Lab 03 - pandas I

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

In [2]:
```
import sys
import numpy as np
import pandas as pd
```

1) Report the Python, Numpy and Pandas version numbers.

In [4]:
```
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
```

2) Now, show the result of pd.show_versions(). Write a comment in the cell about what this shows.

In [6]:
```
pd.show_versions()
# This function shows the versions of all dependencies used by pandas.
```

```
INSTALLED VERSIONS
------------------
commit           : ca60aab7340d9989d9428e11a51467658190bb6b
python           : 3.9.16.final.0
python-bits      : 64
OS               : Windows
OS-release       : 10
Version          : 10.0.22621
machine          : AMD64
processor        : AMD64 Family 25 Model 80 Stepping 0, AuthenticAMD
byteorder        : little
LC_ALL           : None
LANG             : None
LOCALE           : English_United States.1252

pandas           : 1.4.4
numpy            : 1.23.5
pytz             : 2022.7
dateutil         : 2.8.2
setuptools       : 65.6.3
pip              : 22.3.1
Cython           : 0.29.32
pytest           : 7.1.2
hypothesis       : None
sphinx           : 5.0.2
blosc            : None
feather          : None
xlsxwriter       : 3.0.3
lxml.etree       : 4.9.1
html5lib         : None
pymysql          : None
psycopg2         : None
jinja2           : 2.11.3
IPython          : 7.31.1
pandas_datareader: None
bs4              : 4.11.1
bottleneck       : 1.3.5
brotli           :
fastparquet      : None
fsspec           : 2022.11.0
gcsfs            : None
markupsafe       : 2.0.1
matplotlib       : 3.6.2
numba            : 0.56.4
numexpr          : 2.8.4
odfpy            : None
openpyxl         : 3.0.10
pandas_gbq       : None
pyarrow          : None
pyreadstat       : None
pyxlsb           : None
s3fs             : None
scipy            : 1.9.3
snappy           :
sqlalchemy       : 1.4.39
tables           : 3.7.0
tabulate         : 0.8.10
xarray           : 2022.11.0
xlrd             : 2.0.1
```

```
xlwt            : None
zstandard       : 0.18.0
```

3) Explain the relationship between numpy and pandas. How are they tied together? How are they different? What core functionality does pandas add to numpy?

pandas contains data structures and handling tools while NumPy provides numerical computing tools. Though they are different in concept, pandas utilizes an array-based input much like NumPy. However, pandas is designed for handling heterogenous data, whereas NumPy is best used for strictly numerical arrays. pandas allows NumPy arrays to contain special data types by wrapping them as a PandasArray.

4) Compare and contrast Series and DataFrame.

A Series is a one-dimensional array which can contain any data type. Series are indexed by their axis labels. A DataFrame is a 2-dimensional, labeled data structure that can contain columns of different types.

5) What are the data types that can be used to create a Series object in pandas?

Python dictionaries, ndarrays, and scalar values can be used to create a Series object in pandas.

6) What are the data structures that can be used to create a DataFrame object in pandas?

A DataFrame in pandas can be created from: a dictionary of 1-D ndarrays, a 2-D numpy.ndarray, a structured ndarray, a Series, or another DataFrame.

7) When creating a Series object, what role does the index parameter play? Does the index always need to be specified? If not, what happens?

The index parameter is a list of axis labels for the Series. The index does not always need to be specified, and if it is not, then one is automatically created containing values [0, ..., len(data) - 1].

8) Create a numpy array of 10 random 8-bit integers in the range [10,20) using numpy's randint function. Print the type of x, and show x.

In [26]:
```
x = np.array(np.random.randint(10, high = 20, dtype = np.int8, size = 10))
print(type(x))
x
```

Out[26]:
```
<class 'numpy.ndarray'>
array([14, 15, 12, 11, 12, 13, 14, 10, 14, 13], dtype=int8)
```

In [29]:
```
x2 = pd.Series(x, index=[0, 10, 20, 30, 40, 50, 60, 70, 80, 90], name = "x2")
print(type(x2))
x2
```

Out[29]:
```
<class 'pandas.core.series.Series'>
0     14
10    15
20    12
30    11
40    12
50    13
60    14
70    10
80    14
90    13
Name: x2, dtype: int8
```

9) Create a DataFrame object called df_x that has x2 as the first variable with the name of "x2", and an additional variable named "x3" that is 10 random floating point numbers in the range [100,200). Show the types of your dataframe using print(df_x.dtypes), and show the contents of your dataframe. Your index should remain as 0, 10, 20, ... , 90.

In [42]:
```
df_x = pd.DataFrame(x2)
x3 = np.array(np.random.randint(100, high = 200, size = 10), dtype=float)
x3 = pd.Series(x3)
df_x["x3"] = x3.values
print(df_x.dtypes)
df_x
```

```
x2       int8
x3    float64
dtype: object
```

Out[42]:

	x2	x3
0	14	156.0
10	15	124.0
20	12	161.0
30	11	131.0
40	12	199.0
50	13	182.0
60	14	126.0
70	10	188.0
80	14	124.0
90	13	139.0

10) Consider the copy() method of a DataFrame object. What is this method for? Explain the difference between a shallow copy and a deep copy of a DataFrame? Why would I use a shallow copy?

