8.3.5 Create a Function to Clean the Data, Part 1

Filtering out bad data isn't enough. You know that you need to make sure the *good* data that you have is clean enough to use. There's a lot at stake!

Now we're ready to create our function to clean our movie data.

First, write a simple function to make a copy of the movie and return it. As we work with our data, we'll iteratively add more to our code block. To start, call the function `clean_movie`, and have it take `movie` as a parameter.

```
def clean_movie(movie):
```

Because the movies are dicts and we want to make nondestructive edits, make a copy of the incoming movie.

To make a copy of `movie`, we'll use the `dict()` constructor.

IMPORTANT

> **Constructors** are special functions that initialize new objects. They reserve space in memory for the object and perform any initializations the object requires. Also, constructors can take parameters and initialize a new object using those parameters.

When we pass `movie` as a parameter to the `dict()` constructor, it reserves a new space in memory and copies all of the info in `movie` to that new space.

As an example, we could start our function off with this code:

```
def clean_movie(movie):
    movie_copy = dict(movie)
```

However, we have another trick that's even better.

Inside of the function, we can create a new local variable called `movie` and assign it the new copy of the parameter `movie`.

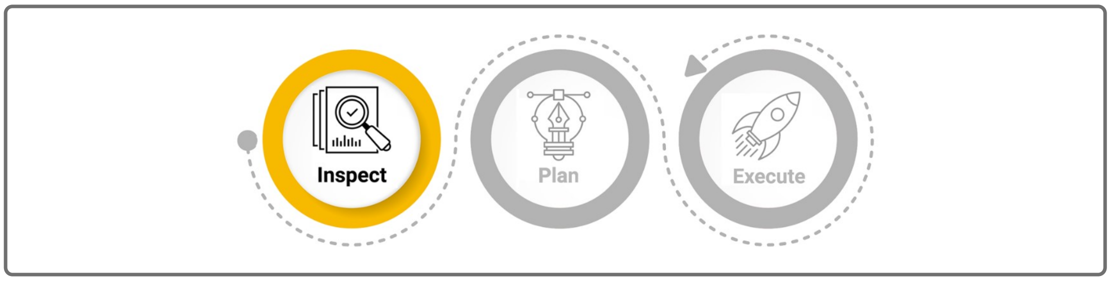
```
def clean_movie(movie):
    movie = dict(movie) #create a non-destructive copy
```

This way, inside of the `clean_movie()` function, `movie` will refer to the local copy. Any changes we make inside `clean_movie()` will now only affect the copy, so if we make a mistake, we still have the original, untouched `movie` to reference.

To finish our skeleton of the `clean_movie` function, return the `movie` variable.

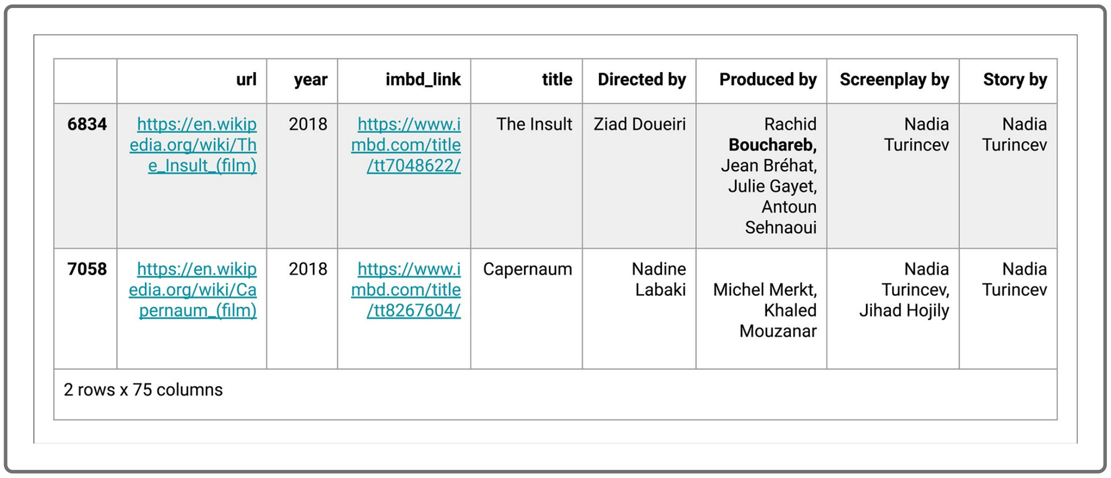
```
def clean_movie(movie):
    movie = dict(movie) #create a non-destructive copy
    return movie
```

This function doesn't do much right now, but we'll be adding more to it soon.



