Guru Nanak Khalsa College (Autonomous) – Mumbai 19
Department of Mathematics & Statistics

Unit II Chapter 1 Introduction to Pandas & Series

What is Panda:

Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures. The name Pandas is derived from the word Panel Data.

Panda is developed by Was McKinny and his team in 2008, when he needed need of high performance, flexible tool for analysis of data.

Panda is a powerful tool for data analysis. Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data — load, prepare, manipulate, model, and analyse.

Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc.

Key features of Panda:

1. Fast and efficient DataFrame object with default and customized indexing.
2. Tools for loading data into in-memory data objects from different file formats.
3. Data alignment and integrated handling of missing data.
4. Reshaping and pivoting of date sets.
5. Label-based slicing, indexing and subsetting of large data sets.
6. Columns from a data structure can be deleted or inserted.
7. Group by data for aggregation and transformations.
8. High performance merging and joining of data.
9. Time Series functionality.

Different Data Structures in Panda:

Pandas deals with the following three data structures:
☐ Series, ☐ DataFrame. ☐ Panel
These data structures are built on top of NumPy array, which means they are fast. The best way to think of these data structures is that the higher dimensional data structure is a container of its lower dimensional data structure. For example, DataFrame is a container of Series, Panel is a container of DataFrame.

The dimension and description of these three data structures are as follows:

Data Sturctures	Dimension	Description
Series	1 – dimension	1D labelled homogeneous array, size immutable.
DataFrame	2 – dimension	General 2D labelled, size-mutable tabular structure with potentially heterogeneously typed columns.
Panel	3 – dimension	General 3D labelled, size-mutable array.

All Pandas data structures are value mutable (can be changed) and except Series all are size mutable. Series is size immutable.

Series:

Series is a one-dimensional array like structure with homogeneous data and an associated array of data labels, called its index. By default, an index is positive integer starting with 0. We can also assign values of other data types as index. We can imagine a Pandas Series as a column in a spreadsheet. Following are some key points of panda series:

1. Homogenous data.
2. Size immutable.
3. Values of data mutable.

Different ways to create Panda Series:

A panda Series() is used to create panda series. A series can be created using Python dictionary, ndarray and using various functions.

```
In [1]: import numpy as np

In [3]: import pandas as pd

In [6]: s1=pd.Series([1,2,-3,-4]); print(s1)
0    1
1    2
2   -3
3   -4
dtype: int64
```

Here note the following points:

1. The default value index is 0 to where is the size of the series.
2. We can get the array representation and index object of the Series via its values and index attributes, respectively.
3. Using attributes index and values one can see index and values associated to given series.

```
In [9]: s1.values
Out[9]: array([ 1,  2, -3, -4], dtype=int64)

In [10]: s1.index
Out[10]: RangeIndex(start=0, stop=4, step=1)
```

4. Using index attribute in Series(), one can create a series with its own index.

```
In [13]: s2=pd.Series([1,-3,-4,56,23], index=['a','b','c','d','e']);
         print(s2)

a     1
b    -3
c    -4
d    56
e    23
dtype: int64
```

5. Creating empty series.

```
In [14]: emty=pd.Series()
         emty

C:\Users\HPWORL~1\AppData\Local\
ll be 'object' instead of 'float
  emty=pd.Series()

Out[14]: Series([], dtype: float64)
```

6. Creating series with ndarray.

```
In [16]: a1=np.array([10,20,30,40,50])
         s4=pd.Series(a1)
         print(s4)

0    10
1    20
2    30
3    40
4    50
dtype: int32
```

7. Creating series with python dictionary.

```
In [17]: d1={'Jan':1,'Feb':2,'Mar':3}
         s5=pd.Series(d1)
         print(s5)

Jan    1
Feb    2
Mar    3
dtype: int64
```

8. Here observe that dictionary key is used as index. Even if one change the order of index, then only values remain same, and for extra index NaN (not a number) is displayed.

```
In [20]: s6=pd.Series(d1, index=['Feb','Mar','Jan','Apr'])
         print(s6)
```

```
Feb    2.0
Mar    3.0
Jan    1.0
Apr    NaN
dtype: float64
```

9. A series with constant value can also be created. In such case the index must be provided. The value will be repeated to match the length of index.

```
In [21]: s7=pd.Series(3.14,index=[0,1,2,3])
         print(s7)
```

```
0    3.14
1    3.14
2    3.14
3    3.14
dtype: float64
```

Accessing the part of series:

1. Accessing data from series with position (Indexing):
   The index value is used to access the data from series. There are two types of indexes one is positional index and other is labelled index. Positional index takes an integer value that corresponds to its position in the series starting from 0, whereas labelled index takes any user-defined label as index.

```
In [21]: print(s2)
```

```
a     1
b    -3
c    -4
d    56
e    23
dtype: int64
```

```
In [22]: print(s2[0], s2[1], s2[-1]) # positional index
```

```
1 -3 23
```

```
In [25]: print(s2['a'],s2['e'],s2[['a','b','d']]) # multiple values using labelled index
```

```
1 23 a     1
b    -3
d    56
dtype: int64
```

```
In [27]: sCap=pd.Series(['Mumbai','Bangalore','Lucknow','bhopal','Shimla'
         ], index=['Maharashtra', 'Karnataka', 'Utter Pradesh','Madhya pradesh','Himanchal Pradesh'])
         print(sCap)

Maharashtra            Mumbai
Karnataka           Bangalore
Utter Pradesh         Lucknow
Madhya pradesh         bhopal
Himanchal Pradesh      Shimla
dtype: object

In [28]: sCap[['Maharashtra','Himanchal Pradesh']]

Out[28]: Maharashtra          Mumbai
         Himanchal Pradesh    Shimla
         dtype: object

In [29]: sCap[[1,3]]

Out[29]: Karnataka         Bangalore
         Madhya pradesh       bhopal
         dtype: object
```

