Module 8.2.1 - Extract the Wikipedia Movies JSON

In [1]:
```
#By Emmanuel Martinez
```

In [2]:
```
import json
import pandas as pd
import numpy as np
import os
```

In [3]:
```
file_dir = "C://Users/Emilio/Google Drive (emilio.martinez@palabradeamor.org)/Col
```

In [4]:
```
f'{file_dir}'
```

Out[4]:
```
'C://Users/Emilio/Google Drive (emilio.martinez@palabradeamor.org)/Columbia Uni
versity/GitHub - emmanuelmartinezs/Movies-ETL'
```

In [5]:
```
with open(f'{file_dir}/wikipedia-movies.json', mode='r') as file:
    wiki_movies_raw = json.load(file)
```

In [6]:
```
len(wiki_movies_raw)
```

Out[6]:
```
7311
```

In [7]:
```
# First 5 records
wiki_movies_raw[:5]
```

Out[7]:
```
[{'url': 'https://en.wikipedia.org/wiki/The_Adventures_of_Ford_Fairlane',
  'year': 1990,
  'imdb_link': 'https://www.imdb.com/title/tt0098987/',
  'title': 'The Adventures of Ford Fairlane',
  'Directed by': 'Renny Harlin',
  'Produced by': ['Steve Perry', 'Joel Silver'],
  'Screenplay by': ['David Arnott', 'James Cappe', 'Daniel Waters'],
  'Story by': ['David Arnott', 'James Cappe'],
  'Based on': ['Characters', 'by Rex Weiner'],
  'Starring': ['Andrew Dice Clay',
   'Wayne Newton',
   'Priscilla Presley',
   'Lauren Holly',
   'Morris Day',
   'Robert Englund',
   "Ed O'Neill"],
  'Narrated by': 'Andrew "Dice" Clay',
  'Music by': ['Cliff Eidelman', 'Yello'],
  'Cinematography': 'Oliver Wood',
```

In [8]:
```
# Last 5 records
wiki_movies_raw[-5:]
```

Out[8]:
```
[{'url': 'https://en.wikipedia.org/wiki/Holmes_%26_Watson',
  'year': 2018,
  'imdb_link': 'https://www.imdb.com/title/tt1255919/',
  'title': 'Holmes & Watson',
  'Directed by': 'Etan Cohen',
  'Produced by': ['Will Ferrell',
   'Adam McKay',
   'Jimmy Miller',
   'Clayton Townsend'],
  'Screenplay by': 'Etan Cohen',
  'Based on': ['Sherlock Holmes',
   'and',
   'Dr. Watson',
   'by',
   'Sir Arthur Conan Doyle'],
  'Starring': ['Will Ferrell',
   'John C. Reilly',
   'Rebecca Hall',
   'Rob Brydon',
```

In [9]:
```
# Some records in the middle
wiki_movies_raw[3600:3605]
```

Out[9]:
```
[{'url': 'https://en.wikipedia.org/wiki/Benji:_Off_the_Leash!',
  'year': 2004,
  'imdb_link': 'https://www.imdb.com/title/tt0315273/',
  'title': 'Benji: Off the Leash!',
  'Directed by': 'Joe Camp',
  'Written by': 'Joe Camp',
  'Starring': ['Benji', 'Nick Whitaker', 'Shaggy', 'Gypsy the Cockatoo'],
  'Music by': 'Antonio di Lorenzo',
  'Productioncompany ': 'Mulberry Square Productions',
  'Distributed by': 'Mulberry Square Productions',
  'Release date': ['March 26, 2004', '(', '2004-03-26', ')'],
  'Running time': '97 min',
  'Country': 'United States',
  'Language': 'English',
  'Box office': '$3,817,362'},
 {'url': 'https://en.wikipedia.org/wiki/The_Best_Thief_in_the_World',
  'year': 2004,
  'imdb_link': 'https://www.imdb.com/title/tt0389796/',
  'title': 'The Best Thief in the World',
```

Module 8.2.2 - Extract the Kaggle Data

In [10]:
```
kaggle_metadata = pd.read_csv(f'{file_dir}/movies_metadata.csv', low_memory=False
ratings = pd.read_csv(f'{file_dir}/ratings.csv')
```

In [11]:
```
kaggle_metadata.head()
```

Out[11]:

	adult	belongs_to_collection	budget	genres	homepage	id	imdb_id
0	False	{'id': 10194, 'name': 'Toy Story Collection', ...	30000000	[{'id': 16, 'name': 'Animation'}, {'id': 35, '...	http://toystory.disney.com/toy-story	862	tt0114709
1	False	NaN	65000000	[{'id': 12, 'name': 'Adventure'}, {'id': 14, '...	NaN	8844	tt0113497
2	False	{'id': 119050, 'name': 'Grumpy Old Men Collect...	0	[{'id': 10749, 'name': 'Romance'}, {'id': 35, ...	NaN	15602	tt0113228
3	False	NaN	16000000	[{'id': 35, 'name': 'Comedy'}, {'id': 18, 'nam...	NaN	31357	tt0114885
4	False	{'id': 96871, 'name': 'Father of the Bride Col...	0	[{'id': 35, 'name': 'Comedy'}]	NaN	11862	tt0113041

5 rows × 24 columns

In [12]:
```
kaggle_metadata.tail()
```

Out[12]:

	adult	belongs_to_collection	budget	genres	homepage	id	in
45461	False	NaN	0	[{'id': 18, 'name': 'Drama'}, {'id': 10751, 'n...	http://www.imdb.com/title/tt6209470/	439050	tt6
45462	False	NaN	0	[{'id': 18, 'name': 'Drama'}]	NaN	111109	tt2
45463	False	NaN	0	[{'id': 28, 'name': 'Action'}, {'id': 18, 'nam...	NaN	67758	tt0
45464	False	NaN	0	[]	NaN	227506	tt0
45465	False	NaN	0	[]	NaN	461257	tt6

5 rows × 24 columns

In [13]:
```
kaggle_metadata.sample()
```

Out[13]:

	adult	belongs_to_collection	budget	genres	homepage	id	imdb_id	original_languag
7244	False	NaN	0	[{'id': 18, 'name': 'Drama'}, {'id': 10749, 'n...	NaN	95548	tt0058930	e

1 rows × 24 columns

In [14]:
```
kaggle_metadata.sample(n=5)
```

Out[14]:

	adult	belongs_to_collection	budget	genres	homepage	id	imdb_id	original_l
281	False	NaN	4800000	[{'id': 27, 'name': 'Horror'}]	NaN	56428	tt0106402	
11139	False	NaN	0	[{'id': 10749, 'name': 'Romance'}, {'id': 35, ...	NaN	2805	tt0269217	
39740	False	NaN	0	[]	NaN	273302	tt0284608	
22469	False	NaN	18000000	[{'id': 18, 'name': 'Drama'}, {'id': 36, 'name...	NaN	127560	tt2058107	
2995	False	NaN	0	[{'id': 18, 'name': 'Drama'}]	NaN	34760	tt0090556	

5 rows × 24 columns

In [15]:
```
ratings.head()
```

Out[15]:

	userId	movieId	rating	timestamp
0	1	110	1.0	1425941529
1	1	147	4.5	1425942435
2	1	858	5.0	1425941523
3	1	1221	5.0	1425941546
4	1	1246	5.0	1425941556

In [16]:
```
ratings.tail()
```

Out[16]:

	userId	movieId	rating	timestamp
26024284	270896	58559	5.0	1257031564
26024285	270896	60069	5.0	1257032032
26024286	270896	63082	4.5	1257031764
26024287	270896	64957	4.5	1257033990
26024288	270896	71878	2.0	1257031858

In [17]:
```
ratings.sample()
```

Out[17]:

	userId	movieId	rating	timestamp
11231193	116363	54775	1.5	1388703762

In [18]:
```
ratings.sample(n=5)
```

Out[18]:

	userId	movieId	rating	timestamp
10686465	110324	1101	1.0	1153943587
18018643	187049	589	4.0	939063561
12400448	128628	750	3.0	1498647104
9996205	103112	3996	1.5	1129835348
15985936	166340	54001	3.5	1185380961

Module 8.3.3 - Investigate the Wikipedia Data

In [19]:
```
wiki_movies_df = pd.DataFrame(wiki_movies_raw)
```

In [20]:
```
wiki_movies_df.head()
```

Out[20]:

	url	year	imdb_link	title
0	https://en.wikipedia.org/wiki/The_Adventures_o...	1990.0	https://www.imdb.com/title/tt0098987/	The Adventures of Ford Fairlane
1	https://en.wikipedia.org/wiki/After_Dark,_My_S...	1990.0	https://www.imdb.com/title/tt0098994/	After Dark, My Sweet
2	https://en.wikipedia.org/wiki/Air_America_(film)	1990.0	https://www.imdb.com/title/tt0099005/	Air America
3	https://en.wikipedia.org/wiki/Alice_(1990_film)	1990.0	https://www.imdb.com/title/tt0099012/	Alice
4	https://en.wikipedia.org/wiki/Almost_an_Angel	1990.0	https://www.imdb.com/title/tt0099018/	Almost an Angel

