie
```

In [50]:
```
# Removing outliers
profit_and_ratings = profit_and_ratings[profit_and_ratings["runtime_minutes
```

In [51]:
```
runtime = profit_and_ratings["runtime_minutes"].tolist()
ratings = profit_and_ratings["averagerating"].tolist()
```

In [52]:
```
median_runtime = np.median(runtime)
print(median_runtime)

median_rating = np.median(ratings)
print(median_rating)
```

```
107.0
6.5
```

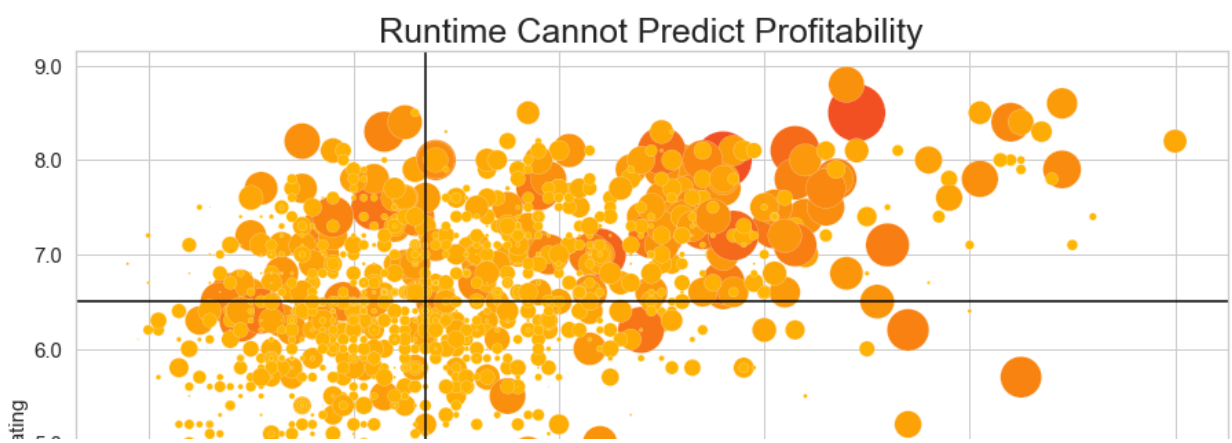
In [53]:
```
profit_and_ratings[["worldwide_profit", "runtime_minutes"]].corr(method="pe
```

Out[53]:

	worldwide_profit	runtime_minutes
worldwide_profit	1.000000	0.303556
runtime_minutes	0.303556	1.000000

In [54]:
```
fig, ax = plt.subplots(figsize=(15, 10))

sns.scatterplot(data=profit_and_ratings,
                x="runtime_minutes",
                y="averagerating",
               hue="worldwide_profit",
               s=profit_and_ratings["worldwide_profit"],
               palette="blend:#FFB900,#F25022")
plt.axvline(x=median_runtime, color="black")
plt.axhline(y=median_rating, color="black")

ax.set_title("Runtime Cannot Predict Profitability", fontsize=25)
ax.set_xlabel("Runtime in Minutes", fontsize=15)
ax.set_ylabel("Rating", fontsize=15)
ax.set_xticklabels(ax.get_xticks(),size = 15)
ax.set_yticklabels(ax.get_yticks(),size = 15)



plt.legend(title="Profit", fontsize=15)