The index values associated with the series can be altered by assigning new index values as shown in the following example:

```
In [30]: sCap.index=[10,20,30,40,50]
         sCap

Out[30]: 10        Mumbai
         20     Bangalore
         30       Lucknow
         40        bhopal
         50        Shimla
         dtype: object
```

2. Slicing:

Slicing is used to extract the part of a series. We can define which part of the series is to be sliced by specifying the start and end parameters [start: end] with the series name. When we use positional indices for slicing, the value at the end index position is excluded.
If labelled indexes are used for slicing, then value at the end index label is also included in the output.

```
In [31]: sCap[1:3] # in positional index, end value is not included

Out[31]: 20    Bangalore
         30      Lucknow
         dtype: object

In [32]: s2

Out[32]: a     1
         b    -3
         c    -4
         d    56
         e    23
         dtype: int64

In [33]: s2['a':'c'] # in leballed index end value is included

Out[33]: a    1
         b   -3
         c   -4
         dtype: int64
```

We can also use slicing to modify the values of series elements as shown in the following example:

```
In [34]: s2['a':'b']=100
         s2

Out[34]: a    100
         b    100
         c     -4
         d     56
         e     23
         dtype: int64
```

Methods in Series:

Consider the following series:

```
In [24]: snum=pd.Series(np.arange(10,30,2))
         print(snum)

         0    10
         1    12
         2    14
         3    16
         4    18
         5    20
         6    22
         7    24
         8    26
         9    28
         dtype: int32
```

head(n): Returns the first members of the series. If the value for is not passed, then by default takes 5 and the first five members are displayed.

```
In [26]: snum.head(2)

Out[26]: 0    10
         1    12
         dtype: int32
```

```
In [27]: snum.head()

Out[27]: 0    10
         1    12
         2    14
         3    16
         4    18
         dtype: int32
```

tail(n): Returns the last n members of the series. If the value for n is not passed, then by default n takes 5 and the last five members are displayed.

```
In [28]: snum.tail(3)

Out[28]: 7    24
         8    26
         9    28
         dtype: int32
```

```
In [29]: snum.tail()

Out[29]: 5    20
         6    22
         7    24
         8    26
         9    28
         dtype: int32
```

Mathematical Operations on Series

Like NumPy array, series in pandas can also be add, substract, multiply and divide elementwise. While performing mathematical operations on series, index matching is implemented and all missing values are filled in with NaN by default.

Consider following series:

```
In [30]: seriesA=pd.Series([1,2,3,4,5],index=['a','b','c','d','e'])
         seriesB=pd.Series([-1,20,32,-45,10],index=['b','a','e','f','g'])
         seriesC=pd.Series([12,34,54],index=['w','x','y'])
```

```
In [31]: seriesA+seriesB

Out[31]: a    21.0
         b     1.0
         c     NaN
         d     NaN
         e    37.0
         f     NaN
         g     NaN
         dtype: float64
```

```
In [32]: seriesA*seriesB

Out[32]: a     20.0
         b     -2.0
         c      NaN
         d      NaN
         e    160.0
         f      NaN
         g      NaN
         dtype: float64
```

```
In [33]: seriesA/seriesB

Out[33]: a    0.05000
         b   -2.00000
         c        NaN
         d        NaN
         e    0.15625
         f        NaN
         g        NaN
         dtype: float64
```

```
In [34]: seriesA-seriesC

Out[34]: a   NaN
         b   NaN
         c   NaN
         d   NaN
         e   NaN
         w   NaN
         x   NaN
         y   NaN
         dtype: float64
```

The same results can be obtained with add(), sub(), mul() and div() methods also with the difference that missing values corresponding to given index is first replaced by any values using fill_value attributes. Then the operation is take place.

Examples:

```
In [38]: seriesA.add(seriesC,fill_value=0)

Out[38]: a     1.0
         b     2.0
         c     3.0
         d     4.0
         e     5.0
         w    12.0
         x    34.0
         y    54.0
         dtype: float64
```

```
In [39]: seriesA.sub(seriesC,fill_value=0)

Out[39]: a     1.0
         b     2.0
         c     3.0
         d     4.0
         e     5.0
         w   -12.0
         x   -34.0
         y   -54.0
         dtype: float64
```

```
In [40]: seriesA.mul(seriesC,fill_value=0)

Out[40]: a    0.0
         b    0.0
         c    0.0
         d    0.0
         e    0.0
         w    0.0
         x    0.0
         y    0.0
         dtype: float64
```

```
In [41]: seriesA.div(seriesC,fill_value=0)

Out[41]: a    inf
         b    inf
         c    inf
         d    inf
         e    inf
         w    0.0
         x    0.0
         y    0.0
         dtype: float64
```

Guru Nanak Khalsa College (Autonomous) – Mumbai 19
Department of Mathematics & Statistics

Unit II Chapter 2 Panda DataFrame

Pandas DataFrame is a widely used data structure which works with a two-dimensional array with labeled axes (rows and columns). DataFrame is defined as a standard way to store data that has two different indexes, i.e., **row index** and **column index**. It consists of the following properties:

- The columns can be heterogeneous types like int, bool, and so on.
- It can be seen as a dictionary of Series structure where both the rows and columns are indexed. It is denoted as "columns" in case of columns and "index" in case of rows.

A pandas DataFrame can be created using the following constructor –

pd.DataFrame(data, index, columns, dtype, copy)

The parameters of the constructor are as follows –

1. data: data takes various forms like ndarray, series, map, lists, dict, constants and also another DataFrame.
2. index: for the row labels, the Index to be used for the resulting frame is Optional Default np.arange(n) if no index is passed.
3. columns: for column labels, the optional default syntax is - np.arange(n). This is only true if no index is passed.
4. dtype: Data type of each column.
5. copy: This command (or whatever it is) is used for copying of data, if the default is False.

