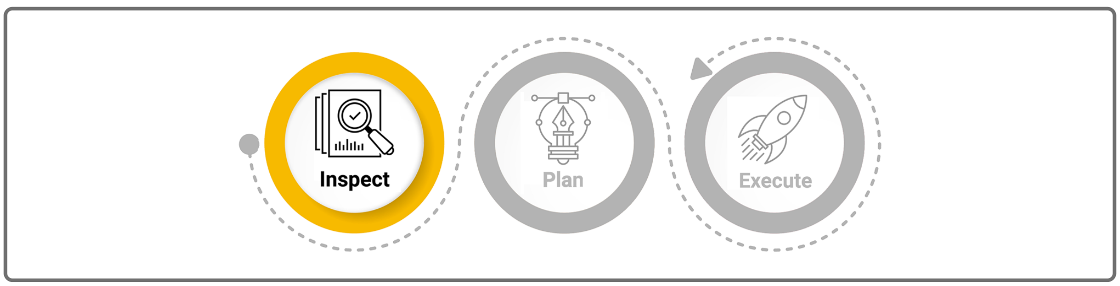
8.3.12 Clean the Kaggle Data

The Kaggle data that Britta found is much more structured, but it still requires some cleaning, including converting strings to correct data types. Therefore, your next task is to clean the Kaggle data.

As always when data cleaning, the first step is to take an initial look at the data you're working with. Let's get started.

Initial Look at the Movie Metadata



Because the Kaggle data came in as a CSV, one of the first things we want to check is that all of the columns came in as the correct data types.

```
kaggle_metadata.dtypes
```

Here's what the output will look like:

```
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
```

Remember, the "object" data type is usually for strings. Only four columns were successfully converted to a data type—`revenue`, `runtime`, `vote_average`, and `vote_count`—but taking a look through the DataFrame, we can see some columns that should be specific data types.

Retake

We'll just go down the list and convert the data types for each of the six columns that need to be converted.

Before we convert the "adult" and "video" columns, we want to check that all the values are either `True` or `False`.

```
kaggle_metadata['adult'].value_counts()
```

Here's what the output will look like.

```
False
True
 Avalanche Sharks tells the story of a bikini contest that turns into a horr
 Rune Balot goes to a casino connected to the October corporation to try to 
 - Written by Ørnås
Name: adult, dtype: int64
```

Clearly, we have some bad data in here. Let's remove it.

Remove Bad Data

To remove the bad data, use the following:

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

Take a closer look at the three movies that appear to have corrupted data:

	adult	belongs_to_collection	budget	genres	homepage	id	imdb_id	original_language	original_title	overv
19730	- Written by Ørnås	0.065736	/ff9qCepilowshEtG2GYWwzt2bs4.jpg	[{'name': 'Carousel Productions', 'id': 11176}...	[{'iso_3166_1': 'CA', 'name': 'Canada'}, {'iso...	1997-08-20	0	104.0	[{'iso_639_1': 'en', 'name': 'English'}]	Relea
29503	Rune Balot goes to a casino connected to the ...	1.931659	/zV8bHuSL6WXoD6FWogP9j4x80bL.jpg	[{'name': 'Aniplex', 'id': 2883}, {'name': 'Go...	[{'iso_3166_1': 'US', 'name': 'United States o...	2012-09-29	0	68.0	[{'iso_639_1': 'ja', 'name': '日本語'}]	Relea
35587	Avalanche Sharks tells the story of a bikini ...	2.185485	/zaSf5OG7V8X8gqFvly88zDdRm46.jpg	[{'name': 'Odyssey Media', 'id': 17161}, {'nam...	[{'iso_3166_1': 'CA', 'name': 'Canada'}]	2014-01-01	0	82.0	[{'iso_639_1': 'en', 'name': 'English'}]	Relea

Somehow the columns got scrambled for these three movies.

