Data Preprocessing

Student name: T.J. Kyner
Student pace: Full time
Instructor name: Abhineet Kulkarni
Cohort: 040521

This notebook is dedicated to exploring, manipulating, and transforming the files available in the `data` folder into a format more conducive for analysis. As a point of reference, the stated business problem is shown below but will be more explicitly answered in the `analysis.ipynb` notebook.

Business Problem

> Microsoft sees all the big companies creating original video content and they want to get in on the fun. They have decided to create a new movie studio, but they don't know anything about creating movies. You are charged with exploring what types of films are currently doing the best at the box office. You must then translate those findings into actionable insights that the head of Microsoft's new movie studio can use to help decide what type of films to create.

Imports

In [1]:
```
from glob import glob
import pandas as pd
import itertools
import numpy as np
```

Available Data

First, let's take a look at all of the different data files available:

In [2]:
```
# Only looking for .csv or .tsv files, not directories
data_files = glob('../data/*.*sv')
data_files
```

Out[2]:
```
['../data\\bom.movie_gross.csv',
 '../data\\imdb.name.basics.csv',
 '../data\\imdb.title.akas.csv',
 '../data\\imdb.title.basics.csv',
 '../data\\imdb.title.crew.csv',
 '../data\\imdb.title.principals.csv',
 '../data\\imdb.title.ratings.csv',
 '../data\\rt.movie_info.tsv',
 '../data\\rt.reviews.tsv',
 '../data\\tmdb.movies.csv',
 '../data\\tn.movie_budgets.csv']
```

Checking the column names in each file by loading it into a pandas DataFrame is a quick way to get a sense of what variables are available and may warrant further investigation.

In [3]:
```
for file in data_files:
    if file.endswith('.tsv'):
        df = pd.read_csv(file, delimiter='\t', encoding='latin')
    else:
        df = pd.read_csv(file)

    # Note: the '\033[1m' and '\033[0m' mark the beginning and end of bold text
    # https://stackoverflow.com/questions/8924173/how-do-i-print-bold-text-in-python/8930747
    print('\033[1m', f'{file[8:]}:', '\033[0m')
    for col in df.columns.values:
        print('-', col)
    print('\n')
```

```
 bom.movie_gross.csv:
- title
- studio
- domestic_gross
- foreign_gross
- year


 imdb.name.basics.csv:
- nconst
- primary_name
- birth_year
- death_year
- primary_profession
- known_for_titles


 imdb.title.akas.csv:
- title_id
- ordering
- title
- region
- language
- types
- attributes
- is_original_title


 imdb.title.basics.csv:
- tconst
- primary_title
- original_title
- start_year
- runtime_minutes
- genres


 imdb.title.crew.csv:
- tconst
- directors
- writers


 imdb.title.principals.csv:
- tconst
- ordering
- nconst
- category
- job
- characters


 imdb.title.ratings.csv:
- tconst
- averagerating
- numvotes


 rt.movie_info.tsv:
- id
- synopsis
- rating
- genre
- director
- writer
- theater_date
- dvd_date
- currency
- box_office
- runtime
```

- studio

rt.reviews.tsv:
- id
- review
- rating
- fresh
- critic
- top_critic
- publisher
- date

tmdb.movies.csv:
- Unnamed: 0
- genre_ids
- id
- original_language
- original_title
- popularity
- release_date
- title
- vote_average
- vote_count

tn.movie_budgets.csv:
- id
- release_date
- movie
- production_budget
- domestic_gross
- worldwide_gross

There appears to be a decent amount of overlap between some of the different files which will be useful for combining them if needed. The following data files appear to contain some of the most pertinent information in relation to the task at hand:

- imdb.title.basics.csv
- imdb.title.ratings.csv
- rt.movie_info.tsv
- tn.movie_budgets.csv

Next, I'll place each of these into their respective variables so I can being working with them.

