Introduction to Pandas:

It is the most important and commonly used library in datascience domain.

Pandas is free ware.

Pandas is open source library.

Pandas is built on top of Numpy.

It allows fast analysis, data cleaning and preparation.

Performance wise and productivity wise pandas is too good to use.

Pandas library also has inbuilt data visualization features.

It can work with data from a wide variety of sources like files etc.

By using pandas we can manipulate data very easily with very less code and in very less time.

Note:

1. Numpy is a Data Analysis Library.

2. Matplotlib is a Data Visualization Library.

3. Pandas is both Data Analysis and Data Visualization Library.

4. Pandas data analysis is based on Numpy where as Data Visualization is based on matplotlib.

official website: https://pandas.pydata.org/

Latest version: 1.3.2 (Aug 15, 2021)

From documentation:

"pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language."

How to install pandas:

There are 2 ways

1. With anaconda distribution, pandas library will be available automatically and we are not required to install separately. If it is not available then we can install as follows

conda install pandas

2. If python is already installed in our system, then we can install by using python package manager(pip),as follows

pip install pandas

```
D:\durgaclasses>pip install pandas
Collecting pandas
  Downloading pandas-1.3.2-cp38-cp38-win_amd64.whl (10.2 MB)
     |████████████████████████████████| 10.2 MB
130 kB/s
Requirement already satisfied: python-dateutil>=2.7.3 in
c:\python38\lib\site-packages (from pandas) (2.8.1)
Requirement already satisfied: numpy>=1.17.3 in
c:\python38\lib\site-packages (from pandas) (1.20.2)
Requirement already satisfied: pytz>=2017.3 in c:\python38\lib\site-
packages (from pandas) (2021.1)
Requirement already satisfied: six>=1.5 in
c:\users\lenovo\appdata\roaming\python\python38\site-packages
(from python-dateutil>=2.7.3->pandas) (1.15.0)
```

Installing collected packages: pandas
Successfully installed pandas-1.3.2

How to check installation:

1st way:

D:\durgaclasses>py
Python 3.8.6 (tags/v3.8.6:db45529, Sep 23 2020, 15:52:53) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import pandas as pd
>>> pd.__version__
'1.3.2'

2nd way:

D:\durgaclasses>pip list
Package Version
---------- ---------
.....
pandas 1.3.2
...

3rd way:

D:\durgaclasses>pip freeze
...

pandas==1.3.2

...

Important Topics:

Series

DataFrames

Missing Data

GroupBy

Merging,Joining and Concatenating

Operations

Data Input and Output

etc..

1. Series:

It is one of key data structures in pandas.

It is one-dimensional labeled arrays. ie a sequence of values associated with labels.

Creation of Series from the Python List:

----

eg-1:

```
import pandas as pd
books_list = ['Python','Java','DataScience']
s = pd.Series(books_list)
print('Type:',type(s)) # Type: <class 'pandas.core.series.Series'>
print(s)
```

o/p:

```
D:\durgaclasses>py test.py
Type: <class 'pandas.core.series.Series'>
0        Python
1          Java
2    DataScience
dtype: object
```

Note:

1. In the above Series object, we have 3 values (Python,Java,DataScience) associated with index labels (0,1,2), which are generated automatically by pandas.

2. For string values, dtype is considered as object.

3. The default index labels are integers starts from 0. But we can define any other type labels also.

4. The labels need not be unique.

eg-2:

```
import pandas as pd
marks_list = [70,80,90]
s = pd.Series(marks_list)
print(s)
```

o/p:

```
D:\durgaclasses>py test.py
0    70
1    80
```

2 90
dtype: int64

eg-3:

```
import pandas as pd
salaries_list = [1000.5,2000.6,3000.7]
s = pd.Series(salaries_list)
print(s)
```

D:\durgaclasses>py test.py
0 1000.5
1 2000.6
2 3000.7
dtype: float64

eg-4:

```
import pandas as pd
hetero_list = [10,'durga',10.5,True]
s = pd.Series(hetero_list)
print(s)
```

o/p:
D:\durgaclasses>py test.py
0 10
1 durga
2 10.5
3 True

dtype: object

Note: The values in Series can be any type even heterogeneous also.

Creation of Series from the Python Dict:

---

eg-1:

```
import pandas as pd
books_dict = {0:'Python',1:'Java',2:'DataScience'}
s = pd.Series(books_dict)
print(s)
```

o/p:

```
D:\durgaclasses>py test.py
0        Python
1          Java
2    DataScience
dtype: object
```

eg-2:

```
import pandas as pd
books_dict = {'Book-1':'Python','Book-2':'Java','Book-3':'DataScience'}
s = pd.Series(books_dict)
print(s)
```

o/p:

```
D:\durgaclasses>py test.py
```

Book-1 Python
Book-2 Java
Book-3 DataScience
dtype: object

eg-3:

```
import pandas as pd
books_dict = {'Book-1':'Python',10:20,10.5:20.6,'Book-
3':'DataScience'}
s = pd.Series(books_dict)
print(s)
```

o/p:

```
D:\durgaclasses>py test.py
Book-1 Python
10 20
10.5 20.6
Book-3 DataScience
dtype: object
```

Note:

1. Index labels and values need not be homogeneneous.
2. Index labels need not be unique.

From source code of pandas:

```
# Series class
```

```
class Series(base.IndexOpsMixin, generic.NDFrame):
    """
    One-dimensional ndarray with axis labels (including time series).

    Labels need not be unique but must be a hashable type. The object
    supports both integer- and label-based indexing and provides a host of
    methods for performing operations involving the index. Statistical
    methods from ndarray have been overridden to automatically exclude
    missing data (currently represented as NaN).

    Operations between Series (+, -, /, *, **) align values based on their
    associated index values-- they need not be the same length. The result
    index will be the sorted union of the two indexes.

    Parameters
    ----------
    data : array-like, Iterable, dict, or scalar value
        Contains data stored in Series. If data is a dict, argument order is
        maintained.
    index : array-like or Index (1d)
        Values must be hashable and have the same length as `data`.
        Non-unique index values are allowed. Will default to
        RangeIndex (0, 1, 2, ..., n) if not provided. If data is dict-like
```

and index is None, then the keys in the data are used as the index. If the

index is not None, the resulting Series is reindexed with the index values.

dtype : str, numpy.dtype, or ExtensionDtype, optional

Data type for the output Series. If not specified, this will be inferred from `data`.

See the :ref:`user guide <basics.dtypes>` for more usages.

name : str, optional

The name to give to the Series.

copy : bool, default False

Copy input data. Only affects Series or 1d ndarray input. See examples.

Examples

Constructing Series from a dictionary with an Index specified

```
>>> d = {'a': 1, 'b': 2, 'c': 3}
>>> ser = pd.Series(data=d, index=['a', 'b', 'c'])
>>> ser
a   1
b   2
c   3
dtype: int64
```

The keys of the dictionary match with the Index values, hence the Index

values have no effect.

```
>>> d = {'a': 1, 'b': 2, 'c': 3}
>>> ser = pd.Series(data=d, index=['x', 'y', 'z'])
>>> ser
x   NaN
y   NaN
z   NaN
dtype: float64
```

Note that the Index is first build with the keys from the dictionary. After this the Series is reindexed with the given Index values, hence we get all NaN as a result.

Constructing Series from a list with `copy=False`.

```
>>> r = [1, 2]
>>> ser = pd.Series(r, copy=False)
>>> ser.iloc[0] = 999
>>> r
[1, 2]
>>> ser
0    999
1      2
dtype: int64
```

Due to input data type the Series has a `copy` of

```
the original data even though `copy=False`, so
the data is unchanged.

Constructing Series from a 1d ndarray with `copy=False`.

>>> r = np.array([1, 2])
>>> ser = pd.Series(r, copy=False)
>>> ser.iloc[0] = 999
>>> r
array([999,   2])
>>> ser
0   999
1     2
dtype: int64

Due to input data type the Series has a `view` on
the original data, so
the data is changed as well.
"""
```

The 5 Parameters of Series Constructor:

---

1. data parameter
2. index parameter
3. dtype parameter
4. name parameter
5. copy parameter

1. data parameter:

----

data parameter can be used to represent data which is required to store inside Series object.

```
import pandas as pd
books_dict = {'Book-1':'Python',10:20,10.5:20.6,'Book-3':'DataScience'}
s = pd.Series(data=books_dict)
print(s)
```

o/p:

```
D:\durgaclasses>py test.py
Book-1        Python
10              20
10.5          20.6
Book-3    DataScience
dtype: object
```

Note:

The following are valid.

```
1. pd.Series(data=[10,20,30])
2. pd.Series(data={0:'A',1:'B',2:'C'})
3. pd.Series(data={'A':'Apple','B':'Ball','C':'Cat'})
4. pd.Series(data=np.array([10,20,30]))
5. pd.Series(data=10)
```

6. pd.Series(data='durga')

eg:

```
import pandas as pd
import numpy as np
s1 = pd.Series(data=np.array([10,20,30]))
print(s1)

s2 = pd.Series(data=10)
print(s2)

s3 = pd.Series(data='Durga')
print(s3)
```

o/p:

```
D:\durgaclasses>py test.py
0    10
1    20
2    30
dtype: int32
0    10
dtype: int64
0    Durga
dtype: object
```

2. index parameter:

We can use index parameter to define our own index values.

The values need not be unique. If we are not using index, then pandas will generate default index labels which are integers starts from 0. The number of index values should be same as the number of values of data parameter.

eg-1:

```
import pandas as pd
name_list = ['Sunny','Bunny','Chinny']
s = pd.Series(data=name_list,index=['S','B','C'])
print(s)
```

```
D:\durgaclasses>py test.py
S     Sunny
B     Bunny
C    Chinny
dtype: object
```

Note:

1. The number of data values and index values should be matched, otherwise error.

eg:

```
import pandas as pd
name_list = ['Sunny','Bunny','Chinny']
s = pd.Series(data=name_list,index=['S','B'])
print(s)
```

```
ValueError: Length of values (3) does not match length of index (2)
```

2. Duplicate index labels possible

eg:

```
import pandas as pd
name_list = ['Sunny','Bunny','Chinny','Binny']
s = pd.Series(data=name_list,index=['S','B','C','B'])
print(s)
```

o/p:

```
D:\durgaclasses>py test.py
S     Sunny
B     Bunny
C    Chinny
B     Binny
dtype: object
```

3. If data is dict, then matched indexes only will be considered from the dict.

eg:

```
import pandas as pd
name_dict = {'S':'Sunny','B':'Bunny','C':'Chinny','N':'Nikhil'}
s = pd.Series(data=name_dict,index=['S','B'])
print(s)
```

o/p:

```
D:\durgaclasses>py test.py
S    Sunny
```

B Bunny
dtype: object

eg:
```
import pandas as pd
name_dict = {'S':'Sunny','B':'Bunny','C':'Chinny','N':'Nikhil'}
s = pd.Series(data=name_dict,index=['S','B','A','B'])
print(s)
```

o/p:
```
D:\durgaclasses>py test.py
S    Sunny
B    Bunny
A      NaN
B    Bunny
dtype: object
```

eg: From pandas source code
```
>>> d = {'a': 1, 'b': 2, 'c': 3}
>>> ser = pd.Series(data=d, index=['x', 'y', 'z'])
>>> ser
x   NaN
y   NaN
z   NaN
dtype: float64
```

Note that the Index is first build with the keys from the dictionary.

After this the Series is reindexed with the given Index values, hence we
get all NaN as a result.

