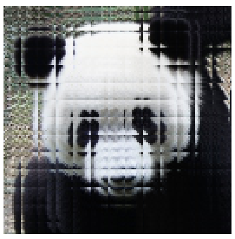
ITCS159 Software Lab for Basic Scientific Problem Solving
Lab 7: Python library for relational table manipulation (Pandas)

September 20, 2021



Objective

This lab aims to describe what *Pandas* is (This lab has nothing to do with the cute panda bears). Python is a great language for doing data analysis. Pandas is one of the Python's packages that makes importing and analyzing data much easier. Pandas builds on packages like NumPy and matplotlib that provides convenient solutions for data analysts. The word pandas is an acronym which is derived from "Python and data analysis" and "panel data".

Python pandas is well suited for different kinds of data, such as:

- Tabular data with heterogeneously-typed columns
- Ordered and unordered time series data
- Arbitrary matrix data with row and column labels
- Unlabelled data
- Statistical data sets

Note that:

- Each lab consists of a number of practical exercises, all of which are required for marks to be attained. In particular, you should try to accomplish the 3 labelled milestones.

- Attendance is compulsory and will be monitored on Mycourse. If you are unable to attend a session you must inform a tutor in advance.
- You can finish all milestones before calling for marking.

During the marking, your understandings will be tested, and the tutor will not give the marks if you do not display an understanding of the problem and its solution. The tutors are there to help you think through any issues — but they won't write the programs for you!

Pandas installation

Open the Anaconda Prompt and remember to run it as the administrator. To install Python Pandas, type `conda install pandas`. In macOS, you can use `pip install pandas`. Once the installation is completed, go to your Jupyter and simply import it by typing: `import pandas as pd`.

Before start coding:

Open the Jupyter Notebook Application. Then, go to Desktop and create folder here named `Lab7`.

Click `New` > `Python3` with name `lab07`. Now you can start the following exercise.

Introduction to Pandas

Pandas is an open-source, BSD-licensed Python library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language. Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, statistics, analytics, etc.

Pandas has three data structures: (These data structures are built on top of Numpy array, which means they are fast.)

- Series
- DataFrame
- Panel

pandas.Series

Series is a one-dimensional array like structure with homogeneous data. For example, the following series is a collection of integers 10, 23, 56, … Let's create Series using the following code.

```
import pandas as pd #import the pandas and aliasing as pd
import numpy as np
data = np.array(['a','b','c','d'])
s1 = pd.Series(data)
print(s1)
```

```
data = np.array([1,2,3,4])
s2 = pd.Series(data)
print(s2)
```

What are the results? What are the differences between `s1` and s2?

Now, it is your turn to create an heterogeneously-typed series. For example, `[1, ‘b’,3, ‘d’]`. What is a data type of the heterogeneously-typed series when you print out the created series?

We can also alter the index assigned to elements using the following code.

```
data = np.array([‘a’,‘b’,‘c’,‘d’])
s = pd.Series(data,index=[100,101,102,103])
print(s)
```

Can you notice that the index is changed?
To access the data from Series, we can use the following pattern.

```
s = pd.Series([1,2,3,4,5],index = [‘a’,‘b’,‘c’,‘d’,‘e’])
print(s[0])
```

Exercise: Write a code to retrieve (show) only the first three elements in Series `s`.

Answer:

Exercise: Let try to use an index (e.g. `‘a’`) to access the element in Series `s`, e.g. `s[[‘a’]]`

Answer:

Note that you can use the following code to retrieve multiple elements using a list of index label values.

```
#retrieve multiple elements
print(s[[‘a’,‘c’,‘d’]])
```

pandas.DataFrame

A *Data frame* is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns.

Use the following code to create an empty data frame:

```
import pandas as pd
df = pd.DataFrame()
print(df)
```

You will see the output as Empty DataFrame. There are so many methods to create a DataFrame. We will try some of them.

Create a DataFrame from Lists

The DataFrame can be created using a single list or a list of lists.

```
data = [1,2,3,4,5]
df1 = pd.DataFrame(data)
print(df1)

data = [['Alex',10],['Bob',12],['Clarke',13]]
df2 = pd.DataFrame(data,columns=['Name','Age'])
print(df2)
```

Create a DataFrame from Dict of ndarrays / Lists

All the ndarrays must be of same length. If index is passed, then the length of the index should equal to the length of the arrays.

```
data = {'Name':['Tom', 'Jack', 'Steve', 'Ricky'],
         'Age':[28,34,29,42]}
df = pd.DataFrame(data)
print(df)
```

Create a DataFrame from Dict of Series

Dictionary of Series can be also passed to form a DataFrame.

```
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
     'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'
        ])}
df = pd.DataFrame(d)
print(df)
```

Notice that we have `NaN`, why?
Exercise: now, this is your turn to create the following table as Dataframe and name this table as `studentDF`.

Table 1: `studentDF` Dataframe

	Math	C_programming	Java	Web_programming
0	90	70	50	55
1	10	20	41	66
2	34	25	67	23
3	23	32	34	76
4	43	73	23	67

Answer:

Row Selection, Addition, and Deletion

According to the Dataframe that you have created in the previous exercise, let try the following code and see the output. You can replace `df` with `studentDF` to see the results.

```
print(df)
print(df[2:4])  #Slice Rows

print(df.loc[0])

print(df.loc[2])

print(df.loc[0:2])
```

Note that rows can be selected by passing row label to a `loc` function.

