4.5.4 Get the Incorrect Student Names

Remember when Maria asked what anomaly you noticed in the `students_complete.csv` file? You found that one of the students' names had the prefix "Dr." Now Maria wants you to find all the students' names that have a professional prefix or suffix. Once you get all the incorrect student names, your next task is to clean the names by removing these professional prefixes and suffixes. Maria is not sure how these names came to have professional prefixes or suffixes, but it's important they are removed. These names are not coinciding with the school records, so the students' scores can't be added their student portfolio and may cause problems when they apply to colleges. Moreover, we can't drop these names because this would affect the total student count of the district as well as the individual schools, which will negatively impact the analysis.

In the `students_complete.csv` file, we will check for all the professional prefixes and suffixes that need to be removed. However, we will not remove family-related suffixes, such as Jr., II, III, and so on.

To begin, we need to do some exploratory analysis and testing to determine how to "clean" a student's name. When cleaning the data, it's a best practice to create a new Jupyter Notebook file rather than using the `PyCitySchools.ipynb` file.

Launch Jupyter Notebook and create a new file named `cleaning_student_names.ipynb`. Another option is to make a copy of the `PyCitySchools.ipynb` file and rename the copy.

We will only need to import `students_complete.csv` and convert it into a DataFrame. Here's the specific code you should be copying (or keeping, if you made a copy of the `PyCitySchools.ipynb` file):

```
# Import dependencies
import pandas as pd
```

```
student_data_to_load = "Resources/students_complete.csv"

student_data_df = pd.read_csv(student_data_to_load)
student_data_df.head()
```

	Student ID	student_name	gender	grade	school_name	reading_score	math_score
0	0	Paul Bradley	M	9th	Huang High School	66	79
1	1	Victor Smith	M	12th	Huang High School	94	61
2	2	Kevin Rodriguez	M	12th	Huang High School	90	60
3	3	Dr. Richard Scott	M	12th	Huang High School	67	58
4	4	Bonnie Ray	F	9th	Huang High School	97	84

IMPORTANT

> The best practice for troubleshooting and testing new coding solutions before you apply them to your original file is to create a copy of the current notebook.
>
> If you do your testing on your original Jupyter Notebook file, the chances of changing the code of the original file is high, which may cause errors that were not present before.

Next, we need to get the students' names in a separate file. This will make it easier to determine how many students' names are affected, as well as what prefixes and suffixes need to be removed. First, we need to get all the students' names in a separate file so we don't affect the DataFrame.

The tolist() Method

To get all the students' names in a separate file, we'll use the Pandas `tolist()` method on a DataFrame with reference to the `student_name` column. Using the `tolist()` method on the `student_name` column will add all the names to a list.

REWIND

Recall that every column in the CSV file and DataFrame is a Series or a list.

In the `cleaning_student_names.ipynb` file, add the following code and run the cell:

```
# Put the student names in a list.
student_names = student_data_df["student_name"].tolist()
student_names
```

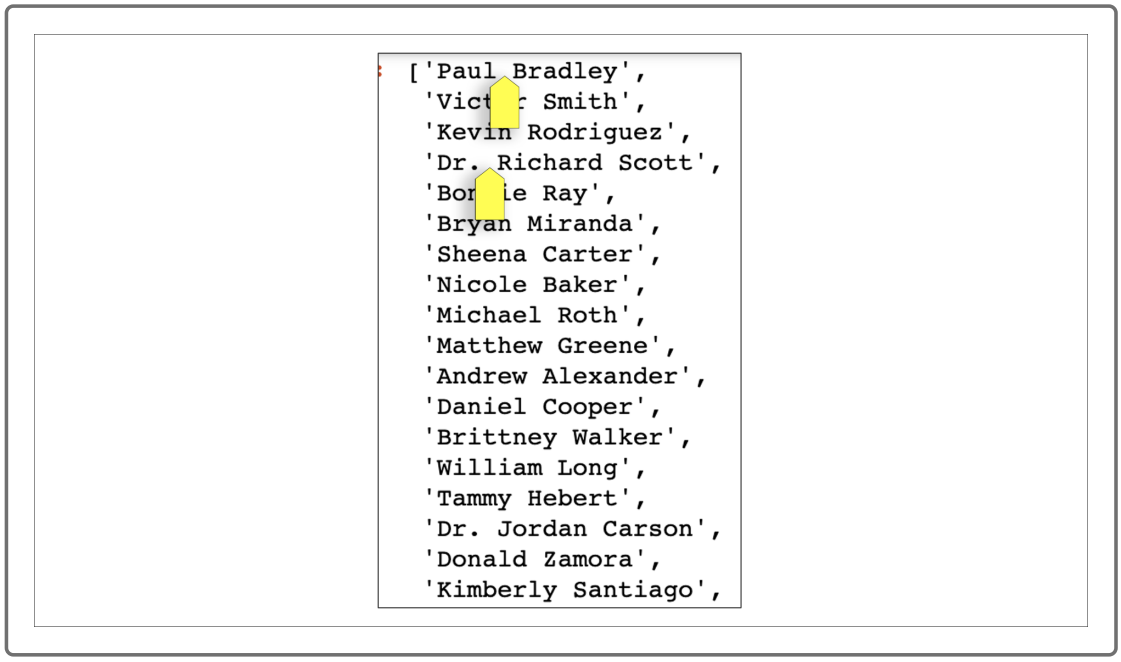
When we run this file, the first few lines in the output will look like this:

```
['Paul Bradley',
 'Victor Smith',
 'Kevin Rodriguez',
 'Dr. Richard Scott',
 'Bonnie Ray',
 'Bryan Miranda',
 'Sheena Carter',
 'Nicole Baker',
 'Michael Roth',
 'Matthew Greene',
 'Andrew Alexander',
 'Daniel Cooper',
 'Brittney Walker',
 'William Long',
 'Tammy Hebert',
 'Dr. Jordan Carson',
 'Donald Zamora',
 'Kimberly Santiago',
```

