UNIT 1
Data Handling Using Pandas

Python module- A python module is a python script file(.py file) containing variables, python classes, functions, statements etc.

Python Library/package- A Python library is a collection of modules that together cater to a specific type of need or application. The advantage of using libraries is that we can directly use functions/methods for performing specific type of application instead of rewriting the code for that particular use. They are used by using the import command as-

import libraryname

at the top of the python code/script file.
Some examples of Python Libraries-

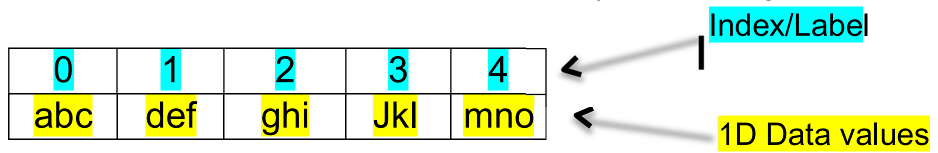
1. **Python standard library-**It is a collection of library which is normally distributed along with Python installation. Some of them are-
   a. **math module-** provides mathematical functions
   b. **random module-** provides functions for generating pseudo-random numbers.
   c. **statistics module-** provides statistical functions
2. **Numpy (Numerical Python) library-** It provides functions for working with large multi-dimensional arrays(ndarrays) and matrices. NumPy provides a large set of mathematical functions that can operate quickly on the entries of the ndarray without the need of loops.
3. **Pandas (PANel + DAta) library-** Pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool. Pandas is built on top of NumPy, relying on ndarrayand its fast andefficient array based mathematical functions.
4. **Matplotlib library-** It provides functions for plotting and drawing graphs.

Data Structure- Data structure is the arrangement of data in such a way that permits efficient access and modification.

Pandas Data Structures- Pandas offers the following data structures-

a) Series - 1D array
b) DataFrame - 2D array

Series- Series is a one-dimensional array with homogeneous data.



Key features of Series-

- A Series has only one dimension, i.e. one axis
- Each element of the Series can be associated with an index/label that can be used to access the datavalue. By default the index starts with 0,1,2,3… but it can be set to any other data type also.
- Series is data mutable i.e. the data values can be changed in-place in memory
- Series is size immutable i.e. once a series object is created in memory with a fixed number of elements, then the number of elements cannot be changed in place. Although the series object canbe assigned a different set of values it will refer to a different location in memory.
- All the elements of the Series are homogenous data i.e. their data type is the same. For example.

0	1	2	3	4
223	367	456	339	927

all data is of int type

a	b	c	de	fg
1	def	10.5	Jkl	True

all data is of object type

Creating a Series- A series object can be created by calling the Series() method in the following ways-

a) **Create an empty Series-** A Series object not containing any elements is an empty Series. It can becreated as follows-

```
import pandas as pd
s1=pd.Series() print(s1)

o/p-
Series([], dtype: float64)
```

b) **Create a series from array without index-** A numpy 1D array can be used to create a Series object asshown below. The default index is 0, 1, 2, …

```
import pandas as pd
import numpy as np
a1=np.array(['hello', 'world', 'good', np.NaN])
s1=pd.Series(a1)
print(s1)
```

```
o/p-
0   hello
1   world
2   good
3    nan
dtype: object
```

c) **Create a series from array with index-** The default index for a Series object can be changed and specified by the programmer by using the index parameter and enclosing the index in square brackets. The number of elements of the array must match the number of index specified otherwisepython gives an error.

```
#Creating a Series object using numpy array and specifying index
import pandas as pd
import numpy as np
a1=np.array(['hello', 'world', 'good', 'morning'])
s1=pd.Series(a1, index=[101, 111, 121, 131])
print(s1)
```

```
o/p-
101   hello
111   world
121   good
131   morning

dtype: object
```

d) **Create a Series from dictionary-** Each element of the dictionary contains a key:value pair. The key ofthe dictionary becomes the index of the Series object and the value of the dictionary becomes the data.

```
#4 Creating a Series object from dictionary
import pandas as pd
d={101:'hello', 111:'world', 121:'good', 131:'morning'} s1=pd.Series(d)
print(s1)

o/p-
101      hello
111      world
121      good
131      morning

dtype: object
```

a) **Create a Series from dictionary, reordering the index-** When we are creating a Series object from adictionary then we can specify which all elements of the dictionary, we want to include in the Seriesobject and in which order by specifying the index argument while calling the Series() method.

