DATA ANALYSIS IN PYTHON (pandas)

```
import numpy
import pandas
import xlrd
import matplotlib
```

pandas

'pandas' is a package useful for data analysis and manipulation.

There are 2 key data structures in pandas – Series and DataFrames.

Series and Dataframes

Series can be understood as a 1 dimensional labelled / indexed array. You can access individual elements of this series through these labels. We can imagine as: Pandas Series is nothing but a column in an excel sheet.

A dataframe is similar to Excel spreadsheet – you have column names referring to columns and you have rows, which can be accessed with use of row numbers. The essential difference being that column names and row numbers are known as column and row index, in case of dataframes.

Series and dataframes form the core data model for Pandas in Python. The data sets are first read into these dataframes and then various operations (e.g. group by, aggregation etc.) can be applied very easily to its columns.

Dataframe is created from csv files, Excel files, text files, Python dictionary, list of tuples, list of dictionaries.

CREATION OF DATAFRAMES

Creating from .csv file

```
>>> import pandas as pd
>>> df = pd.read_csv('f:\\python\PANDAS\empdata.csv')
>>> df
```

Creating from .xlsx file

```
>>> df = pd.read_excel("f:\\python\PANDAS\empdata.xlsx", "Sheet1")
>>> df
```

Creating from .txt file

```
>>>df = pd.read_table('f://python/pandas/textdata.txt',
delim_whitespace=True, names=('A', 'B', 'C', 'Person'))
>>>df
```

Creating from a Python dictionary

```
>>> empdata = {"empid": [1001, 1002, 1003, 1004, 1005, 1006],
"ename": ["Ganesh Rao", "Anil Kumar", "Gaurav Gupta", "Hema Chandra",
"Laxmi Prasanna", "Anant Nag"],
"sal": [10000, 23000.50, 18000.33, 16500.50, 12000.75, 9999.99],
"doj": ["10-10-2000", "3-20-2002", "3-3-2002", "9-10-2000", "10-8-2000",
"9-9-1999"]}

>>> df = pd.DataFrame(empdata)
>>> df
```

Creating from a list of tuples

```
>>> empdata = [(1001, 'Ganesh Rao', 10000.00, '10-10-2000'),
(1002, 'Anil Kumar', 23000.50, '3-20-2002'),
(1003, 'Gaurav Gupta', 18000.33, '03-03-2002'),
(1004, 'Hema Chandra', 16500.50, '10-09-2000'),
(1005, 'Laxmi Prasanna', 12000.75, '08-10-2000'),
(1006, 'Anant Nag', 9999.99, '09-09-1999')]

>>> df = pd.DataFrame(empdata, columns=["eno", "ename", "salary", "doj"])
>>> df
```

VIEWING DATAFRAME USING LOC() AND ILOC()

loc() is useful to view rows and cols based on labels (or names). iloc() is useful to view data based on integer index.

To view all rows with only 'ename' and 'doj' columns.

```
>>> df1 = df.loc[:, ['ename', 'doj']]
>>> df1
           ename         doj
0      Ganesh Rao  10-10-2000
1      Anil Kumar    3-3-2002
2    Gaurav Gupta    3-3-2002
3    Hema Chandra    3-3-2002
4  Laxmi Prasanna   10-8-2000
5       Anant Nag    9-9-1999
```

with iloc

```
>>> df1 = df.iloc[:, [1, 3]]
>>> df1
```

To view only 0 to 3rd rows of 'ename' and 'doj' columns

```
>>> df1 = df.loc[0:3, ['ename', 'doj']]
>>> df1
          ename         doj
0    Ganesh Rao  10-10-2000
1    Anil Kumar    3-3-2002
2  Gaurav Gupta    3-3-2002
3  Hema Chandra    3-3-2002
```

with iloc

```
>>> df1 = df.iloc[0:3, [1,3]]
>>> df1
```

To view 0 to 3rd rows with all columns

```
>>> df1 = df.loc[0:3, :]
(or) df1 = df.loc[0:3, ]
```

with iloc

```
>>> df1 = df.iloc[0:3, :]
(or) df1 = df.iloc[0:3, ]
```

To view 3rd row to 0th row with all columns

```
>>> df1 = df.loc[3:0:-1, :]
>>> df1
```

with iloc

```
>>> df1 = df.iloc[3:0:-1, :]
```

```
>>> df1
```

To retrieve last row only

```
>>> df1 = df.iloc[-1]  (not possible with loc)
```

