In [1]:
```
# importing modules
import os
import pandas as pd
import numpy as np
from glob import glob
import tmdbsimple as tmdb
import requests
```
executed in 299ms, finished 13:39:58 2021-05-01

In [2]:
```
csv_list = glob("./dataFiles/*.csv")
csv_list
```
executed in 12ms, finished 13:39:58 2021-05-01

Out[2]:
```
['./dataFiles\\bom.movie_gross.csv',
 './dataFiles\\expanded_tmdb.csv',
 './dataFiles\\movie_collection.csv',
 './dataFiles\\name.basics.csv',
 './dataFiles\\title.akas.csv',
 './dataFiles\\title.basics.csv',
 './dataFiles\\title.crew.csv',
 './dataFiles\\title.principals.csv',
 './dataFiles\\title.ratings.csv',
 './dataFiles\\tmdb.movies.csv',
 './dataFiles\\tn.movie_budgets.csv']
```

In [3]:
```
cleaned_filenames = [filename.split('\\')[1].replace('.csv', '').replace('.', '_'
                     for filename in csv_list]
```
executed in 18ms, finished 13:39:59 2021-05-01

In [4]:
```
orig_dfs = {}
for idx, file in enumerate(csv_list):
    orig_dfs.update({cleaned_filenames[idx]: pd.read_csv(file)})
```
executed in 2.02s, finished 13:40:01 2021-05-01

In [5]:
```
orig_dfs['bom_movie_gross'].head()
```
executed in 14ms, finished 13:40:01 2021-05-01

Out[5]:

| | 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 |

Now that we have all the files loaded into dataframes in an easily accessible structure, lets investigate them all to see what information we have available and make connections between the data included in each file. Besides, we can see what modifications we may need to make to data

types, to account for missing values, etc. Going forward we will get a lot of information to keep track of, so I will make a separate excel doc for use as a reference / table schema to relate these dataframes in the future.

In [6]:
```
for item in orig_dfs.items():
        print("===============New DF===============")
        print(f'Title:  {item[0]}')
        print(item[1].info())
        print("================End DF===============")
```

executed in 320ms, finished 13:40:02 2021-05-01

```
===============New DF===============
Title:  bom_movie_gross
<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
None
================End DF===============
===============New DF===============
Title:  expanded_tmdb
<class 'pandas.core.frame.DataFrame'>
```

So far, we can notice that, based on column names, there are some potential connections between the different dataframes. Additionally, there are columns that are hard to interpret from their labels, and there are some missing a large majority of values. Lastly, we can note that there are columns with data types that might be undesirable, such as gross columns stored as strings, and date columns stored as objs or ints/floats. First, to get a better understanding of what is in each column, lets look at the head of each dataframe (since print(df.head()) doesn't look as nice in the notebook, create cell for each one instead).

In [7]:
```
orig_dfs['bom_movie_gross'].head()
```

executed in 15ms, finished 13:40:02 2021-05-01

Out[7]:

| | 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 |

In [8]:
```
orig_dfs['name_basics'].head()
```
executed in 14ms, finished 13:40:02 2021-05-01

Out[8]:

| | nconst | 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_departmer |
| 2 | nm0062070 | Bruce Baum | NaN | NaN | miscellaneous,actor,write |
| 3 | nm0062195 | Axel Baumann | NaN | NaN | camera_department,cinematographer,art_departmer |
| 4 | nm0062798 | Pete Baxter | NaN | NaN | production_designer,art_department,set_decorat |

In [9]:
```
orig_dfs['title_akas'].head()
```
executed in 15ms, finished 13:40:02 2021-05-01

Out[9]:

| | title_id | ordering | title | region | language | types | attributes | is_original_title |
|---|---|---|---|---|---|---|---|---|
| 0 | tt0369610 | 10 | Джурасик свят | BG | bg | NaN | NaN | 0.0 |
| 1 | tt0369610 | 11 | Jurashikku warudo | JP | NaN | imdbDisplay | NaN | 0.0 |
| 2 | tt0369610 | 12 | Jurassic World: O Mundo dos Dinossauros | BR | NaN | imdbDisplay | NaN | 0.0 |
| 3 | tt0369610 | 13 | O Mundo dos Dinossauros | BR | NaN | NaN | short title | 0.0 |
| 4 | tt0369610 | 14 | Jurassic World | FR | NaN | imdbDisplay | NaN | 0.0 |

In [10]:
```
orig_dfs['title_basics'].head()
```
executed in 15ms, finished 13:40:03 2021-05-01

Out[10]:

| | 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 |
| 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 [11]:
```
orig_dfs['title_crew'].head()
```
executed in 14ms, finished 13:40:03 2021-05-01

Out[11]:

| | tconst | directors | writers |
|---|---|---|---|
| 0 | tt0285252 | nm0899854 | nm0899854 |
| 1 | tt0438973 | NaN | nm0175726,nm1802864 |
| 2 | tt0462036 | nm1940585 | nm1940585 |
| 3 | tt0835418 | nm0151540 | nm0310087,nm0841532 |
| 4 | tt0878654 | nm0089502,nm2291498,nm2292011 | nm0284943 |

In [12]:
```
orig_dfs['title_principals'].head()
```
executed in 14ms, finished 13:40:03 2021-05-01

Out[12]:

| | tconst | ordering | nconst | category | job | characters |
|---|---|---|---|---|---|---|
| 0 | tt0111414 | 1 | nm0246005 | actor | NaN | ["The Man"] |
| 1 | tt0111414 | 2 | nm0398271 | director | NaN | NaN |
| 2 | tt0111414 | 3 | nm3739909 | producer | producer | NaN |
| 3 | tt0323808 | 10 | nm0059247 | editor | NaN | NaN |
| 4 | tt0323808 | 1 | nm3579312 | actress | NaN | ["Beth Boothby"] |

In [13]:
```
orig_dfs['title_ratings'].head()
```
executed in 14ms, finished 13:40:03 2021-05-01

Out[13]:

| | tconst | 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 [14]:
```
orig_dfs['tmdb_movies'].head()
```
executed in 15ms, finished 13:40:04 2021-05-01

Out[14]:

| | 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 |

In [15]:
```
orig_dfs['tn_movie_budgets'].head(10)
```
executed in 15ms, finished 13:40:04 2021-05-01

Out[15]:

| | 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 |
| **5** | 6 | Dec 18, 2015 | Star Wars Ep. VII: The Force Awakens | $306,000,000 | $936,662,225 | $2,053,311,220 |
| **6** | 7 | Apr 27, 2018 | Avengers: Infinity War | $300,000,000 | $678,815,482 | $2,048,134,200 |
| **7** | 8 | May 24, 2007 | Pirates of the Caribbean: At Worldâ□□s End | $300,000,000 | $309,420,425 | $963,420,425 |
| **8** | 9 | Nov 17, 2017 | Justice League | $300,000,000 | $229,024,295 | $655,945,209 |
| **9** | 10 | Nov 6, 2015 | Spectre | $300,000,000 | $200,074,175 | $879,620,923 |

Seeing these, there are some obvious joins / connections to make on things we can be nearly confident in referencing the same thing, such as on identifiers tconst and nconst. There are some columns we are somewhat confident match but will have to be careful about, such as the different

movie title columns, and some columns we are not sure yet referencing the same movie (or have equivalent values if they do) but potentially could be, such as the ratings and quoted domestic and foreign gross values. Lastly, there are some extra indexing columns we can drop, such as unnamed and id in the last two tables. The ordering columns as well perhaps, as they seem to index rows for multiple records for the same title. Let's get rid of cluttering rows we know aren't useful first. Then we can look at combining dataframes / eliminating other columns.

In [16]:
```
orig_dfs['tmdb_movies'].drop(columns = ['Unnamed: 0'], inplace = True)
orig_dfs['tn_movie_budgets'].drop(columns = ['id'], inplace = True)
orig_dfs['title_principals'].drop(columns = ['ordering'], inplace = True)
orig_dfs['title_akas'].drop(columns = ['ordering'], inplace = True)
```

executed in 46ms, finished 13:40:04 2021-05-01

Now, let's look at other problem columns and fix them up. Starting with bom_movie_gross and moving down the line, we see that in that dataframe there is a studio column that could be more interpretable, since they are currently abbreviations. Let's first see what all the unique abbreviations are.