In [4]:
```
df_imdb_title_basics = pd.read_csv('../data/imdb.title.basics.csv')
df_imdb_title_ratings = pd.read_csv('../data/imdb.title.ratings.csv')
df_rt_movie_info = pd.read_csv('../data/rt.movie_info.tsv', delimiter='\t', encoding='latin')
df_tn_movie_budgets = pd.read_csv('../data/tn.movie_budgets.csv')
```

Summary Information

In [5]:
```
def print_value_counts(df):
    '''
    Description:
    -------------
    Prints the value counts for the top five items in each column of a dataframe.
    The column title will be bold. Does not return anything.


    Paramaters:
    -------------
    df : pandas.DataFrame
        A dataframe with any number of columns or data.


    Example:
    -------------
        >>> df = pandas.DataFrame(data={'col1': [1, 2, 3, 2, 3]})
        >>> print_value_counts(df)

        **col1**
        3    2
        2    2
        1    1
        Name: col1, dtype: int64

    '''
    for col in df.columns:
        print('\033[1m', col, '\033[0m', '\n', df[col].value_counts().head(), '\n\n')
```

IMDb - Title Basics

In [6]:
```
df_imdb_title_basics.head(3)
```

Out[6]:

	tconst	primary_title	original_title	start_year	runtime_minutes	genres
0	tt0063540	Sunghursh	Sunghursh	2013	175.0	Action,Crime,Drama
1	tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	114.0	Biography,Drama
2	tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018	122.0	Drama

In [7]:
```
df_imdb_title_basics.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 146144 entries, 0 to 146143
Data columns (total 6 columns):
 #   Column           Non-Null Count   Dtype
---  ------           --------------   -----
 0   tconst           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
```

In [8]:
```
df_imdb_title_basics.isna().sum()
```

Out[8]:
```
tconst                0
primary_title         0
original_title        21
start_year            0
runtime_minutes    31739
genres              5408
dtype: int64
```

In [9]:
```
print_value_counts(df_imdb_title_basics)
```

```
 tconst
 tt9099640    1
tt5436224    1
tt4898730    1
tt3381982    1
tt2950606    1
Name: tconst, dtype: int64


 primary_title
 Home          24
The Return    20
Broken        20
Homecoming    16
Alone         16
Name: primary_title, dtype: int64


 original_title
 Broken        19
Home          18
The Return    17
Alone         13
Homecoming    13
Name: original_title, dtype: int64


 start_year
 2017    17504
2016    17272
2018    16849
2015    16243
2014    15589
Name: start_year, dtype: int64


 runtime_minutes
 90.0     7131
80.0     3526
85.0     2915
100.0    2662
95.0     2549
Name: runtime_minutes, dtype: int64


 genres
 Documentary     32185
Drama           21486
Comedy           9177
Horror           4372
Comedy,Drama     3519
Name: genres, dtype: int64
```

In [10]:
```
df_imdb_title_basics.describe()
```

Out[10]:

	start_year	runtime_minutes
count	146144.000000	114405.000000
mean	2014.621798	86.187247
std	2.733583	166.360590
min	2010.000000	1.000000
25%	2012.000000	70.000000
50%	2015.000000	87.000000
75%	2017.000000	99.000000
max	2115.000000	51420.000000

Notes:

- The `runtime_minutes` column is missing roughly 22% of its data.
- The maximum value for `runtime_minutes` is equivalent to 857 hours. Further investigation is needed to determine if this is an outlier or incorrect data.
- The rows with missing data in the `genres` column will likely need dropped for any analysis on genres since imputation would be difficult if not impossible.

IMDb - Title Ratings

In [11]:
```
df_imdb_title_ratings.head(3)
```

Out[11]:

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

In [12]:
```
df_imdb_title_ratings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 73856 entries, 0 to 73855
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   tconst         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
```

In [13]:
```
df_imdb_title_ratings.isna().sum()
```

Out[13]:
```
tconst           0
averagerating    0
numvotes         0
dtype: int64
```

In [14]:
```
print_value_counts(df_imdb_title_ratings)
```

```
 tconst
 tt1989435    1
tt1588374    1
tt1584730    1
tt2182267    1
tt4259760    1
Name: tconst, dtype: int64


 averagerating
 7.0    2262
6.6    2251
7.2    2249
6.8    2239
6.5    2221
Name: averagerating, dtype: int64


 numvotes
 6    2875
5    2699
7    2476
8    2167
9    1929
Name: numvotes, dtype: int64
```

In [15]:
```
# Note: the applymap function prevents scientific notation
df_imdb_title_ratings.describe().applymap(lambda x: f'{x:.5f}')
```

Out[15]:

	averagerating	numvotes
count	73856.00000	73856.00000
mean	6.33273	3523.66217
std	1.47498	30294.02297
min	1.00000	5.00000
25%	5.50000	14.00000
50%	6.50000	49.00000
75%	7.40000	282.00000
max	10.00000	1841066.00000

Notes:

- There's no missing data, but the size of this dataframe is about half the size of `df_imdb_title_basics` .