To retrieve last column only

```
>>> df1 = df.loc[:, 'doj']  (or)
>>> df1 = df.iloc[:, -1]
```

BASIC OPERATIONS ON DATAFRAMES

```
>>> df = pd.read_csv("f:\\python\PANDAS\empdata.csv")
>>> df
   empid           ename       sal        doj
0   1001      Ganesh Rao  10000.00   10-10-00
1   1002      Anil Kumar  23000.50  3-20-2002
2   1003    Gaurav Gupta  18000.33   03-03-02
3   1004    Hema Chandra  16500.50   10-09-00
4   1005  Laxmi Prasanna  12000.75   08-10-00
5   1006       Anant Nag   9999.99   09-09-99
```

1. To know the no. of rows and cols – shape

```
>>> df.shape
(6, 4)

>>> r, c = df.shape
>>> r
6
```

2. To display the first or last 5 rows – head(), tail()

```
>>> df.head()
>>> df.tail()

To display the first 2 or last 2 rows

>>> df.head(2)
>>> df.tail(2)
```

3. Displaying range of rows – df[2:5]

To display 2nd row to 4th row:

```
>>> df[2:5]

To display all rows:
>>> df[:]
>>> df
```

4. To display column names – df.columns

```
>>> df.columns
Index(['empid', 'ename', 'sal', 'doj'], dtype='object')
```

5. To display column data – df.columname

```
>>> df.empid  (or)
>>> df['empid']

>>> df.sal  (or)
>>> df['sal']
```

6. To display multiple column data – df[[list of colnames]]

```
>>> df[['empid', 'ename']]
```

```
   empid           ename
0   1001      Ganesh Rao
1   1002      Anil Kumar
2   1003    Gaurav Gupta
3   1004    Hema Chandra
4   1005  Laxmi Prasanna
5   1006       Anant Nag
```

7. Finding maximum and minimum – max() and min()

```
>>> df['sal'].max()
23000.5
>>> df['sal'].min()
9999.9899999999998
```

8. To display statistical information on numerical cols – describe()

```
>>> df.describe()
             empid           sal
count     6.000000      6.000000
mean   1003.500000  14917.011667
std       1.870829   5181.037711
min    1001.000000   9999.990000
25%    1002.250000  10500.187500
50%    1003.500000  14250.625000
75%    1004.750000  17625.372500
max    1006.000000  23000.500000
```

9. Show all rows with a condition

To display all rows where sal>10000

```
>>> df[df.sal>10000]
   empid           ename       sal        doj
1   1002      Anil Kumar  23000.50  3-20-2002
2   1003    Gaurav Gupta  18000.33   03-03-02
3   1004    Hema Chandra  16500.50   10-09-00
4   1005  Laxmi Prasanna  12000.75   08-10-00
```

To retrieve the row where salary is maximum

```
>>> df[df.sal == df.sal.max()]
   empid       ename      sal        doj
1   1002  Anil Kumar  23000.5  3-20-2002
```

10. To show only cols of rows based on condition

```
>>> df[['empid', 'ename']][df.sal>10000]
   empid           ename
1   1002      Anil Kumar
2   1003    Gaurav Gupta
3   1004    Hema Chandra
4   1005  Laxmi Prasanna
```