If we scroll through the list of 39,170 students, we'll see that many of the students' names have the "Dr." prefix, and there are other prefixes and suffixes as well. However, picking out all of the different prefixes and suffixes this way is not an efficient use of our time, and we may not catch all of them. Let's try a different method.

Look carefully at the list of names. Between the prefix and the first name, and between the first and last name, there are whitespaces—spaces with no text. Scrolling down the list, we'll observe the same situation for last name and suffix.

In Python, we can split the names on these whitespaces using the `split()` method. Look at the following image to see an example of whitespace.



The split() Method

In Python, the `split()` method will split a Python string object on the whitespace, or where there is no text. If we split each name, we can count the length of each name. A student with a first and last name only will return a length of 2, whereas a name with a prefix or suffix only will have a length of 3.

Try the following example.

Retake

In the `student_names` list, we can use a `for` loop to iterate through the list, split each name, and print out the name and the length of the name.

In a new cell, add the following code and run the cell.

```
# Split the student name and determine the length of the split name.
for name in student_names:
    print(name.split(), len(name.split()))
```

A sample of the output shows that the length of a majority of the names is 2, which corresponds to names that have only a first and last name. Names with a length greater than 2 have a prefix and/or suffix in addition to the first and last name. The output sample is shown below:

```
['Jon', 'Smith'] 2
['Tracey', 'Yates'] 2
['Jerry', 'Gordon'] 2
['Michael', 'Cox'] 2
['Madeline', 'Snyder', 'MD'] 3
['Allison', 'King'] 2
['Theresa', 'Meyer'] 2
['Jared', 'Wood'] 2
['Eric', 'Maynard'] 2
['Mark', 'Owens'] 2
['Ronnie', 'Conley'] 2
['Kimberly', 'Davis'] 2
['Amy', 'Mitchell'] 2
['Mr.', 'Dylan', 'Taylor', 'MD'] 4
['Elizabeth', 'Henderson'] 2
```

We can filter this list using a conditional statement. If the length of the name is equal to or greater than 3, we can add it to a new list. By getting the length of this new list, we can determine how many students are affected. Your code should look like this:

```
# Create a new list and use it for the for loop to iterate through the list.
students_to_fix = []

# Use an if statement to check the length of the name.
# If the name is greater than or equal to "3", add the name to the list.

for name in student_names:
    if len(name.split()) >= 3:
```

```
        students_to_fix.append(name)

# Get the length of the students whose names are greater than or equal to "3
len(students_to_fix)
```

Retake

The output should show 1,531 students:

```
# Get the length of the students whose names are greater than or equal to "3".
len(students_to_fix)
```

```
1531
```

