Unit 5-Python Notes numpy and pandas

Programming For Problem Solving (SRM Institute of Science and Technology)

NumPy Creating Arrays

NumPy is used to work with arrays. The array object in NumPy is called `ndarray`.

We can create a NumPy `ndarray` object by using the `array()` function.

Example

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)

print(type(arr))
```

```
[1 2 3 4 5]
<class 'numpy.ndarray'>
```

To create an `ndarray`, we can pass a list, tuple or any array-like object into the `array()` method, and it will be converted into an `ndarray`:

Example

Use a tuple to create a NumPy array:

```
import numpy as np

arr = np.array((1, 2, 3, 4, 5))

print(arr)
```

```
[1 2 3 4 5]
```

0-D Arrays

0-D arrays, or Scalars, are the elements in an array. Each value in an array is a 0-D array.

Example

Create a 0-D array with value 42`import numpy as np`

```
arr = np.array(42)

print(arr)
```

```
42
```

1-D Arrays

An array that has 0-D arrays as its elements is called uni-dimensional or 1-D array.

These are the most common and basic arrays.

Example

Create a 1-D array containing the values 1,2,3,4,5:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)
```

```
[1 2 3 4 5]
```

2-D Arrays

An array that has 1-D arrays as its elements is called a 2-D array.

These are often used to represent matrix or 2nd order tensors.

NumPy has a whole sub module dedicated towards matrix operations called `numpy.mat`

Example

Create a 2-D array containing two arrays with the values 1,2,3 and 4,5,6:

```
import numpy as np
```

```
arr = np.array([[1, 2, 3], [4, 5, 6]])

print(arr)
```

```
[[1 2 3]
 [4 5 6]]
```

Check Number of Dimensions?

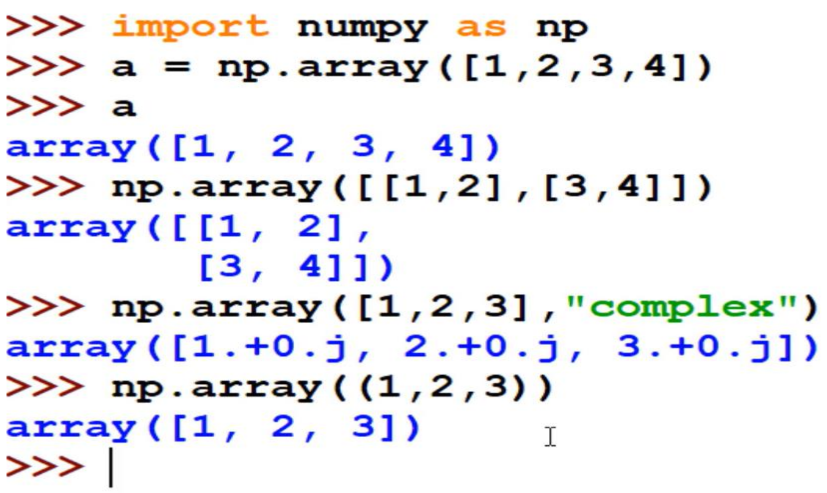
NumPy Arrays provides the `ndim` attribute that returns an integer that tells us how many dimensions the array have.

Example

Check how many dimensions the arrays have:

```
import numpy as np

a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

```
0
1
2
3
```



```
>>> import numpy as np
>>> a = np.array([1,2,3,4])
>>> a
array([1, 2, 3, 4])
>>> np.array([[1,2],[3,4]])
array([[1, 2],
       [3, 4]])
>>> np.array([1,2,3],"complex")
array([1.+0.j, 2.+0.j, 3.+0.j])
>>> np.array((1,2,3))
array([1, 2, 3])
>>> 
```

NumPy Array Indexing

Access Array Elements

Array indexing is the same as accessing an array element.

You can access an array element by referring to its index number.

The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.

Example

Get the first element from the following array:

```
import numpy as np

arr = np.array([1, 2, 3, 4])

print(arr[0])
```

```
1
```

Access 2-D Arrays

To access elements from 2-D arrays we can use comma separated integers representing the dimension and the index of the element.

Think of 2-D arrays like a table with rows and columns, where the row represents the dimension and the index represents the column.

Example

Access the element on the first row, second column:

```
import numpy as np

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('2nd element on 1st row: ', arr[0, 1])
```

```
2nd element on 1st dim:  2
```

Slicing arrays

Slicing in python means taking elements from one given index to another given index.

We pass slice instead of index like this: `[start:end]`.

We can also define the step, like this: `[start:end:step]`.

If we don't pass start its considered 0

If we don't pass end its considered length of array in that dimension

If we don't pass step its considered 1

Example

Slice elements from index 1 to index 5 from the following array:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[1:5])
```

```
[2 3 4 5]
```

Example

Slice elements from index 4 to the end of the array:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[4:])
```

```
[5 6 7]
```

Example

Slice elements from the beginning to index 4 (not included):

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[:4])
```

```
[1 2 3 4]
```

STEP

Use the `step` value to determine the step of the slicing:

Example

Return every other element from index 1 to index 5:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[1:5:2])
```

