

Introducing Python Pandas

Based on CBSE Curriculum

Class -11



Chapter-10

By-

Neha Tyagi

PGT CS

KV 5 Jaipur II Shift

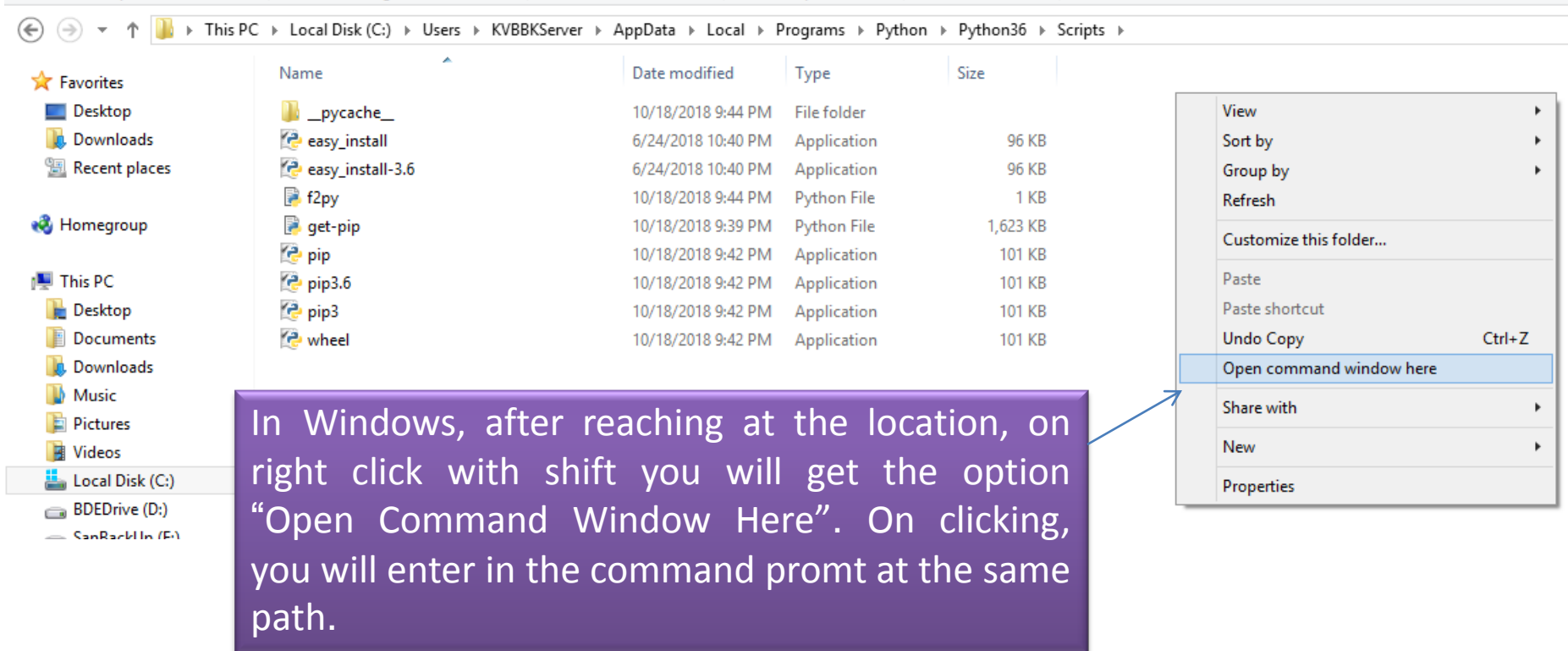
Jaipur Region

Introduction

- Pandas or Python Pandas is a library of Python which is used for data analysis.
- The term Pandas is derived from “Panel data system” , which is an econometric term for multidimensional, structured data set econometrics.
- Now a days, Pandas has become a popular option for Data Analysis.
- Pandas provides various tools for data analysis in simpler form.
- Pandas is an Open Source, BSD library built for Python Programming language.
- Pandas offers high performance, easy to use data structure and data analysis tools.
- The main author of Pandas is Wes McKinney.
- In this chapter, we will learn about Pandas.