The copy method is used to make a copy of the DataFrame object. If a shallow copy is specified while calling copy(), then a new DataFrame object is created without copying the original object's data or index. If copy() is specified as a deep copy, then the new object will be created

using the original object's data and indices. We would want to use a shallow copy when we want to manipulate the copied data without changing the original data.

In [43]:
```
days = ["Mon","Tue","Wed","Thu","Fri"]
scores_1 = [9.5, 8.75, 8, 10, 7.75]
scores_2 = [9, 8, 10, 8.75, 7.25]
```

11) Convert scores_1 and scores_2 into two Series objects. Use days as your index. You should name each Series object using the name parameter as "week_1", and "week_2". Do not show the result, only store the variables.

In [44]:
```
scores_1 = pd.Series(scores_1, index = days, name = "week_1")
scores_2 = pd.Series(scores_2, index = days, name = "week_2")
```

12) Create a pandas DataFrame called scores that represents the above data. Show your data frame.

In [56]:
```
scores = pd.DataFrame([scores_1, scores_2])
scores
```

Out[56]:

	Mon	Tue	Wed	Thu	Fri
week_1	9.5	8.75	8.0	10.00	7.75
week_2	9.0	8.00	10.0	8.75	7.25

13) Use the pd.concat function to add the following data to the end of scores. [8.5,8,9.75,9,8.25]. The row label on the new week is "week_3". Show your updated data frame.

In [58]:
```
# Citation: https://pandas.pydata.org/docs/reference/api/pandas.concat.html
scores_3 = pd.DataFrame([[8.5, 8, 9.75, 9, 8.25]], columns = days, index = ["week_3"])
scores = pd.concat([scores, scores_3], axis = 0)
scores
```

Out[58]:

	Mon	Tue	Wed	Thu	Fri
week_1	9.5	8.75	8.00	10.00	7.75
week_2	9.0	8.00	10.00	8.75	7.25
week_3	8.5	8.00	9.75	9.00	8.25

14) Write yourself a quick one sentence reference for each access method listed, with one example for each using the scores DataFrame.

.at: Access a single value for a specified row/column label pair.

In [60]:
```
# Citation: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.at.html
scores.at["week_1", "Mon"]
```

Out[60]:
```
9.5
```

.iat: Access a single value for a row/column pair by integer position.

In [61]:
```
# Citation: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.iat.html
scores.iat[2, 3]
```

Out[61]:
```
9.0
```

.loc: Access a group of rows and columns by label(s) or a boolean array.

In [63]:
```
# Citation: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.loc.html
scores.loc["week_2"]
```

Out[63]:
```
Mon     9.00
Tue     8.00
Wed    10.00
Thu     8.75
Fri     7.25
Name: week_2, dtype: float64
```

.iloc: Purely integer-location based indexing for selection by position.

In [64]:
```
# Citation: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.iloc.html
scores.iloc[2]
```

Out[64]:
```
Mon    8.50
Tue    8.00
Wed    9.75
Thu    9.00
Fri    8.25
Name: week_3, dtype: float64
```

In [65]:
```
scores.iloc[0]
```

Out[65]:
```
Mon     9.50
Tue     8.75
Wed     8.00
Thu    10.00
Fri     7.75
Name: week_1, dtype: float64
```

In [66]:
```
scores.iloc[0:1]
```

Out[66]:

	Mon	Tue	Wed	Thu	Fri
week_1	9.5	8.75	8.0	10.0	7.75

In [67]:
```
scores.iloc[:,0:1]
```

Out[67]:

	Mon
week_1	9.5
week_2	9.0
week_3	8.5

In [68]:
```
scores[0:1]
```

Out[68]:

	Mon	Tue	Wed	Thu	Fri
week_1	9.5	8.75	8.0	10.0	7.75

In [69]:
```
scores.Mon
```

Out[69]:
```
week_1    9.5
week_2    9.0
week_3    8.5
Name: Mon, dtype: float64
```

In [70]:
```
scores["Mon"]
```

Out[70]:
```
week_1    9.5
week_2    9.0
week_3    8.5
Name: Mon, dtype: float64
```

In [71]:
```
scores.loc[:,"Mon"]
```

Out[71]:
```
week_1    9.5
week_2    9.0
week_3    8.5
Name: Mon, dtype: float64
```

In [72]:
```
scores.loc[:,["Mon"]]
```

Out[72]:

	Mon
week_1	9.5
week_2	9.0
week_3	8.5

15) Show at least two different techniques to select scores for Tuesday using the string "Tue" as a Series.

In [84]:
```
scores["Tue"]
```

Out[84]:
```
week_1    8.75
week_2    8.00
week_3    8.00
Name: Tue, dtype: float64
```

In [85]:
```
scores.loc[:, "Tue"]
```

Out[85]:
```
week_1    8.75
week_2    8.00
week_3    8.00
Name: Tue, dtype: float64
```

16) Show how to retrieve the scores for Tuesday using the named attribute Tue.

