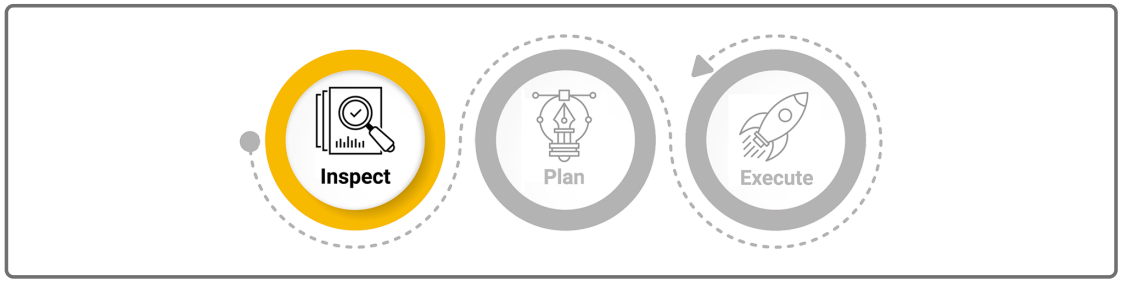
8.3.3 Investigate the Wikipedia Data

Our Wikipedia data is especially messy. As much as editors try to be consistent, each page can be edited by a different person. Besides, because the movie data comes from the sidebar, different movies can have different columns. However, after cleaning the data, the result will be a nice, organized table of data, where every row is a single movie. You'll start by investigating the data for errors.

Initial Investigation



One of the easiest ways to find glaring errors is to just pretend as if there aren't any, and try to jump straight to the finish line. Eventually, we want to clean up the Wikipedia data into tabular data with rows and columns, so let's see what happens if we create a DataFrame from our raw data.

Retake

Run `wiki_movies_df = pd.DataFrame(wiki_movies_raw)`. Then use the `head()` function to take a quick peek at the DataFrame.

```
wiki_movies_df.head()
```

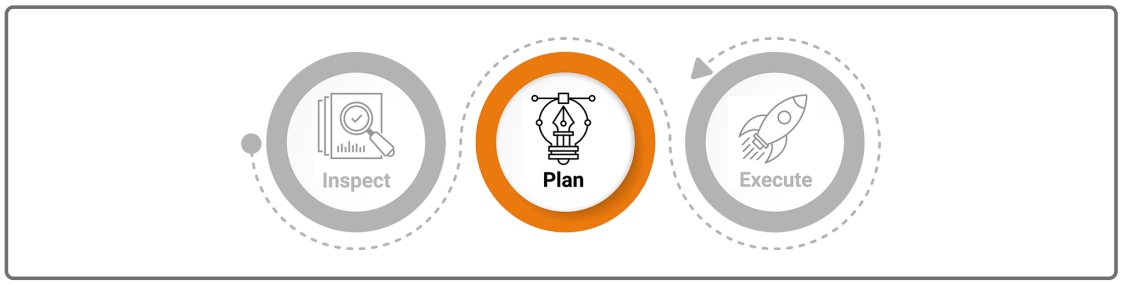
We usually use `head()` to inspect a few rows of data, but it also tells us about the shape of our DataFrame. Below the rows of data, you'll see that it says there are 5 rows of data and 193 columns. That's a lot of columns! Even if we try to use `print(wiki_movies_df.columns)`, they won't all print out. We'll have to convert `wiki_movies_df.columns` to a list to see all of the columns.

Use `wiki_movies_df.columns.tolist()` and run the cell to see all of the column names that were imported. Your output should appear as follows:

```
['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',
'Language',
```

We can identify column names that don't relate to movie data, such as "Dewey Decimal," "Headquarters," and "Number of employees." (There may be other examples that jumped out at you as well.)



Let's modify our JSON data by restricting it to only those entries that have a director and an IMDb link. We can do this with a list comprehension.

Use List Comprehensions to Filter Data

REWIND

We've used list comprehensions previously as a compact way to apply a function to every element in a list. In this module, we'll use list comprehensions to filter data.