5 rows × 193 columns

In [21]:
```
print(wiki_movies_df.columns)
```

```
Index(['url', 'year', 'imdb_link', 'title', 'Directed by', 'Produced by',
       'Screenplay by', 'Story by', 'Based on', 'Starring',
       ...
       'Predecessor', 'Founders', 'Area served', 'Products', 'Services',
       'Russian', 'Hebrew', 'Revenue', 'Operating income', 'Polish'],
      dtype='object', length=193)
```

In [22]:
```
wiki_movies_df.columns
```

Out[22]:
```
Index(['url', 'year', 'imdb_link', 'title', 'Directed by', 'Produced by',
       'Screenplay by', 'Story by', 'Based on', 'Starring',
       ...
       'Predecessor', 'Founders', 'Area served', 'Products', 'Services',
       'Russian', 'Hebrew', 'Revenue', 'Operating income', 'Polish'],
      dtype='object', length=193)
```

In [23]:
```
wiki_movies_df.columns.tolist()
```

Out[23]:
```
['url',
 'year',
 'imdb_link',
 'title',
 'Directed by',
 'Produced by',
 'Screenplay by',
 'Story by',
 'Based on',
 'Starring',
 'Narrated by',
 'Music by',
 'Cinematography',
 'Edited by',
 'Productioncompany ',
 'Distributed by',
 'Release date',
 'Running time',
 'Country',
```

In [24]:
```
# Use List Comprehensions to Filter Data
```

In [25]:
```
wiki_movies = [movie for movie in wiki_movies_raw
               if ('Director' in movie or 'Directed by' in movie)
                   and 'imdb_link' in movie]
len(wiki_movies)
```

Out[25]:
```
7080
```

In [26]:
```
wiki_movies = [movie for movie in wiki_movies_raw
               if ('Director' in movie or 'Directed by' in movie)
                   and 'imdb_link' in movie
                   and 'No. of episodes' not in movie]
```

In [27]:
```
len(wiki_movies)
```

Out[27]:
```
7076
```

Module 8.3.4 - Revisit Functions

In [28]:
```
x = 'global value'

def foo():
    x = 'local value'
    print(x)

foo()
print(x)
```

```
local value
global value
```

In [29]:
```
my_list = [1,2,3]
def append_four(x):
    x.append(4)
append_four(my_list)
print(my_list)
```

```
[1, 2, 3, 4]
```

In [30]:
```
# Lambda Functions
```

In [31]:
```
square = lambda x: x * x
square(5)
```

Out[31]:
```
25
```

Module 8.3.5 - Create a Function to Clean the Data, Part 1

In [32]:
```
def clean_movie(movie):
    movie_copy = dict(movie)
```

In [33]:
```
# Below the best format and function to copy a dataset using a var with same name
```

In [34]:
```
def clean_movie(movie):
    movie = dict(movie) #create a non-destructive copy
    return movie
```

In [35]:
```
wiki_movies_df[wiki_movies_df['Arabic'].notnull()]
```

Out[35]:

	url	year	imdb_link	titl
7060	https://en.wikipedia.org/wiki/The_Insult_(film)	2018.0	https://www.imdb.com/title/tt7048622/	The Insu
7293	https://en.wikipedia.org/wiki/Capernaum_(film)	2018.0	https://www.imdb.com/title/tt8267604/	Capernaur

2 rows × 193 columns

In [36]:
```
wiki_movies_df[wiki_movies_df['Arabic'].notnull()]['url']
```

Out[36]:
```
7060    https://en.wikipedia.org/wiki/The_Insult_(film) (https://en.wikipedia.org/wiki/The_Insult_(film))
7293     https://en.wikipedia.org/wiki/Capernaum_(film) (https://en.wikipedia.org/wiki/Capernaum_(film))
Name: url, dtype: object
```

In [37]:
```
sorted(wiki_movies_df.columns.tolist())
```

Out[37]:
```
['Actor control',
 'Adaptation by',
 'Alias',
 'Alma mater',
 'Also known as',
 'Animation by',
 'Arabic',
 'Area',
 'Area served',
 'Artist(s)',
 'Attraction type',
 'Audio format',
 'Author',
 'Based on',
 'Biographical data',
 'Bopomofo',
 'Born',
 'Box office',
 'Budget',
```

Handle the Alternative Titles

Handle the Alternative Titles

In [38]:
```
# Step 1: Make an empty dict to hold all of the alternative titles.
```

In [39]:
```
def clean_movie(movie):
    movie = dict(movie) #create a non-destructive copy
    alt_titles = {}
    return movie
```

In [40]:
```
# Step 2: Loop through a list of all alternative title keys.
```

In [41]:
```
# this will throw an error!

# def clean_movie(movie):
#     movie = dict(movie) #create a non-destructive copy
#     alt_titles = {}
#     for key in ['Also known as','Arabic','Cantonese','Chinese','French',
#                 'Hangul','Hebrew','Hepburn','Japanese','Literally',
#                 'Mandarin','McCune-Reischauer','Original title','Polish',
#                 'Revised Romanization','Romanized','Russian',
#                 'Simplified','Traditional','Yiddish']:
#
#     return movie
```

In [42]:
```
# Step 2a: Check if the current key exists in the movie object.
```

In [43]:
```
# this will throw an error!

# def clean_movie(movie):
#     movie = dict(movie) #create a non-destructive copy
#     alt_titles = {}
#     for key in ['Also known as','Arabic','Cantonese','Chinese','French',
#                 'Hangul','Hebrew','Hepburn','Japanese','Literally',
#                 'Mandarin','McCune-Reischauer','Original title','Polish',
#                 'Revised Romanization','Romanized','Russian',
#                 'Simplified','Traditional','Yiddish']:
#         if key in movie:
#
#     return movie
```

In [44]:
```
# Step 2b: If so, remove the key-value pair and add to the alternative titles dic
```

In [45]:
```
def clean_movie(movie):
    movie = dict(movie) #create a non-destructive copy
    alt_titles = {}
    for key in ['Also known as','Arabic','Cantonese','Chinese','French',
                'Hangul','Hebrew','Hepburn','Japanese','Literally',
                'Mandarin','McCune-Reischauer','Original title','Polish',
                'Revised Romanization','Romanized','Russian',
                'Simplified','Traditional','Yiddish']:
        if key in movie:
            alt_titles[key] = movie[key]
            movie.pop(key)


    return movie
```

In [46]:
```
# Step 3: After looping through every key, add the alternative titles dict to the
```

In [47]:
```
def clean_movie(movie):
    movie = dict(movie) #create a non-destructive copy
    alt_titles = {}
    for key in ['Also known as','Arabic','Cantonese','Chinese','French',
                'Hangul','Hebrew','Hepburn','Japanese','Literally',
                'Mandarin','McCune-Reischauer','Original title','Polish',
                'Revised Romanization','Romanized','Russian',
                'Simplified','Traditional','Yiddish']:
        if key in movie:
            alt_titles[key] = movie[key]
            movie.pop(key)
    if len(alt_titles) > 0:
        movie['alt_titles'] = alt_titles

    return movie
```

In [48]:
```
clean_movies = [clean_movie(movie) for movie in wiki_movies]
```

In [49]:
```
wiki_movies_df = pd.DataFrame(clean_movies)
sorted(wiki_movies_df.columns.tolist())
```

Out[49]:
```
['Adaptation by',
 'Animation by',
 'Audio format',
 'Based on',
 'Box office',
 'Budget',
 'Cinematography',
 'Color process',
 'Composer(s)',
 'Country',
 'Country of origin',
 'Created by',
 'Directed by',
 'Director',
 'Distributed by',
 'Distributor',
 'Edited by',
 'Editor(s)',
 'Executive producer(s)',
```

Module 8.3.6 - Create a Function to Clean the Data, Part 2

In [50]:

```
def clean_movie(movie):
    movie = dict(movie) #create a non-destructive copy
    alt_titles = {}
    # combine alternate titles into one list
    for key in ['Also known as','Arabic','Cantonese','Chinese','French',
                'Hangul','Hebrew','Hepburn','Japanese','Literally',
                'Mandarin','McCune-Reischauer','Original title','Polish',
                'Revised Romanization','Romanized','Russian',
                'Simplified','Traditional','Yiddish']:
        if key in movie:
            alt_titles[key] = movie[key]
            movie.pop(key)
    if len(alt_titles) > 0:
        movie['alt_titles'] = alt_titles

    # merge column names
    def change_column_name(old_name, new_name):
        if old_name in movie:
            movie[new_name] = movie.pop(old_name)
    change_column_name('Adaptation by', 'Writer(s)')
    change_column_name('Country of origin', 'Country')
    change_column_name('Directed by', 'Director')
    change_column_name('Distributed by', 'Distributor')
    change_column_name('Edited by', 'Editor(s)')
    change_column_name('Length', 'Running time')
    change_column_name('Original release', 'Release date')
    change_column_name('Music by', 'Composer(s)')
    change_column_name('Produced by', 'Producer(s)')
    change_column_name('Producer', 'Producer(s)')
    change_column_name('Productioncompanies ', 'Production company(s)')
    change_column_name('Productioncompany ', 'Production company(s)')
    change_column_name('Released', 'Release Date')
    change_column_name('Release Date', 'Release date')
    change_column_name('Screen story by', 'Writer(s)')
    change_column_name('Screenplay by', 'Writer(s)')
    change_column_name('Story by', 'Writer(s)')
    change_column_name('Theme music composer', 'Composer(s)')
    change_column_name('Written by', 'Writer(s)')

    return movie
```