```
[2 4]
```

Percentiles

25th Percentile - Also known as the first, or lower, quartile. The 25th percentile is the value at which 25% of the answers lie below that value, and 75% of the answers lie above that value.

50th Percentile - Also known as the Median. The median cuts the data set in half. Half of the answers lie below the median and half lie above the median.

75th Percentile - Also known as the third, or upper, quartile. The 75th percentile is the value at which 25% of the answers lie above that value and 75% of the answers lie below that value.

numpy.percentile() in python

numpy.percentile()function used to compute the nth percentile of the given data (array elements) along the specified axis.

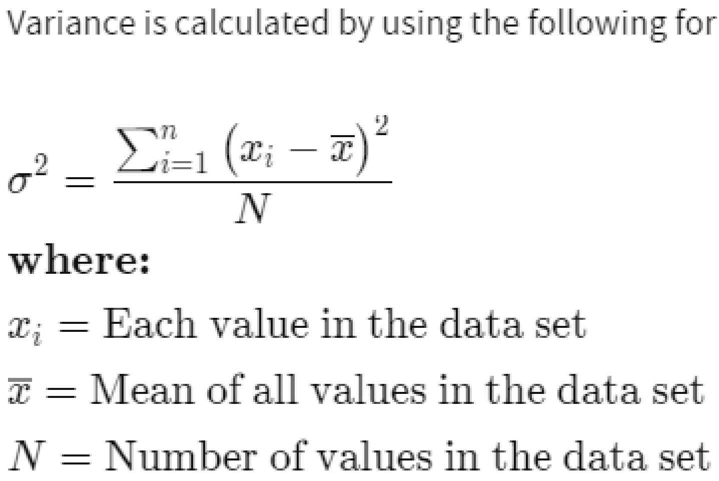
Syntax : *numpy.percentile(arr, n, axis=None, out=None)*
Parameters :
arr :*input array.*
n : *percentile value.*
axis : *axis along which we want to calculate the percentile value. Otherwise, it will consider arr to be flattened(works on all the axis). axis = 0 means along the column and axis = 1 means working along the row.*
out :*Different array in which we want to place the result. The array must have same dimensions as expected output.*
Return :*nth Percentile of the array (a scalar value if axis is none)or array with percentile values along specified axis.*

```
# Python Program illustrating
# numpy.percentile() method

import numpy as np

# 1D array
arr = [20, 2, 7, 1, 34]
print("arr : ", arr)
print("50th percentile of arr : ",
       np.percentile(arr, 50))
print("25th percentile of arr : ",
       np.percentile(arr, 25))
print("75th percentile of arr : ",
       np.percentile(arr, 75))
```

Output :

```
arr :  [20, 2, 7, 1, 34]
50th percentile of arr :  7.0
25th percentile of arr :  2.0
75th percentile of arr :  20.0
```

numpy.var() in Python



Variance is calculated by using the following formula:

$$\sigma^2 = \frac{\sum_{i=1}^{n} \left(x_i - \overline{x}\right)^2}{N}$$

where:

x_i = Each value in the data set

$\overline{x}$ = Mean of all values in the data set

N = Number of values in the data set

`numpy.var(arr, axis = None)` : Compute the variance of the given data (array elements) along the specified axis(if any).

```
# Python Program illustrating
# numpy.var() method
import numpy as np

# 1D array
arr = [20, 2, 7, 1, 34]

print("arr : ", arr)
print("var of arr : ", np.var(arr))
```

Output :

```
arr :  [20, 2, 7, 1, 34]

var of arr :  158.16
```

Pandas

Pandas is an open-source library that is built on top of NumPy library. It is a Python package that offers various data structures and operations for manipulating numerical data and time series. It is mainly popular for importing and analyzing data much easier. Pandas is fast and it has high-performance & productivity for users.

Creating a Pandas Series

Pandas Series is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.). The axis labels are collectively called *index*. 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.

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

# Creating empty series
ser = pd.Series()

print(ser)
```

Output :

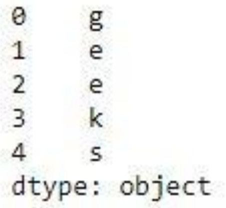
```
Series([], dtype: float64)
```

By default, the data type of Series is float.

Creating a series from array: In order to create a series from NumPy array, we have to import 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:



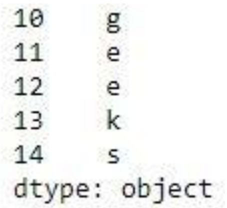
```
0    g
1    e
2    e
3    k
4    s
dtype: object
```

By default, the index of the series starts from 0 till the length of series -1.

Creating a series from array with an index: In order to create a series by explicitly proving index instead of the default, we have to provide a list of elements to the index parameter with the same number of elements as it is an array.

```
# 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'])

# providing an index
ser = pd.Series(data, index=[10, 11, 12, 13, 14])
print(ser)
```

Output:

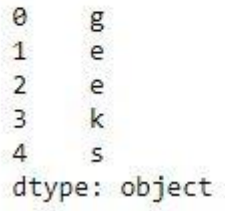


```
10    g
11    e
12    e
13    k
14    s
dtype: object
```

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)
```

Output :

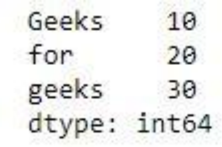


```
0    g
1    e
2    e
3    k
4    s
dtype: object
```

Creating a series from Dictionary: In order to create a series from the dictionary, we have to first create a dictionary after that we can make a series using dictionary. Dictionary keys are used to construct indexes of Series.