So far, we've used list comprehensions in the form to compress code that would have been done in a `for` loop.

```
[expression for element in source_list]
```

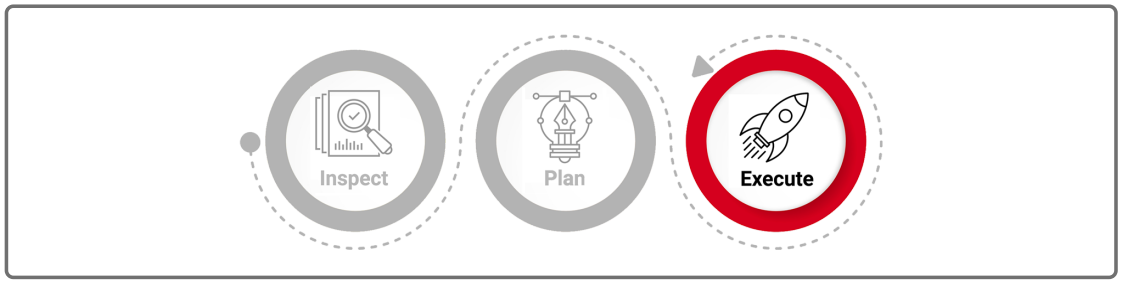
We can also filter out results using a conditional filter expression, as shown below:

```
[expression for element in source_list if filter_expression]
```

The resulting list will only have elements where the filter expression evaluates to True.

To create a filter expression for only movies with a director and an IMDb link, keep in mind that there are two columns in the data for director information. We'll need to check if either "Director" or "Directed by" are keys in the current dict. If there is a director listed, we also want to check that the dict has an IMDb link. Luckily, that information is only in one column, `imdb_link`, so our filter expression will look like the following:

```
if ('Director' in movie or 'Directed by' in movie) and 'imdb_link' in movie
```



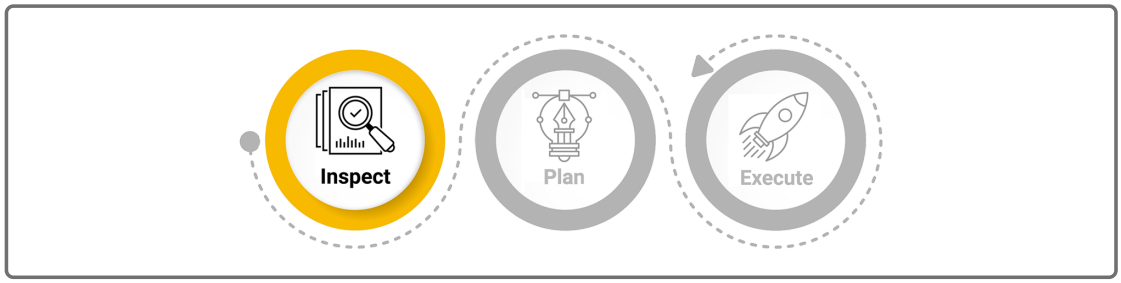
Create a list comprehension with the filter expression we created and save that to an intermediate variable `wiki_movies`. See how many movies are in `wiki_movies` with the `len()` function.

```
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)
```

This only cuts the number of movies down to 7,080. Not too bad. Make a DataFrame from `wiki_movies`, and there should only be 78 columns. It may seem counterintuitive that we have fewer columns in the result set when we constrain our results to rows that must have data in certain columns. But there may be columns that only apply to rows that do not have data in the columns we're targeting. In those cases, the columns will be eliminated, as seen here.

NOTE

> This is why it's easier to load the JSON in first and then convert it to a DataFrame. Instead of trying to identify which columns in our DataFrame don't belong, we just remove the bad data points, and the bad columns never get imported in.