In [51]:
```
clean_movies = [clean_movie(movie) for movie in wiki_movies]
wiki_movies_df = pd.DataFrame(clean_movies)
sorted(wiki_movies_df.columns.tolist())
```

Out[51]:
```
['Animation by',
 'Audio format',
 'Based on',
 'Box office',
 'Budget',
 'Cinematography',
 'Color process',
 'Composer(s)',
 'Country',
 'Created by',
 'Director',
 'Distributor',
 'Editor(s)',
 'Executive producer(s)',
 'Followed by',
 'Genre',
 'Label',
 'Language',
 'McCune–Reischauer',
 'Narrated by',
 'Original language(s)',
 'Original network',
 'Picture format',
 'Preceded by',
 'Producer(s)',
 'Production company(s)',
 'Production location(s)',
 'Recorded',
 'Release date',
 'Running time',
 'Starring',
 'Suggested by',
 'Venue',
 'Voices of',
 'Writer(s)',
 'alt_titles',
 'imdb_link',
 'title',
 'url',
 'year']
```

Module 8.3.7 - Remove Duplicate Rows

In [52]:
```
wiki_movies_df['imdb_id'] = wiki_movies_df['imdb_link'].str.extract(r'(tt\d{7})')
print(len(wiki_movies_df))
wiki_movies_df.drop_duplicates(subset='imdb_id', inplace=True)
print(len(wiki_movies_df))
wiki_movies_df.head()
```

```
7076
7033
```

Out[52]:

	url	year	imdb_link	title
0	https://en.wikipedia.org/wiki/The_Adventures_o...	1990	https://www.imdb.com/title/tt0098987/	The Adventures of Ford Fairlane
1	https://en.wikipedia.org/wiki/After_Dark,_My_S...	1990	https://www.imdb.com/title/tt0098994/	After Dark, My Sweet
2	https://en.wikipedia.org/wiki/Air_America_(film)	1990	https://www.imdb.com/title/tt0099005/	Air America
3	https://en.wikipedia.org/wiki/Alice_(1990_film)	1990	https://www.imdb.com/title/tt0099012/	Alice
4	https://en.wikipedia.org/wiki/Almost_an_Angel	1990	https://www.imdb.com/title/tt0099018/	Almost an Angel

5 rows × 41 columns

In [53]:
```
[[column,wiki_movies_df[column].isnull().sum()] for column in wiki_movies_df.colu
```

Out[53]:
```
[['url', 0],
 ['year', 0],
 ['imdb_link', 0],
 ['title', 1],
 ['Based on', 4852],
 ['Starring', 184],
 ['Narrated by', 6752],
 ['Cinematography', 691],
 ['Release date', 32],
 ['Running time', 139],
 ['Country', 236],
 ['Language', 244],
 ['Budget', 2295],
 ['Box office', 1548],
 ['Director', 0],
 ['Distributor', 357],
 ['Editor(s)', 548],
 ['Composer(s)', 518],
 ['Producer(s)', 202],
 ['Production company(s)', 1678],
 ['Writer(s)', 199],
 ['Genre', 6923],
 ['Original language(s)', 6875],
 ['Original network', 6908],
 ['Executive producer(s)', 6936],
 ['Production location(s)', 6986],
 ['Picture format', 6969],
 ['Audio format', 6972],
 ['Voices of', 7031],
 ['Followed by', 7024],
 ['Created by', 7023],
 ['Preceded by', 7023],
 ['Suggested by', 7032],
 ['alt_titles', 7012],
 ['Recorded', 7031],
 ['Venue', 7032],
 ['Label', 7031],
 ['Animation by', 7031],
 ['Color process', 7032],
 ['McCune–Reischauer', 7031],
 ['imdb_id', 0]]
```

In [54]:
```
[column for column in wiki_movies_df.columns if wiki_movies_df[column].isnull().s
```

Out[54]:
```
['url',
 'year',
 'imdb_link',
 'title',
 'Based on',
 'Starring',
 'Cinematography',
 'Release date',
 'Running time',
 'Country',
 'Language',
 'Budget',
 'Box office',
 'Director',
 'Distributor',
 'Editor(s)',
 'Composer(s)',
 'Producer(s)',
 'Production company(s)',
 'Writer(s)',
 'imdb_id']
```

In [55]:
```
wiki_columns_to_keep = [column for column in wiki_movies_df.columns if wiki_movie
wiki_movies_df = wiki_movies_df[wiki_columns_to_keep]
```

Module 8.3.8 - Make a Plan to Convert and Parse the Data

In [56]:
```
box_office = wiki_movies_df['Box office'].dropna()
```

In [57]:
```
def is_not_a_string(x):
    return type(x) != str
```

In [58]:
```
box_office[box_office.map(is_not_a_string)]
```

Out[58]:
```
34                            [US$, 4,212,828]
54      [$6,698,361 (, United States, ), [2]]
74                     [$6,488,144, (US), [1]]
126                [US$1,531,489, (domestic)]
130                           [US$, 4,803,039]
                          ...
6980               [$99.6, million, [4], [5]]
6994                   [$365.6, million, [1]]
6995                         [$53.8, million]
7015                    [$435, million, [7]]
7048                   [$529.3, million, [4]]
Name: Box office, Length: 135, dtype: object
```

In [59]:
```
lambda x: type(x) != str
```

Out[59]:
```
<function __main__.<lambda>(x)>
```

In [60]:
```
box_office[box_office.map(lambda x: type(x) != str)]
```

Out[60]:
```
34                            [US$, 4,212,828]
54      [$6,698,361 (, United States, ), [2]]
74                     [$6,488,144, (US), [1]]
126                [US$1,531,489, (domestic)]
130                           [US$, 4,803,039]
                          ...
6980               [$99.6, million, [4], [5]]
6994                   [$365.6, million, [1]]
6995                         [$53.8, million]
7015                    [$435, million, [7]]
7048                   [$529.3, million, [4]]
Name: Box office, Length: 135, dtype: object
```

In [61]:
```
some_list = ['One','Two','Three']
'Mississippi '.join(some_list)
```

Out[61]:
```
'OneMississippi TwoMississippi Three'
```

In [62]:
```
box_office = box_office.apply(lambda x: ' '.join(x) if type(x) == list else x)
```

Module 8.3.9 - Write Regular Expressions

In [63]:
```
import re
```

Module 8.3.10 - Parse the Box Office Data

In [64]:
```
form_one = r'\$\d+\.?\d*\s*[mb]illion'
```

In [65]:
```
box_office.str.contains(form_one, flags=re.IGNORECASE).sum()
```

Out[65]:
```
3896
```

In [66]:
```
form_two = r'\$\d{1,3}(?:,\d{3})+'
box_office.str.contains(form_two, flags=re.IGNORECASE).sum()
```

Out[66]:
```
1544
```

In [67]:
```
matches_form_one = box_office.str.contains(form_one, flags=re.IGNORECASE)
matches_form_two = box_office.str.contains(form_two, flags=re.IGNORECASE)
```

In [68]:
```
# this will throw an error!
# box_office[(not matches_form_one) and (not matches_form_two)]
```

In [69]:
```
box_office[~matches_form_one & ~matches_form_two]
```

Out[69]:
```
34                          US$ 4,212,828
79                               $335.000
110                    $4.35-4.37 million
130                          US$ 4,803,039
600                             $5000 (US)
731                          $ 11,146,270
957                              $ 50,004
1070                           35,254,617
1147    $ 407,618 (U.S.) (sub-total) [1]
1446                         $ 11,829,959
1480                           £3 million
1611                             $520.000
1865                         ¥1.1 billion
2032                                 N/A
2091                                $309
2130                US$ 171.8 million [9]
2257                    US$ 3,395,581 [1]
2263            $ 1,223,034 ( domestic )
2347                             $282.175
2638            $ 104,883 (US sub-total)
2665         926,423 admissions (France)
2697      $ 1.7 million (US) (sub-total)
2823                             $414.000
2924                             $621.000
3088           $32 [2] –33.1 million [1]
3631                                 TBA
3859                  $38.9–40.3 million
3879            CN¥3.650 million (China)
4116                         £7,385,434
4123                             $161.000
4261                  $20.7–23.9 million
4306                               $20-30
4492                         $47.7 millon
4561             $45.2k (only in Turkey)
4662                USD$ 8.2 million [2]
5362                   $ 142 million [3]
5447                                £2.56
5784                            413 733$
6013                             Unknown
6145                  $17.5–18.4 million
6234                  $41.8–41.9 million
6369                               $111k
6370                                $588
6593                    less than $372
6829                   $ 41 million [3]
6843                            8 crore
6904                          $6.9 millon
Name: Box office, dtype: object
```

