Lab 04 – pandas II

Submission: `lab04.ipynb`, pushed to your Git repo on `master`.
Points: 10
Due: Monday, January 27, 2pm

Objectives

- More experience with `pandas`

Introduction

Create a **`lab04.ipynb`** file. Create your header cell, then your first cell to set up your imports:

```
import sys
import numpy as np
import pandas as pd
```

As mentioned in the previous lab, a large portion of this lab is taken from snippets scattered throughout the enormous documentation and tutorials from the pandas website at http://pandas.pydata.org/pandas-docs/stable/index.html .

Finally, please take note of the two online O'Reilly books I put on Moodle that are often cited as excellent places to learn numpy and pandas, and to use as a reference later in your work. One book in particular, Python for Data Analysis, is actually written by the developer of pandas, Wes McKinney.

Exercises

To start, create your first cell. Then, place the usual `import` statements at the top of your notebook file.

In the 2nd cell, let's create a bigger version of the quiz `scores` dataset with the following features:

- We have two students, Bob and Jane
- We have four weeks of data. Notice we can have duplicate names in our index. (However, columns must be unique! You can't have duplicate variable names!)
- There are missing quizzes, indicated with a zero in those locations.

Copy the following into your 2nd cell:

```
days = ["Mon","Tue","Wed","Thurs","Fri","Student"]
scores = pd.DataFrame([pd.Series([9.5, 8.75, 8, 10, 7.75, "Bob"], index=days, name="week_1"),
                       pd.Series([9, 8, 10, 8.75, 7.25, "Jane"], index=days, name="week_1"),
                       pd.Series([8.5, 0, 9.75, 9, 6, "Bob"], index=days, name="week_2"),
                       pd.Series([7, 8.25, 9.25, 0, 8, "Jane"], index=days, name="week_2"),
                       pd.Series([7, 8.5, 9.25, 0, 0, "Bob"], index=days, name="week_3"),
                       pd.Series([8.5, 8.25, 9, 8, 7.5, "Jane"], index=days, name="week_3"),
                       pd.Series([9, 7, 8.5, 8, 8, "Bob"], index=days, name="week_4"),
                       pd.Series([9.25, 9.25, 8.5, 7.75, 7.5, "Jane"], index=days,
name="week_4"),
                      ])scores
```

We always start by learning a bit about our data. Generally, we look at the structure first, and then we explore it.

1) [P] Report the shape, the number of dimensions, the size of the data, and the data types of each column. Print each separately. Then create an additional code cell that shows the results of `scores.info()`.

2) [P] Show two different ways to report the number of observations in `scores`.

3) [P] Show the variables (i.e. columns) in `scores`

4) [P] There are repetitive names in the index. This happens quite a bit, depending on how the rows are to be named. For now. print the *unique* index names in `scores`

5) [P] Rename the `'Thurs'` column header to be `'Thu'` . (Hint: Changing `scores.columns[3]` directly will not work, not to mention it would be a poor approach to hard code a location this way! Magic numbers be bad when referencing index or column locations. They may change!) Show the new `scores` data frame. This should be the same, except with **Thurs** changed to **Thu**.

6) [P] Now, make a bigger change. Let's rename `'week_1'` to be `'w1'`, `'week_2'` to be `'w2'`, and so on. Also, suppose you decide to use only 2 letter abbreviations for the days. Rename the days in the column names to be `'Mo'`, `'Tu'`, `'We'`, `'Th'`, `'Fr'`. Your new `scores` should appear as follows:

	Mo	Tu	We	Th	Fr	Student
w1	9.50	8.75	8.00	10.00	7.75	Bob
w1	9.00	8.00	10.00	8.75	7.25	Jane
w2	8.50	0.00	9.75	9.00	6.00	Bob
w2	7.00	8.25	9.25	0.00	8.00	Jane
w3	7.00	8.50	9.25	0.00	0.00	Bob
w3	8.50	8.25	9.00	8.00	7.50	Jane
w4	9.00	7.00	8.50	8.00	8.00	Bob
w4	9.25	9.25	8.50	7.75	7.50	Jane

7) [P] Compare the *type* of the expression `scores['Mo']` vs. `scores[['Mo']]`. What is the difference? (HINT: use the `type()` Python function on both.)