Creating of DataFrame:

1. Creating empty DataFrame:

```
In [3]: dfMT=pd.DataFrame()
        print(dfMT)

        Empty DataFrame
        Columns: []
        Index: []
```

2. Creating DataFrame using list:

i)

```
In [4]: ldata=['aa','bb','cc','dd']
        dfeg1=pd.DataFrame(ldata)
        print(dfeg1)

            0
        0  aa
        1  bb
        2  cc
        3  dd
```

ii) Creating DataFrame with rows and columns using list.

```
In [5]: ldata2=[['Rakesh',43],['Rajesh',40],['Ramesh',45],['Suresh',56]]
        dfeg2=pd.DataFrame(ldata2)
        print(dfeg2)
```

```
        0   1
0  Rakesh  43
1  Rajesh  40
2  Ramesh  45
3  Suresh  56
```

iii) Changing the column heading:

```
In [6]: dfeg3=pd.DataFrame(ldata2,columns=['Name','Age'])
        print(dfeg3)
```

```
     Name  Age
0  Rakesh   43
1  Rajesh   40
2  Ramesh   45
3  Suresh   56
```

iv) DataFrame with user defined index:

```
In [7]: dfeg4=pd.DataFrame(ldata2, index=[1,2,3,4])
        dfeg4
```

Out[7]:

	0	1
1	Rakesh	43
2	Rajesh	40
3	Ramesh	45
4	Suresh	56

v) Creating DataFrame using user defined index and column heading:

```
In [8]: dfeg5=pd.DataFrame(ldata2,index=['rank1','rank2','rank3','rank4'], columns=['Name','Age'])
        dfeg5
```

Out[8]:

	Name	Age
rank1	Rakesh	43
rank2	Rajesh	40
rank3	Ramesh	45
rank4	Suresh	56

vi) Creating DataFrame using dictionary:

```
In [11]: dict1={'Name':['Sita','Gita','Sunita','Madumita'], 'Age':[23,34,19,40]}
         dfeg6=pd.DataFrame(dict1, index=['G1', 'G2','G3','G4'])
         dfeg6
```

Out[11]:

	Name	Age
G1	Sita	23
G2	Gita	34
G3	Sunita	19
G4	Madumita	40

vii) Creating a DataFrame from List of Dictionary:

```
In [13]: ldata3=[{'a':1,'b':2,'c':3},{'a':10,'b':20,'c':30,'d':40}]
         dfeg7=pd.DataFrame(ldata3)
         dfeg7
```

Out[13]:

	a	b	c	d
0	1	2	3	NaN
1	10	20	30	40.0

viii) Creating DataFrame using ndarray:

```
In [14]: arr1=np.array([12,23,34,45,56])
         arr2=np.array([-10,30,-45,30.34])
         arr3=np.array(['aa','bb','cc','dd','ee'])
         dfeg8=pd.DataFrame([arr1,arr2,arr3])
         dfeg8
```

Out[14]:

	0	1	2	3	4
0	12	23	34	45	56
1	-10.0	30.0	-45.0	30.34	None
2	aa	bb	cc	dd	ee

ix) Changing row index and column index:

```
In [15]: dfeg8.index=['R1','R2','R3']
         dfeg8.columns=['S1','S2','S3','S4','S5']
         dfeg8
```

Out[15]:

	S1	S2	S3	S4	S5
R1	12	23	34	45	56
R2	-10.0	30.0	-45.0	30.34	None
R3	aa	bb	cc	dd	ee

Accessing the elements of DataFrame:

.loc() property: The loc property is used to access a group of rows and columns by label(s) or a boolean array. .loc[] is primarily label based, but may also be used with a boolean array.

Allowed inputs are:

- A single label, e.g., 5 or 'a', (note that 5 is interpreted as a label of the index, and never as an integer position along the index).
- A list or array of labels, e.g. ['a', 'b', 'c'].
- A slice object with labels, e.g., 'a':'f'.
- A boolean array of the same length as the axis being sliced, e.g. [True, False, True].
- A callable function with one argument (the calling Series or DataFrame) and that returns valid output for indexing (one of the above)

Examples:

Consider the following DataFrame:

```
In [13]: dfres=pd.DataFrame([[30,40,45, 56,48],[34,56,66,59,39],[44,72,90,68,81]],
                            index=['Maths','Stats','Comp'],
                            columns=['Raj','Sita','Rahim','Joseph','Priti'])
         dfres
```

Out[13]:

	Raj	Sita	Rahim	Joseph	Priti
Maths	30	40	45	56	48
Stats	34	56	66	59	39
Comp	44	72	90	68	81

1. To see the marks obtained in Maths by all the students:

```
In [14]: dfres.loc['Maths']

Out[14]: Raj       30
         Sita      40
         Rahim     45
         Joseph    56
         Priti     48
         Name: Maths, dtype: int64
```

   The same output is obtained by dfres.loc[‘Maths’,:]

2. To see the marks obtained by a particular students say ‘Raj’

```
In [16]: dfres.loc[:,'Raj']

Out[16]: Maths    30
         Stats    34
         Comp     44
         Name: Raj, dtype: int64
```

   The same result can be obtained by print(dfres[‘Raj’])

3. To display the marks of two or more students:

```
In [29]: dfres.loc[:,['Raj','Rahim']]
```

Out[29]:

	Raj	Rahim
Maths	30	45
Stats	34	66
Comp	44	90

4. To display the marks obtained by group of students in continuous columns:

```
In [31]: dfres.loc[:,'Raj':'Rahim']
```

Out[31]:

	Raj	Sita	Rahim
Maths	30	40	45
Stats	34	56	66
Comp	44	72	90

5. To display the all values of two or more rows:

```
In [34]: dfres.loc[['Maths','Comp']]
```

Out[34]:

	Raj	Sita	Rahim	Joseph	Priti
Maths	30	40	45	56	48
Comp	44	72	90	68	81

6. To display the all values from continuous rows:

```
In [35]: dfres.loc['Maths':'Comp']
```

Out[35]:

	Raj	Sita	Rahim	Joseph	Priti
Maths	30	40	45	56	48
Stats	34	56	66	59	39
Comp	44	72	90	68	81

7. To display the all values of the given rows for particular column:

```
In [39]: dfres.loc['Maths':'Stats':,'Joseph']
```

```
Out[39]: Maths    56
         Stats    59
         Name: Joseph, dtype: int64
```

8. To display the columns satisfying the given condition on row:

```
In [18]: dfres.loc['Maths']>50

Out[18]: Raj       False
         Sita      False
         Rahim     False
         Joseph     True
         Priti     False
         Name: Maths, dtype: bool
```

9. To display the rows satisfying the given conditions on rows.

```
In [29]: dfres.loc[:,['Raj','Rahim']]

Out[29]:
```

	Raj	Rahim
Maths	30	45
Stats	34	66
Comp	44	90

iloc() Property: The iloc property returns purely integer-location based indexing for selection by position.

.iloc[] is primarily integer position based (from 0 to length-1 of the axis), but may also be used with a boolean array.

Allowed inputs are:

- An integer, e.g. 5.
- A list or array of integers, e.g. [4, 3, 0].
- A slice object with ints, e.g. 1:7.
- A boolean array.
- A callable function with one argument (the calling Series or DataFrame) and that returns valid output for indexing (one of the above).

Examples:

1. To display specific row say first row:

```
In [52]: dfres.iloc[0]

Out[52]: Raj       30
         Sita      40
         Rahim     45
         Joseph    56
         Priti     48
         Name: Maths, dtype: int64
```

2. To display particular element say first row and second column element:

```
In [53]: dfres.iloc[0,1]

Out[53]: 40
```

3. To display first two rows:

```
In [54]: dfres.iloc[0:2]
```

Out[54]:

	Raj	Sita	Rahim	Joseph	Priti
Maths	30	40	45	56	48
Stats	34	56	66	59	39

4. To display all rows and all columns from 2nd row and 2nd column:

```
In [59]: dfres.iloc[1:,1:]
```

Out[59]:

	Sita	Rahim	Joseph	Priti
Stats	56	66	59	39
Comp	72	90	68	81

Operations on rows and columns in DataFrames

We can perform some basic operations on rows and columns of a DataFrame like selection, deletion, addition, and renaming, as discussed

in this section. Following is the operation that we can perform in on the columns and rows of DataFrame.

1. Rename columns/rows
2. Add columns/rows
3. Delete columns/rows

Renaming Columns/Rows:

Consider the following examples:

```
In [3]: dfresult = pd.DataFrame({'Name': ['AAA','BBB','CCC','DDD'],
                                 'Hindi':[23,45,65,23],'English':[45,65,76,83],
                                 'Maths':[34,54,66,77]})
        dfresult
```

Out[3]:

	Name	Hindi	English	Maths
0	AAA	23	45	34
1	BBB	45	65	54
2	CCC	65	76	66
3	DDD	23	83	77

Consider the following Panda statement:

```
In [4]: dfresult.index={401,402,403,404}
        dfresult
```

Out[4]:

	Name	Hindi	English	Maths
401	AAA	23	45	34
402	BBB	45	65	54
403	CCC	65	76	66
404	DDD	23	83	77

The above method is used when one want to change the entire index of the DataFrame. But when one want to change particular indexes or columns then above method is not useful. At that time we use rename() method.

rename() method: Pandas rename() method is used to rename any index, column or row.

It mainly alters the axes labels based on some of the mapping (dict or Series) or the arbitrary function. The function must be unique and should range from 1 to -1. The labels will be left, if it is not contained in a dict or Series. If you list some extra labels, it will throw an error.

Following is the syntax of rename() method.

DataFrame.rename(mapper=None, index=None, columns=None, axis=None, copy=True, inplace=False, level=None)

Mapper: Dictionary value, key refers to the old name and value refers to new name.

Index: Alternative to specifying axis (mapper, axis=0 is equivalent to index=mapper).

Columns: Alternative to specifying axis (mapper, axis=1 is equivalent to columns=mapper).

Axis: int or string value, 0/'row' for Rows and 1/'columns' for Columns.

copy: It refers to a boolean value that copies the underlying data. The default value of the copy is True.

inplace: It refers to a boolean value and checks whether to return the new DataFrame or not. If it is true, it makes the changes in the original DataFrame. The default value of the inplace is True.

Examples:

1. Changing any indexes:

```
In [5]: dfresult=dfresult.rename({401:501,402:502}, axis='index')
        dfresult
```

Out[5]:

	Name	Hindi	English	Maths
501	AAA	23	45	34
502	BBB	45	65	54
403	CCC	65	76	66
404	DDD	23	83	77

Equivalently,

```
In [6]: dfresult=dfresult.rename(index={403:503, 404:504})
        dfresult
```

Out[6]:

	Name	Hindi	English	Maths
501	AAA	23	45	34
502	BBB	45	65	54
503	CCC	65	76	66
504	DDD	23	83	77

2. To change the columns name:

```
In [7]: dfresult=dfresult.rename(columns={'Hindi':'Statstics'})
        dfresult
```

Out[7]:

	Name	Statstics	English	Maths
501	AAA	23	45	34
502	BBB	45	65	54
503	CCC	65	76	66
504	DDD	23	83	77

Equivalently,

```
In [8]: dfresult=dfresult.rename({'English':'AC'}, axis='columns')
        dfresult
```

Out[8]:

	Name	Statstics	AC	Maths
501	AAA	23	45	34
502	BBB	45	65	54
503	CCC	65	76	66
504	DDD	23	83	77

Adding new Row and Columns in DataFrame:

Columns can easily be added in a DataFrame as shown in the following example:

```
In [9]: dfresult['Physics']=[56,87,45,46]
        dfresult
```

Out[9]:

	Name	Statstics	AC	Maths	Physics
501	AAA	23	45	34	56
502	BBB	45	65	54	87
503	CCC	65	76	66	45
504	DDD	23	83	77	46

A row can be added using loc(), append() and iloc() method.