In [70]:
```
form_one = r'\$\s*\d+\.?\d*\s*[mb]illion'
form_two = r'\$\s*\d{1,3}(?:,\d{3})+'
```

In [71]:
```
form_two = r'\$\s*\d{1,3}(?:[,\.]\d{3})+'
```

In [72]:
```
form_two = r'\$\s*\d{1,3}(?:[,\.]\d{3})+(?!\s[mb]illion)'
```

In [73]:
```
box_office = box_office.str.replace(r'\$.*[-——](?![a-z])', '$', regex=True)
```

In [74]:
```
form_one = r'\$\s*\d+\.?\d*\s*[mb]illi?on'
```

In [75]:
```
box_office.str.extract(f'({form_one}|{form_two})')
```

Out[75]:

	0
0	$21.4 million
1	$2.7 million
2	$57,718,089
3	$7,331,647
4	$6,939,946
...	...
7070	$19.4 million
7071	$41.9 million
7072	$76.1 million
7073	$38.4 million
7074	$5.5 million

5485 rows × 1 columns

In [76]:
```
# def parse_dollars(s):
    # if s is not a string, return NaN

    # if input is of the form $###.# million

        # remove dollar sign and " million"

        # convert to float and multiply by a million

        # return value

    # if input is of the form $###.# billion

        # remove dollar sign and " billion"

        # convert to float and multiply by a billion

        # return value

    # if input is of the form $###,###,###

        # remove dollar sign and commas

        # convert to float

        # return value

# otherwise, return NaN
```

In [77]:
```
# def parse_dollars(s):
    # if s is not a string, return NaN
#    if type(s) != str:
#        return np.nan

    # if input is of the form $###.# million
#    if re.match(r'\$\s*\d+\.?\d*\s*milli?on', s, flags=re.IGNORECASE):

        # remove dollar sign and " million"

        # convert to float and multiply by a million

        # return value

    # if input is of the form $###.# billion
#    elif re.match(r'\$\s*\d+\.?\d*\s*billi?on', s, flags=re.IGNORECASE):

        # remove dollar sign and " billion"

        # convert to float and multiply by a billion

        # return value

    # if input is of the form $###,###,###
#    elif re.match(r'\$\s*\d{1,3}(?:[,\.]\d{3})+(?!\s[mb]illion)', s, flags=re.IG

        # remove dollar sign and commas

        # convert to float

        # return value

    # otherwise, return NaN
#    else:
#        return np.nan
```

In [78]:

```
def parse_dollars(s):
    # if s is not a string, return NaN
    if type(s) != str:
        return np.nan

    # if input is of the form $###.# million
    if re.match(r'\$\s*\d+\.?\d*\s*milli?on', s, flags=re.IGNORECASE):

        # remove dollar sign and " million"
        s = re.sub('\$|\s|[a-zA-Z]','', s)

        # convert to float and multiply by a million

        # return value

    # if input is of the form $###.# billion
    elif re.match(r'\$\s*\d+\.?\d*\s*billi?on', s, flags=re.IGNORECASE):

        # remove dollar sign and " billion"
        s = re.sub('\$|\s|[a-zA-Z]','', s)

        # convert to float and multiply by a billion

        # return value

    # if input is of the form $###,###,###
    elif re.match(r'\$\s*\d{1,3}(?:[,\.]\d{3})+(?!\s[mb]illion)', s, flags=re.IGN

        # remove dollar sign and commas
        s = re.sub('\$|,','', s)

        # convert to float

        # return value

    # otherwise, return NaN
    else:
        return np.nan
```

In [79]:
```
def parse_dollars(s):
    # if s is not a string, return NaN
    if type(s) != str:
        return np.nan

    # if input is of the form $###.# million
    if re.match(r'\$\s*\d+\.?\d*\s*milli?on', s, flags=re.IGNORECASE):

        # remove dollar sign and " million"
        s = re.sub('\$|\s|[a-zA-Z]','', s)

        # convert to float and multiply by a million
        value = float(s) * 10**6

        # return value
        return value

    # if input is of the form $###.# billion
    elif re.match(r'\$\s*\d+\.?\d*\s*billi?on', s, flags=re.IGNORECASE):

        # remove dollar sign and " billion"
        s = re.sub('\$|\s|[a-zA-Z]','', s)

        # convert to float and multiply by a billion
        value = float(s) * 10**9

        # return value
        return value

    # if input is of the form $###,###,###
    elif re.match(r'\$\s*\d{1,3}(?:[,\.]\d{3})+(?!\s[mb]illion)', s, flags=re.IGN

        # remove dollar sign and commas
        s = re.sub('\$|,','', s)

        # convert to float
        value = float(s)

        # return value
        return value

    # otherwise, return NaN
    else:
        return np.nan
```

In [80]:
```
wiki_movies_df['box_office'] = box_office.str.extract(f'({form_one}|{form_two})',
```

In [81]:
```
wiki_movies_df['box_office']
```

Out[81]:
```
0       21400000.0
1        2700000.0
2       57718089.0
3        7331647.0
4        6939946.0
           ...
7071    41900000.0
7072    76100000.0
7073    38400000.0
7074     5500000.0
7075           NaN
Name: box_office, Length: 7033, dtype: float64
```

In [82]:
```
wiki_movies_df.drop('Box office', axis=1, inplace=True)
```

Module 8.3.11 - Parse Budget Data

In [83]:
```
budget = wiki_movies_df['Budget'].dropna()
```

In [84]:
```
budget = budget.map(lambda x: ' '.join(x) if type(x) == list else x)
```

In [85]:
```
budget = budget.str.replace(r'\$.*[---](?![a-z])', '$', regex=True)
```

In [86]:
```
matches_form_one = budget.str.contains(form_one, flags=re.IGNORECASE)
matches_form_two = budget.str.contains(form_two, flags=re.IGNORECASE)
budget[~matches_form_one & ~matches_form_two]
```

Out[86]:
```
136                          Unknown
204     60 million Norwegian Kroner
478                          Unknown
973              $34 [3] [4] million
1126                $120 [4] million
1226                         Unknown
1278                             HBO
1374                     £6,000,000
1397                      13 million
1480                    £2.8 million
1734                   CAD2,000,000
1913     PHP 85 million (estimated)
1948                    102,888,900
1953                   3,500,000 DM
1973                     £2,300,874
2281                      $14 milion
2451                     £6,350,000
3144                   € 40 million
3360                $150 [6] million
3418                         $218.32
3802                    £4.2 million
3906                             N/A
3959                    760,000 USD
4470                       19 crore
4641                     £17 million
5034               $$200 [4] million
5055            $155 [2] [3] million
5419                 $40 [4] million
5424                             N/A
5447                      £4 million
5671                     €14 million
5687                   $ dead link]
6385                £ 12 million [3]
6593                      £3 million
6821                   £12.9 million
6843                       3.5 crore
6895                         919,000
7070                    €4.3 million
Name: Budget, dtype: object
```

In [87]:
```
budget = budget.str.replace(r'\[\d+\]\s*', '')
budget[~matches_form_one & ~matches_form_two]
```

Out[87]:
```
136                          Unknown
204     60 million Norwegian Kroner
478                          Unknown
973                       $34 million
1126                     $120 million
1226                         Unknown
1278                             HBO
1374                      £6,000,000
1397                       13 million
1480                     £2.8 million
1734                     CAD2,000,000
1913     PHP 85 million (estimated)
1948                      102,888,900
1953                    3,500,000 DM
1973                      £2,300,874
2281                       $14 milion
2451                      £6,350,000
3144                    € 40 million
3360                    $150 million
3418                         $218.32
3802                     £4.2 million
3906                             N/A
3959                     760,000 USD
4470                        19 crore
4641                     £17 million
5034                   $$200 million
5055                    $155 million
5419                     $40 million
5424                             N/A
5447                      £4 million
5671                     €14 million
5687                    $ dead link]
6385                   £ 12 million
6593                      £3 million
6821                   £12.9 million
6843                       3.5 crore
6895                         919,000
7070                     €4.3 million
Name: Budget, dtype: object
```

In [88]:
```
wiki_movies_df['budget'] = budget.str.extract(f'({form_one}|{form_two})', flags=r
```

In [89]:
```
wiki_movies_df.drop('Budget', axis=1, inplace=True)
```

In [90]:
```
release_date = wiki_movies_df['Release date'].dropna().apply(lambda x: ' '.join(x
```