Rotten Tomatoes - Movie Info

In [16]:
```
df_rt_movie_info.head(3)
```

Out[16]:

	id	synopsis	rating	genre	director	writer	theater_date	dvd_date	currency	box_office	runtime	studio
0	1	This gritty, fast-paced, and innovative police...	R	Action and Adventure|Classics|Drama	William Friedkin	Ernest Tidyman	Oct 9, 1971	Sep 25, 2001	NaN	NaN	104 minutes	NaN
1	3	New York City, not-too-distant-future: Eric Pa...	R	Drama|Science Fiction and Fantasy	David Cronenberg	David Cronenberg|Don DeLillo	Aug 17, 2012	Jan 1, 2013	$	600,000	108 minutes	Entertainment One
2	5	Illeana Douglas delivers a superb performance ...	R	Drama|Musical and Performing Arts	Allison Anders	Allison Anders	Sep 13, 1996	Apr 18, 2000	NaN	NaN	116 minutes	NaN

In [17]:
```
df_rt_movie_info.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
```

In [18]:
```
df_rt_movie_info.isna().sum()
```

Out[18]:

```
id                0
synopsis         62
rating            3
genre             8
director        199
writer          449
theater_date    359
dvd_date        359
currency       1220
box_office     1220
runtime          30
studio         1066
dtype: int64
```

In [19]:
```
# Excluding the synopsis column since all of the values will be unique
print_value_counts(df_rt_movie_info.drop(columns=['synopsis']))
```

```
 id
 2000    1
697     1
673     1
674     1
675     1
Name: id, dtype: int64


 rating
 R        521
NR        503
PG        240
PG-13     235
G          57
Name: rating, dtype: int64


 genre
 Drama                                151
Comedy                               110
Comedy|Drama                          80
Drama|Mystery and Suspense            67
Art House and International|Drama     62
Name: genre, dtype: int64


 director
 Steven Spielberg    10
Clint Eastwood        8
William Friedkin      4
Curtis Hanson         4
Jim Jarmusch          4
Name: director, dtype: int64


 writer
 Woody Allen           4
Hong Sang-soo         3
Sylvester Stallone    3
John Hughes           3
Jim Jarmusch          3
Name: writer, dtype: int64


 theater_date
 Jan 1, 1987     8
Jan 1, 1994     5
Nov 20, 2009    4
Jan 1, 1940     4
Jan 1, 1973     4
Name: theater_date, dtype: int64


 dvd_date
 Jun 1, 2004     11
Sep 3, 2002      7
Nov 6, 2001      7
Aug 27, 1997     6
May 22, 2001     6
Name: dvd_date, dtype: int64


 currency
 $    340
Name: currency, dtype: int64


 box_office
 20,900,803    2
32,000,000    2
200,000       2
600,000       2
22,715,908    1
Name: box_office, dtype: int64


 runtime
 90 minutes     72
95 minutes     66
```

```
100 minutes    51
93 minutes     47
96 minutes     43
Name: runtime, dtype: int64
```

```
 studio
 Universal Pictures        35
Paramount Pictures        27
20th Century Fox          26
Sony Pictures Classics    22
Warner Bros. Pictures     21
Name: studio, dtype: int64
```

In [20]:
```
df_rt_movie_info.describe()
```

Out[20]:

	id
count	1560.000000
mean	1007.303846
std	579.164527
min	1.000000
25%	504.750000
50%	1007.500000
75%	1503.250000
max	2000.000000

Notes:

- This dataframe has about 26% of all of its data missing. Certain columns seem to have paired missing data such as `currency` and `box_office`. Since much of this data is contained within other files as well, this is not of too much concern.
- The `id` column could be dropped in favor of the dataframe's standard index since it's not providing any information.
- Certain columns have data that is currently stored as a string and should be converted to numeric data (`box_office`, `runtime`, etc.).

This data was worth exploring but given the amount of missing and overlapping data as well as the lack of a viable way to merge this dataframe with the others, I will not be using this data going forward.