In [17]:
```
orig_dfs['bom_movie_gross'].studio.unique()
```

executed in 14ms, finished 13:40:04 2021-05-01

Out[17]:
```
array(['BV', 'WB', 'P/DW', 'Sum.', 'Par.', 'Uni.', 'Fox', 'Wein.', 'Sony',
       'FoxS', 'SGem', 'WB (NL)', 'LGF', 'MBox', 'CL', 'W/Dim.', 'CBS',
       'Focus', 'MGM', 'Over.', 'Mira.', 'IFC', 'CJ', 'NM', 'SPC', 'ParV',
       'Gold.', 'JS', 'RAtt.', 'Magn.', 'Free', '3D', 'UTV', 'Rela.',
       'Zeit.', 'Anch.', 'PDA', 'Lorb.', 'App.', 'Drft.', 'Osci.', 'IW',
       'Rog.', nan, 'Eros', 'Relbig.', 'Viv.', 'Hann.', 'Strand', 'NGE',
       'Scre.', 'Kino', 'Abr.', 'CZ', 'ATO', 'First', 'GK', 'FInd.',
       'NFC', 'TFC', 'Pala.', 'Imag.', 'NAV', 'Arth.', 'CLS', 'Mont.',
       'Olive', 'CGld', 'FOAK', 'IVP', 'Yash', 'ICir', 'FM', 'Vita.',
       'WOW', 'Truly', 'Indic.', 'FD', 'Vari.', 'TriS', 'ORF', 'IM',
       'Elev.', 'Cohen', 'NeoC', 'Jan.', 'MNE', 'Trib.', 'Rocket',
       'OMNI/FSR', 'KKM', 'Argo.', 'SMod', 'Libre', 'FRun', 'WHE', 'P4',
       'KC', 'SD', 'AM', 'MPFT', 'Icar.', 'AGF', 'A23', 'Da.', 'NYer',
       'Rialto', 'DF', 'KL', 'ALP', 'LG/S', 'WGUSA', 'MPI', 'RTWC', 'FIP',
       'RF', 'ArcEnt', 'PalUni', 'EpicPics', 'EOne', 'LD', 'AF', 'TFA',
       'Myr.', 'BM&DH', 'SEG', 'PalT', 'Outs', 'OutF', 'BSM', 'WAMCR',
       'PM&E', 'A24', 'Cdgm.', 'Distrib.', 'Imax', 'PH', 'HTR', 'ELS',
       'PI', 'E1', 'TVC', 'FEF', 'EXCL', 'MSF', 'P/108', 'FCW', 'XL',
       'Shout!', 'SV', 'CE', 'VPD', 'KE', 'Saban', 'CF&SR', 'Triu', 'DR',
       'Crnth', 'Ampl.', 'CP', 'Proud', 'BGP', 'Abk.', 'DLA', 'B360',
       'BWP', 'SEA', 'RME', 'KS', 'VE', 'LGP', 'EC', 'FUN', 'STX', 'AR',
       'BG', 'PFR', 'BST', 'BH Tilt', 'BSC', 'U/P', 'UHE', 'CLF', 'FR',
       'AaF', 'Orch.', 'Alc', 'PBS', 'SHO', 'Grav.', 'Gathr', 'Asp.',
       'ADC', 'Rel.', 'SM', 'AZ', 'UEP', 'ITL', 'TA', 'MR', 'BBC',
       'CFilms', 'Part.', 'FOR', 'TAFC', 'JBG', 'PNT', 'CineGalaxy',
       'Fathom', 'Zee', 'Men.', 'YFG', 'Gaatri', 'Mon', 'Ghop',
       'Cleopatra', 'Dreamwest', 'SDS', 'Linn', 'Electric', 'Jampa', 'HC',
       'GrtIndia', 'Neon', 'ENTMP', 'Good Deed', 'ParC', 'Aviron',
       'Annapurna', 'Amazon', 'Affirm', 'MOM', 'Orion', 'CFI', 'UTMW',
       'Crimson', 'CAVU', 'EF', 'Arrow', 'Hiber', 'Studio 8',
       'Global Road', 'Trafalgar', 'Greenwich', 'Spanglish', 'Blue Fox',
       'RLJ', 'Swen', 'PackYourBag', 'Gaum.', 'Grindstone',
       'Conglomerate', 'MUBI', 'Darin Southa', 'Super', 'CARUSEL', 'PDF',
       'Synergetic'], dtype=object)
```

Okay, well, there's alot. Most or some of them probably do not exist anymore.. maybe we will wait on this.

A useful discovery from the The Movie Database API page for turning genre_id to interpretable strings in that table: MOVIES Action 28 Adventure 12 Animation 16 Comedy 35 Crime 80 Documentary 99 Drama 18 Family 10751 Fantasy 14 History 36 Horror 27 Music 10402 Mystery 9648 Romance 10749 Science Fiction 878 TV Movie 10770 Thriller 53 War 10752 Western 37

We could use this to replace the genre ID's with text.

Now, we may be able to condense the number of tables while leaving appropriate opportunity for joins. We just need to start checking where records are truly pointing toward the same information, then bringing information together. For example, we can imagine for a unique movie, the following information would be useful to collect for generating visualizations: tconst(table primary key), primary_title (for display purposes), genre(s), release_year, run_time, production_studio,

production_budget, domestic_gross, domestic_ROI, worldwide_gross, worldwide_ROI. We can collect the rest of the useful data in two other tables probably. Brainstorming, I would be interested in one table having columns tconst, rating, num_votes and popularity, and a second table having tconst, and then columns for each crew job type (producer, director, writer, actor, etc.) containing lists of the crew of that type for that film. Let's look at the sizes of each dataframe to see if there are ones we can join to / build off of while retaining the maximum amount of information.

In [18]:
```
for item in orig_dfs.items():
        print("===============New DF===============")
        print(f'Title:  {item[0]}')
        print(item[1].shape)
        print("================End DF===============")
```

executed in 14ms, finished 13:40:05 2021-05-01

```
===============New DF===============
Title:  bom_movie_gross
(3387, 5)
================End DF===============
===============New DF===============
Title:  expanded_tmdb
(26517, 15)
================End DF===============
===============New DF===============
Title:  movie_collection
(2373, 12)
================End DF===============
===============New DF===============
Title:  name_basics
(606648, 6)
================End DF===============
===============New DF===============
Title:  title_akas
(331703, 7)
================End DF===============
===============New DF===============
Title:  title_basics
(146144, 6)
================End DF===============
===============New DF===============
Title:  title_crew
(146144, 3)
================End DF===============
===============New DF===============
Title:  title_principals
(1028186, 5)
================End DF===============
===============New DF===============
Title:  title_ratings
(73856, 3)
================End DF===============
===============New DF===============
Title:  tmdb_movies
(26517, 9)
================End DF===============
===============New DF===============
Title:  tn_movie_budgets
(5782, 5)
================End DF===============
```

So we are seeing that we have data for many more movies than we have budget information for. Since we are trying to produce profitable movies, being able to determine indicators driving profit is necessary, unless we can become very confident that something is an effective proxy for profit such as rating or popularity. It is possible that between the tables with budgetary information, i.e.

'bom_movie_gross' and 'tn_movie_budgets' we can cover more movies than either alone, and hopefully fill out some of the other information with the other tables we have. However, we only have the budget information for those from the 'tn_movie_budgets' table, which means we can only calculate ROI's for movies from that table, unless we can find the information elsewhere OR are happy filling in values for NULLs. We can look into that but it is hard to imagine that budget can easily be predicted and its assigned value has a huge impact on ROI.

Upon doing some research, I have determined by looking at lists on the The Numbers and The Movie Database websites that the domestic, foreign, and worldwide gross numbers from each site are close enough that they can be considered the same. Additionally, the 'worldwide_gross' from the 'tn_movie_budgets' table is the same as the sum of the 'domestic' and 'foreign_gross' columns of the 'bom_movie_gross' table, so we can make an aggregate column in the Box Office Mojo table and also produce a 'foreign_gross' column for the The Numbers table. Down the line, it may be possible to scrape The Numbers or Box Office Mojo for budgetary information for more movies.

Additionally, there is a way to query TMDB using an IMDB tconst through a python module called tmdbsimple:

import tmdbsimple as tmdb tmdb.API_KEY = '' response = tmdb.Find('tt0266543').info(external_source='imdb_id') response

{'movie_results': [{'id': 12, 'video': False, 'vote_count': 13478, 'vote_average': 7.8, 'title': 'Finding Nemo', 'release_date': '2003-05-30', 'original_language': 'en', 'original_title': 'Finding Nemo', 'genre_ids': [16, 10751], 'backdrop_path': '/dFYguAfeVt19qAbzJ5mArn7DEJw.jpg', 'adult': False, 'overview': "Nemo, an adventurous young clownfish, is unexpectedly taken from his Great Barrier Reef home to a dentist's office aquarium. It's up to his worrisome father Marlin and a friendly but forgetful fish Dory to bring Nemo home -- meeting vegetarian sharks, surfer dude turtles, hypnotic jellyfish, hungry seagulls, and more along the way.", 'poster_path': '/xVNSgrsvpcAHPnyKf2phYxyppNZ.jpg', 'popularity': 31.27}], 'person_results': [], 'tv_results': [], 'tv_episode_results': [], 'tv_season_results': []}

Or, simply using a get request you can get the above info and more, seen at https://developers.themoviedb.org/3/movies/get-movie-details (https://developers.themoviedb.org/3/movies/get-movie-details).

It may be worth looking into that package or otherwise requesting from the web API to get a complete dataset from one source.

Anyway, let's begin by joining the two tables we have with budgetary information and see what we get.

In [19]:
```
budget_merge = orig_dfs['tn_movie_budgets'].merge(orig_dfs['bom_movie_gross'],
                                                  how='left',
                                                  left_on='movie',
                                                  right_on='title')
budget_merge.head(20)
```

executed in 30ms, finished 13:40:05 2021-05-01

Out[19]:

| | release_date | movie | production_budget | domestic_gross_x | worldwide_gross | title |
|---|---|---|---|---|---|---|
| **0** | Dec 18, 2009 | Avatar | $425,000,000 | $760,507,625 | $2,776,345,279 | NaN |
| **1** | May 20, 2011 | Pirates of the Caribbean: On Stranger Tides | $410,600,000 | $241,063,875 | $1,045,663,875 | Pirates of the Caribbean: On Stranger Tides |
| **2** | Jun 7, 2019 | Dark Phoenix | $350,000,000 | $42,762,350 | $149,762,350 | NaN |
| **3** | May 1, 2015 | Avengers: Age of Ultron | $330,600,000 | $459,005,868 | $1,403,013,963 | Avengers: Age of Ultron |
| **4** | Dec 15, 2017 | Star Wars Ep. VIII: The Last Jedi | $317,000,000 | $620,181,382 | $1,316,721,747 | NaN |
| **5** | Dec 18, 2015 | Star Wars Ep. VII: The Force Awakens | $306,000,000 | $936,662,225 | $2,053,311,220 | NaN |
| **6** | Apr 27, 2018 | Avengers: Infinity War | $300,000,000 | $678,815,482 | $2,048,134,200 | Avengers: Infinity War |
| **7** | May 24, 2007 | Pirates of the Caribbean: At Worldâ□□s End | $300,000,000 | $309,420,425 | $963,420,425 | NaN |
| **8** | Nov 17, 2017 | Justice League | $300,000,000 | $229,024,295 | $655,945,209 | Justice League |
| **9** | Nov 6, 2015 | Spectre | $300,000,000 | $200,074,175 | $879,620,923 | Spectre |
| **10** | Jul 20, 2012 | The Dark Knight Rises | $275,000,000 | $448,139,099 | $1,084,439,099 | The Dark Knight Rises |
| **11** | May 25, 2018 | Solo: A Star Wars Story | $275,000,000 | $213,767,512 | $393,151,347 | Solo: A Star Wars Story |
| **12** | Jul 2, 2013 | The Lone Ranger | $275,000,000 | $89,302,115 | $260,002,115 | The Lone Ranger |
| **13** | Mar 9, 2012 | John Carter | $275,000,000 | $73,058,679 | $282,778,100 | John Carter |
| **14** | Nov 24, 2010 | Tangled | $260,000,000 | $200,821,936 | $586,477,240 | Tangled |
| **15** | May 4, 2007 | Spider-Man 3 | $258,000,000 | $336,530,303 | $894,860,230 | NaN |