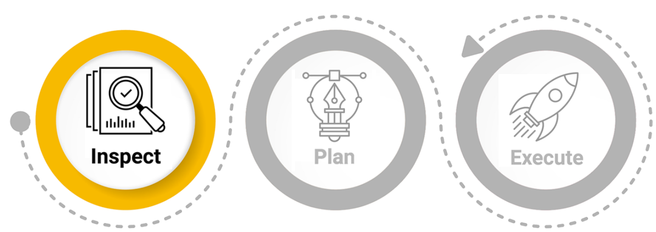
PAUSE

How do we fix the data here?

Show Answer

The following code will keep rows where the adult column is `False`, and then drop the adult column.

```
kaggle_metadata = kaggle_metadata[kaggle_metadata['adult'] == 'False'].drop(
```



Next, we'll look at the values of the video column:

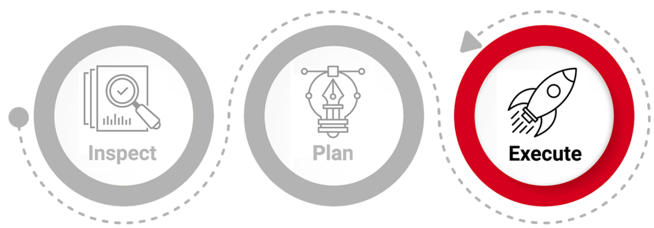
```
kaggle_metadata['video'].value_counts()
```

Here's what the output should look like.

```
False    45358
True        93
Name: video, dtype: int64
```

Great, there are only `False` and `True` values. We can convert `video` fairly easily.

Convert Data Types

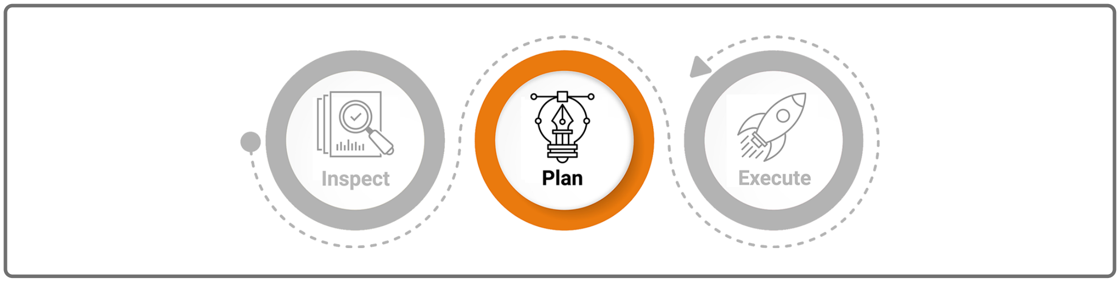


To convert, use the following code:

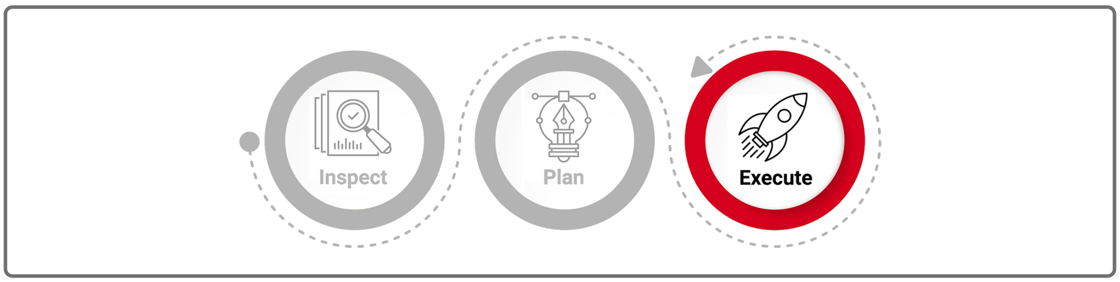
```
kaggle_metadata['video'] == 'True'
```

The above code creates the Boolean column we want. We just need to assign it back to `video`:

```
kaggle_metadata['video'] = kaggle_metadata['video'] == 'True'
```



