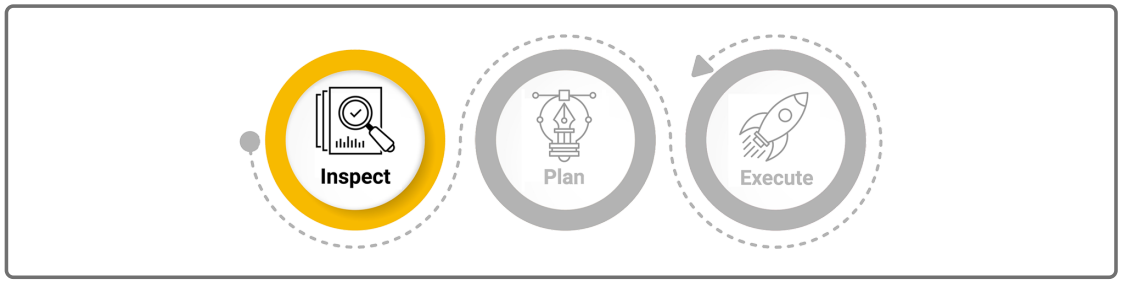
8.4.1 Merge Wikipedia and Kaggle Metadata

Now that the Wikipedia data and Kaggle data are cleaned up and in tabular formats with the right data types for each column, Britta can join them together. However, after they're joined, the data still needs to be cleaned up a bit, especially where Kaggle and Wikipedia data overlap.

With all the tables cleaned up, we're ready to merge them by IMDb ID.

Retake

One of the things we always want to look out for after we've merged data is redundant columns.

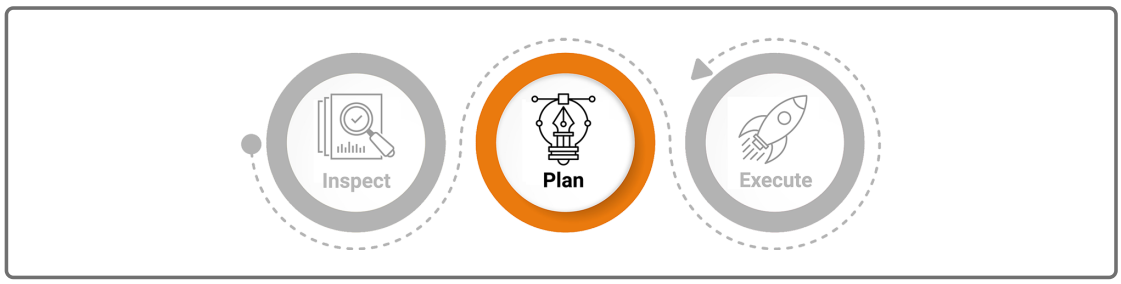


Print out a list of the columns so we can identify which ones are redundant. We'll use the `suffixes` parameter to make it easier to identify which table each column came from. Here's what your code should look like:

```
movies_df = pd.merge(wiki_movies_df, kaggle_metadata, on='imdb_id', suffixes
```

There are seven pairs of columns that have redundant information. We'll look at each pair of columns and decide how to handle the data.

There are a few options when dealing with redundant data. We'll consider two. The simplest is to just drop one of the competing columns, but sometimes that means a loss of good information. Sometimes, one column will have data where the other has missing data, and vice versa. In that case, we'd want the other option: fill in the gaps using both columns.



Below is the list of competing columns. We'll fill in the resolution to each pair as we go along. We'll hold off on implementing the resolutions until we make a decision for each pair because if we did, we might inadvertently remove data that could be helpful in making a later decision.

Wikipedia	**Kaggle**	**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	

You may find it helpful to include a table like this in your Jupyter Notebook that documents the decisions made and the justifications for them. Unfortunately, markdown doesn't support formatting tables. One way to work around that is to just write your text down as comments in a code cell.

```
# 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
```

Let's start comparing columns.

Title

First, just take a quick look at some of the titles.

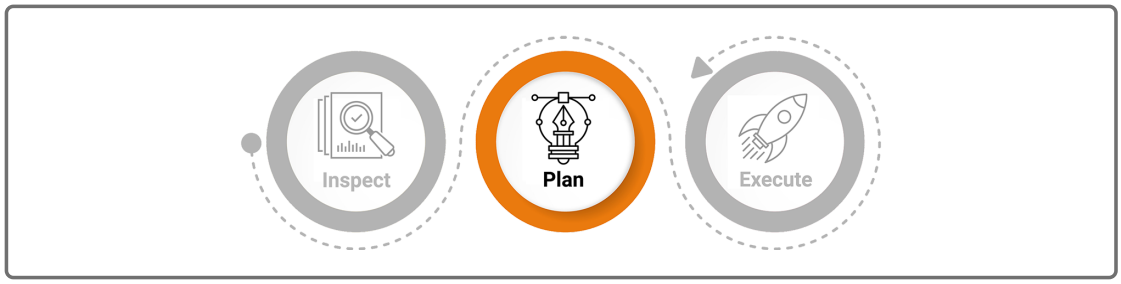
```
movies_df[['title_wiki','title_kaggle']]
```

They both seem pretty consistent, which we'd expect. Look at the rows where the titles don't match.

```
movies_df[movies_df['title_wiki'] != movies_df['title_kaggle']][['title_wiki
```

Both options look pretty good, but the Kaggle data looks just a little bit more consistent. Let's confirm there aren't any missing titles in the Kaggle data with the following code:

```
# Show any rows where title_kaggle is empty
movies_df[(movies_df['title_kaggle'] == '') | (movies_df['title_kaggle'].isn
```

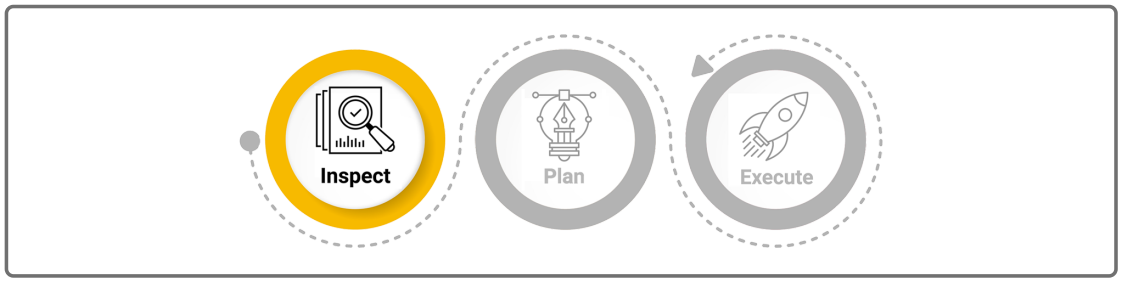


No results were returned, so we can just drop the Wikipedia titles. Note that for now, we're merely noting the resolution.