| | release_date | movie | production_budget | domestic_gross_x | worldwide_gross | title |
|---|---|---|---|---|---|---|
| **16** | May 6, 2016 | Captain America: Civil War | $250,000,000 | $408,084,349 | $1,140,069,413 | Captain America: Civil War |
| **17** | Mar 25, 2016 | Batman v Superman: Dawn of Justice | $250,000,000 | $330,360,194 | $867,500,281 | Batman v Superman: Dawn of Justice |
| **18** | Dec 14, 2012 | The Hobbit: An Unexpected Journey | $250,000,000 | $303,003,568 | $1,017,003,568 | The Hobbit: An Unexpected Journey |
| **19** | Jul 15, 2009 | Harry Potter and the Half-Blood Prince | $250,000,000 | $302,089,278 | $935,213,767 | NaN |

In [20]:
```
budget_merge.shape
```
executed in 15ms, finished 13:40:05 2021-05-01

Out[20]:
```
(5782, 10)
```

To be honest, after looking at this stuff, it is very tempting to take the tmdb movies dataframe and use those IDs or otherwise generate a completely new and much larger dataset from TMDB. It can result in having budget information for way more movies. However, I notice that their budget estimates are quit low compared to The Numbers estimates, and seeing as both are really guesses and are user maintained / reported it is hard to know which to be more confident about. So long as we are consistent, it should be fine. I think, as opposed to replacing the entire dataset, we can use their API to get a few columns of additional information for the tmdb_movies dataframe, i.e. their imdb ID number (eg. tt######) for connecting reliably with other tables, maybe the production studio, and the listed budget and revenue for each film, since that will solve the problem of having to look at an intersection of the information collected from Box Office Mojo and The Numbers.

We can do this very easily using the tmdbsimple python module I mentioned above. First, we had to create an account and request an API key for TMDB. For safekeeping, here is my API key for TMDB: 0c49f68468be72ef8df12e077b3ab923

Below is an example of how to get the budget for a movie (Harry Potter: DH1) we have in our tmdb dataframe:

In [21]:
```
tmdb.API_KEY = '0c49f68468be72ef8df12e077b3ab923'
test_movie = tmdb.Movies(12444)
response = test_movie.info()
```
executed in 62ms, finished 13:40:05 2021-05-01

In [22]:
```
print(test_movie.runtime)
print(test_movie.budget)
print(test_movie.production_companies[0]['name'])
print(test_movie.revenue)
print(test_movie.vote_average)
print(test_movie.vote_count)
print(test_movie.imdb_id)
```

executed in 15ms, finished 13:40:06 2021-05-01

```
146
250000000
Warner Bros. Pictures
954305868
7.8
14487
tt0926084
```

Let's use these values to verify equivalency between this information and some in the other dataframes we already have.

In [23]:
```
orig_dfs['title_basics'].loc[orig_dfs['title_basics']['tconst'] == 'tt0926084']
```

executed in 14ms, finished 13:40:06 2021-05-01

Out[23]:

| | tconst | primary_title | original_title | start_year | runtime_minutes | genres |
|---|---|---|---|---|---|---|
| **457** | tt0926084 | Harry Potter and the Deathly Hallows: Part 1 | Harry Potter and the Deathly Hallows: Part 1 | 2010 | 146.0 | Adventure,Fantasy,Mystery |

In [24]:
```
orig_dfs['title_ratings'].loc[orig_dfs['title_ratings']['tconst'] == 'tt0926084']
```

executed in 12ms, finished 13:40:06 2021-05-01

Out[24]:

| | tconst | averagerating | numvotes |
|---|---|---|---|
| **65053** | tt0926084 | 7.7 | 425530 |

In [25]:
```
orig_dfs['tn_movie_budgets'].loc[orig_dfs['tn_movie_budgets']['movie'] == 'Harry
```

executed in 15ms, finished 13:40:07 2021-05-01

Out[25]:

| | release_date | movie | production_budget | domestic_gross | worldwide_gross |
|---|---|---|---|---|---|

In [26]:
```
orig_dfs['bom_movie_gross'].loc[orig_dfs['bom_movie_gross']['title'] == 'Harry Po
```

executed in 15ms, finished 13:40:07 2021-05-01

Out[26]:

| | title | studio | domestic_gross | foreign_gross | year |
|---|---|---|---|---|---|
| **2** | Harry Potter and the Deathly Hallows Part 1 | WB | 296000000.0 | 664300000 | 2010 |

Okay, we are seeing conformance in some and not in others, by looking at these records and also

manually comparing some values on the actual websites. Notice that grabbing the imdb id allows us to easily connect to other tables as it is indeed the same as tconst, as expected. Likewise, while not exactly equal, TMDB's revenue info seems to be the same as worldwide_gross in tn_movie_budgets (and the same as domestic plus foreign gross in bom_movie_gross, were the record to exist). The production studio appears to be the same, however TMDB will return a list of companies that the name needs to be grabbed from (it may work to just grab the first company in the list if there is more than one as in the example above). Finally, we previously recognize that estimated budgets from TMDB are lower, sometimes much lower than the estimates from The Numbers. However, it is probably better to consistently use values from TMDB than to fill in missing values when joining the other two tables with financial information and limiting the size of available data significantly. On the other hand, we note that, compared to the genres from IMDB, the ones from TMDB aren't necessarily the same, though they may include common values. It may be worth it to compare the two columns and keep like genres as it is most likely those are the primary genre categories and are common between the two sets. Next, we can recognize that each site (IMDB vs TMDB) have separate rating systems, with incongruencies in the number of votes and the average rating, though in this case the ratings are very close. It may work to choose the rating with the most votes at some point.

Well, now that we can easily access all this information, lets write a function to aaply to our tmdb_movies dataframe and add columns with all the desirable information.

In [27]:
```
orig_dfs['tmdb_movies'].head()
```

executed in 14ms, finished 13:40:07 2021-05-01

Out[27]:

| | genre_ids | id | original_language | original_title | popularity | release_date | title | vote_avera |
|---|---|---|---|---|---|---|---|---|
| **0** | [12, 14, 10751] | 12444 | en | Harry Potter and the Deathly Hallows: Part 1 | 33.533 | 2010-11-19 | Harry Potter and the Deathly Hallows: Part 1 | 7 |
| **1** | [14, 12, 16, 10751] | 10191 | en | How to Train Your Dragon | 28.734 | 2010-03-26 | How to Train Your Dragon | 7 |
| **2** | [12, 28, 878] | 10138 | en | Iron Man 2 | 28.515 | 2010-05-07 | Iron Man 2 | 6 |
| **3** | [16, 35, 10751] | 862 | en | Toy Story | 28.005 | 1995-11-22 | Toy Story | 7 |
| **4** | [28, 878, 12] | 27205 | en | Inception | 27.920 | 2010-07-16 | Inception | 8 |

In [28]:
```
def grab_data_tmdb(row):
    movie_id = row['id']
    row_list = []
    try:
        movie_info = tmdb.Movies(movie_id).info()
    except:
        print('Missing or changed Movie ID')
        row_list = [np.nan] * 6
    else:
        row_list.extend( [ movie_info['runtime'],
                           movie_info['budget'],
                           movie_info['revenue'],
                           movie_info['imdb_id'] ] )
        if movie_info['title'] != row['original_title'] and movie_info['title'] !
            print('Non-matching Movie Titles')
            row_list.append(1)
        else:
            row_list.append(0)
        production_raw = movie_info['production_companies']
        company_names = []
        for company in production_raw:
            company_names.append(company['name'])
        row_list.append(company_names)
    finally:
        return row.append(pd.Series(row_list, index = ['runtime','budget_est','re
```

executed in 14ms, finished 13:40:07 2021-05-01

In [29]:
```
# expanded_tmdb = orig_dfs['tmdb_movies'].apply(grab_data_tmdb,axis = 1)
```

executed in 15ms, finished 13:40:08 2021-05-01

In [30]:
```
# expanded_tmdb.head()
```

executed in 14ms, finished 13:40:08 2021-05-01

Nice! Seems to have worked. Above, there was some output indicating missing or potentially changed movie IDs, so we will have to look into that. We will also have to see how many films didn't have data for the columns we hoped to retrieve. However, since it took an hour to collect the extra data via API requests, lets save the expanded dataframe as a new csv for future retrieval.