Now take a look at what's going on with those languages. The first one on the list is Arabic, so let's see which movies have a value for "Arabic."

```
wiki_movies_df[wiki_movies_df['Arabic'].notnull()]
```



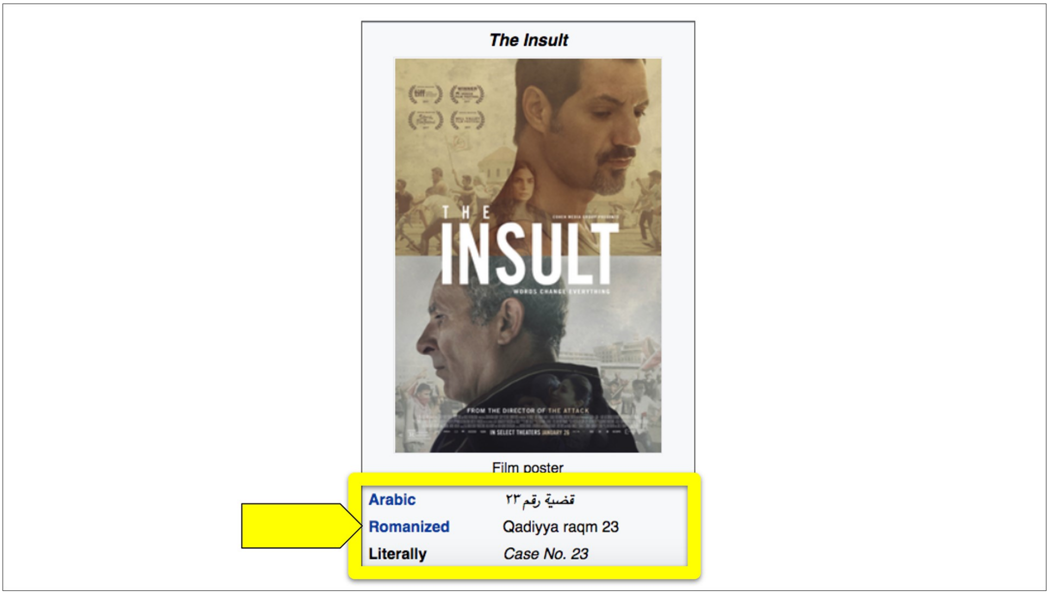
	url	year	imbd_link	title	Directed by	Produced by	Screenplay by	Story by
6834	https://en.wikipedia.org/wiki/The_Insult_(film)	2018	https://www.imbd.com/title/tt7048622/	The Insult	Ziad Doueiri	Rachid **Bouchareb,** Jean Bréhat, Julie Gayet, Antoun Sehnaoui	Nadia Turincev	Nadia Turincev
7058	https://en.wikipedia.org/wiki/Capernaum_(film)	2018	https://www.imbd.com/title/tt8267604/	Capernaum	Nadine Labaki	Michel Merkt, Khaled Mouzanar	Nadia Turincev, Jihad Hojily	Nadia Turincev

2 rows x 75 columns

The results return two movies, the first listed is *The Insult*. Visit the movie's **Wikipedia page (https://en.wikipedia.org/wiki/The_Insult_(film))** for more details.

```
wiki_movies_df[wiki_movies_df['Arabic'].notnull()]['url']
```

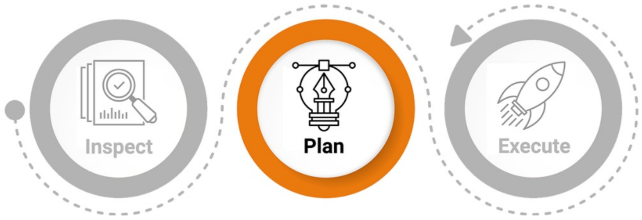
```
6834    https://en.wikipedia.org/wiki/The_Insult_(film)
7058     https://en.wikipedia.org/wiki/Capernaum_(film)
Name: url, dtype: object
```



The Insult

Film poster

Arabic	قضية رقم ٢٣
Romanized	Qadiyya raqm 23
Literally	*Case No. 23*



The different language columns are for alternate titles of the movie. Let's combine all of them into one dictionary that has all the alternate titles.

To do that, we need to go through each of the columns, one by one, and determine which are alternate titles. Some might be tricky. If you're not sure what a column name means, google it. Also, review a column's data to understand the type of content in that column.