Wikipedia	**Kaggle**	**Resolution**
title_wiki	title_kaggle	**Drop Wikipedia.**

running_time	runtime	
budget_wiki	budget_kaggle	
box_office	revenue	
release_date_wiki	release_date_kaggle	
Language	original_language	
Production company(s)	production_companies	

Runtime



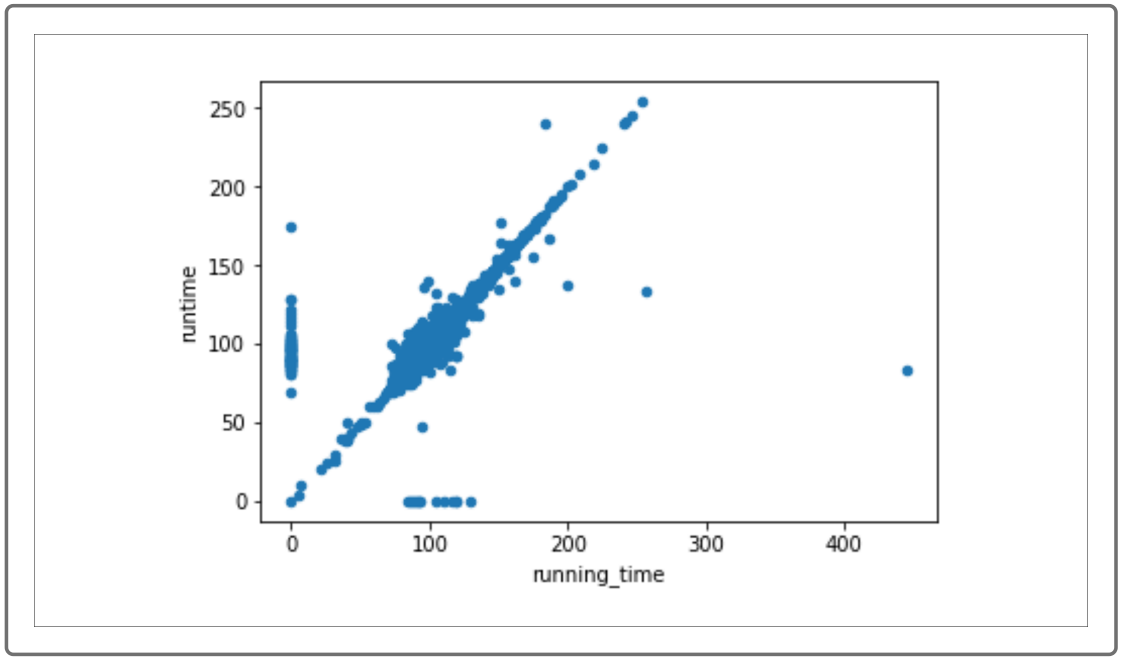
Next, look at `running_time` versus `runtime`. A scatter plot is a great way to give us a sense of how similar the columns are to each other. If the two columns were exactly the same, we'd see a scatter plot of a perfectly straight line. Any wildly different values will show up as dots far from that central line, and if one column is missing data, those values will fall on the x-axis or y-axis.

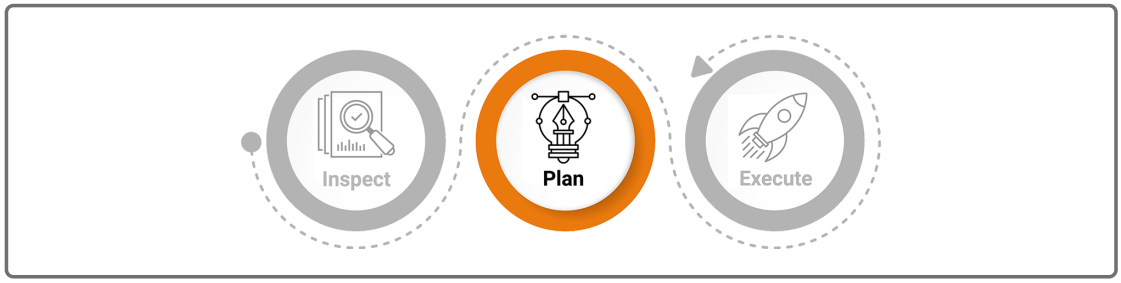
CAUTION

> Because we're dealing with merged data, we should expect there to be missing values. Scatter plots won't show null values, so we need to fill them in with zeros when we're making our plots to get the whole picture.

The following code will fill in missing values with zero and make the scatter plot:

```
movies_df.fillna(0).plot(x='running_time', y='runtime', kind='scatter')
```

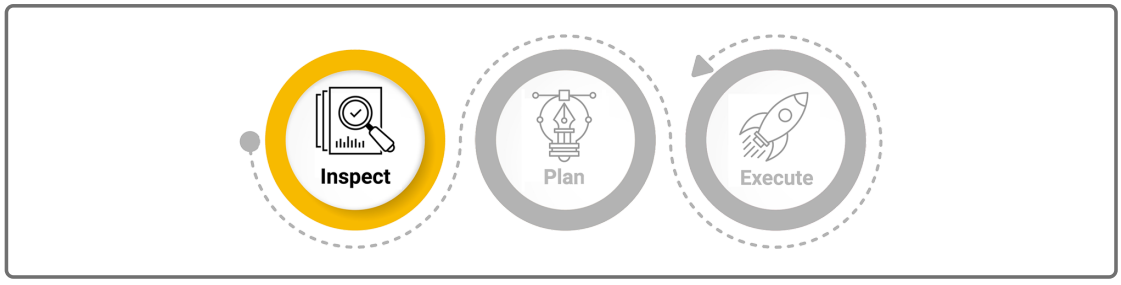




Notice that there are more data points on the origin of the Y axis than on the origin of the X axis. Since the X axis is Wikipedia and the Y axis is Kaggle, this means there are more missing entries in the Wikipedia data set than in the Kaggle data set. Also, most of the runtimes are pretty close to each other but the Wikipedia data has some outliers, so the Kaggle data is probably a better choice here. However, we can also see from the scatter plot that there are movies where Kaggle has 0 for the runtime but Wikipedia has data, so we'll fill in the gaps with Wikipedia data.