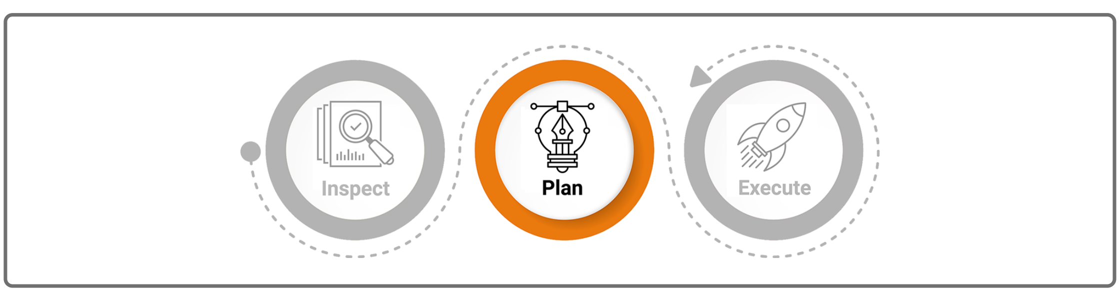
For the numeric columns, we can just use the `to_numeric()` method from Pandas. We'll make sure the `errors=` argument is set to `'raise'`, so we'll know if there's any data that can't be converted to numbers.



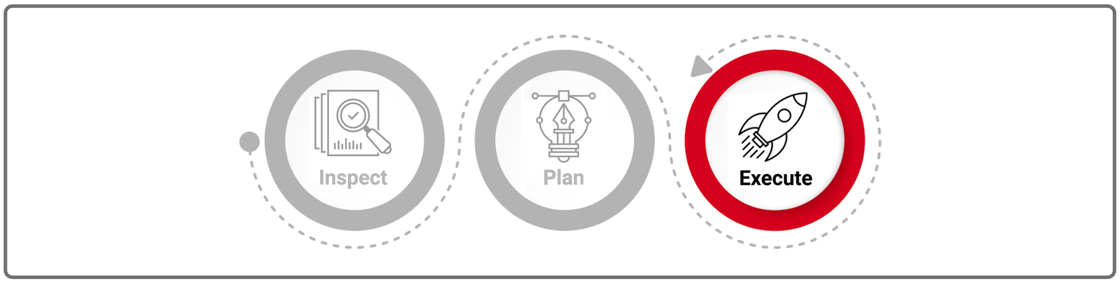

```
kaggle_metadata['budget'] = kaggle_metadata['budget'].astype(int)
kaggle_metadata['id'] = pd.to_numeric(kaggle_metadata['id'], errors='raise')
kaggle_metadata['popularity'] = pd.to_numeric(kaggle_metadata['popularity'],
```

This code runs without errors, so everything converted fine.

Finally, we need to convert `release_date` to datetime. Luckily, Pandas has a built-in function for that as well: `to_datetime()`.



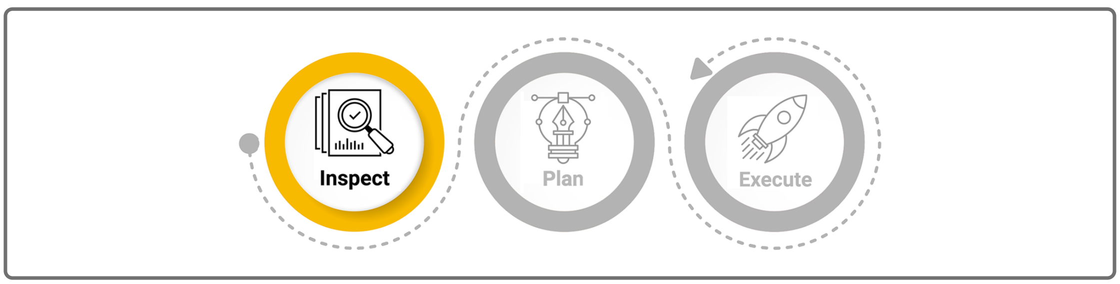
Since `release_date` is in a standard format, `to_datetime()` will convert it without any fuss.




```
kaggle_metadata['release_date'] = pd.to_datetime(kaggle_metadata['release_da
```

And that's it for cleaning the Kaggle metadata!

