Lab 04 - pandas II

Name: Nick Satriano Class: CSCI 349 - Intro to Data Mining Semester: Spring 2023 Instructor: Brian King

In [151…
```
import sys
import numpy as np
import pandas as pd

print("Python version: ", sys.version)
print("Numpy version: ", np.__version__)
print("Pandas version: ", pd.__version__)
```

```
Python version:  3.9.16 (main, Jan 11 2023, 16:16:36) [MSC v.1916 64 bit (AMD64)]
Numpy version:  1.23.5
Pandas version:  1.4.4
```

In [152…
```
days = ["Mon","Tues","Wed","Thu","Fri","Student"]

scores = pd.DataFrame([pd.Series([8.75, 9.5, 8, 10, 7.75, "Bob"], index=days, name="we
scores
```

Out[152]:

	Mon	Tues	Wed	Thu	Fri	Student
week_1	8.75	9.50	8.00	10.00	7.75	Bob
week_1	8.00	9.00	10.00	8.75	7.25	Jane
week_2	0.00	8.00	9.75	9.00	6.00	Bob
week_2	8.25	7.00	0.00	9.25	8.00	Jane
week_3	8.50	7.00	9.25	0.00	0.00	Bob
week_3	8.25	8.25	0.00	8.00	7.50	Jane
week_4	6.50	8.75	8.50	8.00	6.00	Bob
week_4	10.00	9.25	8.50	7.75	7.50	Jane

1) 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(). Comment on what the info() method does on data frames.

In [153…
```
print("Shape: ", scores.shape)
print("Number of Dimensions: ", scores.ndim)
print("Size: ", scores.size)
```

```
Shape:  (8, 6)
Number of Dimensions:  2
Size:  48
```

In [154…
```
# The info() method prints a description of the dimensions as well as the types and lo
scores.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 8 entries, week_1 to week_4
Data columns (total 6 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   Mon      8 non-null      float64
 1   Tues     8 non-null      float64
 2   Wed      8 non-null      float64
 3   Thu      8 non-null      float64
 4   Fri      8 non-null      float64
 5   Student  8 non-null      object 
dtypes: float64(5), object(1)
memory usage: 448.0+ bytes
```

2) Show two different ways to report the number of observations in scores.

In [155…
```
print(len(scores))
print(scores.shape[0])
```

```
8
8
```

3) Show the variables (i.e. columns) in scores.

In [156…
```
scores[:0]
```

Out[156]:

Mon	Tues	Wed	Thu	Fri	Student

4) There are repetitive names in the index. This happens quite a bit, depending on how the data is organized. For now, print the unique index names in scores

In [157…
```
scores.index.unique()
```

Out[157]:
```
Index(['week_1', 'week_2', 'week_3', 'week_4'], dtype='object')
```

5) Rename the 'Tues' column header to be 'Tue'. Show the new scores data frame. This should be the same, except with Tues changed to Tue.

In [158…
```
# Citation: chatGPT
scores.rename(columns = {"Tues": "Tue"}, inplace=True)
scores
```

Out[158]:

	Mon	Tue	Wed	Thu	Fri	Student
week_1	8.75	9.50	8.00	10.00	7.75	Bob
week_1	8.00	9.00	10.00	8.75	7.25	Jane
week_2	0.00	8.00	9.75	9.00	6.00	Bob
week_2	8.25	7.00	0.00	9.25	8.00	Jane
week_3	8.50	7.00	9.25	0.00	0.00	Bob
week_3	8.25	8.25	0.00	8.00	7.50	Jane
week_4	6.50	8.75	8.50	8.00	6.00	Bob
week_4	10.00	9.25	8.50	7.75	7.50	Jane

6) 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'. Show the updated scores dataframe.

In [159…

```
scores.rename(index = {"week_1": "w1", "week_2": "w2", "week_3": "w3", "week_4": "w4"}

scores.rename(columns = {"Mon": "Mo", "Tue": "Tu", "Wed": "We", "Thu": "Th", "Fri": "F

scores
```

Out[159]:

	Mo	Tu	We	Th	Fr	Student
w1	8.75	9.50	8.00	10.00	7.75	Bob
w1	8.00	9.00	10.00	8.75	7.25	Jane
w2	0.00	8.00	9.75	9.00	6.00	Bob
w2	8.25	7.00	0.00	9.25	8.00	Jane
w3	8.50	7.00	9.25	0.00	0.00	Bob
w3	8.25	8.25	0.00	8.00	7.50	Jane
w4	6.50	8.75	8.50	8.00	6.00	Bob
w4	10.00	9.25	8.50	7.75	7.50	Jane

7) Compare the type of the expression scores['Mo'] vs. scores[['Mo']]. What is the difference?