The Numbers - Movie Budgets

In [21]:
```
df_tn_movie_budgets.head(3)
```

Out[21]:

	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

In [22]:
```
df_tn_movie_budgets.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
```

In [23]:
```
df_tn_movie_budgets.isna().sum()
```

Out[23]:
```
id                 0
release_date       0
movie              0
production_budget  0
domestic_gross     0
worldwide_gross    0
dtype: int64
```

In [24]:
```
print_value_counts(df_tn_movie_budgets)
```

```
 id
 4     58
53     58
61     58
65     58
69     58
Name: id, dtype: int64


 release_date
 Dec 31, 2014    24
Dec 31, 2015     23
Dec 31, 2010     15
Dec 31, 2008     14
Dec 31, 2012     13
Name: release_date, dtype: int64


 movie
 Home          3
King Kong      3
Halloween      3
Snitch         2
Twilight       2
Name: movie, dtype: int64


 production_budget
 $20,000,000    231
$10,000,000     212
$30,000,000     177
$15,000,000     173
$25,000,000     171
Name: production_budget, dtype: int64


 domestic_gross
 $0              548
$8,000,000        9
$2,000,000        7
$7,000,000        7
$10,000,000       6
Name: domestic_gross, dtype: int64


 worldwide_gross
 $0             367
$8,000,000       9
$2,000,000       6
$7,000,000       6
$5,000,000       4
Name: worldwide_gross, dtype: int64
```

In [25]:
```
df_tn_movie_budgets.describe()
```

Out[25]:

	id
count	5782.000000
mean	50.372363
std	28.821076
min	1.000000
25%	25.000000
50%	50.000000
75%	75.000000
max	100.000000

Notes:

- While there's no missing data, some of the top value counts warrant further investigation. Specifically, are there actually hundreds of movies that didn't make any money at the box office at all? Or are those just placeholder values?
- The `id` column could be dropped in favor of the dataframe's standard index since it's not providing any information.
- The `production_budget`, `domestic_gross`, and `worldwide_gross` columns should be converted to numeric data.
- A `foreign_gross` column could be calculated by subtracting `domestic_gross` from `worldwide_gross`.

Cleaning the Data

In this section, I address some of the questions and concerns about the data I explored in the **Summary Information** section and make adjustments to the data as needed.

Cleaning IMDb Title Basics

To start, I want to investigate the movies with extremely long runtimes. Based on general knowledge of how long typical movies are, there should be relatively few that exceed four hours.

In [26]:
```
df_long_runtimes = df_imdb_title_basics.loc[df_imdb_title_basics['runtime_minutes'] > 240]
df_long_runtimes
```

Out[26]:

	tconst	primary_title	original_title	start_year	runtime_minutes	genres
70	tt0396123	Den milde smerte	Den milde smerte	2010	280.0	Drama
1199	tt10094362	The Blood Will Murder Roses	The Blood Will Murder Roses	2014	288.0	Romance
1958	tt10189122	Reading in/Reading out	Reading in/Reading out	2019	260.0	Documentary
2422	tt10244756	Ang hupa	Ang hupa	2019	276.0	Sci-Fi
3799	tt10366986	3 Games to Glory VI	3 Games to Glory VI	2019	350.0	Sport
...	...	...	...	...	...	...
143603	tt9552194	The Freshman Experience	The Freshman Experience	2017	447.0	Drama
143844	tt9591836	The Greatest Adventure: The Book of Dragons	The Greatest Adventure: The Book of Dragons	2018	244.0	Animation
143903	tt9602094	A Smartphone User's Guide to Etiquette	A Smartphone User's Guide to Etiquette	2018	250.0	Musical
144951	tt9743020	Beauty Lives in Freedom	Beauty Lives in Freedom	2018	330.0	Documentary
145184	tt9782956	The Phineas And Ferb Show	The Phineas And Ferb Show	2018	285.0	Comedy

207 rows × 6 columns

In [27]:
```
df_long_runtimes.genres.value_counts()
```

Out[27]:
```
Documentary               80
Drama                     30
Action                     6
Documentary,History        6
Music                      4
                          ..
Drama,Music                1
Mystery                    1
Documentary,Drama,News     1
Family                     1
Animation,Sci-Fi           1
Name: genres, Length: 62, dtype: int64
```

As expected, there are very few movies that have runtimes exceeding four hours. These movies are typically obscure documentaries and are not relevant to what is being investigated. As a result, these rows will be dropped. While there is also a high number of rows that are missing runtime information as noted in the Summary Information section, they still contain information on the titles and genres which can be useful. Therefore, those rows will be kept.