RangeIndex:

If we are not providing index parameter, then pandas will consider default index values from RangeIndex (0, 1, 2, ..., n) object internally.

eg:

```
import pandas as pd
name_list = ['Sunny','Bunny','Chinny','Binny']
s = pd.Series(data=name_list)
print(s.index) #RangeIndex(start=0, stop=4, step=1)
```

eg:

```
import pandas as pd
name_list = ['Sunny','Bunny','Chinny','Binny']
s = pd.Series(data=name_list)
s.index = pd.RangeIndex(start=10,stop=14,step=1)
print(s.index)
print(s)
```

o/p:

```
RangeIndex(start=10, stop=14, step=1)
10    Sunny
11    Bunny
12   Chinny
```

13 Binny
dtype: object

eg:

```
import pandas as pd
name_list = ['Sunny','Bunny','Chinny','Binny']
s = pd.Series(data=name_list)
s.index = pd.RangeIndex(start=10,stop=17,step=2)
print(s.index)
print(s)
```

o/p:

```
D:\durgaclasses>py test.py
RangeIndex(start=10, stop=17, step=2)
10    Sunny
12    Bunny
14   Chinny
16    Binny
dtype: object
```

3. dtype parameter:

We can use dtype parameter to specify data type for the output Series.

eg-1:

```
import pandas as pd
num_list = [10,20,30,40]
```

```
s = pd.Series(data=num_list,dtype='float')
print(s)
```

o/p:

```
D:\durgaclasses>py test.py
0    10
1    20
2    30
3    40
dtype: int64
```

eg-2:

```
import pandas as pd
num_list = [10,20,30,40,0]
s = pd.Series(data=num_list,dtype='bool')
print(s)
```

o/p:

```
D:\durgaclasses>py test.py
0     True
1     True
2     True
3     True
4    False
dtype: bool
```

eg:

```
import pandas as pd
```

```
num_list = [10,20,30,40,0]
s = pd.Series(data=num_list,dtype='str')
print(s)
```

o/p:

```
D:\durgaclasses>py test.py
0    10
1    20
2    30
3    40
4     0
dtype: object
```

4. name parameter:

We can assign name also to the Series. For this we have to use name parameter. The default name is None.

eg:

```
import pandas as pd
num_list = [10,20,30,40,0]
s = pd.Series(data=num_list)
print(s.name) #None
```

eg:

```
import pandas as pd
num_list = [10,20,30,40,0]
s = pd.Series(data=num_list,name='My Favourite Numbers')
```

#print(s.name) #My Favourite Numbers
print(s)

o/p:
D:\durgaclasses>py test.py
0 10
1 20
2 30
3 40
4 0
Name: My Favourite Numbers, dtype: int64

We can also set name as follows:
s.name='My Favourite Numbers'

eg:
import pandas as pd
num_list = [10,20,30,40,0]
s = pd.Series(data=num_list)
s.name='My Favourite Numbers'
print(s)

o/p:
D:\durgaclasses>py test.py
0 10
1 20
2 30
3 40

4 0
Name: My Favourite Numbers, dtype: int64

Setting Name to the indexes:

We can also set name to the indexes.
s.index.name = 'Default Indexes'

eg:
```
import pandas as pd
num_list = [10,20,30,40,0]
s = pd.Series(data=num_list)
s.name='My Favourite Numbers'
s.index.name = 'Default Indexes'
print(s)
```

o/p:
```
D:\durgaclasses>py test.py
Default Indexes
0  10
1  20
2  30
3  40
4   0
Name: My Favourite Numbers, dtype: int64
```

5. copy parameter:

This parameter decides whether it is required to create View or Copy.
The default value is False, ie new object won't be created.
It is applicable only for ndarray input.

eg-1: For list input

```
import pandas as pd
import numpy as np
num_list = [10,20]
s = pd.Series(data=num_list,copy=False)
s.iloc[0]=999
print(s)
print(num_list)
```

o/p:

```
D:\durgaclasses>py test.py
0    999
1     20
dtype: int64
[10, 20]
```

Even we changed Series content, that change not reflected to the list input.

eg-2: For ndarray input

```
import pandas as pd
import numpy as np
arr = np.array([10,20])
```

```
s = pd.Series(data=arr,copy=False)
s.iloc[0]=999
print(s)
print(arr)
```

o/p:

```
D:\durgaclasses>py test.py
0    999
1     20
dtype: int32
[999  20]
```

The changes of Series object reflected automatically inside ndarray input.

eg-3:

```
import pandas as pd
import numpy as np
arr = np.array([10,20,30,40])
s = pd.Series(data=arr,copy=True)
s[0]=9999
print(s)
print(arr)
```

Even input is ndarray, separate copy got created because copy=True.

Exercise:

---

1. Create a python list named with student_list with 5 student names?
2. Create another Python list named with marks_list with corresponding student marks?
3. Create a Series object that stores student marks as values and student names as index labels. Assign name 'students' for this series?
4. Create a python dictionary with student_list and marks_list and Create a Series object with that dictionary?

solution:

```
import pandas as pd
student_list = ['Sunny', 'Bunny','Vinny','Binny','Pinny']
marks_list = [40,50,60,70,80]
ser = pd.Series(data=marks_list , index=student_list, name='students')
print(ser)
```

o/p:

```
D:\durgaclasses>py test.py
Sunny    40
Bunny    50
Vinny    60
Binny    70
Pinny    80
Name: students, dtype: int64
```

solution:

```
import pandas as pd
```

```
student_list = ['Sunny', 'Bunny','Vinny','Binny','Pinny']
marks_list = [40,50,60,70,80]
#students_dict = dict(zip(student_list,marks_list))
students_dict = {name:marks for name,marks in
zip(student_list,marks_list)}
ser = pd.Series(data=students_dict,name='students')
print(ser)
```

o/p:

```
Sunny    40
Bunny    50
Vinny    60
Binny    70
Pinny    80
Name: students, dtype: int64
```

Accessing values from Series by using head() and tail() methods:

---

head():

Series.head(n=5)

Return the first n rows. The default value for n is 5

This function returns the first n rows for the object based on position. It is useful for quickly testing if your object has the right type of data in it.

For negative values of n, this function returns all rows except the last n rows.

eg:

```
import pandas as pd
s = pd.Series([i for i in range(50)])
print(s.head()) # Returns first 5 values
```

o/p:

```
D:\durgaclasses>py test.py
0  0
1  1
2  2
3  3
4  4
dtype: int64
```

eg-2:

```
import pandas as pd
s = pd.Series([i for i in range(50)])
print(s.head(3)) # Returns first 3 values
```

o/p:

```
D:\durgaclasses>py test.py
0  0
1  1
2  2
dtype: int64
```

eg-3:

```
----
import pandas as pd
s = pd.Series([i for i in range(50)])
print(s.head(n=-46)) # Returns the rows except last 46 rows. ie first 4 rows

D:\durgaclasses>py test.py
0  0
1  1
2  2
3  3
dtype: int64
```

tail():

```
-------
Series.tail(n=5)
  Return the last n rows. The default value is 5
```

This function returns last n rows from the object based on position. It is useful for quickly verifying data, for example, after sorting or appending rows.

For negative values of n, this function returns all rows except the first n rows, equivalent to df[n:].

eg:

```
import pandas as pd
s = pd.Series([i for i in range(50)])
#print(s.tail()) # Returns the last 5 rows
#print(s.tail(3)) # Returns the last 3 rows
print(s.tail(n=-3)) # Returns all rows except first 3 rows
```

Q. Returns values from 10th to 15th by using head() and tail() methods?

```
import pandas as pd
s = pd.Series([i for i in range(50)])
print(s.head(15).tail(6))
```

o/p:

```
D:\durgaclasses>py test.py
9     9
10    10
11    11
12    12
13    13
14    14
dtype: int64
```

Extract Values from Series by index position(Index Based Selection):

Syntax: s[x]

x can be index value

x can be list of indices

x can be slice also

eg:

s[5] --->Returns value present at index 5

s[[1,3,5]]---->Returns Series of values present at indices 1,3 and 5.

s[2:6]--->Returns Series fo values from 2nd index to 5th index

eg: Series contains all upper case alphabet symbols as values.

1st way:

```
import pandas as pd
alphabets=list('ABCDEFGHIJKLMNOPQRSTUVWXYZ') #['A','B',....]
s = pd.Series(data=alphabets)
print(s)
```

2nd way:

```
import pandas as pd
from string import ascii_uppercase
alphabets = list(ascii_uppercase)
s = pd.Series(alphabets)
print(s)
```

Q1. To get First Character?

```
s[0]
```

Q2. To get Last Character?

```
s[25]   or   s[s.size-1]
```

Q3. To get Character present at 10th index position?

```
s[10]
```

Q4. To get characters present at indices:5,10,15,20

```
s[[5,10,15,20]]
```

```
import pandas as pd
alphabets = list('ABCDEFGHIJKLMNOPQRSTUVWXYZ')
s = pd.Series(data=alphabets)
print(s[[5,10,15,20]])
```

```
D:\durgaclasses>py test.py
5    F
10   K
15   P
20   U
dtype: object
```

Q5. To get characters from 10th index to 16th index?

```
s[10:17]
```

```
import pandas as pd
```

```
alphabets = list('ABCDEFGHIJKLMNOPQRSTUVWXYZ')
s = pd.Series(data=alphabets)
print(s[10:17])
```

```
D:\durgaclasses>py test.py
10    K
11    L
12    M
13    N
14    O
15    P
16    Q
dtype: object
```

Q6. To get every other character ie every alternative character?

```
s[::2]
```

```
import pandas as pd
alphabets = list('ABCDEFGHIJKLMNOPQRSTUVWXYZ')
s = pd.Series(data=alphabets)
print(s[::2])
```

```
D:\durgaclasses>py test.py
0    A
2    C
4    E
6    G
```

8 I
10 K
12 M
14 O
16 Q
18 S
20 U
22 W
24 Y
dtype: object

Q7. To get first 5 characters?
s[0:5] or s[:5] or s.head() or s.head(5) or s.head(n=5)

Q8. To get last 3 characters?
s[-3:] or s[s.size-3:] or s.tail(3) or s.tail(n=3)

Note: -ve indexing is applicable only for slice input.
s[-1] ---->invalid
s[[-1,-2,-3]]--->invalid
s[-3:]--->valid

Note: Accessing based on position is applicable even for custom labeled Series also.

Extract Values from Series by labels(Label based Selection):

Syntax:

```
s['label']
s[['label-1','label-2','label-3']]
s[label1:label2]
```

sample code to append 'Label_' for every index:

```
alphabets = list('ABCDEFGHIJKLMNOPQRSTUVWXYZ')
print(list(map(lambda x:'Label_'+x,alphabets)))
print(['Label_'+x for x in alphabets])
```

pandas inbuilt add_prefix() and add_suffix() methods:

```
import pandas as pd
alphabets = list('ABCDEFGHIJKLMNOPQRSTUVWXYZ')
s = pd.Series(data=alphabets,index=alphabets)
#s = s.add_prefix('Label_')
s = s.add_suffix('_Label')
print(s)
```

Questions:

```
import pandas as pd
alphabets = list('ABCDEFGHIJKLMNOPQRSTUVWXYZ')
s = pd.Series(data=alphabets,index=alphabets)
s = s.add_prefix('Label_')
print(s)
```

Q1. To get first character?
s[0] or   s['Label_A']

Q2. To get 10th character?
s[9] or s['Label_J']

Q3. To get values associated with the labels:
'Label_K','Label_S','Label_X'
s[['Label_K','Label_S','Label_X']]

Q4. To get values from 'Label_H' to 'Label_S'?
s['Label_H':'Label_S']

```
import pandas as pd
alphabets = list('ABCDEFGHIJKLMNOPQRSTUVWXYZ')
s = pd.Series(data=alphabets,index=alphabets)
s = s.add_prefix('Label_')
print(s['Label_H':'Label_S'])

D:\durgaclasses>py test.py
Label_H    H
Label_I    I
Label_J    J
Label_K    K
Label_L    L
Label_M    M
Label_N    N
```

Label_O O
Label_P P
Label_Q Q
Label_R R
Label_S S
dtype: object

Note:
1. In the index based slicing, end/stop attribute is not inclusive
s[2:5] here 5 is not inclusive and it returns values from 2nd index to 4th index

2. But in Label based slicing, end/stop attribute is inclusive.
s['Label_H':'Label_S']--->'Label_S' is inclusive

Using dot notation to access data by labels:

Syntax:
s.label
Returns the value associated with specified label
eg:
print(s.Label_Z) #Z

Limitation: But this approach is not applicable for index and here we cannot use slice operator.

eg:
s.0 ---->invalid

s.Label1:Label5 --->Invalid

Extracting values by using get() method:

In position based selection or label based selection if the specified index or label is not available then we will get error.

```
import pandas as pd
alphabets = list('ABCDEFGHIJKLMNOPQRSTUVWXYZ')
s = pd.Series(data=alphabets,index=alphabets)
s = s.add_prefix('Label_')
print(s[100]) #IndexError: index 100 is out of bounds for axis 0 with size 26
print(s['Label_ZZ']) #KeyError: 'Label_ZZ'
```

To overcome this problem we should go for get() method. If the specified index or label is not available then we will get None but not Error.