```
import pandas as pd

# a simple dictionary
dict = {'Geeks': 10,
        'for': 20,
        'geeks': 30}

# create series from dictionary
ser = pd.Series(dict)
print(ser)
```

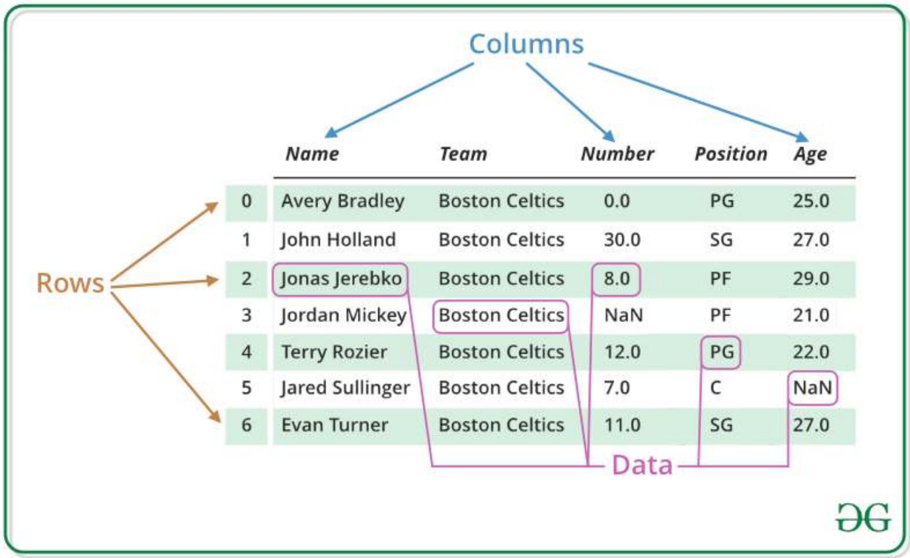
Output:



```
Geeks    10
for      20
geeks    30
dtype: int64
```

Pandas DataFrame

Pandas DataFrame is two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. Pandas DataFrame consists of three principal components, the **data**, **rows**, and **columns**.



	Name	Team	Number	Position	Age
0	Avery Bradley	Boston Celtics	0.0	PG	25.0
1	John Holland	Boston Celtics	30.0	SG	27.0
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0
4	Terry Rozier	Boston Celtics	12.0	PG	22.0
5	Jared Sullinger	Boston Celtics	7.0	C	NaN
6	Evan Turner	Boston Celtics	11.0	SG	27.0

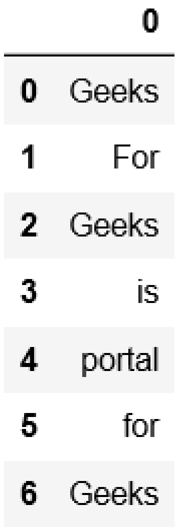
Creating a Pandas DataFrame

In the real world, a Pandas DataFrame will be created by loading the datasets from existing storage, storage can be SQL Database, CSV file, and Excel file. Pandas DataFrame can be created from the lists, dictionary, and from a list of dictionary etc. Dataframe can be created in different ways here are some ways by which we create a dataframe:

Creating a dataframe using List: DataFrame can be created using a single list or a list of lists.

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

# list of strings
lst = ['Geeks', 'For', 'Geeks', 'is',
            'portal', 'for', 'Geeks']

# Calling DataFrame constructor on list
df = pd.DataFrame(lst)
print(df)
```

Output:

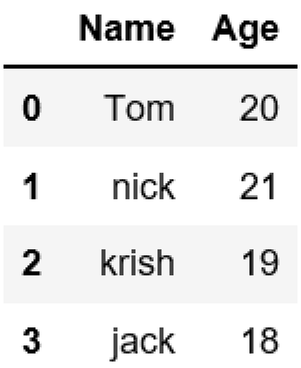


	0
0	Geeks
1	For
2	Geeks
3	is
4	portal
5	for
6	Geeks

Creating DataFrame from dict of ndarray/lists: To create DataFrame from dict of narray/list, all the narray must be of same length. If index is passed then the length index should be equal to the length of arrays. If no index is passed, then by default, index will be range(n) where n is the array length.

```
# Python code demonstrate creating

# DataFrame from dict narray / lists

# By default addresses.


import pandas as pd


# intialise data of lists.

data = {'Name':['Tom', 'nick', 'krish', 'jack'],

        'Age':[20, 21, 19, 18]}


# Create DataFrame

df = pd.DataFrame(data)


# Print the output.

print(df)
```

Output:



	Name	Age
0	Tom	20
1	nick	21
2	krish	19
3	jack	18

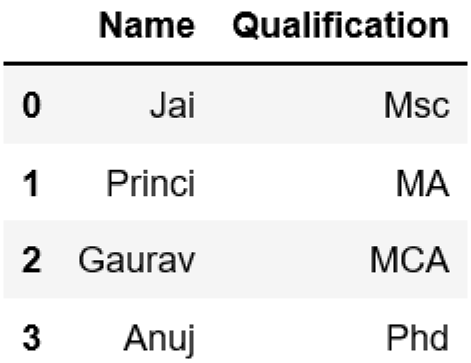
Dealing with Rows and Columns

A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. We can perform basic operations on rows/columns like selecting, deleting, adding, and renaming.