In [79]:
```
scores.Tue
```

Out[79]:
```
week_1    8.75
week_2    8.00
week_3    8.00
Name: Tue, dtype: float64
```

17) Show at least three techniques to select scores for Wednesday using an integer. The return type can be either a DataFrame or a Series.

In [97]:
```
scores.iloc[:, 2:3]
```

Out[97]:

	Wed
week_1	8.00
week_2	10.00
week_3	9.75

In [98]:
```
# Citation: ChatGPT
scores.columns[2]
```

Out[98]:
```
'Wed'
```

In [96]:
```
# Citation: ChatGPT
scores.columns.values[2]
```

Out[96]:
```
'Wed'
```

18) Select the data for the first week using the string "week_1". Your result should return a Series representing the scores for week 1.

In [102…
```
scores.loc["week_1"]
```

Out[102]:
```
Mon     9.50
Tue     8.75
Wed     8.00
Thu    10.00
Fri     7.75
Name: week_1, dtype: float64
```

19) Select the data for the first week using the string "week_1". Your result should return a DataFrame, representing the subset of the scores DataFrame for week 1 only.

In [103…
```
scores.loc[["week_1"]]
```

Out[103]:

	Mon	Tue	Wed	Thu	Fri
week_1	9.5	8.75	8.0	10.0	7.75

20) Select the data for the first week using a slice.

In [104…
```
scores.loc[:"week_1"]
```

Out[104]:

	Mon	Tue	Wed	Thu	Fri
week_1	9.5	8.75	8.0	10.0	7.75

21) Use .iloc to select Monday and Friday of the first and third week.

In [106…
```
# Citation: ChatGPT
scores.iloc[[0,2], [0,4]]
```

Out[106]:

	Mon	Fri
week_1	9.5	7.75
week_3	8.5	8.25

22) Repeat the previous exercise, but use the .loc selector. You should have the same result.

In [116…
```
scores.loc[["week_1", "week_3"], ["Mon", "Fri"]]
```

Out[116]:

	Mon	Fri
week_1	9.5	7.75
week_3	8.5	8.25

23) Report the mean score for each day.

In [108…
```
scores.mean()
```

Out[108]:
```
Mon    9.00
Tue    8.25
Wed    9.25
Thu    9.25
Fri    7.75
dtype: float64
```

24) Report the mean score for each week

In [115…
```
# Citation: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.mean.html
scores.mean(axis = 1)
```

Out[115]:
```
week_1    8.8
week_2    8.6
week_3    8.7
dtype: float64
```

25) For each week, report how much each day's score for that week differed from the mean for the week.

In [117…
```
mean_scores = scores.mean(axis = 1)
diff_scores = scores.sub(mean_scores, axis = 0)
diff_scores
```

Out[117]:

	Mon	Tue	Wed	Thu	Fri
week_1	0.7	-0.05	-0.80	1.20	-1.05
week_2	0.4	-0.60	1.40	0.15	-1.35
week_3	-0.2	-0.70	1.05	0.30	-0.45

26) Report the maximum score for each week. Your result should be a Series.

In [118…
```
scores.max(axis = 1)
```

Out[118]:
```
week_1    10.00
week_2    10.00
week_3     9.75
dtype: float64
```

27) For each week, report which day had the largest score. Again, you should be reporting your result as a Series.

In [119…
```
# Citation: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.idxmax.html
scores.idxmax(axis = 1)
```

Out[119]:
```
week_1    Thu
week_2    Wed
week_3    Wed
dtype: object
```

28) Report the week that had the highest total quiz score. Your answer should only be the name of the week from the index.

In [159…
```
scores.max(axis=1).idxmax()
```

Out[159]:
```
'week_1'
```

29) How many days over the entire dataset had a score >= 9.25?

In [163…
```
# Citation: chatGPT
(scores >= 9.25).sum().sum()
```

Out[163]:
```
4
```

30) Report the number of days of each week that had a score >= 9.25.

In [167…
```
(scores >= 9.25).sum(axis = 1)
```

Out[167]:
```
week_1    2
week_2    1
week_3    1
dtype: int64
```

31) Report the average score for each week with the lowest score for each week dropped.

In [184…
```
(scores.sum(axis = 1) - scores.min(axis = 1) ) / (scores.shape[1] - 1)
```

Out[184]:
```
week_1    9.0625
week_2    8.9375
week_3    8.8750
dtype: float64
```

32) Report the scores rescaled to fall between 0 and 100, instead of 0 to 10 as they are now.

In [185…
```
scores * 10
```

Out[185]:

	Mon	Tue	Wed	Thu	Fri
week_1	95.0	87.5	80.0	100.0	77.5
week_2	90.0	80.0	100.0	87.5	72.5
week_3	85.0	80.0	97.5	90.0	82.5

33) How would you convert your scores DataFrame to a Numpy array? Demonstrate it by showing the type of your conversion using Python's type function.

In [187…
```
type(scores.values)
```

Out[187]:
```
numpy.ndarray
```