9.2.2 Save Query Results

In order to have easy access to the results, we need to put them in a DataFrame. This will help solidify a repeatable process and make it easier to run this analysis again—for example, when we need to figure out where to open our second surf and ice cream shop.

We have our weather results saved in a variable. In order to access it in the future, we'll save it to a Python Pandas DataFrame. We'll start by creating a DataFrame variable, `df`, which we can use to save our query results.

In order to save our results as a DataFrame, we need to provide our results variable as one parameter and specify the column names as our second parameter. To do this, we'll add the following line to our code:

```
df = pd.DataFrame(results, columns=['date','precipitation'])
```

This saves our query results in two columns, `date` and `precipitation`. Now we can manipulate the results however we would like. There are many

functions you can use to manipulate how DataFrames look, but we'll start with using the `set_index()` function.

Use the set_index() Function

The `set_index()` function can be a little tricky, but let's jump in. For example, let's say our DataFrame looks something like below. Note that our data will not look like this exactly—this is just an example.

```
   date        precipitation
0   08/23/2017  1
1   08/22/2017  2
2   08/21/2017  1
3   08/20/2017  1
```

The first column is auto-generated and contains the row number. However, we want the index column to be the date column, so we'll need to get rid of those row numbers.

To do this, set the index to the date column. This will make the date column the first column.

For this project, we're going to experiment and write over our original DataFrame. By doing this, we can reduce the complexity of our code and use fewer variables.

REWIND

We can use the variable `inplace` to specify whether or not we want to create a new DataFrame.

Let's go ahead and use the same DataFrame. By setting `inplace=True`, we're saying that we do not want to create a new DataFrame with the modified specifications. If we set it to "False," then we would create a new DataFrame. Add the following to your code:

```
df.set_index(df['date'], inplace=True)
```

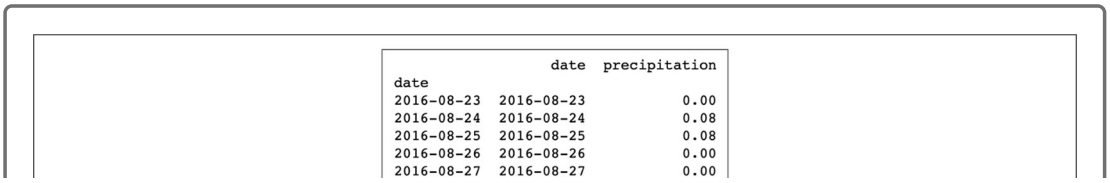
Run the code. Then print the DataFrame with and without the index so that you can see the difference.

Print the DataFrame With and Without the Index

To print a DataFrame with the index, use the following code:

```
print(df)
```

Here's what the printed DataFrame with the index should look like:



	date	precipitation
date		
2016-08-23	2016-08-23	0.00
2016-08-24	2016-08-24	0.08
2016-08-25	2016-08-25	0.08
2016-08-26	2016-08-26	0.00
2016-08-27	2016-08-27	0.00

```
2016-08-28  2016-08-28          0.01
2016-08-29  2016-08-29          0.00
2016-08-30  2016-08-30          0.00
2016-08-31  2016-08-31          0.13
2016-09-01  2016-09-01          0.00
2016-09-02  2016-09-02          0.00
2016-09-03  2016-09-03          0.00
2016-09-04  2016-09-04          0.03
2016-09-05  2016-09-05           NaN
2016-09-06  2016-09-06           NaN
2016-09-07  2016-09-07          0.05
2016-09-08  2016-09-08          0.00
2016-09-09  2016-09-09          0.03
2016-09-10  2016-09-10          0.00
2016-09-11  2016-09-11          0.05
2016-09-12  2016-09-12          0.00
2016-09-13  2016-09-13          0.02
2016-09-14  2016-09-14          1.32
2016-09-15  2016-09-15          0.42
2016-09-16  2016-09-16          0.06
2016-09-17  2016-09-17          0.05
2016-09-18  2016-09-18          0.00
2016-09-19  2016-09-19          0.00
2016-09-20  2016-09-20          0.00
```

Great work! Our DataFrame looks good. However, because we are using the date as the index, the DataFrame has two date columns, which is confusing. So we'll print the DataFrame without the index so we can see just the date and precipitation.

For this task, we'll need to use a slightly different print statement. First we'll convert the DataFrame to strings, and then we'll set our index to "False." This will allow us to print the DataFrame without the index. Add the following to your code:

```
print(df.to_string(index=False))
```

When you run the code, your results should look like the image below. Note that this image shows only the first several lines—your results will likely be longer.

```
      date  precipitation

2016-08-23           0.00
2016-08-24           0.08
2016-08-25           0.08
2016-08-26           0.00
2016-08-27           0.00
2016-08-28           0.01
2016-08-29           0.00
2016-08-30           0.00
2016-08-31           0.13
2016-09-01           0.00
```

2016-09-02	0.00
2016-09-03	0.00
2016-09-04	0.03
2016-09-05	NaN
2016-09-06	NaN
2016-09-07	0.05
2016-09-08	0.00

Good work! Now that our DataFrame is set up, we can sort the results within the DataFrame. If you scroll through the data, you may notice that many dates are not in chronological order. Our next order of business is to fix that.