Column Selection: In Order to select a column in Pandas DataFrame, we can access the columns by calling them by their columns name.

```
# Import pandas package

import pandas as pd


# Define a dictionary containing employee data

data = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],

        'Age':[27, 24, 22, 32],

        'Address':['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],

        'Qualification':['Msc', 'MA', 'MCA', 'Phd']}


# Convert the dictionary into DataFrame

df = pd.DataFrame(data)


# select two columns

print(df[['Name', 'Qualification']])
```

Output:



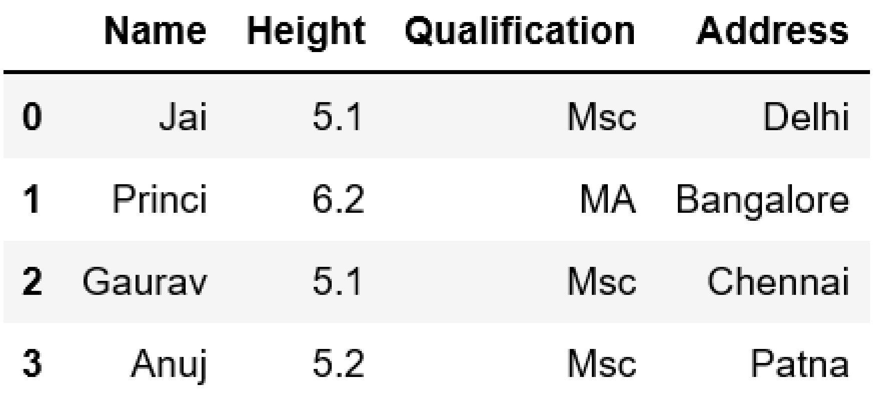
	Name	Qualification
0	Jai	Msc
1	Princi	MA
2	Gaurav	MCA
3	Anuj	Phd

Column Addition:

In Order to add a column in Pandas DataFrame, we can declare a new list as a column and add to a existing Dataframe.

```
# Import pandas package

import pandas as pd



# Define a dictionary containing Students data

data = {'Name': ['Jai', 'Princi', 'Gaurav', 'Anuj'],

        'Height': [5.1, 6.2, 5.1, 5.2],

        'Qualification': ['Msc', 'MA', 'Msc', 'Msc']}



# Convert the dictionary into DataFrame

df = pd.DataFrame(data)



# Declare a list that is to be converted into a column

address = ['Delhi', 'Bangalore', 'Chennai', 'Patna']



# Using 'Address' as the column name

# and equating it to the list

df['Address'] = address



# Observe the result

print(df)
```

Output:



	Name	Height	Qualification	Address
0	Jai	5.1	Msc	Delhi
1	Princi	6.2	MA	Bangalore
2	Gaurav	5.1	Msc	Chennai
3	Anuj	5.2	Msc	Patna

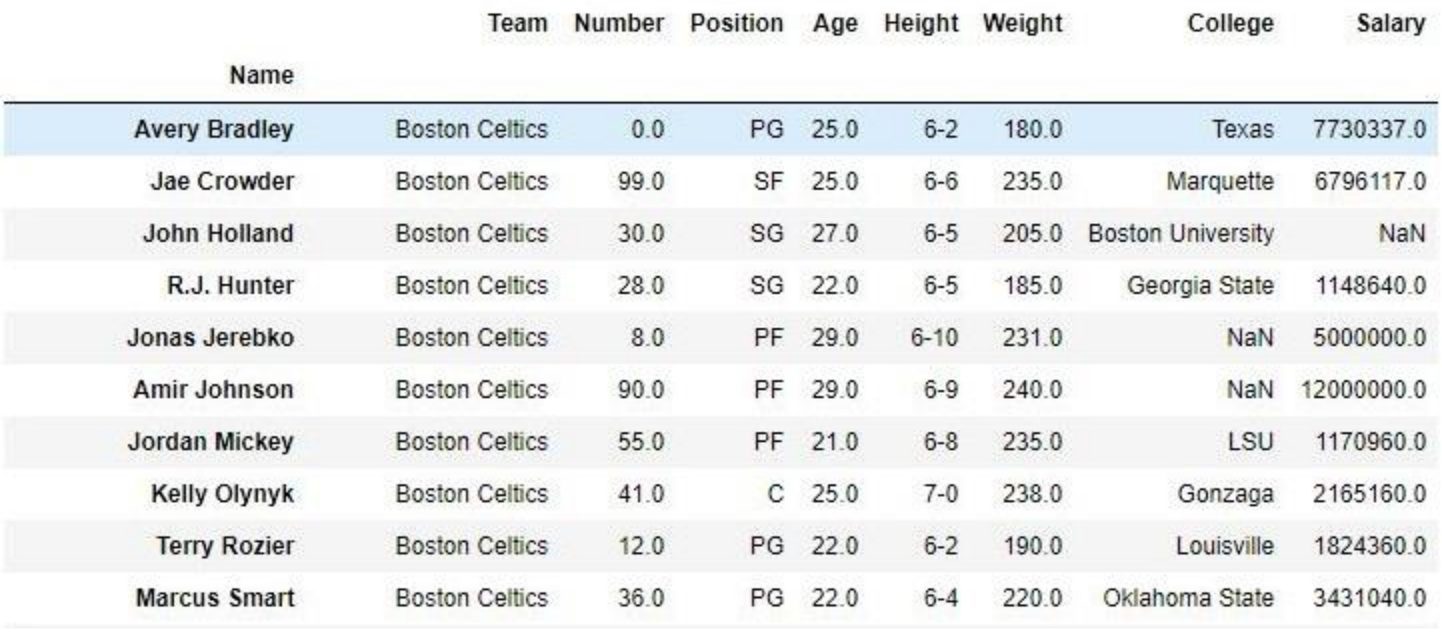
Column Deletion:
In Order to delete a column in Pandas DataFrame, we can use the `drop()` method. Columns is deleted by dropping columns with column names.

```
# importing pandas module
import pandas as pd

# making data frame from csv file
data = pd.read_csv("nba.csv", index_col ="Name" )

# dropping passed columns
data.drop(["Team", "Weight"], axis = 1, inplace = True)

# display
print(data)
```

Output:

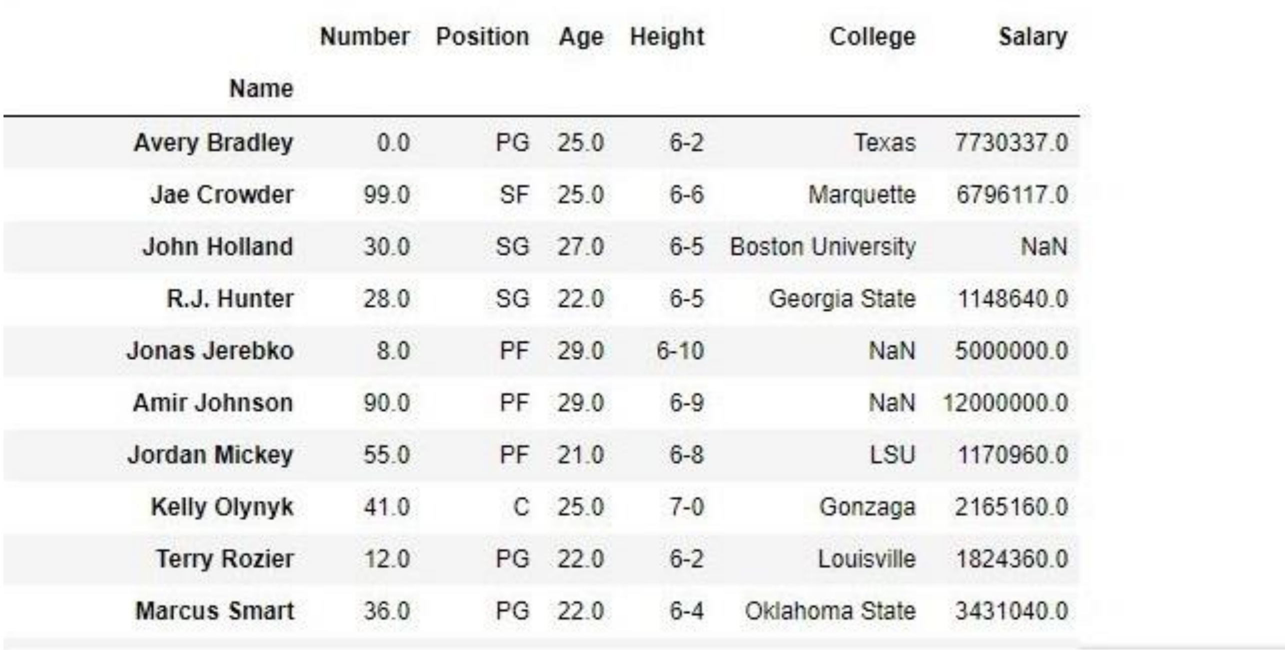
As shown in the output images, the new output doesn't have the passed columns. Those values were dropped since axis was set equal to 1 and the changes were made in the original data frame since inplace was True.

Data Frame before Dropping Columns-



	Team	Number	Position	Age	Height	Weight	College	Salary
Name								
Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN
R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0
Amir Johnson	Boston Celtics	90.0	PF	29.0	6-9	240.0	NaN	12000000.0
Jordan Mickey	Boston Celtics	55.0	PF	21.0	6-8	235.0	LSU	1170960.0
Kelly Olynyk	Boston Celtics	41.0	C	25.0	7-0	238.0	Gonzaga	2165160.0
Terry Rozier	Boston Celtics	12.0	PG	22.0	6-2	190.0	Louisville	1824360.0
Marcus Smart	Boston Celtics	36.0	PG	22.0	6-4	220.0	Oklahoma State	3431040.0

Data Frame after Dropping Columns-



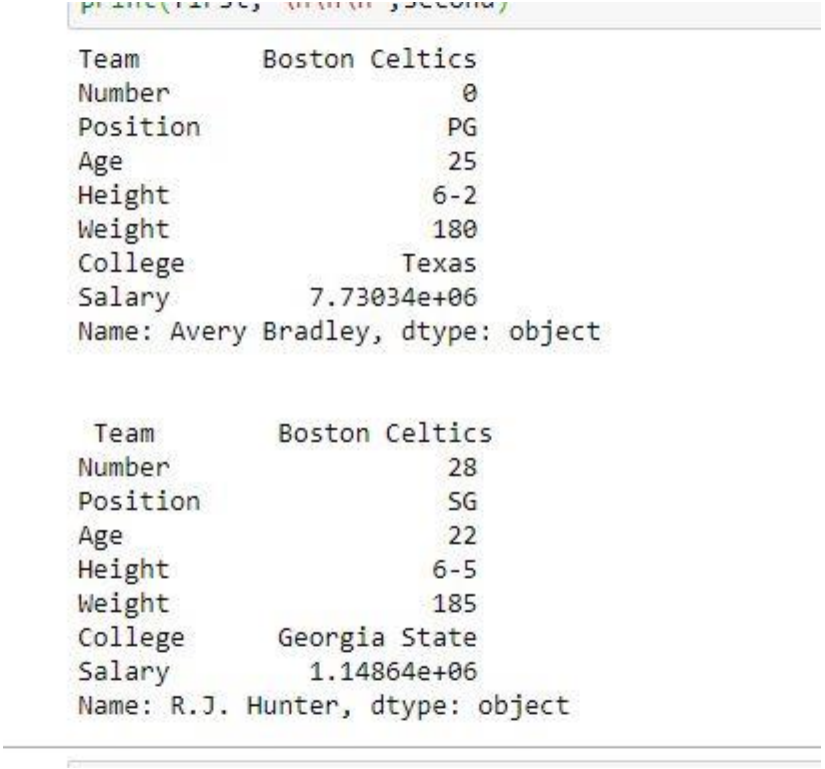
Name	Number	Position	Age	Height	College	Salary
Avery Bradley	0.0	PG	25.0	6-2	Texas	7730337.0
Jae Crowder	99.0	SF	25.0	6-6	Marquette	6796117.0
John Holland	30.0	SG	27.0	6-5	Boston University	NaN
R.J. Hunter	28.0	SG	22.0	6-5	Georgia State	1148640.0
Jonas Jerebko	8.0	PF	29.0	6-10	NaN	5000000.0
Amir Johnson	90.0	PF	29.0	6-9	NaN	12000000.0
Jordan Mickey	55.0	PF	21.0	6-8	LSU	1170960.0
Kelly Olynyk	41.0	C	25.0	7-0	Gonzaga	2165160.0
Terry Rozier	12.0	PG	22.0	6-2	Louisville	1824360.0
Marcus Smart	36.0	PG	22.0	6-4	Oklahoma State	3431040.0