In [160…

```
print(type(scores["Mo"]))
print(type(scores[["Mo"]]))
```

```
<class 'pandas.core.series.Series'>
<class 'pandas.core.frame.DataFrame'>
```

For the first expression, a Series is returned whereas for the second, a DataFrame is returned.

8) Demonstrate the describe() method on scores. What type does it return?

In [161…
```
print(scores.describe())

print(type(scores.describe()))
```

```
              Mo         Tu          We          Th         Fr
count   8.000000   8.000000   8.000000   8.000000   8.000000
mean    7.281250   8.343750   6.750000   7.593750   6.250000
std     3.095035   0.963045   4.219428   3.159276   2.635608
min     0.000000   7.000000   0.000000   0.000000   0.000000
25%     7.625000   7.750000   6.000000   7.937500   6.000000
50%     8.250000   8.500000   8.500000   8.375000   7.375000
75%     8.562500   9.062500   9.375000   9.062500   7.562500
max    10.000000   9.500000  10.000000  10.000000   8.000000
<class 'pandas.core.frame.DataFrame'>
```

9) Store the output of describe() as a variable. Then, using this data frame, report the day that had the largest standard deviation in quiz scores and its value. Clean your temporary variables when complete (i.e. use del)

In [162…
```
x = scores.describe()

print(x.loc["std"].idxmax(), "had the highest standard deviation: ", x.loc["std"].max(
del x
```

```
We had the highest standard deviation:  4.219427855595049
```

10) Write the Python code to repeat the previous exercise without using describe().

In [163…
```
x = scores.std(numeric_only = True).idxmax()
print(x, "had the highest standard deviation: ", scores.std(numeric_only = True).max()
del x
```

```
We had the highest standard deviation:  4.219427855595049
```

11) 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.

In [164…
```
scores.replace(0, np.nan, inplace=True)

scores
```

Out[164]:

	Mo	Tu	We	Th	Fr	Student
w1	8.75	9.50	8.00	10.00	7.75	Bob
w1	8.00	9.00	10.00	8.75	7.25	Jane
w2	NaN	8.00	9.75	9.00	6.00	Bob
w2	8.25	7.00	NaN	9.25	8.00	Jane
w3	8.50	7.00	9.25	NaN	NaN	Bob
w3	8.25	8.25	NaN	8.00	7.50	Jane
w4	6.50	8.75	8.50	8.00	6.00	Bob
w4	10.00	9.25	8.50	7.75	7.50	Jane

12) Show the output of describe() again.

In [165…
```
print(scores.describe())

print(scores.std(numeric_only = True).idxmax(), "had the highest standard deviation: "
```

```
              Mo        Tu         We         Th        Fr
count   7.000000  8.000000   6.000000   7.000000  7.000000
mean    8.321429  8.343750   9.000000   8.678571  7.142857
std     1.037970  0.963045   0.790569   0.812843  0.814672
min     6.500000  7.000000   8.000000   7.750000  6.000000
25%     8.125000  7.750000   8.500000   8.000000  6.625000
50%     8.250000  8.500000   8.875000   8.750000  7.500000
75%     8.625000  9.062500   9.625000   9.125000  7.625000
max    10.000000  9.500000  10.000000  10.000000  8.000000
Mo had the highest standard deviation:  1.0379696297970151
```

13) Show the output of scores.values. What does the .values attribute do?

In [166…
```
print(scores.values)
```

```
[[8.75 9.5 8.0 10.0 7.75 'Bob']
 [8.0 9.0 10.0 8.75 7.25 'Jane']
 [nan 8.0 9.75 9.0 6.0 'Bob']
 [8.25 7.0 nan 9.25 8.0 'Jane']
 [8.5 7.0 9.25 nan nan 'Bob']
 [8.25 8.25 nan 8.0 7.5 'Jane']
 [6.5 8.75 8.5 8.0 6.0 'Bob']
 [10.0 9.25 8.5 7.75 7.5 'Jane']]
```

The .values attribute returns each observation in the DataFrame.

