

# KEY\_Lesson15\_Pandas-Reading

February 18, 2020

## 1 Reading Data with Pandas

In the last lesson, we learned how `pandas` stores data as rows and columns in `DataFrames`. We previously used a small dataset that was hard-coded right in the notebook. But in the real world, we want to be able to use large datasets that can't be easily hard-coded or typed out by hand. One way that we can store large datasets as files is in the CSV format. This is a format which can be opened by many different programs like Excel, Google Sheets, or our Python programs, which allows us to share data easily.

Let's start by importing `pandas`. We can use the `pd` nickname like before:

```
[1]: # import the pandas package
import pandas as pd
```

Now we're ready to read our dataset into Python with `pandas`! We'll use a function called `read_csv`. Our dataset is in our GWC GitHub repository, and we need to tell `read_csv` exactly where to find it. `read_csv` will create a `DataFrame` for us. Let's call it `tips`:

```
[2]: # load the tips csv
path = 'https://raw.githubusercontent.com/GWC-DCMB/ClubCurriculum/master/'
tips = pd.read_csv(path + 'SampleData/tips.csv')
```

Since we saved the data to a variable, `pandas` didn't show us what it looks like. How would you view the beginning of the `tips` `DataFrame` without seeing every row? Try it below:

```
[3]: # View just the beginning of the tips DataFrame
tips.head()
```

```
[3]:   total_bill  tip  sex smoker  day  time  size
0      16.99  1.01 Female    No  Sun  Dinner    2
1      10.34  1.66   Male    No  Sun  Dinner    3
2      21.01  3.50   Male    No  Sun  Dinner    3
3      23.68  3.31   Male    No  Sun  Dinner    2
4      24.59  3.61 Female    No  Sun  Dinner    4
```

Look at the column names of the `tips` `DataFrame`. We have `total_bill`, `tip`, `sex`, `smoker`, `day`, `time`, and `size`. Based on the column names and some of the values in the `DataFrame`, what do you think the rows each represent?

**The rows represent:** waiters and waitresses in a restaurant

Now let's take a look at the end of the `DataFrame`:

```
[4]: # View the end of the tips DataFrame
tips.tail()
```

```
[4]:      total_bill  tip    sex smoker  day  time  size
239      29.03  5.92   Male     No   Sat  Dinner    3
240      27.18  2.00  Female     Yes   Sat  Dinner    2
241      22.67  2.00   Male     Yes   Sat  Dinner    2
242      17.82  1.75   Male     No   Sat  Dinner    2
243      18.78  3.00  Female     No  Thur  Dinner    2
```

Notice the numbers on the far left side of the `DataFrame`. `pandas` assigned a number to every row. What number did `pandas` assign to the very first row of the `DataFrame`? (Scroll up if you need to.) So how many rows do we have in this `DataFrame`?

**Number of rows:** 244

The column of numbers that label the rows is called the **index** of the `DataFrame`. The **index** is an **attribute**, a special variable which belongs to variables of the `DataFrame` type. An example of an attribute would be if you had a variable `dog` with an attribute `dog.owner` to store the name of the person who owns the dog.

We can view the `DataFrame`'s **index** like this:

```
[5]: # view the index
tips.index
```

```
[5]: RangeIndex(start=0, stop=244, step=1)
```

So our index starts at 0, ends at 244, and increases by 1 for each row. Another way to count the number of rows is to take the length of the index using the `len` function:

```
[6]: # get the length of the index
len(tips.index)
```

```
[6]: 244
```

Like the **index** labels the rows of the `DataFrame`, there is an **attribute** called **columns** that refers to the columns of the `DataFrame`. Let's take a look:

```
[7]: # view the columns
tips.columns
```

```
[7]: Index(['total_bill', 'tip', 'sex', 'smoker', 'day', 'time', 'size'],
      dtype='object')
```

We could count the number of columns -- there aren't too many -- but what's the fun in that? Let's write a line of code to tell us the number of columns:

```
[8]: # length of the DataFrame's columns
len(tips.columns)
```

[8]: 7

Conveniently, we can also call `len` on the `DataFrame` itself. Try it here! Is the result equal to the number of rows or the number of columns?

```
[9]: # use len on tips
len(tips)
```

[9]: 244

Based on the number of rows and columns, how many data points are in the `tips` `DataFrame`?

```
[10]: # calculate the number of data points in tips
7 * 244
```

[10]: 1708

That's a lot more data than we've handled before. But that's nothing for `pandas` -- it can handle `DataFrames` with *millions* of rows! Data scientists use `pandas` to handle very large datasets from the real world.

Instead of typing the number of rows and columns in the `DataFrame`, we could put both commands with `len` on the same line. Try it here:

```
[11]: # Multiply the length of rows & columns without typing numbers
len(tips) * len(tips.columns)
```

[11]: 1708

This way, if the `tips` data changes, we can quickly re-run the above cell to find the number of values in it, without having to manually type out the number of rows and columns.

You just learned:

- How to read datasets into `pandas` `DataFrames`.
- The `index` and `columns` attributes of `DataFrames`.
- How to find the number of rows, columns, and number of data points in a `DataFrame`.