If we print the list of students, we can see the different prefixes and suffixes:

```
print(students_to_fix)
```

```
['Dr. Richard Scott', 'Dr. Jordan Carson', 'Madeline Snyder MD', 'Mr. Dylan Taylor MD', 'Dr. Scott Gill', 'Miss Madis
on Everett', 'Virginia Ramirez MD', 'Joseph Morales III', 'Angela Perkins DVM', 'Heather Allen MD', 'Luke Lowery MD',
'Dr. Larry Hines', 'Emily Cardenas MD', 'Mrs. Elizabeth Espinoza DDS', 'Mrs. Lisa Becker', 'Mr. Travis Young', 'Paula
Fernandez DVM', 'Dr. Xavier Bell II', 'Andrew Hanson MD', 'Mr. Roberto Wright MD', 'Mr. Dale Miller', 'Mary Gonzalez
DDS', 'Mrs. Cindy Brown', 'James White MD', 'Donald Gill DDS', 'Mrs. Catherine Williams MD', 'Ashley Caldwell MD', 'M
r. Jared Campbell', 'Michael Moreno PhD', 'Tara Parker MD', 'Shawn Williams DVM', 'Dr. Alicia Martinez', 'Mr. Timothy
Anderson', 'Mrs. Paula Lee', 'Mr. Elijah Hall', 'Mr. Raymond Stone', 'John Dickerson MD', 'Mr. Alex Franklin', 'Mrs.
Jessica Dunn', 'Mr. Michael Stevenson', 'Jason Jones DDS', 'Adriana Wade MD', 'Erica Hurley MD', 'Dr. Garrett Wrigh
t', 'Mr. James Allen', 'Mr. Timothy Harper MD', 'Dale Brown MD', 'Dr. Christina Moore', 'Angela Peterson MD', 'Jennif
er Ferguson MD', 'Anthony Wilson DDS', 'Dr. Jessica Lopez', 'David Miller Jr.', 'Dr. John Jones', 'Ms. Samantha Brow
n', 'Cynthia Hansen DVM', 'Rebecca Montoya MD', 'Dr. Jacob Matthews', 'Dr. Jacob Bell', 'Mr. Bruce Thomas', 'Robert M
artin MD', 'Mrs. Marcia Jones PhD', 'Reginald Garcia IV', 'Patricia Smith MD', 'Amy Dyer DDS', 'Brian Hall DDS', 'Dou
glas Levy DDS', 'Mr. Eric Ibarra', 'Benjamin Ramirez MD', 'Raymond Freeman DDS', 'Carla Reeves MD', 'Dr. Donald Willi
ams', 'Mr. John Carter', 'William Jackson DVM', 'Mr. Mitchell Cooper', 'Kevin Brown IV', 'Mrs. Kathleen Reed', 'Donna
Robinson MD', 'Mrs. Dawn Olson DDS', 'Mr. Dustin Glover', 'Alexis French MD', 'Lisa Guerrero MD', 'David Nguyen DVM',
'William Washington Jr.', 'Julie Patterson MD', 'Mr. James Kelly', 'Dr. Robert Martin', 'Darlene Thomas PhD', 'Mr. Ju
an Jenkins', 'Mr. Juan Bryant II', 'Mr. Matthew Coleman', 'Bonnie Hammond MD', 'Kathleen Harris DDS', 'Steven Velez D
DS', 'Tammy Cruz DDS', 'Dr. Marie Williams', 'Kenneth Paul Jr.', 'Mr. Jose Morris', 'Scott Rivers III', 'Melissa Rodr
iguez MD', 'Geoffrey Ortiz MD', 'Matthew Stewart MD', 'Sandra Thompson DDS', 'Mark Myers Jr.', 'Mrs. Maria Smith DD
```

Great job! Using the `split()` method is much more efficient than scrolling through all 39,170 names!

By glancing at the output, we can see that prefixes include Dr., Mr., Mrs., and Miss, and the suffixes include MD, DDS, DVM, Ph.D., Jr., II, III, and IV. However, there could be more.

Going through this list name by name to find all of the prefixes and suffixes would be time-consuming. Instead, we can write an algorithm to get the length of each name, including the prefix or suffix from the list. This will allow us to filter out prefixes and suffixes.

Get All the Prefixes

To get all of the prefixes in the student data, we're going to write an algorithm that will iterate through the `students_to_fix` list and determine if the length of the prefix, or the first item in the list after using `name.split()`, is less than or equal to 4. We'll use 4 as the upper limit because the longest prefix is "Miss," which is four characters long. If the length is less than or equal to 4, we'll add it to a new list named `prefixes`.

In a new cell, add the following code and run the cell:

```
# Add the prefixes less than or equal to 4 to a new list.
prefixes = []
for name in students_to_fix:
    if len(name.split()[0]) <= 4:
        prefixes.append(name.split()[0])

print(prefixes)
```

With the `split()` method, we can get the length of not only each name in the `students_to_fix` list but also the length of each **substring** in the list. In this case, a substring can be either the prefix, first name, last name, or suffix.