Reasonability Checks on Ratings Data



Lastly, we'll take a look at the ratings data. We'll use the `info()` method on the DataFrame. Since the ratings dataset has so many rows, we need to set the `null_counts` option to `True`.

```
ratings.info(null_counts=True)
```

The output should look like the following.

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

For our own analysis, we won't be using the timestamp column; however, we will be storing the rating data as its own table in SQL, so we'll need to convert it to a datetime data type. From the MovieLens documentation, the timestamp is the number of seconds since midnight of January 1, 1970.

IMPORTANT

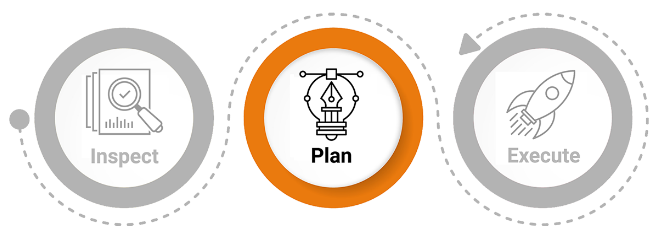
> Storing time values as a data type is difficult, and there are many, many standards out there for time values. Some store time values as text strings, like the ISO format "1955-11-05T12:00:00," but then calculating the difference between two time values is complicated and computationally expensive. The Unix time standard stores points of time as integers, specifically as the number of seconds that have elapsed since midnight of January 1, 1970. This is known as the Unix **epoch**. There are other epochs in use, but the Unix epoch is by far the most widespread.

We'll specify in `to_datetime()` that the origin is `'unix'` and the time unit is seconds.

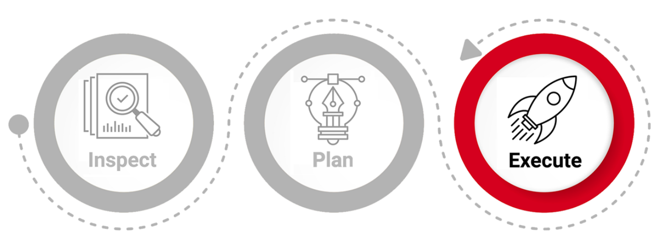
```
pd.to_datetime(ratings['timestamp'], unit='s')
```

The output should look like the following.

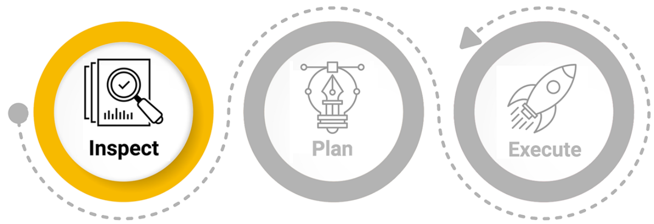
```
0        2015-03-09 22:52:09
1        2015-03-09 23:07:15
2        2015-03-09 22:52:03
3        2015-03-09 22:52:26
4        2015-03-09 22:52:36
5        2015-03-09 23:02:28
6        2015-03-09 22:48:20
7        2015-03-09 22:53:13
8        2015-03-09 22:53:21
9        2015-03-09 23:03:48
10       2015-03-09 22:50:34
11       2015-03-09 22:49:57
12       2015-03-09 23:00:05
13       2015-03-09 22:48:33
14       2015-03-09 23:00:07
15       2015-03-09 22:51:42
16       2015-03-09 22:51:04
17       2015-03-09 23:02:19
18       2015-03-09 23:11:39
19       2015-03-09 23:02:13
```



These dates don't seem outlandish—the years are within expected bounds, and there appears to be some consistency from one entry to the next. Since the output looks reasonable, assign it to the timestamp column.



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



Finally, we'll look at the statistics of the actual ratings and see if there are any glaring errors. A quick, easy way to do this is to look at a histogram of the rating distributions, and then use the `describe()` method to print out some stats on central tendency and spread.

NOTE

> A **histogram** is a bar chart that displays how often a data point shows up in the data. A histogram is a quick, visual way to get a sense of how a dataset is distributed.