In [31]:
```
# expanded_tmdb.to_csv('./dataFiles/expanded_tmdb.csv',index=False)
```

executed in 13ms, finished 13:40:08 2021-05-01

In [32]:
```
expanded_tmdb = pd.read_csv('./dataFiles/expanded_tmdb.csv')
```

executed in 61ms, finished 13:40:09 2021-05-01

In [33]:
```
expanded_tmdb.head()
```
executed in 14ms, finished 13:40:09 2021-05-01

Out[33]:

| | genre_ids | id | original_language | original_title | popularity | release_date | title | vote_avera |
|---|---|---|---|---|---|---|---|---|
| **0** | [12, 14, 10751] | 12444 | en | Harry Potter and the Deathly Hallows: Part 1 | 33.533 | 2010-11-19 | Harry Potter and the Deathly Hallows: Part 1 | 7 |
| **1** | [14, 12, 16, 10751] | 10191 | en | How to Train Your Dragon | 28.734 | 2010-03-26 | How to Train Your Dragon | 7 |
| **2** | [12, 28, 878] | 10138 | en | Iron Man 2 | 28.515 | 2010-05-07 | Iron Man 2 | 6 |
| **3** | [16, 35, 10751] | 862 | en | Toy Story | 28.005 | 1995-11-22 | Toy Story | 7 |
| **4** | [28, 878, 12] | 27205 | en | Inception | 27.920 | 2010-07-16 | Inception | 8 |

Now, let's see if we actually got any extra data. Since our budgetary information is most important, let's see how much of the original dataframe we were able to find this extra information for:

In [34]:
```
cleaned_tmdb = expanded_tmdb.loc[(expanded_tmdb['budget_est'] != 0) &
                                 (expanded_tmdb['revenue'] != 0) &
                                 (expanded_tmdb['req_title'] != 1)]
```
executed in 12ms, finished 13:40:09 2021-05-01

In [35]:
```
cleaned_tmdb = cleaned_tmdb.dropna(axis = 0)
cleaned_tmdb = cleaned_tmdb.reset_index()
cleaned_tmdb.shape
```
executed in 10ms, finished 13:40:10 2021-05-01

Out[35]:
```
(2348, 16)
```

Okay.. not great. So, out of roughly 27000 movies, we were able to collect budget and revenue data for just under 10% of the records. Note: we also required that the title from the request response was the same as either the original_title or title column value from the original dataframe. This is to account for movie IDs that have been changed in the database since the original data file was retrieved, which is stated as possible in their API documentation. If we wanted to do extra work to check responses where the title did not match either, i.e. req_title = 1 in the expanded dataframe, we could potentially have an additional 450 records with financial information. At this time, it does not seem worth expending the effort, as anything up to date probably contains the

most popular and highest grossing films as is. So, in a roundabout way, all we have done is to reduce the tmdb_movies dataframe to a smaller size but with extra useful information, including (hopefully, if TMDB has properly sourced and updated this information) the ability to easily make connections to the imdb tables using tconst. Let's clean this table a little bit further below.

In [36]:
```
cleaned_tmdb = cleaned_tmdb.drop(columns = ['index','req_title'])
```
executed in 14ms, finished 13:40:10 2021-05-01

In [37]:
```
cleaned_tmdb = cleaned_tmdb.rename(columns = {'id': 'tmdb_id'})
```
executed in 14ms, finished 13:40:10 2021-05-01

Since our dream of a big dataset all from TMDB more or less failed without going further, we will get back to combining the tables we have with that information. As a reminder, lets .head() them here for quick reference.

In [38]:
```
cleaned_tmdb.head()
```
executed in 16ms, finished 13:40:11 2021-05-01

Out[38]:

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

In [39]:
```
orig_dfs['bom_movie_gross'].head()
```
executed in 15ms, finished 13:40:11 2021-05-01

Out[39]:

| | 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 |

In [40]:
```
orig_dfs['tn_movie_budgets'].head()
```
executed in 14ms, finished 13:40:11 2021-05-01

Out[40]:

| | 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 |
| **3** | May 1, 2015 | Avengers: Age of Ultron | $330,600,000 | $459,005,868 | $1,403,013,963 |
| **4** | Dec 15, 2017 | Star Wars Ep. VIII: The Last Jedi | $317,000,000 | $620,181,382 | $1,316,721,747 |

Lets double check the .info and some .value_counts() for each, to see if they're all ready for merging.

In [41]:
```
cleaned_tmdb.info()
```
executed in 14ms, finished 13:40:12 2021-05-01

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2348 entries, 0 to 2347
Data columns (total 14 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   genre_ids             2348 non-null   object 
 1   tmdb_id               2348 non-null   int64  
 2   original_language     2348 non-null   object 
 3   original_title        2348 non-null   object 
 4   popularity            2348 non-null   float64
 5   release_date          2348 non-null   object 
 6   title                 2348 non-null   object 
 7   vote_average          2348 non-null   float64
 8   vote_count            2348 non-null   int64  
 9   runtime               2348 non-null   float64
 10  budget_est            2348 non-null   float64
 11  revenue               2348 non-null   float64
 12  imdb_id               2348 non-null   object 
 13  production_companies  2348 non-null   object 
dtypes: float64(5), int64(2), object(7)
memory usage: 256.9+ KB
```

In [42]:
```
orig_dfs['bom_movie_gross'].info()
```
executed in 14ms, finished 13:40:12 2021-05-01

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

In [43]:
```
orig_dfs['bom_movie_gross'].foreign_gross.value_counts()
```
executed in 14ms, finished 13:40:12 2021-05-01

Out[43]:
```
1200000      23
1100000      14
4200000      12
1900000      12
2500000      11
             ..
46200000      1
128000000     1
93700000      1
29000000      1
63300000      1
Name: foreign_gross, Length: 1204, dtype: int64
```

In [44]:
```
orig_dfs['tn_movie_budgets'].info()
```
executed in 14ms, finished 13:40:13 2021-05-01

```
<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   object
 3   domestic_gross     5782 non-null   object
 4   worldwide_gross    5782 non-null   object
dtypes: object(5)
memory usage: 226.0+ KB
```

In [45]:
```
orig_dfs['tn_movie_budgets'].production_budget.value_counts()
```
executed in 14ms, finished 13:40:13 2021-05-01

Out[45]:
```
$20,000,000     231
$10,000,000     212
$30,000,000     177
$15,000,000     173
$25,000,000     171
                ...
$6,600,000        1
$640,000          1
$270,000          1
$217,000,000      1
$730,000          1
Name: production_budget, Length: 509, dtype: int64
```

In [46]:
```
orig_dfs['tn_movie_budgets'].worldwide_gross.value_counts()
```
executed in 15ms, finished 13:40:13 2021-05-01

Out[46]:
```
$0              367
$8,000,000        9
$7,000,000        6
$2,000,000        6
$11,000,000       4
                 ...
$67,255,916       1
$10,447,579       1
$205,298,907      1
$13,205,411       1
$211,562,435      1
Name: worldwide_gross, Length: 5356, dtype: int64
```

Okay, so it looks like in the BOM data there are a lot of NULLs in foreign_gross, whereas for the TN data there are a lot of zero values, which could be either unreleased movies or movies for which this data hasn't been collected / reported. Additionally, all of the TN columns are object type, so we need to change the type of those columns to int64. Since we need at least this information for both, lets convert data types then get rid of any rows with NaNs or zeros in each dataframe before merging.

In [47]:
```
cleaned_bom = orig_dfs['bom_movie_gross'].dropna(axis = 0)
cleaned_bom = cleaned_bom.reset_index()
cleaned_bom = cleaned_bom.drop(columns = 'index')
cleaned_bom.info()
```
executed in 15ms, finished 13:40:13 2021-05-01

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2007 entries, 0 to 2006
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   title           2007 non-null   object
 1   studio          2007 non-null   object
 2   domestic_gross  2007 non-null   float64
 3   foreign_gross   2007 non-null   object
 4   year            2007 non-null   int64 
dtypes: float64(1), int64(1), object(3)
memory usage: 78.5+ KB
```

In [48]:
```
cleaned_bom.foreign_gross.value_counts()
```
executed in 12ms, finished 13:40:14 2021-05-01

Out[48]:
```
1200000      23
1100000      13
4200000      12
2500000      11
1900000      11
             ..
76000000      1
161000        1
45400000      1
133100000     1
63300000      1
Name: foreign_gross, Length: 1193, dtype: int64
```

In [49]:
```
cleaned_tn = orig_dfs['tn_movie_budgets'].copy()
cleaned_tn['production_budget'] = cleaned_tn['production_budget'].apply(lambda x:
cleaned_tn['domestic_gross'] = cleaned_tn['domestic_gross'].apply(lambda x: int(x
cleaned_tn['worldwide_gross'] = cleaned_tn['worldwide_gross'].apply(lambda x: int
cleaned_tn.head()
```
executed in 31ms, finished 13:40:14 2021-05-01

Out[49]:

| | 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 |
| **3** | May 1, 2015 | Avengers: Age of Ultron | 330600000 | 459005868 | 1403013963 |
| **4** | Dec 15, 2017 | Star Wars Ep. VIII: The Last Jedi | 317000000 | 620181382 | 1316721747 |

In [50]:
```
cleaned_tn = cleaned_tn.loc[(cleaned_tn['production_budget']!= 0) &
                            (cleaned_tn['domestic_gross'] !=0) &
                            (cleaned_tn['worldwide_gross'] !=0) ]
```
executed in 14ms, finished 13:40:15 2021-05-01

In [51]:
```
cleaned_tn.production_budget.value_counts()
```
executed in 10ms, finished 13:40:15 2021-05-01

Out[51]:
```
20000000     221
10000000     197
30000000     171
40000000     163
15000000     162
             ...
420000         1
7303082        1
306000000      1
4638783        1
28500000       1
Name: production_budget, Length: 472, dtype: int64
```

In [52]:
```
cleaned_tn.worldwide_gross.value_counts()
```
executed in 14ms, finished 13:40:16 2021-05-01

Out[52]:
```
8000000      9
2000000      6
7000000      6
11000000     4
9000000      4
             ..
478595       1
58545540     1
92618117     1
3902679      1
104267443    1
Name: worldwide_gross, Length: 5177, dtype: int64
```

In [53]:
```
cleaned_tn = cleaned_tn.reset_index()
cleaned_tn = cleaned_tn.drop(columns = 'index')
cleaned_tn.head()
```
executed in 15ms, finished 13:40:16 2021-05-01

Out[53]:

| | 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 |
| **3** | May 1, 2015 | Avengers: Age of Ultron | 330600000 | 459005868 | 1403013963 |
| **4** | Dec 15, 2017 | Star Wars Ep. VIII: The Last Jedi | 317000000 | 620181382 | 1316721747 |

In [54]:
```
print(cleaned_tmdb.tmdb_id.nunique())
print(cleaned_tn.movie.nunique())
print(cleaned_tmdb.shape)
print(cleaned_tn.shape)
```

executed in 26ms, finished 13:40:16 2021-05-01

```
2143
5159
(2348, 14)
(5234, 5)
```

Hmm... it looks like there may be some duplicate movie records to take care of. A couple hundred in each, let's remove them.