In [28]:
```
df_cleaned_imdb_title_basics = df_imdb_title_basics.drop(index=df_long_runtimes.index)

# Checking to make sure no movies over four hours are left
df_cleaned_imdb_title_basics.loc[df_cleaned_imdb_title_basics['runtime_minutes'] > 240]
```

Out[28]:

	tconst	primary_title	original_title	start_year	runtime_minutes	genres

The number of rows missing genre data is relatively low compared to the overall dataset and therefore I am comfortable with simply dropping them given my anticipation that genre data will play a critical role in my analysis. However, those rows *with* genre data also require cleaning since multiple genres can be applied to one movie.

In [29]:
```
# Dropping missing genre rows
df_cleaned_imdb_title_basics.dropna(subset=['genres'], inplace=True)
df_cleaned_imdb_title_basics.isna().sum()
```

Out[29]:
```
tconst                0
primary_title         0
original_title        2
start_year            0
runtime_minutes    28503
genres                0
dtype: int64
```

In [30]:
```
# Gets a list of each unique genre
genres = set(list(itertools.chain(*[g.split(',') for g in df_cleaned_imdb_title_basics['genres']])))
genres
```

Out[30]:
```
{'Action',
 'Adult',
 'Adventure',
 'Animation',
 'Biography',
 'Comedy',
 'Crime',
 'Documentary',
 'Drama',
 'Family',
 'Fantasy',
 'Game-Show',
 'History',
 'Horror',
 'Music',
 'Musical',
 'Mystery',
 'News',
 'Reality-TV',
 'Romance',
 'Sci-Fi',
 'Short',
 'Sport',
 'Talk-Show',
 'Thriller',
 'War',
 'Western'}
```

In [31]:
```
# Adding a column for each genre with placeholder values of 0
for genre in genres:
    num_rows = df_cleaned_imdb_title_basics.shape[0]
    df_cleaned_imdb_title_basics[genre] = np.zeros(shape=num_rows)

df_cleaned_imdb_title_basics.head(3)
```

Out[31]:

	tconst	primary_title	original_title	start_year	runtime_minutes	genres	Fantasy	Short	Animation	Sci-Fi	...	Drama	Adult	Talk-Show	War	Sport	I
0	tt0063540	Sunghursh	Sunghursh	2013	175.0	Action,Crime,Drama	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	
1	tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	114.0	Biography,Drama	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	
2	tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018	122.0	Drama	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	

3 rows × 33 columns

Now that there's an individual column for each unique genre, I need to iterate over each row and change the value of the movie's matching genres from 0 to 1. Afterwards, the `genres` column can be dropped. The code below does all of that but takes several minutes to run due to the size of the dataframe. To prevent waiting for the cell to execute every time this notebook is run, I have exported the cleaned dataframe to `../data/cleaned/df_cleaned_imdb_title_basics.csv`. That file will be pulled in and used for the remainder of the notebook.

In [32]:
```
# Note: this cell takes several minutes to fully execute and has been commented out
# Updating each genre column with a 1 if the genre matches the movie
# for index, row in df_cleaned_imdb_title_basics.iterrows():
#     if row['genres']:
#         for genre in row['genres'].split(','):
#             df_cleaned_imdb_title_basics.loc[index, genre] = 1

# df_cleaned_imdb_title_basics.drop(columns=['genres'], inplace=True)
```

In [33]:
```
df_cleaned_imdb_title_basics = pd.read_csv('../data/cleaned/df_cleaned_imdb_title_basics.csv')
df_cleaned_imdb_title_basics.head(3)
```

Out[33]:

	tconst	primary_title	original_title	start_year	runtime_minutes	Game-Show	Mystery	Musical	Family	War	...	Adventure	Fantasy	Thriller	Animation	Biograp
0	tt0063540	Sunghursh	Sunghursh	2013	175.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	
1	tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	114.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	
2	tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018	122.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	

3 rows × 32 columns

Cleaning IMDb Title Ratings

No adjustments needed. For the sake of consistency, this dataframe will also be exported as a "cleaned" version under the `../data/cleaned/` directory.

In [34]:
```
df_imdb_title_ratings.to_csv('../data/cleaned/df_cleaned_imdb_title_ratings.csv', index=False)
```

In [35]:
```
df_cleaned_imdb_title_ratings = pd.read_csv('../data/cleaned/df_cleaned_imdb_title_ratings.csv')
df_cleaned_imdb_title_ratings.head(3)
```

Out[35]:

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

Cleaning The Numbers Movie Budgets

This dataframe is one of the most critical for answering the task at hand since it contains revenue and expense information that can be used to calculate ROI metrics in the analysis notebook.