Even in the case of get() method, we have the facility to provide default value if the specified index or lable is not available.

Syntax:

```
s.get(0)
s.get([0,3,5])
s.get('Label-1')
s.get(['Label-1','Label-2','Label-3'])
```

eg:

```
import pandas as pd
alphabets = list('ABCDEFGHIJKLMNOPQRSTUVWXYZ')
s = pd.Series(data=alphabets,index=alphabets)
s = s.add_prefix('Label_')
print(s.get(100)) #None
print(s.get('Label_ZZ')) #None
```

Eventhough index/label not available, we won't get any error.

eg:

```
import pandas as pd
alphabets = list('ABCDEFGHIJKLMNOPQRSTUVWXYZ')
s = pd.Series(data=alphabets,index=alphabets)
s = s.add_prefix('Label_')
print(s.get(100,default='default value')) #default value
print(s.get('Label_ZZ',default='default value')) #default value
```

Here default value will be considered because specified key and label are not available.

```
import pandas as pd
alphabets = list('ABCDEFGHIJKLMNOPQRSTUVWXYZ')
s = pd.Series(data=alphabets,index=alphabets)
s = s.add_prefix('Label_')
print(s.get(0)) #A
print(s.get([0,3,7]))
print(s.get('Label_D')) #D
print(s.get(['Label_D','Label_G','Label_M']))
```

o/p:

```
D:\durgaclasses>py test.py
A
Label_A    A
Label_D    D
Label_H    H
dtype: object
D
Label_D    D
Label_G    G
Label_M    M
dtype: object
```

Extracting values by using loc and iloc indexers:

----

loc indexer --->For Label based selection

iloc indexer--->For position based selection

iloc --->integer loc

iloc indexer:

For position based selection. The argument should be index.

Syntax:

s.iloc[i]

s.iloc[[0,1,2]]

s.iloc[m:n]

eg:

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

Q1. To get first value?

s[0] or s.iloc[0]

Q2. To get values at indies: 0,1,2

s.iloc[[0,1,2]]

Q3. To get first 3 values:

s.iloc[:3]

Q4. To get last value?

s[-1] --->invalid

s.iloc[-1] --->valid

```
import pandas as pd
s = pd.Series([10,20,30,40,50])
print(s.iloc[0])
print(s.iloc[[0,1,2]])
print(s.iloc[:3])
print(s.iloc[-1])
```

```
D:\durgaclasses>py test.py
10
0   10
1   20
2   30
dtype: int64
0   10
1   20
2   30
dtype: int64
50

loc indexer:
-----------
For label based selection

Syntax:
-------
s.loc[label]
s.loc[[label1,label2,label3]]
s.loc[labelm:labeln], here labeln is inclusive

eg:
import pandas as pd
alphabets = list('ABCDEFGHIJKLMNOPQRSTUVWXYZ')
s = pd.Series(data=alphabets,index=alphabets)
```

s = s.add_prefix('Label_')
print(s)

Q1. To get value associated with Label_A
s.loc['Label_A']

Q2. To get values associated with labels: Label_A,Label_K and Label_Y
s.loc[['Label_A','Label_K','Label_Y']]

Q3. To get values from Label_H to Label_N?
s['Label_H':'Label_N']
Here Label_N inclusive

eg:
```
import pandas as pd
alphabets = list('ABCDEFGHIJKLMNOPQRSTUVWXYZ')
s = pd.Series(data=alphabets,index=alphabets)
s = s.add_prefix('Label_')
print(s.loc['Label_A'])
print(s.loc[['Label_A','Label_K','Label_Y']])
print(s['Label_H':'Label_N'])
```

```
D:\durgaclasses>py test.py
A
Label_A    A
Label_K    K
Label_Y    Y
dtype: object
```

Label_H H
Label_I I
Label_J J
Label_K K
Label_L L
Label_M M
Label_N N
dtype: object

Q4. To get alternative values from Label_H to Label_N?
s.loc['Label_H':'Label_N':2]

D:\durgaclasses>py test.py
Label_H H
Label_J J
Label_L L
Label_N N
dtype: object

Q. Assume s is the Series object, which of the following are valid syntactically?

A. s[0]
B. s['Label_A']
C. s.iloc[0]
D. s.iloc['Label_A']
E. s.loc[0]
F. s.loc['Label_A']

Ans: A,B,C,F

Note:

1. The main advantage of loc and iloc indexers when compared normal indexer is performance will be improved.

2. iloc and loc indexers are commonly used in dataframes.

3. In normal indexer we cannot pass negative index value. But in iloc indexer we can pass.

s[-10] ---->invalid

s.iloc[-10] --->valid

4. If we are depending on default indexes then there is no difference between loc and iloc indexers.

eg:

```
import pandas as pd
s = pd.Series([10,20,30,40,50])
print(s[0])
print(s.iloc[0])
print(s.loc[0])
```

```
D:\durgaclasses>py test.py
10
10
```

10

Boolean masking For Condition Based Selection:

Condition based selection.

We have to provide array of boolean values and selects values from Series where True value present.

It is applicable for normal indexer,loc and iloc indexers also.

Syntax:

```
s[[True,False,...]]
s.loc[[True,False,...]]
s.iloc[[True,False,...]]
s.get([True,False,...])
```

eg:

```
import pandas as pd
s = pd.Series([10,20,30,40,50])
print(s[[True,False,False,True,True]])
print(s.loc[[True,False,False,True,True]])
print(s.iloc[[True,False,False,True,True]])
print(s.get([True,False,False,True,True]))
```

```
D:\durgaclasses>py test.py
0  10
3  40
4  50
```

dtype: int64
0 10
3 40
4 50
dtype: int64
0 10
3 40
4 50
dtype: int64
0 10
3 40
4 50
dtype: int64

*****Note: The number of boolean values passed and the number of values in Series must be matched, otherwise we will get error.**

```
import pandas as pd
s = pd.Series([10,20,30,40,50])
print(s[[True,False,False,True]])
```

```
D:\durgaclasses>py test.py
IndexError: Boolean index has wrong length: 4 instead of 5
```

But in the case of get() method we won't get any error and just we will get None.

eg:

```
import pandas as pd
s = pd.Series([10,20,30,40,50])
print(s.get([True,False,False,True])) #None
```

Note: This approach is very helpful to get values based on some condition.

eg-1: To select all values which are > 25.

```
import pandas as pd
s = pd.Series([10,20,30,40,50])
print(s[s>25])
```

```
D:\durgaclasses>py test.py
2    30
3    40
4    50
dtype: int64
```

eg-2:

```
import pandas as pd
s = pd.Series([i for i in range(20)])
print(s[s%3 == 0]) #To select values which are divisible by 3
```

```
D:\durgaclasses>py test.py
0     0
3     3
6     6
```

9 9
12 12
15 15
18 18
dtype: int64

Usage of Callables in Selecting Elements:

We can use Callable object like function while selecting values from the Series.
It should return anything,which should be valid argument for indexers and get method.

eg:

```
import pandas as pd
s = pd.Series([i for i in range(20)])

def odd_selection(s):
        return [True if i%2==1 else False for i in range(s.size)]

print(s[odd_selection])
```

D:\durgaclasses>py test.py
1 1
3 3
5 5
7 7

9 9
11 11
13 13
15 15
17 17
19 19
dtype: int64

We can pass Callable object to normal indexer, loc and iloc indexers and for get() method also.

```
import pandas as pd
s = pd.Series([i for i in range(20)])
print(s[lambda s: [True if i%2==1 else False for i in range(s.size)]])
print(s.loc[lambda s: [True if i%2==1 else False for i in range(s.size)]])
print(s.iloc[lambda s: [True if i%2==1 else False for i in range(s.size)]])
print(s.get(lambda s: [True if i%2==1 else False for i in range(s.size)]))
```

Q. To get every 3rd value:

```
s.get(lambda s: [True if i%3==0 else False for i in range(s.size)])
```

```
import pandas as pd
s = pd.Series([i for i in range(20)])
print(s.get(lambda s: [True if i%3==0 else False for i in range(s.size)]))
```

```
D:\durgaclasses>py test.py
0 0
```

```
3     3
6     6
9     9
12    12
15    15
18    18
dtype: int64
```

Q. Consider the Series:

```
Sunny     1000
Bunny     2000
Chinny    3000
Vinny     4500
Pinny     6000
dtype: int64
```

Select all values where salary is in between 2500 to 5000?

Solution:

```
import pandas as pd
s = pd.Series(
      data=[1000,2000,3000,4500,6000],
                index=['Sunny','Bunny','Chinny','Vinny','Pinny']
                )
#print(s[lambda s: [ True if sal>=2500 and sal<=5000 else False for sal
in s]])
```

print(s[lambda s: [True if sal in range(2500,5001) else False for sal in s]])

o/p:

```
D:\durgaclasses>py test.py
Chinny    3000
Vinny     4500
dtype: int64
```

Summary: How to get values from Series object:

----

1. s.head(n)
2. s.tail(n)
3. s[index]
4. s[[index1,index2,index3]]
5. s[indexm:indexn] here indexn is not inclusive
6. s[label]
7. s[[label1,label2,label3]]
8. s[[labelm:labeln]] here labeln is inclusive
9. s.iloc[index]
10. s.iloc[[index1,index2,index3]]
11. s.iloc[indexm:indexn] here indexn is not inclusive
12. s.loc[label]
13. s.loc[[label1,label2,label3]]
14. s.loc[[labelm:labeln]] here labeln is inclusive
15. s.get(index)
16. s.get([index1,index2,index3])
17. s.get(label)

18. s.get([label1,label2,label3])

Even we can provide boolean mask values and callable objects as arguments.

The important attributes of Series object:

Attributes are nothing but properties which provides information about the Series object.

The following are various important attributes of Series object.

1. s.values:

Returns values present inside Series object. Mostly it returns ndarray.

eg:

```
import pandas as pd
s = pd.Series(
     data=['Sunny','Bunny','Chinny','Vinny','Pinny','Nikhil'],
          index=[10,20,30,40,50,60]
          )
print(s.values)
```

```
D:\durgaclasses>py test.py
['Sunny' 'Bunny' 'Chinny' 'Vinny' 'Pinny' 'Nikhil']
```

2. s.index:

Rerurns the index (axis labels) of the Series.

```
import pandas as pd
s = pd.Series(
     data=['Sunny','Bunny','Chinny','Vinny','Pinny','Nikhil'],
          index=[10,20,30,40,50,60]
          )
print(s.index)
```

```
D:\durgaclasses>py test.py
Int64Index([10, 20, 30, 40, 50, 60], dtype='int64')
```

3. s.dtype:

Return the dtype object of the underlying data.

```
print(s.dtype)
```

```
D:\durgaclasses>py test.py
object
```

4. s.size:

Return the number of elements present in the Series object.

5. s.shape:

Return a tuple of the shape of the underlying data.

In the case of Series, it is single valued tuple, which represents the number of elements present in the Series object.

```
print(s.shape)
(6,)
```

6. s.ndim:

Returns number of dimensions of the underlying data, by definition 1.

```
print(s.ndim)
1
```

7. s.name:

Return the name of the Series. Default name is None.

