Data Analysis in Python using Pandas

Pandas

- Pyton Data analysis library
- Built on top of Numpy
- Abbreviation of **Pan**el **Da**ta **S**ystem
- Used in production in many companies

The Ideal tool for data Scientists

- Managing data
- Cleaning data
- Analyzing
- Modeling data
- Organizing the data in a form suitable for plotting or tabular display

Introduction to pandas Data Structures

There are two data structure in pandas

--Series
--Dataframe

Series

A Series is a one-dimensional array-like object containing an array of data (of any NumPy data type) and an associated array of data labels, called its *index*. The simplest Series is formed from only an array of data:

```
In [4]: obj = Series([4, 7, -5, 3])

In [5]: obj
Out[5]:
0    4
1    7
2   -5
3    3
```

Viewing the values of the variable [object]

```
In [6]: obj.values
Out[6]: array([ 4,  7, -5,  3])

In [7]: obj.index
Out[7]: Int64Index([0, 1, 2, 3])
```

Adding index to the series

Often it will be desirable to create a Series with an index identifying each data point:

```
In [8]: obj2 = Series([4, 7, -5, 3], index=['d', 'b', 'a', 'c'])

In [9]: obj2
Out[9]:
d    4
b    7
a   -5
c    3
```

Convert Python Dict to Pandas Series

Should you have data contained in a Python dict, you can create a Series from it by passing the dict:

```
In [20]: sdata = {'Ohio': 35000, 'Texas': 71000, 'Oregon': 16000, 'Utah': 5000}

In [21]: obj3 = Series(sdata)

In [22]: obj3
Out[22]:
Ohio      35000
Oregon    16000
```

Pandas Series Examples

```
# import pandas as pd
import pandas as pd

# simple array
data = [1, 2, 3, 4]

ser = pd.Series(data)
print(ser)
```

Output

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

The axis labels are collectively called *index*. Pandas Series is nothing but a column in an

Creating a series from array: In order to create a series from array, we have to import a numpy module and have to use array() function.

```
# import pandas as pd
import pandas as pd

# import numpy as np
import numpy as np

# simple array
data = np.array(['g','e','e','k','s'])

ser = pd.Series(data)
print(ser)
```

Output

```
0    g
1    e
2    e
3    k
4    s
dtype: object
```

Creating a series from Lists:

In order to create a series from list, we have to first create a list after that we can create a series from list.

```
import pandas as pd # a simple list
 List= ['g', 'e', 'e', 'k', 's']
# create series form a list
ser = pd.Series(List)
print(ser)
```

Accessing first 5 elements of Series.

```
# import pandas and numpy
import pandas as pd
import numpy as np

# creating simple array
data = np.array(['g','e','e','k','s','f', 'o','r','g','e','e','k','s'])
ser = pd.Series(data)


#retrieve the first element
print(ser[:5])
```

Accessing Element Using Label (index) :

```
# import pandas and numpy
import pandas as pd
import numpy as np

# creating simple array
data = np.array(['g','e','e','k','s','f', 'o','r','g','e','e','k','s'])
ser =
pd.Series(data,index=[10,11,12,13,14,15,16,17,18,19,2
0,21,22])


# accessing a element using index element
print(ser[16])
```

Indexing a Series using indexing operator [] :

```
# importing pandas module
import pandas as pd

# making data frame
df = pd.read_csv("nba.csv")

ser = pd.Series(df['Name'])
data = ser.head(10)
data
```

```
0    Avery Bradley
1      Jae Crowder
2     John Holland
3      R.J. Hunter
4    Jonas Jerebko
5     Amir Johnson
6    Jordan Mickey
7     Kelly Olynyk
8     Terry Rozier
9     Marcus Smart
Name: Name, dtype: object
```

Now we access the element of series using index operator [ ].

```
# using indexing operator
data[3:6]
```

```
3      R.J. Hunter
4    Jonas Jerebko
5     Amir Johnson
Name: Name, dtype: object
```

Changing Series' Index

A Series's index can be altered in place by assignment:

```
In [35]: obj.index = ['Bob', 'Steve', 'Jeff', 'Ryan']

In [36]: obj
Out[36]:
Bob      4
Steve    7
Jeff    -5
Ryan     3
```

DataFrame

- Python DataFrame is a data structure containing and ordered collecetions of columns.