In [55]:
```
cleaned_tmdb = cleaned_tmdb.drop_duplicates(subset = ['tmdb_id'])
cleaned_tn = cleaned_tn.drop_duplicates(subset = ['movie','release_date'])
```

executed in 14ms, finished 13:40:17 2021-05-01

In [56]:
```
print(cleaned_tmdb.tmdb_id.nunique())
print(cleaned_tn.movie.nunique())
print(cleaned_tmdb.shape)
print(cleaned_tn.shape)
```

executed in 13ms, finished 13:40:17 2021-05-01

```
2143
5159
(2143, 14)
(5234, 5)
```

I also am noticing that there may be some duplicate movie titles with different release dates that we want to keep as separate records. The easiest way to handle this will be to separate out the release date into three columns each, release day, month, and year, then finally merge on both the movie title and the release year.

In [57]:
```
def create_date_cols(row):
    timestamp_obj = pd.to_datetime(row['release_date'])
    day = timestamp_obj.day
    month = timestamp_obj.month
    year = timestamp_obj.year
    day_of_week = timestamp_obj.dayofweek
    return row.append(pd.Series([day_of_week, day, month, year],
                                index=['release_week_day', 'release_day', 'releas
```

executed in 13ms, finished 13:40:18 2021-05-01

In [58]:
```
cleaned_tmdb = cleaned_tmdb.apply(create_date_cols,axis=1)
cleaned_tn = cleaned_tn.apply(create_date_cols,axis=1)
```

executed in 4.27s, finished 13:40:22 2021-05-01

In [59]:
```
cleaned_tn.head()
```
executed in 14ms, finished 13:40:23 2021-05-01

Out[59]:

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

In [60]:
```
cleaned_tmdb = cleaned_tmdb.drop(columns = 'release_date')
cleaned_tn = cleaned_tn.drop(columns = 'release_date')
```
executed in 15ms, finished 13:40:23 2021-05-01

I'm thinking now that I would like to see what other information I can add to the TN dataset before merging. For example, we can use title_basics to add a tconst (better matching), the runtime, and genres. Then, since that table will have tconst, we can add information from the title_ratings dataframe. So, cleaned_tn <- title_basics <- title_ratings. Let's check them out pre-merging:

In [61]:
```
orig_dfs['title_basics'].head()
```
executed in 14ms, finished 13:40:23 2021-05-01

Out[61]:

| | 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 |
| 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 [62]:
```
orig_dfs['title_basics'].info()
```
executed in 25ms, finished 13:40:24 2021-05-01

```
<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 [63]:
```
orig_dfs['title_basics'].tconst.nunique()
```
executed in 30ms, finished 13:40:24 2021-05-01

Out[63]: 146144

Okay, so there are some missing values here and there, but the ones we will want to join on, tconst, titles and start_year are basically all there, and there are no duplicate rows based on tconst.

In [64]:
```
orig_dfs['title_ratings'].head()
```
executed in 15ms, finished 13:40:25 2021-05-01

Out[64]:

| | tconst | 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 [65]:
```
orig_dfs['title_ratings'].info()
```
executed in 15ms, finished 13:40:25 2021-05-01

```
<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 [66]:
```
orig_dfs['title_ratings'].tconst.nunique()
```
executed in 15ms, finished 13:40:26 2021-05-01

Out[66]: 73856

The same seems to be true for title_ratings as well, though we have ratings information for many less movies than we have title basics for. Anyway, lets quickly merge the two title dataframes:

In [67]:
```
titles_plus_ratings = orig_dfs['title_basics'].merge(orig_dfs['title_ratings'],
                                                     how='left',
                                                     on='tconst')
titles_plus_ratings.head(20)
```

executed in 92ms, finished 13:40:26 2021-05-01

Out[67]:

| | 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 |
| 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 |
| 5 | tt0111414 | A Thin Life | A Thin Life | 2018 | 75.0 | Comedy |
| 6 | tt0112502 | Bigfoot | Bigfoot | 2017 | NaN | Horror,Thriller |
| 7 | tt0137204 | Joe Finds Grace | Joe Finds Grace | 2017 | 83.0 | Adventure,Animation,Comedy |
| 8 | tt0139613 | O Silêncio | O Silêncio | 2012 | NaN | Documentary,History |
| 9 | tt0144449 | Nema aviona za Zagreb | Nema aviona za Zagreb | 2012 | 82.0 | Biography |
| 10 | tt0146592 | Pál Adrienn | Pál Adrienn | 2010 | 136.0 | Drama |
| 11 | tt0154039 | So Much for Justice! | Oda az igazság | 2010 | 100.0 | History |
| 12 | tt0159369 | Cooper and Hemingway: The True Gen | Cooper and Hemingway: The True Gen | 2013 | 180.0 | Documentary |
| 13 | tt0162942 | Children of the Green Dragon | A zöld sárkány gyermekei | 2010 | 89.0 | Drama |
| 14 | tt0170651 | T.G.M. - osvoboditel | T.G.M. - osvoboditel | 2018 | 60.0 | Documentary |
| 15 | tt0176694 | The Tragedy of Man | Az ember tragédiája | 2011 | 160.0 | Animation,Drama,History |
| 16 | tt0187902 | How Huang Fei-hong Rescued the Orphan from the... | How Huang Fei-hong Rescued the Orphan from the... | 2011 | NaN | NaN |
| 17 | tt0192528 | Heaven & Hell | Reverse Heaven | 2018 | 104.0 | Drama |
| 18 | tt0230212 | The Final Journey | The Final Journey | 2010 | 120.0 | Drama |

| | tconst | primary_title | original_title | start_year | runtime_minutes | genres |
|---|---|---|---|---|---|---|
| **19** | tt0247643 | Los pájaros se van con la muerte | Los pájaros se van con la muerte | 2011 | 110.0 | Drama,Mystery |

In [68]:
```
titles_plus_ratings.shape[0] - titles_plus_ratings.dropna().shape[0]
```
executed in 45ms, finished 13:40:26 2021-05-01

Out[68]: 80424

In [69]:
```
titles_plus_ratings.tconst.nunique()
```
executed in 30ms, finished 13:40:27 2021-05-01

Out[69]: 146144

So, there are a majority of rows without complete information, but when we merge with cleaned_tn it may not matter.

In [70]:
```
cleaned_tn.shape
```
executed in 14ms, finished 13:40:27 2021-05-01

Out[70]: (5234, 8)

Since there are multiple title type columns to match on, we can avoid duplicates by creating two dataframes, one where the original_title and primary_title are the same and one where they are different, and then join cleaned_tn to those before concatenating all the results.

In [71]:
```
diff_titles = titles_plus_ratings.loc[titles_plus_ratings.primary_title != titles
same_titles = titles_plus_ratings.loc[titles_plus_ratings.primary_title == titles
```
executed in 43ms, finished 13:40:28 2021-05-01

In [72]:
```
diff_tn1 = cleaned_tn.merge(diff_titles,
                            how='inner',
                            left_on=['movie', 'release_year'],
                            right_on=['primary_title', 'start_year'])
diff_tn2 = cleaned_tn.merge(diff_titles,
                            how='inner',
                            left_on=['movie', 'release_year'],
                            right_on=['original_title', 'start_year'])
same_tn1 = cleaned_tn.merge(same_titles,
                            how='inner',
                            left_on=['movie', 'release_year'],
                            right_on=['original_title', 'start_year'])
```
executed in 59ms, finished 13:40:28 2021-05-01

In [73]:
```
diff_tn1.shape
```
executed in 8ms, finished 13:40:42 2021-05-01

Out[73]: (32, 16)

In [74]:
```
diff_tn2.shape
```
executed in 14ms, finished 13:40:43 2021-05-01

Out[74]: (18, 16)

In [75]:
```
same_tn1.shape
```
executed in 14ms, finished 13:40:44 2021-05-01

Out[75]: (1338, 16)

In [76]:
```
buffed_tn = pd.concat([diff_tn1,diff_tn2,same_tn1])
```
executed in 14ms, finished 13:40:44 2021-05-01

In [77]:
```
buffed_tn.head()
```
executed in 14ms, finished 13:40:45 2021-05-01

Out[77]:

| | movie | production_budget | domestic_gross | worldwide_gross | release_week_day | release_day | r |
|---|---|---|---|---|---|---|---|
| **0** | Rogue One: A Star Wars Story | 200000000 | 532177324 | 1049102856 | 4 | 16 | |
| **1** | Iron Man 3 | 200000000 | 408992272 | 1215392272 | 4 | 3 | |
| **2** | Furious 7 | 190000000 | 353007020 | 1518722794 | 4 | 3 | |
| **3** | X-Men: First Class | 160000000 | 146408305 | 355408305 | 4 | 3 | |
| **4** | Home | 130000000 | 177397510 | 385997896 | 4 | 27 | |

In [78]:
```
buffed_tn.shape
```
executed in 14ms, finished 13:40:45 2021-05-01

Out[78]: (1388, 16)