8. s.is_unique:

Returns True if values in the object are unique.

eg:

```
import pandas as pd
s1 = pd.Series(['Sunny','Chinny','Vinny','Pinny','Nikhil'])
print(s1.is_unique) #True
```

```
s2 = pd.Series(['Sunny','Chinny','Vinny','Sunny','Pinny','Nikhil'])
print(s2.is_unique) #False
```

Note:
To get number of unique values, we can use nunique() method. Bydefault it ignores(drops) NaN values. If we want to consider NaN values also then we have to use dropna=False.
The default value for dropna is True.

```
Series.nunique(dropna=True)
```

eg:

```
import pandas as pd
import numpy as np
s1 =
pd.Series(['Sunny','Chinny','Vinny','Pinny','Nikhil','Sunny','Chinny',np.
NaN])
print(s1)
print('The Number of unique values with NaN ignore:',s1.nunique())
#5
print('The Number of unique values without NaN
ignore:',s1.nunique(dropna=False)) #6
```

9. s.is_monotonic, s.is_monotonic_increasing and s.is_monotonic_decreasing:

---

monotonic means whether values are in some order or not like ascending order or descending order etc.

s.is_monotonic--->returns true if values in the object are monotonic_increasing.
s.is_monotonic_increasing --->Alias for is_monotonic.
s.is_monotonic_decreasing--->Return True if values in the object are monotonic_decreasing.

eg:

```
import pandas as pd
import numpy as np
s1 = pd.Series([10,20,30,40])
s2 = pd.Series([40,30,20,10])
s3 = pd.Series([10,20,40,30])
print(s1.is_monotonic) #True
print(s2.is_monotonic) #False
print(s3.is_monotonic) #False

print(s1.is_monotonic_increasing) #True
print(s2.is_monotonic_increasing) #False
print(s3.is_monotonic_increasing) #False

print(s1.is_monotonic_decreasing) #False
print(s2.is_monotonic_decreasing) #True
print(s3.is_monotonic_decreasing) #False
```

Q. Which of the following attributes of Series object,are equal?

A. s.is_monotonic

B. s.is_monotonic_increasing
C. s.is_monotonic_decreasing
D. All of these

Ans: A and B

10. hasnans:

Returns True if Series contains NaN or None.
ie we can use this attribute to check whether some values are absent/missing or not.

eg:

```
import pandas as pd
s1 = pd.Series([10,20,30,40])
s2 = pd.Series([40,30,20,10,pd.NA])
s3 = pd.Series([10,20,40,30,None])

print(s1.hasnans) #False
print(s2.hasnans) #True
print(s3.hasnans) #True
```

Summary:
----
1. s.values
2. s.index
3. s.dtype

4. s.size
5. s.shape
6. s.ndim
7. s.name
8. s.is_unique / s.nunique()
9. s.is_monotonic/s.is_montonic_increasing/s.is_monotonic_decreasing
10. s.hashnans

Passing Series object to the Python's inbuilt functions:

We can pass pandas Series object to Python's inbuilt functions.

1. len(s):

It returns the number of elements present in the Series object.

2. type(s):

It returns the type of Series object. ie <class 'pandas.core.series.Series'>

3. dir(s):

It returns a list of all members(variables and methods) which are applicable for Series object.

4. sorted(s):

It will sort the elements present in the Series object and returns List of those values.

5. list(s):

To get Series object values in the form of List. It is Series to List conversion.

List contains only values but not index labels.

6. dict(s):

To convert Series object to dictionary.

dict keys -->Series Object index labels

dict values --->Series Object values

7. max(s):

Returns the maximum value present in the Series object

8. min(s):

Returns the minimum value present in the Series object.

eg:

import pandas as pd

```
s = pd.Series([10,40,30,20])
print(s)
print(len(s)) #4
print(type(s)) #<class 'pandas.core.series.Series'>
print(dir(s)) #['T', '_AXIS_LEN', '_AXIS_ORDERS',...]
print(sorted(s)) #[10, 20, 30, 40]
print(list(s)) #[10, 40, 30, 20]
print(dict(s)) #{0: 10, 1: 40, 2: 30, 3: 20}
print(max(s)) #40
print(min(s)) #10
```

Q. What is the main difference between pandas Series object and Python's dict object?

In the case of pandas Series object, duplicate index labels(keys) are possible.
But in the case of Python's dict, duplicate keys are not possible.

Whenever we are trying to convert Series object to Python's dict, if duplicate index labels are there, then with those duplicate index labels and values, a Series object will be created and assign that series object to the corresponding key in the dictionary. The advantage of this approach is, we are not missing any data in the conversion from Series object to dict.

eg:

```
import pandas as pd
s = pd.Series(
```

```
   data=['Sunny','Bunny','Chinny','Vinny','Pinny'],
        index=[10,20,30,40,10]
        )
print(s)
d = dict(s)
print(d)
print(type(d[10]))
```

```
D:\durgaclasses>py test.py
10     Sunny
20     Bunny
30    Chinny
40     Vinny
10     Pinny
dtype: object
{10: 10    Sunny
10    Pinny
dtype: object, 20: 'Bunny', 30: 'Chinny', 40: 'Vinny'}
<class 'pandas.core.series.Series'>
```

Note:

```
d = dict(s)
for k,v in d.items():
        print(f'{k} --->{type(v)}')
```

```
10 ---><class 'pandas.core.series.Series'>
```

20 ---><class 'str'>
30 ---><class 'str'>
40 ---><class 'str'>

Creation of Series object with the data from the csv file:

We know already the creation of Series object from list,dict,ndarray and scalar values.

We can create Series object with the data from multiple sources like csv file, excel file,json file, html file etc

Pandas library contains multiple functions for this like
- **pd.read_csv()**
- **pd.read_excel()**
- **pd.read_html()**
- **pd.read_json()**
- **etc**

We have to read_csv() function to create Series object with the data from the csv file.

Syntax:
pandas.read_csv(filepath_or_buffer, sep=<no_default>, delimiter=None, header='infer', names=<no_default>, index_col=None, usecols=None, squeeze=False, prefix=<no_default>, mangle_dupe_cols=True, dtype=None, engine=None, converters=None, true_values=None, false_values=None,

skipinitialspace=False, skiprows=None, skipfooter=0, nrows=None, na_values=None, keep_default_na=True, na_filter=True, verbose=False, skip_blank_lines=True, parse_dates=False, infer_datetime_format=False, keep_date_col=False, date_parser=None, dayfirst=False, cache_dates=True, iterator=False, chunksize=None, compression='infer', thousands=None, decimal='.', lineterminator=None, quotechar='"', quoting=0, doublequote=True, escapechar=None, comment=None, encoding=None, encoding_errors='strict', dialect=None, error_bad_lines=None, warn_bad_lines=None, on_bad_lines=None, delim_whitespace=False, low_memory=True, memory_map=False, float_precision=None, storage_options=None)

Read a comma-separated values (csv) file into DataFrame. ie this method returns DataFrame object bydefault but not Series object.

students.csv:

Name of Student	Marks
Sunny	100
Bunny	200
Chinny	300
Vinny	200
Pinny	400
Zinny	300
Kinny	500
Minny	600

```
Dinny      400
Ginny      700
Sachin     300
Dravid     900
Kohli 1000
Rahul      800
Ameer      600
Sharukh    500
Salman     700
Ranveer    600
Katrtina   300
Kareena    400
```

```
eg-1:
import pandas as pd
df = pd.read_csv('students.csv')
print('The Return Type:',type(df)) #<class
'pandas.core.frame.DataFrame'>
print(df)

D:\durgaclasses>py test.py
The Return Type: <class 'pandas.core.frame.DataFrame'>
  Name of Student  Marks
0          Sunny    100
1          Bunny    200
2         Chinny    300
3          Vinny    200
```

4 Pinny 400
5 Zinny 300
6 Kinny 500
7 Minny 600

Note:
If data contains only one column, then we can get Series object directly by passing squeeze=True.

eg:

```
import pandas as pd
s = pd.read_csv('students.csv',usecols=['Name of Student'],squeeze=True)
print(type(s))
print(s)
```

```
D:\durgaclasses>py test.py
<class 'pandas.core.series.Series'>
0    Sunny
1    Bunny
2   Chinny
3    Vinny
4    Pinny
5    Zinny
6    Kinny
7    Minny
8    Dinny
9    Ginny
```

```
10      Sachin
11      Dravid
12       Kohli
13       Rahul
14       Ameer
15     Sharukh
16      Salman
17     Ranveer
18    Katrtina
19     Kareena
Name: Name of Student, dtype: object
```

eg-2: Creation of Series object where name of the student as index label and marks as values by using students.csv file?

```
import pandas as pd
s = pd.read_csv(
   'students.csv',
        usecols=['Name of Student','Marks'],
        index_col='Name of Student',
        squeeze=True)
print(type(s))
print(s)
print('Name of Series:',s.name)
print('Name of Index of Series:',s.index.name)
```

```
D:\durgaclasses>py test.py
<class 'pandas.core.series.Series'>
Name of Student
Sunny       100
Bunny       200
Chinny      300
Vinny       200
Pinny       400
Zinny       300
Kinny       500
Minny       600
Dinny       400
Ginny       700
Sachin      300
Dravid      900
Kohli      1000
Rahul       800
Ameer       600
Sharukh     500
Salman      700
Ranveer     600
Katrtina    300
Kareena     400
Name: Marks, dtype: int64
Name of Series: Marks
Name of Index of Series: Name of Student
```

eg-3: Create a series object from any csv file which is available online?

Google Search: sample csv file from the net

https://www.stats.govt.nz/large-datasets/csv-files-for-download/

annual-enterprise-survey-2020-financial-year-provisional-csv
url: https://www.stats.govt.nz/assets/Uploads/Annual-enterprise-survey/Annual-enterprise-survey-2020-financial-year-provisional/Download-data/annual-enterprise-survey-2020-financial-year-provisional-csv.csv

eg:

```
import pandas as pd
s = pd.read_csv('https://www.stats.govt.nz/assets/Uploads/Annual-enterprise-survey/Annual-enterprise-survey-2020-financial-year-provisional/Download-data/annual-enterprise-survey-2020-financial-year-provisional-csv.csv',
        usecols=['Variable_name','Value'],
        index_col = 'Variable_name',
        squeeze = True)
print('Type of s:', type(s))
print('The Total number of values:',s.size)
print('The First 10 values:')
print('#'*30)
print(s.head(10))
```

D:\durgaclasses>py test.py

Type of s: <class 'pandas.core.series.Series'>
The Total number of values: 37080
The First 10 values:
################################
Variable_name
Total income 733,258
Sales, government funding, grants and subsidies 660,630
Interest, dividends and donations 54,342
Non-operating income 18,285
Total expenditure 654,872
Interest and donations 32,730
Indirect taxes 7,509
Depreciation 26,821
Salaries and wages paid 119,387
Redundancy and severance 305
Name: Value, dtype: object

The importance of count() method:

Series.count()
Returns the number of non-NA/null observations in the Series.

Note: In the csv file--->blank/null/NaN/nan is always treated as NaN.
But None is not treated as NaN

But from Python List,None is also treated as missing data.

students1.csv:

Name of Student Marks

Sunny 100

Bunny 200

Chinny 300

Vinny null

Pinny 400

Zinny 300

Kinny 500

Minny 600

Dinny NaN

Ginny 700

Sachin 300

Dravid

Kohli 1000

Rahul 800

Ameer 600

Sharukh 500

Salman 700

Ranveer nan

Katrtina 300

Kareena 400

test.py:

```
import pandas as pd
s = pd.read_csv('students1.csv',
        usecols=['Name of Student','Marks'],
```

```
        index_col = 'Name of Student',
        squeeze = True)
print(s)
print(s.size)
print(s.count())


D:\durgaclasses>py test.py
Name of Student
Sunny       100.0
Bunny       200.0
Chinny      300.0
Vinny        NaN
Pinny       400.0
Zinny       300.0
Kinny       500.0
Minny       600.0
Dinny        NaN
Ginny       700.0
Sachin      300.0
Dravid       NaN
Kohli      1000.0
Rahul       800.0
Ameer       600.0
Sharukh     500.0
Salman      700.0
Ranveer      NaN
Katrtina    300.0
```

Kareena 400.0
Name: Marks, dtype: float64
20
16

eg-2:

```
import pandas as pd
import numpy as np
s = pd.Series([10,20,30,None,pd.NA,np.nan])
print(s)
print('Size:',s.size)
print('Count:',s.count())
```

```
D:\durgaclasses>py test.py
0     10
1     20
2     30
3   None
4   <NA>
5    NaN
dtype: object
Size: 6
Count: 3
```

size attribute vs count() method:

---

size attribute returns the number of values including NAs and null values.