plt.savefig("images/profit_runtime.png")
plt.show()
```

```
<ipython-input-54-98a9c1fbad77>:15: UserWarning: FixedFormatter should on
ly be used together with FixedLocator
  ax.set_xticklabels(ax.get_xticks(),size = 15)
<ipython-input-54-98a9c1fbad77>:16: UserWarning: FixedFormatter should on
ly be used together with FixedLocator
  ax.set_yticklabels(ax.get_yticks(),size = 15)
```



3. Identifying the Actors Most Likely to Generate Profits

The principals table from IMDb database is filtered on the category "actor" and "actress" to then be merged with the persons table from the same database.

This creates the actors_names" dataframe, associating person_id with the person"s name - for actors and actresses only.

The age of the actors is not analyzed here so columns related to birth and death are dropped, along with whether the person has other professions.

actors_names is then merged with the top_profit_movies. The final dataframe top_profit_movies_actors is created.

Finally, using groupby, the sum of profit generated by actors and actresses between 2010 and 2018 is calculated and stored in the dataframe top_10_actors_profit.

Among the top actors, Robert Downey Jr. is the one who created the most profit with 7.8 billion dollars, closely followed by Dwayne Johnson with 6.7 billion dollars, and Chris Evans, with 5.8 billion.

Casting these actors in movies is more likely to contribute in the movie"s success, hence reaching higher profits.

In [55]:
```
# Filtering only on category of persons identified as actor
df_actors = principals_df[["movie_id", "person_id", "category"]]
df_actors = df_actors[(df_actors["category"] == "actor") | (df_actors["cate
df_actors.head()
```

Out[55]:

	movie_id	person_id	category
0	tt0111414	nm0246005	actor
4	tt0323808	nm3579312	actress
5	tt0323808	nm2694680	actor
6	tt0323808	nm0574615	actor
7	tt0323808	nm0502652	actress

In [56]:
```
actors_names = pd.merge(df_actors, persons_df, how="inner",
                        left_on=["person_id"],
                        right_on=["person_id"]
                        )
```

In [57]:
```
actors_names.head()
```

Out[57]:

	movie_id	person_id	category	primary_name	birth_year	death_year	primary_profession
0	tt0111414	nm0246005	actor	Tommy Dysart	NaN	NaN	actor
1	tt0323808	nm3579312	actress	Brittania Nicol	NaN	NaN	actress,soundtrack
2	tt0323808	nm2694680	actor	Henry Garrett	NaN	NaN	actor
3	tt0323808	nm0574615	actor	Graham McTavish	1961.0	NaN	actor,soundtrack,director
4	tt1680140	nm0574615	actor	Graham McTavish	1961.0	NaN	actor,soundtrack,director

In [58]:
```
# Drop unnecessary columns
actors_names = actors_names.drop(columns=["birth_year", "death_year", "prim
```

In [59]:
```
actors_names.head()
```

Out[59]:

	movie_id	person_id	category	primary_name
0	tt0111414	nm0246005	actor	Tommy Dysart
1	tt0323808	nm3579312	actress	Brittania Nicol
2	tt0323808	nm2694680	actor	Henry Garrett
3	tt0323808	nm0574615	actor	Graham McTavish
4	tt1680140	nm0574615	actor	Graham McTavish

In [60]:
```
top_profit_movies_actors = pd.merge(top_profit_movies, actors_names, how="i
                                    left_on=["movie_id"],
                                    right_on=["movie_id"])
```

In [61]:
```
top_profit_movies_actors.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4433 entries, 0 to 4432
Data columns (total 14 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   production_budget    4433 non-null   int64  
 1   domestic_gross       4433 non-null   int64  
 2   worldwide_gross      4433 non-null   int64  
 3   clean_movie          4433 non-null   object 
 4   worldwide_profit     4433 non-null   float64
 5   year                 4433 non-null   int64  
 6   movie_title_episode  4433 non-null   object 
 7   movie_id             4433 non-null   object 
 8   runtime_minutes      4425 non-null   float64
 9   genres               4430 non-null   object 
 10  main_genre           4430 non-null   object 
 11  person_id            4433 non-null   object 
 12  category             4433 non-null   object 
 13  primary_name         4433 non-null   object 
dtypes: float64(2), int64(4), object(8)
memory usage: 519.5+ KB
```

In [62]:
```
top_profit_movies_actors.head()
```

Out[62]:

	production_budget	domestic_gross	worldwide_gross	clean_movie	worldwide_profit	year	movie
0	410600000	241063875	1045663875	Pirates of the Caribbean: On Stranger Tides	635.063875	2011	
1	410600000	241063875	1045663875	Pirates of the Caribbean: On Stranger Tides	635.063875	2011	
2	410600000	241063875	1045663875	Pirates of the Caribbean: On Stranger Tides	635.063875	2011	
3	410600000	241063875	1045663875	Pirates of the Caribbean: On Stranger Tides	635.063875	2011	
4	330600000	459005868	1403013963	Avengers: Age of Ultron	1072.413963	2015	a'

In [63]:
```
top_10_actors_profit = top_profit_movies_actors.groupby(["primary_name"])[[
```

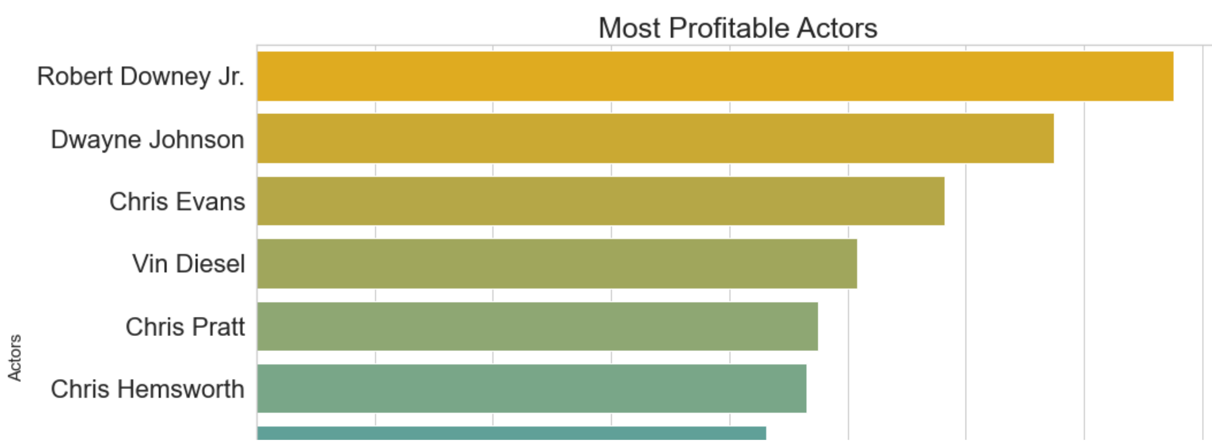
In [64]:
```
top_10_actors_profit
```

Out[64]:

	primary_name	worldwide_gross	worldwide_profit
0	Robert Downey Jr.	9649390358	7758.790358
1	Dwayne Johnson	8620447217	6747.447217
2	Chris Evans	7291117419	5823.517419
3	Vin Diesel	6376832737	5078.832737
4	Chris Pratt	5967404450	4748.904450
5	Chris Hemsworth	6127577586	4658.977586
6	Mark Ruffalo	5397048105	4317.448105
7	Anne Hathaway	5349458875	4047.458875
8	Scarlett Johansson	4839122463	3893.622463
9	Jennifer Lawrence	4963879467	3821.729467

In [65]:
```
# Displaying sum of profit by actors
fig2, ax2 = plt.subplots(figsize=(15, 10))