Row Selection: Pandas provide a unique method to retrieve rows from a Data frame. `DataFrame.loc[]` method is used to retrieve rows from Pandas DataFrame. Rows can also be selected by passing integer location to an iloc[] function.

Note: We'll be using `nba.csv` file in below examples.

```
# importing pandas package

import pandas as pd


# making data frame from csv file

data = pd.read_csv("nba.csv", index_col ="Name")


# retrieving row by loc method

first = data.loc["Avery Bradley"]

second = data.loc["R.J. Hunter"]



print(first, "\n\n\n", second)
```

Output:
As shown in the output image, two series were returned since there was only one parameter both of the times.



```
print(first, "\n\n\n",second)

Team        Boston Celtics
Number                   0
Position                PG
Age                     25
Height                 6-2
Weight                 180
College              Texas
Salary         7.73034e+06
Name: Avery Bradley, dtype: object


 Team        Boston Celtics
Number                   28
Position                 SG
Age                      22
Height                  6-5
Weight                  185
College       Georgia State
Salary          1.14864e+06
Name: R.J. Hunter, dtype: object
```

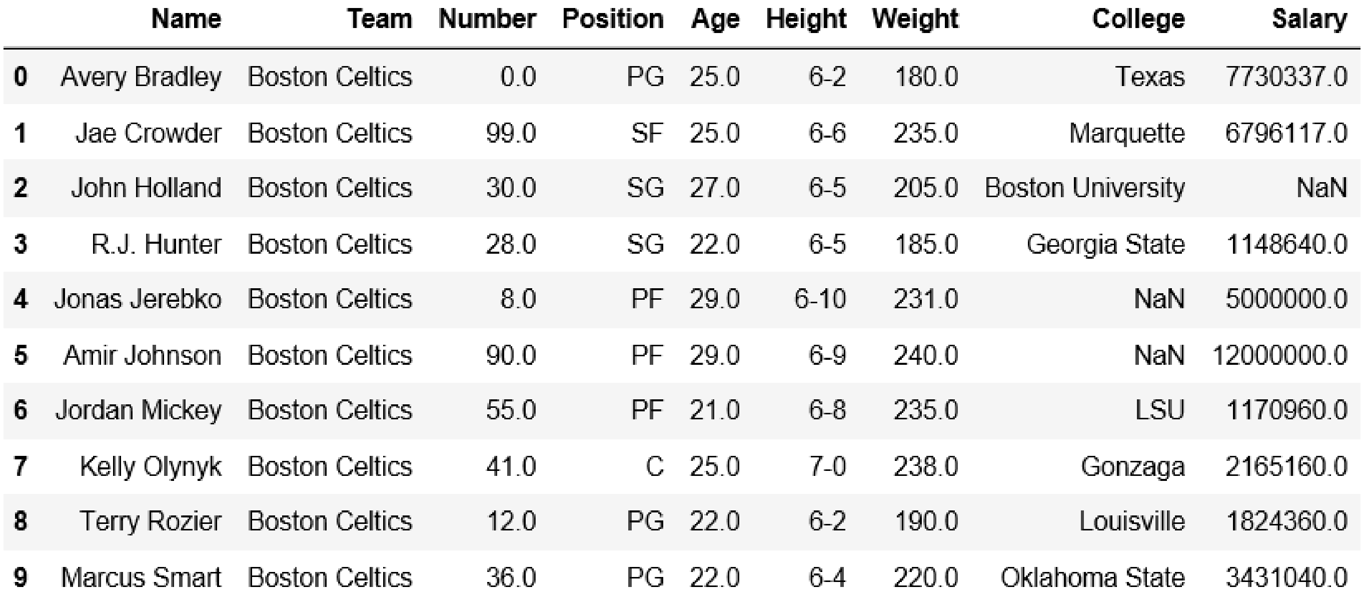
Row Addition:

In Order to add a Row in Pandas DataFrame, we can concat the old dataframe with new one.

```
# importing pandas module

import pandas as pd



# making data frame

df = pd.read_csv("nba.csv", index_col ="Name")



df.head(10)



new_row = pd.DataFrame({'Name':'Geeks', 'Team':'Boston', 'Number':3,

                        'Position':'PG', 'Age':33, 'Height':'6-2',
                                     'Weight':189, 'College':'MIT', 'Salary':99999},

                                                                             index =[0])

# simply concatenate both dataframes

df = pd.concat([new_row, df]).reset_index(drop = True)

df.head(5)
```