But count() method returns number of non-NA/null observations in the Series.

isnull()/ isna() method:

Series.isnull()

Detect missing values.

Return a boolean same-sized object indicating if the values are NA. NA values, such as None or numpy.NaN, gets mapped to True values. Everything else gets mapped to False values.

eg:

```
import pandas as pd
import numpy as np
s = pd.Series([10,20,30,None,pd.NA,np.nan])
s1= s.isnull()
print(s1)
```

```
D:\durgaclasses>py test.py
0    False
1    False
2    False
3     True
4     True
5     True
dtype: bool
```

To get only missing data:

```
------------------------
import pandas as pd
import numpy as np
s = pd.Series([10,20,30,None,pd.NA,np.nan])
s1=s[s.isnull()]
print(s1)

D:\durgaclasses>py test.py
3    None
4    <NA>
5     NaN
dtype: object
```

Note: s.isnull() returns boolean series object, which is used for boolean masking to select only values where NA is available.

eg2:

```
import pandas as pd
s = pd.read_csv('students1.csv',
        usecols=['Name of Student','Marks'],
        index_col = 'Name of Student',
        squeeze = True)
#To get data where values are missing
s1 = s[s.isnull()]
print(s1)

D:\durgaclasses>py test.py
```

Name of Student
Vinny NaN
Dinny NaN
Dravid NaN
Ranveer NaN
Name: Marks, dtype: float64

Note:s.isna() is just alias name for s.isnull(). Hence we can use these two methods interchangeably.

How to get the number of missing values:

There are multiple ways

1st way: s.size-s.count()
2nd way: s.isnull().sum()
3rd way: s[s.isnull()].size

```
import pandas as pd
s = pd.read_csv('students1.csv',
        usecols=['Name of Student','Marks'],
        index_col = 'Name of Student',
        squeeze = True)
#To get number of missing values
print(s.size-s.count()) #4
print(s.isnull().sum()) #4
```

Note: while performing sum() operation, False is treated as 0 and True is treated as 1.

s.notnull()/s.notna():

Series.notnull()

Detect existing (non-missing) values.

Return a boolean same-sized object indicating if the values are not NA. Non-missing values get mapped to True. Missing values are mapped to False.

Note: s.notna() is alias for s.notnull()

eg:

```
import pandas as pd
s = pd.read_csv('students1.csv',
        usecols=['Name of Student','Marks'],
        index_col = 'Name of Student',
        squeeze = True)
#To get non-missing values
print(s[s.notnull()])
print('The number of values:',s[s.notna()].size)
```

Questions:

Q1. How to check whether Series object contains NaN/null values or not?

By using hasnans attribute.

eg: print(s.hasnans) #True

Q2. How to get only missing values?

s[s.isnull()] or s.loc[s.isnull()]

s[s.isna()] or s.loc[s.isna()]

Q3. To get number of missing values?

1st way: s.size-s.count()

2nd way: s.isnull().sum()

3rd way: s[s.isnull()].size

Q4. How to get non-missing values?

s[s.notnull()] or s.loc[s.notnull()]

s[s.notna()] or s.loc[s.notna()]

Q5. How to get number of non-missing values?

1st way: s.notnull().sum() or s.notna().sum()

2nd way: s[s.notnull()].size or s[s.notna()].size

3rd way: s.count()

Q6. Which of the following expressions returns True?

A. s.size == s.isnull().sum() + s.notnull().sum()

B. s.count() == s.isnull().sum() + s.notnull().sum()

Ans: A

How to drop NAs?

Series class contains dropna() method for this.

Series.dropna()

Return a new Series with missing values removed.Because of this method there is no change in the existing Series object.

eg:

```
import pandas as pd
s = pd.read_csv('students1.csv',
     usecols=['Name of Student','Marks'],
          index_col='Name of Student',
          squeeze=True)
s1 = s.dropna()
print(s1)
```

```
D:\durgaclasses>py test.py
Name of Student
Sunny       100.0
Bunny       200.0
Chinny      300.0
Pinny       400.0
Zinny       300.0
Kinny       500.0
Minny       600.0
Ginny       700.0
Sachin      300.0
```

```
Kohli      1000.0
Rahul       800.0
Ameer       600.0
Sharukh     500.0
Salman      700.0
Katrtina    300.0
Kareena     400.0
Name: Marks, dtype: float64
```

In the above example, still s contains NAs because dropna() method returns a new Series object.

How to drop NAs in the existing object only:

We have to set inplace parameter with True value. The default value False.

In this case, dropna() method returns None.

```
import pandas as pd
s = pd.read_csv('students1.csv',
     usecols=['Name of Student','Marks'],
          index_col='Name of Student',
          squeeze=True)
s.dropna(inplace=True)
print(s)
```

D:\durgaclasses>py test.py

```
Name of Student
Sunny       100.0
Bunny       200.0
Chinny      300.0
Pinny       400.0
Zinny       300.0
Kinny       500.0
Minny       600.0
Ginny       700.0
Sachin      300.0
Kohli      1000.0
Rahul       800.0
Ameer       600.0
Sharukh     500.0
Salman      700.0
Katrtina    300.0
Kareena     400.0
Name: Marks, dtype: float64
```

How to replace NAs with our required value?

By using fillna(), we can replace NAs with our required value. This method returns a new Series object. In we want to perform modification in the existing object then we have to use inplace parameter.

eg:

```
import pandas as pd
s = pd.read_csv('students1.csv',
      usecols=['Name of Student','Marks'],
           index_col='Name of Student',
           squeeze=True)
s1 = s.fillna(0)
print(s1)

D:\durgaclasses>py test.py
Name of Student
Sunny       100.0
Bunny       200.0
Chinny      300.0
Vinny         0.0
Pinny       400.0
Zinny       300.0
Kinny       500.0
Minny       600.0
Dinny         0.0
Ginny       700.0
Sachin      300.0
Dravid        0.0
Kohli      1000.0
Rahul       800.0
Ameer       600.0
Sharukh     500.0
Salman      700.0
Ranveer       0.0
```

```
Katrtina     300.0
Kareena      400.0
Name: Marks, dtype: float64
```

To perform modification in the existing object only:

```
import pandas as pd
s = pd.read_csv('students1.csv',
      usecols=['Name of Student','Marks'],
           index_col='Name of Student',
           squeeze=True)
s.fillna(value=0,inplace=True)
print(s)
```

```
D:\durgaclasses>py test.py
Name of Student
Sunny        100.0
Bunny        200.0
Chinny       300.0
Vinny          0.0
Pinny        400.0
Zinny        300.0
Kinny        500.0
Minny        600.0
Dinny          0.0
Ginny        700.0
Sachin       300.0
Dravid         0.0
```

Kohli 1000.0
Rahul 800.0
Ameer 600.0
Sharukh 500.0
Salman 700.0
Ranveer 0.0
Katrtina 300.0
Kareena 400.0
Name: Marks, dtype: float64

Summary:
1. s.dropna() -->Returns a new series object by dropping NAs.
2. s.dropna(inplace=True)-->In the existing Series object, NAs will be dropped.
3. s.fillna(newvalue) -->Returns a new series object by replacing NAs with our provided value.
4. s.fillna(value=newvalue,inplace=True) -->In the existing Series object, NAs will be replaced with our provided value.

Basic Statistics for Series object:
----
1. s.sum():

Returns the sum of values, present inside Series object.
This method ignores NAs automatically.

2. s.mean():

Returns mean value of the series.
Mean means average.

```
print('The Mean Value:',s.mean()) #481.25
print('The Mean Value:',s.sum()/s.count()) #481.25
```

3. s.median()

It returns middle element in the sorted list of values.

Number of values odd: returns middle value
1,2,3,4,5,6,7 --->median is:4

Number of values even: returns mean of middle 2 values.
1,2,3,4,5,6,7,8 --->median is:4.5 (mean of 4 and 5)

```
eg: print('The Median Value:',s.median()) #450.0
```

4. s.var()

It returns the variance of values of the Series object.

5. s.std():

It returns the standard deviation of values of Series object.
It is the square root of variance.

```
print('The Variance Value:',s.var()) #57625.0
```

print('The Stadard Deviation:',s.std()) #240.05207768315609
print(int(s.std()2) == int(s.var())) #True**

6. s.mode():

Returns the most repeated value. ie the most frequently occurred value.

eg: print('The Mode:',s.mode())

Q. How to find the number of times value repeated?
s[s==s.mode()[0]].size

print('The Mode:',s.mode()) #returns series object with only one value 300.0
print('The number of times mode value repeated:',s[s==300].size)
print('The number of times mode value repeated:',s[s==s.mode()[0]].size)

Demo program:

```
import pandas as pd
s = pd.read_csv('students1.csv',
      usecols=['Name of Student','Marks'],
          index_col='Name of Student',
          squeeze=True)
print('The Sum of all values:',s.sum()) #7700.0
```

```
print('The Mean Value:',s.mean()) #481.25
print('The Mean Value:',s.sum()/s.count()) #481.25
print('The Median Value:',s.median()) #450.0
print('The Variance Value:',s.var()) #57625.0
print('The Stadard Deviation:',s.std()) #240.05207768315609
print(int(s.std()**2) == int(s.var())) #True
print('The Mode:',s.mode()) #returns series object with only one value
300.0
print('The number of times mode value repeated:',s[s==300].size)
print('The number of times mode value
repeated:',s[s==s.mode()[0]].size)
```

7. s.value_counts():

Series.value_counts(normalize=False, sort=True, ascending=False, bins=None, dropna=True)

Return a Series containing counts of unique values.

The resulting object will be in descending order so that the first element is the most frequently-occurring element. Excludes NA values by default.

eg:

```
import pandas as pd
s = pd.read_csv('students1.csv',
      usecols=['Name of Student','Marks'],
          index_col='Name of Student',
```

```
        squeeze=True)
s1 = s.value_counts()
print(s1)

D:\durgaclasses>py test.py
300.0    4
400.0    2
500.0    2
600.0    2
700.0    2
100.0    1
200.0    1
1000.0   1
800.0    1
Name: Marks, dtype: int64
```

Note: If we use normalize=True then we will get frequency in percentages

eg:

```
import pandas as pd
s = pd.read_csv('students1.csv',
     usecols=['Name of Student','Marks'],
         index_col='Name of Student',
         squeeze=True)
s1 = s.value_counts(normalize=True)
print(s1)

D:\durgaclasses>py test.py
```

300.0 0.2500
400.0 0.1250
500.0 0.1250
600.0 0.1250
700.0 0.1250
100.0 0.0625
200.0 0.0625
1000.0 0.0625
800.0 0.0625
Name: Marks, dtype: float64

8. s.min()

Returns minimum value present in the Series.

9. s.max()

Returns maximum value present in the Series.

eg:

```
import pandas as pd
s = pd.read_csv('students1.csv',
     usecols=['Name of Student','Marks'],
          index_col='Name of Student',
          squeeze=True)
print('The Minimum Value:',s.min())
print('The Maximum Value:',s.max())
```

The importance of describe() method:

It generates descriptive statistics like count,mean,std,min,max etc. Before analyzing our data, it is recommended to use this method to get descriptive statistics about our Series object.

eg:

```
import pandas as pd
s = pd.read_csv('students1.csv',
      usecols=['Name of Student','Marks'],
          index_col='Name of Student',
          squeeze=True)
print(s.describe())
```

```
D:\durgaclasses>py test.py
count     16.000000
mean     481.250000
std      240.052078
min      100.000000
25%      300.000000
50%      450.000000
75%      625.000000
max     1000.000000
Name: Marks, dtype: float64
```

Note:

min 100.000000

25% of values are less than or equal to 300.