11. To know the index range - index

```
>>> df.index
RangeIndex(start=0, stop=6, step=1)
```

12. To change the index to a column – set_index()

```
>>> df1 = df.set_index('empid')

(or) to modify the same Data Frame:
>>> df.set_index('empid', inplace=True)

>>> df
               ename       sal        doj
empid
```

```
1001       Ganesh Rao  10000.00   10-10-00
1002       Anil Kumar  23000.50  3-20-2002
1003     Gaurav Gupta  18000.33   03-03-02
1004     Hema Chandra  16500.50   10-09-00
1005   Laxmi Prasanna  12000.75   08-10-00
1006        Anant Nag   9999.99   09-09-99
```

NOTE: Now it is possible to search on empid value using loc[].

```
>>> df.loc[1004]
ename    Hema Chandra
sal           16500.5
doj          10-09-00
Name: 1004, dtype: object
```

13. To reset the index back – reset_index()

```
>>> df.reset_index(inplace=True)
>>> df
   empid           ename       sal        doj
0   1001      Ganesh Rao  10000.00   10-10-00
1   1002      Anil Kumar  23000.50  3-20-2002
2   1003    Gaurav Gupta  18000.33   03-03-02
3   1004    Hema Chandra  16500.50   10-09-00
4   1005  Laxmi Prasanna  12000.75   08-10-00
5   1006       Anant Nag   9999.99   09-09-99
```

HANDLING MISSING DATA

Read .csv file data into Data Frame

```
>>> df = pd.read_csv("f:\\python\PANDAS\empdata1.csv")
>>> df
   empid           ename       sal       doj
0   1001      Ganesh Rao  10000.00  10-10-00
1   1002      Anil Kumar  23000.50  03-03-02
2   1003             NaN  18000.33  03-03-02
3   1004    Hema Chandra       NaN       NaN
4   1005  Laxmi Prasanna  12000.75  10-08-00
5   1006       Anant Nag   9999.99  09-09-99
```

To set the empid as index – set_index()

```
>>> df.set_index('empid', inplace=True)
>>> df
                ename       sal       doj
empid
1001       Ganesh Rao  10000.00  10-10-00
1002       Anil Kumar  23000.50  03-03-02
1003              NaN  18000.33  03-03-02
1004     Hema Chandra       NaN       NaN
1005   Laxmi Prasanna  12000.75  10-08-00
1006        Anant Nag   9999.99  09-09-99
```

To fill the NaN values by 0 – fillna(0)

```
>>> df1 = df.fillna(0)
>>> df1
                ename       sal       doj
empid
1001       Ganesh Rao  10000.00  10-10-00
1002       Anil Kumar  23000.50  03-03-02
1003                0  18000.33  03-03-02
1004     Hema Chandra      0.00         0
1005   Laxmi Prasanna  12000.75  10-08-00
1006        Anant Nag   9999.99  09-09-99
```

To fill columns with different data – fillna(dictionary)

```
>>> df1 = df.fillna({'ename': 'Name missing', 'sal': 0.0, 'doj':'00-00-
00'})
>>> df1
                ename       sal       doj
empid
1001       Ganesh Rao  10000.00  10-10-00
1002       Anil Kumar  23000.50  03-03-02
1003     Name missing  18000.33  03-03-02
1004     Hema Chandra      0.00  00-00-00
1005   Laxmi Prasanna  12000.75  10-08-00
1006        Anant Nag   9999.99  09-09-99
```

To delete all rows with NaN values – dropna()

```
>>> df1 = df.dropna()
>>> df1
                ename       sal       doj
empid
1001       Ganesh Rao  10000.00  10-10-00
1002       Anil Kumar  23000.50  03-03-02
1005   Laxmi Prasanna  12000.75  10-08-00
1006        Anant Nag   9999.99  09-09-99
```