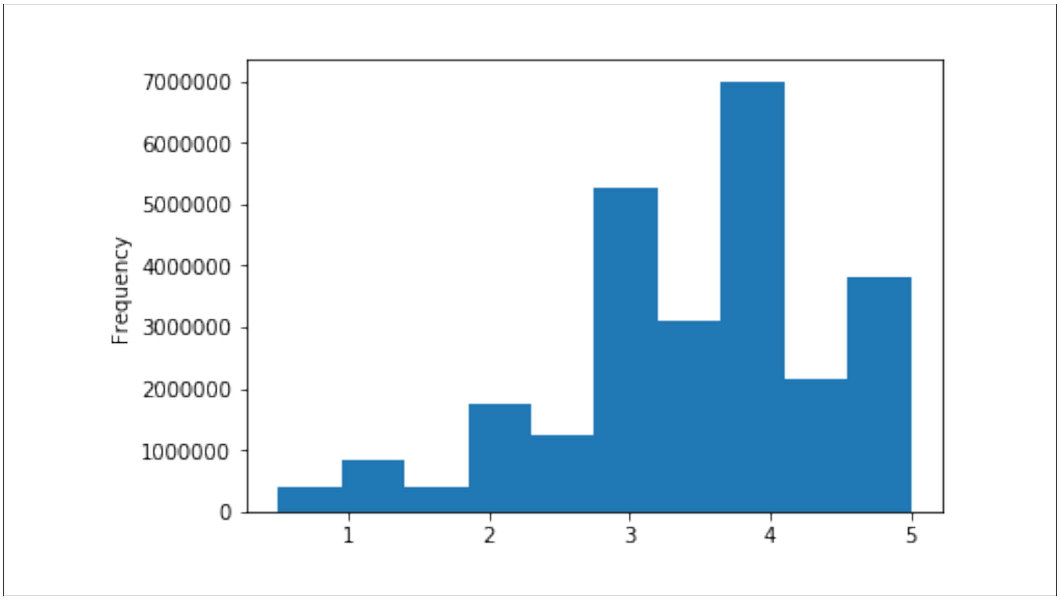
Your code should look like this:

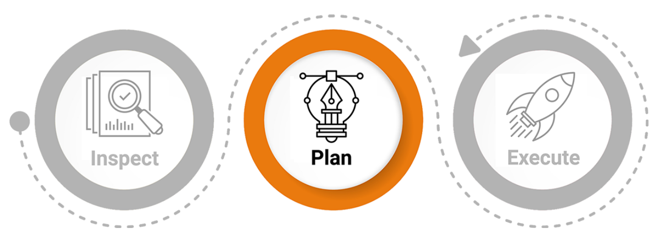
```
pd.options.display.float_format = '{:20,.2f}'.format
ratings['rating'].plot(kind='hist')
ratings['rating'].describe()
```

The output should look like the following.

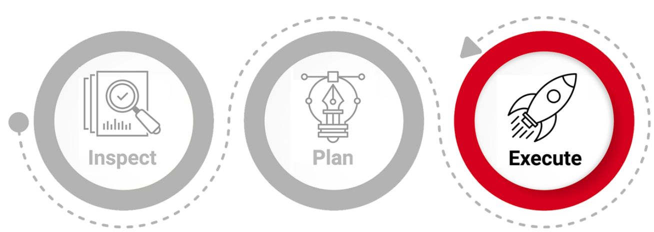
```
count     2.602429e+07
mean      3.528090e+00
std       1.065443e+00
min       5.000000e-01
25%       3.000000e+00
```

```
50%      3.500000e+00
75%      4.000000e+00
max      5.000000e+00
Name: rating, dtype: float64
```





That seems to make sense. People are more likely to give whole number ratings than half, which explains the spikes in the histogram. The median score is 3.5, the mean is 3.53, and all the ratings are between 0 and 5.



The ratings dataset looks good to go, which means we're done with the first half of the Transform step. Let's get ready to finish it.

ADD/COMMIT/PUSH

Remember to add, commit, and push your work!