Final Project Submission

Please fill out:

- **Student name:** Shayan Abdul Karim Khan
- **Student pace:** Self Paced
- **Scheduled project review date/time:** Friday Aug 26th 2022
- **Instructor name:** Abhineet Kulkarni
- **Blog post URL:**

Problem Overview

Venture Opportunity:

A lot of companies are producing original video content generating huge revenues, including box office movies. In the post-pandemic world, people have filled theaters and cinema halls allowing top movies to rack up millions in box office revenues.

Client Insight:

Microsoft has decided to venture into the world of movie production with a new movie studio. In order to make the movie studio a success, Microsoft needs to understand what kind of movies to produce.

Client Goals:

This notebook focuses on three client goals:

1. Positive viewer response
2. Domestic vs Foreign Launch
3. Profitable

Business Questions

This notebook will focus on three business questions to help achieve the client's goals. There are **three** business questions that this notebook will recommend answers to:

1. Which genres to focus on?
    - The popular genres will identify what type of movie will receive positive viewer responses

It is important to understand the customer base and how different factors influence their behavior. Any product should primmarily be focused on providing the customer with whata they need. This guarantees success in the long run and develops customer loyalty. Customer satisfaction is directly correlated with profits and better reviews which will allow long-term success.

2. Should the movie be launched internationally?
    - This will help the client understand whether it is a viable option to launch the movie internationally or only focus on domestic launch. If the movie needs to be launched internationally, it will need to have a cast, crew, story, advertising and marketing plans made accordingly. Whereas only a US specific production would require different plans.

3. What should be the expected budget/initial investment?
    - The financial aspects will help identify how successful the business side will be. It is important to remember that a movie can be very popular but if it's not adequately profitable, it will be difficult for the movie studio to continue operations.

Value Proposition:

This notebook will look at multiple sources of data to understand what will allow Microsoft to generate the most promising content and set it up for long-term success. The strategy described below will be used to generate actionable insight to answer the **Business Questions** based on two main criterion:

1. *Movie Reviews*
2. *Movie Profits*

Strategy

To answer the complex question of the secret ingredients of a successful movie, there are a myriad of aspects to consider. The different aspects analyzed in this notebook will be evaluated based on the two KPIs listed above; Profits and reviews.

Good **Movie Reviews** are important indicators of the quality of a movie nonetheless they can not solely justify an investment opportunity. Therefore **Movie Profits** will be used as an indicator of high potential investment opportunities. The two KPIs coupled together can give the client a more thorough lay of the land.

These KPIs will be used to answer the three main business questions in the following manner:

1. Which genres to focus on?
    - genre vs average reviews analysis
    - genre vs profit analysis
    - genre vs average reviews vs profit analysis

2. Should the movie be launched internationally?
    - genre vs domestic profit analysis
    - genre vs foregin profit analysis

3. What should be the expected budget/initial investment?
    - genres vs production budget vs domestic & foreign profits analysis

Data Sources

To solve the problem, 5 data sources have been gathered. These data sources will be investigated to identify which data will be helpful in the analysis. The data sources are listed below:

- Box Office Mojo (https://www.boxofficemojo.com/)
- The Numbers (https://www.the-numbers.com/)
- Rotten Tomatoes (https://www.rottentomatoes.com/)
- IMDB (https://www.imdb.com/)
- TheMovieDB (https://www.themoviedb.org/)

The content and relevance of the data available is dealt with in detail in the **Data Understanding** Section.

Data Understanding

The data dources listed have the following characteristics discussed in this section:

- Contents of the datasets
- Features of the datasets
- Relevance of the datasets to the project
- Relevant features of the datasets that will be used for analysis
- Relation between the different datasets
- Limitations of the datasets
- Avenues of analysis that will be pursued

Importing Python Libraries

We will start by importing the appropriate python libraries to explore the datasets.

In [1]:
```
import pandas as pd #imports the pandas library as pd to work on databa
import sqlite3 as sql # imports the sqlite3 library to leverage sql wit
from pandasql import sqldf # imports pandas sql library
import matplotlib.pyplot as plt # importing matplotlib for visualizatio
%matplotlib inline
import numpy as np # imports the numpy library
import datetime as dt #import datetime module
import seaborn as sns #import seaborn
from collections import Counter #import Counter
```

In [2]:
```
#lambda function for sqldf for use later
pysqldf = lambda q: sqldf(q, globals())
```

The Data Sources

Box Office Mojo Dataset

The **Box Office Mojo** dataset is stored in the `zippedData` folder.

The file is called `bom.movie_gross.csv.gz` .

We can see from the extension that this dataset is a `CSV` file therefore we will use the `read_csv()` function of pandas to explore this dataset.

In [3]:
```
bom_df = pd.read_csv('zippedData/bom.movie_gross.csv.gz') # reading the

bom_df.head() # Previewing the data
```

Out[3]:

	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

Lets look at the overview of the data frame using the `.info()` function

In [4]:
```
bom_df.info() # getting the overvieww info of the dataframe records
```

```
<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
```

There are **five** columns in this dataset:

- `title` : This is the movie title. It is an important column because it shows which movie the record is for. This will be valuable in comparing/merging data to give more context to the analysis.

- `studio` : This gives an outlook on the competitors in the market. This won't be valuabale in answering any of the three business questions.

- `domestic_gross` : This data is particularly important because it provides the financial overview within the US market. Since there is no currency sign associated, we will assume `$` but more on that in the **Data Prepartion** section.

- `foreign_gross` : This is similar to `domestic gross` and equally valuable. It provides insights on foreign performance of a movie.

- `year` : This is the year that the movie was released. It is important to keep the year in mind so that the analysis isn't skewed. We will later evaluate which timelines to focus on for the analysis.

There are `3387` records in total in this dataset which is a good size for our analysis.

Nonetheless, lets explore the distribution of the `year` column to understand the spread of the data to ensure that the timeline we want to look isn't skewed.

In [5]:
```
#get the counts for the unique values in the year column
bom_unique_value_counts = bom_df['year'].value_counts(sort=False)

print(bom_unique_value_counts) #print the values
```

```
2010    328
2011    399
2012    400
2013    350
2014    395
2015    450
2016    436
2017    321
2018    308
Name: year, dtype: int64
```

We can see that the data is between **2010** and **2018** with a very good distribution. This allows us to choose our timelines without fear of loosing data or skewing the results.

Referring back to the `.info()`, this dataset has some values missing.

- `studio` column has 5 missing values which we can drop since it will have an insignificant on impact the total number of records but we will take a look at the records we are dropping in the **Data Preparation** section.

- `foreign gross` column has more than 1000 missing values. We will correlate this with data we find from other datasets to understand whether these movies weren't launched in foregin markets or is there a gap in our data. We will explore this more in the **Data Preparation** section.

Summary:

The **Box Office Mojo** data will be valuable to conduct financial analysis but we have another financial informatiion dataset that we'll look at next. We will compare these two datasets to understand what information to carry into our analysis from these.

The Numbers Dataset

This dataset is stored in the `zippedData` folder.

The file is called `tn.movie_budgets.csv.gz` .

We can see from the extension that this dataset is a `CSV` file therefore we will use the `pd.read_csv()` function of pandas to explore this dataset.

In [6]:
```
tn_df = pd.read_csv('zippedData/tn.movie_budgets.csv.gz') # reading the

tn_df.head() # Previewing the data
```

Out[6]:

	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

Lets continue to explore further with the `.info()` function

In [7]:
```
1 tn_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5782 entries, 0 to 5781
Data columns (total 6 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
dtypes: int64(1), object(5)
memory usage: 271.2+ KB
```

There are **six** columns in this dataset with a total of `5782` records with no missing values.

Lets explore the id column to understand more about it

In [8]:
```
1 # getting the counts for the unique values in the id column
2 tn_df['id'].value_counts()
```

Out[8]:
```
1      58
63     58
61     58
60     58
59     58
       ..
86     57
85     57
84     57
83     57
100    57
Name: id, Length: 100, dtype: int64
```

It is difficult to guage what the `id` column is doing in this dataset. We should be able to drop this column since it doesn't signify anything unique. This will be done in the **Data Preparation** section.

Lets check whether the `worldwide_gross` column is a sum of `domestic_gross` and foreign revenue or does it only signify foreign revenue. For this we will compare values of the same movies between `bom_df` and `tn_df`

In [9]:
```
# extracting the Avengers record of tn_df
avengers_tn = tn_df.iloc[3]

# extracting the Avengers record of bom_df by setting
# the index to title and searching for the specific title

avengers_bom = bom_df.set_index('title').loc['Avengers: Age of Ultron']

# printing the respective records
print('tn_df avengers record:\n', avengers_tn)
print('\nbom_df avengers record:\n', avengers_bom)
```

```
tn_df avengers record:
 id                                          4
release_date                      May 1, 2015
movie                 Avengers: Age of Ultron
production_budget                $330,600,000
domestic_gross                   $459,005,868
worldwide_gross                $1,403,013,963
Name: 3, dtype: object

bom_df avengers record:
 studio                       BV
domestic_gross    459000000.0
foreign_gross       946400000
year                     2015
Name: Avengers: Age of Ultron, dtype: object
```

The `tn_df` records have more significant figures than the `bom_df` record. Lets round off the sum and see whether they are similar.

In [10]:
```
# convert the bom_df record to integer data type
avengers_bom['domestic_gross'] = int(avengers_bom['domestic_gross'])
avengers_bom['foreign_gross'] = int(avengers_bom['foreign_gross'])

# replace the tn_df record string components "," and "$"
avengers_tn['worldwide_gross'] = avengers_tn['worldwide_gross'].replace

#convert tn_df record to integer
avengers_tn['worldwide_gross'] = int(avengers_tn['worldwide_gross'])

#round off the numbers to the nearest million
x1 = round(avengers_bom['domestic_gross'] + avengers_bom['foreign_gross
x2 = round(avengers_tn['worldwide_gross'],-8)

#check if the values are the same
x1 == x2
# round(avengers_bom['domestic_gross'].astype('int32')
#       + int(avengers_bom['foreign_gross']),-8)
# == round(int(avengers_tn['worldwide_gross'].replace(',','' ).replace(
```

```
/var/folders/qv/0z2v23tn1f1b2fnpqqgqsxch0000gn/T/ipykernel_15142/29723243
31.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

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)
  avengers_tn['worldwide_gross'] = avengers_tn['worldwide_gross'].replace
(',','' ).replace('$','')
/var/folders/qv/0z2v23tn1f1b2fnpqqgqsxch0000gn/T/ipykernel_15142/29723243
31.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

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)
  avengers_tn['worldwide_gross'] = int(avengers_tn['worldwide_gross'])
```

Out[10]:
```
True
```

From the above True statement we can deduce that the `worldwide_gross` in `tn_df` is indeed the sum of `domestic_gross` and `foreign_gross` of the `bom_df` dataframe.

With that in mind, a brief overview of the columns of interest in the `tn_df` dataframe is as follows:

- `release_date` : This is the release date of the movie and will be valuable in determining if we wwant to limit the timeline of the dataset we use.
- `movie` : This is the movie name which shows which movie the record is for.
- `production_budget` : This column is important in understanding the initial investments that have to be made by the client. This will also allow us to calculate the net profits.

- `domestic-gross` : Similar to `bom_df` , this column provides insight into the revenue generated in the US market by the movie.

- `worldwide-gross` : This is slightly different than the `foreign_gross` column of `bom_df` because it adds up the domestic gross of the movie and provides the total revenue for the movie.

An important thing to note is that `tn_df` has `5782` records while `bom_df` has `3378` .

Summary:

`tn_df` has similar data bins (i.e. columns) as `bom_df` but more records. Considering that `tn_df` has more records and a complete dataset, therefore it would be better to use this dataset for analysis.

`tn_df` will be the main dataset we will be using to conduct financial analysis.

Rotten Tomatoes Dataset

There are 2 datasets from Rotten Tomatoes that we are going to explore and understand their utility for our analysis.

- `rt.movie_info.tsv`
- `rt.reviews.tsv`

Both of these files are stored in the `zippedData` folder.

We can see from the file extensions that these datasets are `TSV` (tab separated values) files. We will use the same `.read_csv()` function with the parameter `sep = '\t'` .

In [11]:
```
# store the rotten tomatoes movie info dataset in a pandas dataframe
rtmov_df = pd.read_csv('zippedData/rt.movie_info.tsv', '\t')

#previewing the dataframe
rtmov_df.head()

```

```
/opt/anaconda3/lib/python3.9/site-packages/IPython/core/interactiveshell.
py:3369: FutureWarning: In a future version of pandas all arguments of re
ad_csv except for the argument 'filepath_or_buffer' will be keyword-only.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

Out[11]:

	id	synopsis	rating	genre	director	writer	theater_date	dvd_
0	1	This gritty, fast-paced, and innovative police...	R	Action and Adventure|Classics|Drama	William Friedkin	Ernest Tidyman	Oct 9, 1971	Se
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	J
2	5	Illeana Douglas delivers a superb performance ...	R	Drama|Musical and Performing Arts	Allison Anders	Allison Anders	Sep 13, 1996	Ap
3	6	Michael Douglas runs afoul of a treacherous su...	R	Drama|Mystery and Suspense	Barry Levinson	Paul Attanasio|Michael Crichton	Dec 9, 1994	Au
4	7	NaN	NR	Drama|Romance	Rodney Bennett	Giles Cooper	NaN	

In [12]:
```
# store the rotten tomatoes reviews datset in a pandas dataframe
rtrev_df = pd.read_csv('zippedData/rt.reviews.tsv', '\t', encoding='win

#previewing the dataframe
rtrev_df.head()
```

```
/opt/anaconda3/lib/python3.9/site-packages/IPython/core/interactiveshell.
py:3369: FutureWarning: In a future version of pandas all arguments of re
ad_csv except for the argument 'filepath_or_buffer' will be keyword-only.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

Out[12]:

	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

Lets continue to explore further with the `.info()` function

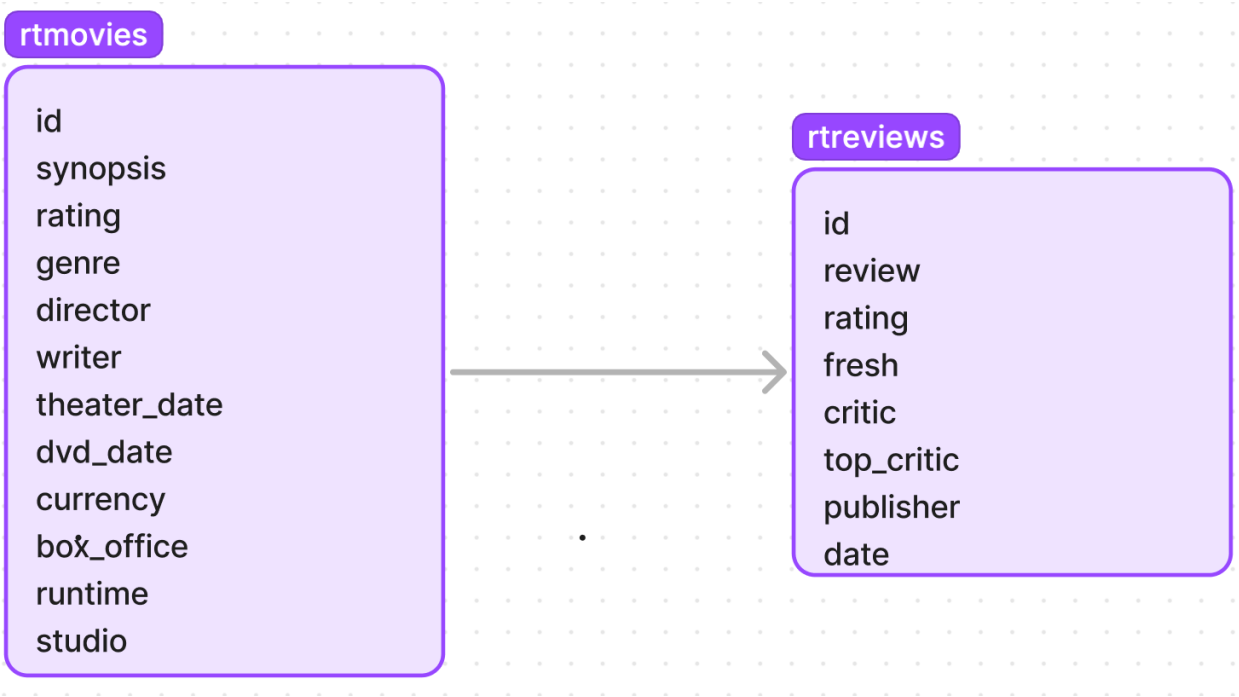
In [13]:
```
print(rtmov_df.info(),'\n')
print('\n', rtrev_df.info())
```

```
<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
None

<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

 None
```

The two datasets are related through the id column as shown below. Since there are multiple reviews for movies, we see a big discrepancy between the total number of records between the two datasets.