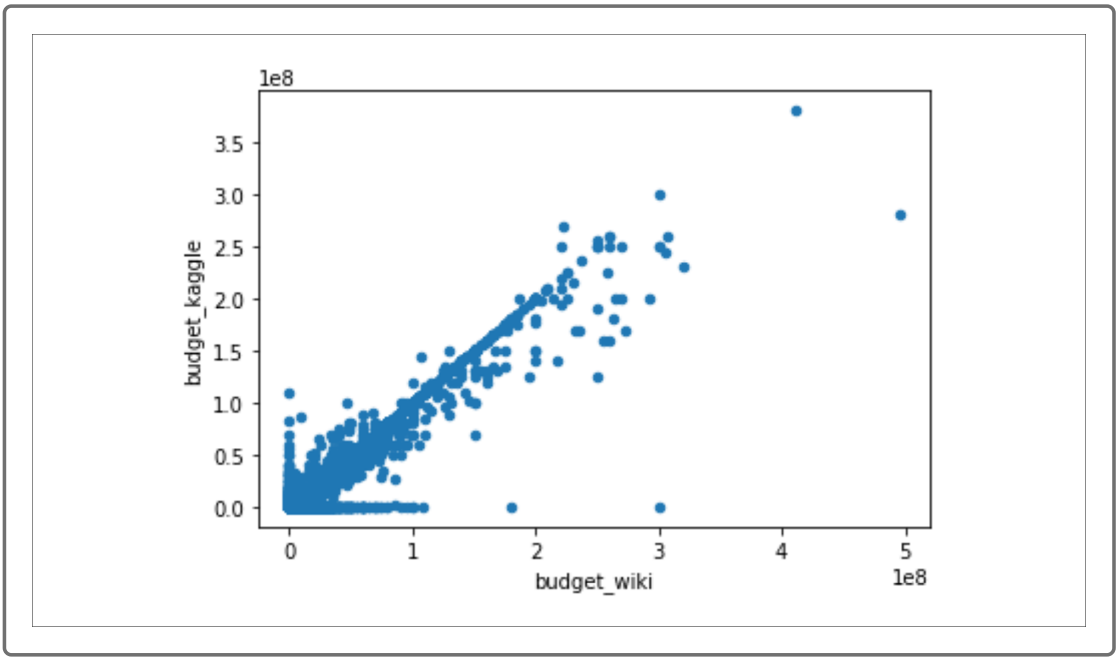
Wikipedia	**Kaggle**	**Resolution**
title_wiki	title_kaggle	Drop Wikipedia.
running_time	runtime	**Keep Kaggle; fill in zeros with Wikipedia data.**
budget_wiki	budget_kaggle	
box_office	revenue	
release_date_wiki	release_date_kaggle	
Language	original_language	
Production company(s)	production_companies	

Budget



Since budget_wiki and budget_kaggle are numeric, we'll make another scatter plot to compare the values:

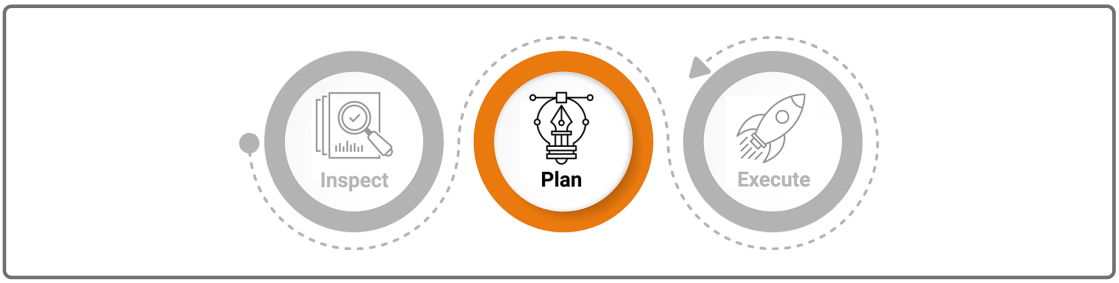
```
movies_df.fillna(0).plot(x='budget_wiki',y='budget_kaggle', kind='scatter')
```



NOTE

Here are some questions to consider when interpreting this scatter plot:

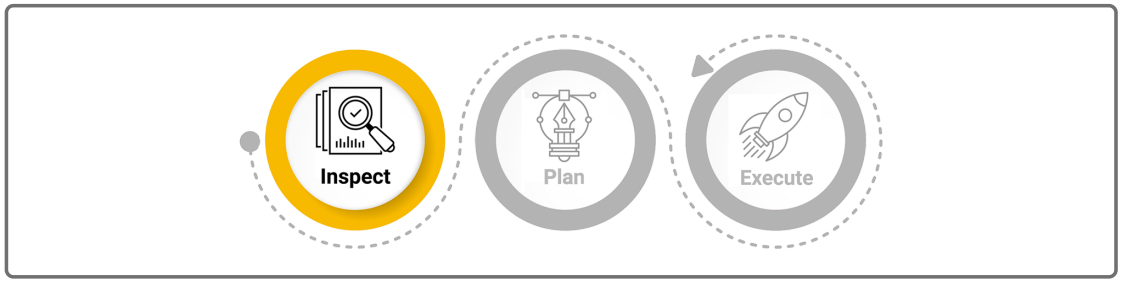
- Which dataset seems to have more outliers?
- Which dataset seems to have more missing data points?
- If we were to fill in the missing data points of one set with the other, which would be more likely to give us consistent data?
- Is it better to start with a base of consistent data and fill in missing points with possible outliers? Or is it better to start with a base of data with outliers and fill in missing points with more consistent data?



The Wikipedia data appears to have more outliers compared to the Kaggle data. However, there are quite a few movies with no data in the Kaggle column, while Wikipedia does have budget data. Therefore, we'll fill in the gaps with Wikipedia's data.