In [91]:
```
date_form_one = r'(?:January|February|March|April|May|June|July|August|September|
date_form_two = r'\d{4}.[01]\d.[123]\d'
date_form_three = r'(?:January|February|March|April|May|June|July|August|Septembe
date_form_four = r'\d{4}'
```

In [92]:
```
release_date.str.extract(f'({date_form_one}|{date_form_two}|{date_form_three}|{da
```

Out[92]:

	0
0	July 11, 1990
1	May 17, 1990
2	August 10, 1990
3	December 25, 1990
4	December 19, 1990
...	...
7071	December 25, 2018
7072	December 11, 2018
7073	2018
7074	August 31, 2018
7075	December 2018

7001 rows × 1 columns

In [93]:
```
wiki_movies_df['release_date'] = pd.to_datetime(release_date.str.extract(f'({date
```

In [94]:
```
running_time = wiki_movies_df['Running time'].dropna().apply(lambda x: ' '.join(x
```

In [95]:
```
running_time.str.contains(r'^\d*\s*minutes$', flags=re.IGNORECASE).sum()
```

Out[95]:
```
6528
```

In [96]:
```
running_time[running_time.str.contains(r'^\d*\s*minutes$', flags=re.IGNORECASE) !
```

Out[96]:
```
9                                                    102 min
26                                                    93 min
28                                                   32 min.
34                                                   101 min
35                                                    97 min
                                  ...
6500       114 minutes [1] 120 minutes (extended edition)
6643                                                 104 mins
6709    90 minutes (theatrical) [1] 91 minutes (unrate...
7057    108 minutes (Original cut) 98 minutes (UK cut)...
7075                 Variable; 90 minutes for default path
Name: Running time, Length: 366, dtype: object
```

In [97]:
```
running_time.str.contains(r'^\d*\s*m', flags=re.IGNORECASE).sum()
```

Out[97]:
```
6877
```

In [98]:
```
running_time[running_time.str.contains(r'^\d*\s*m', flags=re.IGNORECASE) != True]
```

Out[98]:
```
668                     UK:84 min (DVD version) US:86 min
727                         78-102 min (depending on cut)
840                       Varies (79 [3] –84 [1] minutes)
1347                                              25 : 03
1443    United States: 77 minutes Argentina: 94 minute...
1499                                             1hr 35min
1551                                                varies
1774                    Netherlands:96 min, Canada:95 min
1777                                        approx. 14 min
2273                                            1 h 43 min
2993                                                1h 48m
3925                                               4 hours
4425    US domestic version: 86 minutes Original versi...
4967    Theatrical cut: 97 minutes Unrated cut: 107 mi...
5424                    115 [1] /123 [2] /128 [3] minutes
5447                                    1 hour 32 minutes
7075                 Variable; 90 minutes for default path
Name: Running time, dtype: object
```

In [99]:
```
running_time_extract = running_time.str.extract(r'(\d+)\s*ho?u?r?s?\s*(\d*)|(\d+)
```

In [100]:
```
running_time_extract = running_time_extract.apply(lambda col: pd.to_numeric(col,
```

In [101]:
```
wiki_movies_df['running_time'] = running_time_extract.apply(lambda row: row[0]*60
```

In [102]:
```
wiki_movies_df.drop('Running time', axis=1, inplace=True)
```

Module 8.3.12 - Clean the Kaggle Data

In [103]:
```
# Initial Look at the Movie Metadata
```

In [104]:
```
kaggle_metadata.dtypes
```

Out[104]:
```
adult                     object
belongs_to_collection     object
budget                    object
genres                    object
homepage                  object
id                        object
imdb_id                   object
original_language         object
original_title            object
overview                  object
popularity                object
poster_path               object
production_companies      object
production_countries      object
release_date              object
revenue                  float64
runtime                  float64
spoken_languages          object
status                    object
tagline                   object
title                     object
video                     object
vote_average             float64
vote_count               float64
dtype: object
```

In [105]:
```
kaggle_metadata['adult'].value_counts()
```

Out[105]:
```
False
45454
True
9
 Avalanche Sharks tells the story of a bikini contest that turns into a horrify
ing affair when it is hit by a shark avalanche.        1
 - Written by Ørnås
1
 Rune Balot goes to a casino connected to the October corporation to try to wra
p up her case once and for all.                        1
Name: adult, dtype: int64
```

In [106]:
```
# Remove Bad Data
```

In [107]:
```
kaggle_metadata[~kaggle_metadata['adult'].isin(['True','False'])]
```

Out[107]:

	adult	belongs_to_collection	budget	genres	home
19730	- Written by Ørnås	0.065736	/ff9qCepilowshEtG2GYWwzt2bs4.jpg	[{'name': 'Carousel Productions', 'id': 11176}...	[{'iso_316 'CA', 'na 'Cana {
29503	Rune Balot goes to a casino connected to the ...	1.931659	/zV8bHuSL6WXoD6FWogP9j4x80bL.jpg	[{'name': 'Aniplex', 'id': 2883}, {'name': 'Go...	[{'iso_316 'US', 'na 'United S
35587	Avalanche Sharks tells the story of a bikini ...	2.185485	/zaSf5OG7V8X8gqFvly88zDdRm46.jpg	[{'name': 'Odyssey Media', 'id': 17161}, {'nam...	[{'iso_316 'CA', 'na 'Cana

3 rows × 24 columns

In [108]:
```
kaggle_metadata = kaggle_metadata[kaggle_metadata['adult'] == 'False'].drop('adul
```

In [109]:
```
kaggle_metadata['video'].value_counts()
```

Out[109]:
```
False    45358
True        93
Name: video, dtype: int64
```

In [110]:
```
# Convert Data Types
```

In [111]:
```
kaggle_metadata['video'] == 'True'
```

Out[111]:
```
0        False
1        False
2        False
3        False
4        False
         ...
45461    False
45462    False
45463    False
45464    False
45465    False
Name: video, Length: 45454, dtype: bool
```

In [112]:
```
kaggle_metadata['video'] = kaggle_metadata['video'] == 'True'
```

In [113]:
```
kaggle_metadata['budget'] = kaggle_metadata['budget'].astype(int)
kaggle_metadata['id'] = pd.to_numeric(kaggle_metadata['id'], errors='raise')
kaggle_metadata['popularity'] = pd.to_numeric(kaggle_metadata['popularity'], erro
```

In [114]:
```
kaggle_metadata['release_date'] = pd.to_datetime(kaggle_metadata['release_date'])
```