`rtmov_df` has `1560` records

`rtrev_df` has `54432` records



Since there is no information on the movie name that these records are for, it becomes very difficult to join/compare this data with other datasets.

Note that the `rating` columns in the two dataframes show different data. The one in `rtmov_df` is to signify what age group the movie was for (eg. R, PG13, etc) whereas the one in `rtrev_df` is the movie review on a scale of 0 to 5.

The columns of interest are described below:

- `id` : This is the key column in both dataframes. We will be using this column to join the dataframes.
- `rtmov_df` `rating` : This column contains information on the age group that the movie was released for. For example R-rated movies have age restrictions. This will be valuable in understanding the demographic of the users.
- `rtmov_df` `genre` : This column provides an insight into what genre the movie was. Comparing this column with `rating` and `reviews` will showcase which genres are highly competetive and which ones are low performingor have high standards.
- `rtrev_df` `rating` : This column stores the ratings reviewers gave the movie out of 5. This data will be used as the basepoint to compare what type of movies are successful. This column will be renamed to avoid confusion with the `rtmov rating` data.

An important thing to note is that the **Rotten Romatoes** data only has `1560` movie records. This is much smaller than the other two data sources we have looked at.

Therefore, if we find a dataset with similar information but more records, we will ignore this dataset.

Summary:

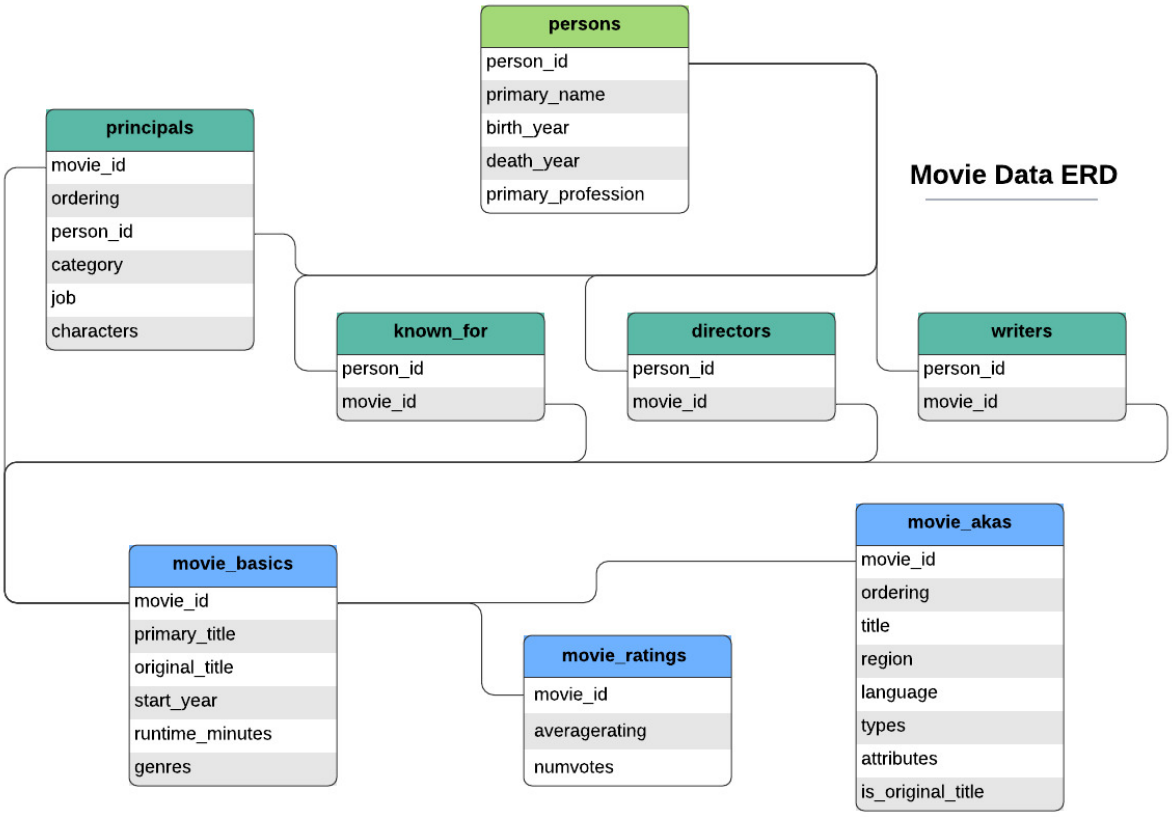
While this data is valuable for analysing genres vs reviews, we need to look at the other datasets with more records before deciding how critical the rotten tomatoes data will be.

IMDB Database

The **IMDB** Database is stored in the `zippedData` folder.

The file is called `im.db` .

We have the following Entity Relationship Diagram (ERD) explaining the different tables in the database:



Lets explore the tables of the database. We will start with `persons` .

In [14]:
```
1 conn = sql.connect('zippedData/im.db') # making a conenction to the dat
```

In [15]:
```
# writing the sql query for the persons table
q1 = """
SELECT *
FROM persons
"""

# querying the persons table and storing it in a dataframe
im_persons_df = pd.read_sql(q1,conn)

# previewing the data and table info
print(im_persons_df.info())
im_persons_df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 606648 entries, 0 to 606647
Data columns (total 5 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   person_id          606648 non-null  object 
 1   primary_name       606648 non-null  object 
 2   birth_year         82736 non-null   float64
 3   death_year         6783 non-null    float64
 4   primary_profession  555308 non-null  object 
dtypes: float64(2), object(3)
memory usage: 23.1+ MB
None
```

Out[15]:

	person_id	primary_name	birth_year	death_year	primary_professio
0	nm0061671	Mary Ellen Bauder	NaN	NaN	miscellaneous,production_manager,produce
1	nm0061865	Joseph Bauer	NaN	NaN	composer,music_department,sound_departmen
2	nm0062070	Bruce Baum	NaN	NaN	miscellaneous,actor,write
3	nm0062195	Axel Baumann	NaN	NaN	camera_department,cinematographer,art_departmen
4	nm0062798	Pete Baxter	NaN	NaN	production_designer,art_department,set_decorato

These are the records for the cast and crew of the movies which are not relevant to the business questions that this notebook explores.

Nonetheless, this is something that should be investigated separately. Alongside this data, experts in the field of movie production should be consulted to understand which cast and crew would be ideal for the kind of movie that the client decides to pursue.

Leveraging the **ERD** above, we can skip exploring `writers` , `directors` , `principals` and `known_for` tables because these are information about the cast and crew.

We can move on to exploring the `movie_basic` table.

In [16]:
```
# writing the sql query for the movie_basics table
q2 = """
SELECT *
FROM movie_basics
"""

# querying the table and storing it in a dataframe
im_movbas_df = pd.read_sql(q2,conn)

# previewing the data and table info
print(im_movbas_df.info())
im_movbas_df.head()
```

```
<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
```

Out[16]:

	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

This is one of the most important tables in this database. It stores the primary information about the movies.

It has **six** columns, all with valuable information, with `146144` records. This will improve the analysis.

- `movie_id` : This is the primary key of the table. It can be seen in the ERD that it relates to the other tables based on this id.

- `primary_title` : This column represents the changed title for different markets. These can be translations if the movie is a foreign production. A lot of movies are released with different names in different regions. This might end up not being a useful column for analysis but further exploration will calrify that.

- `original_title` : This column is self-explanatory. It has the original title of the movie in the local language of the region it was prooduced in. This will be important to link to the other datasets and other tables within the database.

- `start_year` : This is the year that the movie was launched. This will be important to filter the records for the timeline being evaluated.

- `runtime_minutes` : This column contains the data on the total length of the movie. This data does not contribute to the business questions being answered.

- `genres` : This column contains the genres of the movie. This will be critical in our analysis to understand which genres should the client focus on.

There are some columns with missing data for some columns but they are small in number as compared to the overall records and we should be able to adjust the dataframe for effective usage in the **Data Preparation** section.

Before we move on to the next table, lets explore the range and spread of `start_year` to understand what timeline data we have available.

In [17]:
```
1 im_movbas_df['start_year'].value_counts(sort = False) #exploring the un
```

Out[17]:
```
2013    14709
2019     8379
2018    16849
2017    17504
2012    13787
2010    11849
2011    12900
2015    16243
2021       83
2016    17272
2014    15589
2020      937
2022       32
2023        5
2024        2
2026        1
2025        1
2115        1
2027        1
Name: start_year, dtype: int64
```

- There is a good spread from 2010 - 2019.
- There is very few data available for 2020-2022 as compared to 2010-2019.
- This column will be filtered based on the years we have financial data available.

- There also seem to be some outlandish years in there which we will drop.

Summary:

The `movie_basics` table give us the principal information about the movies that will be used for joining and relating with other tables. It will be important to use this table in analysis to relate reviews and profits to genres.

Lets explore the `movie_ratings` table.

In [18]:
```
# writing the sql query for the movie_ratings table
q3 = """
SELECT *
FROM movie_ratings
"""

# querying the table and storing it in a dataframe
im_movrat_df = pd.read_sql(q3,conn)

# previewing the data and table info
print(im_movrat_df.info())
im_movrat_df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 73856 entries, 0 to 73855
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   movie_id       73856 non-null  object 
 1   averagerating  73856 non-null  float64
 2   numvotes       73856 non-null  int64  
dtypes: float64(1), int64(1), object(1)
memory usage: 1.7+ MB
None
```

Out[18]:

	movie_id	averagerating	numvotes
0	tt10356526	8.3	31
1	tt10384606	8.9	559
2	tt1042974	6.4	20
3	tt1043726	4.2	50352
4	tt1060240	6.5	21

This is the dataframe storing the average reviews for the movies and the corresponding total number of votes for each movie.

Nonetheless, it only has `73856` records compared to `movie_basics` `146144` records. Only movies that have reviews data will be used in the analysis.

These reviews can be correlated with the genres of the movies from `movie_basics` using the `movie_id` column. This will give us a good comparison of genres vs reviews.

These can be further analysed with profits for every movie, resulting in a genres vs reviews vs profits comparison.

This dataframe has **three** columns:

- `movie_id` : This is the key that is relating this table to the `movie_basics` table.
- `averagerating` : This is the average of all the ratings for a specific movie.
- `numvotes` : This is the total number of reviews that a movie received. It will be important to set a baseline of minimum number of votes to use for filtering the data to avoid skewwing results.

Summary:

This table will be extremely valuable in analyzing genres vs reviews and generating recommendations for which type of movie should the client pursue.

Moving on to the other table related to `movie_basics` in the ERD, `movie_akas` .

In [19]:
```
# writing the sql query for the movie_akas table
q4 = """
SELECT *
FROM movie_akas
"""

# querying the table and storing it in a dataframe
im_movaka_df = pd.read_sql(q4,conn)

# previewing the data and table info
print(im_movaka_df.info())
im_movaka_df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 331703 entries, 0 to 331702
Data columns (total 8 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   movie_id           331703 non-null  object 
 1   ordering           331703 non-null  int64  
 2   title              331703 non-null  object 
 3   region             278410 non-null  object 
 4   language           41715 non-null   object 
 5   types              168447 non-null  object 
 6   attributes         14925 non-null   object 
 7   is_original_title  331678 non-null  float64
dtypes: float64(1), int64(1), object(6)
memory usage: 20.2+ MB
None
```

Out[19]:

	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
3	tt0369610	13	O Mundo dos Dinossauros	BR	None	None	short title	0.0
4	tt0369610	14	Jurassic World	FR	None	imdbDisplay	None	0.0

This is a relatively large dataframe with `331703` records. This is because the `title` column has separate names that movies had in different regions.

We will use the laanguage column to identify which movies were produced for most of the US population.

Using **English** as the baseline laanguage will allow the movie to have the greatest reach in doemstic and international markets.

Lets explore whether `types` and `attributes` column will be useful for our analysis.

In [20]:
```
1 im_movaka_df['types'].value_counts() # exploring the unique value count
```

Out[20]:
```
imdbDisplay          100461
original              44700
working                8680
alternative            6564
festival               3307
dvd                    2995
tv                     1617
video                   121
dvd imdbDisplay           1
festival working          1
Name: types, dtype: int64
```

In [21]:
```
1 im_movaka_df['attributes'].value_counts() # exploring the unique value
```

Out[21]:
```
new title                       1700
alternative spelling            1394
literal English title           1054
complete title                  1034
original subtitled version       879
                                ...
8mm release title                  1
reissue title short version        1
first season title                 1
TV listings title                  1
X-rated version                    1
Name: attributes, Length: 77, dtype: int64
```

Looks like both of these columns won't be of any use in our analysis so we'll drop them later.

Lets take a look at how many original titles we have and which regions they belong to

In [22]:
```
1 im_movaka_df.value_counts('is_original_title') # checking the value_cou
```

Out[22]:
```
is_original_title
0.0    286978
1.0     44700
dtype: int64
```

In [23]:
```
# filtering for original titles
im_movaka_df[im_movaka_df['is_original_title'] == 1.0]
```

Out[23]:

	movie_id	ordering	title	region	language	types	attributes	is_original_title
38	tt0369610	45	Jurassic World	None	None	original	None	1.0
80	tt0401729	7	John Carter	None	None	original	None	1.0
83	tt10010134	1	Versailles Rediscovered - The Sun King's Vanis...	None	None	original	None	1.0
86	tt10027708	1	Miguelito - Canto a Borinquen	None	None	original	None	1.0
90	tt10050722	1	Thing I Don't Get	None	None	original	None	1.0
...	...	...	...	...	...	...	...	...
331690	tt9723084	2	Anderswo. Allein in Afrika	None	None	original	None	1.0
331692	tt9726638	2	Monkey King: The Volcano	None	None	original	None	1.0
331696	tt9755806	3	Big Shark	None	None	original	None	1.0
331698	tt9827784	2	Sayonara kuchibiru	None	None	original	None	1.0
331700	tt9880178	1	La atención	None	None	original	None	1.0

44700 rows × 8 columns

In [24]:
```
#finding the value_counts for the region and language columns

print('The number of original titles in the dataset is:', len(im_movaka

print(im_movaka_df[im_movaka_df['is_original_title'] == 1.0]['region'].

print(im_movaka_df[im_movaka_df['is_original_title'] == 1.0]['language'

```

```
The number of original titles in the dataset is: 44700

US     3
XWW    2
CN     1
Name: region, dtype: int64

en     3
cmn    1
Name: language, dtype: int64
```

The are `44700` original titles in this dataset but almost all of them are missing region and lanaguage data.

This renders this table invaluable for our analysis since it doesn't have any extra information that will help answer the **Business Questions**

That finishes the exploration of the `im.db` database.

Summary:

- `persons` , `principals` , `known_for` , `directors` and `writers` contain information on cast and crew. These tables will not be used in the analysis because they do not add value to answering the **Business Questions** laid out in the **Problem Overview** section.
- `movie_basics` is an important table wih the principal information about the movie. This table is linked to two other important tables and will be critical in the analysis.
- `movie_ratings` stores the average ratings and number of votes for movies. This will serve as the main reference for analyses based on reviews.
- `movie_akas` contains movie names in different regions. Because of missing values, it doesn't provide us any valuable data to use in our analysis. This table will also be ignored.

The MovieDB Dataset

The MovieDB Dataset is stored in the **zippedData** folder.

The file is called `tmdb.movies.csv` .

We can see from the file extension that this is a `CSV` file. We will use pandas `.read_csv()` function to explore this dataset.

In [25]:
```
tmdb_df = pd.read_csv('zippedData/tmdb.movies.csv') # extracting the da

# previewing the info and the dataframe
print(tmdb_df.info(), '\n')

tmdb_df.head()
```

```
<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
None
```

Out[25]:

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

There are **ten** columns and `26517` records in this dataset.

- The `Unnamed` column is the index of the records therefore this column can be ignored.

- The `genre_ids` column can be ignored since we can grab the genres from other tables which would be more helpful
- The `id` column can be dropped because we can use `original_title` as the key to relate to the `im.db` database.