50% of values are less than or equal to 450. It is the median value.
75% of values are less than or equal to 625.
max 1000.000000

Exercise:
----
1. Separate non-nulls from the student series, which is generated from students1.csv file, and assign this series to existing_marks variable?

2. Find the sum of all student marks ?

3. Find the students whose marks are >= 500?

4. Find the sum of all student marks which are >=500?

5. How many students got marks less than 350?

6. How many students got marks >=400 ?

7. Find highest marks in the Series?

8. Find least marks in the Series?
Solution:
----
1. Separate non-nulls from the student series, which is generated from students1.csv file, and assign this series to existing_marks variable?

```
import pandas as pd
students = pd.read_csv('students1.csv',
      usecols=['Name of Student','Marks'],
           index_col='Name of Student',
           squeeze=True)
#print(students)
existing_marks = students[students.notnull()]
print(existing_marks)
```

2. Find the sum of all student marks ?

```
print('The Sum of all student marks:',existing_marks.sum())
```

3. Find the students whose marks are >= 500?

```
print(existing_marks[ existing_marks >= 500])
```

4. Find the sum of all student marks which are >=500?

```
print(existing_marks[ existing_marks >= 500].sum())
```

5. How many students got marks less than 350?

```
print('The number of students whose marks < 350:',existing_marks[
existing_marks < 350].size)
```

6. How many students got marks >=400 ?

```
print('The number of students whose marks >= 400:',existing_marks[
existing_marks >= 400].size)
```

7. Find highest marks in the Series?

```
print('Highest Marks:',existing_marks.max())
```

8. Find least marks in the Series?

```
print('Least Marks:',existing_marks.min())
```

Demo program:

```
import pandas as pd
students = pd.read_csv('students1.csv',
      usecols=['Name of Student','Marks'],
           index_col='Name of Student',
           squeeze=True)
#print(students)
existing_marks = students[students.notnull()]
print(existing_marks)
print('The Sum of all student marks:',existing_marks.sum())
print(existing_marks[ existing_marks >= 500])
print(existing_marks[ existing_marks >= 500].sum())
print('The number of students whose marks < 350:',existing_marks[
existing_marks < 350].size)
print('The number of students whose marks >= 400:',existing_marks[
existing_marks >= 400].size)
print('Highest Marks:',existing_marks.max())
```

```
print('Least Marks:',existing_marks.min())
```

o/p:

```
D:\durgaclasses>py test.py
Name of Student
Sunny       100.0
Bunny       200.0
Chinny      300.0
Pinny       400.0
Zinny       300.0
Kinny       500.0
Minny       600.0
Ginny       700.0
Sachin      300.0
Kohli      1000.0
Rahul       800.0
Ameer       600.0
Sharukh     500.0
Salman      700.0
Katrtina    300.0
Kareena     400.0
Name: Marks, dtype: float64
The Sum of all student marks: 7700.0
Name of Student
Kinny       500.0
Minny       600.0
Ginny       700.0
```

```
Kohli     1000.0
Rahul      800.0
Ameer      600.0
Sharukh    500.0
Salman     700.0
Name: Marks, dtype: float64
5400.0
The number of students whose marks < 350: 6
The number of students whose marks >= 400: 10
Highest Marks: 1000.0
Least Marks: 100.0

Finding index labels associated with maxvalue and minvalue:
------------------------------------------------------------
without using readymade methods:
-------------------------------
s.max() -->Returns max value.
s[s==s.max()] --->Returns Series object, where max value available
s[s==s.max()].index --->Returns Index object
s[s==s.max()].index[0] --->Returns index label which is associated with
max value.


import pandas as pd
s = pd.read_csv('students1.csv',
     usecols=['Name of Student','Marks'],
          index_col='Name of Student',
          squeeze=True)
```

```
print(s.max())
print(s[s == s.max()])
print(s[s == s.max()].index)
print(s[s == s.max()].index[0])

D:\durgaclasses>py test.py
1000.0
Name of Student
Kohli    1000.0
Name: Marks, dtype: float64
Index(['Kohli'], dtype='object', name='Name of Student')
Kohli
```

We can do the same thing directly by using ready made methods: idxmax() and idxmin()

s.idxmax()--->Returns index label associated with max value.
s.idxmin()--->Returns index label associated with min value.
s.max() --->Returns only max value but not index label
s.min() --->Returns only min value but not index label

Note: If multiple max/min values, then idxmax() and idxmin() returns only first matched index label.

demo program:

```
------------
```

```
import pandas as pd
s = pd.read_csv('students1.csv',
      usecols=['Name of Student','Marks'],
           index_col='Name of Student',
           squeeze=True)
print('The Name of Student who got highest marks:',s.idxmax())
print('The Name of Student who got least marks:',s.idxmin())
```

o/p:

```
D:\durgaclasses>py test.py
The Name of Student who got highest marks: Kohli
The Name of Student who got least marks: Sunny
```

Finding first n largest and n smallest values: nlargest() and nsmallest():

```
s.nlargest(n=5) -->Returns the largest n elements.
s.nsmallest(n=5) -->Returns the smallest n elements.
```

```
import pandas as pd
s = pd.read_csv('students1.csv',
      usecols=['Name of Student','Marks'],
           index_col='Name of Student',
           squeeze=True)
print(s.nlargest(n=3))
print(s.nsmallest(n=3))
```

```
D:\durgaclasses>py test.py
```

Name of Student
Kohli 1000.0
Rahul 800.0
Ginny 700.0
Name: Marks, dtype: float64
Name of Student
Sunny 100.0
Bunny 200.0
Chinny 300.0
Name: Marks, dtype: float64

eg-2:

```
import pandas as pd
s=['alluarjun','maheshbabu','ramcharan','pawankalyan','nani','ntr','ra
viteja','prabash']
s1=pd.Series(index=s,data=[15,30,15,50,10,18,10,25],name='top 5
heros remuniration')
print(s1.nlargest(n=5))
```

```
D:\durgaclasses>py test.py
pawankalyan 50
maheshbabu 30
prabash 25
ntr 18
alluarjun 15
Name: top 5 heros remuniration, dtype: int64
```

Sorting of values by using sort_values() method:

This method is helpful to sort only based on values but not based on index labels.

Syntax:

Series.sort_values(ascending=True, inplace=False, kind='quicksort', na_position='last')

parameters:

----

1. ascending: If True, sort values in ascending order, otherwise descending. Default is True which is meant for ascending order.

2. inplace: default False

If True, perform operation in-place.

If we are not passing this parameter then sort_values() method returns a new Series object.If we want to sort in the existing object only then we have to set inplace=True.

3. kind: Choice of sorting algorithm.

{‘quicksort’, ‘mergesort’, ‘heapsort’, ‘stable’}, default ‘quicksort’

Based on our requirement,we can choose sorting algorithms.

4. na_position: {‘first’ or ‘last’}, default ‘last’

Argument ‘first’ puts NaNs at the beginning, ‘last’ puts NaNs at the end.

demo program:

```
import pandas as pd
s = pd.read_csv('students1.csv',
      usecols=['Name of Student','Marks'],
           index_col='Name of Student',
           squeeze=True)
s.sort_values(inplace=True,na_position='first',ascending=False)
print(s)
```

Sorting based on index labels by using sort_index() method:

Exactly same as sort_values() method including parameters except that sorting is based on index labels but not based on values.

demo program:

```
import pandas as pd
s = pd.read_csv('students1.csv',
      usecols=['Name of Student','Marks'],
           index_col='Name of Student',
           squeeze=True)
s.sort_index(inplace=True,na_position='first',ascending=True)
print(s)
```

o/p:

```
D:\durgaclasses>py test.py
Name of Student
Ameer       600.0
```

```
Bunny       200.0
Chinny      300.0
Dinny        NaN
Dravid       NaN
Ginny       700.0
Kareena     400.0
Katrtina    300.0
Kinny       500.0
Kohli      1000.0
Minny       600.0
Pinny       400.0
Rahul       800.0
Ranveer      NaN
Sachin      300.0
Salman      700.0
Sharukh     500.0
Sunny       100.0
Vinny        NaN
Zinny       300.0
Name: Marks, dtype: float64
```

Note:

sort_values() : returns a new Series object sorted based on values.

sort_index() : returns a new Series object sorted based on index labels.

Default prameters:

ascending = True

inplace = False
na_position = 'last'
kind = 'quicksort'

Basic Arithmetic Operations for Series objects:

----

1. Arithmetic Operations with Scalar value:

----

scalar means constant value.

We can perform arithmetic operations between Series object and scalar value.

Operation will be performed for every element.

eg:

```
import pandas as pd
s = pd.Series([10,20,30,40,50])
print(s)
print(s+10)
print(s-3)
print(s*3)
print(s/2)
```

```
D:\durgaclasses>py test.py
0  10
1  20
2  30
3  40
4  50
```

dtype: int64

0 20

1 30

2 40

3 50

4 60

dtype: int64

0 7

1 17

2 27

3 37

4 47

dtype: int64

0 30

1 60

2 90

3 120

4 150

dtype: int64

0 5.0

1 10.0

2 15.0

3 20.0

4 25.0

dtype: float64

Note: If the value is NA, then after performing scalar operations the result is always NA only.

Reason: If we perform any operation on NA then result is always NA.

eg:

```
import pandas as pd
import numpy as np
s = pd.Series([10,pd.NA,np.NaN,None])
print(s)
print(s+10)
```

```
D:\durgaclasses>py test.py
0     10
1    <NA>
2     NaN
3    None
dtype: object
0     20
1    NaN
2    NaN
3    NaN
dtype: object
```

2. Arithmetic operations between 2 Series objects:

We can perform arithmetic operations between two Series objects.

These operations will be performed only on matched indexes.
For unmatched indexes, NaN will be returned.

eg-1:

```
import pandas as pd
import numpy as np
s1 = pd.Series([10,20,30]) #index:0,1,2
s2 = pd.Series([10,20,30]) #index:0,1,2
print(s1+s2)
```

```
D:\durgaclasses>py test.py
0    20
1    40
2    60
dtype: int64
```

eg-2:

```
import pandas as pd
import numpy as np
s1 = pd.Series(data=[10,20,30,40,50], index=['A','B','C','D','E'])
s2 = pd.Series(data=[10,20,30,40,50], index=['C','D','E','F','G'])
print(s1+s2)
```

```
D:\durgaclasses>py test.py
A     NaN
B     NaN
C    40.0
D    60.0
```

E 80.0
F NaN
G NaN
dtype: float64

Note: Series class contains equivalent methods for arithmetic operations.
s1+s2 ----->s1.add(s2)
s1-s2 ----->s1.sub(s2)
s1*s2 ----->s1.mul(s2)
s1/s2 ----->s1.div(s2)

eg:

```
import pandas as pd
s1 = pd.Series(data=[10,20,30,40,50], index=['A','B','C','D','E'])
s2 = pd.Series(data=[10,20,30,40,50], index=['C','D','E','F','G'])
print(s1.add(s2))
```

```
D:\durgaclasses>py test.py
A    NaN
B    NaN
C   40.0
D   60.0
E   80.0
F    NaN
G    NaN
dtype: float64
```

fill_value parameter:

We can pass fill_value parameter for add(),sub(),mul() and div() methods.