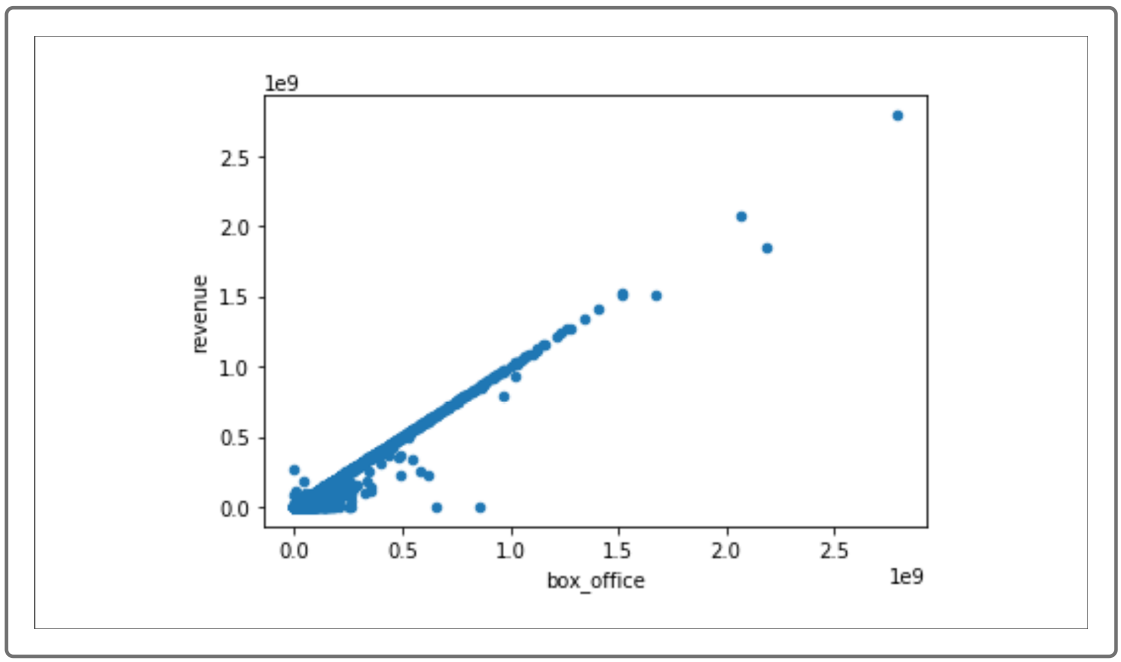
Wikipedia	**Kaggle**	**Resolution**
title_wiki	title_kaggle	Drop Wikipedia.
running_time	runtime	Keep Kaggle; fill in zeros with Wikipedia data.
budget_wiki	budget_kaggle	**Keep Kaggle; fill in zeros with Wikipedia data.**
box_office	revenue	
release_date_wiki	release_date_kaggle	
Language	original_language	
Production company(s)	production_companies	

Box Office



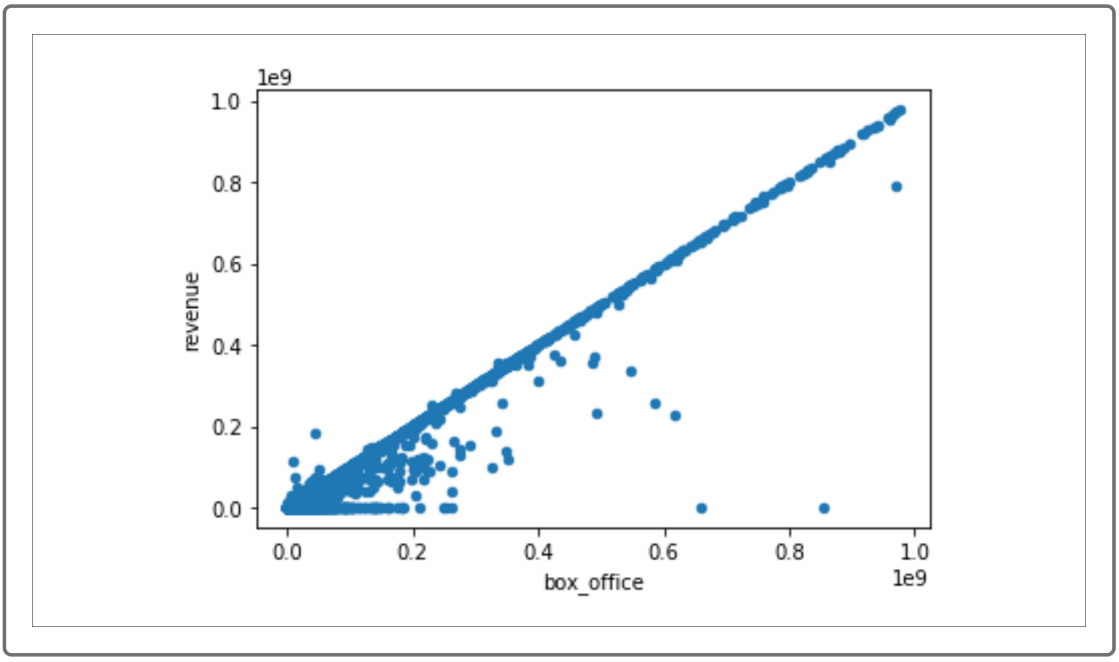
The box_office and revenue columns are numeric, so we'll make another scatter plot.

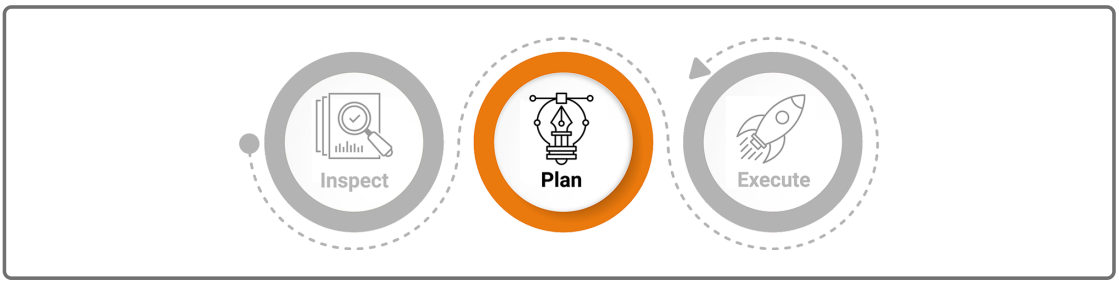
```
movies_df.fillna(0).plot(x='box_office', y='revenue', kind='scatter')
```



That looks pretty close, but we might be getting thrown off by the scale of that large data point. Let's look at the scatter plot for everything less than $1 billion in box_office.