The columns that we will be carrying into analysis will be the following:

- The `original_language` column can be used to determine which movies will have the greatest reach. Using **English** as a the baseline language, will allow the maximum domestic and international reach.
- `original-title` : This is the original title of the movie which we can use to correlate data in other dataframes.
- `popularity` : This will be an important data to look at alongside reviews to analyse the response to the movies. While the scale isn't known, we can manipulate the values to create a custom scale to better understand the values.
- The `release_date` column can be used to filter records for the timeline that will be used in analsis.
- `title` : This will be an important column to keep in case we need to link the `bommovies` dataset using the title column instead of `original_title` .
- `vote_average` : This gives us an extra benchmark to use for comparison of reviews to the `imm.db` database
- `vote_count` : The number of votes allows us to filter to ensure that wew are not skewing data because of very good or very bad reviews of a only a handful viewers.

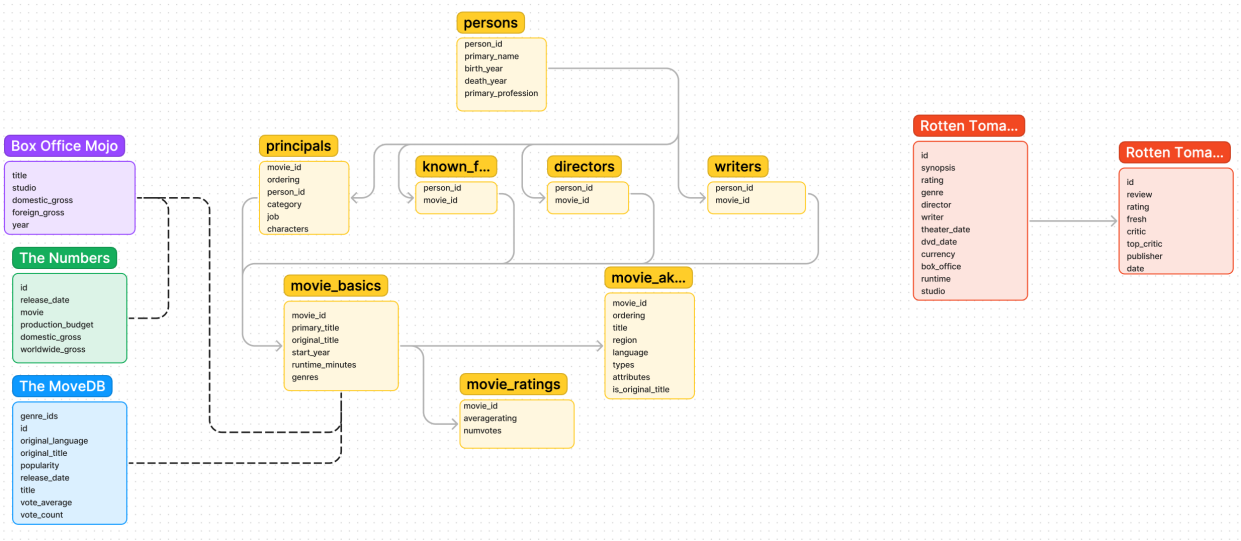
Summmary:

The `tmdb_db` dataset will be very useful for analyzing movie reviews vs proftis vs genres. We can directly link this dataset to the `im_db` tables using the `original_title` column.

Review

That was a lot of information. Lets take an overall look at the ERD of all of our datasets and talk about which ones we will be using in our analysis, and how we'll use them.

In the diagram below, yellow is for the **IMDB** database. Everything else is labelled.



`im.db` database is the most valuable database which has all the info for the movies except for revenues.

- `persons`, `principals`, `known_for`, `directors`, `writers`, and `movie_akas` datasets will be ignored because they are not relevant to the **Business Questions** mentioned earlier.

- `movie_basics` will be the main dataset used for the principal information for the movies, i.e genre, title, etc.

- `movie_ratings` will be used for tallying up the reviews and correlating genre with review and financial performance.

`Box Office Mojo` and `The Numbers` datasets are next most valuable ones because they contain information on financial performance of the movies.

- `Box Office Mojo` and `The Numbers` data contains the same financial information.

- Since `The Numbers` dataset has more records, the `Box Office Mojo` dataset will be ignored. `The Number` dataset will be used to calculate the profits for financial analysis.

- The genre, reviews, and financial performance data will be joined with the imdb dataset. The joined dataset will be the main financial analysis daataframe that this notebook will use.

- The timeline available in `The Numbers` dataset will be one of the limiting factors for analysis.

`The MovieDB` datasets will serve as an additional resource to compare with `movie_ratings`. The dataset wwill be joined with the `im.db` database and the reviews will be averaged between the two.

The `Rotten Tomatoes` datasets will be ignored because they don't provide any additional value

for analysing customer reviews. `im.db` provides movie-specific data with more records.

Forewarning

Although the data and analysis we are conducting will provide us impactful insights, there are still blindspots in the analysis and gaps in the data that should be kept in mind when making any final decisions.

- We are not conidering client's company's internal factors
- We are not considering outside influences that can impact the data trend. For instance, social movements, world crisis, inflation, recession, etc
- None of the financial numbers have been adjusted for inflation
- No individual analysis of how much money was spent in marketing and advertising and that plays a part in the success of the movie
- We are not considering the recent shift to streaming platforms and how much extra revenue this brings in. Also it is possible that some movies never made it to the box-office or weren't boox-office hits and made most of the profits through streaming services.
- We are not accounting for the demographic of the reviewers which can provide greater insight and show any bias in the data
- There is no demographic data being analysed to identify a specific target population for the client
- We are only looking at US-based movies and movie houses
- The latest financial data we are looking at is for 2018 which is still 4 years old and doesn't account for the latest trends.
- The data, analysis and recommendations will be based on historical figures. There is no analysis done on changing trends across multiple fields to predict future trends.

Data Preparation

The following steps will be followed in preparing the data:

- Data cleaning
- Data Structuring
- Data Organizing

IMDB Database

movie_basics

In [26]:
```
1 im_movbas_df.head() # preview of dataframe
```

Out[26]:

	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

In [27]:
```
1 im_movbas_df.info() #check for missing values
```

```
<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
```

Lets drop the `runtime_minutes` column because we won't be using it in analysis.

In [28]:
```
1 mov_bas = im_movbas_df.drop(['runtime_minutes'], axis = 1) # dropping t
```

In [29]:
```
1 mov_bas.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 146144 entries, 0 to 146143
Data columns (total 5 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   genres          140736 non-null  object
dtypes: int64(1), object(4)
memory usage: 5.6+ MB
```

Since the numbers of records with missing values is very few as compared to the total number oof records, the missing records can be dropped.

In [30]:
```
1 mov_bas = mov_bas.dropna() # dropping the rows with missing data
2 
3 mov_bas.info() # check to make sure no missing values left over
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 140734 entries, 0 to 146143
Data columns (total 5 columns):
 #   Column          Non-Null Count   Dtype 
---  ------          --------------   ----- 
 0   movie_id        140734 non-null  object
 1   primary_title   140734 non-null  object
 2   original_title  140734 non-null  object
 3   start_year      140734 non-null  int64 
 4   genres          140734 non-null  object
dtypes: int64(1), object(4)
memory usage: 6.4+ MB
```

Now we will check for any duplicate titles or movie ids.

In [31]:
```
1 mov_bas['movie_id'].duplicated().value_counts() # check movie_id column
```

Out[31]:
```
False    140734
Name: movie_id, dtype: int64
```

In [32]:
```
1 mov_bas['primary_title'].duplicated().value_counts()# check primary_tit
```

Out[32]:
```
False    131334
True       9400
Name: primary_title, dtype: int64
```

In [33]:
```
mov_bas[mov_bas['primary_title'].duplicated(keep=False)].sort_values('p
```

Out[33]:

	movie_id	primary_title	original_title	start_year	genres
131857	tt8219776	#5	#5	2018	Documentary
52892	tt3120962	#5	#5	2013	Biography,Comedy,Fantasy
106201	tt6214664	(aguirre)	(aguirre)	2016	Biography,Comedy,Documentary
103890	tt6085916	(aguirre)	(aguirre)	2016	Biography,Documentary
100818	tt5891614	1	1	2016	Documentary
...	...	...	...	...	...
66989	tt3815122	Ângelo de Sousa - Tudo o Que Sou Capaz	Ângelo de Sousa - Tudo o Que Sou Capaz	2010	Biography,Documentary
37636	tt2362758	Éden	Éden	2013	Drama
23712	tt1961689	Éden	Éden	2011	Documentary
93912	tt5471216	Ódio	Ódio	2017	Action
98200	tt5737878	Ódio	Ódio	2016	Drama

15214 rows × 5 columns

Looks like some of these have the same primary title but are different movies. Lets check for records with identical values in all columns except for movie_id since we have previously checked that `movie_id` column has all unique values.

Duplicate records should have all the values same except for `movie_id`

In [34]:
```
# storing the column names we want to check in a separate list
col_chk = mov_bas.columns[1:]

#print the column names to check whether these are the ones we want
print(col_chk)

# check for duplicates on the other 5 columns
mov_bas.duplicated(col_chk, keep=False).value_counts()

```

```
Index(['primary_title', 'original_title', 'start_year', 'genres'], dtype
='object')
```

Out[34]:
```
False    140204
True        530
dtype: int64
```

In [35]:
```
# look at duplicate records and sort them by primary title for a previe

mov_bas[mov_bas.duplicated(col_chk, keep=False)].sort_values('primary_t
```

Out[35]:

	movie_id	primary_title	original_title	start_year	genres
129962	tt8032828	100 Milioni di bracciate	100 Milioni di bracciate	2017	Biography
129979	tt8034014	100 Milioni di bracciate	100 Milioni di bracciate	2017	Biography
145118	tt9773302	3. Elma	3. Elma	2014	Drama
144392	tt9660588	3. Elma	3. Elma	2014	Drama
144337	tt9653930	3. Elma	3. Elma	2014	Drama
...	...	...	...	...	...
66992	tt3815128	Ângelo de Sousa - Tudo o Que Sou Capaz	Ângelo de Sousa - Tudo o Que Sou Capaz	2010	Biography,Documentary
66993	tt3815130	Ângelo de Sousa - Tudo o Que Sou Capaz	Ângelo de Sousa - Tudo o Que Sou Capaz	2010	Biography,Documentary
66994	tt3815132	Ângelo de Sousa - Tudo o Que Sou Capaz	Ângelo de Sousa - Tudo o Que Sou Capaz	2010	Biography,Documentary
66995	tt3815134	Ângelo de Sousa - Tudo o Que Sou Capaz	Ângelo de Sousa - Tudo o Que Sou Capaz	2010	Biography,Documentary
66991	tt3815126	Ângelo de Sousa - Tudo o Que Sou Capaz	Ângelo de Sousa - Tudo o Que Sou Capaz	2010	Biography,Documentary

530 rows × 5 columns

These records look more like the exact duplicates. We will keep the first ones and get rid of the rest.

In [36]:
```
# dropping duplicate records but keeping the first occurance

mov_bas.drop_duplicates(col_chk, keep = 'first', inplace=True)

```

In [37]:
```
# check to see if any duplicates left
mov_bas.duplicated(col_chk, keep=False).value_counts()
```

Out[37]:
```
False    140460
dtype: int64
```

We saw that the `genres` column has different number of values for each movie. Lets split this into separate columns to make it easier for analyzinng genre groupings.

We will keep the original genres column for comparison with data that will be joined.

Notice that there are no spaces between the multiple genres. They are only separated by `,`.

We will strip the data of any spaces `" "` and then split it using `,` . This will be stored in a separate dataframe and then copied over into `mov_bas` so that the original data isn't corrupted.

In [38]:
```
# strip the white space from genres and split it usinng ','
genre_split = mov_bas['genres'].str.strip(" ").str.split(',', expand=Tr

# preview the new dataframe to check if the code above worked
print(genre_split.info(), '\n')
genre_split.head()

```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 140460 entries, 0 to 146143
Data columns (total 3 columns):
 #   Column  Non-Null Count   Dtype 
---  ------  --------------   ----- 
 0   0       140460 non-null  object
 1   1       59329 non-null   object
 2   2       29421 non-null   object
dtypes: object(3)
memory usage: 4.3+ MB
None
```

Out[38]:

	0	1	2
0	Action	Crime	Drama
1	Biography	Drama	None
2	Drama	None	None

We have the same number of records as the `mov_bas` dataframe and 3 columns. This means that the total number of new columns we need to add to `mov_bas` is **three**

In [39]:
```
#create new columnns in mov_bas with the values from genre_split
mov_bas['genre1'] = genre_split[0]

mov_bas['genre2'] = genre_split[1]

mov_bas['genre3'] = genre_split[2]

#preview the updated dataframe
mov_bas.head()
```

Out[39]:

	movie_id	primary_title	original_title	start_year	genres	genre1	genre2	genre
0	tt0063540	Sunghursh	Sunghursh	2013	Action,Crime,Drama	Action	Crime	Dram
1	tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	Biography,Drama	Biography	Drama	Nor
2	tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018	Drama	Drama	None	Nor
3	tt0069204	Sabse Bada Sukh	Sabse Bada Sukh	2018	Comedy,Drama	Comedy	Drama	Nor
4	tt0100275	The Wandering Soap Opera	La Telenovela Errante	2017	Comedy,Drama,Fantasy	Comedy	Drama	Fantas

Lets replace the `None` values with `NA` so that we can keep a track of them of our analysis

In [40]:
```
mov_bas = mov_bas.fillna('NA') #filling missing values with 'NA'
```

Lets preview the results before moving on to the next table

In [41]:
```
mov_bas.head()
```

Out[41]:

	movie_id	primary_title	original_title	start_year	genres	genre1	genre2	genre
0	tt0063540	Sunghursh	Sunghursh	2013	Action,Crime,Drama	Action	Crime	Dram
1	tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	Biography,Drama	Biography	Drama	N
2	tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018	Drama	Drama	NA	N
3	tt0069204	Sabse Bada Sukh	Sabse Bada Sukh	2018	Comedy,Drama	Comedy	Drama	N
4	tt0100275	The Wandering Soap Opera	La Telenovela Errante	2017	Comedy,Drama,Fantasy	Comedy	Drama	Fantas

movie_ratings

In [42]:
```
1 im_movrat_df.head() # preview the data
```

Out[42]:

	movie_id	averagerating	numvotes
0	tt10356526	8.3	31
1	tt10384606	8.9	559
2	tt1042974	6.4	20
3	tt1043726	4.2	50352
4	tt1060240	6.5	21

In [43]:
```
1 im_movrat_df.info() # get dataframe info
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 73856 entries, 0 to 73855
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   movie_id       73856 non-null  object 
 1   averagerating  73856 non-null  float64
 2   numvotes       73856 non-null  int64  
dtypes: float64(1), int64(1), object(1)
memory usage: 1.7+ MB
```

There are not any missing values in this dataframe. Nonetheless it does have less records than `movie_basics`. This will become the limiting factor in how many records we have for analysis.

Lets go ahead and merge `mov_bas` and this record.

In [44]:
```
1 mov_rat = im_movrat_df.copy() # make a copy of the dataframe
```

In [45]:
```
1 # set index to movie_id for both tables
2
3 mov_bas = mov_bas.set_index('movie_id')
4
5 mov_rat = mov_rat.set_index('movie_id')
```

In [46]:
```
# join mov_bas and mov_rat using movie_id
mov_bas_rat = mov_bas.join(mov_rat, how = 'inner')