- Each column may hold numeric, string, boolean etc. Values

- DataFrame has both row and column index

Creating a DataFrame

- A pandas DataFrame can be created using various inputs like
  - --Lists
  - --Dict
  - --Series
  - --Numpy ndarrays
  - --Another DataFrame

Create an Empty DataFrame

A basic DataFrame, which can be created is an Empty Dataframe.

Example

```
#import the pandas library and aliasing as pd
import pandas as pd
df = pd.DataFrame()
print df
```

Try it

Its **output** is as follows –

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

Create a DataFrame from Lists

The DataFrame can be created using a single list or a list of lists.

Example 1

```
import pandas as pd
data = [1,2,3,4,5]
df = pd.DataFrame(data)
print df
```

Its **output** is as follows –

```
   0
0  1
1  2
2  3
3  4
4  5
```

Example 2

```
import pandas as pd
data = [['Alex',10],['Bob',12],['Clarke',13]]
df = pd.DataFrame(data,columns=['Name','Age'])
print df
```

Its **output** is as follows –

```
      Name      Age
0     Alex      10
1     Bob       12
2     Clarke    13
```

Example 2

```
import pandas as pd
data = [['Alex',10],['Bob',12],['Clarke',13]]
df = pd.DataFrame(data,columns=['Name','Age'])
print df
```

Its **output** is as follows –

```
      Name      Age
0     Alex      10
1     Bob       12
2     Clarke    13
```

Example 3

```
import pandas as pd
data = [['Alex',10],['Bob',12],['Clarke',13]]
df = pd.DataFrame(data,columns=['Name','Age'],dtype=float)
print df
```

Its **output** is as follows –

```
      Name     Age
0     Alex     10.0
1     Bob      12.0
2     Clarke   13.0
```

Note – Observe, the **dtype** parameter changes the type of Age column to floating point.

Create a DataFrame from Dict of ndarrays / Lists

All the **ndarrays** must be of same length. If index is passed, then the length of the index should equal to the length of the arrays.

If no index is passed, then by default, index will be range(n), where **n** is the array length.

Example 1

```
import pandas as pd
data = {'Name':['Tom', 'Jack', 'Steve', 'Ricky'],'Age':[28,34,29,42]}
df = pd.DataFrame(data)
print df
```

Its **output** is as follows –

```
      Age      Name
0     28        Tom
1     34       Jack
2     29      Steve
3     42      Ricky
```

Note – Observe the values 0,1,2,3. They are the default index assigned to each using the function range(n).

Example 2

Let us now create an indexed DataFrame using arrays.

```
import pandas as pd
data = {'Name':['Tom', 'Jack', 'Steve', 'Ricky'],'Age':[28,34,29,42]}
df = pd.DataFrame(data, index=['rank1','rank2','rank3','rank4'])
print df
```

Its **output** is as follows –

```
        Age    Name
rank1    28     Tom
rank2    34    Jack
rank3    29   Steve
rank4    42   Ricky
```

Note – Observe, the **index** parameter assigns an index to each row.

Create a DataFrame from List of Dicts

List of Dictionaries can be passed as input data to create a DataFrame. The dictionary keys are by default taken as column names.

Example 1

The following example shows how to create a DataFrame by passing a list of dictionaries.

```
import pandas as pd
data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]
df = pd.DataFrame(data)
print df
```

Its **output** is as follows –

```
   a   b     c
0  1   2   NaN
1  5   10  20.0
```

Note – Observe, NaN (Not a Number) is appended in missing areas.

Create a DataFrame from Dict of Series

Dictionary of Series can be passed to form a DataFrame. The resultant index is the union of all the series indexes passed.

Example

```
import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
      'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(d)
print df
```

Its **output** is as follows −

```
      one    two
a     1.0    1
b     2.0    2
c     3.0    3
d     NaN    4
```

Note − Observe, for the series one, there is no label **'d'** passed, but in the result, for the **d** label, NaN is appended with NaN.