14) What is the mean quiz score for each day? Do not use describe().

In [167…
```
scores.mean(axis=0, numeric_only = True)
```

Out[167]:
```
Mo    8.321429
Tu    8.343750
We    9.000000
Th    8.678571
Fr    7.142857
dtype: float64
```

15) What is the mean quiz score over the entire dataset, ignoring all missing values?

In [168…
```
scores.mean(numeric_only = True).mean()
```

Out[168]:
```
8.29732142857143
```

16) Show the mean for each week over all students.

In [169…
```
# Citation: chatGPT
scores.groupby(by = scores.index).mean(numeric_only = True).mean(axis = 1)
```

Out[169]:
```
w1    8.700
w2    8.325
w3    8.150
w4    8.075
dtype: float64
```

17) Show the mean score for each student. Again, ignore all missing values.

In [170…
```
scores.groupby(by = scores.Student).mean(numeric_only = True).mean(axis = 1)
```

Out[170]:
```
Student
Bob     8.1375
Jane    8.4500
dtype: float64
```

18) Select the data that includes only data for Monday and the Student variables using .loc, and again using .iloc

In [171…
```
scores.loc[:, ["Mo", "Student"]]
```

Out[171]:

	Mo	Student
w1	8.75	Bob
w1	8.00	Jane
w2	NaN	Bob
w2	8.25	Jane
w3	8.50	Bob
w3	8.25	Jane
w4	6.50	Bob
w4	10.00	Jane

In [172…
```
scores.iloc[:, [0, 5]]
```

Out[172]:

	Mo	Student
w1	8.75	Bob
w1	8.00	Jane
w2	NaN	Bob
w2	8.25	Jane
w3	8.50	Bob
w3	8.25	Jane
w4	6.50	Bob
w4	10.00	Jane

19) Show a new DataFrame containing the week as an index (like the original), but contains only each student with the count of their quizzes and the mean of their scores.

In [225…

```
new_scores = pd.DataFrame(scores.loc[:, ["Student"]])
new_scores["count"] = scores.count(axis = 1, numeric_only = True)
new_scores["mean"] = scores.mean(axis = 1, numeric_only = True)
new_scores
```

Out[225]:

	Student	count	mean
w1	Bob	5	8.8000
w1	Jane	5	8.6000
w2	Bob	4	8.1875
w2	Jane	4	8.1250
w3	Bob	3	8.2500
w3	Jane	4	8.0000
w4	Bob	5	7.5500
w4	Jane	5	8.6000

20) This time, show the total number of quizzes taken each week by all students.

In [226…

```
new_scores.groupby(by = new_scores.index).sum(numeric_only = True)
```

Out[226]:

	count	mean
w1	10	17.4000
w2	8	16.3125
w3	7	16.2500
w4	10	16.1500

21) Show the number of times Friday's score was <= 7.5 for each student. Start by selecting a subset of observations, then use value_counts() on the Student variable of the selected data.

In [227…
```
x = scores.loc[:, ["Student", "Fr"]]
x[x["Fr"] <= 7.5].Student.value_counts()
```

Out[227]:
```
Jane    3
Bob     2
Name: Student, dtype: int64
```

22) Repeat the previous exercise, but this time use the where() method on scores to select your data. You should have the same output.

In [256…
```
scores.where(scores["Fr"] <= 7.5).Student.value_counts()
```

Out[256]:
```
Jane    3
Bob     2
Name: Student, dtype: int64
```

23) Select the scores that were greater than the mean score over the entire dataset (without the missing values). Then, report the number of scores for each day that exceeded that global mean.

In [306…
```
avg = (scores.mean(numeric_only = True).mean())

scores.where(scores.iloc[:,0:5] > avg).count()
```

Out[306]:
```
Mo          3
Tu          4
We          5
Th          4
Fr          0
Student     0
dtype: int64
```

24) Sometimes when we deal with enormous datasets for modeling, we drop all observations (rows!) that have any missing data. Show a new data frame that has only complete observations.

In [295…
```
new_df = scores.dropna()

new_df
```

Out[295]:

	Mo	Tu	We	Th	Fr	Student
w1	8.75	9.50	8.0	10.00	7.75	Bob
w1	8.00	9.00	10.0	8.75	7.25	Jane
w4	6.50	8.75	8.5	8.00	6.00	Bob
w4	10.00	9.25	8.5	7.75	7.50	Jane