# preview results
print(mov_bas_rat.info(), '\n')
mov_bas_rat.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 72998 entries, tt0063540 to tt9916160
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   primary_title   72998 non-null  object 
 1   original_title  72998 non-null  object 
 2   start_year      72998 non-null  int64  
 3   genres          72998 non-null  object 
 4   genre1          72998 non-null  object 
 5   genre2          72998 non-null  object 
 6   genre3          72998 non-null  object 
 7   averagerating   72998 non-null  float64
 8   numvotes        72998 non-null  int64  
dtypes: float64(1), int64(2), object(6)
memory usage: 5.6+ MB
None
```

Out[46]:

movie_id	primary_title	original_title	start_year	genres	genre1	genre2	genre3
tt0063540	Sunghursh	Sunghursh	2013	Action,Crime,Drama	Action	Crime	Drama
tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	Biography,Drama	Biography	Drama	NA
tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018	Drama	Drama	NA	NA
tt0069204	Sabse Bada Sukh	Sabse Bada Sukh	2018	Comedy,Drama	Comedy	Drama	NA
tt0100275	The Wandering Soap Opera	La Telenovela Errante	2017	Comedy,Drama,Fantasy	Comedy	Drama	Fantasy

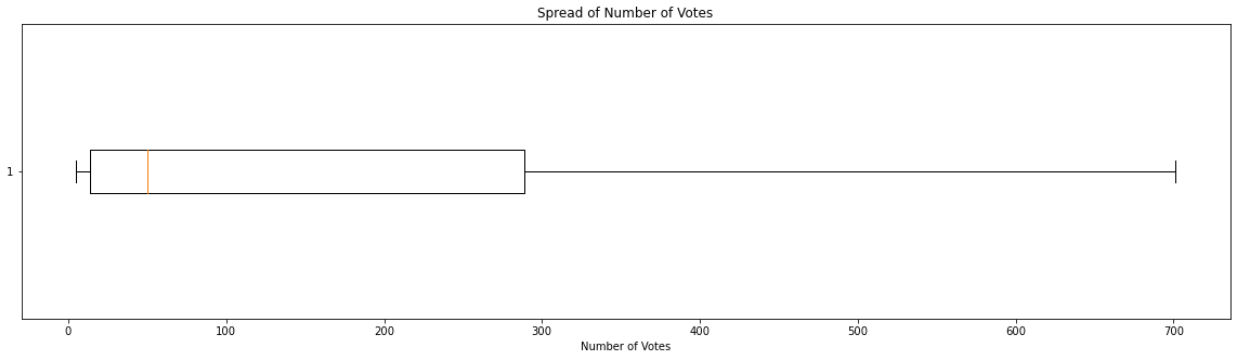
We need to decide on the minimum number of votes to use as a benchmark for our analysis.

It is widely known that not all viewers leave reviews. It is possible that the reviews can only be just from unsatisfied viewers or passionate fans. When the num of votes is less, it is possible that the rating is skewed to either to high or too low. For this purpose, we will look at a box and whisker plot to understand the range of the number of votes and then set a benchmark for filter the data.

We will not look at the outliers and decide what statistical value we want to use as the baseline. The outliers greater than will meet our criteria and the ones less than our baseline will be dropped therefore outliers cana be ignored in this analysis.

We will establish the number of votes baseline later when we have merged data from `The MovieDB` dataset but lets take a look at the box plot for the data in this dataframe to understand what the spread looks like.

In [47]:
```
fig1, ax1 = plt.subplots() # setting the structure

fig1.set_size_inches(20, 5, forward=True) # adjusting the size of the f

ax1.set_title('Spread of Number of Votes') # setting the title of the p

ax1.set_xlabel('Number of Votes') # setting the x-axis label


ax1.boxplot(mov_bas_rat['numvotes'], vert = False, showfliers=False);

```



We can see that the box plot is **right-skewed** which means that most movies do not get a lot of reviews. To establish a solid benchmark to use as analysis we will conduct further analysis to understand the spread of the reviews vs average rating after merging with **The MovieDB** dataset.

The MovieDB

In [48]:
```
# previewing the data and it's info
print(tmdb_df.info(), '\n')

tmdb_df.head()
```

```
<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
None
```

Out[48]:

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

Looks like this dataset doesn't have any missing values. It does have a few redundant columns that are not relevant (that we talked about earlier) to our analysis.

We will go ahead and drop them.

In [49]:
```
# make a copy of the dataframe
tmdb = tmdb_df.copy()

# drop irrelevant columns
tmdb = tmdb.drop(['Unnamed: 0', 'genre_ids', 'id', 'release_date', 'tit

#preview the data and info
print(tmdb.info(), '\n')

tmdb.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26517 entries, 0 to 26516
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   original_language  26517 non-null  object 
 1   original_title     26517 non-null  object 
 2   popularity         26517 non-null  float64
 3   vote_average       26517 non-null  float64
 4   vote_count         26517 non-null  int64  
dtypes: float64(2), int64(1), object(2)
memory usage: 1.0+ MB
None
```

Out[49]:

	original_language	original_title	popularity	vote_average	vote_count
0	en	Harry Potter and the Deathly Hallows: Part 1	33.533	7.7	10788

Next we are going to filter our data for the original language because we want to focus on analyzing movies that were made for a US audience.

We will first strip any white space and then check records for `en` which is the abbreviation for **English** in this dataset.

In [50]:
```
# filterting for english in original_language

tmdb = tmdb[tmdb['original_language'].str.strip(' ') == 'en']

# previewing the results
tmdb.head()
```

Out[50]:

	original_language	original_title	popularity	vote_average	vote_count
0	en	Harry Potter and the Deathly Hallows: Part 1	33.533	7.7	10788
1	en	How to Train Your Dragon	28.734	7.7	7610
2	en	Iron Man 2	28.515	6.8	12368
3	en	Toy Story	28.005	7.9	10174
4	en	Inception	27.920	8.3	22186

Now we'll merge this data with the `mov_bas_rat` dataframe to perform comparitive analysis of the two sets of reviews.

First we have to rename the columns in `tmdb` to make it easier to distinguish between the reviews of `mov_rat` and this dataset.

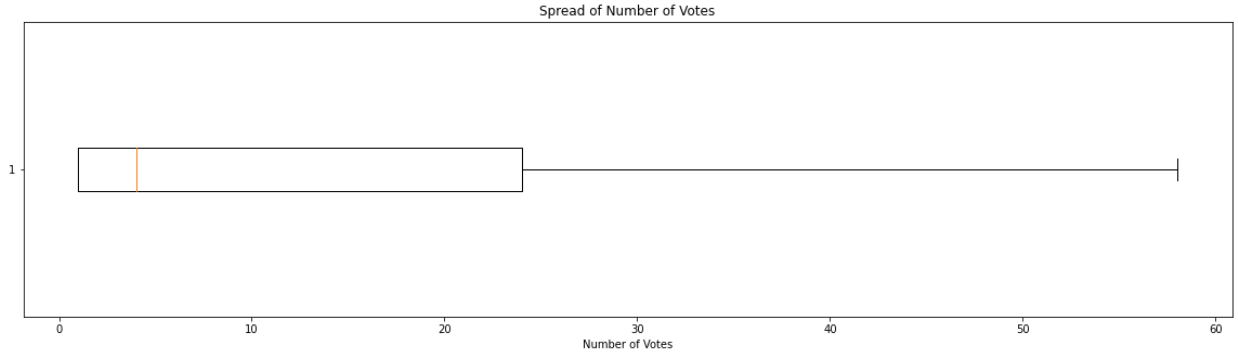
In [51]:
```
# rename column
tmdb = tmdb.rename(columns = {'vote_average' : 'tmdb_averagerating',
                              'vote_count' : 'tmdb_numvotes'})

# preview the results
tmdb.head()
```

Out[51]:

	original_language	original_title	popularity	tmdb_averagerating	tmdb_numvotes
0	en	Harry Potter and the Deathly Hallows: Part 1	33.533	7.7	10788
1	en	How to Train Your Dragon	28.734	7.7	7610
2	en	Iron Man 2	28.515	6.8	12368
3	en	Toy Story	28.005	7.9	10174
4	en	Inception	27.920	8.3	22186

Lets check the spread of the number of votes in this dataset to compare against the previous dataset.

In [52]:
```
fig2, ax2 = plt.subplots() # setting the structure

fig2.set_size_inches(20, 5, forward=True) # adjusting the size of the f

ax2.set_title('Spread of Number of Votes') # setting the title of the p

ax2.set_xlabel('Number of Votes') # setting the x-axis label


ax2.boxplot(tmdb['tmdb_numvotes'], vert = False, showfliers=False);
```



Similiar to our previous dataset, this data is also **right skewed** meaning that there are more movies getting low number of votes. We will compare the number of votes to the average rating after the merge of the two datasets.

In [53]:
```
# join tmdb with mov_bas_rat based on original_title
# use pandasql to merge the two

q = """
SELECT *
FROM mov_bas_rat
INNER JOIN tmdb
    USING(original_title)
"""

mov_brt = pysqldf(q)

# preview results
mov_brt.head()
```

Out[53]:

	movie_id	primary_title	original_title	start_year	genres	genre1	genre2	
0	tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018	Drama	Drama	NA	
1	tt0112502	Bigfoot	Bigfoot	2017	Horror,Thriller	Horror	Thriller	
2	tt0249516	Foodfight!	Foodfight!	2012	Action,Animation,Comedy	Action	Animation	C
3	tt0255820	Return to Babylon	Return to Babylon	2013	Biography,Comedy,Drama	Biography	Comedy	
4	tt0285252	Life's a Beach	Life's a Beach	2012	Comedy	Comedy	NA	

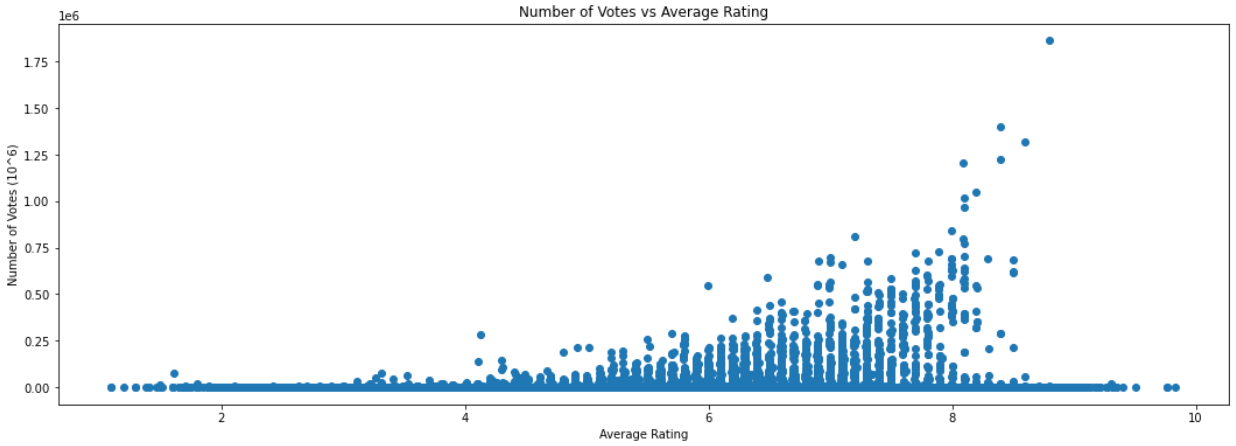
Since the two datasets with reviews have been merged, lets average out the reviews between the two merged datasets and then investigate what baseline we should have for minimum number of votes.

In [54]:
```
#calculate average rating between the two datasets

#total votes
totvotes = mov_brt['numvotes'] + mov_brt['tmdb_numvotes']
mov_brt['tot_votes'] = totvotes

#imdb average multiplied by total votes
imdb_average = mov_brt['averagerating'].multiply(mov_brt['numvotes'])

#tmdb average multiplied by total votes
tmdb_average = mov_brt['tmdb_averagerating'].multiply(mov_brt['tmdb_num

#averaging out the ratings
mov_brt['avg_rating'] = (imdb_average + tmdb_average).div(totvotes)

#dropping other review and votes rating
mov_brt = mov_brt.drop(['averagerating','numvotes', 'tmdb_averagerating

#preview the data
mov_brt.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16224 entries, 0 to 16223
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   movie_id           16224 non-null  object 
 1   primary_title      16224 non-null  object 
 2   original_title     16224 non-null  object 
 3   start_year         16224 non-null  int64  
 4   genres             16224 non-null  object 
 5   genre1             16224 non-null  object 
 6   genre2             16224 non-null  object 
 7   genre3             16224 non-null  object 
 8   original_language  16224 non-null  object 
 9   tot_votes          16224 non-null  int64  
 10  avg_rating         16224 non-null  float64
dtypes: float64(1), int64(2), object(8)
memory usage: 1.4+ MB
```

Lets take a look at the spread of the review ratings vs total votes.

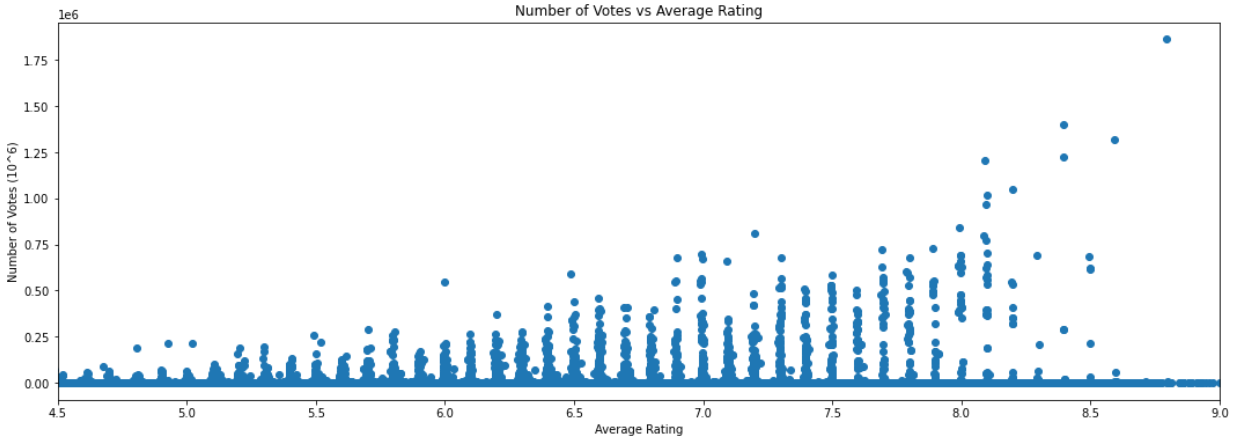
In [55]:
```
fig, ax = plt.subplots(figsize=(18,6))

ax.scatter(y = mov_brt['tot_votes'], x = mov_brt['avg_rating']);
plt.title('Number of Votes vs Average Rating')
plt.ylabel('Number of Votes (10^6)')
plt.xlabel('Average Rating');
```



It is clear from the plot above that low number of reviews are linked to either extremely bad ratings, i.e. less than 5, or extremely high reviews, i.e. greater than 8.5, but as the number for reviews increase, we generally see a trend of an average rating between 5 and 8.5.

Lets zoom in between rataings `5` and `8.5` and investigate the spread over there.

In [56]:
```
#set the plot parameters
fig2, ax2 = plt.subplots(figsize=(18,6))

#plot a scatter plot
ax2.scatter(y = mov_brt['tot_votes'], x = mov_brt['avg_rating']);

#limitthe scales with 0.5 tolerance to make it clear whether 5 and 8.5
plt.xlim(4.5,9);

#set labels
plt.title('Number of Votes vs Average Rating')
plt.ylabel('Number of Votes (10^6)')
plt.xlabel('Average Rating');
```



It is a lot clearer that a good spread of reviews is between `5` and `8.5`. Another thing to notice is that as the number of votes increase, their is a general trend towards an increased rating.

This signifies that if the client pursues a genre with high ratings, they will get good customer feedback in large quantities therefore increasing fan following.

Since we want to look at movies that get great customer feedback, it would be beneficial to filter out based on ratings since we know the baand of ratings where the number of votes have balanced it out.

In [57]:
```
mov_brt = mov_brt[mov_brt['avg_rating'] >= 5] #take records with averag
```

In [58]:
```
mov_brt = mov_brt[mov_brt['avg_rating'] <= 8.5] #take records with aver
```

This takes care of ensuring that we only have those records that have ample votes to balance out the ratings.

While the greater the rating, the better the movie, there needs to be a minimum break-off point for a movie to be categorized as above-average. To investigate which genres generally land in the above average category, we will investigate what the average of the `avg_rating` column is.

In [59]:
```
#calculate the average of the avg_column
average = mov_brt['avg_rating'].mean()
print(average, '\n')

#filter records based on this average. Drop the movies with below avera
mov_brt = mov_brt[mov_brt['avg_rating'] > average]