If the matched index is not available, then fill_value will be considered in the place of missing element.

fill_value parameter is the advantage of add()/sub()/mul()/div() methods when compared with +,-,*,/ operators.

eg-1:

```
import pandas as pd
import numpy as np
s1 = pd.Series(data=[10,20,30,40,50], index=['A','B','C','D','E'])
s2 = pd.Series(data=[10,20,30,40,50], index=['C','D','E','F','G'])
print(s1.add(s2,fill_value=0))
```

```
D:\durgaclasses>py test.py
A    10.0
B    20.0
C    40.0
D    60.0
E    80.0
F    40.0
G    50.0
dtype: float64
```

eg-3:

```
import pandas as pd
```

```
import numpy as np
s1 = pd.Series(data=[10,np.NaN], index=['A','Z'])
s2 = pd.Series(data=[10,20], index=['A','B'])
print(s1.add(s2,fill_value=0))

D:\durgaclasses>py test.py
A    20.0
B    20.0
Z     NaN
dtype: float64
```

```
eg-4:
import pandas as pd
import numpy as np
s1 = pd.Series(data=[10,20,30,40,50], index=['A','B','C','D','E'])
s2 = pd.Series(data=[10,20,30,40,50], index=['C','D','E','F','G'])
print(s1.sub(s2))
print(s1.sub(s2,fill_value=0))
print(s1.mul(s2))
print(s1.mul(s2,fill_value=0))
print(s1.div(s2))
print(s1.div(s2,fill_value=0))

D:\durgaclasses>py test.py
A     NaN
B     NaN
C    20.0
```

D 20.0
E 20.0
F NaN
G NaN
dtype: float64
A 10.0
B 20.0
C 20.0
D 20.0
E 20.0
F -40.0
G -50.0
dtype: float64
A NaN
B NaN
C 300.0
D 800.0
E 1500.0
F NaN
G NaN
dtype: float64
A 0.0
B 0.0
C 300.0
D 800.0
E 1500.0
F 0.0
G 0.0

dtype: float64
A NaN
B NaN
C 3.000000
D 2.000000
E 1.666667
F NaN
G NaN
dtype: float64
A inf
B inf
C 3.000000
D 2.000000
E 1.666667
F 0.000000
G 0.000000
dtype: float64

Cumulative Operations/Progressive Operations:

There are multiple cumulative operations applicable for Series object.

1.sum()--->To find the sum of all values.
2.cumsum()--->Returns a Series of the same size containing the cumulative sum.

eg:
import pandas as pd

```
s = pd.Series(data=[10,20,30,40,50])
print(s.sum())
print(s.cumsum())

D:\durgaclasses>py test.py
150
0     10
1     30
2     60
3    100
4    150
dtype: int64
```

Note: Bydefault cumsum() method ignores NAs while performing cumulative sum operation. By using skipna parameter we can customize this behaviour. The default value is True.

eg-1:

```
import pandas as pd
s = pd.Series(data=[pd.NA,10,20,30,40,50])
print(s.sum())
print(s.cumsum())

D:\durgaclasses>py test.py
150
0     NaN
1    10.0
2    30.0
```

3 60.0
4 100.0
5 150.0
dtype: object

eg-2:

```
import pandas as pd
s = pd.Series(data=[pd.NA,10,20,30,40,50])
print(s.sum())
print(s.cumsum(skipna=False))
```

```
D:\durgaclasses>py test.py
150
0   <NA>
1   <NA>
2   <NA>
3   <NA>
4   <NA>
5   <NA>
dtype: object
```

s.prod() and s.cumprod():

---

s.prod()-->Returns the product of all values.

s.cumprod()-->Returns the cumulative product of values.

eg:

```
import pandas as pd
s = pd.Series(data=[1,2,3,4,5])
print(s.prod())
print(s.cumprod())
```

```
D:\durgaclasses>py test.py
120
0     1
1     2
2     6
3    24
4   120
dtype: int64
```

min() and cummin():

---

s.min() --->Returns the minimum value.

s.cummin() --->Returns the cumulative minimum value including current vlaue.

eg:

```
import pandas as pd
s = pd.Series(data=[1,2,3,4,5])
print(s.min())
print(s.cummin())
```

```
D:\durgaclasses>py test.py
```

1
0 1
1 1
2 1
3 1
4 1
dtype: int64

max() and cummax()
----

s.max() --->Returns the maximum value.
s.cummax() --->Returns the cumulative maximum value including current vlaue.

eg:
```
import pandas as pd
s = pd.Series(data=[1,2,3,4,5])
print(s.max())
print(s.cummax())
```

D:\durgaclasses>py test.py
5
0 1
1 2
2 3
3 4
4 5

dtype: int64

Summary:

----

1. cumsum()

2. cumprod()

3. cummin()

4. cummax()

Finding Discrete Difference by using diff() method:

Series.diff(periods=1)

First discrete difference of element.

Calculates the difference of a Series element compared with another element in the Series (default is element in previous row).

eg:

```
import pandas as pd
s = pd.Series([10,20,30,40,50])
print(s.diff()) # s.diff(periods=1)
```

```
D:\durgaclasses>py test.py
0    NaN
1   10.0
2   10.0
3   10.0
4   10.0
```

dtype: float64

Note:
i1--->v1
i2--->v2
i3--->v3
i4--->v4
i5--->v5

periods=1:
----
i1--->v1-NaN
i2--->v2-v1
i3--->v3-v2
i4--->v4-v3
i5--->v5-v4

periods=2: (difference with 2nd previous element)
----
i1--->v1-NaN
i2--->v2-NaN
i3--->v3-v1
i4--->v4-v2
i5--->v5-v3

```
import pandas as pd
s = pd.Series([10,20,30,40,50])
print(s.diff(periods=2))
```

D:\durgaclasses>py test.py

```
0    NaN
1    NaN
2   20.0
3   20.0
4   20.0
dtype: float64
```

periods=-1: (difference with next element)

----

i1--->v1-v2

i2--->v2-v3

i3--->v3-v4

i4--->v4-v5

i5--->v5-NaN

```
import pandas as pd
s = pd.Series([10,20,30,40,50])
print(s.diff(periods=-1))
```

D:\durgaclasses>py test.py

```
0  -10.0
1  -10.0
2  -10.0
3  -10.0
4    NaN
```

dtype: float64

Note: This diff() method is very helpful while working with Time Series in DataScience.

Filtering elements of Series based on values:

By using boolean masking or callable functions, we can filter elements.

eg: To filter all students whose marks are less than 300.

```
import pandas as pd
s = pd.read_csv('students1.csv',
     usecols=['Name of Student','Marks'],
          index_col='Name of Student',
          squeeze=True)
print(s[s<300]) #Boolean masking
print(s.loc[s<300]) #Boolean masking

def lt_300(x):
        return x <300

print(s[lt_300]) #passing callable object
```

o/p:

```
D:\durgaclasses>py test.py
Name of Student
Sunny    100.0
```

Bunny 200.0
Name: Marks, dtype: float64

D:\durgaclasses>py test.py
Name of Student
Sunny 100.0
Bunny 200.0
Name: Marks, dtype: float64
Name of Student
Sunny 100.0
Bunny 200.0
Name: Marks, dtype: float64

D:\durgaclasses>py test.py
Name of Student
Sunny 100.0
Bunny 200.0
Name: Marks, dtype: float64
Name of Student
Sunny 100.0
Bunny 200.0
Name: Marks, dtype: float64
Name of Student
Sunny 100.0
Bunny 200.0
Name: Marks, dtype: float64

Note:

In the above example filtering happend based values but not based on index labels.
If we want to filter elements based on index labels then we should go for filter() method.

Filtering elements of Series based on index labels by using filter() method:

```
Series.filter(items=None, like=None, regex=None, axis=None)
```

Subset the Series rows according to the specified index labels.

Note that this routine does not filter a series on its contents. The filter is applied to the labels of the index.

eg-1: To select rows of Series where index labels are starts with 'S':

We have to use regex parameter (regex-->regular expression)

```
import pandas as pd
s = pd.read_csv('students1.csv',
    usecols=['Name of Student','Marks'],
        index_col='Name of Student',
        squeeze=True)
print(s.filter(regex='^S'))
```

```
D:\durgaclasses>py test.py
Name of Student
Sunny     100.0
```

```
Sachin     300.0
Sharukh    500.0
Salman     700.0
Name: Marks, dtype: float64
```

Note that filteing happend based on index label but not based on values.

eg-2: To select rows of Series where index labels ends with 'n':

```
-----------------------------------------------------------------
import pandas as pd
s = pd.read_csv('students1.csv',
      usecols=['Name of Student','Marks'],
           index_col='Name of Student',
           squeeze=True)
print(s.filter(regex='n$'))

D:\durgaclasses>py test.py
Name of Student
Sachin    300.0
Salman    700.0
Name: Marks, dtype: float64
```

eg-3: To select rows of Series where index labels starts with 'A' or 'B':

```
-----------------------------------------------------------------------
import pandas as pd
```

```
s = pd.read_csv('students1.csv',
      usecols=['Name of Student','Marks'],
           index_col='Name of Student',
           squeeze=True)
print(s.filter(regex='^[AB]'))

D:\durgaclasses>py test.py
Name of Student
Bunny    200.0
Ameer    600.0
Name: Marks, dtype: float64
```

eg-3: To select rows of Series where index labels contain substring 'nn':

```
-----------------------------------------------------------------------
import pandas as pd
s = pd.read_csv('students1.csv',
      usecols=['Name of Student','Marks'],
           index_col='Name of Student',
           squeeze=True)
print(s.filter(like='nn'))

D:\durgaclasses>py test.py
Name of Student
Sunny     100.0
Bunny     200.0
Chinny    300.0
Vinny      NaN
```

```
Pinny     400.0
Zinny     300.0
Kinny     500.0
Minny     600.0
Dinny      NaN
Ginny     700.0
Name: Marks, dtype: float64
```

Note: like parameter is similar to like keyword in SQL queries.

Replacing elements of Series by using mask() method:

Syntax:

```
Series.mask(cond, other=nan, inplace=False, axis=None, level=None,
errors='raise', try_cast=NoDefault.no_default)
```

Replace values where the condition is True.

eg-1: Replace value as 'Failed' where marks are < 300?

```
import pandas as pd
import numpy as np
s = pd.read_csv('students1.csv',
      usecols=['Name of Student','Marks'],
          index_col='Name of Student',
          squeeze=True)
s1 = s.mask(lambda x: x<300,other='Failed')
```

```
print(s1)

D:\durgaclasses>py test.py
Name of Student
Sunny      Failed
Bunny      Failed
Chinny      300.0
Vinny        NaN
Pinny       400.0
Zinny       300.0
Kinny       500.0
Minny       600.0
Dinny        NaN
Ginny       700.0
Sachin      300.0
Dravid       NaN
Kohli      1000.0
Rahul       800.0
Ameer       600.0
Sharukh     500.0
Salman      700.0
Ranveer      NaN
Katrtina    300.0
Kareena     400.0
Name: Marks, dtype: object
```

eg-2: Replace value as first class where marks > 500?

```
import pandas as pd
```

```
import numpy as np
s = pd.read_csv('students1.csv',
      usecols=['Name of Student','Marks'],
           index_col='Name of Student',
           squeeze=True)
s1 = s.mask(lambda x: x>500,other='First Class')
print(s1)
```

Replacing elements of Series by using where() method:

It is counter part of mask() method.

Syntax:

Series.where(cond, other=nan, inplace=False, axis=None, level=None, errors='raise', try_cast=NoDefault.no_default)

Replace values where the condition is False. ie if condition is True then replacement won't be happend.

By using other parameter we can provide new value.

eg-1: Replace value as 'Failed' where marks are < 300?

```
import pandas as pd
import numpy as np
s = pd.read_csv('students1.csv',
      usecols=['Name of Student','Marks'],
           index_col='Name of Student',
           squeeze=True)
s1 = s.where(lambda x: x>= 300,other='Failed')
```

print(s1)

Note: mask() replaces value if the condition is True where as where() replaces value if the condition is False.

Transforming Series object:

Transforming means updating values of Series object.

There are 2 types of transformations

1. **Partial Transformation**
2. **Global/Full Transformation**

1. Partial Tranformation:

A subset of records will be updated but not all.

We can perform this operation by using update() method.

2. Global/Full Transoformation:

It will update full set of records/all records.

We can perform this operation by using either map() method or apply() method.

Transforming Series object by using update() method:

----

without update() method:

We can update a particular element by using normal indexing or loc/iloc indexers.

eg:

```
import pandas as pd
import numpy as np
s = pd.Series([10,20,30,40])
s[0]=100
s.loc[3]=400
print(s)
```

```
D:\durgaclasses>py test.py
0    100
1     20
2     30
3    400
dtype: int64
```

In the above program, to update n elements, we have to use n lines of code.