In [79]:
```
buffed_tn.info()
```

executed in 14ms, finished 13:40:46 2021-05-01

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1388 entries, 0 to 1337
Data columns (total 16 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   movie              1388 non-null   object 
 1   production_budget  1388 non-null   int64  
 2   domestic_gross     1388 non-null   int64  
 3   worldwide_gross    1388 non-null   int64  
 4   release_week_day   1388 non-null   int64  
 5   release_day        1388 non-null   int64  
 6   release_month      1388 non-null   int64  
 7   release_year       1388 non-null   int64  
 8   tconst             1388 non-null   object 
 9   primary_title      1388 non-null   object 
 10  original_title     1388 non-null   object 
 11  start_year         1388 non-null   int64  
 12  runtime_minutes    1375 non-null   float64
 13  genres             1385 non-null   object 
 14  averagerating      1360 non-null   float64
 15  numvotes           1360 non-null   float64
dtypes: float64(3), int64(8), object(5)
memory usage: 184.3+ KB
```

Okay, we had a few rows sneak in from titles_plus_ratings where we had nulls for some of the values, so let's ditch those few rows.

In [80]:
```
buffed_tn = buffed_tn.dropna(axis = 0)
buffed_tn = buffed_tn.reset_index()
buffed_tn = buffed_tn.drop(columns = 'index')
buffed_tn.info()
```

executed in 14ms, finished 13:40:46 2021-05-01

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1355 entries, 0 to 1354
Data columns (total 16 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   movie              1355 non-null   object 
 1   production_budget  1355 non-null   int64  
 2   domestic_gross     1355 non-null   int64  
 3   worldwide_gross    1355 non-null   int64  
 4   release_week_day   1355 non-null   int64  
 5   release_day        1355 non-null   int64  
 6   release_month      1355 non-null   int64  
 7   release_year       1355 non-null   int64  
 8   tconst             1355 non-null   object 
 9   primary_title      1355 non-null   object 
 10  original_title     1355 non-null   object 
 11  start_year         1355 non-null   int64  
 12  runtime_minutes    1355 non-null   float64
 13  genres             1355 non-null   object 
 14  averagerating      1355 non-null   float64
 15  numvotes           1355 non-null   float64
dtypes: float64(3), int64(8), object(5)
memory usage: 169.5+ KB
```

In [81]:
```
buffed_tn.head()
```

executed in 14ms, finished 13:40:47 2021-05-01

Out[81]:

| | movie | production_budget | domestic_gross | worldwide_gross | release_week_day | release_day |
|---|---|---|---|---|---|---|
| **0** | Rogue One: A Star Wars Story | 200000000 | 532177324 | 1049102856 | 4 | 16 |
| **1** | Iron Man 3 | 200000000 | 408992272 | 1215392272 | 4 | 3 |
| **2** | Furious 7 | 190000000 | 353007020 | 1518722794 | 4 | 3 |
| **3** | X-Men: First Class | 160000000 | 146408305 | 355408305 | 4 | 3 |
| **4** | The Promise | 90000000 | 8224288 | 10551417 | 4 | 21 |

Okay, so now we finally have two dataframes with as much categorical and numerical information for their repective movies as is currently possible, buffed_tn and cleaned_tmdb. We should be able to easily concatenate these and then remove duplicate rows based on tconst. Let's make them

both contain all the same columns in the same order to prepare for concatenating.

In [82]:
```
cleaned_tmdb.head()
```
executed in 14ms, finished 13:40:48 2021-05-01

Out[82]:

| | genre_ids | tmdb_id | original_language | original_title | popularity | title | vote_average | vote_co |
|---|---|---|---|---|---|---|---|---|
| **0** | [12, 14, 10751] | 12444 | en | Harry Potter and the Deathly Hallows: Part 1 | 33.533 | Harry Potter and the Deathly Hallows: Part 1 | 7.7 | 10 |
| **1** | [14, 12, 16, 10751] | 10191 | en | How to Train Your Dragon | 28.734 | How to Train Your Dragon | 7.7 | 7( |
| **2** | [12, 28, 878] | 10138 | en | Iron Man 2 | 28.515 | Iron Man 2 | 6.8 | 12 |
| **3** | [16, 35, 10751] | 862 | en | Toy Story | 28.005 | Toy Story | 7.9 | 10 |
| **4** | [28, 878, 12] | 27205 | en | Inception | 27.920 | Inception | 8.3 | 22 |

In [83]:
```
cleaned_tmdb.columns
```
executed in 15ms, finished 13:40:48 2021-05-01

Out[83]:
```
Index(['genre_ids', 'tmdb_id', 'original_language', 'original_title',
       'popularity', 'title', 'vote_average', 'vote_count', 'runtime',
       'budget_est', 'revenue', 'imdb_id', 'production_companies',
       'release_week_day', 'release_day', 'release_month', 'release_year'],
      dtype='object')
```

In [84]:
```
cleaned_tmdb = cleaned_tmdb.drop(columns = ['tmdb_id','original_language','origir
```
executed in 14ms, finished 13:40:49 2021-05-01

In [85]:
```
cleaned_tmdb.columns
```
executed in 15ms, finished 13:40:49 2021-05-01

Out[85]:
```
Index(['genre_ids', 'title', 'vote_average', 'vote_count', 'runtime',
       'budget_est', 'revenue', 'imdb_id', 'release_week_day', 'release_day',
       'release_month', 'release_year'],
      dtype='object')
```

In [86]:
```
buffed_tn.columns
```
executed in 15ms, finished 13:40:50 2021-05-01

Out[86]:
```
Index(['movie', 'production_budget', 'domestic_gross', 'worldwide_gross',
       'release_week_day', 'release_day', 'release_month', 'release_year',
       'tconst', 'primary_title', 'original_title', 'start_year',
       'runtime_minutes', 'genres', 'averagerating', 'numvotes'],
      dtype='object')
```

In [87]:
```
buffed_tn = buffed_tn.drop(columns = ['primary_title','original_title','domestic_
```
executed in 15ms, finished 13:40:50 2021-05-01

In [88]:
```
buffed_tn.columns
```
executed in 14ms, finished 13:40:51 2021-05-01

Out[88]:
```
Index(['movie', 'production_budget', 'worldwide_gross', 'release_week_day',
       'release_day', 'release_month', 'release_year', 'tconst',
       'runtime_minutes', 'genres', 'averagerating', 'numvotes'],
      dtype='object')
```

In [89]:
```
cleaned_tmdb = cleaned_tmdb.rename(columns={'genre_ids': 'Genres', 'title': 'Titl
                    'runtime': 'runtime_minutes', 'revenue': 'worldwide_gross',
```
executed in 15ms, finished 13:40:51 2021-05-01

In [90]:
```
cleaned_tmdb.columns
```
executed in 15ms, finished 13:40:52 2021-05-01

Out[90]:
```
Index(['Genres', 'Title', 'vote_average', 'vote_count', 'runtime_minutes',
       'budget_est', 'worldwide_gross', 'tconst', 'release_week_day',
       'release_day', 'release_month', 'release_year'],
      dtype='object')
```

In [91]:
```
buffed_tn = buffed_tn.rename(
    columns={'movie': 'Title', 'production_budget': 'budget_est', 'genres': 'Genr
             'averagerating': 'vote_average', 'numvotes': 'vote_count'})
```
executed in 14ms, finished 13:40:53 2021-05-01

In [92]:
```
buffed_tn.columns
```
executed in 14ms, finished 13:40:53 2021-05-01

Out[92]:
```
Index(['Title', 'budget_est', 'worldwide_gross', 'release_week_day',
       'release_day', 'release_month', 'release_year', 'tconst',
       'runtime_minutes', 'Genres', 'vote_average', 'vote_count'],
      dtype='object')
```

In [93]:
```
len(buffed_tn.columns)
```
executed in 14ms, finished 13:40:54 2021-05-01

Out[93]:
```
12
```

In [94]:
```
len(cleaned_tmdb.columns)
```
executed in 15ms, finished 13:40:54 2021-05-01

Out[94]: 12

In [95]:
```
cleaned_tmdb = cleaned_tmdb[['tconst', 'Title', 'Genres', 'runtime_minutes',
                             'vote_average', 'vote_count', 'budget_est', 'worldwi
                             'release_week_day', 'release_day', 'release_month',
```
executed in 14ms, finished 13:40:55 2021-05-01

In [96]:
```
cleaned_tmdb.head()
```
executed in 28ms, finished 13:40:55 2021-05-01

Out[96]:

| | tconst | Title | Genres | runtime_minutes | vote_average | vote_count | budget_est | worldwide |
|---|---|---|---|---|---|---|---|---|
| **0** | tt0926084 | Harry Potter and the Deathly Hallows: Part 1 | [12, 14, 10751] | 146.0 | 7.7 | 10788 | 250000000.0 | 9543 |
| **1** | tt0892769 | How to Train Your Dragon | [14, 12, 16, 10751] | 98.0 | 7.7 | 7610 | 165000000.0 | 4948 |
| **2** | tt1228705 | Iron Man 2 | [12, 28, 878] | 124.0 | 6.8 | 12368 | 200000000.0 | 6239 |
| **3** | tt0114709 | Toy Story | [16, 35, 10751] | 81.0 | 7.9 | 10174 | 30000000.0 | 3735 |
| **4** | tt1375666 | Inception | [28, 878, 12] | 148.0 | 8.3 | 22186 | 160000000.0 | 8255 |

In [97]:
```
buffed_tn = buffed_tn[['tconst', 'Title', 'Genres', 'runtime_minutes',
                       'vote_average', 'vote_count', 'budget_est', 'worldwi
                       'release_week_day', 'release_day', 'release_month',
```
executed in 15ms, finished 13:40:56 2021-05-01

In [98]:
```
buffed_tn.head()
```
executed in 27ms, finished 13:40:56 2021-05-01

Out[98]:

| | tconst | Title | Genres | runtime_minutes | vote_average | vote_count | budget_est |
|---|---|---|---|---|---|---|---|
| **0** | tt3748528 | Rogue One: A Star Wars Story | Action,Adventure,Sci-Fi | 133.0 | 7.8 | 478592.0 | 200000000 |
| **1** | tt1300854 | Iron Man 3 | Action,Adventure,Sci-Fi | 130.0 | 7.2 | 692794.0 | 200000000 |
| **2** | tt2820852 | Furious 7 | Action,Crime,Thriller | 137.0 | 7.2 | 335074.0 | 190000000 |
| **3** | tt1270798 | X-Men: First Class | Action,Adventure,Sci-Fi | 131.0 | 7.7 | 608930.0 | 160000000 |
| **4** | tt7232438 | The Promise | Drama,Horror,Thriller | 114.0 | 6.1 | 629.0 | 90000000 |

There is one last thing to do before we can work with these. We need to fix that in buffed_tn Genres is a comma separated string and in cleaned_tmdb it is a list of integer IDs. For buffed_tn, we need to simply turn the string into a list using a comma as a delimiter. For cleaned_tmdb we have a dictionary that we can convert the integer ID's into string values for, and then leave as a list. Thus a list of genres as strings in both dataframes.