Adding rows using append(): append() used to add the row at the end of DataFrame. The parameter ignore_index is set to be true.

```
In [10]: dfresult.append({'Name':'FFF','Maths':55,'Physics':44},ignore_index=True)
```

Out[10]:

	Name	Statstics	AC	Maths	Physics
0	AAA	23.0	45.0	34	56
1	BBB	45.0	65.0	54	87
2	CCC	65.0	76.0	66	45
3	DDD	23.0	83.0	77	46
4	FFF	NaN	NaN	55	44

Adding rows using loc[]: The loc[] property of DataFrame add the row at the end of DataFrame.

```
In [11]: dfresult.loc[505]=['EEE',45,66,52,45]
         dfresult
```

Out[11]:

	Name	Statstics	AC	Maths	Physics
501	AAA	23	45	34	56
502	BBB	45	65	54	87
503	CCC	65	76	66	45
504	DDD	23	83	77	46
505	EEE	45	66	52	45

For pre-existing index, loc[] will update the values.

```
In [12]: dfresult.loc[504]=['DDD',55,66,77,88]
         dfresult
```

Out[12]:

	Name	Statstics	AC	Maths	Physics
501	AAA	23	45	34	56
502	BBB	45	65	54	87
503	CCC	65	76	66	45
504	DDD	55	66	77	88
505	EEE	45	66	52	45

Adding row using iloc[] property: iloc[] used to add the row in the DataFrame at the specified index.

Deleting row or column:

We can use the DataFrame.drop() method to delete rows and columns from a DataFrame. We need to specify the names of the labels to be dropped and the axis from which they need to be dropped. To delete a row, the parameter axis is assigned the value 0 (or 'index') and for deleting a column, the parameter axis is assigned the value 1 (or 'columns').

```
In [13]: dfresult=dfresult.drop(503,axis='index')
         dfresult
```

Out[13]:

	Name	Statstics	AC	Maths	Physics
501	AAA	23	45	34	56
502	BBB	45	65	54	87
504	DDD	55	66	77	88
505	EEE	45	66	52	45

```
In [14]: dfresult=dfresult.drop(['Statstics','AC'],axis='columns')
         dfresult
```

Out[14]:

	Name	Maths	Physics
501	AAA	34	56
502	BBB	54	87
504	DDD	77	88
505	EEE	52	45

Guru Nanak Khalsa College (Autonomous) – Mumbai 19
Department of Mathematics & Statistics

Unit II Chapter 3 Basic operations on DataFrame

In the previous chapter we saw how to create DataFrame, how to rename, add delete the row index and columns in the DataFrame. In this chapter we will see some of the attributes of DataFrame and how to arrange DataFrame in particular order by sorting.
Some Attributes of DataFrame:

1. DataFrame.index : To display row labels

```
In [3]: dfresult.index

Out[3]: Index(['Maths', 'Science', 'Hindi'], dtype='object')
```

2. DataFrame.columns: to displays the columns labels.

```
In [4]: dfresult.columns

Out[4]: Index(['Arnab', 'Ramit', 'Samridhi', 'Riya', 'Mallika'], dtype='object')
```

3. DataFrame.values: to display data type of each column in the DataFrame.

```
In [5]: dfresult.values

Out[5]: array([[90, 92, 89, 81, 94],
               [91, 81, 91, 71, 95],
               [97, 96, 88, 67, 99]], dtype=int64)
```

4. DataFrame.shape : To display a tuple representing the dimensionality of the DataFrame.

```
In [6]: dfresult.shape

Out[6]: (3, 5)
```

5. DataFrame.size: To display a integer representing the total number of values in the data frame.

```
In [7]: dfresult.size

Out[7]: 15
```

6. DataFrame.empty : To returns the value True if DataFrame is empty and False otherwise.

```
In [9]: dfresult.empty

Out[9]: False
```

7. DataFrame.T: To transpose the DataFrame. Means, row indices and column labels of the DataFrame replace each other's position.

```
In [8]: dfresult.T
```

Out[8]:

	Maths	Science	Hindi
Arnab	90	91	97
Ramit	92	81	96
Samridhi	89	91	88
Riya	81	71	67
Mallika	94	95	99

8. DataFrame.head(n): To display the first n rows in the DataFrame. If n is not mentioned, then it will display first 5 rows.

```
In [10]: dfresult.head()
```

Out[10]:

	Arnab	Ramit	Samridhi	Riya	Mallika
Maths	90	92	89	81	94
Science	91	81	91	71	95
Hindi	97	96	88	67	99

9. DataFrame.tail(n): To display the last n rows in the DataFrame.

```
In [11]: dfresult.tail(2)
```

Out[11]:

	Arnab	Ramit	Samridhi	Riya	Mallika
Science	91	81	91	71	95
Hindi	97	96	88	67	99

Importing a CSV file to a DataFrame:
Panda read_csv() file is used to read the data from CSV (comma separated values) file as shown in the example.

```
In [12]: marks=pd.read_csv("E:\Python Programming\panda_prac.csv",sep=',', header=0)
         marks
```

Out[12]:

	rollno	name	PI	PII	PIII	PIV
0	401	AAA	34	45	43	23
1	402	BBB	33	12	23	31
2	403	CCCC	44	13	24	32
3	404	DDD	55	14	25	33
4	405	EEE	11	15	26	34

In the above example

1. The first argument is the name of the comma separated data file along with its path.

2. The argument sep specifies whether the values are separated by comma, semicolon, tab, or any other character. The default value for sep is a space.
3. The parameter header specifies the number of the row whose values are to be used as the column names. It also marks the start of the data to be fetched. header=0 implies that column names are inferred from the first line of the file. By default, header=0.

