Fourth Industrial Summer School

Day 1

Data Manipulation

Session Objectives

- ✓ Indexing and Selecting data
- ✓ Sampling data
- ✓ Sorting and Ranking
- ✓ Hierarchical Indexing
- ✓ Merging/Joining
- ✓ Iterating over Dataframes
- ✓ Reshaping Dataframes

Introduction

- While performing data analysis, quite often it is required to
  - filter the data to ignore unnecessary rows or columns
  - sort the data according to a particular variable
  - merge different datasets

Indexing

- Indexing could mean selecting all the rows and some of the columns, some of the rows and all of the columns, or some of each of the rows and columns.
  - Indexing can also be known as **Subset Selection**.

- Indexing in pandas means simply selecting particular rows and columns of data from a DataFrame.

Indexing

- There are a lot of ways to pull the elements, rows, and columns from a DataFrame.

- There are some indexing method in Pandas which help in getting an element from a DataFrame.

- Pandas support four types of Multi-axes indexing they are:
  - Dataframe.[] ; This function also known as indexing operator
  - Dataframe.loc[] : This function is used for labels.
  - Dataframe.iloc[] : This function is used for positions or integer based

- Collectively, they are called the **indexers**.

Indexing operator to refer to df [ ]

- In order to select a single column, we simply put the name of the column in-between the brackets

```
# importing pandas package
import pandas as pd

# making data frame from csv file
data = pd.read_csv("employees.csv",
                   index_col ="Name")

# retrieving columns by indexing operator
subset = data["Team"]
subset
```

Output: one series was returned since there was only one parameter.

Indexing operator to refer to df [ ]

- In order to select multiple columns, we have to pass a list of columns in an indexing operator.

```
# making data frame from csv file
data = pd.read_csv("employees.csv",
                   index_col ="Name")

# retrieving columns by indexing operator
subset = data[["Team","Salary","Position"]]
subset
```

Name	Team	Salary	Position

Indexing a DataFrame using .loc[ ]

- This function selects data by the **label** of the rows and columns.
- .loc[ ] selects data in a different way than just the indexing operator.
  - It can select subsets of rows or columns.
  - It can also simultaneously select subsets of rows and columns.

Indexing a DataFrame using .loc[ ]

- To select a row using .loc[], we put a single row label in a .loc function.

```
# making data frame from csv file
data = pd.read_csv("employees.csv",
                   index_col ="Name")

# retrieving row by loc method
first = data.loc["Maria"]
second = data.loc["John"]

print(first, "\n\n\n", second)
```

Output: in the used data the index is repeated, so it is expected to see more then one row for each indexing

```
# retrieving multiple rows by loc method
first = data.loc[["Maria", "John"]]

first
```

Indexing a DataFrame using .loc[ ]

- .loc[] can be used also to select rows (using a particular index) and a sub set of columns

```
# making data frame from csv file
data = pd.read_csv("employees.csv",
                   index_col ="Name")

# retrieving by loc method
first = data.loc[["Maria", "John"],
                 ["Team","Position"]]
```

- To select all of the rows and some columns, we use single colon **[:]**

```
# retrieving all rows and some columns by loc method
first = data.loc[:, ["Team", "Position"]]
```

Indexing a DataFrame using .iloc[ ]

- This function allows us to retrieve rows and columns by position.
- The .iloc[] indexer is very similar to .loc[] but only uses integer locations to make its selections.
  - It requires identifying the positions of the target rows and columns

Indexing a DataFrame using .iloc[ ]

- To select a single row using .iloc[], → pass a single integer to .iloc[] function.

```
# retrieving rows by iloc method
sample = data.iloc[5]
sample
```

```
Gender                    Male
Start Date              4/18/87
Last Login Time         1:35 AM
Salary                   115163
Bonus %                  10.125
Senior Management         False
Team                      Legal
Position                     PF
Name: Dennis, dtype: object
```

You can also pass a list of integer to .iloc[] function to select multiple rows
e.g. data.iloc [[3, 5, 7]]

```
data.iloc [:, [1, 2]]
```

To select all rows and some columns

Indexing a DataFrame using .iloc[ ]

- To select subset of rows and columns:

```
sample2 = data.iloc [[3, 4], [5, 7]]
```

```
sample3 = data.iloc [1:20, 2:5]
```

Selecting rows in pandas DataFrame

- Selecting rows based on particular column value using '>', '=', '=', '<=', '!=' operator.

```
# selecting rows based on condition
rslt_df = SubData[SubData['Salary'] < 100000]
print('\nResult dataframe :\n', rslt_df)
```

```
Result dataframe :
                          Team Position  Salary
S_Rank
410.0                Marketing       PG   99747
411.0                    Legal       SF   99326
412.0             Distribution       PF   99283
```

```
rslt_df = SubData.loc[SubData['Salary'] < 100000]
```

Selecting rows in pandas DataFrame

- Selecting those rows whose column value is present in a list using isin() method of the dataframe.

```
options = ['Marketing', 'Finance']

# selecting rows based on condition
rslt_df = data[data['Team'].isin(options)]

print('\nResult dataframe :\n', rslt_df)
```

```
Result dataframe :
            Gender Start Date  ...       Team
Name                           ...
Douglas       Male     8/6/93  ...  Marketing
Maria       Female    4/23/93  ...    Finance
Jerry         Male     3/4/05  ...    Finance
```

```
rslt_df = data.loc[data['Team'].isin(options)]
```

  - Selecting all the rows from the given dataframe in which 'Team' is present in the options list using basic method.

Selecting rows in pandas DataFrame

- Selecting rows based on multiple column conditions using '&' operator.

```
options = ['Marketing', 'Finance']

# selecting rows based on two conditions
rslt_df = data[(data['Position'] == 'PG') &
               data['Team'].isin(options)]

print('\nResult dataframe :\n', rslt_df)
```

```
Result dataframe :
            Gender Start Date  ...       Team  Position
Name                           ...
Douglas       Male     8/6/93  ...  Marketing        PG
Alan           NaN     3/3/14  ...    Finance        PG
Rachel      Female    8/16/99  ...    Finance        PG
Doris       Female    8/20/04  ...    Finance        PG
```

Sampling (random selection)

- Using sample() method

```
# Select one row randomaly using sample()
# without give any parameters
data.sample()
```

  - Select **n** numbers of rows randomly using sample(n) or sample(n=n).
  - Each time you run this, you get n different rows.

Sampling (random selection)

- frac parameter can be used to do fraction of axis items and get rows.
  - For example, if frac= .5 then sample method return 50% of rows.

```
print(data.shape)

# Select 70% of data for training
train=data.sample(frac=.7)
# Select 30% of data for testing
test=data.sample(frac=.3)

print(train.shape)
print(test.shape)
```

```
(1000, 8)
(700, 8)
(300, 8)
```

Sampling (random selection)

- replace parameter
  - It gives a permission to select one rows many time.
  - Default value of replace parameter of sample() method is False so you never select more than total number of rows.

```
# Select more than rows with using replace
# default it is False
data.sample(n = 100, replace = True)
```

Sorting in Pandas

- There are two kinds of sorting available in Pandas. They are −
  - By label
  - By Actual Value

Sorting by Label

- sort_index() method,
  - by passing the axis arguments and the order of sorting,
  - By default, sorting is done on row labels in ascending order.

```
sorted_data= data.sort_index()
sorted_data
```

```
        Gender Start Date  ...
Name                       ...
Aaron     Male    2/17/12  ...
Aaron     Male    1/29/94  ...
Aaron     Male    7/22/90  ...
Aaron      NaN    1/22/86  ...
```

```
sorted_data2 = data.sort_index(ascending=False)
```

The order of the sorting can be controlled by passing a Boolean value to ascending parameter

Sort the coulmns

- By passing the axis argument with a value 0 or 1, the sorting can be done on the column labels.
  - By default, axis=0, sort by row.

```
sorted_data3= data.sort_index(axis=1)

print (sorted_data3)
```

```
          Bonus %  Gender  ... Start Date
Name                       ...
Douglas     6.945    Male  ...      8/6/93
Thomas      4.170    Male  ...     3/31/96
Maria      11.858  Female  ...     4/23/93
Jerry       9.340    Male  ...      3/4/05
```

```
data.sort_index(axis=1,ascending=False)
```

Sorting by Value

- sort_values() is the method for sorting by values.
  - It accepts a 'by' argument which will use the column name of the DataFrame with which the values are to be sorted.

```
sorted_data5 = data.sort_values(by='Gender')
print(sorted_data5)
```

```
             Gender Start Date  ...
Name                            ...
Christine    Female     2/1/03  ...
Andrea       Female    10/1/10  ...
Rose         Female    5/28/15  ...
Andrea       Female    7/22/99  ...
```

'by' argument takes a list of column values
e.g. sort_values(by=['col1','col2'])

Hands on session

Problem Solving

Ranking Rows of Pandas DataFrame

- To rank the rows of Pandas DataFrame we can use the DataFrame.rank() method which returns a rank of every respective index of a series passed.
  - The rank is returned on the basis of position after sorting.

Ranking Rows of Pandas DataFrame

```
# retrieving all rows and some columns by loc method
SubData = data.loc[:, ["Team", "Position","Salary"]]

# Create a column S_Rank which contains Salary order
SubData["S_Rank"]= SubData['Salary'].rank(ascending=0)

# Set the index to newly created column, S_Rank
SubData = SubData.set_index('S_Rank')

print(SubData)
```

```
                          Team Position  Salary
S_Rank
435.0                Marketing       PG   97308
756.0                      NaN       SF   61933
153.0                  Finance       SG  130590
90.0                   Finance       SG  138705
400.0          Client Services       PF  101004
```

```
sorted_df = SubData.sort_values(by='S_Rank')
```

Ranking Rows of Pandas DataFrame

- sort_index() can be used to sort the created index

```
SubData = SubData.sort_index()
print(SubData)
```

```
                          Team Position  Salary
S_Rank
1.0                    Finance       PG  149908
2.0          Human Resources       SG  149903
3.0                    Product       PG  149684
4.0                      Sales       SG  149654
5.0                    Finance       PF  149563
6.0                  Marketing       SF  149456
```

Hierarchical Data In pandas

```
# Set the hierarchical index
Sample = data.set_index(['Team','Position'])

sorted_sample=Sample.sort_index()
print (sorted_sample)
```

```
                                Gender Start Date
Team                  Position
Business Development  C         Female    9/13/86
                      C            NaN    6/28/15
                      C         Female    11/6/01
                      C         Female   12/17/99
                      C         Female    6/19/99
                      C            NaN     3/1/90
                      C         Female    2/22/94
```

Hierarchical Data In pandas

```
print(Sample.sum(level='Position'))
print(Sample.sum(level='Team'))
```

```
            Salary   Bonus %
Position
PG        18220268  2175.749
SF        15233081  1626.513
SG        20207685  2321.107
PF        22523064  2460.131
C         14478083  1624.055
                       Salary   Bonus %
Team
Marketing                8862688  1014.638
Finance                  9406387  1039.061
Client Services          9351789  1112.481
Legal                    7858718   908.409
```

Merging/Joining

- Pandas has full-featured, high performance in-memory join operations very similar to relational databases like SQL.

- We can join, merge, and concat dataframe using different methods.
  - df.merge(),
  - df.join(), and
  - df.concat()

DataFrames Concatenation

- concat() function performs the concatenation operations along an axis while performing optional set logic (union or intersection) of the indexes (if any) on the other axes.

df1

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

df2

	A	B	C	D
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

df3

	A	B	C	D
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

Result

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

DataFrames Concatenation

```
# Creating first dataframe
df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
                    'B': ['B0', 'B1', 'B2', 'B3'],
                    'C': ['C0', 'C1', 'C2', 'C3'],
                    'D': ['D0', 'D1', 'D2', 'D3']},
                   index = [0, 1, 2, 3])

# Creating second dataframe
df2 = pd.DataFrame({'A': ['A4', 'A5', 'A6', 'A7'],
                    'B': ['B4', 'B5', 'B6', 'B7'],
                    'C': ['C4', 'C5', 'C6', 'C7'],
                    'D': ['D4', 'D5', 'D6', 'D7']},
                   index = [4, 5, 6, 7])

# Creating third dataframe
df3 = pd.DataFrame({'A': ['A8', 'A9', 'A10', 'A11'],
                    'B': ['B8', 'B9', 'B10', 'B11'],
                    'C': ['C8', 'C9', 'C10', 'C11'],
                    'D': ['D8', 'D9', 'D10', 'D11']},
                   index = [8, 9, 10, 11])

# Concatenating the dataframes
pd.concat([df1, df2, df3])
```

sort=False

axis=1

DataFrames Merge

- Pandas provides a single function, merge(), as the entry point for all standard database join operations between DataFrame objects.

```
# Python program to merge
# dataframes using Panda

# Dataframe created
left = pd.DataFrame({'Key': ['K0', 'K1', 'K2', 'K3'],
                     'A': ['A0', 'A1', 'A2', 'A3'],
                     'B': ['B0', 'B1', 'B2', 'B3']})

right = pd.DataFrame({'Key': ['K0', 'K1', 'K2', 'K3'],
                      'C': ['C0', 'C1', 'C2', 'C3'],
                      'D': ['D0', 'D1', 'D2', 'D3']})

# Merging the dataframes
pd.merge(left, right, on ='Key')
```

	Key	A	B	C	D
0	K0	A0	B0	C0	D0
1	K1	A1	B1	C1	D1
2	K2	A2	B2	C2	D2
3	K3	A3	B3	C3	D3

DataFrames Join

```
# Python program to join
# dataframes using Panda

left = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
                     'B': ['B0', 'B1', 'B2', 'B3']},
                     index = ['K0', 'K1', 'K2', 'K3'])

right = pd.DataFrame({'C': ['C0', 'C1', 'C2', 'C3'],
                      'D': ['D0', 'D1', 'D2', 'D3']},
                      index = ['K0', 'K1', 'K2', 'K3'])

# Joining the dataframes
left.join(right)
```

	A	B	C	D
K0	A0	B0	C0	D0
K1	A1	B1	C1	D1
K2	A2	B2	C2	D2
K3	A3	B3	C3	D3

Iterating in Pandas DataFrame

- In Pandas Dataframe we can iterate an element in two ways:
  - Iterating over rows
  - Iterating over columns

- Three functions can be used for iteration over rows:
  - iteritems(),
  - iterrows(), and
  - itertuples().

Iterating over rows

- iterrows() function returns each index value along with a series containing the data in each row.

```
for i, j in data.iterrows():
    print(i, j)
    print()
```

```
Douglas Gender                      Male
Start Date                       8/6/93
Last Login Time                12:42 PM
Salary                            97308
Bonus %                           6.945
Senior Management                  True
Team                          Marketing
Position                             PG
Name: Douglas, dtype: object
```

Iterating over rows

- iteritems() function iterates over each column as key, value pair with label as key and column value as a Series object.

```
# using iteritems() function to retrieve rows
for key, value in data.iteritems():
    print(key, value)
    print()
```

```
Gender Name
Douglas       Male
Thomas        Male
Maria       Female
Jerry         Male
```

Iterating over rows

- itertuples() function returns a tuple for each row in the DataFrame.
  - The first element of the tuple will be the row's corresponding index value, while the remaining values are the row values.

```
# using a itertuples()
for i in data.itertuples():
    print(i)
```

```
Pandas(Index='Douglas', Gender='Male',
Pandas(Index='Thomas', Gender='Male', _
```

Iterating over Columns

- To iterate through columns we first create a list of dataframe columns and then iterate through list.

```
# creating a list of dataframe columns
columns = list(data)

for i in columns:
    # printing the third element of the column
    print (data[i][4])
```

```
Male
1/24/98
4:47 PM
101004
1.389
True
Client Services
PF
```

Reshape a pandas DataFrame

- Pandas use various methods to reshape the dataframe and series.

- Stack method works with the MultiIndex objects in DataFrame,
  - it returning a DataFrame with an index with a new inner-most level of row labels.
  - It changes the wide table to a long table.

```
data_stacked = data.stack()

print(data_stacked.head(20))
```

```
Name
Douglas  Gender                  Male
         Start Date             8/6/93
         Last Login Time     12:42 PM
         Salary                 97308
         Bonus %                6.945
         Senior Management       True
         Team                Marketing
         Position                   PG
Thomas   Gender                  Male
         Start Date            3/31/96
         Last Login Time      6:53 AM
         Salary                 61933
         Bonus %                 4.17
         Senior Management       True
```

Reshape a pandas DataFrame

- unstack is similar to stack method,
  - It also works with multi-index objects in dataframe, producing a reshaped DataFrame with a new inner-most level of column labels.

```
# unstack() method
d = data_stacked.unstack()
```

Hands on session

Problem Solving