- If any key of the dictionary is missing in the index argument, then that element is not addedto the Series object.

- If the index argument contains a key not present in the dictionary then a value of NaN isassigned to that particular index.
- The order in which the index arguments are specified determines the order of the elementsin the Series object.

```
#5 Creating a Series object from dictionary reordering
the index
import pandas as pd

d={101:'hello', 111:'world', 121:'good', 131:'morning'}
s1=pd.Series(d, index=[131, 111, 121, 199])
print(s1)

o/p-
131     morning
111     world
121     good
199     NaN

dtype: object
```

Create a Series from a scalar value- A Series object can be created from a single value i.e. a scalarvalue and that scalar value can be repeated many times by specifying the index arguments that many number of times.

```
#6 Creating a Series object from scalar value
import pandas as pd

s1=pd.Series(7, index=[101, 111, 121])print(s1)

o/p-
101        7
111        7
121        7
dtype: int64
```

a) **Create a Series from a List-** A Series object can be created from a list as shown below.

```
#7 Creating a Series object from list

import pandas as pd L=['abc', 'def', 'ghi', 'jkl']s1=pd.Series(L)
print(s1)

o/p-
0            abc
1            def ghi
2            jkl
3

dtype: object
```

b) **Create a Series from a Numpy Array (using various array creation methods) -** A Series object can becreated from a numpy array as shown below. All the methods of numpy array creation can be used to create a Series object.

```
#7a Creating a Series object from list
import pandas as pd
import numpy as np

#a      Create an array consisting of elements of a list [2,4,7,10,
a1=np.array([2,4,7,10, 13.5, 20.4])
s1=pd.Series(a1)
print('s1=', s1)

#b      Create an array consisting of ten
a2=np.zeros(10)
s2=pd.Series(a2, index=range(101,
111)) print('s2=', s2)

#c   Create an array consisting of five
a3=np.ones(5)
s3=pd.Series(a3)
print('s3=', s3)
```

```
#d.Create an array consisting of the elements from
1.1, 1.2, 1.3,1.4, 1.5, 1.6, 1.7
a4=np.arange(1.1,1.8,0.1)
s4=pd.Series(a4)print('s4=', s4)

#e.    Create an array of 10 elements which are linearly spaced between 1 and 10
(both inclusive)
a5=np.linspace(1,10,4)
s5=pd.Series(a5)print('s5=', s5)

#f.    Create an array containing each of the characters of the word ‘helloworld’
a6=np.fromiter('helloworld', dtype='U1')
s6=pd.Series(a6)print('s6=', s6)
```

o/p:

```
s1= 0  2.0
1   4.0
2   7.0
3   10.0
4   13.5
5   20.4
dtype: float64
```

```
          s2= 101  0.0
          102    0.0
          103    0.0
          104    0.0
          105    0.0
          106    0.0
          107    0.0
          108    0.0
          109    0.0
          110    0.0
          dtype: float64
          s3= 0 1.0
          1    1.0
          2    1.0
          3    1.0
          4    1.0
          dtype: float64
          s4= 0 1.1
          1    1.2
          2    1.3
          3    1.4
          4    1.5
          5    1.6
          6    1.7
          dtype: float64
          s5= 01.0
1         4.0
2         2   7.0
3         3   10.0
dtype: float64
s6= 0 h
1         e
2         l
3         l
4         o
5         w
6         o
7         r
8         l
9         d
dtype: object
```

Index

SERIES : Operations on Series objects-

1. **Accessing elements of a Series object**

   The elements of a series object can be accessed using different methods as shown below-

   a) Using the indexing operator []