#preview the results
mov_brt
```

```
6.429480446144693
```

Out[59]:

	movie_id	primary_title	original_title	start_year	genres	genre1	gen
0	tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018	Drama	Drama	
5	tt0293069	Dark Blood	Dark Blood	2012	Thriller	Thriller	
6	tt0297400	Snowblind	Snowblind	2015	Crime,Drama	Crime	Dra
12	tt0359950	The Secret Life of Walter Mitty	The Secret Life of Walter Mitty	2013	Adventure,Comedy,Drama	Adventure	Com
14	tt0365545	Nappily Ever After	Nappily Ever After	2018	Comedy,Drama,Romance	Comedy	Dra
...	...	...	...	...	...	...	
16211	tt9598566	Ave Maria	Ave Maria	2018	Drama	Drama	
16214	tt9647790	Celia	Celia	2018	Documentary	Documentary	
16217	tt9777830	John Leguizamo's Latin History for Morons	John Leguizamo's Latin History for Morons	2018	Comedy	Comedy	
16219	tt9899880	Columbus	Columbus	2018	Comedy	Comedy	
16223	tt9914642	Albatross	Albatross	2017	Documentary	Documentary	

5584 rows × 11 columns

The Numbers

In [60]:
```
#preview the info and data
print(tn_df.info(),'\n')

tn_df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5782 entries, 0 to 5781
Data columns (total 6 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
dtypes: int64(1), object(5)
memory usage: 271.2+ KB
None
```

Out[60]:

	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

Looks like there are no missing values.

Nonetheless, there is still some cleaning to do.

We will only keep the years from the `release_date` column

Rename the `movie` column to `title` for consistency between dataframes

We should also adjust the financial columns so that the `$` sign and `,` is removed.

We will also change the data type of the financial columns to a numeric type.

In [61]:
```
#copy the dataframe
tn = tn_df.copy()
```

In [62]:
```
#keep only the years in release_date column
#use the datetime module to extract the years

tn['release_date'] = tn['release_date'].map(
    lambda x: dt.datetime.strptime(x.replace(',',''),'%b %d %Y').year).
```

In [63]:
```
#rename the movie column to title
tn.rename(columns = {'movie':'title'},inplace = True)
```

In [64]:
```
#drop the id column
tn = tn.drop('id', axis=1)
```

In [65]:
```
#changing the datatype to integer for financial information

#change everything to string
#repalce ',' and then strip the '$' sign.
#change to integer type
tn['production_budget'] = tn['production_budget'].astype(str).str.repla

tn['domestic_gross'] = tn['domestic_gross'].astype(str).str.replace(','

tn['worldwide_gross'] = tn['worldwide_gross'].astype(str).str.replace('

#preview results
print(tn.info(), '\n')
tn.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5782 entries, 0 to 5781
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   release_date       5782 non-null   int64 
 1   title              5782 non-null   object
 2   production_budget  5782 non-null   int64 
 3   domestic_gross     5782 non-null   int64 
 4   worldwide_gross    5782 non-null   int64 
dtypes: int64(4), object(1)
memory usage: 226.0+ KB
None
```

Out[65]:

	release_date	title	production_budget	domestic_gross	worldwide_gross
0	2009	Avatar	425000000	760507625	2776345279
		Pirates of the Caribbean: On			

Since our client is considering making an impactful entry in the market, we will be evaluating movies that had atleast **$1 million** in domestic gross revenue. This also focuses on increased profitability.

We will also improve the readability of the financial values by rounding them to the nearest million and adding to columnns `$(millions)`

In [66]:
```
#dropping records with less than 1 million domestic_gross
tn = tn[tn['domestic_gross'] >= 1000000]

#reducing significant figures
tn['production_budget'] = tn['production_budget'].div(1000000).astype('
tn['domestic_gross'] = tn['domestic_gross'].div(1000000).astype('int64'
tn['worldwide_gross'] = tn['worldwide_gross'].div(1000000).astype('int6

#rounding everything
tn['production_budget'] = tn['production_budget'].round(-1)
tn['domestic_gross'] = tn['domestic_gross'].round(-1)
tn['worldwide_gross'] = tn['worldwide_gross'].round(-1)

#rename the columns
tn.rename(columns = {'production_budget':'production_budget ($ millions
                     'domestic_gross' : 'domestic_gross ($ millions)',
                      'worldwide_gross' : 'worldwide_gross ($ million
                      }, inplace=True)
```

In [67]:
```
#preview the results
print(tn.info(), '\n')

tn.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4451 entries, 0 to 5773
Data columns (total 5 columns):
 #   Column                          Non-Null Count  Dtype 
---  ------                          --------------  ----- 
 0   release_date                    4451 non-null   int64 
 1   title                           4451 non-null   object
 2   production_budget ($ millions)  4451 non-null   int64 
 3   domestic_gross ($ millions)     4451 non-null   int64 
 4   worldwide_gross ($ millions)    4451 non-null   int64 
dtypes: int64(4), object(1)
memory usage: 208.6+ KB
None
```

Out[67]:

	release_date	title	production_budget ($ millions)	domestic_gross ($ millions)	worldwide_gross ($ millions)
0	2009	Avatar	420	760	2780
1	2011	Pirates of the Caribbean: On Stranger Tides	410	240	1040
2	2019	Dark Phoenix	350	40	150
3	2015	Avengers: Age of Ultron	330	460	1400
4	2017	Star Wars Ep. VIII: The Last Jedi	320	620	1320

Now, we are going to join this dataset with `mov_brt` to complete our Dataframe for analysis. This will create a master dataframe that can be used for financial analysis in conjunction with genres and movie ratings.

In [68]:
```
#Join the two datasets using SQL

q1 = """
SELECT *
FROM mov_brt AS m
INNER JOIN tn AS t
    ON m.primary_title = t.title
"""

brt_tn = pysqldf(q1)

#preview the dataframe
brt_tn
```

Out[68]:

	movie_id	primary_title	original_title	start_year	genres	genre1	ger
0	tt0359950	The Secret Life of Walter Mitty	The Secret Life of Walter Mitty	2013	Adventure,Comedy,Drama	Adventure	Com
1	tt0365907	A Walk Among the Tombstones	A Walk Among the Tombstones	2014	Action,Crime,Drama	Action	Cı
2	tt0369610	Jurassic World	Jurassic World	2015	Action,Adventure,Sci-Fi	Action	Adven
3	tt0398286	Tangled	Tangled	2010	Adventure,Animation,Comedy	Adventure	Anima
4	tt0401729	John Carter	John Carter	2012	Action,Adventure,Sci-Fi	Action	Adven
...	...	...	...	...	...	...	
1056	tt8917752	Joy	Joy	2018	Drama	Drama	
1057	tt8919194	Genius	Genius	2018	Drama	Drama	
1058	tt9077038	Monster	Monster	2019	Comedy,Drama,Family	Comedy	Dra
1059	tt9077038	Monster	Monster	2019	Comedy,Drama,Family	Comedy	Dra
1060	tt9899880	Columbus	Columbus	2018	Comedy	Comedy	

1061 rows × 16 columns

We can see the gross revenue in this dataset but we will be using profits to conduct our analysis. So lets go ahead and calculate the domestic and foreign profits and populate them in a separate column.

- **Foreign Gross:** Calculate the difference between worldwide gross and domestic gross.
- **Net Profit:** Calculate the difference between worldwide gross revenue and the production budget.
- **Foreign Profit:** Distribute the net profit based on the ratio of foreign gross in worldwide gross.
- **Domestic Profit:** Distribute the net profit based on the ratio of domestic gross in worldwide gross.

In [69]:
```
#Calculate foregin gross
fgross = brt_tn['worldwide_gross ($ millions)'] - brt_tn['domestic_gros

#Calculate Net Profit
nprofit = brt_tn['worldwide_gross ($ millions)'] - brt_tn['production_b

#worldwide gross
wgross = brt_tn['worldwide_gross ($ millions)']

#Calculate foreign profit and round to the nearest Unit
fprofit = round((fgross/wgross) * nprofit, 0)

#Calculate domestic profit
dprofit = round((brt_tn['domestic_gross ($ millions)']/wgross) * nprofi

#Storing in the dataframe
brt_tn['foreign_gross ($ millions)'] = fgross

brt_tn['netprofit ($ millions)'] = nprofit

brt_tn['domesticprofit ($ millions)'] = dprofit

brt_tn['foreignprofit ($ millions)'] = fprofit

brt_tn.head()
```

Out[69]:

	movie_id	primary_title	original_title	start_year	genres	genre1	genre2
0	tt0359950	The Secret Life of Walter Mitty	The Secret Life of Walter Mitty	2013	Adventure,Comedy,Drama	Adventure	Comedy
1	tt0365907	A Walk Among the Tombstones	A Walk Among the Tombstones	2014	Action,Crime,Drama	Action	Crime
2	tt0369610	Jurassic World	Jurassic World	2015	Action,Adventure,Sci-Fi	Action	Adventure
3	tt0398286	Tangled	Tangled	2010	Adventure,Animation,Comedy	Adventure	Animation
4	tt0401729	John Carter	John Carter	2012	Action,Adventure,Sci-Fi	Action	Adventure

Review

The client had 3 goals that we wanted to help achieve:

1. Positive Viewer Response
2. Domestic vs Foreign Launch
3. Profitability

To help achieve these goals, we decided to pose three three questions for our analysis to answer:

1. Which genres bring in the highest ratings and profits?
    - This allows us to produce insights for the profitability and positive viewer response

2. Should the movie be launched internationally?
    - This allows to gauge whether an international launch is viable?

3. What should the budget target be for the movie?
    - This allows us to understand the profit margins the client can expect

We talked about two main KPIs to use for our analysis. Our dataframes have been organized to have both of these KPIs included in them. These two KPIs are listed below.

1. Ratings
2. Profits

To answer these questions, we will perform the following analysis:

1. Which genres bring in the highest ratings and profits?
    - genre vs ratings analysis
    - genre vs profits analysis
    - genre vs ratings vs profits analysis

2. Should the movie be launched internationally?
    - genres vs domestic profit analysis
    - genres vs foreign profit analysis

3. What should the budget target be for the movie?
    - genres vs production budget vs domestic & foreign profit analysis

Data Interpretation and Visualizations

Which genres bring in the highest ratings and profits?

Genre vs Reviews Analysis:

Looking at genre vs ratings will give the client insights into which genres are the most popular amongst viewers

Lets preview the dataframe we will be using and make a copy of it for analysis.

In [70]:
```
#make a copy
brt_analysis = brt_tn.copy()

#preview the data
brt_analysis.head()
```

Out[70]:

	movie_id	primary_title	original_title	start_year	genres	genre1	genre2
0	tt0359950	The Secret Life of Walter Mitty	The Secret Life of Walter Mitty	2013	Adventure,Comedy,Drama	Adventure	Comedy
1	tt0365907	A Walk Among the Tombstones	A Walk Among the Tombstones	2014	Action,Crime,Drama	Action	Crime
2	tt0369610	Jurassic World	Jurassic World	2015	Action,Adventure,Sci-Fi	Action	Adventure
3	tt0398286	Tangled	Tangled	2010	Adventure,Animation,Comedy	Adventure	Animation
4	tt0401729	John Carter	John Carter	2012	Action,Adventure,Sci-Fi	Action	Adventure

Lets extract the columns we need to do the genre vs rating analysis.

We will need `genres` , `genre1` , `genre2` , `genre3` , `tot_votes` , `avg_rating`

In [71]:
```
#drop the columns not needed
brt_analysis = brt_analysis.drop(['primary_title', 'original_title', 'o
```

In [72]:
```
# preview the result
brt_analysis
```

Out[72]:

	movie_id	start_year	genres	genre1	genre2	genre3	tot_votes	av
0	tt0359950	2013	Adventure,Comedy,Drama	Adventure	Comedy	Drama	280159	
1	tt0365907	2014	Action,Crime,Drama	Action	Crime	Drama	106801	
2	tt0369610	2015	Action,Adventure,Sci-Fi	Action	Adventure	Sci-Fi	553394	
3	tt0398286	2010	Adventure,Animation,Comedy	Adventure	Animation	Comedy	372773	
4	tt0401729	2012	Action,Adventure,Sci-Fi	Action	Adventure	Sci-Fi	245130	
...	...	...	...	...	...	...	...	
1056	tt8917752	2018	Drama	Drama	NA	NA	3153	
1057	tt8919194	2018	Drama	Drama	NA	NA	444	
1058	tt9077038	2019	Comedy,Drama,Family	Comedy	Drama	Family	50	
1059	tt9077038	2019	Comedy,Drama,Family	Comedy	Drama	Family	50	
1060	tt9899880	2018	Comedy	Comedy	NA	NA	135	

1061 rows × 17 columns

Lets use SQL to group our records by genres and calculate the count of movie sin each combination.

In [73]:
```
#write a sql quer to group by genres and count the number of movies
q = """
SELECT *, COUNT(movie_id) as count_of_movies
FROM brt_analysis
GROUP BY genre1, genre2, genre3
ORDER BY count_of_movies DESC
"""

brt_ratc = pysqldf(q)

#preview the results
brt_ratc.head(10)
```

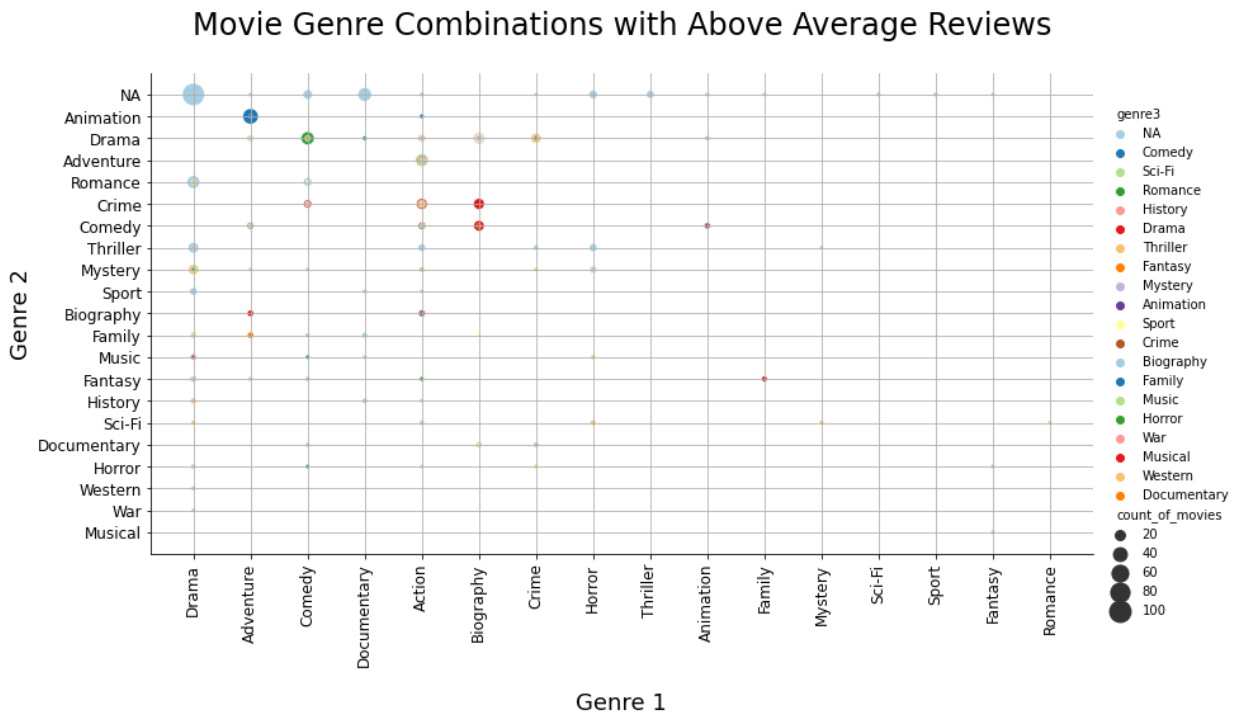
Out[73]:

	movie_id	start_year	genres	genre1	genre2	genre3	tot_votes	a
0	tt0935075	2010	Drama	Drama	NA	NA	45649	
1	tt0398286	2010	Adventure,Animation,Comedy	Adventure	Animation	Comedy	372773	
2	tt0878835	2010	Comedy,Drama	Comedy	Drama	NA	10733	
3	tt1529567	2010	Documentary	Documentary	NA	NA	376	
4	tt0369610	2015	Action,Adventure,Sci-Fi	Action	Adventure	Sci-Fi	553394	
5	tt0804497	2010	Comedy,Drama,Romance	Comedy	Drama	Romance	126907	
6	tt1067583	2011	Drama,Romance	Drama	Romance	NA	107439	
7	tt0443272	2012	Biography,Drama,History	Biography	Drama	History	230962	
8	tt0993846	2013	Biography,Crime,Drama	Biography	Crime	Drama	1047769	
9	tt0365907	2014	Action,Crime,Drama	Action	Crime	Drama	106801	

Lets plot this using `seaborn` . The goal is to look at the ooverlapa of the genres and see which combination is the most common in movies that are above the average rating.

In [74]:

```
#setting a custom color palette to have distinguishable results
cmap = sns.color_palette("Paired",n_colors=20)

#plotting the graph
g = sns.relplot(
    data=brt_ratc,
    x="genre1", y="genre2",
    hue="genre3", size="count_of_movies",
    palette = cmap,
     sizes=(10, 300),
    height = 6,
    aspect = 2
)

#set the title and labels
plt.title('Movie Genre Combinations with Above Average Reviews',fontdic

plt.xlabel('Genre 1', labelpad = 20, fontdict={'fontsize':18});
plt.ylabel('Genre 2', fontdict={'fontsize':18});

#set rotation and size of x ticks
plt.xticks(rotation = 90, fontsize = 12);
plt.yticks(fontsize = 12);

#set the grid to make it easy for reading
plt.grid(True)

```



We can see that there are a lot of genres that have very little number of movies to be significant for our analysis.

Lets break down the groupings and review which genres are the most popular amongst viewers. We will be using the Top 10.

In [75]:

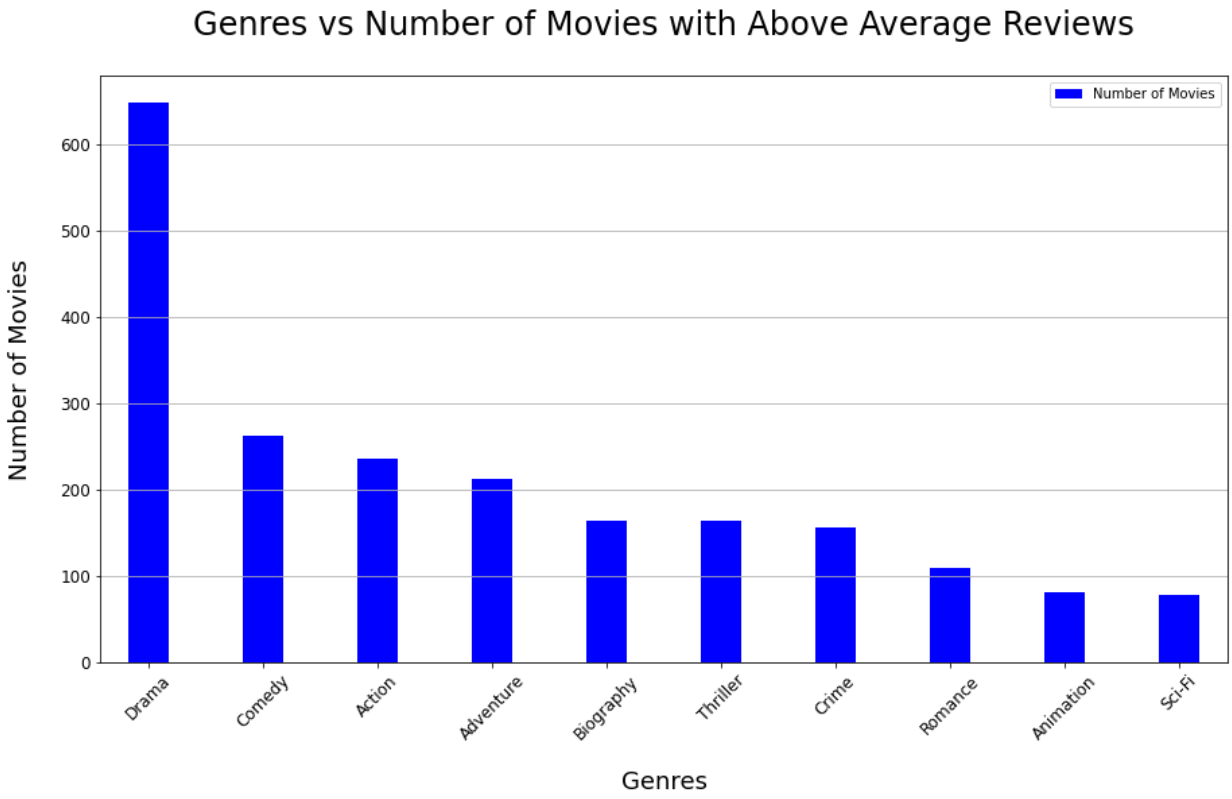
```
#Counter genre1
g1 = Counter(brt_analysis['genre1'])

#Counter genre2
g2 = Counter(brt_analysis['genre2'])

#Counter genre3
g3 = Counter(brt_analysis['genre3'])

gcounter = {}

#count all genres
for k in (g1,g2,g3):
    for key, value in k.items():
        if key == 'NA':
            pass
        elif key in gcounter.keys():
            gcounter[key] = gcounter[key] + value
        elif key not in gcounter.keys():
            gcounter[key] = value

#converting to a dataframe
genre_popular = pd.DataFrame.from_dict(gcounter, orient = 'index')

#resetting the index
genre_popular = genre_popular.reset_index()

#giving columns names
genre_popular.columns = ['genre', 'num_of_movies']

#sort the data
genre_popular = genre_popular.sort_values(by = ['num_of_movies'], axis

#select the top 10
genre_popular = genre_popular.iloc[0:10]

#preview the data
genre_popular
```

Out[75]:

	genre	num_of_movies
2	Drama	647
7	Comedy	262
1	Action	236
0	Adventure	212
3	Biography	164
12	Thriller	164
5	Crime	156
11	Romance	108
8	Animation	81
15	Sci-Fi	78

We will use the top 10 to visualize this information to see the most popular genres

In [76]:
```
#plot a bar chart of genres vs number of movies
genre_popular.plot(kind='bar', width = 0.35, figsize = (15,8), color =

ax = plt.gca()


# set xticks
ax.set_xticklabels(genre_popular['genre'])

#set axis labels
plt.xlabel('Genres', labelpad = 20, fontdict={'fontsize':18})
plt.ylabel('Number of Movies', labelpad= 20, fontdict={'fontsize':18})

#set the title
plt.title('Genres vs Number of Movies with Above Average Reviews',fontd

#set size and rotation of ticks
plt.xticks(rotation = 45, fontsize = 12);
plt.yticks(fontsize = 12);

#set the grid to make it easy for reading
plt.grid(True, axis = 'y')

#set the legend
plt.legend(['Number of Movies'])

plt.show()
```



We can easilly decypher that the top 10 most popular genres that are rated above average are:

1. Drama
2. Comedy
3. Action
4. Adventure
5. Biography
6. Thriller
7. Crime
8. Romance
9. Animation
10. Sci-Fi

Nonetheless, purely high reviews can't be only reference of producing a successful movie. It needs to be looked at from the financial lens also. For this purpose, we will look at genre vs profits analysis next.

Genre vs Profits analysis

Lets take a look at how these popular genres do against the test of money. We will evaluate which genre combinations will allow the client to be most profitable.

Lets preview the dataframe that we'll be using for this analysis and make a copy of it to work with.

In [77]:
```
#make a copy
brttn = brt_tn.copy()

#preview the dataframe
brttn.head()
```

Out[77]:

	movie_id	primary_title	original_title	start_year	genres	genre1	genre2
0	tt0359950	The Secret Life of Walter Mitty	The Secret Life of Walter Mitty	2013	Adventure,Comedy,Drama	Adventure	Comedy
1	tt0365907	A Walk Among the Tombstones	A Walk Among the Tombstones	2014	Action,Crime,Drama	Action	Crime
2	tt0369610	Jurassic World	Jurassic World	2015	Action,Adventure,Sci-Fi	Action	Adventure
3	tt0398286	Tangled	Tangled	2010	Adventure,Animation,Comedy	Adventure	Animation
4	tt0401729	John Carter	John Carter	2012	Action,Adventure,Sci-Fi	Action	Adventure

Lets extract the columns that we'll be using for analysis. These will be the genre, ratings, votes and finanical columns.

In [78]:
```
#extract the necessary columns
brttn = brttn[['movie_id',
               'genres',
               'genre1',
               'genre2',
               'genre3',
               'avg_rating',
               'tot_votes',
               'production_budget ($ millions)',
               'netprofit ($ millions)',
               'domesticprofit ($ millions)',
               'foreignprofit ($ millions)']]

#preview the dataframe
brttn.head()
```

Out[78]:

	movie_id	genres	genre1	genre2	genre3	avg_rating	tot_votes	prod
0	tt0359950	Adventure,Comedy,Drama	Adventure	Comedy	Drama	7.296531	280159	
1	tt0365907	Action,Crime,Drama	Action	Crime	Drama	6.496845	106801	
2	tt0369610	Action,Adventure,Sci-Fi	Action	Adventure	Sci-Fi	6.989840	553394	
3	tt0398286	Adventure,Animation,Comedy	Adventure	Animation	Comedy	7.794844	372773	
4	tt0401729	Action,Adventure,Sci-Fi	Action	Adventure	Sci-Fi	6.593191	245130	

Since we already have the movies filtered for average ratings, we can move on to grouping our data genres and calculating the average net, domestic and gross profits per genre combination.

In [79]:
```
a = """
SELECT *,
    AVG("netprofit ($ millions)") AS avg_netprofits,
    AVG("domesticprofit ($ millions)") AS avg_domprofits,
    AVG("foreignprofit ($ millions)") AS avg_foreignprofits
FROM brttn
GROUP BY genre1, genre2, genre3
ORDER BY avg_netprofits DESC
"""

brttn_g = pysqldf(a)

brttn_g
```

Out[79]:

	movie_id	genres	genre1	genre2	genre3	avg_rating	tot_votes	productic ($
0	tt2771200	Family,Fantasy,Musical	Family	Fantasy	Musical	7.211739	244055	
1	tt2049386	Fantasy,Musical	Fantasy	Musical	NA	6.599931	8719	
2	tt1170358	Adventure,Fantasy	Adventure	Fantasy	NA	7.797300	573302	
3	tt0903624	Adventure,Family,Fantasy	Adventure	Family	Fantasy	7.888447	731705	
4	tt2071483	Family	Family	NA	NA	7.999174	12706	
...	...	...	...	...	...	...	...	
171	tt2798920	Adventure,Drama,Horror	Adventure	Drama	Horror	6.888509	233129	
172	tt0968264	Crime,Drama,History	Crime	Drama	History	6.896607	26642	
173	tt7598276	Crime,Horror,Thriller	Crime	Horror	Thriller	6.432979	188	
174	tt3779570	Sport	Sport	NA	NA	7.836250	80	
175	tt0490215	Adventure,Drama,History	Adventure	Drama	History	7.198203	86802	

176 rows × 14 columns

While there are a lot of genre combinations with high profits, we want to be looking at the most lucrataive ones for the most success. Therefore we will extract the top 20 genre combinations by highest average net profits.

In [80]:
```
gen_nprof = brttn_g.head(20)
gen_nprof
```

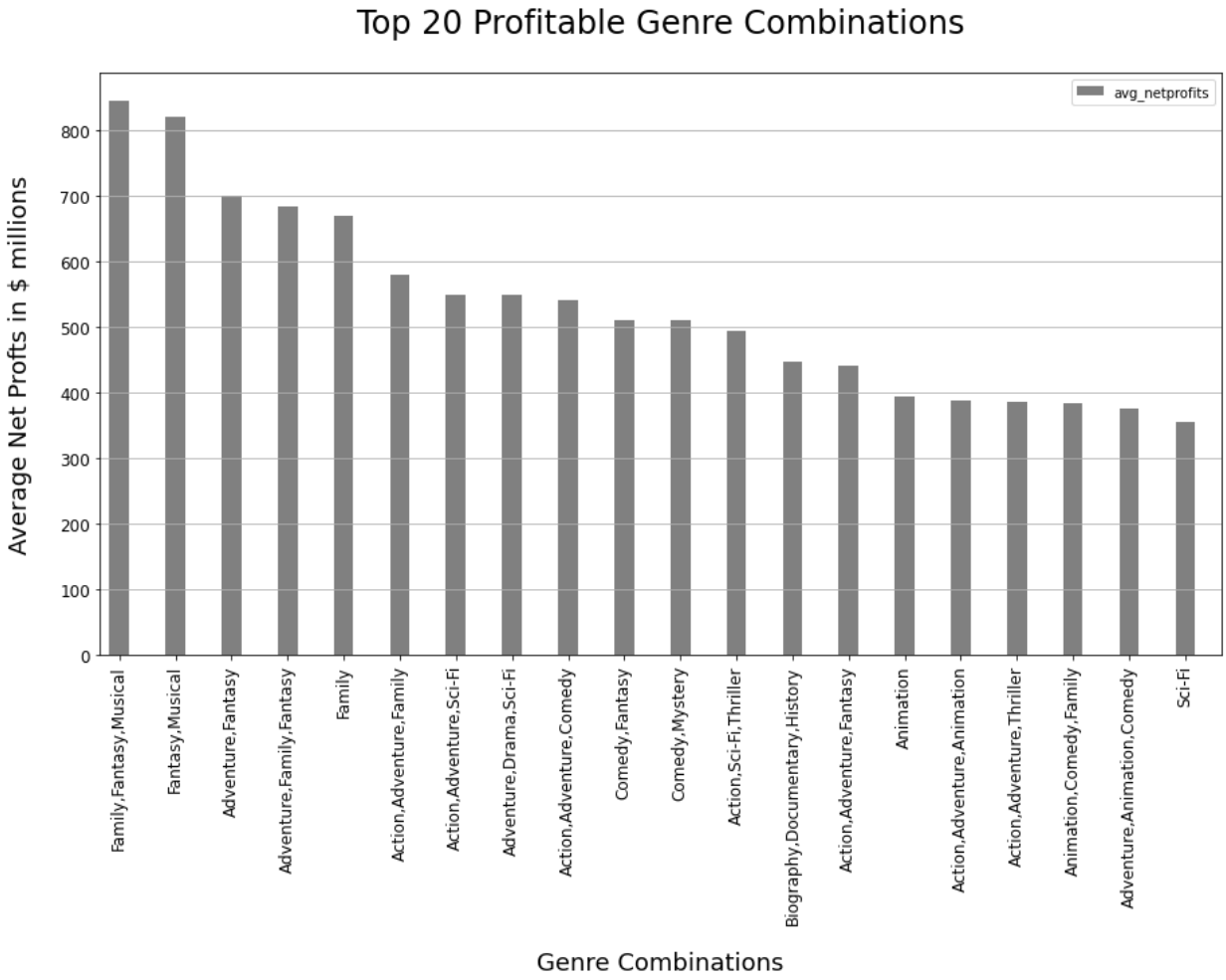
Out[80]:

	movie_id	genres	genre1	genre2	genre3	avg_rating	tot_votes
0	tt2771200	Family,Fantasy,Musical	Family	Fantasy	Musical	7.211739	244055
1	tt2049386	Fantasy,Musical	Fantasy	Musical	NA	6.599931	8719
2	tt1170358	Adventure,Fantasy	Adventure	Fantasy	NA	7.797300	573302
3	tt0903624	Adventure,Family,Fantasy	Adventure	Family	Fantasy	7.888447	731705
4	tt2071483	Family	Family	NA	NA	7.999174	12706
5	tt1587310	Action,Adventure,Family	Action	Adventure	Family	7.000000	313975
6	tt0369610	Action,Adventure,Sci-Fi	Action	Adventure	Sci-Fi	6.989840	553394
7	tt3659388	Adventure,Drama,Sci-Fi	Adventure	Drama	Sci-Fi	7.994725	692288
8	tt0478970	Action,Adventure,Comedy	Action	Adventure	Comedy	7.295343	513162
9	tt1637725	Comedy,Fantasy	Comedy	Fantasy	NA	6.891510	546094
10	tt1411697	Comedy,Mystery	Comedy	Mystery	NA	6.498581	439031
11	tt1270797	Action,Sci-Fi,Thriller	Action	Sci-Fi	Thriller	6.699948	275412
12	tt2608638	Biography,Documentary,History	Biography	Documentary	History	6.784615	78
13	tt0451279	Action,Adventure,Fantasy	Action	Adventure	Fantasy	7.499888	487561
14	tt2226178	Animation	Animation	NA	NA	6.802220	5316
15	tt0448694	Action,Adventure,Animation	Action	Adventure	Animation	6.596835	134777
16	tt0429493	Action,Adventure,Thriller	Action	Adventure	Thriller	6.794320	237959
17	tt0837562	Animation,Comedy,Family	Animation	Comedy	Family	7.095216	209584
18	tt0398286	Adventure,Animation,Comedy	Adventure	Animation	Comedy	7.794844	372773
19	tt3971092	Sci-Fi	Sci-Fi	NA	NA	7.070767	821

Lets visualize these results to analyze the differences amongst them

In [81]:

```
#plot a bar chart of genres vs net profts
gen_nprof[['genres','avg_netprofits']].plot(kind='bar', width = 0.35, f

ax = plt.gca()

#set xlimits
plt.xlim([-0.35, len(gen_nprof['avg_netprofits'])-0.35])

# set xticks
ax.set_xticklabels(gen_nprof['genres'])

#set axis labels
plt.xlabel('Genre Combinations', labelpad = 20, fontdict={'fontsize':18
plt.ylabel('Average Net Profts in $ millions', labelpad= 20, fontdict={

#set the title
plt.title('Top 20 Profitable Genre Combinations',fontdict={'fontsize':2



#set size of x ticks
plt.xticks(fontsize = 12);
plt.yticks(fontsize = 12);

#set the grid to make it easy for reading
plt.grid(True, axis = 'y')



plt.show()
```



We can very clearly see repetitions of certain genres in the top 20.

Lets use the counter to calculate the most common genres

In [82]:
```
#Counter genre1
g1c = Counter(gen_nprof['genre1'])

#Counter genre2
g2c = Counter(gen_nprof['genre2'])

#Counter genre3
g3c = Counter(gen_nprof['genre3'])

dd = {}

for k in (g1c,g2c,g3c):
    for key, value in k.items():
        if key in dd.keys():
            dd[key] = dd[key] + value
        elif key not in dd.keys():
            dd[key] = value


print(dd)
```

```
{'Family': 5, 'Fantasy': 6, 'Adventure': 10, 'Action': 7, 'Comedy': 5, 'B
iography': 1, 'Animation': 4, 'Sci-Fi': 4, 'Musical': 2, 'NA': 10, 'Dram
a': 1, 'Mystery': 1, 'Documentary': 1, 'Thriller': 2, 'History': 1}
```

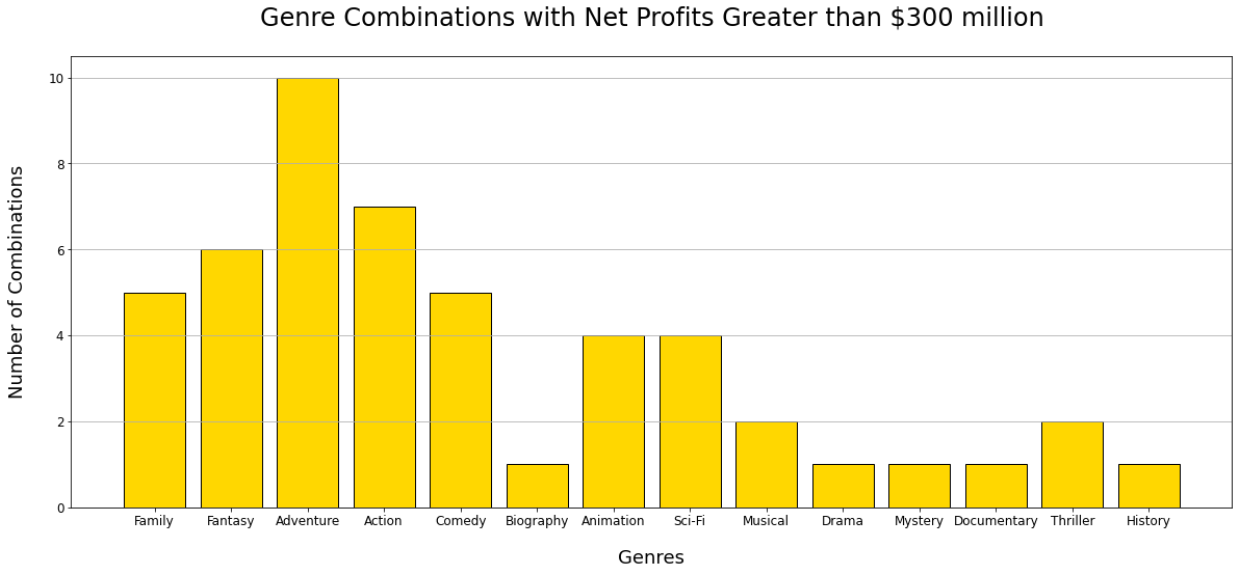
Lets plot this out for a better understanding but lets delete `NA`

In [83]:
```
#use .pop to drop the 'NA' genre
dd.pop('NA', None)
```

Out[83]: 10

In [84]:

```
#separate the keys and values
genres_dd = list(dd.keys())
combination_counts = list(dd.values())

#set plotting parameters
fig, ax = plt.subplots(figsize=(20,8))

#plot the graph
plt.bar(range(len(dd)), combination_counts, tick_label = genres_dd, col

#set axis labels
plt.xlabel('Genres', labelpad = 20, fontdict={'fontsize':18})
plt.ylabel('Number of Combinations', labelpad= 20, fontdict={'fontsize'

#set the title
plt.title('Genre Combinations with Net Profits Greater than $300 millio

#set size of x ticks
plt.xticks(fontsize = 12);
plt.yticks(fontsize = 12);

#set the grid to make it easy for reading
plt.grid(True, axis = 'y')

plt.show()
```



In [85]:
```
print(f"""

The most common genre in the top 20 are
1. Adventure with {dd['Adventure']} groupings
2. Action with {dd['Action']} groupings
3. Fantasy with {dd['Fantasy']} groupings
4. Family {dd['Family']} and Comedy with {dd['Comedy']} groupings
5. Animation {dd['Animation']} and Sci-fi with {dd['Sci-Fi']} groupings
6. Musical {dd['Musical']} and Thriller with {dd['Thriller']} groupings
7. Drama, Mystery, Documentary, History, Biography with {dd['History']}

"""
)
```

```
The most common genre in the top 20 are
1. Adventure with 10 groupings
2. Action with 7 groupings
3. Fantasy with 6 groupings
4. Family 5 and Comedy with 5 groupings
5. Animation 4 and Sci-fi with 4 groupings
6. Musical 2 and Thriller with 2 groupings
7. Drama, Mystery, Documentary, History, Biography with 1 groupings each
```

That being said, it is very clear that the most common type of successsful movies are a combination of atleast one of these genres but it is to be noted that the combination of these three together does not produce the most profitable mmovie. The addition of `Musical` allows movie to cross the **$800 million** barrier but `Musical` wasn't one of the highest rated categories which means that it is rare for `Musial` commbinations to be rated very high.

Lets recap the most popular genres in the above average categories and compare those with the findings here.

1. `Drama`
    - Drama only had a single combination in the top 20 profitable categor therefore we can ignore this one. We want to investigate movies that have a high chance of getting good reviews and making a good profit.
2. `Comedy`
    - Comedy fared well with 5 combinations in the top 20 profitable category.
3. `Action`
    - Action was the second most repeated genre combination with 7 groupings.
4. `Adventure`
    - Adventure was the most repeated genre with 10 combinations.

5. `Biography`
    - Biography only had a single combination in the top 20 profitable categor therefore we can ignore this one.
6. `Thriller`
    - Thriller only had 2 combinations therefore we will ignore this one too.
7. `Crime`
    - Crime didn't have any occurance so we can confidently drop it from consideration.
8. `Romance`
    - Romance didn't have any occurance so we can confidently drop it from consideration too.
9. `Animation`
    - Animation had 4 combinaitons in the top 20 profitable category.
10. `Sci-Fi`
    - Sci-Fi had 4 combinaitons in the top 20 profitable category

This tells us that we should keep an eye out for the following categories going forward:

1. `Adventure`
2. `Action`
3. `Comedy`
4. `Animation`
5. `Sci-fi`

Genre vs Reviews vs Profits analysis

Lets move on to evaluating the difference between the ratings between the top 10 profitable genre combinations.

This will give us a better understanding of what will allow the client to garner the best customer feedback with a balance of good profits.

We will focus on the 5 genres that we have shortlisted and their combinations. We will filter out the `gen_nprof` dataframe to extract records with these genres. The shortlisted genres are:

1. `Adventure`
2. `Action`
3. `Comedy`
4. `Animation`
5. `Sci-fi`

In [86]:
```
#Use SQL LIKE statement to match genres column with shortlisted genres
z = """
SELECT *
FROM gen_nprof
WHERE
    (genres LIKE '%Adventure%')
    OR
    (genres LIKE '%Action%')
    OR
    (genres LIKE '%Comedy%')
    OR
    (genres LIKE '%Animation%')
    OR
    (genres LIKE '%Sci-Fi%')
ORDER BY avg_netprofits DESC;

"""

gen_short = pysqldf(z)
gen_short
```

Out[86]:

	movie_id	genres	genre1	genre2	genre3	avg_rating	tot_votes	pr
0	tt1170358	Adventure,Fantasy	Adventure	Fantasy	NA	7.797300	573302	
1	tt0903624	Adventure,Family,Fantasy	Adventure	Family	Fantasy	7.888447	731705	
2	tt1587310	Action,Adventure,Family	Action	Adventure	Family	7.000000	313975	
3	tt0369610	Action,Adventure,Sci-Fi	Action	Adventure	Sci-Fi	6.989840	553394	
4	tt3659388	Adventure,Drama,Sci-Fi	Adventure	Drama	Sci-Fi	7.994725	692288	
5	tt0478970	Action,Adventure,Comedy	Action	Adventure	Comedy	7.295343	513162	
6	tt1637725	Comedy,Fantasy	Comedy	Fantasy	NA	6.891510	546094	
7	tt1411697	Comedy,Mystery	Comedy	Mystery	NA	6.498581	439031	
8	tt1270797	Action,Sci-Fi,Thriller	Action	Sci-Fi	Thriller	6.699948	275412	
9	tt0451279	Action,Adventure,Fantasy	Action	Adventure	Fantasy	7.499888	487561	
10	tt2226178	Animation	Animation	NA	NA	6.802220	5316	
11	tt0448694	Action,Adventure,Animation	Action	Adventure	Animation	6.596835	134777	
12	tt0429493	Action,Adventure,Thriller	Action	Adventure	Thriller	6.794320	237959	
13	tt0837562	Animation,Comedy,Family	Animation	Comedy	Family	7.095216	209584	
14	tt0398286	Adventure,Animation,Comedy	Adventure	Animation	Comedy	7.794844	372773	
15	tt3971092	Sci-Fi	Sci-Fi	NA	NA	7.070767	821	

In [87]:
```
#set the parameters
fig, ax1 = plt.subplots(figsize=(15,10))

#plot the bar
plt.bar(x = gen_short['genres'], height = gen_short['avg_netprofits'],

#plot the average reviews line
ax2 = gen_short['avg_rating'].plot(secondary_y=True, color = 'red')

#set xlimit
plt.xlim([-0.35, len(gen_short['avg_netprofits'])-0.35])

#set the title
plt.title('Top 20 Profitable Genre Combinations vs Average Customer Rev

#set size of x ticks and rotation
ax1.tick_params(axis='x', labelrotation = 90, labelsize = 12);
plt.yticks(fontsize = 12);

#set axis labels for axis 1
ax1.set_xlabel('Genre Combinations', labelpad = 20, fontdict={'fontsize
ax1.set_ylabel('Average Net Profits in $ millions', labelpad = 20, font

#set y-axis label for axis 2
ax2.set_ylabel('Average Customer Reviews', labelpad= 20, fontdict={'fon

#set the grid to make it easy for reading
plt.grid(True, axis = 'y')

#set legend
ax1.legend(['Net Profits'],loc=1);
ax2.legend(['Avg Reviews'],loc=1, bbox_to_anchor=(0.5, 0.45, 0.495, 0.5

```

Recapping the 5 genres we shortlisted:

1. `Adventure`
2. `Action`
3. `Comedy`
4. `Animation`
5. `Sci-fi`

We can clearly see some good and bad spikes in the graph:

- `Adventure` is the common genre in all the 5 peaks of movie ratings with the grouping of `Fantasy` in 3 of them.
- In the 8 groupings with 7 or below average ratings, `Adventure` was a part of 4 of them which shows that `Adventure` genre on it's own is a mixed fortune. It is the pairings with `Adventure` that play a big part.
- 4 out of the 7 groupings of `Action` ended up having an average rating of 7 or less. That shows that it is difficult to generate good customer reviews with this genre. In all the 3 groupings that were above 7 were with `Adventure` . We will continue to explore `Action` but in context of pairing with `Adventure`
- `Comedy` faired better with 3 out of 5 combinations racking up average ratings higher than 7 in the most profitable combinations. Out of these 3, 2 were pairings with `Adventure`
- `Animation` had an average performance with 2 out of 4 combinations getting above 7 ratings. Both of these were paired with `Comedy` . We will be looking at `Animation` in combination with `Comedy` pairings for better insights going forward.
- `Sci-Fi` did really good on its own or when paired wiht with `Adventure` and `Drama` . Pairings with `Action` proved to bring down the ratings. We will continue to explore `Sci-Fi` as a possible pairing with `Adventure` or other genres.

In light of these insights, we will shift our focus to looking at pairings with `Adventure` and the following genres:

1. `Action`
2. `Comedy`
3. `Animation`
4. `Sci-Fi`

We will take a look at foreign success and movie budgets next to understand what pairings will prove to be the best with `Adventure` .

Should the movie be launched internationally?

To answer this question, we will first take a look the yearly trend of domestic vs foreign profits and then evaluate how the Adventure combinations recognized have fared in the two markets.

In [88]:
```
#make a copy of brt_tn
dom_for = brt_tn.copy()

#preview the results
dom_for.head()
```

Out[88]:

	movie_id	primary_title	original_title	start_year	genres	genre1	genre2
0	tt0359950	The Secret Life of Walter Mitty	The Secret Life of Walter Mitty	2013	Adventure,Comedy,Drama	Adventure	Comedy
1	tt0365907	A Walk Among the Tombstones	A Walk Among the Tombstones	2014	Action,Crime,Drama	Action	Crime
2	tt0369610	Jurassic World	Jurassic World	2015	Action,Adventure,Sci-Fi	Action	Adventure
3	tt0398286	Tangled	Tangled	2010	Adventure,Animation,Comedy	Adventure	Animation
4	tt0401729	John Carter	John Carter	2012	Action,Adventure,Sci-Fi	Action	Adventure

In [89]:
```
#check spread of values for years
dom_for['start_year'].value_counts()
```

Out[89]:
```
2016    158
2015    149
2017    116
2011    114
2014    109
2012    109
2013    108
2010    105
2018     88
2019      5
Name: start_year, dtype: int64
```

In [106]:
```
#group by dates and find average of the profits through SQL Query
#select the columns required for analysis

x = """
SELECT
    start_year,
    SUM("foreignprofit ($ millions)") AS foreign_profit,
    SUM("domesticprofit ($ millions)") AS domestic_profit
FROM dom_for
GROUP BY start_year
ORDER BY start_year ASC;
"""

domestic_foreign = pysqldf(x)

#preview the results
domestic_foreign
```

Out[106]:

	start_year	foreign_profit	domestic_profit
0	2010	5583.0	4267.0
1	2011	7240.0	4720.0
2	2012	9642.0	5938.0
3	2013	9534.0	6866.0
4	2014	8730.0	6040.0
5	2015	9765.0	6745.0
6	2016	13490.0	9730.0
7	2017	14540.0	9280.0
8	2018	8952.0	6318.0
9	2019	80.0	60.0

In [107]:
```
#drop 2019 because of small amount of data
domestic_foreign = domestic_foreign.iloc[0:9]

#preview the data
domestic_foreign
```

Out[107]:

	start_year	foreign_profit	domestic_profit
0	2010	5583.0	4267.0
1	2011	7240.0	4720.0
2	2012	9642.0	5938.0
3	2013	9534.0	6866.0
4	2014	8730.0	6040.0
5	2015	9765.0	6745.0
6	2016	13490.0	9730.0
7	2017	14540.0	9280.0
8	2018	8952.0	6318.0

In [108]:
```
#convert the values to obillions for easy reading
domestic_foreign[['foreign_profit', 'domestic_profit']] = domestic_fore