Sorting: Arranging dataframe in a lexicographic order refers as sorting of the dataframe. A dataframe can be sort in index wise, column wise or values wise. Sorting can take place in ascending order as well as descending order. The sort_index() method is used to sort the dataframe along rows and columns. Consider the following dataframe.

```
In [3]: dfem=pd.read_csv('E:\Class 22 - 23\pract.csv',sep=',',header=0)
        dfem.index=[301, 304,501,222,401]
        dfem
```

Out[3]:

	Name	Empno	Desig	Dept	Sal	HRA
301	Roshan	203	Manger	Finance	39999	3400
304	Amit	210	Clerk	Prog	32000	2100
501	Rita	204	HR	HR	40000	2900
222	Ramesh	201	CEO	Finance	90000	3500
401	Maya	212	Prog	Prog	55000	1200

Examples:

1. To sort in ascending order in the order of indexes.

```
In [4]: dfem.sort_index()
```

Out[4]:

	Name	Empno	Desig	Dept	Sal	HRA
222	Ramesh	201	CEO	Finance	90000	3500
301	Roshan	203	Manger	Finance	39999	3400
304	Amit	210	Clerk	Prog	32000	2100
401	Maya	212	Prog	Prog	55000	1200
501	Rita	204	HR	HR	40000	2900

2. To sort in descending order in the order of indexes.

```
In [5]: dfem.sort_index(ascending=False)
```

Out[5]:

	Name	Empno	Desig	Dept	Sal	HRA
501	Rita	204	HR	HR	40000	2900
401	Maya	212	Prog	Prog	55000	1200
304	Amit	210	Clerk	Prog	32000	2100
301	Roshan	203	Manger	Finance	39999	3400
222	Ramesh	201	CEO	Finance	90000	3500

3. To sort in ascending order along the columns:

```
In [6]: dfem.sort_index(axis=1)
```

Out[6]:

	Dept	Desig	Empno	HRA	Name	Sal
301	Finance	Manger	203	3400	Roshan	39999
304	Prog	Clerk	210	2100	Amit	32000
501	HR	HR	204	2900	Rita	40000
222	Finance	CEO	201	3500	Ramesh	90000
401	Prog	Prog	212	1200	Maya	55000

4. To sort in descending order along the columns:

```
In [7]: dfem.sort_index(axis=1, ascending = False)
```

Out[7]:

	Sal	Name	HRA	Empno	Desig	Dept
301	39999	Roshan	3400	203	Manger	Finance
304	32000	Amit	2100	210	Clerk	Prog
501	40000	Rita	2900	204	HR	HR
222	90000	Ramesh	3500	201	CEO	Finance
401	55000	Maya	1200	212	Prog	Prog

Method sort_values(): This method is used to sort the dataframe along the values of the dataframe. While sorting in ascending order the values with NaN will appear at the end.

Examples:

1. Sorting along a particular column.

```
In [8]: dfem.sort_values('Sal')
```

Out[8]:

	Name	Empno	Desig	Dept	Sal	HRA
304	Amit	210	Clerk	Prog	32000	2100
301	Roshan	203	Manger	Finance	39999	3400
501	Rita	204	HR	HR	40000	2900
401	Maya	212	Prog	Prog	55000	1200
222	Ramesh	201	CEO	Finance	90000	3500

2. Sorting along multiple columns and descending order:

```
In [9]: dfem.sort_values(['Sal', 'HRA'],ascending=False)
```

Out[9]:

	Name	Empno	Desig	Dept	Sal	HRA
222	Ramesh	201	CEO	Finance	90000	3500
401	Maya	212	Prog	Prog	55000	1200
501	Rita	204	HR	HR	40000	2900
301	Roshan	203	Manger	Finance	39999	3400
304	Amit	210	Clerk	Prog	32000	2100

Descriptive Statistics:

Once the dataframe is created, it can be analysis with all types of statistical function. These methods can be applied on dataframe index as well as columns. Some of these functions we will operate over the following dataframe.

```
In [16]: result
```

Out[16]:

	Maths	Stats	Physics	Chemsitry	Applied Comp
Meena	44	12	32	56	76
Saroj	45	65	66	70	49
Ismail	76	34	67	66	23
Nagesh	90	67	84	71	80
Robin	56	66	12	32	45
Sukhwinder	23	32	54	29	44
Mumtaj	29	85	45	69	30

1. max() : This method returns maximum value along the index as well as columns.
   Examples:
   a) To find maximum value in all columns.

```
In [17]: result.max()

Out[17]: Maths          90
         Stats          85
         Physics        84
         Chemsitry      71
         Applied Comp   80
         dtype: int64
```

   b) To find the maximum of selected columns:

```
In [18]: result[['Maths','Stats']].max()

Out[18]: Maths    90
         Stats    85
         dtype: int64
```

c) To find maximum along the rows, use argument *axis =1*.

```
In [19]: result.max(axis=1)

Out[19]: Meena          76
         Saroj          70
         Ismail         76
         Nagesh         90
         Robin          66
         Sukhwinder     54
         Mumtaj         85
         dtype: int64
```

d) To find the maximum for particular row, we use loc[] method.

```
In [20]: result.loc['Meena'].max()

Out[20]: 76
```

e) To find the maximum along more than one rows.

```
In [22]: result.loc[['Sukhwinder','Mumtaj']].max()

Out[22]: Name             Sukhwinder
         Maths                    29
         Stats                    85
         Physics                  54
         Chemsitry                69
         Applied Comp             44
         dtype: object
```

f) When a dataframe's some columns are string and some columns are numeric then we set the value of 'numeric_only

```
In [21]: result.min(numeric_only=True,axis=1)

Out[21]: Meena          12
         Saroj          45
         Ismail         23
         Nagesh         67
         Robin          12
         Sukhwinder     23
         Mumtaj         29
         dtype: int64
```