   The square brackets [] can be used to access a data value stored in a Series object. The indexof the element must be entered within the square brackets. If the index is a string then the index must be written in quotes. If the index is a number then the index must be written without the quotes. Attempting to use an index which does not exist leads to error.

```
#8 Accessing elements of Series using index
import pandas as pd

d={101:'hello', 'abc':'world', 121:'good', 131:'morning'}s=pd.Series(d)
print(s['abc'])print(s[131])

o/p- world
   morning
```

a) Using the *loc* property of the Series object

The *loc* property of a Series object can be used to access a range of data values using thelabel/index name inside [] brackets in the following ways:

1. A single index can be passed to the loc property. This will return back a single value.
2. A list of indexes can be passed. This will return back a Series object containing themultiple values
3. A slice notation using labels/index such as *startindex:stopindex.* Here contrary to theslice notation the ending index value also is included in the result.
4. A boolean array of the same length as the axis being sliced, e.g. [True, False, True].

b) Using the *iloc* property of the Series object

The *iloc* property of a Series object can be used to access a range of data values using the
index position numbers inside [] brackets in the following ways:

5. A single int can be passed to the iloc property. This will return back a single value.
6. A list of int representing index position numbers can be passed. This will return back aSeries object containing the multiple values
7. A slice notation using index position numbers can be passed. The data values

at the sliceposition numbers will the included in the returned Series object

8. A boolean array of the same length as the axis being sliced, e.g. [True, False, True].

2. Accessing the top elements of a Series object

The head() method can be used to return back the top elements of a Series object. This function returns back another Series object. If no parameter is passed to the head() method it returns back the top 5 elements. If an integer parameter (say n) is passed to the head() method, then the top n elements of theSeries object is returned back. The index of the respective elements is returned as it was in the original object.

```
#14 Accessing the top elements of a Series object
import pandas as pd

L=[101, 111, 121, 131, 141, 151, 161, 171, 181, 191, 201, 211]
s=pd.Series(L)
```

```
x=s.head()
print('x=\n', x)
y=s.head(3)
print('y=\n', y)

o/p:
x=
0    101
1    111
2    121
3    131
4    141
dtype: int64y=
0    101
1    111
2    121
dtype: int64
```

1. Accessing the bottom elements of a Series object

The tail() method can be used to return back the bottom elements of a Series object. This function returns back another Series object. If no parameter is passed to the tail() method it returns back the bottom 5 elements. If an integer parameter (say n) is passed to the tail() method, then the bottom n elements of theSeries object is returned back. The index of the respective elements is returned as it was in the original object.

```
#15 Accessing the bottom elements of a Series object
timport pandas as pd

L=[101, 111, 121, 131, 141, 151, 161, 171, 181, 191, 201, 211]
s=pd.Series(L)
x=s.tail()
```

```
print('x=\n', x)
y=s.tail(3)
print('y=\n', y)
```

```
o/p:
x=
7    171
8    181
9    191
10   201
11   211
dtype: int64
```

```
y=
9    191
10   201
11   211
dtype: int64
```

3. Indexing/Slicing a Series object-

The index [] operator can be used to perform indexing and slicing operations on a Series object. The index[]operator can accept either-

- **a)** Index/labels
- **b)** Integer index positions

a) Using the index operator with labels-

The index operator can be used in the following ways-

i) **Using a single label inside the square brackets-** Using a single label/index inside the square bracketswill return only the corresponding element referred to by that label/index.

```
# 16 indexing a Series object single label
import pandas as pd

d={'a':101, 'b':102, 'c':103, 'd':104, 'e':105, 'f':106}
s=pd.Series(d)
t=s['b']
print(t)

o/p:
102
```

Using multiple labels- We can pass multiple labels in any order that is present in the Series object. The multiple labels must be passed as a list i.e. the multiple labels must be separated by commas and enclosed in double square brackets. Passing a label is passed that is not present in the Series object, should be avoided as it right now gives NaN as the value but in future will be considered as an error byPython.

```
# 17 indexing a Series object
multiple labelsimport pandas as pd

d={'a':101, 'b':102, 'c':103, 'd':104, 'e':105, 'f':106}
s=pd.Serie
s(d)
```

```
u=s[['b', 'a',
'f']]
print(u)

o/p:
b        102
a        101
f        106
dtype: int64
```