In `len(name.split()[0])`, we're getting the length of the first item, or prefix. When we print out the `prefixes` list, we'll see first names and prefixes that

are less than or equal to 4:

```
['Dr.',
 'Dr.',
 'Mr.',
 'Dr.',
 'Miss',
 'Luke',
 'Dr.',
 'Mrs.',
 'Mrs.',
 'Mr.',
 'Dr.',
 'Mr.',
 'Mr.',
 'Mary',
 'Mrs.',
 'Mrs.',
 'Mr.',
 'Tara',
```

Get All the Suffixes

Let's apply the same methodology to get the suffixes. We'll iterate through the `students_to_fix` list and determine if the last item `[-1]` after using `name.split()` is less than or equal to 3.

REWIND

We can use negative indexing to get the last item in the list.

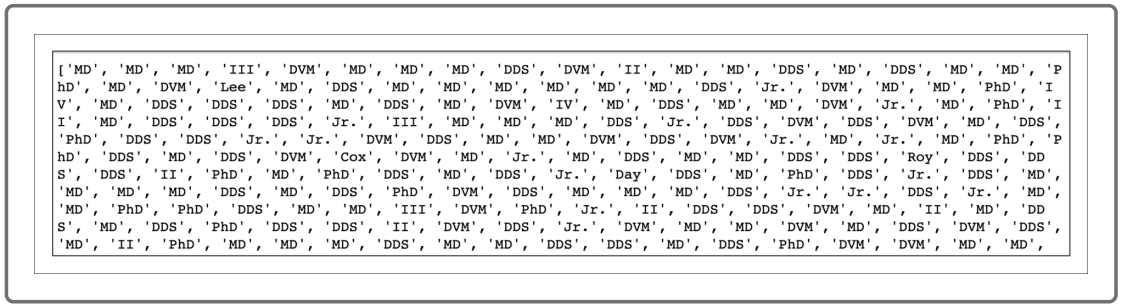
We'll use 3 as the upper limit because the longest suffixes (DDS, DVM, and Ph.D.) are three characters long. If the length is less than or equal to 3, we'll add it to a new list named `suffixes`.

In a new cell, add the following code and run the cell.

```
# Add the suffixes less than or equal to 3 to a new list.
suffixes = []
for name in students_to_fix:
    if len(name.split()[-1]) <= 3:
        suffixes.append(name.split()[-1])

print(suffixes)
```

When we print out the `suffixes` list, we'll see all the suffixes that are less than or equal to 3:



Both of the lists, `prefixes` and `suffixes`, are long. Fortunately, we can make these lists shorter by getting the unique items in each list with a Python method called `set()`.

NOTE

> For more information, read the **Python documentation on splitting strings (https://docs.python.org/3/library/stdtypes.html#string-methods)**.

The set() Method

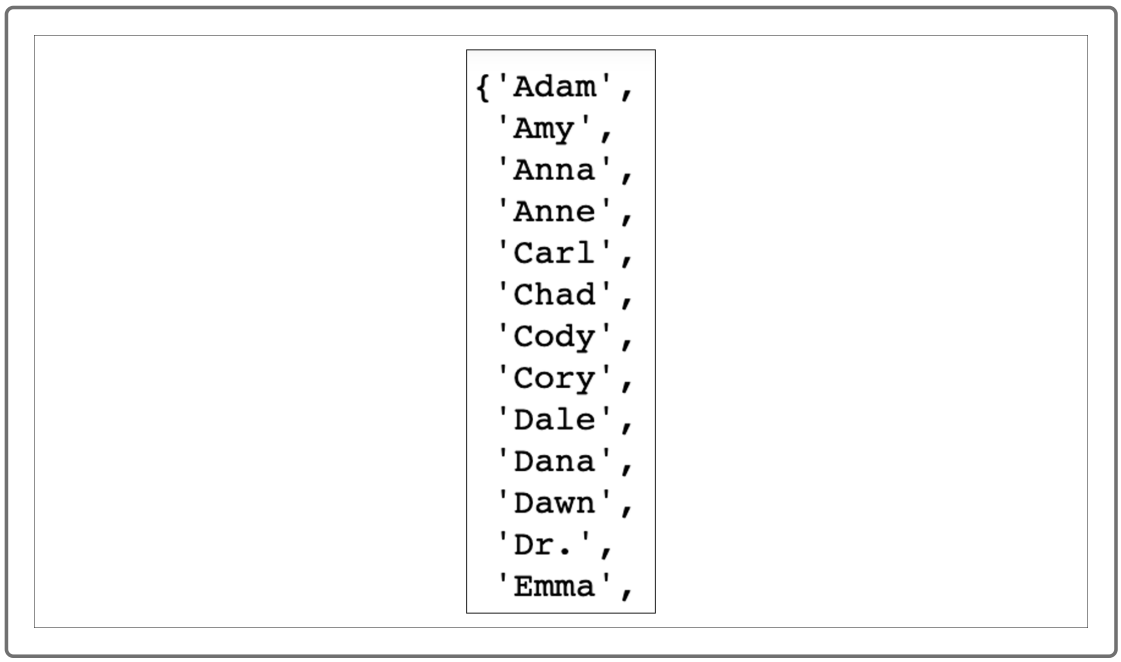
The `set()` method returns all unique items in a list when that list is added inside the parentheses. If the list contains strings, they will be ordered