g) sum(): This method gives sum of each row or columns.
To find sum of each rows:

```
In [49]: result.sum()

Out[49]: Name           ServashMeenaSarojIsmailNageshRobinSukhwinderRa...
         Maths                                                        542
         Stats                                                        439
         Physics                                                      460
         Chemsitry                                                    564
         Applied Comp                                                 479
         Class                                          FYSYFYSYTYTYSYFYTYSY
         dtype: object
```

h) To display sum of selected columns:

```
In [50]: result[['Maths','Stats']].sum()

Out[50]: Maths    542
         Stats    439
         dtype: int64
```

i) To display the sum along the rows:

```
In [51]: result.loc[['Robin','Nagesh']].sum(numeric_only=True, axis=1)

Out[51]: Robin     211
         Nagesh    392
         dtype: int64
```

j) mean() is used to find the mean.
To find the mean of all columns:

```
In [52]: result.mean(numeric_only=True)

Out[52]: Maths           54.2
         Stats           43.9
         Physics         46.0
         Chemsitry       56.4
         Applied Comp    47.9
         dtype: float64
```

k) To find the mean of selected columns:

```
In [53]: result[['Maths','Physics']].mean()

Out[53]: Maths      54.2
         Physics    46.0
         dtype: float64
```

l) To find the mean of selected rows.

```
In [54]: result.loc[['Mumtaj','Robin']].mean(numeric_only=True, axis=1)

Out[54]: Mumtaj    51.6
         Robin     42.2
         dtype: float64
```

m) To find the mean of all rows:

```
In [56]: result.mean(numeric_only=True, axis=1)

Out[56]: Servash       50.8
         Meena         44.0
         Saroj         59.0
         Ismail        53.2
         Nagesh        78.4
         Robin         42.2
         Sukhwinder    36.4
         Rajnish       54.0
         Mumtaj        51.6
         Rohit         27.2
         dtype: float64
```

n) On the same line median() function is used to find the median, mode() is used to find the mode, var() is used to find variance and std() is used to find the standard deviation of rows or columns.

o) count() method is used to count the number of rows and columns.
To count the number of rows:

```
In [57]: result.count()

Out[57]: Name            10
         Maths           10
         Stats           10
         Physics         10
         Chemsitry       10
         Applied Comp    10
         Class           10
         dtype: int64
```

p) To count the number of columns:

```
In [58]: result.count(axis=1)

Out[58]: Servash      7
         Meena        7
         Saroj        7
         Ismail       7
         Nagesh       7
         Robin        7
         Sukhwinder   7
         Rajnish      7
         Mumtaj       7
         Rohit        7
         dtype: int64
```

q) quantile() method is used to calculate the quartile along the given axis. Default axis is column. To calculate quartile along the column, parameter *axis=1* is added in the quantile(). There are three quartiles, first quartile (25%), second quartile (50%) and third quartile (75%). By default, it calculates 50% quartile. To get 25% or 75% quartile, the argument 025 or 0.75 is added the quantile() method.

To calculate 50% quartile along the row:

```
In [59]: result.quantile()

Out[59]: Maths           50.5
         Stats           39.0
         Physics         49.5
         Chemsitry       61.0
         Applied Comp    46.0
         Name: 0.5, dtype: float64
```

r) To calculate 75% quartile along the columns:

```
In [60]: result.quantile(.75)

Out[60]: Maths           73.75
         Stats           65.75
         Physics         63.00
         Chemsitry       69.75
         Applied Comp    61.75
         Name: 0.75, dtype: float64
```

s) To display 25% and 75% quartile along the row:

```
In [61]: result.quantile([.25,.75], axis =1)

Out[61]:
```

	Servash	Meena	Saroj	Ismail	Nagesh	Robin	Sukhwinder	Rajnish	Mumtaj	Rohit
0.25	44.0	32.0	49.0	34.0	71.0	32.0	29.0	36.0	30.0	10.0
0.75	56.0	56.0	66.0	67.0	84.0	56.0	44.0	67.0	69.0	20.0

t) describe() method is used to describe the basic statistical quantity for each column.

```
In [62]: result.describe()
```

Out[62]:

	Maths	Stats	Physics	Chemsitry	Applied Comp
count	10.000000	10.000000	10.00000	10.000000	10.000000
mean	54.200000	43.900000	46.00000	56.400000	47.900000
std	22.841483	25.795995	23.94902	23.119016	20.946758
min	23.000000	9.000000	10.00000	20.000000	19.000000
25%	36.500000	26.750000	33.00000	38.000000	33.500000
50%	50.500000	39.000000	49.50000	61.000000	46.000000
75%	73.750000	65.750000	63.00000	69.750000	61.750000
max	90.000000	85.000000	84.00000	95.000000	80.000000

Aggregate function: The aggregation function is one which takes multiple individual values and returns a summary. In the majority of the cases, this summary is a single value. Aggregate functions are max(),min(), sum(), count(), std(), var().
Examples:

a) To find the maximum along the column:

```
In [64]: result.agg('max')
```

```
Out[64]: Name            Sukhwinder
         Maths                   90
         Stats                   85
         Physics                 84
         Chemsitry               95
         Applied Comp            80
         Class                   TY
         dtype: object
```

b) To find the sum along the row:

```
In [72]: result.agg('sum',axis=1)
```

```
Out[72]: Series([], dtype: float64)
```

c) To use more than one aggregation functions:

```
In [73]: result.agg(['std','mean'])
```

Out[73]:

	Maths	Stats	Physics	Chemsitry	Applied Comp
std	22.841483	25.795995	23.94902	23.119016	20.946758
mean	54.200000	43.900000	46.00000	56.400000	47.900000

d) Pandas dataframe.corr() is used to find the pairwise correlation of all columns in the dataframe. Any NaN values are automatically excluded. For any non-numeric data type columns in the dataframe it is ignored.