ii) **Using slice notation startlabel:endlabel-** Inside the index operator we can pass startlabel:endlabel. Here contrary to the slice concept all the items from startlabel values till the endlabel values including the endlabel values is returned back.

```
# 18 indexing a Series object using startlabel:endlabel
import pandas as pd

d={'a':101, 'b':102, 'c':103, 'd':104, 'e':105, 'f':106}
s=pd.Series(d)
```

```
u=s['b': 'e']
print(u)

o/p:
b   102
c   103
d   104
e   105

dtype: int64
```

b) Slicing a Series object using Integer Index positions-

The concept of slicing a Series object is similar to that of slicing python lists, strings etc. Even though the datatype of the labels can be anything each element of the Series object is associated with two integer numbers:

- In forward indexing method the elements are numbered from 0,1,2,3, … with 0 being assigned to the first element, 1 being assigned to the second element and so on.
- In backward indexing method the elements are numbered from -1,-2, -3, … with -1 being assigned to the last element, -2 being assigned to the second last element and so on.

For example consider the following Series object-

```
d={'a':101, 'b':102, 'c':103, 'd':104, 'e':105, 'f':106}
s=pd.Series(d)
```

The Series object is having the following integer index positions-

forward indexing--->	0	1	2	3	4	5	
	a	b	c	d	e	f	
	101	111	121	131	141	151	
	-6	-5	-4	-3	-2	-1	<-----backward indexing

Slice concept-

The basic concept of slicing using integer index positions are common to Python object such as strings, list, tuples, Series, Dataframe etc. Slice creates a new object using elements of an existing object. It is created as: *ExistingObjectName[start : stop : step]* where start, stop , step are integers

The basic rules of slice:

i. The slice generates index/integers from : start, start + step, start + step + step, and so on. All the numbers generated must be less than the stop value when step is positive.
ii. If step value is missing then by default is taken to be 1
iii. If start value is missing and step is positive then start value is by default taken as 0.
iv. If stop value is missing and step is positive then start value is by default taken to mean till you reach the ending index(including the ending index)
v. A negative step value means the numbers are generated in backwards order i.e. from - start, then start - step, then start -step -step and so on. All the numbers generated in negative step must be greater than the stop value.
vi. If start value is missing and step is negative then start value takes default value -1
vii. If stop value is missing and step is negative then stop value is by default taken to be till you reach the first element(including the 0 index element)

```
#16 Slicing a Series object
import pandas as pd

d={'a':101, 'b':111, 'c':121, 'd':131, 'e':141, 'f':151}
s=pd.Series(d)

x=s[1: :2]
print('x=\n', x)

y=s[-1: :-1]
print('y=\n', y)

z=s[1: -2: 2]
print('z=\n', z)
```

```
o/p:
x=
b   111
d    131
f     151
dtype: int64
y=
f   151
e  141
d   131
c   121
b  111
a  101
dtype: int64
z=
b   111
d  131
dtype: int64
```

QUESTION AND ANSWER SECTION

1 What will be the output of following code-

```
import pandas as pd
s1=pd.Series([1,2,2,7,’Sachin’,77.5])
print(s1.head())
print(s1.head(3))
```

Ans:

```
0       1
1       2
2       2
3       7
4   Sachin
dtype: object

0   1
1   2
2   2
dtype: object
```

2 In pandas, S is a series with the following result:

```
S=pd.Series([5,10,15,20,25])
```

The series object is automatically indexed as 0,1,2,3,4. Write a statement to assign the series as a, b, c, d, e index explicitly.

Ans:

```
S=pd.Series([5,10,15,20,25],index=['a','b','c','d','e'])
```

3 Name any two attributes of Series in Python

Ans. Two attributes of Series in Python are :

1. index
2. values

4 Write the output of the following :

```
import numpy as num
import pandas as pd
arr=num.array([1,7,21])
S1 = pd.Series(arr)
print(S1)
```

Ans.

```
0    1
1    7
2   21
dtype: int32
```

5 Write the output of the following code :

```
import pandas as pd
S1 = pd.Series([31, 28, 31, 30, 31], index = ["Jan", "Feb", "Mar", "Apr", "May"])
print("-----------")
print(S1[1:3])
print("-----------")
print(S1[:5])
print("-----------")
print(S1[3:3])
print("-----------")
print(S1["Jan":"May"])
```

