4.5.6 Replace Substrings

Now that you have found a way to replace the suffixes and prefixes in the students' names, Maria wants you to test how to remove them before you apply the procedure to the DataFrame.

We have determined that we can use the `replace()` method to remove prefixes and suffixes from the students' names. However, we don't want to write a different `replace()` statement for each prefix and suffix; this is not efficient or practical programming.

A more efficient way to remove the prefixes and suffixes is to do the following:

1. Declare a list named `prefixes_suffixes` that holds the common prefixes and suffixes as strings.
2. Iterate through this list with a `for` loop.
3. For each item in the list, use the `replace()` method on the student_name column.

Let's start cleaning! In the `cleaning_student_names.ipynb` file, type the following code in a new cell and run the cell.

```
# Add each prefix and suffix to remove to a list.
prefixes_suffixes = ["Dr. ", "Mr. ","Ms. ", "Mrs. ", "Miss ", " MD", " DDS",
```

Remember that there is a whitespace between the prefix and the first name, as well as between the last name and the suffix, as shown in the above image. We're also going to add a whitespace after each prefix and before each suffix in our list of prefixes and suffixes.

NOTE

> The `replace()` method can be used only on Python strings.

REWIND

Remember, we used the `dtypes` method to determine the data types of each column in a DataFrame. For the `student_data_df` DataFrame, we determined that the `student_name` column was an object, not a string.

Even though an object is typically a string, we need to convert the object data type to a string while using the `replace()` method. Luckily, Python

provides us with a way to access and manipulate strings with the `str` attribute.

We can chain the `str` method with the `replace()` method on the `student_name` column in the `student_data_df`, like this:

```
student_data_df["student_name"].str.replace()
```

Next, iterate through the list of prefixes and suffixes by passing the prefix or suffix as `word` in the `replace()` method, as shown in the following code.

```
# Iterate through the "prefixes_suffixes" list and replace them with an empty
for word in prefixes_suffixes:
    student_data_df["student_name"] = student_data_df["student_name"].str.re
```

There's a lot going on in this code, so let's break it down.

- First, we iterate through the `prefixes_suffixes` list.
- In the first part of the `for` loop, we assign each student name in the `student_name` column with the same student name after we replace the prefix or suffix with an empty string.
- To replace the prefix or suffix with an empty string, we convert the name of the student to a string with `student_data_df["student_name"].str`. For every `word` in the `prefixes_suffixes` list, we replace that prefix or suffix, if the name has one, with an empty string using `replace(word,"")`.

To check if this works, let's print the first 10 rows of `student_data_df`. The results should look like this:

It looks like it worked! Now we have to confirm that all of the prefixes and suffixes were caught in our script.

Reusing some of our previous code to check the length of each name, we will:

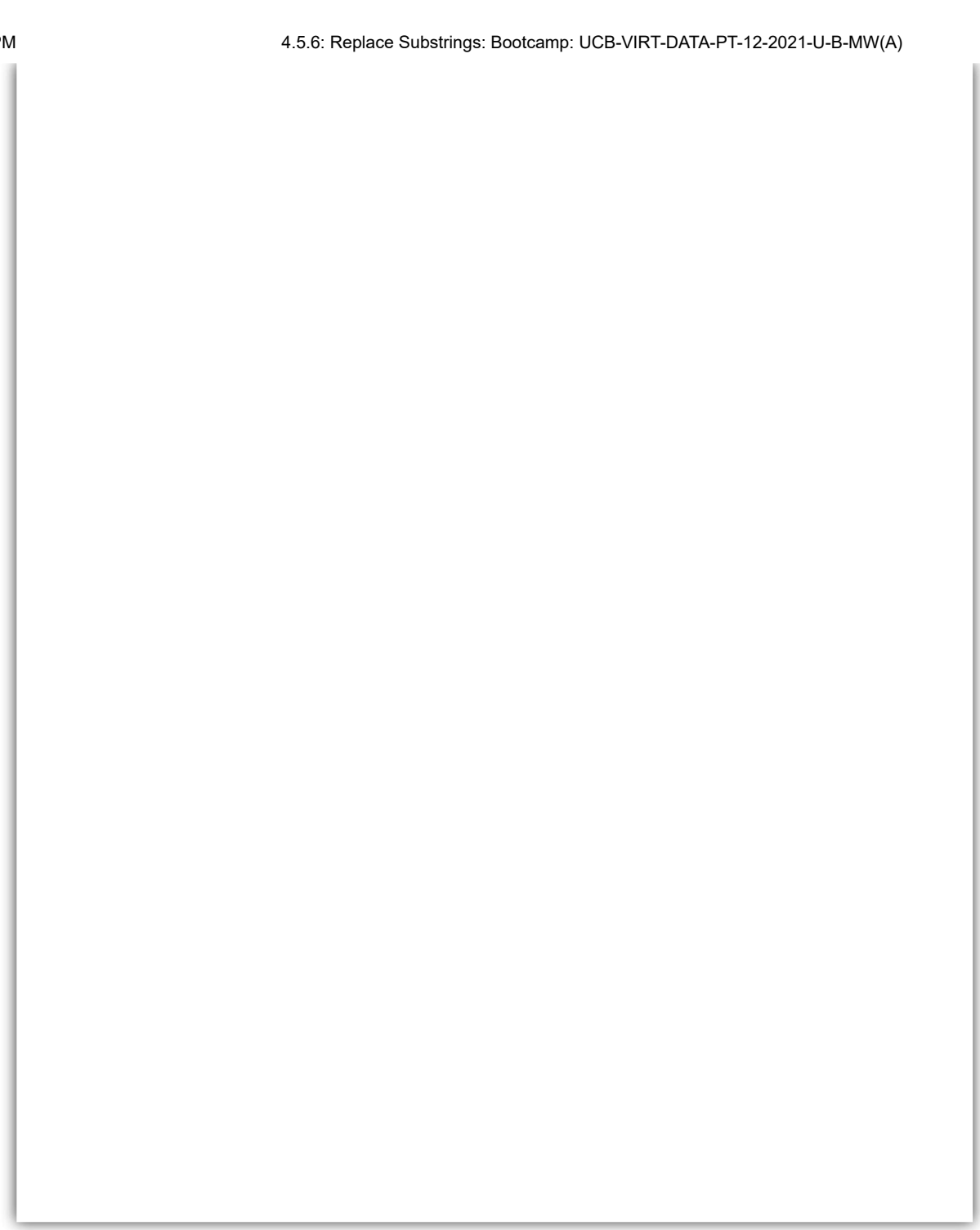
1. Declare a new variable and assign it to the `student_data_df["student_name"]`.
2. Convert the `student_data_df["student_name"]` to a list using `.tolist()`.