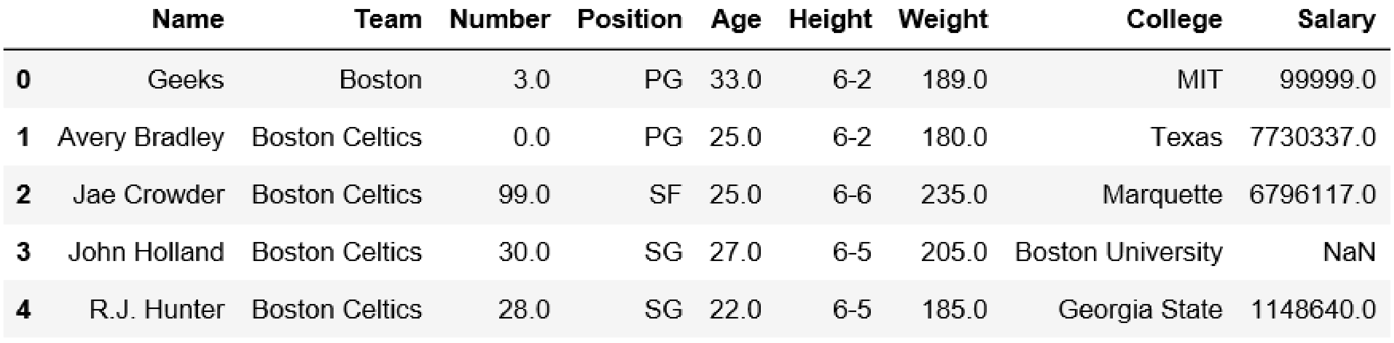
Output:

Data Frame before Adding Row-



	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
2	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN
3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
4	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0
5	Amir Johnson	Boston Celtics	90.0	PF	29.0	6-9	240.0	NaN	12000000.0
6	Jordan Mickey	Boston Celtics	55.0	PF	21.0	6-8	235.0	LSU	1170960.0
7	Kelly Olynyk	Boston Celtics	41.0	C	25.0	7-0	238.0	Gonzaga	2165160.0
8	Terry Rozier	Boston Celtics	12.0	PG	22.0	6-2	190.0	Louisville	1824360.0
9	Marcus Smart	Boston Celtics	36.0	PG	22.0	6-4	220.0	Oklahoma State	3431040.0

Data Frame after Adding Row-



	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Geeks	Boston	3.0	PG	33.0	6-2	189.0	MIT	99999.0
1	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
2	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
3	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN
4	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0

Row Deletion**:**

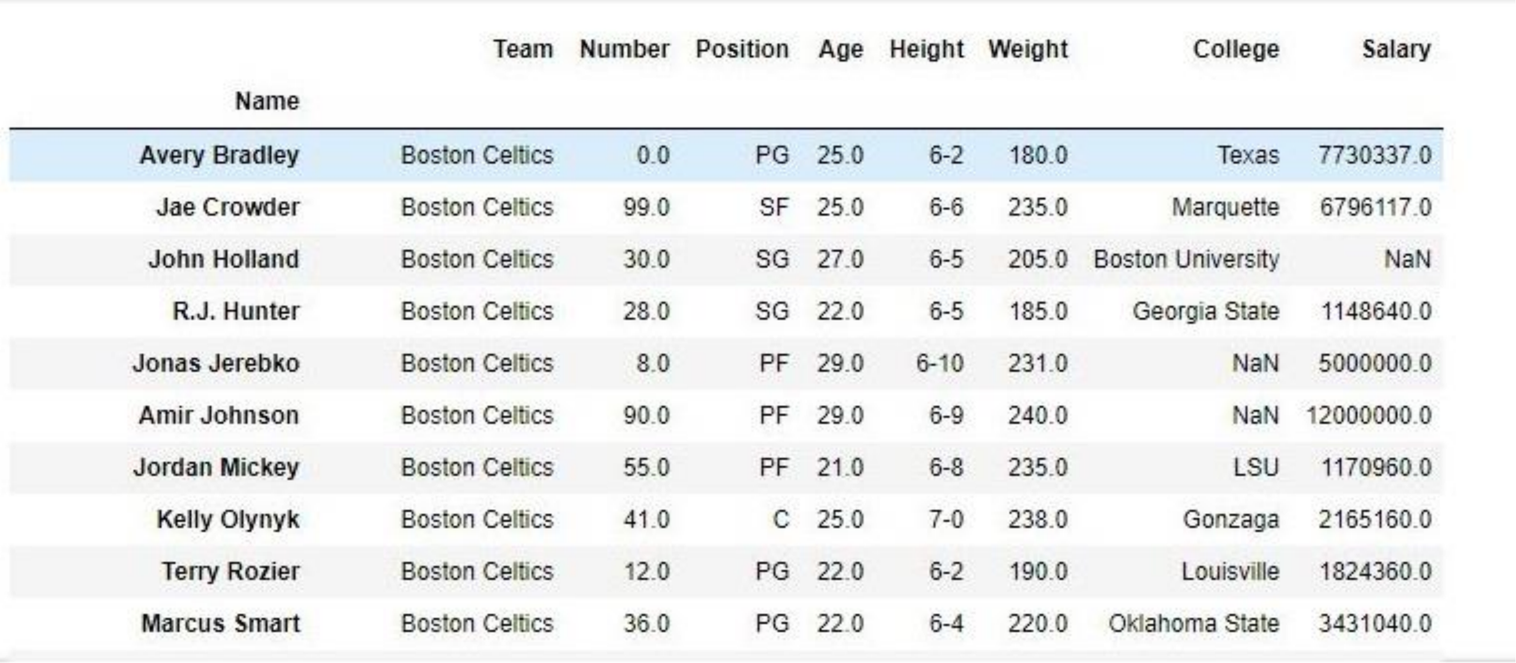
In Order to delete a row in Pandas DataFrame, we can use the drop() method. Rows is deleted by dropping Rows by index label.

```
# importing pandas module

import pandas as pd



# making data frame from csv file

data = pd.read_csv("nba.csv", index_col ="Name" )



# dropping passed values

data.drop(["Avery Bradley", "John Holland", "R.J. Hunter",

                            "R.J. Hunter"], inplace = True)



# display

data
```