Column Selection

We will understand this by selecting a column from the DataFrame.

Example

```
import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
      'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(d)
print df ['one']
```

Its **output** is as follows —

```
a     1.0
b     2.0
c     3.0
d     NaN
Name: one, dtype: float64
```

Column Addition

We will understand this by adding a new column to an existing data frame.

Example

In [ ]:

```
import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
     'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(d)

# Adding a new column to an existing DataFrame object with column label by passing new series

print ("Adding a new column by passing as Series:")
df['three']=pd.Series([10,20,30],index=['a','b','c'])
print df

print ("Adding a new column using the existing columns in DataFrame:")
df['four']=df['one']+df['three']

print df
```

Its **output** is as follows —

```
Adding a new column by passing as Series:
      one   two   three
a     1.0     1    10.0
b     2.0     2    20.0
c     3.0     3    30.0
d     NaN     4     NaN

Adding a new column using the existing columns in DataFrame:
       one   two   three    four
a      1.0     1    10.0    11.0
b      2.0     2    20.0    22.0
c      3.0     3    30.0    33.0
d      NaN     4     NaN     NaN
```

Column Deletion

Columns can be deleted or popped; let us take an example to understand how.

Example

```
# Using the previous DataFrame, we will delete a column
# using del function
import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
     'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd']),
     'three' : pd.Series([10,20,30], index=['a','b','c'])}

df = pd.DataFrame(d)
print ("Our dataframe is:")
print df

# using del function
print ("Deleting the first column using DEL function:")
del df['one']
print df

# using pop function
print ("Deleting another column using POP function:")
df.pop('two')
print df
```

Addition of Rows

Add new rows to a DataFrame using the **append** function. This function will append the rows at the end.

```
import pandas as pd

df = pd.DataFrame([[1, 2], [3, 4]], columns = ['a','b'])
df2 = pd.DataFrame([[5, 6], [7, 8]], columns = ['a','b'])

df = df.append(df2)
print df
```

Its **output** is as follows –

```
   a  b
0  1  2
1  3  4
0  5  6
1  7  8
```

Slice Rows

Multiple rows can be selected using ` : ’ operator.

```
import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
    'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(d)
print df[2:4]
```

Its **output** is as follows –

```
      one    two
c     3.0     3
d     NaN     4
```

Python Pandas Input/Output TOOLS

- The **Pandas I/O API** is a set of top level reader functions accessed like **pd.read_csv()** that generally return a Pandas object.

- The two functions for reading text files are **read_csv()** and **read_table()**. They both intelligently convert tabular data into a **DataFrame** object

```
pandas.read_csv(filepath_or_buffer, sep=',', delimiter=None, header='infer',
names=None, index_col=None, usecols=None
```

```
pandas.read_csv(filepath_or_buffer, sep='\t', delimiter=None, header='infer',
names=None, index_col=None, usecols=None
```

Here is how the **csv** file data looks like –

```
S.No,Name,Age,City,Salary
1,Tom,28,Toronto,20000
2,Lee,32,HongKong,3000
3,Steven,43,Bay Area,8300
4,Ram,38,Hyderabad,3900
```

Save this data as **temp.csv** and conduct operations on it.


```
S.No,Name,Age,City,Salary
1,Tom,28,Toronto,20000
2,Lee,32,HongKong,3000
3,Steven,43,Bay Area,8300
4,Ram,38,Hyderabad,3900
```

Save this data as **temp.csv** and conduct operations on it.

read.csv

read.csv reads data from the csv files and creates a DataFrame object.

```
import pandas as pd
df=pd.read_csv("temp.csv")
print df
```

Its **output** is as follows –

```
   S.No     Name   Age       City   Salary
0     1      Tom    28    Toronto    20000
1     2      Lee    32   HongKong     3000
2     3   Steven    43   Bay Area     8300
3     4      Ram    38  Hyderabad     3900
```

Python Pandas

Let us create a DataFrame and use this object throughout this chapter for all the operati