To start, I will remove the unnecessary `id` column.

In [36]:
```
df_cleaned_tn_movie_budgets = df_tn_movie_budgets.drop(columns=['id'])
df_cleaned_tn_movie_budgets.head(3)
```

Out[36]:

	release_date	movie	production_budget	domestic_gross	worldwide_gross
0	Dec 18, 2009	Avatar	$425,000,000	$760,507,625	$2,776,345,279
1	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	$410,600,000	$241,063,875	$1,045,663,875
2	Jun 7, 2019	Dark Phoenix	$350,000,000	$42,762,350	$149,762,350

Next, I need to convert the `production_budget`, `domestic_gross`, and `worldwide_gross` columns into numeric data types.

In [37]:
```
cols_to_convert = ['production_budget', 'domestic_gross', 'worldwide_gross']
for col in cols_to_convert:
    func = lambda x: int(x.replace('$', '').replace(',', ''))
    df_cleaned_tn_movie_budgets[col] = df_cleaned_tn_movie_budgets[col].apply(func)
```

In [38]:
```
df_cleaned_tn_movie_budgets.info()
```

```
<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   object
 1   movie              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(3), object(2)
memory usage: 226.0+ KB
```

In [39]:
```
df_cleaned_tn_movie_budgets.head(3)
```

Out[39]:

	release_date	movie	production_budget	domestic_gross	worldwide_gross
0	Dec 18, 2009	Avatar	425000000	760507625	2776345279
1	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	410600000	241063875	1045663875
2	Jun 7, 2019	Dark Phoenix	350000000	42762350	149762350

There are two additional columns that I would like to insert into this dataframe:

1. `release_year`: a column that contains just the release year instead of the full date. This will be useful when combining the data to ensure that both the title of the movie *and* the release year match up.
2. `foreign_gross`: a column that contains the amount of revenue from foreign box offices. Calculated as `worldwide_gross` less `domestic_gross`

In [40]:
```
release_years = df_cleaned_tn_movie_budgets['release_date'].apply(lambda x: int(x[-4:]))
df_cleaned_tn_movie_budgets.insert(loc=1, column='release_year', value=release_years)
df_cleaned_tn_movie_budgets.head(3)
```

Out[40]:

	release_date	release_year	movie	production_budget	domestic_gross	worldwide_gross
0	Dec 18, 2009	2009	Avatar	425000000	760507625	2776345279
1	May 20, 2011	2011	Pirates of the Caribbean: On Stranger Tides	410600000	241063875	1045663875
2	Jun 7, 2019	2019	Dark Phoenix	350000000	42762350	149762350

In [41]:
```
foreign_gross = df_cleaned_tn_movie_budgets['worldwide_gross'] - df_cleaned_tn_movie_budgets['domestic_gross']
df_cleaned_tn_movie_budgets.insert(loc=5, column='foreign_gross', value=foreign_gross)
df_cleaned_tn_movie_budgets.head(3)
```

Out[41]:

	release_date	release_year	movie	production_budget	domestic_gross	foreign_gross	worldwide_gross
0	Dec 18, 2009	2009	Avatar	425000000	760507625	2015837654	2776345279
1	May 20, 2011	2011	Pirates of the Caribbean: On Stranger Tides	410600000	241063875	804600000	1045663875
2	Jun 7, 2019	2019	Dark Phoenix	350000000	42762350	107000000	149762350

Finally, I will also save this dataframe as a cleaned version for easier access.

In [42]:
```
df_cleaned_tn_movie_budgets.to_csv('../data/cleaned/df_cleaned_tn_movie_budgets.csv', index=False)
```

In [43]:
```
df_cleaned_tn_movie_budgets = pd.read_csv('../data/cleaned/df_cleaned_tn_movie_budgets.csv')
df_cleaned_tn_movie_budgets.head(3)
```

Out[43]:

	release_date	release_year	movie	production_budget	domestic_gross	foreign_gross	worldwide_gross
0	Dec 18, 2009	2009	Avatar	425000000	760507625	2015837654	2776345279
1	May 20, 2011	2011	Pirates of the Caribbean: On Stranger Tides	410600000	241063875	804600000	1045663875
2	Jun 7, 2019	2019	Dark Phoenix	350000000	42762350	107000000	149762350

Combining the Data

Now that I have three cleaned dataframes, I intend to merge them together in a single dataframe for use in the `analysis.ipynb` notebook. To ensure that the correct dataframes are being used, I'll first reread each of the cleaned .csv files into their respective variables.