For example, perhaps you've never heard of "McCune–Reischauer." Is it an esoteric filmmaking technique? Google it, and you'll learn it's a romanization system for Korean. Now look at the actual values contained in the column. If the values don't make sense to you either, google them, too.

NOTE

> The `value_counts()` method is a quick, easy way to see what non-null values there are in a column.

Try the following Skill Drill. If you're not sure, don't guess. Look at the data, and investigate the source if you have any questions. There are no shortcuts in this task.

SKILL DRILL

Go through each of the columns, one by one, and determine which columns hold alternate titles.

Hint: You might find it easier to sort the column names first as you're going through them. The following will display columns in alphabetical order.

`sorted(wiki_movies_df.columns.tolist())`

Which of these columns hold alternate title data?

Based on

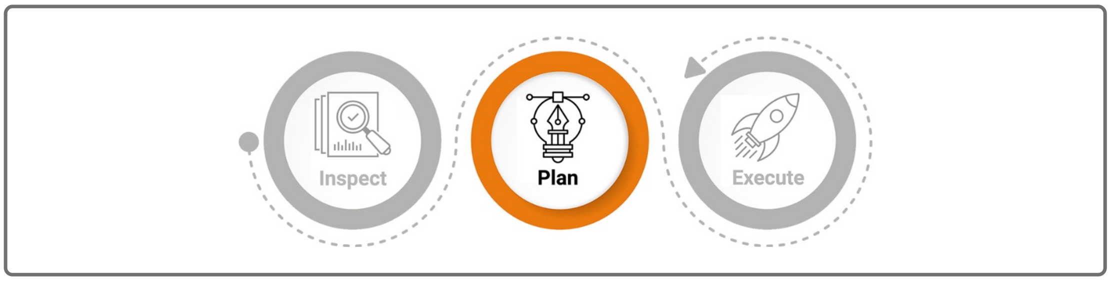
Directed

Language

Polish

Check Answer

Handle the Alternative Titles



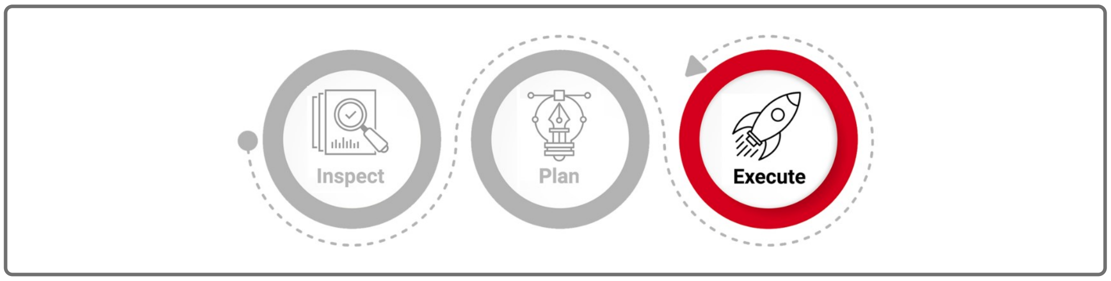
Now we can add in code to handle the alternative titles. The logic we need to implement follows:

1. Make an empty dict to hold all of the alternative titles.
2. Loop through a list of all alternative title keys:
   - Check if the current key exists in the movie object.
   - If so, remove the key-value pair and add to the alternative titles dict.
3. After looping through every key, add the alternative titles dict to the movie object.

SKILL DRILL

Try to implement the logic above in your `clean_movie` function on your own.

Hint: To remove a key-value pair from a dict in Python, use the `pop()` method.



Step 1: Make an empty dict to hold all of the alternative titles.

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

Step 2: Loop through a list of all alternative title keys.

```
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
```

Step 2a: Check if the current key exists in the movie object.

```
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
```

Step 2b: If so, remove the key-value pair and add to the alternative titles dictionary.

```
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
```

Step 3: After looping through every key, add the alternative titles dict to the movie object.

```
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
```

We can make a list of cleaned movies with a list comprehension:

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

Set `wiki_movies_df` to be the DataFrame created from `clean_movies`, and print out a list of the columns.

```
wiki_movies_df = pd.DataFrame(clean_movies)
sorted(wiki_movies_df.columns.tolist())
```

Here's the printed list:

```
['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)',
 'Followed by'
```

We're making a lot of progress! If you've been staring at your screen for a while, now is a great time to take a brief mental break.

ADD/COMMIT/PUSH

Remember to add, commit, and push your work!