In [99]:
```
buffed_tn['Genres'] = buffed_tn['Genres'].apply( lambda x: x.split(',') )
```
executed in 13ms, finished 13:40:57 2021-05-01

In [100]:
```
buffed_tn.head()
```
executed in 15ms, finished 13:40:58 2021-05-01

Out[100]:

| | tconst | Title | Genres | runtime_minutes | vote_average | vote_count | budget_est | worldwi |
|---|---|---|---|---|---|---|---|---|
| **0** | tt3748528 | Rogue One: A Star Wars Story | [Action, Adventure, Sci-Fi] | 133.0 | 7.8 | 478592.0 | 200000000 | 10 |
| **1** | tt1300854 | Iron Man 3 | [Action, Adventure, Sci-Fi] | 130.0 | 7.2 | 692794.0 | 200000000 | 12 |
| **2** | tt2820852 | Furious 7 | [Action, Crime, Thriller] | 137.0 | 7.2 | 335074.0 | 190000000 | 15 |
| **3** | tt1270798 | X-Men: First Class | [Action, Adventure, Sci-Fi] | 131.0 | 7.7 | 608930.0 | 160000000 | 3 |
| **4** | tt7232438 | The Promise | [Drama, Horror, Thriller] | 114.0 | 6.1 | 629.0 | 90000000 | |

In [101]:
```
buffed_tn.Genres[0]
```
executed in 14ms, finished 13:40:58 2021-05-01

Out[101]:
```
['Action', 'Adventure', 'Sci-Fi']
```

For cleaned_tmdb, it's a bit more challenging because even though it looks like a list, it is really stored as a string, e.g. '[1,2,3]' instead of [1,2,3]. I found this module to handle that, ast, and it will convert the former to the latter.

In [102]:
```
import ast
```
executed in 11ms, finished 13:40:59 2021-05-01

In [103]:
```
ast.literal_eval(cleaned_tmdb.Genres[0])
```
executed in 14ms, finished 13:40:59 2021-05-01

Out[103]:
```
[12, 14, 10751]
```

In [104]:
```
genre_dict = {28: 'Action',
              12: 'Adventure',
              16: 'Animation',
              35: 'Comedy',
              80: 'Crime',
              99: 'Documentary',
              18: 'Drama',
              10751: 'Family',
              14: 'Fantasy',
              36: 'History',
              27: 'Horror',
              10402: 'Music',
              9648: 'Mystery',
              10749: 'Romance',
              878: 'Science Fiction',
              10770: 'TV Movie',
              53: 'Thriller',
              10752: 'War',
              37: 'Western'}
cleaned_tmdb['Genres'] = cleaned_tmdb['Genres'].apply(
    lambda x: [genre_dict[genre_id] for genre_id in ast.literal_eval(x)])
```

executed in 44ms, finished 13:41:00 2021-05-01

In [105]:
```
cleaned_tmdb.Genres[0]
```

executed in 15ms, finished 13:41:01 2021-05-01

Out[105]:
```
['Adventure', 'Fantasy', 'Family']
```

In [106]:
```
cleaned_tmdb.head()
```
executed in 15ms, finished 13:41:01 2021-05-01

Out[106]:

| | tconst | Title | Genres | runtime_minutes | vote_average | vote_count | budget_est | worldv |
|---|---|---|---|---|---|---|---|---|
| **0** | tt0926084 | Harry Potter and the Deathly Hallows: Part 1 | [Adventure, Fantasy, Family] | 146.0 | 7.7 | 10788 | 250000000.0 | 9 |
| **1** | tt0892769 | How to Train Your Dragon | [Fantasy, Adventure, Animation, Family] | 98.0 | 7.7 | 7610 | 165000000.0 | 4 |
| **2** | tt1228705 | Iron Man 2 | [Adventure, Action, Science Fiction] | 124.0 | 6.8 | 12368 | 200000000.0 | 6 |
| **3** | tt0114709 | Toy Story | [Animation, Comedy, Family] | 81.0 | 7.9 | 10174 | 30000000.0 | 3 |
| **4** | tt1375666 | Inception | [Action, Science Fiction, Adventure] | 148.0 | 8.3 | 22186 | 160000000.0 | 8 |

Finally, lets concatenate these two dataframes. It is likely there are some duplicate movies, so we will have to weed them out. For the duplicates, however, instead of just taking the first and dropping the second, since they are from different sources and we are using a combined dataset, I would like to average things like the budget est and worldwide gross, and do a weighted reaveraging of the vote_average and counts. So, let's merge, find the subsection of duplicates and deal with them using a second merge, and then find any entries not in the intersection from each original dataframe and include those records as well.

In [107]:
```
movie_collection_w_dups = pd.concat([cleaned_tmdb,buffed_tn])
```
executed in 12ms, finished 13:41:02 2021-05-01

In [108]:
```
movie_collection_w_dups.head()
```
executed in 27ms, finished 13:41:02 2021-05-01

Out[108]:

| | tconst | Title | Genres | runtime_minutes | vote_average | vote_count | budget_est | worldv |
|---|---|---|---|---|---|---|---|---|
| **0** | tt0926084 | Harry Potter and the Deathly Hallows: Part 1 | [Adventure, Fantasy, Family] | 146.0 | 7.7 | 10788.0 | 250000000.0 | 9 |
| **1** | tt0892769 | How to Train Your Dragon | [Fantasy, Adventure, Animation, Family] | 98.0 | 7.7 | 7610.0 | 165000000.0 | 4 |
| **2** | tt1228705 | Iron Man 2 | [Adventure, Action, Science Fiction] | 124.0 | 6.8 | 12368.0 | 200000000.0 | 6 |
| **3** | tt0114709 | Toy Story | [Animation, Comedy, Family] | 81.0 | 7.9 | 10174.0 | 30000000.0 | 3 |
| **4** | tt1375666 | Inception | [Action, Science Fiction, Adventure] | 148.0 | 8.3 | 22186.0 | 160000000.0 | 8 |

In [109]:
```
movie_collection_w_dups.info()
```
executed in 15ms, finished 13:41:03 2021-05-01

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3498 entries, 0 to 1354
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   tconst            3498 non-null   object 
 1   Title             3498 non-null   object 
 2   Genres            3498 non-null   object 
 3   runtime_minutes   3498 non-null   float64
 4   vote_average      3498 non-null   float64
 5   vote_count        3498 non-null   float64
 6   budget_est        3498 non-null   float64
 7   worldwide_gross   3498 non-null   float64
 8   release_week_day  3498 non-null   int64  
 9   release_day       3498 non-null   int64  
 10  release_month     3498 non-null   int64  
 11  release_year      3498 non-null   int64  
dtypes: float64(5), int64(4), object(3)
memory usage: 355.3+ KB
```

In [110]:
```
movie_collection_w_dups.tconst.nunique()
```
executed in 14ms, finished 13:41:03 2021-05-01

Out[110]:
```
2373
```

So, out of ~3500 records there are only 2373 unique movies.

In [111]:
```
inner_sect = cleaned_tmdb.merge(buffed_tn, on = 'tconst')
```
executed in 15ms, finished 13:41:04 2021-05-01

In [112]:
```
inner_sect.info()
```
executed in 15ms, finished 13:41:05 2021-05-01

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1125 entries, 0 to 1124
Data columns (total 23 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   tconst              1125 non-null   object 
 1   Title_x             1125 non-null   object 
 2   Genres_x            1125 non-null   object 
 3   runtime_minutes_x   1125 non-null   float64
 4   vote_average_x      1125 non-null   float64
 5   vote_count_x        1125 non-null   int64  
 6   budget_est_x        1125 non-null   float64
 7   worldwide_gross_x   1125 non-null   float64
 8   release_week_day_x  1125 non-null   int64  
 9   release_day_x       1125 non-null   int64  
 10  release_month_x     1125 non-null   int64  
 11  release_year_x      1125 non-null   int64  
 12  Title_y             1125 non-null   object 
 13  Genres_y            1125 non-null   object 
 14  runtime_minutes_y   1125 non-null   float64
 15  vote_average_y      1125 non-null   float64
 16  vote_count_y        1125 non-null   float64
 17  budget_est_y        1125 non-null   int64  
 18  worldwide_gross_y   1125 non-null   int64  
 19  release_week_day_y  1125 non-null   int64  
 20  release_day_y       1125 non-null   int64  
 21  release_month_y     1125 non-null   int64  
 22  release_year_y      1125 non-null   int64  
dtypes: float64(7), int64(11), object(5)
memory usage: 210.9+ KB
```

In [113]:
```
inner_sect.head()
```
executed in 30ms, finished 13:41:05 2021-05-01

Out[113]:

| | tconst | Title_x | Genres_x | runtime_minutes_x | vote_average_x | vote_count_x | budget_est |
|---|---|---|---|---|---|---|---|
| **0** | tt0892769 | How to Train Your Dragon | [Fantasy, Adventure, Animation, Family] | 98.0 | 7.7 | 7610 | 165000000 |
| **1** | tt1228705 | Iron Man 2 | [Adventure, Action, Science Fiction] | 124.0 | 6.8 | 12368 | 200000000 |
| **2** | tt1375666 | Inception | [Action, Science Fiction, Adventure] | 148.0 | 8.3 | 22186 | 160000000 |
| **3** | tt0814255 | Percy Jackson & the Olympians: The Lightning T... | [Adventure, Fantasy, Family] | 118.0 | 6.1 | 4229 | 95000000 |
| **4** | tt0435761 | Toy Story 3 | [Animation, Family, Comedy] | 103.0 | 7.7 | 8340 | 200000000 |

5 rows × 23 columns

In [114]:
```
common_tconst = pd.DataFrame([inner_sect.tconst,
                              inner_sect.Title_x,
                              inner_sect.Genres_x,
                              (inner_sect.runtime_minutes_x + inner_sect.runtime_
                             (((inner_sect.vote_average_x * inner_sect.vote_count
                             (inner_sect.vote_count_x + inner_sect.vote_count_y),
                             (inner_sect.budget_est_x + inner_sect.budget_est_y)
                             (inner_sect.worldwide_gross_x + inner_sect.worldwide
                              inner_sect.release_week_day_x, inner_sect.release_d
                              inner_sect.release_month_x, inner_sect.release_year
```
executed in 92ms, finished 13:41:06 2021-05-01

In [115]:
```
common_tconst.head()
```
executed in 15ms, finished 13:41:07 2021-05-01

Out[115]:

| | tconst | Title_x | Genres_x | Unnamed 0 | Unnamed 1 | Unnamed 2 | Unnamed 3 | Unnamed 4 | rel |
|---|---|---|---|---|---|---|---|---|---|
| **0** | tt0892769 | How to Train Your Dragon | [Fantasy, Adventure, Animation, Family] | 98 | 8.09508 | 618909 | 1.65e+08 | 4.94875e+08 | |
| **1** | tt1228705 | Iron Man 2 | [Adventure, Action, Science Fiction] | 124 | 6.99631 | 670058 | 1.85e+08 | 6.22545e+08 | |
| **2** | tt1375666 | Inception | [Action, Science Fiction, Adventure] | 148 | 8.79405 | 1.86325e+06 | 1.6e+08 | 8.30529e+08 | |
| **3** | tt0814255 | Percy Jackson & the Olympians: The Lightning T... | [Adventure, Fantasy, Family] | 118 | 5.90499 | 169337 | 9.5e+07 | 2.24774e+08 | |
| **4** | tt0435761 | Toy Story 3 | [Animation, Family, Comedy] | 103 | 8.29275 | 690558 | 2e+08 | 1.06792e+09 | |

In [116]:
```
common_tconst.columns = cleaned_tmdb.columns
```
executed in 14ms, finished 13:41:07 2021-05-01

In [117]:
```
common_tconst.head()
```
executed in 15ms, finished 13:41:08 2021-05-01

Out[117]:

| | tconst | Title | Genres | runtime_minutes | vote_average | vote_count | budget_est | worl |
|---|---|---|---|---|---|---|---|---|
| **0** | tt0892769 | How to Train Your Dragon | [Fantasy, Adventure, Animation, Family] | 98 | 8.09508 | 618909 | 1.65e+08 | |
| **1** | tt1228705 | Iron Man 2 | [Adventure, Action, Science Fiction] | 124 | 6.99631 | 670058 | 1.85e+08 | |
| **2** | tt1375666 | Inception | [Action, Science Fiction, Adventure] | 148 | 8.79405 | 1.86325e+06 | 1.6e+08 | |
| **3** | tt0814255 | Percy Jackson & the Olympians: The Lightning T... | [Adventure, Fantasy, Family] | 118 | 5.90499 | 169337 | 9.5e+07 | |
| **4** | tt0435761 | Toy Story 3 | [Animation, Family, Comedy] | 103 | 8.29275 | 690558 | 2e+08 | |

In [118]:
```
outer_sect = movie_collection_w_dups.loc[~movie_collection_w_dups.tconst.isin(com
```
executed in 15ms, finished 13:41:09 2021-05-01

In [119]:
```
outer_sect.head()
```
executed in 13ms, finished 13:41:09 2021-05-01

Out[119]:

| | tconst | Title | Genres | runtime_minutes | vote_average | vote_count | budget_est | wo |
|---|---|---|---|---|---|---|---|---|
| **0** | tt0926084 | Harry Potter and the Deathly Hallows: Part 1 | [Adventure, Fantasy, Family] | 146.0 | 7.7 | 10788.0 | 250000000.0 | |
| **3** | tt0114709 | Toy Story | [Animation, Comedy, Family] | 81.0 | 7.9 | 10174.0 | 30000000.0 | |
| **6** | tt0499549 | Avatar | [Action, Adventure, Fantasy, Science Fiction] | 162.0 | 7.4 | 18676.0 | 237000000.0 | |
| **10** | tt0120363 | Toy Story 2 | [Animation, Comedy, Family] | 92.0 | 7.5 | 7553.0 | 90000000.0 | |
| **17** | tt0372183 | The Bourne Supremacy | [Action, Drama, Thriller] | 108.0 | 7.3 | 4367.0 | 75000000.0 | |

In [120]:
```
outer_sect.info()
```
executed in 11ms, finished 13:41:10 2021-05-01

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1248 entries, 0 to 1354
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   tconst            1248 non-null   object 
 1   Title             1248 non-null   object 
 2   Genres            1248 non-null   object 
 3   runtime_minutes   1248 non-null   float64
 4   vote_average      1248 non-null   float64
 5   vote_count        1248 non-null   float64
 6   budget_est        1248 non-null   float64
 7   worldwide_gross   1248 non-null   float64
 8   release_week_day  1248 non-null   int64  
 9   release_day       1248 non-null   int64  
 10  release_month     1248 non-null   int64  
 11  release_year      1248 non-null   int64  
dtypes: float64(5), int64(4), object(3)
memory usage: 126.8+ KB
```

In [121]:
```
common_tconst.info()
```
executed in 14ms, finished 13:41:11 2021-05-01

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1125 entries, 0 to 1124
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   tconst            1125 non-null   object
 1   Title             1125 non-null   object
 2   Genres            1125 non-null   object
 3   runtime_minutes   1125 non-null   object
 4   vote_average      1125 non-null   object
 5   vote_count        1125 non-null   object
 6   budget_est        1125 non-null   object
 7   worldwide_gross   1125 non-null   object
 8   release_week_day  1125 non-null   object
 9   release_day       1125 non-null   object
 10  release_month     1125 non-null   object
 11  release_year      1125 non-null   object
dtypes: object(12)
memory usage: 154.3+ KB
```

In [122]:
```
movie_collection = pd.concat([common_tconst,outer_sect])
```
executed in 14ms, finished 13:41:12 2021-05-01

In [123]:
```
movie_collection.head()
```
executed in 15ms, finished 13:41:12 2021-05-01

Out[123]:

| | tconst | Title | Genres | runtime_minutes | vote_average | vote_count | budget_est | worl |
|---|---|---|---|---|---|---|---|---|
| **0** | tt0892769 | How to Train Your Dragon | [Fantasy, Adventure, Animation, Family] | 98 | 8.09508 | 618909 | 1.65e+08 | |
| **1** | tt1228705 | Iron Man 2 | [Adventure, Action, Science Fiction] | 124 | 6.99631 | 670058 | 1.85e+08 | |
| **2** | tt1375666 | Inception | [Action, Science Fiction, Adventure] | 148 | 8.79405 | 1.86325e+06 | 1.6e+08 | |
| **3** | tt0814255 | Percy Jackson & the Olympians: The Lightning T... | [Adventure, Fantasy, Family] | 118 | 5.90499 | 169337 | 9.5e+07 | |
| **4** | tt0435761 | Toy Story 3 | [Animation, Family, Comedy] | 103 | 8.29275 | 690558 | 2e+08 | |

In [124]:
```
movie_collection = movie_collection.reset_index()
movie_collection = movie_collection.drop(columns = 'index')
movie_collection.info()
```

executed in 15ms, finished 13:41:13 2021-05-01

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2373 entries, 0 to 2372
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   tconst            2373 non-null   object
 1   Title             2373 non-null   object
 2   Genres            2373 non-null   object
 3   runtime_minutes   2373 non-null   object
 4   vote_average      2373 non-null   object
 5   vote_count        2373 non-null   object
 6   budget_est        2373 non-null   object
 7   worldwide_gross   2373 non-null   object
 8   release_week_day  2373 non-null   object
 9   release_day       2373 non-null   object
 10  release_month     2373 non-null   object
 11  release_year      2373 non-null   object
dtypes: object(12)
memory usage: 222.6+ KB
```

In [125]:
```
movie_collection.tconst.nunique()
```

executed in 14ms, finished 13:41:14 2021-05-01

Out[125]:
```
2373
```

Sick! The dataframe is basically ready. For some reason all the columns are of object datatype, let's see if we can cast the numeric columns differently.

In [126]:
```
movie_collection = movie_collection.astype({'runtime_minutes': 'int32', 'vote_ave
                                            'budget_est': 'float64', 'worldwide_g
                                            'release_month': 'int32', 'release_ye
```

executed in 14ms, finished 13:41:14 2021-05-01

In [127]:
```
movie_collection.info()
```
executed in 15ms, finished 13:41:15 2021-05-01

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2373 entries, 0 to 2372
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   tconst            2373 non-null   object 
 1   Title             2373 non-null   object 
 2   Genres            2373 non-null   object 
 3   runtime_minutes   2373 non-null   int32  
 4   vote_average      2373 non-null   float64
 5   vote_count        2373 non-null   int32  
 6   budget_est        2373 non-null   float64
 7   worldwide_gross   2373 non-null   float64
 8   release_week_day  2373 non-null   object 
 9   release_day       2373 non-null   int32  
 10  release_month     2373 non-null   int32  
 11  release_year      2373 non-null   int32  
dtypes: float64(3), int32(5), object(4)
memory usage: 176.2+ KB
```

In [128]:
```
movie_collection.to_csv('./dataFiles/movie_collection.csv',index=False)
```
executed in 44ms, finished 13:41:16 2021-05-01

Woohoo! It's done. Now, in the analytics notebook we will still want to add columns such as ROI and maybe release day of the week. Additionally, we should be able to connect to different cast and crew members using tconst and joining title_principals and name_basics.