Installing Pandas

- “pip” command is used to install Pandas. For this, open the location of pip storage in command prompt (cmd). Goto the location in windows where pip file is stored. look at the following screen-



The screenshot shows a Windows File Explorer window with the address bar displaying the path: This PC > Local Disk (C:) > Users > KVBBKServer > AppData > Local > Programs > Python > Python36 > Scripts. The main pane shows a list of files and folders:

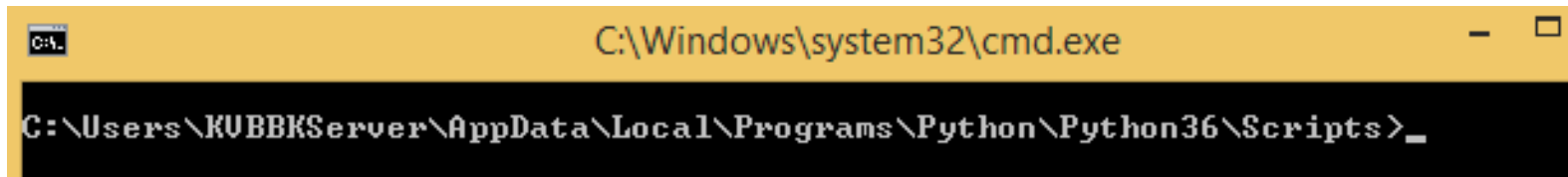
Name	Date modified	Type	Size
__pycache__	10/18/2018 9:44 PM	File folder	
easy_install	6/24/2018 10:40 PM	Application	96 KB
easy_install-3.6	6/24/2018 10:40 PM	Application	96 KB
f2py	10/18/2018 9:44 PM	Python File	1 KB
get-pip	10/18/2018 9:39 PM	Python File	1,623 KB
pip	10/18/2018 9:42 PM	Application	101 KB
pip3.6	10/18/2018 9:42 PM	Application	101 KB
pip3	10/18/2018 9:42 PM	Application	101 KB
wheel	10/18/2018 9:42 PM	Application	101 KB

A context menu is open over the 'pip' file, with the option 'Open command window here' highlighted. A blue arrow points from this option to a purple text box containing the following text:

In Windows, after reaching at the location, on right click with shift you will get the option “Open Command Window Here”. On clicking, you will enter in the command prompt at the same path.

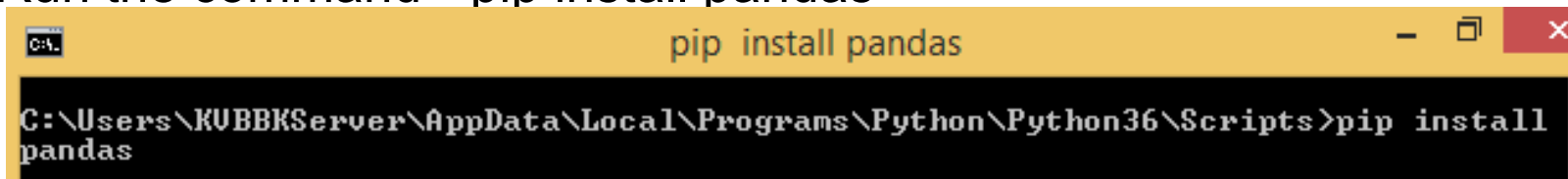
Installing Pandas

- Command window will look like-



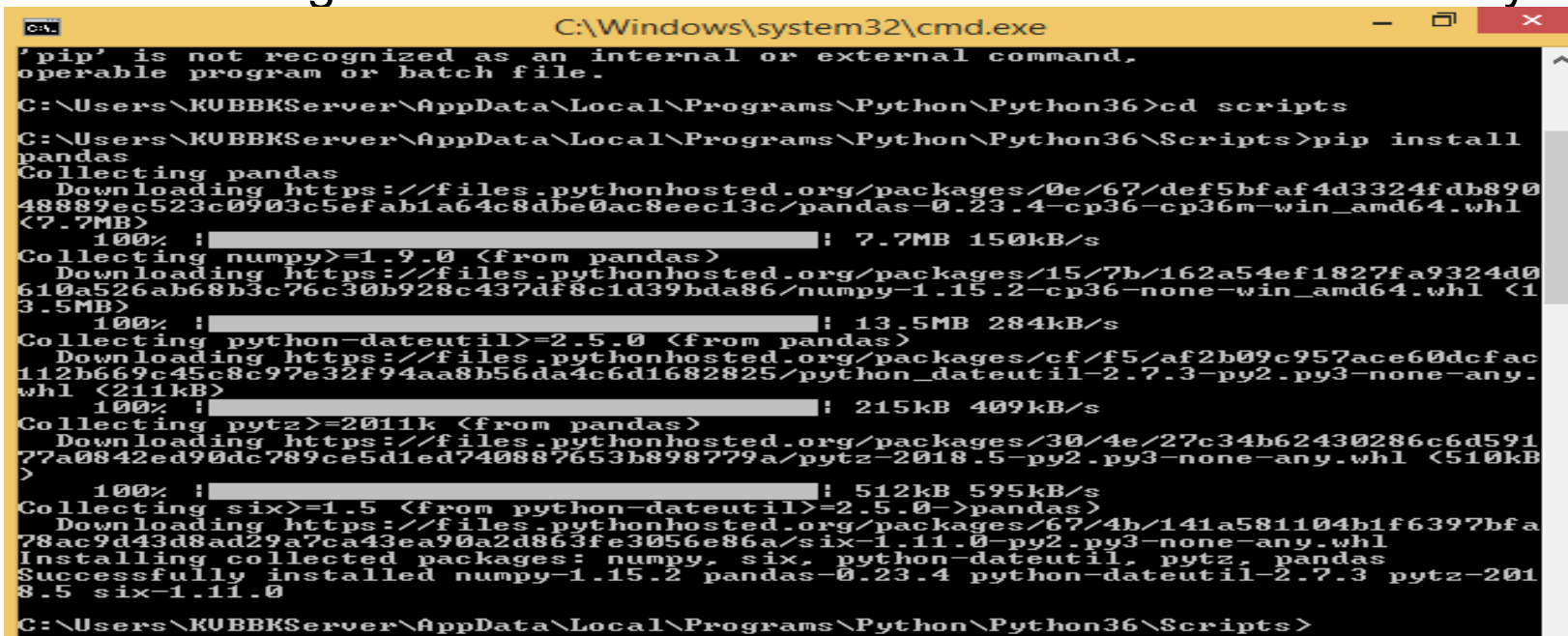
```
C:\Windows\system32\cmd.exe  
C:\Users\KUBBKServer\AppData\Local\Programs\Python\Python36\Scripts>
```

- Run the command- “pip install pandas”



```
C:\Windows\system32\cmd.exe  
pip install pandas  
C:\Users\KUBBKServer\AppData\Local\Programs\Python\Python36\Scripts>pip install pandas
```

- The following screen comes after and Pandas will be successfully installed.



```
C:\Windows\system32\cmd.exe  
'pip' is not recognized as an internal or external command,  
operable program or batch file.  
C:\Users\KUBBKServer\AppData\Local\Programs\Python\Python36>cd scripts  
C:\Users\KUBBKServer\AppData\Local\Programs\Python\Python36\Scripts>pip install pandas  
Collecting pandas  
  Downloading https://files.pythonhosted.org/packages/0e/67/def5bfaf4d3324fdb89048889ec523c0903c5efab1a64c8dbe0ac8eec13c/pandas-0.23.4-cp36-cp36m-win_amd64.whl (7.7MB)  
100% |#####| 7.7MB 150kB/s  
Collecting numpy>=1.9.0 (from pandas)  
  Downloading https://files.pythonhosted.org/packages/15/7b/162a54ef1827fa9324d0610a526ab68b3c76c30b928c437df8c1d39bda86/numpy-1.15.2-cp36-none-win_amd64.whl (13.5MB)  
100% |#####| 13.5MB 284kB/s  
Collecting python-dateutil>=2.5.0 (from pandas)  
  Downloading https://files.pythonhosted.org/packages/cf/f5/af2b09c957ace60dcfac112b669c45c8c97e32f94aa8b56da4c6d1682825/python_dateutil-2.7.3-py2.py3-none-any.whl (211kB)  
100% |#####| 215kB 409kB/s  
Collecting pytz>=2011k (from pandas)  
  Downloading https://files.pythonhosted.org/packages/30/4e/27c34b62430286c6d59177a0842ed90dc789ce5d1ed740887653b898779a/pytz-2018.5-py2.py3-none-any.whl (510kB)  
>  
100% |#####| 512kB 595kB/s  
Collecting six>=1.5 (from python-dateutil>=2.5.0->pandas)  
  Downloading https://files.pythonhosted.org/packages/67/4b/141a581104b1f6397bfa78ac9d43d8ad29a7ca43ea90a2d863fe3056e86a/six-1.11.0-py2.py3-none-any.whl  
Installing collected packages: numpy, six, python-dateutil, pytz, pandas  
Successfully installed numpy-1.15.2 pandas-0.23.4 python-dateutil-2.7.3 pytz-2018.5 six-1.11.0  
C:\Users\KUBBKServer\AppData\Local\Programs\Python\Python36\Scripts>
```

Using Pandas

- Before proceeding, we need to first import the Pandas.

```
>>> import pandas
>>> help(pandas)
Help on package pandas:
```

Help(pandas) command will give you all information about Pandas module.

```
NAME
```

```
    pandas
```

```
DESCRIPTION
```

```
pandas - a powerful data analysis and manipulation library for Python
```

```
=====
```

```
pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block
```

```
for
```

```
doing practical, real world data analysis in Python. Additionally, it has
```

```
s
```

```
the broader goal of becoming the most powerful and flexible open source data
```

```
ta
```

```
analysis / manipulation tool available in any language. It is already well
```

```
on
```

```
its way toward this goal.
```

Features of Pandas

- Pandas, is the most popular library in Scientific Python ecosystem for doing data analysis. Pandas is capable of many tasks including-
 1. It can read or write in many different data formats(Integer, float, double etc).
 2. It can calculate in all ways data is organized.
 3. It can easily select subsets of data from bulky data sets and even combine multiple datasets together.
 4. It has functionality to find and fill missing data.
 5. It allows you to apply operations to independent groups within the data.
 6. It supports reshaping of data into different forms.
 7. It supports advanced time-series functionality(which is the use of a model to predict future values based on previously observed values).
 8. It supports visualization by integrating matplotlib and seaborn etc libraries.
- Pandas is best at handling huge tabular data sets comprising different data formats.

NumPy Arrays

- Before proceeding towards Pandas' data structure, let us have a brief review of NumPy arrays because-
 1. Pandas' some functions return result in form of NumPy array.
 2. It will give you a jumpstart with data structure.
- NumPy ("Numerical Python" or Numeric Python") is an open source module of Python that provides functions for fast mathematical computation on arrays and matrices.
- To use NumPy, it is needed to import. Syntax for that is-

```
>>>import numpy as np
```

(here np, is an alias for numpy which is optional)

- NumPy arrays come in two forms-
 - 1-D array – also known as Vectors.
 - Multidimensional arrays –
Also known as Matrices.

```
>>> import numpy as np
>>> lst = [1,2,3,4]
>>> a1=np.array(lst)
>>> lst
[1, 2, 3, 4]
>>> print(a1)
[1 2 3 4]
>>> a1
array([1, 2, 3, 4])
```

See the difference between List and array

2D NumPy Arrays

```
>>> import numpy as np
>>> A=np.array([[10,11,12,13],[21,22,23,24]])
>>> A[1,3]
24
>>> A[1][3]
24
>>> A
array([[10, 11, 12, 13],
       [21, 22, 23, 24]])
>>> print(A)
[[10 11 12 13]
 [21 22 23 24]]
>>> type(A),type(a1)
(<class 'numpy.ndarray'>, <class 'numpy.ndarray'>)
>>> a1.shape
(4,)
>>> A.shape
(2, 4)
>>> A.itemsize
4
```

Accessing Array
elemets with
index

With the help
of list, 2D array
is created.

Printing of Array

To see type of
Array

To see shape of
Array (use of
different functions)

NumPy arrays arr also known as ndarray (n-dimentional array)

NumPy Arrays Vs Python Lists

- Although NumPy array also holds elements like Python List, yet NumPy arrays are different data structures from Python list. The key differences are-
- Once a NumPy array is created, you cannot change its size. you will have to create a new array or overwrite the existing one.
- NumPy array contain elements of homogenous type, unlike python lists.
- An equivalent NumPy array occupies much less space than a Python list.
- NumPy array supports Vectorized operation, i.e. you need to perform any function on every item one by one which is not in

```
>>> import numpy as np
>>> lst=[1,2,3,4]
>>> a1=np.array(lst)
>>> lst+2
Traceback (most recent
File "<pyshell#4>", l
```

In list, it will generate error but will be executed in arrays.

```
>>> a1+2
array([3, 4, 5, 6])
```

NumPy Data Types

NumPy supports following data types-

No.	Data Type	Description	Size
1.	bool_	Boolean data type (stores <i>True</i> or <i>False</i>)	1 byte
2.	int_	Default type to store integers in <i>int32</i> or <i>int64</i>	4 or 8 bytes
3.	int8	Stores signed integers in range -128 to 127	1 byte
4.	int16	Stores signed integers in range -32768 to 32767	2 bytes
5.	int32	Stores signed integers in range -2^{16} to $2^{16} - 1$	4 bytes
6.	int64	Stores signed integers in range -2^{32} to $2^{32} - 1$	8 bytes
7.	uint8	Stores unsigned integers in range 0 to 255	1 byte
8.	uint16	Stores integers in range 0 to $2^{16} - 1$	2 bytes
9.	uint32	Stores integers in range 0 to $2^{32} - 1$	4 bytes
10.	uint64	Stores integers in range 0 to $2^{64} - 1$	8 bytes
11.	float_	Default type to store floating point (<i>float64</i>)	8 bytes
12.	float16	Stores half precision floating point values (5 bits exponent, 10 bit mantissa)	2 bytes
13.	float32	Stores single precision floating point values (8 bits exponent, 23 bit mantissa)	4 bytes
14.	float64	Stores double precision floating point values (11 bits exponent, 52 bit mantissa)	8 bytes
15.	complex_	Default type to store complex numbers (<i>complex128</i>)	16 bytes
16.	complex64	Complex numbers represented by <i>two float32 numbers</i> for real and <i>imaginary</i> value components.	8 bytes
17.	complex128	Complex numbers represented by <i>two float64 numbers</i> for real and <i>imaginary</i> value components.	16 bytes
18.	string_	Fixed-length string type.	1 byte per character
19.	unicode_	Fixed-length Unicode type.	number of bytes platform specific

Ways to Create NumPy Arrays

- `empty()` function can be used to create empty array or an uninitialized array of specified shape and dtype.

```
numpy.empty(Shape,[dtype=<datatype>],[ order = 'C' or 'F']
```

Where:dtype: is a data type of python or numpy to set initial values.

Shape: is dimension.

Order : 'C' means arrangement of data as row wise(C means C like).

Order : 'F' means arrangement of data as row wise (F means Fortran like)

```
>>> arr1=np.empty([2,3],dtype=np.int64, order='C')
>>> arr1
array([[0, 0, 0],
       [0, 0, 0]], dtype=int64)
```

Here, array is of all zeros

```
>>> arr2=np.empty([3,2])
>>> arr2
array([[6.23042070e-307, 4.67296746e-307],
       [1.69121096e-306, 2.00271247e-307],
       [9.34587382e-307, 1.60219035e-306]])
```

Here, array is of all garbage values and of default type "float"

Ways to Create NumPy Arrays

1. `arange()` function is used to create array from a range.

```
<arrayname> = numpy.arange([start],stop,[step],[dtype])
```

```
>>> import numpy as np
>>> arr1=np.arange(5)
>>> arr1
array([0, 1, 2, 3, 4])
```

Here, only stop value is passed.

```
>>> import numpy as np
>>> arr2=np.arange(1,7,2,dtype=np.float32)
>>> arr2
array([1., 3., 5.], dtype=float32)
```

Here, from 1-7 at the step of 2.

2. `linspace()` function can be used to prepare array of range.

```
<arrayname> = numpy.linspace([start],stop,[dtype])
```

```
>>> import numpy as np
>>> arr3 = np.linspace(2,3,6)
>>> arr3
array([2. , 2.2, 2.4, 2.6, 2.8, 3. ])
```

Here, an array of 6 values is created between the values 2 and 3.

```
>>> import numpy as np
>>> arr4 = np.linspace(2.5,5,8)
>>> arr4
array([2.5       , 2.85714286, 3.21428571, 3.57142857, 3.92857143,
       4.28571429, 4.64285714, 5.         ])
```

Here, an array of 8 values is created between the values 2.5 and 5.

Pandas Data Structure

“A data structure is a particular way of storing and organizing data in a computer so that it can be accessed and worked with in appropriate ways. For ex-

-If you want to store similar type of data items together and process them in identical way , array is the solution.

- If you want to store data in such a way so that you get access of the very last data item you inserted, stack is the solution.

-If you want to store data in such a way so that data item inserted first get accessed first, Queue is the solution.

there are many more other types of data structure suited for different types of functionality.

Further, We will come to know about Series and DataFrame data structures of Python.

Series Data Structure

- Series is a data structure of pandas. It represents a 1D array of indexed data.
- It has two main components-
 - An array of actual data.
 - An associated array of indexes or data labels.
- Both components are 1D arrays with the same length.

Index	Data
0	21
1	23
2	18
3	25

Index	Data
Jan	31
Feb	28
Mar	31
Apr	30

Index	Data
'A'	91
'B'	81
'C'	71
'D'	61

Examples of series type objects.

Creation of Series Objects

– There are many ways to create series type object.

1. Using Series ()-

<Series Object> = pandas.Series() it will create empty series.


```
>>> import pandas as pd
>>> ob = pd.Series()
>>> ob
Series([], dtype: float64)
```

2. Non-empty series creation–


Import pandas as pd

<Series Object> = pd.Series(data, index=idx) where data can be python sequence, ndarray, python dictionary or scaler value.

```
>>> import pandas as pd
>>> ob = pd.Series(range(5))
>>> ob
0    0
1    1
2    2
3    3
4    4
dtype: int64
```




```
>>> import pandas as pd
>>> obj=pd.Series([3,5,4,4.5])
>>> obj
0    3.0
1    5.0
2    4.0
3    4.5
dtype: float64
```



Series Objects creation

1. Creation of series with Dictionary-

Index of
Keys



```
>>> import pandas as pd
>>> obj=pd.Series({'Jan':31, 'Feb':28, 'Mar':31})
>>> obj
Jan      31
Feb      28
Mar      31
dtype: int64
```

2. Creation of series with Scalar value-

```
>>> import pandas as pd
>>> a=pd.Series(10,index=range(0,3))
>>> a
0      10
1      10
2      10
dtype: int64
```

```
>>> import pandas as pd
>>> b=pd.Series(15,index=range(1,6,2))
>>> b
1      15
3      15
5      15
dtype: int64
```

```
>>> import pandas as pd
>>> c=pd.Series('Welcome to BBK', index=['Hema', 'Rahul', 'Anup'])
>>> c
Hema      Welcome to BBK
Rahul     Welcome to BBK
Anup      Welcome to BBK
dtype: object
```


Creation of Series Objects –Additional functionality

1. When it is needed to create a series with missing values, this can be achieved by filling missing data with a NaN (“Not a Number”) value.

```
>>> import pandas as pd
>>> import numpy as np
>>> ob=pd.Series([6.5,np.NaN,2.34])
>>> ob
0      6.50
1      NaN
2      2.34
dtype: float64
```

2. Index can also be given as-

```
>>> import pandas as pd
>>> s=pd.Series(range(1,15,3), index=[x for x in 'abcde'])
>>> s
a      1
b      4
c      7
d     10
e     13
dtype: int64
```

Loop is used to give Index

Creation of Series Objects –Additional functionality

3. Dtype can also be passed with Data and index

```
>>> import pandas as pd
>>> import numpy as np
>>> ob=pd.Series(data=arr,index=mon, dtype=np.float64)
>>> ob
Jan      31.0
Feb      28.0
Mar      31.0
Apr      30.0
dtype: float64
```

Important: it is not necessary to have unique indices but it will give error when search will be according to index.

4. Mathematical function/Expression can also be used-

```
>>> import pandas as pd
>>> import numpy as np
>>> a=np.arange(9,13)
>>> a
array([ 9, 10, 11, 12])
>>> ob=pd.Series(index=a, data=a*2)
>>> ob
9      18
10     20
11     22
12     24
dtype: int32
```

```
>>> import pandas as pd
>>> import numpy as np
>>> a=np.arange(9,13)
>>> ob=pd.Series(index=a, data=a**2)
>>> ob
9      81
10    100
11    121
12    144
dtype: int32
```

Series Object Attributes

3. Some common attributes-

<series object>.<AttributeName>

Attribute	Description
Series.index	Returns index of the series
Series.values	Returns ndarray
Series.dtype	Returns dtype object of the underlying data
Series.shape	Returns tuple of the shape of underlying data
Series.nbytes	Return number of bytes of underlying data
Series.ndim	Returns the number of dimension
Series.size	Returns number of elements
Series.itemsize	Returns the size of the dtype
Series.hasnans	Returns true if there are any NaN
Series.empty	Returns true if series object is empty

Series Object Attributes

```
>>> import pandas as pd
>>> s=pd.Series(range(1,15,3), index=[x for x in 'abcde'])
>>> s.index
Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
>>> s.values
array([ 1,  4,  7, 10, 13], dtype=int64)
>>> s.shape
(5,)
>>> s.size
5
>>> s.nbytes
40
>>> s.ndim
1
>>> s.itemsize
```

Accessing Series Object

```
>>> import pandas as pd
>>> import numpy as np
>>> a=np.arange(9,13)
>>> ob=pd.Series(index=a, data=a**2)
>>> ob
9      81
10     100
11     121
12     144
dtype: int32
>>> ob[10]
100
```

Printing object value

Printing Individual value

```
>>> ob[2:4]
11     121
12     144
dtype: int32
>>> ob[1:]
10     100
11     121
12     144
dtype: int32
>>> ob[0::2]
9      81
11     121
dtype: int32
```

Object
slicing



```
>>> ob[::-1]
12     144
11     121
10     100
9      81
dtype: int32
```

For Object slicing, follow the following syntax-

<objectName>[<start>:<stop>:<step >]

Operations on Series Object

1. Elements modification-

<series object>[index] = <new_data_value>

```
>>> import pandas as pd
>>> s=pd.Series(range(1,15,3), index=[x for x in 'abcde'])
```

```
>>> s
a      1
b      4
c      7
d     10
e     13
dtype: int64
>>> s['c']=17
>>> s
a      1
b      4
c     17
d     10
e     13
dtype: int64
```

To change individual value

```
>>> s
a      1
b      4
c     17
d     10
e     13
dtype: int64
>>> s[2:4]=100
>>> s
a      1
b      4
c    100
d    100
e     13
dtype: int64
```

To change value in a certain slice

```
>>> s[1:5:2]=100
>>> s
a      1
b    100
c      7
d    100
e     13
dtype: int64
```

Operations on Series Object

1. It is possible to change indexes

<series object>.<index> = <new_index_array>

```
>>> import pandas as pd
>>> s=pd.Series(range(1,15,3), index=[x for x in 'abcde'])
```

```
>>> s
a      1
b      4
c      7
d     10
e     13
dtype: int64
>>> s.index=['u','v','w','x','y']
>>> s
u      1
v      4
w      7
x     10
y     13
dtype: int64
```

Here, indexes got changed.

head() and tail () Function

1. head(<n>) function fetch first n rows from a pandas object. If you do not provide any value for n, will return first 5 rows.
2. tail(<n>) function fetch last n rows from a pandas object. If you do not provide any value for n, will return last 5 rows.

```
>>> import pandas as pd
>>> import math
>>> s=pd.Series(data=[math.sqrt(x) for x in range(1,10)],index=[x for x in range(1,10)])
```

```
>>> s
1    1.000000
2    1.414214
3    1.732051
4    2.000000
5    2.236068
6    2.449490
7    2.645751
8    2.828427
9    3.000000
dtype: float64
```

```
>>> s.head(6)
1    1.000000
2    1.414214
3    1.732051
4    2.000000
5    2.236068
6    2.449490
dtype: float64
```

```
>>> s.tail(7)
3    1.732051
4    2.000000
5    2.236068
6    2.449490
7    2.645751
8    2.828427
9    3.000000
dtype: float64
```

```
>>> s.head()
1    1.000000
2    1.414214
3    1.732051
4    2.000000
5    2.236068
dtype: float64
>>> s.tail()
5    2.236068
6    2.449490
7    2.645751
8    2.828427
9    3.000000
dtype: float64
```


Series Objects - Vector Operations


```
>>> s
1    11
2    12
3    13
4    14
dtype: int64
```

```
>>> s+2
1    13
2    14
3    15
4    16
dtype: int64
```

```
>>> s*3
1    33
2    36
3    39
4    42
dtype: int64
```

```
>>> s**2
1    121
2    144
3    169
4    196
dtype: int64
```

All these are
vector operations



```
>>> s>13
1    False
2    False
3    False
4     True
dtype: bool
```

Series Objects - Arithmetic Operations

```
>>> s
1    11
2    12
3    13
4    14
dtype: int64
```

```
>>> s1
1    21
2    22
3    23
4    24
dtype: int64
```

```
>>> s3
101    21
102    22
103    23
104    24
dtype: int64
```

```
>>> s+s1
1    32
2    34
3    36
4    38
dtype: int64
```

```
>>> s*s1
1    231
2    264
3    299
4    336
dtype: int64
```

```
>>> s/s1
1    0.523810
2    0.545455
3    0.565217
4    0.583333
dtype: float64
```

```
>>> s+s3
1     NaN
2     NaN
3     NaN
4     NaN
101    NaN
102    NaN
103    NaN
104    NaN
dtype: float64
```

Arithmetic operation is
possible on objects of
same index otherwise
will result as NaN.

Entries Filtering

<seriesObject> <series - boolean expression >

```
>>> s
1    1.000000
2    1.414214
3    1.732051
4    2.000000
dtype: float64
>>> s<2
1     True
2     True
3     True
4    False
dtype: bool
>>> s[s<2]
1    1.000000
2    1.414214
3    1.732051
dtype: float64
>>> s[s>=2]
4    2.0
dtype: float64
```

Other feature

```
>>> s
1    1.000000
2    1.414214
3    1.732051
4    2.000000
dtype: float64
>>> s.drop(3)
1    1.000000
2    1.414214
4    2.000000
dtype: float64
```

To delete value of index

Difference between NumPy array Series objects

1. In case of ndarray, vector operation is possible only when ndarray are of similar shape. Whereas in case of series object, it will be aligned only with matching index otherwise NaN will be returned.

```
>>> import numpy as np
>>> a=np.array([1,2,3])
>>> b=np.array([1,2,3,45,5])
>>> a+b
Traceback (most recent call last):
  File "<pyshell#143>", line 1, in <module>
    a+b
ValueError: operands could not be broadcast together with shapes (3,) (5,)
```

2. In ndarray, index always starts from 0 and always numeric. Whereas, in series, index can be of any type including number and not necessary to start from 0.

Thank you

Please follow us on our blog

www.pythontrends.wordpress.com