Add the following code to a new cell and run the cell.

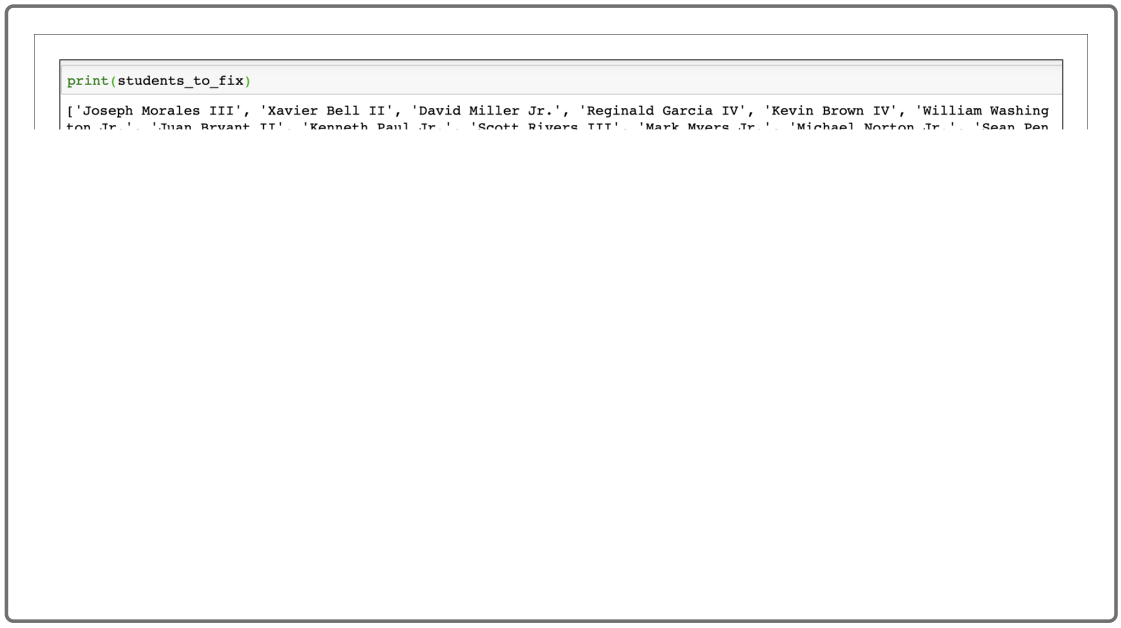
```
# Put the cleaned students' names in another list.
student_names = student_data_df["student_name"].tolist()
student_names
```

Now, add all the students' names that are greater than or equal to 3 to a list. Add the following code in a new cell and run the cell.

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

# 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_fixed.append(name)

# Get the length of the students' names that are greater than or equal to 3.
len(students_fixed)
```



Here's the printed list:



There are now only names with familial designations like Jr., II, III, IV, and V in the `students_fixed` list, which confirms that we caught all the professional prefixes and suffixes. Congratulations on cleaning the data! Let's apply the same process to the data in the `PyCitySchools.ipynb` file.