x = top_10_actors_profit["worldwide_profit"]
y = top_10_actors_profit["primary_name"]

sns.set(style="whitegrid", color_codes=True)

sns.barplot(x, y, palette="blend:#FFB900,#00A4EF", orient="h")

ax2.set_title("Most Profitable Actors", fontsize=25)
ax2.set_xlabel("Profit in Million USD", fontsize=15)
ax2.set_ylabel("Actors", fontsize=15)



plt.savefig("images/profit_actors.png")

plt.show()
```

```
/Users/albane/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/se
aborn/_decorators.py:36: FutureWarning: Pass the following variables as k
eyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword w
ill result in an error or misinterpretation.
  warnings.warn(
```



Conclusions

Below are three recommendations to create profitable movies

1. Genre

- The first recommendation is to produce Action movies - which have generated the most profit from 2010 to 2018. This genre created 82 billion dollars profit over these years: 45 billion more than the second most profitable genre: Adventure. Comedy movies would come third recommendation for movies' genre to ensure reaching the highest profits rapidly.

2. Runtime

- The second recommendation is to not fall into the trap of making a shorter movie to hope to reach a broader audience. No correlation was found between a movie's length in minutes and the profit it generated. While the median movie runs for 107 minutes, the most profitable ones have seen to be longer (140 minutes) or shorter (90 minutes), against initial expectations.

3. Casting

- The last recommendation is to pick actors carefully. A list of actors became notably famous in their genre and are linked to higher profits generated. Actors such as Robert Downey Jr., Dwayne Johnson, Chris Evans have starred in the most profitable movies and became icons for the Action, Adventure and Comedy movies and are more likely to arouse interest for the movies they are part of. Care should be exercised to ensure these actors are associated with the genre they are famous for.

Limitations

- The analysis was run on the years 2010 to 2018 and would be more precise if it included even more recent data
- Some of the movies were not matched with a genre due to title differences and higher precision would be gained by pairing movie_id rather than merging on movie titles
- The analysis is based on box office profits, which do not include all other more recent revenue generators such as streaming revenue and product placement.

In [66]:
```
# Closing connection
conn.close()
```