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



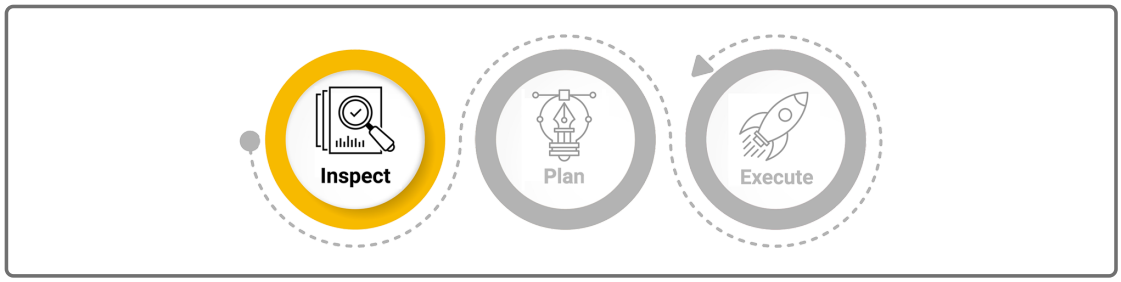


This looks similar to what we've seen for budget, so we'll make the same decision: keep the Kaggle data, but fill in the zeros with Wikipedia data.

Wikipedia	**Kaggle**	**Resolution**
title_wiki	title_kaggle	Drop Wikipedia.
running_time	runtime	Keep Kaggle; fill in zeros with Wikipedia data.
budget_wiki	budget_kaggle	Keep Kaggle; fill in zeros with Wikipedia data.
box_office	revenue	**Keep Kaggle; fill in zeros with Wikipedia data.**
release_date_wiki	release_date_kaggle	

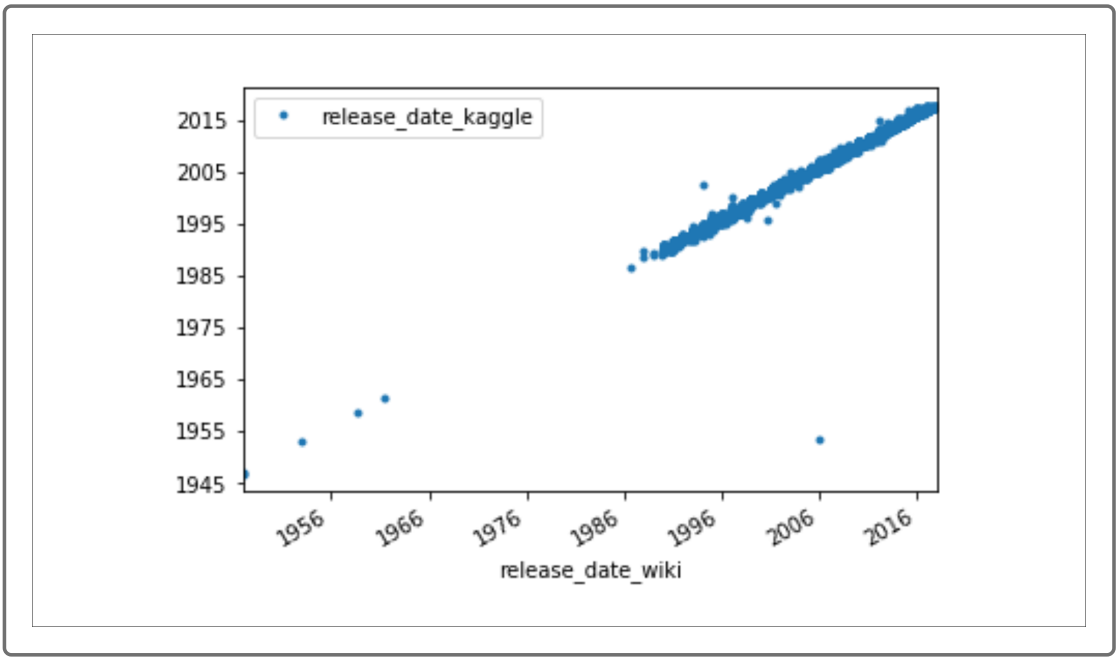
Language	original_language	
Production company(s)	production_companies	

Release Date



For release_date_wiki and release_date_kaggle, we can't directly make a scatter plot, because the scatter plot only works on numeric data. However, there's a tricky workaround that we can use. We'll use the regular line plot (which can plot date data), and change the style to only put dots by adding `style='.'` to the `plot()` method:

```
movies_df[['release_date_wiki','release_date_kaggle']].plot(x='release_date_
```



We should investigate that wild outlier around 2006. We're just going to choose some rough cutoff dates to single out that one movie. We'll look for any movie whose release date according to Wikipedia is after 1996, but whose release date according to Kaggle is before 1965. Here's what your code should look like:

```
movies_df[(movies_df['release_date_wiki'] > '1996-01-01') & (movies_df['rele
```

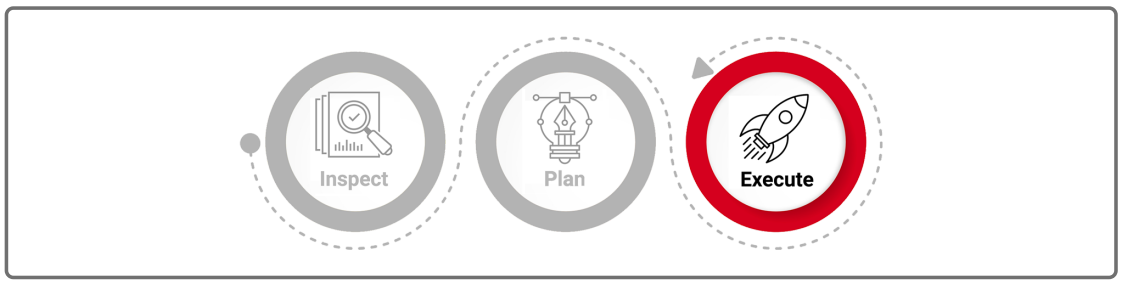
	url	year	imdb_link	title_wiki	Based on	Starring	Cinematography	Release date	Country	Languag
3607	https://en.wikipedia.org/wiki/The_Holiday	2006	https://www.imdb.com/title/tt0457939/	The Holiday	NaN	[Kate Winslet, Cameron Diaz, Jude Law, Jack Bl...	Dean Cundey	[December 8, 2006, (, 2006-12-08,)]	United States	Englis