In [115]:
```
 # Reasonability Checks on Ratings Data
```

In [116]:
```
ratings.info(null_counts=True)
```

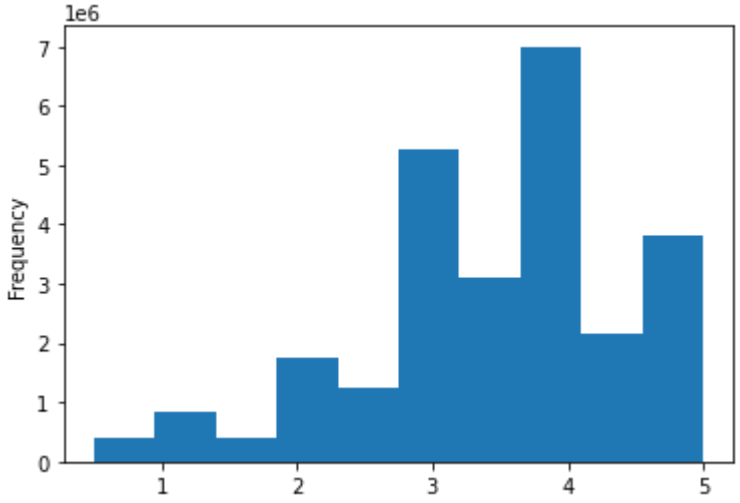
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26024289 entries, 0 to 26024288
Data columns (total 4 columns):
 #   Column     Non-Null Count     Dtype  
---  ------     --------------     -----  
 0   userId     26024289 non-null  int64  
 1   movieId    26024289 non-null  int64  
 2   rating     26024289 non-null  float64
 3   timestamp  26024289 non-null  int64  
dtypes: float64(1), int64(3)
memory usage: 794.2 MB
```

In [117]:
```
pd.to_datetime(ratings['timestamp'], unit='s')
```

Out[117]:
```
0          2015-03-09 22:52:09
1          2015-03-09 23:07:15
2          2015-03-09 22:52:03
3          2015-03-09 22:52:26
4          2015-03-09 22:52:36
                   ...
26024284   2009-10-31 23:26:04
26024285   2009-10-31 23:33:52
26024286   2009-10-31 23:29:24
26024287   2009-11-01 00:06:30
26024288   2009-10-31 23:30:58
Name: timestamp, Length: 26024289, dtype: datetime64[ns]
```

In [118]:
```
pd.options.display.float_format = '{:20,.2f}'.format
ratings['rating'].plot(kind='hist')
ratings['rating'].describe()
```

Out[118]:
```
count          26,024,289.00
mean                    3.53
std                     1.07
min                     0.50
25%                     3.00
50%                     3.50
75%                     4.00
max                     5.00
Name: rating, dtype: float64
```



Module 8.4.1 - Merge Wikipedia and Kaggle Metadata

In [119]:
```
movies_df = pd.merge(wiki_movies_df, kaggle_metadata, on='imdb_id', suffixes=['_w
```

In [120]:
```
# Competing data:
# Wiki                     Movielens                Resolution
#--------------------------------------------------------------------------------
# title_wiki               title_kaggle
# running_time             runtime
# budget_wiki              budget_kaggle
# box_office               revenue
# release_date_wiki        release_date_kaggle
# Language                 original_language
# Production company(s)    production_companies
```

In [121]:
```
# Title
```

In [122]:
```
movies_df[['title_wiki','title_kaggle']]
```

Out[122]:

	title_wiki	title_kaggle
0	The Adventures of Ford Fairlane	The Adventures of Ford Fairlane
1	After Dark, My Sweet	After Dark, My Sweet
2	Air America	Air America
3	Alice	Alice
4	Almost an Angel	Almost an Angel
...	...	...
6047	A Fantastic Woman	A Fantastic Woman
6048	Permission	Permission
6049	Loveless	Loveless
6050	Gemini	Gemini
6051	How to Talk to Girls at Parties	How to Talk to Girls at Parties

6052 rows × 2 columns

In [123]:
```
movies_df[movies_df['title_wiki'] != movies_df['title_kaggle']][['title_wiki','ti
```

Out[123]:

	title_wiki	title_kaggle
27	China Cry	China Cry: A True Story
36	Daddy's Dyin' ...Who's Got the Will?	Daddy's Dyin'... Who's Got the Will?
38	The Dark Side of the Moon	The Dark Side of The Moon
42	Delta Force 2	Delta Force 2: The Colombian Connection
48	DuckTales the Movie:Treasure of the Lost Lamp	DuckTales: The Movie - Treasure of the Lost Lamp
...	...	...
5956	Chips	CHiPS
5971	Spark	Spark: A Space Tail
5994	Pirates of the Caribbean:Dead Men Tell No Tales	Pirates of the Caribbean: Dead Men Tell No Tales
6023	Valerian and the Cityof a Thousand Planets	Valerian and the City of a Thousand Planets
6028	An Inconvenient Sequel:Truth to Power	An Inconvenient Sequel: Truth to Power

438 rows × 2 columns

In [124]:
```
# Show any rows where title_kaggle is empty
movies_df[(movies_df['title_kaggle'] == '') | (movies_df['title_kaggle'].isnull()
```

Out[124]:

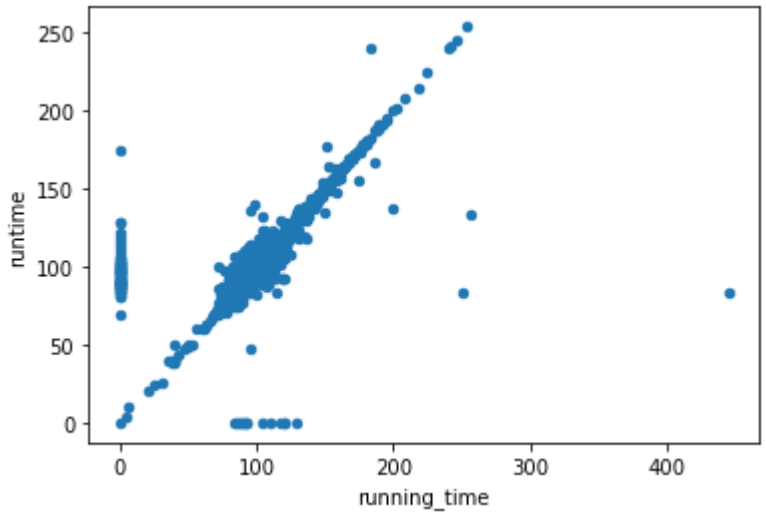
url	year	imdb_link	title_wiki	Based on	Starring	Cinematography	Release date	Country	Language	.

0 rows × 44 columns

In [125]:
```
# Runtime
```

In [126]:
```
movies_df.fillna(0).plot(x='running_time', y='runtime', kind='scatter')
```

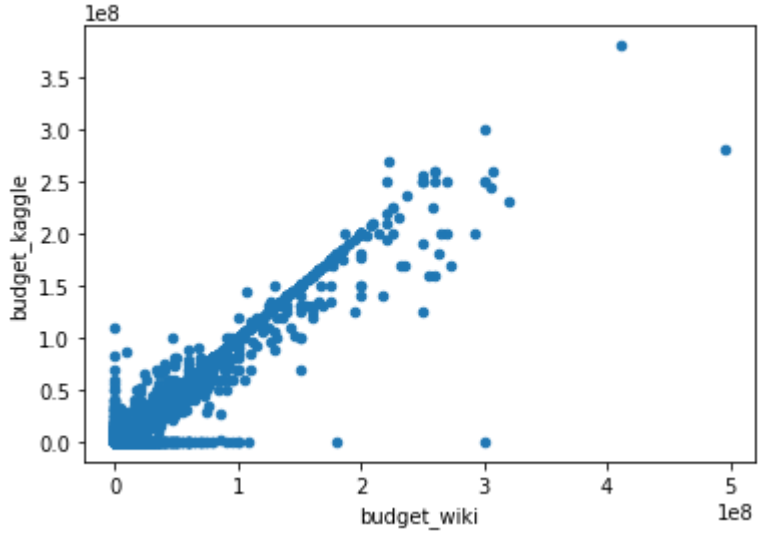
Out[126]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d60534d088>
```



In [127]:
```
# Budget
```

In [128]:
```
movies_df.fillna(0).plot(x='budget_wiki',y='budget_kaggle', kind='scatter')
```

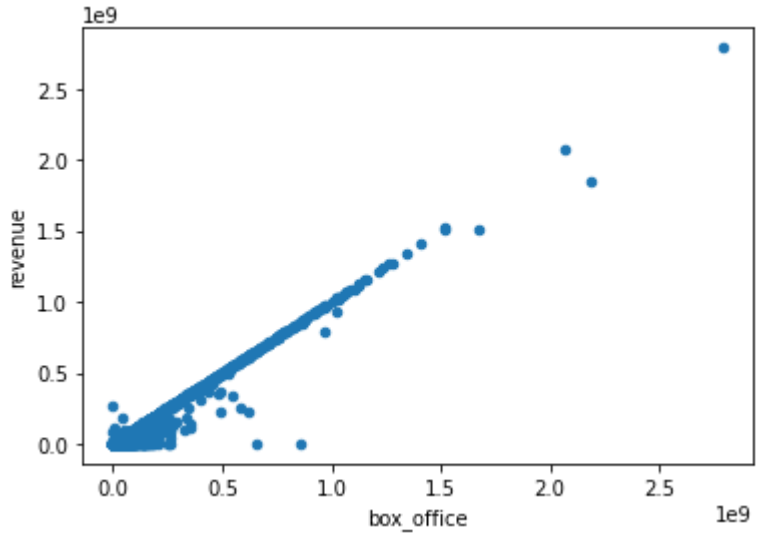
Out[128]: <matplotlib.axes._subplots.AxesSubplot at 0x1d60a83b948>



In [129]:
```
# Box Office
```

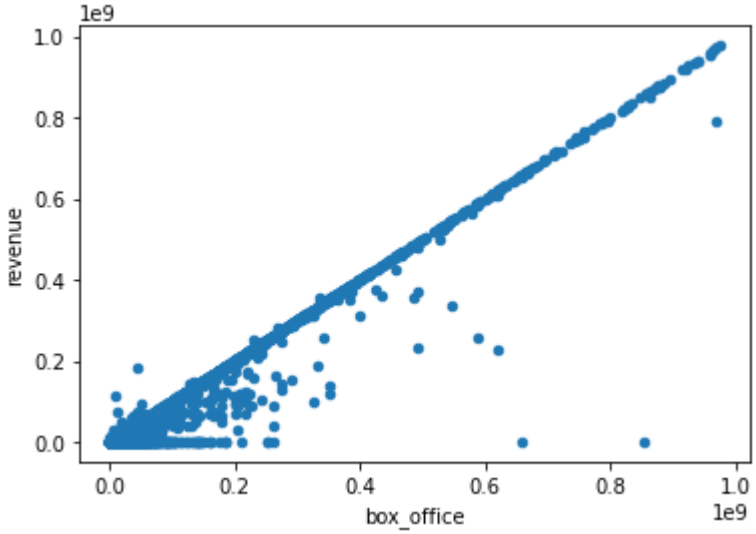
In [130]:
```
movies_df.fillna(0).plot(x='box_office', y='revenue', kind='scatter')
```

Out[130]: <matplotlib.axes._subplots.AxesSubplot at 0x1d60b1c8788>



In [131]:
```
movies_df.fillna(0)[movies_df['box_office'] < 10**9].plot(x='box_office', y='reve
```