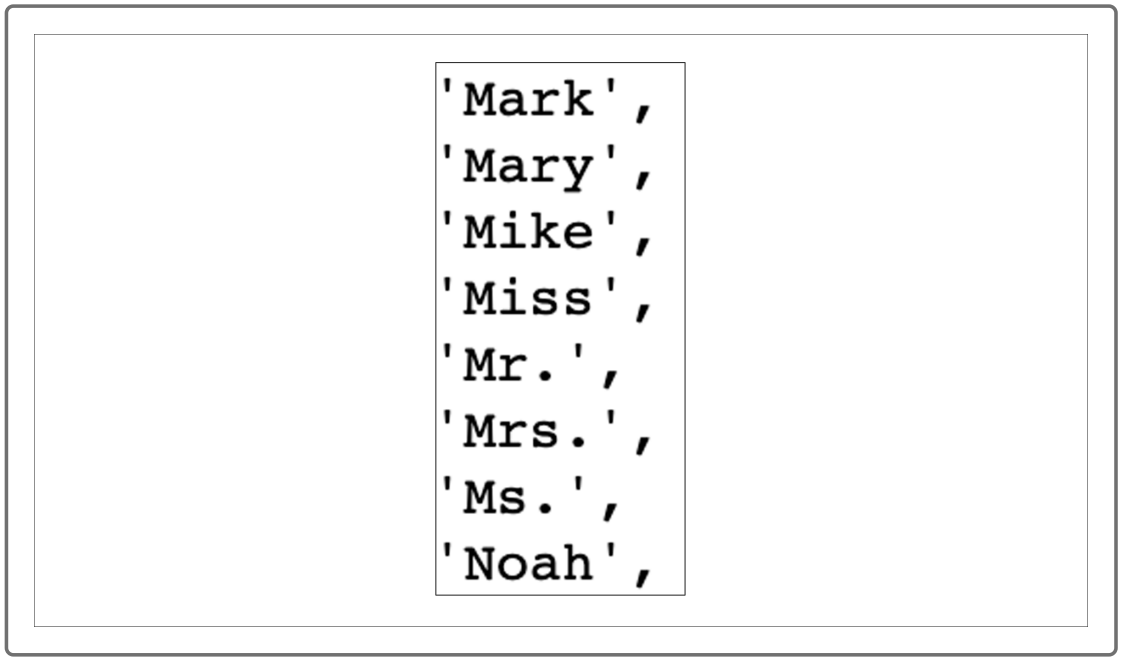
alphabetically. Let's apply this method to the `prefixes` list first.

In a new cell, add the following code and run the cell.

```
# Get the unique items in the "prefixes" list.
set(prefixes)
```

Looking at the output, we can pick out the prefixes Dr., Mr., Mrs., Ms., and Miss:





Now let's apply the same method to the suffixes. In a new cell, add the following code and run the cell.

```
# Get the unique items in the "suffixes" list.
set(suffixes)
```

In the output, you should see the suffixes MD, DDS, DVM, Ph.D., Jr., II, III, IV, and V.

```
set(suffixes)
```

```
{'Cox',
 'DDS',
 'DVM',
 'Day',
 'II',
 'III',
 'IV',
 'Jr.',
 'Kim',
 'Lee',
 'Li',
 'MD',
 'PhD',
 'Roy',
 'V'}
```

Congratulations on finding the unique prefixes and suffixes from the names! Our exploratory analysis was a success. Now we can remove Dr., Mr., Mrs., Ms., Miss, MD, DDS, DVM, and Ph.D. from the students' names in `student_data_df`.