Syntax: DataFrame.corr(method='pearson', min_periods=1)

Where method:

pearson: standard correlation coefficient
kendall: Kendall Tau correlation coefficient
spearman: Spearman rank correlation
min_periods: Minimum number of observations required per pair of columns to have a valid result.

```
In [61]: result.corr()
```

Out[61]:

	Maths	Stats	Physics	Chemsitry	Applied Comp
Maths	1.000000	-0.180427	0.070075	0.155745	-0.043613
Stats	-0.180427	1.000000	0.377152	0.258486	0.079970
Physics	0.070075	0.377152	1.000000	0.512332	0.355490
Chemsitry	0.155745	0.258486	0.512332	1.000000	0.274274
Applied Comp	-0.043613	0.079970	0.355490	0.274274	1.000000

e) Pandas dataframe.cov() is used to compute pairwise covariance of columns. If some of the cells in a column contain NaN value, then it is ignored.

Syntax: DataFrame.cov(min_periods=None)

where min_periods is minimum number of observations required per pair of columns to have a valid result.

```
In [62]: result.cov()
```

Out[62]:

	Maths	Stats	Physics	Chemsitry	Applied Comp
Maths	521.733333	-106.311111	38.333333	82.244444	-20.866667
Stats	-106.311111	665.433333	233.000000	154.155556	43.211111
Physics	38.333333	233.000000	573.555556	283.666667	178.333333
Chemsitry	82.244444	154.155556	283.666667	534.488889	132.822222
Applied Comp	-20.866667	43.211111	178.333333	132.822222	438.766667

Guru Nanak Khalsa College (Autonomous) – Mumbai 19
Department of Mathematics & Statistics

Unit II Chapter 4 Group By Function in Panda

Group by() function:

groupby() function:

In pandas, DataFrame.groupby() function is used to split the data into groups based on some criteria. The groupby() function works based on a split-apply-combine strategy which is shown below using a 3-step process:

- Step 1: Split the data into groups by creating a groupby object from the original DataFrame.
- Step 2: Apply the required function.
- Step 3: Combine the results to form a new DataFrame.

Consider the following dataframe:

```
In [33]: emp=pd.DataFrame({'Name':['Akbar','Amar','Antony','Seeta','Geeta','Ram','Rahim','Sherya'],
                           'Dept':['Finance','HR','Marketing','HR','Prog','Finance','Finance','Prog'],
                           'Basic sal':[45034,56034,56980,34000,56981,65090,67890,88090]})
         emp
```

Out[33]:

	Name	Dept	Basic sal
0	Akbar	Finance	45034
1	Amar	HR	56034
2	Antony	Marketing	56980
3	Seeta	HR	34000
4	Geeta	Prog	56981
5	Ram	Finance	65090
6	Rahim	Finance	67890
7	Sherya	Prog	88090

1. Creating group on department:

```
In [34]: dep=emp.groupby('Dept')
         print(dep)

<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000001
7A913E7BB0>
```

2. Displaying the first record from each group.

```
In [36]: dep.first()
```

Out[36]:

Dept	Name	Basic sal
Finance	Akbar	45034
HR	Amar	56034
Marketing	Antony	56980
Prog	Geeta	56981

3. To display first 2 record from each group:

```
In [37]: dep.head(2)
```

Out[37]:

	Name	Dept	Basic sal
0	Akbar	Finance	45034
1	Amar	HR	56034
2	Antony	Marketing	56980
3	Seeta	HR	34000
4	Geeta	Prog	56981
5	Ram	Finance	65090
7	Sherya	Prog	88090

4. To display last record from each group.

```
In [38]: dep.tail(1)
```

Out[38]:

	Name	Dept	Basic sal
2	Antony	Marketing	56980
3	Seeta	HR	34000
6	Rahim	Finance	67890
7	Sherya	Prog	88090

5. To display the number of record from each group:

```
In [39]: dep.size()
```

Out[39]:

```
Dept
Finance      3
HR           2
Marketing    1
Prog         2
dtype: int64
```

6. To display the position of records in each group:

```
In [40]: dep.groups
```

Out[40]:

```
{'Finance': [0, 5, 6], 'HR': [1, 3], 'Marketing': [2], 'Prog':
[4, 7]}
```

7. To display the records from the selected group.

```
In [41]: dep.get_group('HR')
```

Out[41]:

	Name	Dept	Basic sal
1	Amar	HR	56034
3	Seeta	HR	34000

8. Add one more column in dataframe:

```
In [43]: emp['Year']=[2000,2001,2000,2003,2001,2000,2001,2003]
         emp
```

Out[43]:

	Name	Dept	Basic sal	Year
0	Akbar	Finance	45034	2000
1	Amar	HR	56034	2001
2	Antony	Marketing	56980	2000
3	Seeta	HR	34000	2003
4	Geeta	Prog	56981	2001
5	Ram	Finance	65090	2000
6	Rahim	Finance	67890	2001
7	Sherya	Prog	88090	2003

9. Now creating groups with more than one field:

```
In [45]: dep1=emp.groupby(['Dept','Year'])
         dep1
```

```
Out[45]: <pandas.core.groupby.generic.DataFrameGroupBy object at 0x000001
         7A9136C340>
```

10. To display first record of dep1:

```
In [46]: dep1.first()
```

Out[46]:

		Name	Basic sal
Dept	Year		
Finance	2000	Akbar	45034
	2001	Rahim	67890
HR	2001	Amar	56034
	2003	Seeta	34000
Marketing	2000	Antony	56980
Prog	2001	Geeta	56981
	2003	Sherya	88090

11. To display the size of dep1:

```
In [48]: dep1.size()
```

```
Out[48]: Dept       Year
         Finance    2000    2
                    2001    1
         HR         2001    1
                    2003    1
         Marketing  2000    1
         Prog       2001    1
                    2003    1
         dtype: int64
```

12. To apply aggregate function:

```
In [50]: dep1.agg('mean')
```

Out[50]:

		Basic sal
Dept	**Year**	
Finance	**2000**	55062.0
	2001	67890.0
HR	**2001**	56034.0
	2003	34000.0
Marketing	**2000**	56980.0
Prog	**2001**	56981.0
	2003	88090.0