#preview the results
domestic_foreign
```

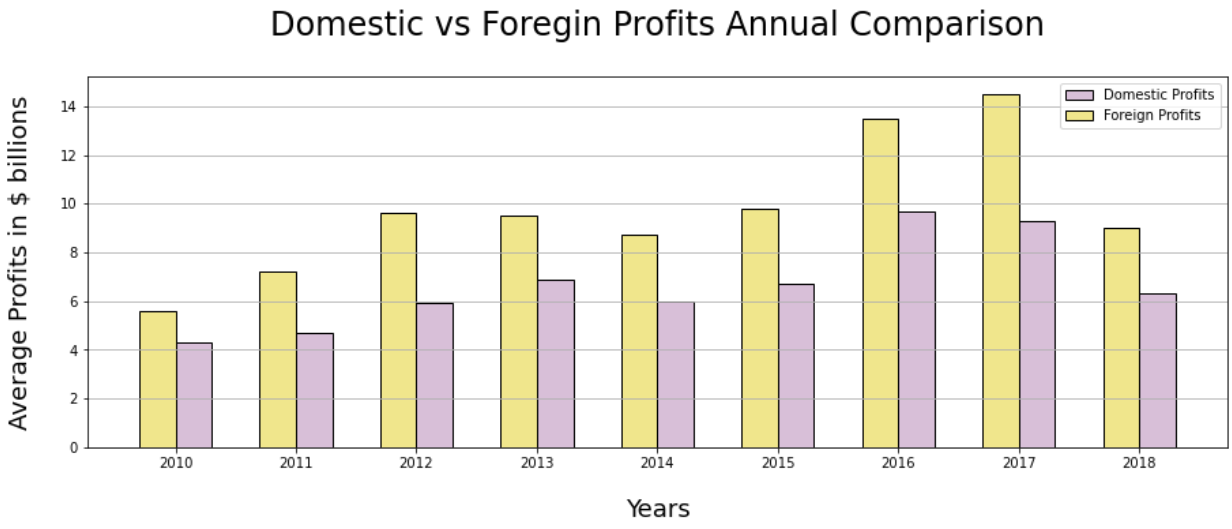
```
/var/folders/qv/0z2v23tn1f1b2fnpqqgqsxch0000gn/T/ipykernel_15142/52901554
5.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-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)
  domestic_foreign[['foreign_profit', 'domestic_profit']] = domestic_fore
ign[['foreign_profit', 'domestic_profit']].round(-2)/1000
```

Out[108]:

	start_year	foreign_profit	domestic_profit
0	2010	5.6	4.3
1	2011	7.2	4.7
2	2012	9.6	5.9
3	2013	9.5	6.9
4	2014	8.7	6.0
5	2015	9.8	6.7
6	2016	13.5	9.7
7	2017	14.5	9.3
8	2018	9.0	6.3

In [109]:

```
#set the width of the bars
width = 0.3

#set the parameters of the plot
fig, ax = plt.subplots(figsize=(15,5));

#the x markers to manipulate the position of the bars
x = np.arange(9)

#plot the domestic profit bars, set the labels and choose the color
ax.bar(x+width/2, domestic_foreign['domestic_profit'], width,
               label='domestic profits', color = 'thistle', edgecolor='b

#plot the foreign profit bars,
#set domestic profit as the bar to show at the bottom,
#set the labels and choose the color
ax.bar(x-width/2 , domestic_foreign['foreign_profit'],
               width, label='foreign profits', color = 'khaki',edgecolor

#change the x-axis labels to the genre names
plt.xticks(x,domestic_foreign['start_year'])

#show the legend
ax.legend(['Domestic Profits', 'Foreign Profits']);

#set the title
plt.title('Domestic vs Foregin Profits Annual Comparison',fontdict={'fo

#set axis labels for axis 1
ax.set_xlabel('Years', labelpad = 20, fontdict={'fontsize':18})
ax.set_ylabel('Average Profits in $ billions', labelpad = 20, fontdict=

#set the grid to make it easy for reading
plt.grid(True, axis = 'y')
```



We can clearly see that foreign profits are at the least **$1 Billion** more than the foreign profits in any year. This shows that foreign profits dominate the net profits every year.

Now lets take a look at the breakout for the specific Advenutre genre combinations that we want to focus on.

Similar to the last analysis, we will cut down our shortlist to the 5 genres and minimum average rating of 7 to pick out the top genre combinations. Then we will evaluate how did these genres perform in the foreign and domestic markets.

In [93]:
```
#Use SQL LIKE statement to match genres column with shortlisted genres
y = """
SELECT *
FROM gen_short
WHERE
    (genres LIKE '%Adventure%')
    And
    ((genres LIKE '%Action%')
    OR
    (genres LIKE '%Comedy%')
    OR
    (genres LIKE '%Animation%')
    OR
    (genres LIKE '%Sci-Fi%'))
    AND
    (avg_rating >= 7)
ORDER BY avg_netprofits DESC;

"""

gen_dom_f = pysqldf(y)
gen_dom_f
```

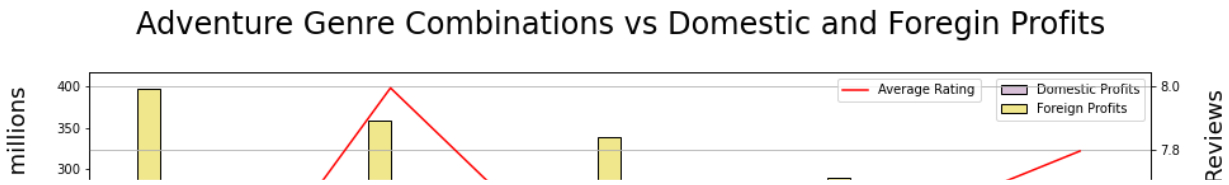
Out[93]:

	movie_id	genres	genre1	genre2	genre3	avg_rating	tot_votes	prodı
0	tt1587310	Action,Adventure,Family	Action	Adventure	Family	7.000000	313975	
1	tt3659388	Adventure,Drama,Sci-Fi	Adventure	Drama	Sci-Fi	7.994725	692288	
2	tt0478970	Action,Adventure,Comedy	Action	Adventure	Comedy	7.295343	513162	
3	tt0451279	Action,Adventure,Fantasy	Action	Adventure	Fantasy	7.499888	487561	
4	tt0398286	Adventure,Animation,Comedy	Adventure	Animation	Comedy	7.794844	372773	

Lets plot this out to compare the different genre combinations and profits. We will use a stacked bar chart to show the domestic and foreign profits comparison.

In [94]:

```
#set the width of the bars
width = 0.1

#set the parameters of the plot
fig, ax = plt.subplots(figsize=(15,5));

#the x markers to manipulate the position of the bars
x = np.arange(5)

#plot the domestic profit bars, set the labels and choose the color
ax.bar(x+width/2, gen_dom_f['avg_domprofits'], width,
               label='domestic profits', color = 'thistle', edgecolor='b

#plot the foreign profit bars,
#set domestic profit as the bar to show at the bottom,
#set the labels and choose the color
ax.bar(x-width/2, gen_dom_f['avg_foreignprofits'],
               width,
               label='foreign profits', color = 'khaki',edgecolor = 'bla

#plot the average rating line
ax1 = gen_dom_f['avg_rating'].plot(secondary_y=True, color='red');

#change the x-axis labels to the genre names
plt.xticks(x,gen_dom_f['genres'])

#show the legend
ax.legend(['Domestic Profits', 'Foreign Profits']);
ax1.legend(['Average Rating'], loc=0, bbox_to_anchor=(0.35, 0.5, 0.495,

#set the title
plt.title('Adventure Genre Combinations vs Domestic and Foregin Profits

#set axis labels for axis 1
ax.set_xlabel('Genre Combinations', labelpad = 20, fontdict={'fontsize'
ax.set_ylabel('Average Profits in $ millions', labelpad = 20, fontdict=

#set y-axis label for axis 2
ax1.set_ylabel('Average Customer Reviews', labelpad= 20, fontdict={'fon

#set the grid to make it easy for reading
plt.grid(True, axis = 'y')
```



Lets take a look at genre specific combinations with `Adventure`

This chart makes it clear that the launch of a movie in foreign merkets is essential to have high profitability and success. In most of the genre combinations, the foreign profits are eiher dominating the total profits or contributing as much as the domestic profits.

Lets see how have our shortlisted genres done in domestic and foreign markets:

The `Adventure` genre has consistently brought in almost $200 million profits in the US market.

The most repeated combination with `Adventure` has been of `Action` with 3 out of the 5 shortlisted.

The next most repeated one has been with `Comedy` with 2 combinations.

`Sci-Fi` and `Animation` both had a single combination each with `Adventure` .

The combination with `Sci-Fi` was the highest rated amongst all 5 aand a close second for net profit with almost $550 million.

Interestingly, the combination of `Action, Adventure, Comedy` didn't end up being the most profitable or the best reviewed amongst the 5 but those are the 3 genres which are showing up to be the most consistent in performance. But this combination did bring in the highest domestic profit amongst the shortlisted genres. Therefore, if he client chooses to not pursue an international launch, this cmbination would be the most reliable to bring in high domestic profits.

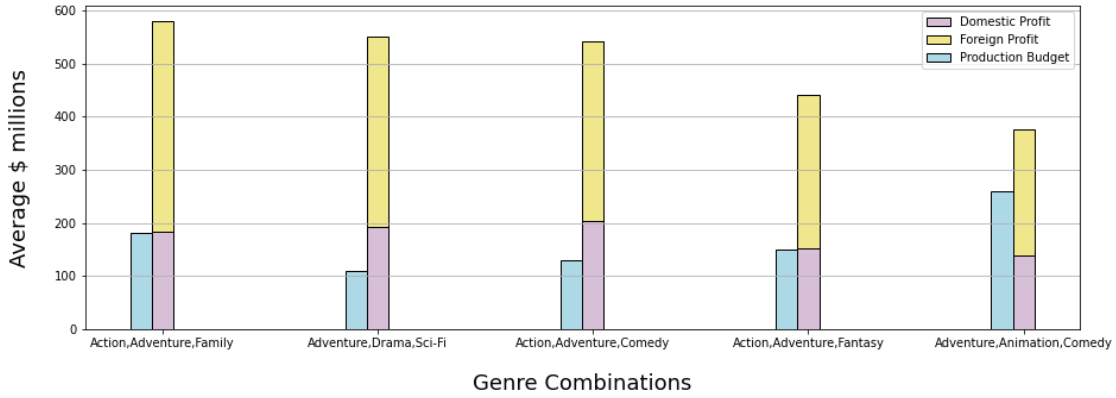
To finally choose which combination will work best, we still need to look at the production budget required for these genres. This will provide the client insight on which genre to choose depending on the money available to the client.

What should the budget target be for the movie?

In [95]:

```
#set the width
width = 0.1

#set the plotting parameters
fig, ax = plt.subplots(figsize=(15,5));

#the x markers to manipulate the position of the bars
x = np.arange(5)

#plot the stacked bar charts
#plot the domestic profit bars, set the labels and choose the color
ax.bar(x, gen_dom_f['avg_domprofits'], width,
               label='domestic profits', color = 'thistle', edgecolor='b

#plot the foreign profit bars,
#set domestic profit as the bar to show at the bottom,
#set the labels and choose the color
ax.bar(x, gen_dom_f['avg_foreignprofits'],
               width, bottom = gen_dom_f['avg_domprofits'],
               label='foreign profits', color = 'khaki',edgecolor = 'bla

#plot the production budget bar
plt.bar(x-width, gen_dom_f['production_budget ($ millions)'],
                label = "Production Budget", color='lightblue',
                width = width, edgecolor='black');

#show the legend
ax.legend(['Domestic Profit', 'Foreign Profit', 'Production Budget'])

#label the x-axis ticks
plt.xticks(x,gen_dom_f['genres']);

#set the title
plt.title('Adventure Genre Combinations vs Domestic and Foregin Profits

#set axis labels for axis 1
ax.set_xlabel('Genre Combinations', labelpad = 20, fontdict={'fontsize'
ax.set_ylabel('Average $ millions', labelpad = 20, fontdict={'fontsize'


#set the grid to make it easy for reading
plt.grid(True, axis = 'y')
```

Adventure Genre Combinations vs Domestic and Foregin Profits vs Production Budget



The highest production budget with the lowest profits was the combination of `Adventure, Animation, Comedy` which rules it out of selection.

The lowest production budget was for `Adventure, Drama, Sci-Fi` and then `Action, Adventure, Comedy` with only a small difference between the two. Both of these genres have similar net profits therefore the financial impacts will also be similar.

This gives the client options on which genre combination to choose from for pursuing the most financially sound option.

Conclusion and Recommendation

The client had 3 goals that we wanted to help achieve:

1. Positive Viewer Response
2. Domestic vs Foreign Launch
3. Profitability

To help achieve these goals, we decided to pose three three questions for our analysis to answer:

1. Which genres bring in the highest ratings and profits?
    - Understand what genre combinations will get the best viewer response and profit margins
2. Should the movie be launched internationally?
    - Understand what will be the most successful mmarket to launch, domestic vs foreign
3. What should the budget target be for the movie?
    - Understand the monetary constraints and expectations for different genres

Our analysis gave us the following insights for the questions posed:

1. Which genres bring in the highest ratings and profits?
    - `Adventure` genre is one of the most common genres to produce high customer reviews and good profit margins but it has to be paired with the correct genres to make this possible.

2. Should the movie be launched internationally?
    - The shortlisted genres were focused arouond `Adventure` because statistically, `Adventure` was the most common genre doing well with reviewers and with profit

margins.
- Between domestic and foreign markets, foreign markets proved to be the more lucrative ones. They contributed significantly to the net profits

3. What should the budget target be for the movie?
    - The shortlisted genres were focused arouond `Adventure` for this analysis.
    - Most of the shortlisted genres had production budgets of $200 million or less except for one exception which was chosen to be ignored.

The recommendations from the analysis are as follows:

The client has 3 options of genres to choose from for **maximum profitability** and **positive viewer reponse**. The pros and cons are listed below with the recommendations:

1. `Action, Adventure, Family` :

    - Net Profit: ~$600 million
        - This genre combinatiion has the most net profits amongst the three final candidates.
    - Average Rating: ~7.0
        - `Family` genre combinations are not poopular in having high average ratings so the chances of getting a **positive viewer response** with the genre combination are low.
    - Production Budget: ~$175 million
        - This genre combination has the highest production budget amongst the final candidates.

2. `Adventure, Drama, Sci-Fi` :

    - Net Profit: ~$550 million
        - This is the second highest net profit amongst the final options
        - `Drama` combinations can garner a lot of **positive viewer responses** but they were a rare occurance in the top 20 most profitable genre combinations therefore are not a guarantee to be highly **profitable**.
    - Average Rating: ~8.0
        - This is the highest average rating amongst the final candidates and well above the average of 6.5 of the dataset and 7.0 of IMDB
        - `Drama` genre combinations are the most popular amongst viewers. They were the most common in the high average rating analysis.
    - Production Budget: ~$100 million
        - This is the lowest production budget amongst the final candidates

3. `Action, Adventure, Comedy` :

    - Net Profit: ~$525 million
        - This is the least amount of net profit amongst the final three options.
        - All of the three genres in this combination were a consistent occurance in the top 20 most profitable genre combinations.
    - Average Rating: ~7.3

    - All of the three genres were the most consistent genres to show up for high average ratings and high proftiability which means that they can garner a lot of **positive viewer response** and be highly **profitable**
- Production Budget: ~$150 million
    - This is the second highest production budget from the final 3 options.

Domestic vs Foreign Launch The financial performance in domestic and foreign markets of the 3 options is given below:

```
1. `Action, Adventure, Family` :
    - Barely break even. Most of the profits are from international
markets
    - Cannot pursue

2. `Adventure, Drama, Sci-Fi` :
    - Domestic profit of ~$200 million.
        - Foreign profit of ~$350 million.

3. `Action, Adventure, Comedy` :
    - Domestic profit of ~$200 million.
        - Foreign profit of ~$325 million.
```

This shows that for a profitable success, international markets can not be ignored and have to be catered to.

With all of that in mind, the final recommendation would be to pursue a film in the genre commbination of `Action, Adventure, Comedy` .

This is because it will meet all 3 of the client's goals. These 3 genres are most common occurances in the high customer reviews list and the most profitable genres list. This increases the statistical chances of the movie having high **profits** and high **positive viewer responses**.

Alongside the recommendation would be to launch this movie in the international market for greater exposure and profits.

Also, the average production budget for this genre is consideraably lower and can be recouped just through the domestic revenue streams which is a guarantee of success.

Ensuring that the movie has high chances of good reviews and has a wide exposure across **foreign markets** will ensure success.

Additionally, the client should consider looking at the potential of merchanidising, streaming platforms and game licences for the movie to increase revenue generation and develop a fan base.