1 rows × 45 columns

	Country	Language	...	release_date_kaggle	revenue	runtime	spoken_languages	status	tagline	title_kaggle	video	vote_average	vote_count
r -]	United States	English	...	1953-08-28	30500000.0	118.0	[{'iso_639_1': 'en', 'name': 'English'}]	Released	Pouring out of impassioned pages...brawling th...	From Here to Eternity	False	7.2	137.0

Based on the output, it looks like somehow *The Holiday* in the Wikipedia data got merged with *From Here to Eternity*. We'll have to drop that row from our DataFrame. We'll get the index of that row with the following:

```
movies_df[(movies_df['release_date_wiki'] > '1996-01-01') & (movies_df['rele
```



Then we can drop that row like this:

```
movies_df = movies_df.drop(movies_df[(movies_df['release_date_wiki'] > '1996
```

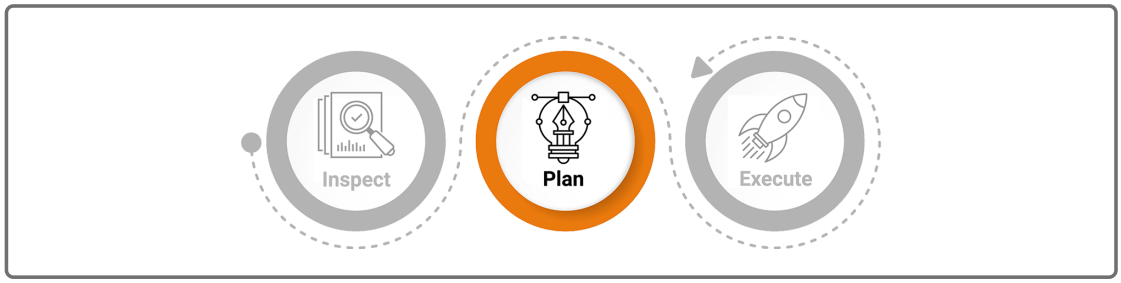
Now, see if there are any null values:

```
movies_df[movies_df['release_date_wiki'].isnull()]
```

The Wikipedia data is missing release dates for 11 movies:

	url	year	imdb_link	title_wiki	Based on	Starring	Cinematography	Release date	Country
1008	https://en.wikipedia.org/wiki/Black_Scorpion_(...	1995	https://www.imdb.com/title/tt0112519/	Black Scorpion	NaN	[Joan Severance, Bruce Abbott, Garrett Morris]	Geoff George	NaN	United States
1061	https://en.wikipedia.org/wiki/Flirt_(1995_film)	1995	https://www.imdb.com/title/tt0113080/	Flirt	NaN	[Bill Sage, Dwight Ewell, Miho Nikaido]	Michael Spiller	NaN	[United States, Germany, Japan, [1]]
1121	https://en.wikipedia.org/wiki/Let_It_Be_Me_(19...	1995	https://www.imdb.com/title/tt0113638/	Let It Be Me	NaN	NaN	Miroslav Ondrícek	NaN	NaN
1564	https://en.wikipedia.org/wiki/A_Brooklyn_State...	1997	https://www.imdb.com/title/tt0118782/	A Brooklyn State of Mind	NaN	NaN	Ken Kelsch	NaN	NaN
1633	https://en.wikipedia.org/wiki/Highball_(film)	1997	https://www.imdb.com/title/tt0119291/	Highball	NaN	[Justine Bateman, Peter Bogdanovich, Chris Eig...	Steven Bernstein	NaN	United States
1775	https://en.wikipedia.org/wiki/Velocity_Trap	1997	https://www.imdb.com/title/tt0120435/	Velocity Trap	NaN	[Olivier Gruner, Alicia Coppola, Ken Olandt]	Philip D. Schwartz	NaN	United States
2386	https://en.wikipedia.org/wiki/The_Visit_(2000_...	2000	https://www.imdb.com/title/tt0199129/	The Visit	NaN	[Hill Harper, Billy Dee Williams, Obba Babatun...	John L. Demps Jr.	NaN	NaN
2786	https://en.wikipedia.org/wiki/Stevie_(2002_film)	2002	https://www.imdb.com/title/tt0334416/	Stevie	NaN	NaN	[Dana Kupper, Gordon Quinn, Peter Gilbert]	NaN	United States
3174	https://en.wikipedia.org/wiki/Return_to_Sender...	2004	https://www.imdb.com/title/tt0396190/	Return to Sender	NaN	[Aidan Quinn, Connie Nielsen, Mark Holton]	NaN	NaN	[Denmark, USA, UK]
3651	https://en.wikipedia.org/wiki/Live_Free_or_Die...	2006	https://www.imdb.com/title/tt0432318/	Live Free or Die	NaN	[Aaron Stanford, Paul Schneider, Ebon Moss-Bac...	NaN	NaN	United States
4967	https://en.wikipedia.org/wiki/For_the_Love_of_...	2012	https://www.imdb.com/title/tt1730294/	For the Love of Money	NaN	[Yehuda Levi, Edward Furlong, James Caan, Jeff...	Andrzej Sekuła	NaN	United States

11 rows × 45 columns

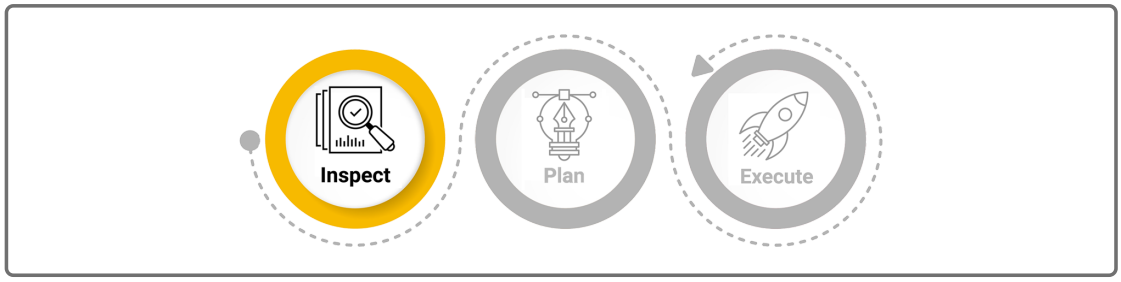


\ But the Kaggle data isn't missing any release dates. In this case, we'll just drop the Wikipedia data.