In [44]:
```
df_cleaned_imdb_title_basics = pd.read_csv('../data/cleaned/df_cleaned_imdb_title_basics.csv')
df_cleaned_imdb_title_ratings = pd.read_csv('../data/cleaned/df_cleaned_imdb_title_ratings.csv')
df_cleaned_tn_movie_budgets = pd.read_csv('../data/cleaned/df_cleaned_tn_movie_budgets.csv')
```

The two IMDb dataframes share a unique identifier for each movie in the `tconst` column. Merging these two togething is fairly straightforward. Since `df_cleaned_imdb_title_basics` contains roughly doubly the amount of titles as `df_cleaned_imdb_title_ratings` does, it will serve as the left table in the merge.

In [45]:
```
df_merged_imdb = df_cleaned_imdb_title_basics.merge(df_cleaned_imdb_title_ratings, how='left', on='tconst')
print('Shape:', df_merged_imdb.shape)
df_merged_imdb.head(3)
```

```
Shape: (140536, 34)
```

Out[45]:

	tconst	primary_title	original_title	start_year	runtime_minutes	Game-Show	Mystery	Musical	Family	War	...	Thriller	Animation	Biography	Crime	Romance
0	tt0063540	Sunghursh	Sunghursh	2013	175.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	1.0	0.0
1	tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	114.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0
2	tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018	122.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0

3 rows × 34 columns

Merging `df_cleaned_tn_movie_budgets` to the newly created `df_merged_imdb` dataframe is a bit trickier since there is no unique identifier linking the two together. However, using both the movie title as well as the release year should work. Including the release year is important since some movies may have the same title if they were released far enough apart.

In [46]:
```
df_merged_final = df_merged_imdb.merge(df_cleaned_tn_movie_budgets, how='left', left_on=['primary_title', 'start_year'],
                                       right_on=['movie', 'release_year'])

# Showing just the rows with worldwide_gross data to check the merge
df_merged_final.loc[df_merged_final['worldwide_gross'] > 0].head()
```

Out[46]:

	tconst	primary_title	original_title	start_year	runtime_minutes	Game-Show	Mystery	Musical	Family	War	...	Drama	averagerating	numvotes	release_da
19	tt0249516	Foodfight!	Foodfight!	2012	91.0	0.0	0.0	0.0	0.0	0.0	...	0.0	1.9	8248.0	Dec 31, 20
48	tt0359950	The Secret Life of Walter Mitty	The Secret Life of Walter Mitty	2013	114.0	0.0	0.0	0.0	0.0	0.0	...	1.0	7.3	275300.0	Dec 25, 20
52	tt0365907	A Walk Among the Tombstones	A Walk Among the Tombstones	2014	114.0	0.0	0.0	0.0	0.0	0.0	...	1.0	6.5	105116.0	Sep 19, 20
54	tt0369610	Jurassic World	Jurassic World	2015	124.0	0.0	0.0	0.0	0.0	0.0	...	0.0	7.0	539338.0	Jun 12, 20
56	tt0376136	The Rum Diary	The Rum Diary	2011	119.0	0.0	0.0	0.0	0.0	0.0	...	1.0	6.2	94787.0	Oct 28, 20

5 rows × 41 columns

Exporting the Results

With all of the pertinent information consolidated into one dataframe, the final step is to save it for use in `analysis.ipynb` .

In [47]:
```
df_merged_final.to_csv('../data/cleaned/df_merged_final.csv', index=False)
```

In [48]:
```
df_merged_final = pd.read_csv('../data/cleaned/df_merged_final.csv')
df_merged_final.head(3)
```

Out[48]:

	tconst	primary_title	original_title	start_year	runtime_minutes	Game-Show	Mystery	Musical	Family	War	...	Drama	averagerating	numvotes	release_date
0	tt0063540	Sunghursh	Sunghursh	2013	175.0	0.0	0.0	0.0	0.0	0.0	...	1.0	7.0	77.0	NaN
1	tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	114.0	0.0	0.0	0.0	0.0	0.0	...	1.0	7.2	43.0	NaN
2	tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018	122.0	0.0	0.0	0.0	0.0	0.0	...	1.0	6.9	4517.0	NaN

3 rows × 41 columns

The project continues in `analysis.ipynb` where the results of this notebook are analyzed.