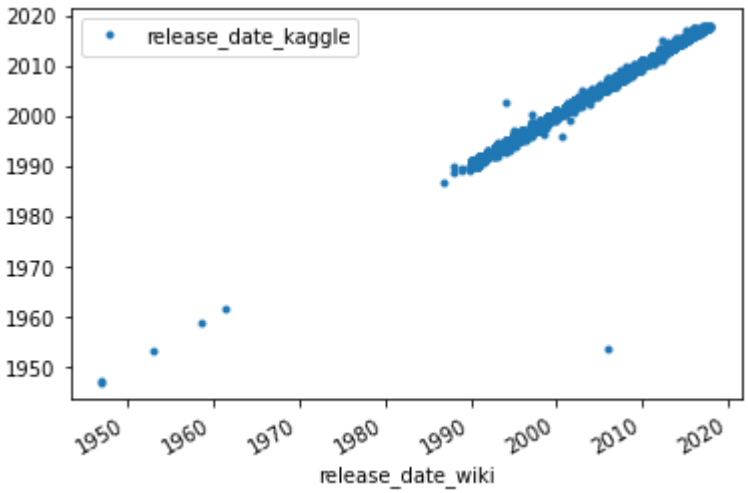
Out[131]: <matplotlib.axes._subplots.AxesSubplot at 0x1d60697afc8>



In [132]:
```
# Release Date
```

In [133]:
```
movies_df[['release_date_wiki','release_date_kaggle']].plot(x='release_date_wiki'
```

Out[133]: <matplotlib.axes._subplots.AxesSubplot at 0x1d6068fc348>



In [134]:
```
movies_df[(movies_df['release_date_wiki'] > '1996-01-01') & (movies_df['release_d
```

Out[134]:

	url	year	imdb_link	title_wiki	Base d
3607	https://en.wikipedia.org/wiki/The_Holiday	2006	https://www.imdb.com/title/tt00457939/	The Holiday	Na

1 rows × 44 columns

In [135]:
```
movies_df[(movies_df['release_date_wiki'] > '1996-01-01') & (movies_df['release_d
```

Out[135]:
```
Int64Index([3607], dtype='int64')
```

In [136]:
```
movies_df = movies_df.drop(movies_df[(movies_df['release_date_wiki'] > '1996-01-0
```

In [137]:
```
movies_df[movies_df['release_date_wiki'].isnull()]
```

Out[137]:

	url	year	imdb_link	title_wiki
1008	https://en.wikipedia.org/wiki/Black_Scorpion_(...	1995	https://www.imdb.com/title/tt0112519/	Black Scorpion
1061	https://en.wikipedia.org/wiki/Flirt_(1995_film)	1995	https://www.imdb.com/title/tt0113080/	Flirt
1121	https://en.wikipedia.org/wiki/Let_It_Be_Me_(19...	1995	https://www.imdb.com/title/tt0113638/	Let It Be Me
1564	https://en.wikipedia.org/wiki/A_Brooklyn_State...	1997	https://www.imdb.com/title/tt0118782/	A Brooklyn State of Mind
1633	https://en.wikipedia.org/wiki/Highball_(film)	1997	https://www.imdb.com/title/tt0119291/	Highball
1775	https://en.wikipedia.org/wiki/Velocity_Trap	1997	https://www.imdb.com/title/tt0120435/	Velocity Trap
2386	https://en.wikipedia.org/wiki/The_Visit_(2000_...	2000	https://www.imdb.com/title/tt0199129/	The Visit
2786	https://en.wikipedia.org/wiki/Stevie_(2002_film)	2002	https://www.imdb.com/title/tt0334416/	Stevie
3174	https://en.wikipedia.org/wiki/Return_to_Sender...	2004	https://www.imdb.com/title/tt0396190/	Return to Sender
3651	https://en.wikipedia.org/wiki/Live_Free_or_Die...	2006	https://www.imdb.com/title/tt0432318/	Live Free or Die
4967	https://en.wikipedia.org/wiki/For_the_Love_of_...	2012	https://www.imdb.com/title/tt1730294/	For the Love of Money

11 rows × 44 columns

In [138]:
```
# Language
```

In [139]:
```
movies_df['Language'].value_counts()
```

Out[139]:
```
English                                 5479
[English, Spanish]                        68
[English, French]                         35
[English, Japanese]                       25
[English, Russian]                        23
                                        ...
Yucatec Mayan                              1
[Aramaic, Latin, Hebrew]                   1
[Spanish, Quechua]                         1
[English, Scottish Gaelic]                 1
[Albanian, English, French, Spanish]       1
Name: Language, Length: 197, dtype: int64
```

In [140]:
```
# TypeError: unhashable type: 'list'
```

In [141]:
```
movies_df['Language'].apply(lambda x: tuple(x) if type(x) == list else x).value_c
```

Out[141]:
```
English                             5479
NaN                                  134
(English, Spanish)                    68
(English, French)                     35
(English, Japanese)                   25
                                    ...
(English, Yiddish)                     1
(English, French, Kinyarwanda)         1
(English, Italian, Swedish)            1
(English, German, Tibetan)             1
(Chinese, English)                     1
Name: Language, Length: 198, dtype: int64
```

In [142]:
```
movies_df['original_language'].value_counts(dropna=False)
```

Out[142]:
```
en    5987
fr      16
es      10
it       8
de       6
pt       4
ja       4
zh       4
hi       2
da       2
ab       1
ko       1
ar       1
sv       1
tr       1
cn       1
he       1
ru       1
Name: original_language, dtype: int64
```

In [143]:
```
# Production Companies
```

In [144]:
```
movies_df[['Production company(s)','production_companies']]
```

Out[144]:

	Production company(s)	production_companies
0	Silver Pictures	[{'name': 'Twentieth Century Fox Film Corporat...
1	Avenue Pictures	[{'name': 'Avenue Pictures Productions', 'id':...
2	[Carolco Pictures, IndieProd Company]	[{'name': 'IndieProd Company Productions', 'id...
3	NaN	[{'name': 'Orion Pictures', 'id': 41}]
4	NaN	[{'name': 'Paramount Pictures', 'id': 4}]
...	...	...
6047	[Fabula, Komplizen Film]	[{'name': 'Komplizen Film', 'id': 1618}, {'nam...
6048	Ball & Chain Productions	[{'name': 'Ball & Chain Productions', 'id': 74...
6049	[Arte France Cinéma, Why Not Productions]	[{'name': 'ARTE France Cinéma', 'id': 94}, {'n...
6050	[Film Science, Rough House Pictures, Syncopate...	[{'name': 'Film Science', 'id': 1976}, {'name'...
6051	[HanWay Films, Little Punk, See-Saw Films]	[{'name': 'HanWay Films', 'id': 2395}, {'name'...

6051 rows × 2 columns

In [145]:
```
# Put It All Together
```

In [146]:
```
movies_df.drop(columns=['title_wiki','release_date_wiki','Language','Production c
```

In [147]:
```
def fill_missing_kaggle_data(df, kaggle_column, wiki_column):
    df[kaggle_column] = df.apply(
        lambda row: row[wiki_column] if row[kaggle_column] == 0 else row[kaggle_c
        , axis=1)
    df.drop(columns=wiki_column, inplace=True)
```

In [148]:
```
fill_missing_kaggle_data(movies_df, 'runtime', 'running_time')
fill_missing_kaggle_data(movies_df, 'budget_kaggle', 'budget_wiki')
fill_missing_kaggle_data(movies_df, 'revenue', 'box_office')
movies_df
```

Out[148]:

	url	year	imdb_link	Based
0	https://en.wikipedia.org/wiki/The_Adventures_o...	1990	https://www.imdb.com/title/tt0098987/	[Characte by R Wein
1	https://en.wikipedia.org/wiki/After_Dark,_My_S...	1990	https://www.imdb.com/title/tt0098994/	[the nov After Da My Swe by, J Thon
2	https://en.wikipedia.org/wiki/Air_America_(film)	1990	https://www.imdb.com/title/tt0099005/	[America, Christoph Robbi
3	https://en.wikipedia.org/wiki/Alice_(1990_film)	1990	https://www.imdb.com/title/tt0099012/	N
4	https://en.wikipedia.org/wiki/Almost_an_Angel	1990	https://www.imdb.com/title/tt0099018/	N
...	...	...	...	
6047	https://en.wikipedia.org/wiki/A_Fantastic_Woman	2018	https://www.imdb.com/title/tt5639354/	N
6048	https://en.wikipedia.org/wiki/Permission_(film)	2018	https://www.imdb.com/title/tt5390066/	N
6049	https://en.wikipedia.org/wiki/Loveless_(film)	2018	https://www.imdb.com/title/tt6304162/	N

	url	year	imdb_link	Based
6050	https://en.wikipedia.org/wiki/Gemini_(2017_film)	2018	https://www.imdb.com/title/tt5795086/	N
6051	https://en.wikipedia.org/wiki/How_to_Talk_to_G...	2018	https://www.imdb.com/title/tt3859310/	[", How Talk to G at Parties by, Ne