Read about the `describe()` method of data frames. (FYI - This is essentially the equivalent to the `summary()` function in R.)

8) [P] Demonstrate the `describe()` method on `scores`. What type does it return? From your observations of the output, which day had the largest standard deviation in quiz scores? (You can just hard code the answer to this question. You'll write code to do this next.)

9) [P] While you could look at the output of `describe()` and infer information, it's more important to write code to output the specific results you are looking for. Write the Python code that shows the largest standard deviation and the day it occurred without using `describe()`.

OK. Take a moment and look over that output. If you've been doing everything correctly, then that result should be **~4.0372 occurring on Thursday**, and that is way too large! Think back to your stats course. What does standard deviation mean? It's the deviation from the mean! That's a large spread in the data! Why? There are two missing values represented by 0 that are being used to compute the deviation, and that's not what we want. Missing data are usually not included with summary statistics (and many other calculations!) We need to deal with missing data properly. So, instead, when we have missing values in data, we almost always use a **`na.nan`** value in its place to

represent that the data is missing in that specific cell.

10) [P] Write the code that changes all `0.0` entries to `np.nan`. (NOTE: You should be able to do this with just one line of code using `pandas` selection techniques!) Then, show `scores`. All 0.0 entries should be replaced with NaN in the output.

At this point, now you need to start paying attention to NaN values in your data. These represent missing values, and are important to ignore in most calculations, especially preliminary EDA. We will learn about many techniques for dealing with missing values soon. Many functions will have an optional parameter such as `skipna` to make sure that whatever you are trying to do does not include these missing data. Pay attention to those parameters. Most of the time, the default parameter values for handling NaN values is to skip or ignore them.

11) [P] Show the output of `describe()` again. Yes, this is better! Note how much more informative this output is when properly dealing with missing data. Copy your code from above to output the day with the largest standard deviation in quiz scores. You should have a substantially different result than you did before!

12) [P] Show the output of `scores.values`. What does the `.values` attribute do?

13) [P] What is the mean quiz score for each day? Do not use `describe()`

14) [P] What is the mean quiz score over the **entire** dataset? (HINT: there are many ways to do this, and none should involve writing any loops! The only right answer is 8.375. If you aren't getting that, then you are not handling NaN values correctly. You only want the mean over all scores that are NOT NaN!)

15) [P] Now, show the mean for each week over all students. Again, you are not counting missing quizzes into the average. (Yes, technically you could write a loop to compute this exhaustively. However, you should rarely need to do that. Use pandas!)

    You have the correct result if you report the following (shown with 5 places of significant digits):

```
Mean for each week
w1    8.70000
w2    8.21875
w3    8.25000
w4    8.27500
```

16) [P] Select the data frame for only `Monday` and `Student` variables using `.loc`, and again using `.iloc`

17) [P] Show a new DataFrame containing the week as an index (like the original), but the mean of their scores and the count of the quizzes they took. Your result should look as follows:

	Student	count	mean
w1	Bob	5	8.8000
w1	Jane	5	8.6000
w2	Bob	4	8.3125
w2	Jane	4	8.1250
w3	Bob	3	8.2500
w3	Jane	5	8.2500
w4	Bob	5	8.1000
w4	Jane	5	8.4500

18) [P] Show the number of times Friday's score was greater than 7.25 for each student. Start by selecting your data with the `[ ]` operator. Your result should look as follows:

```
Jane    3
Bob     2
Name: Student, dtype: int64
```

19) [P] Use the `where()` method on `scores` to repeat the previous exercise. (HINT: You might need to use `dropna()`). You should have the same output

20) [P] Select the scores that were greater than the mean score over the entire dataset (without the missing values). Again, no hard coding values here (i.e. do NOT hard code the mean, but recompute it. Then, report the number of scores for each day that exceeded that global mean. Your result should look like:

```
Mo    6
Tu    3
We    7
Th    3
Fr    0
dtype: int64
```

This suggests that Wednesday seems to have the best quiz performance, whereas Friday… not so much.

Deliverables

Commit and push `lab04.ipynb`. Be sure you have every cell run, and output generated. Verify that your file is pushed properly on Gitlab.