To perform this operation in a simple way, we should go for update() method.

with update() method:

```
----------------------
import pandas as pd
import numpy as np
s = pd.Series([10,20,30,40])
```

```
s.update(pd.Series([100,400],index=[0,3]))
print(s)

D:\durgaclasses>py test.py
0    100
1     20
2     30
3    400
dtype: int64
```

Syntax of update():

Series.update(other)

Modify Series in place using values from passed Series.

Uses non-NA values from passed Series to make updates. Aligns on index.

Parameters

otherSeries, or object coercible into Series like List or dict etc

eg-1:

```
import pandas as pd
s = pd.Series([1, 2, 3])
s.update(pd.Series([4, 5, 6]))
print(s)

D:\durgaclasses>py test.py
0    4
```

```
1    5
2    6
dtype: int64
```

eg-2:

```
import pandas as pd
s = pd.Series(['a', 'b', 'c'])
s.update(pd.Series(['d', 'e'], index=[0, 2]))
print(s)
```

```
D:\durgaclasses>py test.py
0    d
1    b
2    e
dtype: object
```

eg-2A:

```
import pandas as pd
s1=pd.Series([1,2,3,4],index=[1,2,3,4])
s2=pd.Series([5,6,7,8],index=['a','b','c','d'])
s1.update(s2)
print(s1)
```

```
D:\durgaclasses>py test.py
1    1
2    2
3    3
```

4 4
dtype: int64

eg-3:

```
import pandas as pd
s = pd.Series([1, 2, 3])
s.update(pd.Series([4, 5, 6, 7, 8])) #extra elements will be ignored
print(s)
```

```
D:\durgaclasses>py test.py
0 4
1 5
2 6
dtype: int64
```

eg-4:

If other/argument contains NaNs the corresponding values are not updated in the original Series.

```
import pandas as pd
import numpy as np
s = pd.Series([1, 2, 3])
s.update(pd.Series([4, np.nan, 6]))
print(s)
```

```
D:\durgaclasses>py test.py
0 4
1 2
```

2 6
dtype: int64

eg-5:
Other can also be a non-Series object type like list or dict, that is coercible into a Series.

```
import pandas as pd
import numpy as np
s = pd.Series([1, 2, 3])
s.update([4, np.nan, 6]) # argument is list but not series
print(s)
```

```
D:\durgaclasses>py test.py
0 4
1 2
2 6
dtype: int64
```

eg-6:

```
import pandas as pd
s = pd.Series([1, 2, 3])
s.update({1: 9}) #argument is dict but not series object
print(s)
```

```
D:\durgaclasses>py test.py
0 1
1 9
```

2 3
dtype: int64

Global Transformation of Series by using apply() method:

Syntax:

Series.apply(func, convert_dtype=True, args=(), **kwargs)

Invoke function on every value of the series and returned values will be considered in the result.
1. The function can be ufunc (a NumPy function that applies to the entire Series) or a normal Python function.
2. It returns a new Series with returned values of the function. ie inplace updation is not happend.

eg-1: Square the values by defining a function and passing it as an argument to apply().

```
import pandas as pd
s = pd.Series(data = [20, 21, 12],index=['London', 'New York', 'Helsinki'])

def square(x):
    return x*x

s = s.apply(square)
```

```
print(s)
```

```
D:\durgaclasses>py test.py
London      400
New York    441
Helsinki    144
dtype: int64
```

eg-2: Square the values by passing an anonymous function/Lambda function as an argument to apply().

```
import pandas as pd
s = pd.Series(data = [20, 21, 12],index=['London', 'New York',
'Helsinki'])
s = s.apply(lambda x: x**2)
print(s)
```

eg2A: Square the values by passing numpy ufunc:square as an argument to apply().

```
import pandas as pd
import numpy as np
s = pd.Series(data = [20, 21, 12],index=['London', 'New York',
'Helsinki'])
s1 = s.apply(np.square)
print(s1)
```

```
D:\durgaclasses>py test.py
```

```
London      400
New York    441
Helsinki    144
dtype: int64
```

eg-3: Define a custom function that needs additional positional arguments and pass these additional arguments using the args keyword.
increment every value by 10.

```
import pandas as pd
s = pd.Series(data = [20, 21, 12],index=['London', 'New York', 'Helsinki'])

def increment(x,increment):
        return x+increment

s1 = s.apply(increment,args=(10,))
print(s1)

s2 = s.apply(increment,args=(20,))
print(s2)
```

```
D:\durgaclasses>py test.py
London      30
New York    31
Helsinki    22
dtype: int64
```

```
London      40
New York    41
Helsinki    32
dtype: int64
```

eg-4: Define a custom function that takes keyword arguments and pass these arguments to apply.

```
import pandas as pd
s = pd.Series(data = [20, 21, 12],index=['London', 'New York',
'Helsinki'])

def increment(x,increment_value):
        return x+increment_value

s1 = s.apply(increment,increment_value=10)
print(s1)

s2 = s.apply(increment,increment_value=20)
print(s2)

D:\durgaclasses>py test.py
London      30
New York    31
Helsinki    22
dtype: int64
London      40
New York    41
```

Helsinki 32
dtype: int64

Note:
1. We can pass parameters to the input function by using either args argument or by using keyword arguments.
2. The function can be normal python function/lamda function/numpy ufunc

Global Transformation of Series by using map() method:

It is limited version of apply() method.
Here we cannot pass arguments to the input function.

Syntax:
Series.map(arg, na_action=None)
Map values of Series according to input correspondence.

Used for substituting each value in a Series with another value, that may be derived from a function, a dict or a Series.

eg-1: with dict argument:

Values that are not found in the dict are converted to NaN

```
import pandas as pd
import numpy as np
```

```
s = pd.Series(['cat', 'dog', np.nan, 'rabbit'])
s1 = s.map({'cat': 'kitten', 'dog': 'puppy'})
print(s1)

D:\durgaclasses>py test.py
0    kitten
1     puppy
2       NaN
3       NaN
dtype: object
```

Note: Values for na.nan and rabbit not found and hence converted to NaN.

eg-2: with function argument:

```
----------------------------
import pandas as pd
import numpy as np
s = pd.Series(['cat', 'dog', np.nan, 'rabbit'])
s1 = s.map(lambda x: f'The value is:{x}')
print(s1)

D:\durgaclasses>py test.py
0       The value is:cat
1       The value is:dog
2       The value is:nan
3    The value is:rabbit
dtype: object
```

eg2-A:

```
import pandas as pd
import numpy as np
s = pd.Series(['cat', 'dog', np.nan, 'rabbit'])
def contentadd(x):
        return f'I am {x}'

s1 = s.map(contentadd)
print(s1)
```

```
D:\durgaclasses>py test.py
0      I am cat
1      I am dog
2      I am nan
3    I am rabbit
dtype: object
```

***Note: Here we cannot pass extra arguments to the function.**

Note:
To avoid applying the function to missing values (and keep them as NaN) na_action='ignore' can be used:

eg:

```
import pandas as pd
import numpy as np
```

```
s = pd.Series(['cat', 'dog', np.nan, 'rabbit'])
def contentadd(x):
        return f'I am {x}'

s1 = s.map(contentadd,na_action='ignore')
print(s1)

D:\durgaclasses>py test.py
0      I am cat
1      I am dog
2           NaN
3    I am rabbit
dtype: object

Iterating Elements of the Series:
---------------------------------
Iterating means getting elements one by one.

To get only values:
-------------------
import pandas as pd
s = pd.Series([10,20,30,40,50])
for v in s:
        print(v)

D:\durgaclasses>py test.py
10
20
```

```
30
40
50

To get index labels:
--------------------
import pandas as pd
s = pd.Series([10,20,30,40,50])
for i in s.index:
        print(i)

D:\durgaclasses>py test.py
0
1
2
3
4

To get both index labels and values:
-----------------------------------
1st way:
--------
import pandas as pd
s = pd.Series([10,20,30,40,50])
for i in s.index:
        print(f'{i} --->{s[i]}')

D:\durgaclasses>py test.py
```

```
0 --->10
1 --->20
2 --->30
3 --->40
4 --->50
```

2nd way:

```
-------
import pandas as pd
s = pd.Series([10,20,30,40,50])
for i,v in s.items():
        print(f'{i} --->{v}')

D:\durgaclasses>py test.py
0 --->10
1 --->20
2 --->30
3 --->40
4 --->50
```

3rd way:

```
--------
import pandas as pd
s = pd.Series([10,20,30,40,50])
for i,v in s.iteritems():
        print(f'{i} --->{v}')

D:\durgaclasses>py test.py
```

0 --->10

1 --->20

2 --->30

3 --->40

4 --->50

Case Study:

File Name: learners2.cvs

_id	name	marks
1	narayan pradhan	10
2	abhilash	20
3	rasika	30
4	pankaj bhandari	40
5	Sheshanand Singh	50

....

Q1. What is the highest marks?

Q2. What is the least marks?

Q3. What is the mean value of marks?

Q4. What is the stadard deviation of marks?

Q5. What is the variance of marks?

Q6. What is the median value of marks?

Q7. Select only 20 students data and identify whether the marks are relatively low or high to the rest of the sample?

Solutions:

```
import pandas as pd
s = pd.read_csv('learners2.csv',
      usecols=['name','marks'],
           index_col='name',
           squeeze=True)
print('Highest Marks:',s.max())
print('Least Marks:',s.min())
print('Mean Value of the Marks:',s.mean())
print('Standard Deviation of the Marks:',s.std())
print('Variance of the Marks:',s.var())
print('Median of the Marks:',s.median())
```

Q7. Select only 20 students data and identify whether the marks are relatively low or high to the rest of the sample?

```
import pandas as pd
s = pd.read_csv('learners2.csv',
      usecols=['name','marks'],
           index_col='name',
           squeeze=True)
first_20s = s.head(20) #s[:20]
print((first_20s-s.mean()).apply(lambda x: 'Less Marks' if x<0 else
'High Marks'))
print((s-s.mean()).apply(lambda x: 'Less Marks' if x<0 else 'High
Marks').value_counts())
```

```
D:\durgaclasses>py test.py
```

name
narayan pradhan Less Marks
abhilash Less Marks
rasika Less Marks
pankaj bhandari Less Marks
Sheshanand Singh Less Marks
dhanaraju High Marks
Satyasundar Panigrahi High Marks
jyothi High Marks
Hari High Marks
bindhiya High Marks
vikas kale Less Marks
Sunita Kumati Choudhuri Less Marks
shashank sanap Less Marks
Atul Less Marks
TharunK Less Marks
aron High Marks
pooja High Marks
Dusmant Kumar Mohapatra High Marks
Deepak High Marks
Bhim Kumar High Marks
Name: marks, dtype: object
Less Marks 3286
High Marks 3285
Name: marks, dtype: int64

Quiz:

Q1. Which of the following method can be used to call a python function for every value present in the Series object, even that function is allow extra arguments?

A. update()
B. apply()
C. map()
D. items()

Ans: B

Q2. In general, which of the following parameter can be used to perform modifictons in the existing series object only instead of creating new series object?

A. inline
B. inplace
C. overwrite
D. existing

Ans: B

Q3. Which of the following statements are valid about Series object?

A. It is always one dimensional.
B. Values can be duplicated.
C. Values should be unique.

D. Index Labels can be duplicated.
E. Index Labels should be unique.

Ans:A,B,D

Q4. Which of the following is valid way to check whether index label 'Durga' present in Series object s?

A. print('Durga' in s)
B. print('Durga' in s.values)
C. print('Durga' in s.index)

Ans: A,C
Explanation: in operator bydefault will search in indexes.

Q5. Default number of rows returned by head() or tail() methods on Series object?

A. 10
B. 20
C. 7
D. 5

Ans: D

Q6. The default return type of read_csv() function is

A. DataFrame

B. Series

Ans: A

Q7. To get Series object from read_csv() function which of the following parameter we have to use?

A. series = True
B. dataframe = False
C. squeeze = True
D. squeeze = False

Ans: C
squeeze parameter can be used to convert DataFrame to Series object.