Example

```
import pandas as pd
import numpy as np

#Create a Dictionary of series
d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack',
   'Lee','David','Gasper','Betina','Andres']),
   'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])}

#Create a DataFrame
df = pd.DataFrame(d)
print df
```

Its **output** is as follows –

```
    Age  Name   Rating
0   25   Tom     4.23
1   26   James   3.24
2   25   Ricky   3.98
3   23   Vin     2.56
4   30   Steve   3.20
5   29   Smith   4.60
6   23   Jack    3.80
7   34   Lee     3.78
8   40   David   2.98
9   30   Gasper  4.80
10  51   Betina  4.10
11  46   Andres  3.65
```

sum()

Returns the sum of the values for the requested axis. By default, axis is index (axis=0).

```
import pandas as pd
import numpy as np

#Create a Dictionary of series
d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack',
   'Lee','David','Gasper','Betina','Andres']),
   'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])}

#Create a DataFrame
df = pd.DataFrame(d)
print df.sum()
```

Its **output** is as follows –

```
Age                                               382
Name      TomJamesRickyVinSteveSmithJackLeeDavidGasperBe...
Rating                                          44.92
dtype: object
```

mean()

Returns the average value

```
import pandas as pd
import numpy as np

#Create a Dictionary of series
d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack',
   'Lee','David','Gasper','Betina','Andres']),
   'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])}

#Create a DataFrame
df = pd.DataFrame(d)
print df.mean()
```

Its **output** is as follows –

```
Age       31.833333
Rating     3.743333
dtype: float64
```

std()

Returns the Bressel standard deviation of the numerical columns.

```
import pandas as pd
import numpy as np

#Create a Dictionary of series
d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack',
   'Lee','David','Gasper','Betina','Andres']),
   'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])}

#Create a DataFrame
df = pd.DataFrame(d)
print df.std()
```

Its **output** is as follows –

```
Age       9.232682
Rating    0.661628
dtype: float64
```

Summarizing Data

The **describe()** function computes a summary of statistics pertaining to the DataFrame columns.

```
import pandas as pd
import numpy as np

#Create a Dictionary of series
d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack',
   'Lee','David','Gasper','Betina','Andres']),
   'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])}

#Create a DataFrame
df = pd.DataFrame(d)
print df.describe()
```

Its **output** is as follows –

```
             Age        Rating
count   12.000000     12.000000
mean    31.833333      3.743333
std      9.232682      0.661628
min     23.000000      2.560000
25%     25.000000      3.230000
50%     29.500000      3.790000
75%     35.500000      4.132500
max     51.000000      4.800000
```

This function gives the **mean, std** and **IQR** values. And, function excludes the character columns and given summary about numeric columns. **'include'** is the argument which is used to pass necessary information regarding what columns need to be considered for summarizing. Takes the list of values; by default, 'number'.

Python Pandas

Concatenation

The **concat** function does all of the heavy lifting of performing concatenation operations along an axis. Let us create different objects and do concatenation.

```
import pandas as pd
one = pd.DataFrame({
         'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
         'subject_id':['sub1','sub2','sub4','sub6','sub5'],
         'Marks_scored':[98,90,87,69,78]},
         index=[1,2,3,4,5])
two = pd.DataFrame({
         'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
         'subject_id':['sub2','sub4','sub3','sub6','sub5'],
         'Marks_scored':[89,80,79,97,88]},
         index=[1,2,3,4,5])
print pd.concat([one,two])
```

Try it

Its **output** is as follows –

	Marks_scored	Name	subject_id
1	98	Alex	sub1
2	90	Amy	sub2
3	87	Allen	sub4
4	69	Alice	sub6
5	78	Ayoung	sub5
1	89	Billy	sub2
2	80	Brian	sub4
3	79	Bran	sub3

- **unique():** This method is used to get all unique values from the given column.

Syntax:

```
dataframe['column_name].unique()
```

- **nunique():** This method is similar to unique but it will return the count the unique values.

Syntax:

```
dataframe_name['column_name].nunique()
```

- **info():** This command is used to get the data types and columns information

Syntax:

```
dataframe.info()
```

- **columns:** This command is used to display all the column names present in data frame

Syntax:

```
dataframe.columns
```