To add new rows to a DataFrame, you can use the `append` function. This function will append the rows at the end.

```
df = pd.DataFrame([[1, 2], [3, 4]], columns = ['a','b'])
df2 = pd.DataFrame([[5, 6], [7, 8]], columns = ['a','b'])

df = df.append(df2)
print(df)
```

Exercise: use `append` to add the following table to table `studentDF` that you have created in the previous exercise.

Table 2: New rows of `studentDF`

	Math	C_programming	Java	Web_programming
5	59	37	84	82
6	41	34	73	55

Note that we need the index of 5 and 6 for the new rows.

Answer:

To delete rows, you can use index label to delete or drop rows from a DataFrame. If label is duplicated, then multiple rows will be dropped. See the following code

```
# Drop rows with label 0
df = df.drop(0)

print(df)
```

Exercise: use `drop` function to remove the last row from table `studentDF`.

Answer:

For Panal, it is 3D container of data. In other words, Panel can group multiple Dataframe objects together and those Dataframes can be referred by indexing techniques. However, this lab focus on only Dataframe.

Milestone 1: You have now completed the Milestone 1. Keep going!

Descriptive Statistics

A large number of methods collectively compute descriptive statistics and other related operations on DataFrame. In this practice, you will explore those functions and apply it on table `studentDF`.

See the following code, it shows how to use `sum()` function.

```
data = {'Name':['Tom', 'Jack', 'Steve', 'Ricky'],
         'Age':[28,34,29,42]}
df = pd.DataFrame(data)
df.sum()
```

Exercise: try to apply `sum()` with table `studentDF`. What is the result?

Answer:

Exercise: try to use `axis` attribute to change the orientation of the summation.

Answer:

The following table list down the important functions.

Function	Description
count()	Number of non-null observations
sum()	Sum of values
mean()	Mean of Values
median()	Median of Values
mode()	Mode of values
std()	Standard Deviation of the Values
min()	Minimum Value
max()	Maximum Value
abs()	Absolute Value
prod()	Product of Values
cumsum()	Cumulative Sum
cumprod()	Cumulative Product

Exercise: use the above statistical functions on table `studentDF`, and see the results.

Answer:

You can also use `describe()` to compute a summary of statistics pertaining to the DataFrame columns. See the following code.

```
import pandas as pd
import numpy as np

#Create a Dictionary of series
d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','
    Smith','Jack',
        'Lee','David','Gasper','Betina','Andres']),
        'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),
        'Rating':pd.Series
            ([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])
            }

#Create a DataFrame
df = pd.DataFrame(d)
df.describe()
```

Exercise: use `describe()` on table `studentDF` and answer the following questions

- What is the mean value of Math score?
- What is the score at the third quartile of Java?
- What is the standard deviation of Web programming score?

Missing Data

Missing data is always a problem in real life scenarios. For example, in an online survey, people do not answer all questions. Those are considered as missing data.

To explore functions to handle missing data, we will use reindexing to create a DataFrame with missing values as shown in the code below.

```
studentDF = studentDF.reindex([0,1,2,3,4,5,6,7])
print(studentDF)
```

Note that in the output, `NaN` means 'Not a Number'. How many `NaN` appear in table `studentDF`.

Check for Missing Values

To make detecting missing values easier (and across different array dtypes), Pandas provides the `isnull()` and `notnull()` functions, which are also methods on Series and DataFrame objects. Try the following code,

```
df = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e',
    'f', 'h'],columns=['one', 'two', 'three'])

df = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])

print(df['one'].isnull())
print(df['two'].notnull())
```

Exercise: use `isnull()` and `notnull()` with table `studentDF` and see the results.

Answer:

Filling Missing Data

Pandas provides various methods for cleaning the missing values. The fillna function can "fill in" NA values with non-null data. Try the following code with table `studentDF`.

```
s1 = studentDF
s2 = studentDF

s1 = s1.fillna(0)
s2 = s2.fillna(method='pad')

print(s1)
print(s2)
```

Exercise: What is the differences between `s1` and `s2`.

Answer:

Milestone 2: You have now completed the Milestone 2. Keep going!

Joining/Grouping

Pandas provides *join* operations very similar to relational databases like SQL. it is a single function, `merge`, as the entry point for all standard database join operations between DataFrame objects. The operation `join` combines columns from one or more tables based on their corresponding columns, e.g., ID.

Try the following code:

```
left = pd.DataFrame({
       'id':[1,2,3,4,5],
       'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
       'subject_id':['sub1','sub2','sub4','sub6','sub5']})

right = pd.DataFrame(
{'id':[1,2,3,4,5],
       'subject_name':['Math','English',
       'Social Science','Programming','Physics']})
print(left)
print(right)
```

What is the result?

Now, you create `left` and `right` Dataframes prepared for using `merge`. Try the following code to merge these two dataframes:

```
merged_table = pd.merge(left,right,on='id')
```

What is the result from using `merge()` on joining `left` and `right`

Exercise: create table `section` and create table `student` as the following. You have to use `merge()` to join the section table with the student table below and name your table as `score`. Note that use `id` as a joining key

id	section
0	one
1	one
2	one
3	two
4	two
5	two

id	name	math	english
0	Mat	34	23
1	Mike	12	13
2	John	34	51
3	Peecha	12	62
4	Prabu	65	12
5	Bunchu	23	42

Answer:

Try the following code for grouping and see the result

```
score_sec = score.groupby('section')
score_sec.sum()
```

What is the result from using `sum()` on the `groupby` table?

Answer:

Milestone 3: You have now completed the Milestone 3. Raise YOUR HAND NOW!