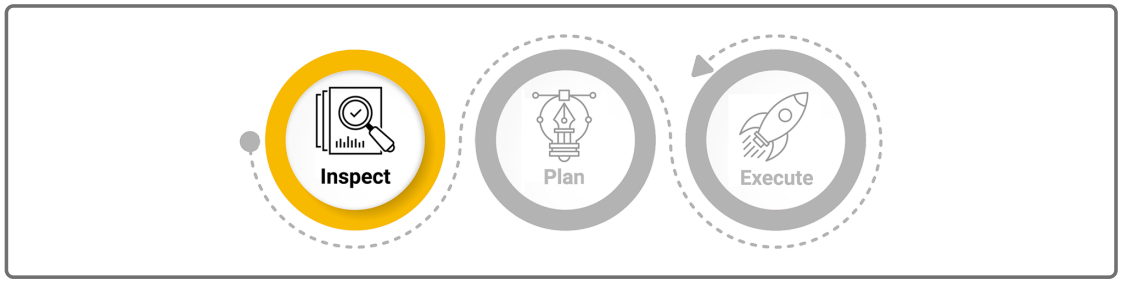


78 columns are still a lot of columns, so let's keep investigating.

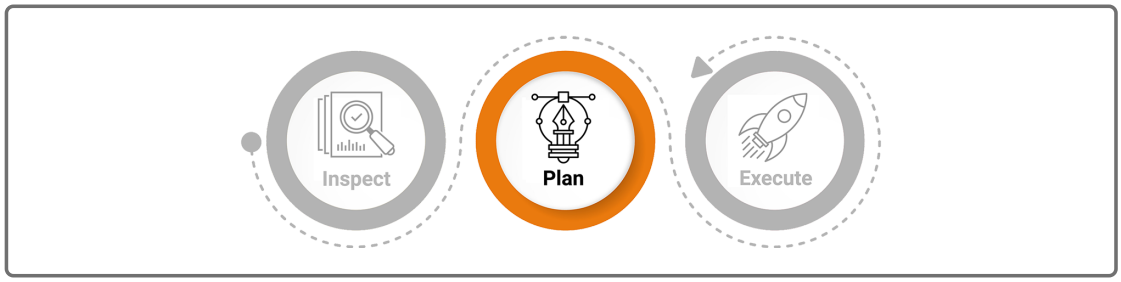
IMPORTANT

> One thing to watch out for is to make **nondestructive** edits as much as possible while designing your pipeline. That means it's better to keep your raw data in one variable, and put the cleaned data in another

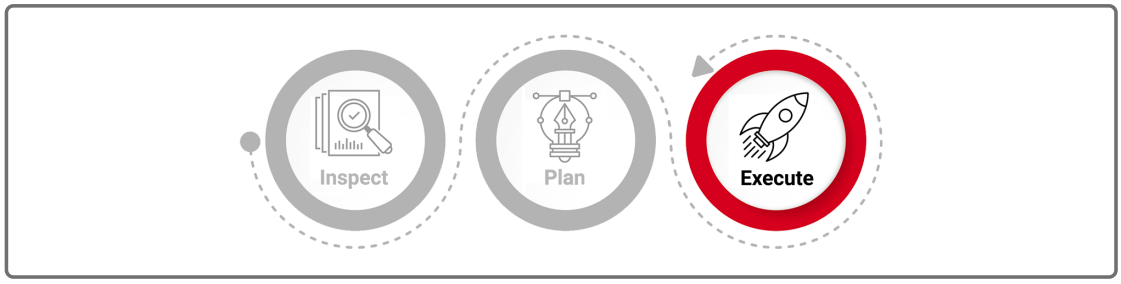
> variable. It takes up more memory, but it makes tracking the iterative process of data cleaning easier.
>
> For example, if we had just deleted the movies from `wiki_movies_raw` that didn't have "Directed by" as a key, we'd have made a destructive edit and missed that some have "Director" as the key instead. This can cause errors to creep in until we realize our mistake, and if we made destructive edits, it would be impossible to see what caused those errors. Using nondestructive edits helps determine the origin of errors.



There sure are a lot of languages—we'll get to those shortly. For now, one of the columns that stands out is "No. of episodes."



It looks like we've got some TV shows in our data instead of movies. We'll want to get rid of those, too.



We'll add that filter to our list comprehension.

```
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]
```

IMPORTANT

> Don't worry if you didn't catch the "No. of episodes" column in the list. Cleaning data is an iterative process, and if you started with cleaning up the language data first, or some other part of the data, you would see the "No. of episodes" column soon enough. The key is to keep reworking the pipeline bit by bit.