Wikipedia	**Kaggle**	**Resolution**
title_wiki	title_kaggle	Drop Wikipedia.
running_time	runtime	Keep Kaggle; fill in zeros with Wikipedia data.

budget_wiki	budget_kaggle	Keep Kaggle; fill in zeros with Wikipedia data.
box_office	revenue	Keep Kaggle; fill in zeros with Wikipedia data.
release_date_wiki	release_date_kaggle	**Drop Wikipedia.**
Language	original_language	
Production company(s)	production_companies	

Language



For the language data, we'll compare the value counts of each. However, consider the following code:

```
movies_df['Language'].value_counts()
```

This code throws an error because some of the language data points are stored as lists.

```
TypeError: unhashable type: 'list'
```

NOTE

> We don't need to worry about what hashing is right now, but if you're curious, **hashing** is a clever computer science trick that can be used to speed up algorithms like getting value counts. Hashing converts values, even arbitrarily long strings, to a limited space of numerical values. We'll talk about hashing more when we get to machine learning, but for now, the important part is that Python creates hash values when new objects are created if they are immutable. Since mutable objects can have their values change after being created, the values might change and not match the hash, so Python just refuses.

We need to convert the lists in `Language` to tuples so that the `value_counts()` method will work. See the following code:

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

Here's what the output will look like.

```
English                                                 5479
NaN                                                      134
(English, Spanish)                                        68
(English, French)                                         35
(English, Japanese)                                       25
                                                         ...
(English, German, Russian, Ukrainian)                      1
(English, Lao)                                             1
(English, Italian, Swedish)                                1
(English, Afrikaans, German)                               1
(English, French, Hebrew, Spanish, Arabic, Italian)        1
Name: Language, Length: 198, dtype: int64
```

For the Kaggle data, there are no lists, so we can just run `value_counts()` on it.

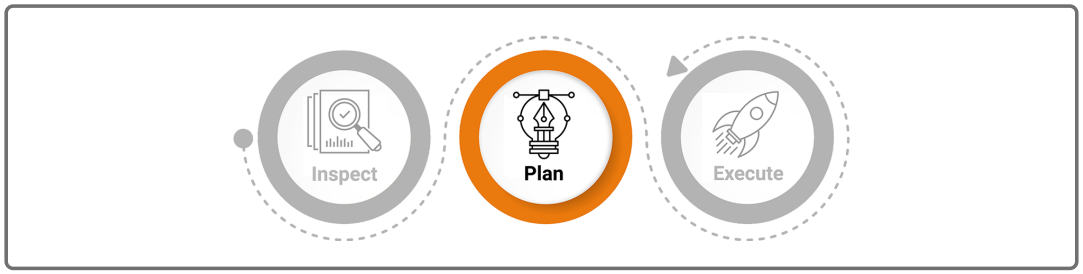
```
movies_df['original_language'].value_counts(dropna=False)
```

Here's what the output will look like.

```
en    5982
fr      16
es       9
it       8
de       6
zh       4
ja       4
pt       4
da       2
hi       2
tr       1
cn       1
ar       1
ab       1
sv       1
ko       1
he       1
Name: original_language, dtype: int64
```

NOTE

There's a trade-off here between the Wikipedia language data and the Kaggle language data. While the Wikipedia data has more information about multiple languages, the Kaggle data is already in a consistent and usable format. Parsing the Wikipedia data may create too many difficulties to make it worthwhile, though.

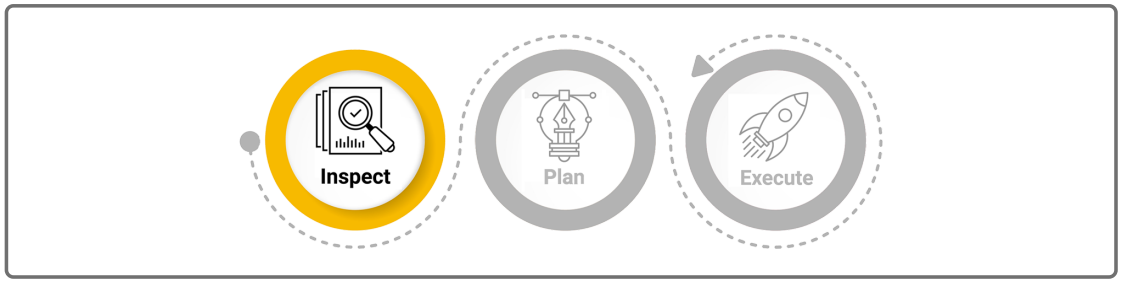


> This is another judgment call; there's no clear-cut answer here. However, for better or for worse, decisions that save time are usually the ones that win, so we'll use the Kaggle data here.

Here's our updated plan:

Wikipedia	**Kaggle**	**Resolution**
title_wiki	title_kaggle	Drop Wikipedia.
running_time	runtime	Keep Kaggle; fill in zeros with Wikipedia data.
budget_wiki	budget_kaggle	Keep Kaggle; fill in zeros with Wikipedia data.
box_office	revenue	Keep Kaggle; fill in zeros with Wikipedia data.
release_date_wiki	release_date_kaggle	Drop Wikipedia.
Language	original_language	**Drop Wikipedia.**
Production company(s)	production_companies	

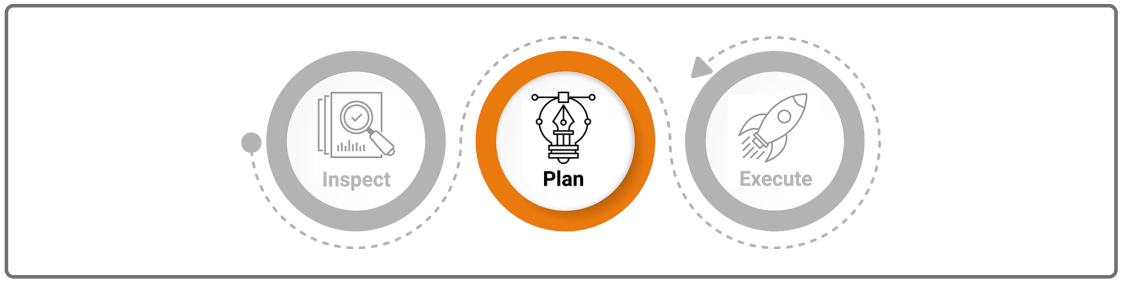
Production Companies



Again, we'll start off just taking a look at a small number of samples.