6051 rows × 37 columns

In [149]:
```
for col in movies_df.columns:
    lists_to_tuples = lambda x: tuple(x) if type(x) == list else x
    value_counts = movies_df[col].apply(lists_to_tuples).value_counts(dropna=Fals
    num_values = len(value_counts)
    if num_values == 1:
        print(col)
```

```
video
```

In [150]:
```
movies_df['video'].value_counts(dropna=False)
```

Out[150]:
```
False    6051
Name: video, dtype: int64
```

In [151]:
```
movies_df = movies_df.loc[:, ['imdb_id','id','title_kaggle','original_title','tag
                       'runtime','budget_kaggle','revenue','release_date_kaggle',
                       'genres','original_language','overview','spoken_languages'
                       'production_companies','production_countries','Distributor
                       'Producer(s)','Director','Starring','Cinematography','Edit
                      ]]
```

In [152]:
```
movies_df.rename({'id':'kaggle_id',
                  'title_kaggle':'title',
                  'url':'wikipedia_url',
                  'budget_kaggle':'budget',
                  'release_date_kaggle':'release_date',
                  'Country':'country',
                  'Distributor':'distributor',
                  'Producer(s)':'producers',
                  'Director':'director',
                  'Starring':'starring',
                  'Cinematography':'cinematography',
                  'Editor(s)':'editors',
                  'Writer(s)':'writers',
                  'Composer(s)':'composers',
                  'Based on':'based_on'
                 }, axis='columns', inplace=True)
```

Module 8.4.2 - Transform and Merge Rating Data

In [153]:
```
rating_counts = ratings.groupby(['movieId','rating'], as_index=False).count()
```

In [154]:
```
rating_counts = ratings.groupby(['movieId','rating'], as_index=False).count() \
                .rename({'userId':'count'}, axis=1)
```

In [155]:
```
rating_counts = ratings.groupby(['movieId','rating'], as_index=False).count() \
                .rename({'userId':'count'}, axis=1) \
                .pivot(index='movieId',columns='rating', values='count')
```

In [156]:
```
rating_counts.columns = ['rating_' + str(col) for col in rating_counts.columns]
```

In [157]:
```
movies_with_ratings_df = pd.merge(movies_df, rating_counts, left_on='kaggle_id',
```

In [158]:
```
movies_with_ratings_df[rating_counts.columns] = movies_with_ratings_df[rating_cou
```

Module 8.5.1 - Connect Pandas and SQL

In [159]:
```
# Import Modules
```

In [160]:
```
from sqlalchemy import create_engine
```

In [161]:
```
# Create the Database Engine
```

In [162]:
```
# "postgres://[user]:[password]@[location]:[port]/[database]"
```

In [163]:
```
from config import db_password
```

In [164]:
```
db_string = f"postgres://postgres:{db_password}@127.0.0.1:5432/movie_data"
```

In [165]:
```
!pip install psycopg2
```

```
Requirement already satisfied: psycopg2 in c:\users\emilio\anaconda3\envs\pytho
ndata\lib\site-packages (2.8.6)
```

In [169]:
```
engine = create_engine(db_string)
```

In [170]:
```
# Import the Movie Data
```

In [ ]:
```
movies_df.to_sql(name='movies', con=engine)
```

In [172]:
```
# Import the Ratings Data
```

In [173]:
```
# Do not run this yet!
# for data in pd.read_csv(f'{file_dir}ratings.csv', chunksize=1000000):
#     data.to_sql(name='ratings', con=engine, if_exists='append')
```

In [174]:
```
# Step 1: Print Number of Imported Rows


```

In [175]:
```
# create a variable for the number of rows imported

# for data in pd.read_csv(f'{file_dir}ratings.csv', chunksize=1000000):

    # print out the range of rows that are being imported

#     data.to_sql(name='ratings', con=engine, if_exists='append')

    # increment the number of rows imported by the chunksize

    # print that the rows have finished importing
```

In [178]:
```
# create a variable for the number of rows imported
rows_imported = 0
for data in pd.read_csv(f'{file_dir}/ratings.csv', chunksize=1000000):

    # print out the range of rows that are being imported
    print(f'importing rows {rows_imported} to {rows_imported + len(data)}...', er

    data.to_sql(name='ratings', con=engine, if_exists='append')

    # increment the number of rows imported by the size of 'data'
    rows_imported += len(data)

    # print that the rows have finished importing
    print('Done.')
```

```
importing rows 0 to 1000000...Done.
importing rows 1000000 to 2000000...Done.
importing rows 2000000 to 3000000...Done.
importing rows 3000000 to 4000000...Done.
importing rows 4000000 to 5000000...Done.
importing rows 5000000 to 6000000...Done.
importing rows 6000000 to 7000000...Done.
importing rows 7000000 to 8000000...Done.
importing rows 8000000 to 9000000...Done.
importing rows 9000000 to 10000000...Done.
importing rows 10000000 to 11000000...Done.
importing rows 11000000 to 12000000...Done.
importing rows 12000000 to 13000000...Done.
importing rows 13000000 to 14000000...Done.
importing rows 14000000 to 15000000...Done.
importing rows 15000000 to 16000000...Done.
importing rows 16000000 to 17000000...Done.
importing rows 17000000 to 18000000...Done.
importing rows 18000000 to 19000000...Done.
importing rows 19000000 to 20000000...Done.
importing rows 20000000 to 21000000...Done.
importing rows 21000000 to 22000000...Done.
importing rows 22000000 to 23000000...Done.
importing rows 23000000 to 24000000...Done.
importing rows 24000000 to 25000000...Done.
importing rows 25000000 to 26000000...Done.
importing rows 26000000 to 26024289...Done.
```

In [179]:
```
import time
```

In [180]:
```
# Step 2: Print Elapsed Time
```

In [181]:
```
# get the start_time from time.time()
```

In [182]:
```
rows_imported = 0
# get the start_time from time.time()
start_time = time.time()
for data in pd.read_csv(f'{file_dir}/ratings.csv', chunksize=1000000):
    print(f'importing rows {rows_imported} to {rows_imported + len(data)}...', en
    data.to_sql(name='ratings', con=engine, if_exists='append')
    rows_imported += len(data)

    # add elapsed time to final print out
    print(f'Done. {time.time() - start_time} total seconds elapsed')
```

```
importing rows 0 to 1000000...Done. 279.7709655761719 total seconds elapsed
importing rows 1000000 to 2000000...Done. 558.9502174854279 total seconds ela
psed
importing rows 2000000 to 3000000...Done. 838.1660115718842 total seconds ela
psed
importing rows 3000000 to 4000000...Done. 1118.3159835338593 total seconds el
apsed
importing rows 4000000 to 5000000...Done. 1396.1269843578339 total seconds el
apsed
importing rows 5000000 to 6000000...Done. 1675.5041906833649 total seconds el
apsed
importing rows 6000000 to 7000000...Done. 1955.1149377822876 total seconds el
apsed
importing rows 7000000 to 8000000...Done. 2234.8241362571716 total seconds el
apsed
importing rows 8000000 to 9000000...Done. 2504.6730632781982 total seconds el
apsed
importing rows 9000000 to 10000000...Done. 2786.339669942856 total seconds el
apsed
importing rows 10000000 to 11000000...Done. 3067.3560807704926 total seconds
elapsed
importing rows 11000000 to 12000000...Done. 3343.399118423462 total seconds e
lapsed
importing rows 12000000 to 13000000...Done. 3623.635448694229 total seconds e
lapsed
importing rows 13000000 to 14000000...Done. 3896.3703632354736 total seconds
elapsed
importing rows 14000000 to 15000000...Done. 4176.3099637031555 total seconds
elapsed
importing rows 15000000 to 16000000...Done. 4459.1170127391815 total seconds
elapsed
importing rows 16000000 to 17000000...Done. 4739.811790943146 total seconds e
lapsed
importing rows 17000000 to 18000000...Done. 5018.869275569916 total seconds e
lapsed
importing rows 18000000 to 19000000...Done. 5298.579102516174 total seconds e
lapsed
importing rows 19000000 to 20000000...Done. 5570.119524478912 total seconds e
lapsed
importing rows 20000000 to 21000000...Done. 5833.210102558136 total seconds e
lapsed
importing rows 21000000 to 22000000...Done. 6115.7309238910675 total seconds
elapsed
importing rows 22000000 to 23000000...Done. 6386.627951383591 total seconds e
lapsed
```

```
importing rows 23000000 to 24000000...Done. 6662.576581954956 total seconds e
lapsed
importing rows 24000000 to 25000000...Done. 6939.167383670807 total seconds e
lapsed
importing rows 25000000 to 26000000...Done. 7217.398390293121 total seconds e
lapsed
importing rows 26000000 to 26024289...Done. 7224.33641242981 total seconds el
apsed
```

In [ ]: