Python-Module3
Pandas

Introduction

- Pandas stands for **Pan**el **Da**ta.
- It is an open-source Python Library.
- It provides high-performance data manipulation and analysis tool through its data structures.
- Standard Python distribution doesn't come bundled with Pandas module. To install Pandas use the following command:
  - Pip install pandas
- PIP stands for Python package installer
- **Pandas is used to analyze data.**

Why Use Pandas?

- Pandas allows us to analyze big data and make conclusions based on statistical theories.
- Pandas can clean messy data sets, and make them readable and relevant.
- Relevant data is very important in data science.

Introduction

- Before using the objects and routines available in Pandas we have to write the following command:
  - Import pandas
  - import pandas as pd
  - from pandas import *

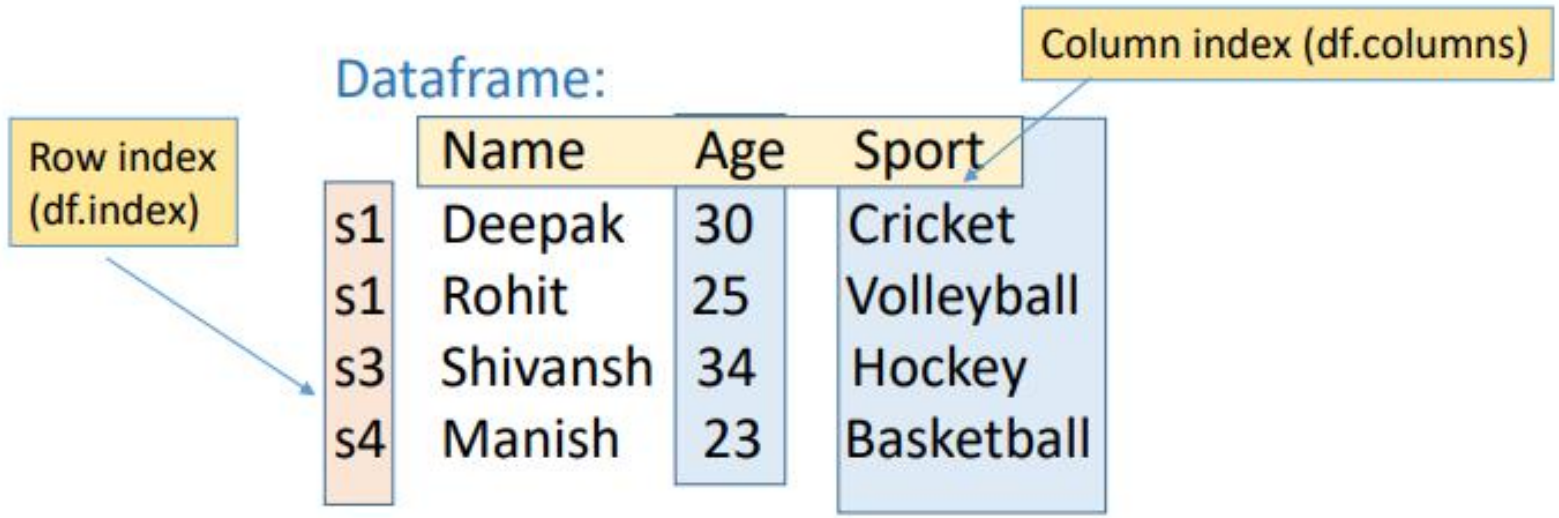
Pandas Data Structures

- Series
  - 1-D array of data and associated array of data labels (index).
- Dataframes
  - 2-D data structure with labels which can store heterogeneous data.

Series:

Index	Data
1	10.0
3	20.0
4	30.0

Dataframe:



Data Structure-Series

- A Pandas Series is like a column in a table.
- It is a one-dimensional array holding data of any type.
- A pandas Series can be created using the various inputs like array, dictionary and constants. Its syntax is:

  pandas.Series(data, index, dtype)

  - data: takes various forms like ndarray, list, constants
  - Index: to specify the index of elements
  - dtype: data type for data

Data Structure-Series

```
import pandas as pd
a = [1, 7, 2]
myvar = pd.Series(a)
print(myvar)
```

Output:

```
0  1
1  7
2  2
dtype: int64
```

Create a Series from ndarray without index value:

```
from pandas import *
from numpy import *
data=array(['a','b','c'])
s = Series(data)
print(s)
```

Output:

```
0  a
1  b
2  c
dtype: object
```

- **Note:if nothing else is specified, the values are labeled with their index number. First value has index 0, second value has index 1 etc.**

Data Structure-Series

- Create a Series from ndarray with customized index value:

```
from pandas import *
from numpy import *
data=array(['a','b','c','d','e'])
s = Series(data, index=[100,101,102,103,104])
print(s)
```

Data Structure-Series

- Create a Series from dictionary: A dictionary can be passed as input and if no index is specified, then the dictionary keys are taken in a sorted order to construct index.

```
from pandas import *
data={'a' : 0., 'b' : 1., 'c' : 2.}
s = Series(data)
print(s)
```

Output:

- **a 0.0**
- **b 1.0**
- **c 2.0**
- **dtype: float64**

Data Structure-Series

- Create a Series from dictionary: If index is passed, the values in data corresponding to the labels in the index will be pulled out.

```
from pandas import *
data={'a' : 0., 'b' : 1., 'c' : 2.}
s = Series(data, index=['b', 'c', 'd', 'a'])
print(s)
```

Output:

```
b    1.0
c    2.0
d    NaN
a    0.0
dtype: float64
```

Data Structure-Series

- Create a Series from constant/scalar: If data is a constant value, an index must be provided. The value will be repeated to match the length of index

```
from pandas import *
s = Series(5, index=[0,1,2,3])
print(s)
```

Output:

```
0  5
1  5
2  5
3  5
dtype: int64
```

Data Structure-Series

- Retrieve Data Using Label (Index) : A Series is like a fixed-size dictionary in that you can get and set values by index label. If a label is not contained, an exception is raised.

```
from pandas import *
s = Series([1,2,3,4,5],index= ['a','b','c','d','e'])
print(s['a'])
print(s[[‘b','c','e']])
```

Data Structure-Series-Basic Functions

Sr. No.	Attribute or Method	Description
1	axes	Returns a list of the row axis labels.
2	dtype	Returns the dtype of the object.
3	empty	Returns True if series is empty.
4	ndim	Returns the number of dimensions of the underlying data, by definition 1.
5	size	Returns the number of elements in the underlying data.
6	values	Returns the Series as one dimensional array.
7	head(n)	Returns the first n rows.
8	tail(n)	Returns the last n rows.

Data Structure-Series

```
from pandas import *
from numpy import *
s = Series(random.randn(10))
print(s)
print("Row axis Label of the series: ",s.axes)
print("datatype of series is:",s.dtype)
print ("Is the Object empty?: ", s.empty)
print("no of dimension are: ", s.ndim)
print("No of elements in this series are: ", s.size)
print("Series elements in the form of array are:", s.values)
print("First three rows of series are: ",s.head(3))
print("Last four rows of series are",s.tail(4))
```

```
0   -0.275933
1    0.208448
2   -0.277693
3   -0.482954
4   -1.141907
5    1.256792
6   -1.690586
7    1.195104
8    2.325411
9    0.032288
dtype: float64
Row axis Label of the series:  [RangeIndex(start=0, stop=10,
step=1)]
datatype of series is: float64
Is the Object empty?:  False
no of dimension are:  1
No of elements in this series are:  10
Series elements in the form of array are: [-0.2759329   0.2084481
0.27769338 -0.48295376 -1.14190741  1.25679206 -1.6905861   1.19510438  2.
0.03228808]
First three rows of series are:  0   -0.275933
1    0.208448
2   -0.277693
Last four rows of series are 6   -1.690586
7    1.195104
8    2.325411
9    0.032288
dtype: float64
```

Data Structure-DataFrame

- A Pandas DataFrame is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns.
- A pandas DataFrame can be created using the various inputs like ndarray, series, map, list, dictionary and constants. Its syntax is:

  pandas.DataFrame(data, index, columns, dtype)

  - data: takes various forms like ndarray, series, map, lists, dict, and constants.
  - index: value must be same as data length
  - columns: For column labels, the optional default syntax is - np.arrange(n). This is only true if no index is passed.
  - dtype: data type of each column

```
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

#load data into a DataFrame object:
df = pd.DataFrame(data)

print(df)
```

Output

```
   calories  duration
0       420        50
1       380        40
2       390        45
```

Locate Row

As you can see from the result above, the DataFrame is like a table with rows and columns.

Pandas use the loc attribute to return one or more specified row(s)

Example

Return row 0:

#refer to the row index:

print(df.loc[0])

Result

```
calories    420
duration     50
Name: 0, dtype: int64
```

Return row 0 and 1:

#use a list of indexes:

print(df.loc[[0, 1]])

Result

	calories	duration
0	420	50
1	380	40

Named Indexes

With the index argument, you can name your own indexes.

Example: Add a list of names to give each row a name:

```
import pandas as pd
data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}
df = pd.DataFrame(data, index = ["day1", "day2", "day3"])
print(df)
```

Result

```
      calories  duration
day1       420        50
day2       380        40
day3       390        45
```

Data Structure-DataFrame

- To Create an Empty DataFrame:

```
from pandas import *
s = DataFrame()
print(s)
```

Iterating over a DataFrame

- DataFrame follow the dict-like convention of iterating over the “keys” of the objects.

Basic iteration (for i in object) produces:

- Series : values
- DataFrame : column labels

dtf1

	population	avg income	per capita
Delhi	7897667	78976	0.010000
Mumbai	4577637	45637	0.009970
Dehradun	6457324	67324	0.010426

```
population=pd.Series([7897667,4577637,6457324],index=["Delhi","Mumbai","Dehradun"])
avgincome=pd.Series([78976,45637,67324],index=["Delhi","Mumbai","Dehradun"])
percapita=avgincome/population
dict1={"population":population,"avg income":avgincome,"per capita":percapita}
dtf1=pd.DataFrame(dict1)
for i in dtf1:
    print(i)
```

```
population
avg income
per capita
```

Iterating over a DataFrame

- iterrows(): Iterate over the rows of a DataFrame as (index, Series) pairs.
- iteritems(): Iterate over the columns of a DataFrame as (Column, Series) pairs.

```
for row,rowseries in dtf1.iterrows():
    print("Row name:",row)
    print("Values:\n",rowseries)
```

```
Row name: Delhi
Values:
 population    7897667.00
avg income      78976.00
per capita          0.01
Name: Delhi, dtype: float64
Row name: Mumbai
Values:
 population    4.577637e+06
avg income    4.563700e+04
per capita    9.969554e-03
Name: Mumbai, dtype: float64
Row name: Dehradun
```

Iterating over a DataFrame

```
for col,colseries in dtf1.iteritems():
    print("Col name:",col)
    print("Values:\n",colseries)
```

```
Col name: population
Values:
 Delhi      7897667
Mumbai     4577637
Dehradun    6457324
Name: population, dtype: int64
Col name: avg income
Values:
 Delhi      78976
Mumbai     45637
Dehradun    67324
Name: avg income, dtype: int64
Col name: per capita
Values:
 Delhi      0.010000
Mumbai     0.009970
Dehradun    0.010426
Name: per capita, dtype: float64
```

Pandas Read CSV

- A simple way to store big data sets is to use CSV files (comma separated files).
- CSV files contains plain text and is a well know format that can be read by everyone including Pandas.

```
Duration,Pulse,Maxpulse,Calories
60,110,130,409.1
60,117,145,479.0
60,103,135,340.0
45,109,175,282.4
45,117,148,406.0
60,102,127,300.0
60,110,136,374.0
45,104,134,253.3
30,109,133,195.1
60,98,124,269.0
60,103,147,329.3
60,100,120,250.7
60,106,128,345.3
60,104,132,379.3
60,98,123,275.0
60,98,120,215.2
60,100,120,300.0
45,90,112,
60,103,123,323.0
45,97,125,243.0
60,108,131,364.2
45,100,119,282.0
60,130,101,300.0
45,105,132,246.0
60,102,126,334.5
60,100,120,250.0
60,92,118,241.0
60,103,132,
60,100,132,280.0
60,102,129,380.3
```

- Load the CSV into a DataFrame:
- import pandas as pd
  df = pd.read_csv('data.csv')
  print(df.to_string())

```
import pandas as pd

df = pd.read_csv('data.csv')

print(df.to_string())
```

```
    Duration  Pulse  Maxpulse  Calories
0         60    110       130     409.1
1         60    117       145     479.0
2         60    103       135     340.0
3         45    109       175     282.4
4         45    117       148     406.0
5         60    102       127     300.5
6         60    110       136     374.0
7         45    104       134     253.3
8         30    109       133     195.1
9         60     98       124     269.0
10        60    103       147     329.3
11        60    100       120     250.7
```

If you have a large DataFrame with many rows, Pandas will only return the first 5 rows, and the last 5 rows. Print the DataFrame without the to_string() method

```
import pandas as pd

df = pd.read_csv('data.csv')

print(df)
```

```
     Duration  Pulse  Maxpulse  Calories
0          60    110       130     409.1
1          60    117       145     479.0
2          60    103       135     340.0
3          45    109       175     282.4
4          45    117       148     406.0
..        ...    ...       ...       ...
164        60    105       140     290.8
165        60    110       145     300.4
166        60    115       145     310.2
167        75    120       150     320.4
168        75    125       150     330.4

[169 rows x 4 columns]
```