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

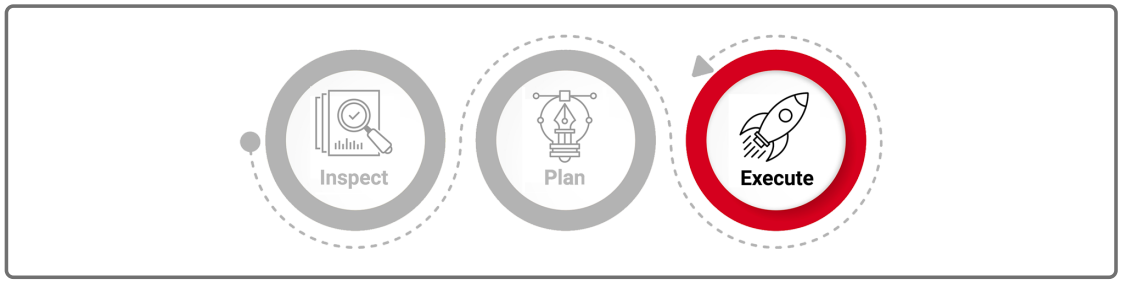
The Kaggle data is much more consistent, and it would be difficult, if not impossible, to translate the Wikipedia data into the same format.



We'll drop the Wikipedia data in this case.

Wikipedia	**Kaggle**	**Resolution**
title_wiki	title_kaggle	Drop Wikipedia.
running_time	runtime	Keep Kaggle; fill in zeros with Wikipedia data.
budget_wiki	budget_kaggle	Keep Kaggle; fill in zeros with Wikipedia data.
box_office	revenue	Keep Kaggle; fill in zeros with Wikipedia data.
release_date_wiki	release_date_kaggle	Drop Wikipedia.
Language	original_language	Drop Wikipedia.
Production company(s)	production_companies	**Drop Wikipedia.**

Put It All Together



First, we'll drop the title_wiki, release_date_wiki, Language, and Production company(s) columns.

```
movies_df.drop(columns=['title_wiki','release_date_wiki','Language','Product
```

Next, to save a little time, we'll make a function that fills in missing data for a column pair and then drops the redundant column.

```
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[kag
        , axis=1)
    df.drop(columns=wiki_column, inplace=True)
```

Now we can run the function for the three column pairs that we decided to fill in zeros.

```
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
```

Since we've merged our data and filled in values, it's good to check that there aren't any columns with only one value, since that doesn't really

provide any information. Don't forget, we need to convert lists to tuples for `value_counts()` to work.

```
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
    num_values = len(value_counts)
    if num_values == 1:
        print(col)
```

Running this, we see that `'video'` only has one value:

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

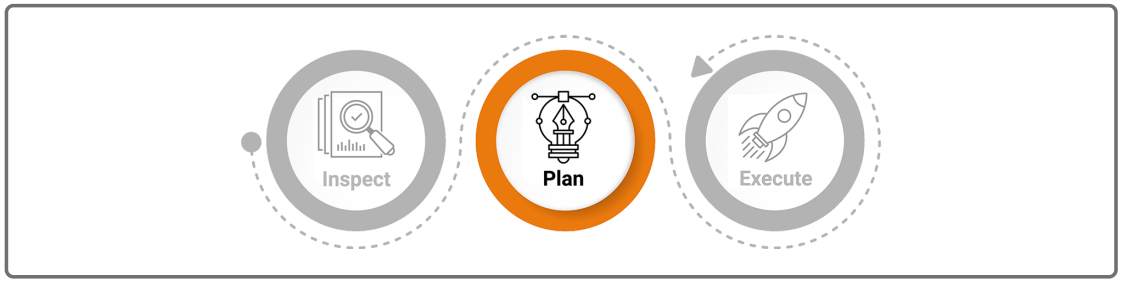
Here's what the output will look like.

```
False    6044
Name: video, dtype: int64
```

Since it's false for every row, we don't need to include this column.

SKILL DRILL

How could you replace the previous `for` loop with a list comprehension?

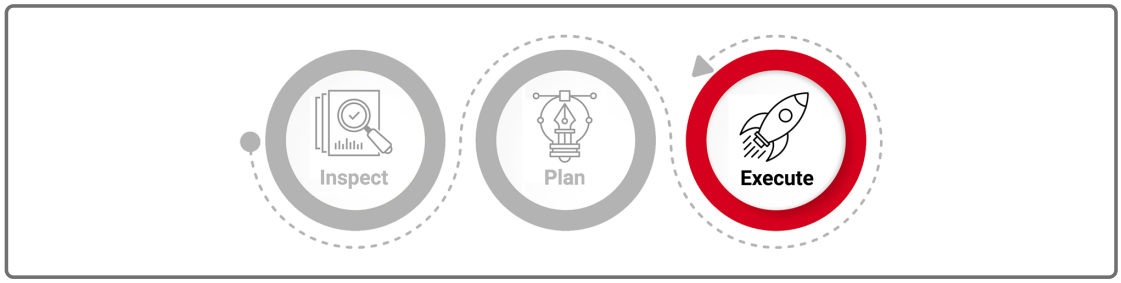


We should reorder the columns to make the dataset easier to read for the hackathon participants. Having similar columns near each other helps people looking through the data get a better sense of what information is available. One way to reorder them would be to consider the columns roughly in groups, like this:

1. Identifying information (IDs, titles, URLs, etc.)
2. Quantitative facts (runtime, budget, revenue, etc.)
3. Qualitative facts (genres, languages, country, etc.)
4. Business data (production companies, distributors, etc.)
5. People (producers, director, cast, writers, etc.)

The following code is one way to reorder the columns:

```
movies_df = movies_df.loc[:, ['imdb_id','id','title_kaggle','original_title'
                       'runtime','budget_kaggle','revenue','release_date_kag
                       'genres','original_language','overview','spoken_langu
                       'production_companies','production_countries','Distri
                       'Producer(s)','Director','Starring','Cinematography',
                      ]]
```



Finally, we need to rename the columns to be consistent.

```
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)
```

NOTE

> If you did not use `.loc` to reorder the columns and instead passed a list of column names to the indexing operator (i.e. `movies_df = movies_df[['imdb_id', 'title_kaggle', … ]]`), you may receive a SettingWithCopyWarning. Don't panic! This isn't an error, so your code will continue to work, but it is a warning that your code may not behave as you expect. In this case, your code will work fine, but for best practices, use `.loc` instead to avoid this warning.

Your first merge is done! We got the tough one out of the way first, and now we're almost done.

ADD/COMMIT/PUSH

Remember to add, commit, and push your work!