Ans.

```
-----------
Feb    28
Mar    31
dtype: int64
-----------
Jan    31
Feb    28
Mar    31
Apr   30
May    31
dtype: int64
-----------
Series([ ], dtype: int64)
-----------
Jan    31
Feb    28
Mar    31
Apr   30
May    31
dtype: int64
```

6 Differentiate between Pandas Series and NumPy Arrays

Ans. Differences are

Pandas Series	NumPy Arrays
In series we can define our own labeled index to access elements of an array.	In NumPy Arrays we can not define our own labelled index to access elements of an array
Series require more memory	NumPy occupies lesser memory.
The elements can be indexed in descending order also.	The indexing starts with zero for the firstelement and the index is fixed

7

1. What do you mean by Pandas in Python?
Ans. PANDAS (PANel DAta) is a high-level data manipulation tool used for analyzing data. Pandas library has a very rich set of functions.

1. Series
2. DataFrame
3. Panel

8

Name three data structures available in Pandas.

Ans. Three data structures available in Pandas are :

1. Series
2. DataFrame
3. Panel

9

Write the code in python to create an empty Series.
Ans.

```
import pandas as pd
S1 = pd.Series( )
print(S1)
```

OR

```
import pandas as pd
S1 = pd.Series( None)
print(S1)
```

```
OUTPUT : Series([ ], dtype: float64)
```

10

Define data structure in Python.
Ans. A data structure is a collection of data values and operations that can be applied to that data.

11

What do you mean by Series in Python?
Ans. A Series is a one-dimensional array containing a sequence of values of any data type (int, float, list, string, etc) which by default have numeric data labels (called index) starting from zero. Example of a series containing names of students is given below:

```
Index Value
0 Arnab
1 Samridhi
2 Ramit
3 Divyam
4 Kritika
```

12	Write command to install pandas in python. Ans. pip install pandas

13	Write the output of the following :
	`import pandas as pd`
	`S1 = pd.Series(range(100, 150, 10), index=[x for x in "My name is Amit Gandhi".split()])`
	`print(S1)`
	Ans.
	
	`My        100`
	`name      110`
	`is        120`
	`Amit      130`
	`Gandhi    140`
	`dtype: int64`

14	Write the output of the following :
	`import pandas as pd`
	`L1=[1,"A",21]`
	`S1 = pd.Series(data=2*L1)`
	`print(S1)`
	Ans.
	
	`0     1`
	`1     A`
	`2    21`
	`3     1`
	`4     A`
	`5    21`
	`dtype: object`

15	Which property of series return all the index value? Ans. index property return all the index value

16	Write the output of the following :
	`import pandas as pd`
	`S1 = pd.Series(range(1,15,3), index=[x for x in "super"])`
	`print(S1)`
	Ans.
	
	`s     1`
	`u     4`
	`p     7`
	`e    10`
	`r    13`
	`dtype: int64`

17 Complete the code to get the required output :

```
import _______ as pd
_________ = pd.Series([31, 28, 31], index = ["Jan", "Feb", "Mar"] )
print(S1["________"])
```

OUTPUT :

```
28
```

Ans.

```
import pandas as pd
S1 = pd.Series([31,28,31], index = ["Jan","Feb","Mar"])
print(S1["Feb"])
```

18 Fill in the blank of given code, if the output is 71.

```
import pandas as pd
S1 = pd.Series([10, 20, 30, 40, 71,50])
print(S1[ ___________ ])
```

Ans.

```
import pandas as pd
S1 = pd.Series([10, 20, 30, 40, 71,50])
print(S1[ 4 ])
```

19 Write a program to modify the value 5000 to 7000 in the following Series “S1”

```
A     25000
B     12000
C     8000
D     5000
```

Ans.

```
import pandas as pd
S1[3]=7000
print(S1)
```

20 Write a program to display only those values greater than 200 in the given Series “S1”

```
0     300
1     100
2    1200
3    1700
```

Ans.

```
import pandas as pd
S1 = pd.Series([300, 100, 1200, 1700])
print(S1[S1>200])
```