SORTING THE DATA

Read .csv file data into Data Frame and indicate to consider 'doj' as date type field

```
>>> df = pd.read_csv("f:\\python\PANDAS\empdata2.csv",
parse_dates=['doj'])
>>> df
empid           ename       sal        doj
0   1001      Ganesh Rao  10000.00 2000-10-10
1   1002      Anil Kumar  23000.50 2002-03-03
2   1003    Gaurav Gupta  18000.33 2002-03-03
3   1004    Hema Chandra  16500.50 2002-03-03
4   1005  Laxmi Prasanna  12000.75 2000-08-10
5   1006       Anant Nag   9999.99 1999-09-09
```

To sort on a column – sort_values(colname)

```
>>> df1 = df.sort_values('doj')
>>> df1
>>> df1
   empid           ename       sal        doj
5   1006       Anant Nag   9999.99 1999-09-09
4   1005  Laxmi Prasanna  12000.75 2000-08-10
0   1001      Ganesh Rao  10000.00 2000-10-10
1   1002      Anil Kumar  23000.50 2002-03-03
2   1003    Gaurav Gupta  18000.33 2002-03-03
3   1004    Hema Chandra  16500.50 2002-03-03
```

NOTE: To sort in descending order:

```
>>> df1 = df.sort_values('doj', ascending=False)
```

To sort multiple columns differently – sort_values(by =[], ascending = [])

To sort on 'doj' in descending order and in that on 'sal' in ascending order:

```
>>> df1 = df.sort_values(by=['doj', 'sal'], ascending=[False, True])
>>> df1
   empid           ename       sal        doj
3   1004    Hema Chandra  16500.50 2002-03-03
2   1003    Gaurav Gupta  18000.33 2002-03-03
```

```
1   1002      Anil Kumar  23000.50 2002-03-03
0   1001      Ganesh Rao  10000.00 2000-10-10
4   1005  Laxmi Prasanna  12000.75 2000-08-10
5   1006       Anant Nag   9999.99 1999-09-09
```

CREATION OF SERIES

We can create series from a dataframe or an array or dictionary.

Create Series object from dataframe

```
>>> import pandas as pd
>>> df = pd.read_csv("f://python/pandas/nba.csv")
>>> ser = pd.Series(df['Name'])
>>> ser

>>> ser1 = pd.Series(df['Age'])
>>> ser1
```

Adding two series and make a dataframe

```
>>> df1 = pd.concat([ser, ser1], axis=1)  # axis=1 add columns
>>> df1

Now, if needed change the column names
>>> df1.columns = ['empname', 'empage']
>>> df1
```

Another way to add two series

```
>>> mydict = {'empname': ser, 'empage': ser1}  # first create dict
>>> df1 = pd.DataFrame(mydict)
>>> df1
```

Convert series into numpy arrays

```
>>> a = ser.values
>>> a
```

Creating Series from Numpy Array

```
>>> import numpy as np
>>> arr = np.array([10, 20, 30, 40])
>>> ser = pd.Series(arr)
>>> ser
0    10
1    20
2    30
3    40
dtype: int32
```

Creating Series from Dictionary

```
>>> my_dict = {
'name' : ["a", "b", "c", "d", "e"],
'age' : [10,20, 30, 40, 50],
'designation': ["CEO", "VP", "SVP", "AM", "DEV"]
}

>>> ser = pd.Series(my_dict)
>>> ser
name                    [a, b, c, d, e]
age               [10, 20, 30, 40, 50]
designation    [CEO, VP, SVP, AM, DEV]
dtype: object
```

ACCESSING ELEMENTS OF A SERIES

To retrieve all names

```
>>> names = ser['name']
>>> names
['a', 'b', 'c', 'd', 'e']
```

To retrieve 0^{th} name

```
>>> names = ser['name'][0]
>>> names
'a'
```

WRITING DATA INTO DATASET FILE

```
df.loc[8] = [1007, 'New name', 33000.55, '10-1-1999']
df

df.to_csv("E:/PYTHON/PANDAS/empmodified.csv")

df.to_excel("E:/PYTHON/PANDAS/modified1.xlsx")
```