Output:

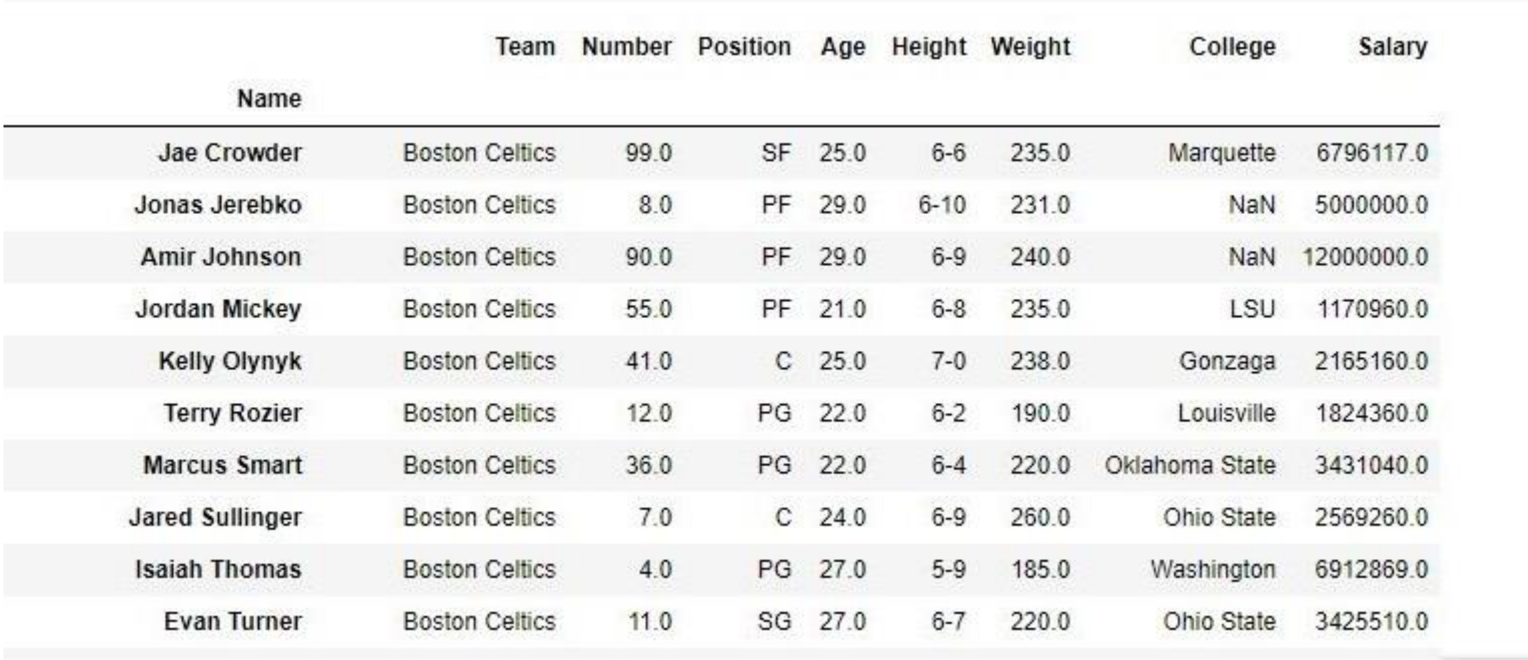
As shown in the output images, the new output doesn't have the passed values. Those values were dropped and the changes were made in the original data frame since inplace was True.

Data Frame before Dropping values-



	Team	Number	Position	Age	Height	Weight	College	Salary
Name								
Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN
R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0
Amir Johnson	Boston Celtics	90.0	PF	29.0	6-9	240.0	NaN	12000000.0
Jordan Mickey	Boston Celtics	55.0	PF	21.0	6-8	235.0	LSU	1170960.0
Kelly Olynyk	Boston Celtics	41.0	C	25.0	7-0	238.0	Gonzaga	2165160.0
Terry Rozier	Boston Celtics	12.0	PG	22.0	6-2	190.0	Louisville	1824360.0
Marcus Smart	Boston Celtics	36.0	PG	22.0	6-4	220.0	Oklahoma State	3431040.0

Data Frame after Dropping values-



	Team	Number	Position	Age	Height	Weight	College	Salary
Name								
Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0
Amir Johnson	Boston Celtics	90.0	PF	29.0	6-9	240.0	NaN	12000000.0
Jordan Mickey	Boston Celtics	55.0	PF	21.0	6-8	235.0	LSU	1170960.0
Kelly Olynyk	Boston Celtics	41.0	C	25.0	7-0	238.0	Gonzaga	2165160.0
Terry Rozier	Boston Celtics	12.0	PG	22.0	6-2	190.0	Louisville	1824360.0
Marcus Smart	Boston Celtics	36.0	PG	22.0	6-4	220.0	Oklahoma State	3431040.0
Jared Sullinger	Boston Celtics	7.0	C	24.0	6-9	260.0	Ohio State	2569260.0
Isaiah Thomas	Boston Celtics	4.0	PG	27.0	5-9	185.0	Washington	6912869.0
Evan Turner	Boston Celtics	11.0	SG	27.0	6-7	220.0	Ohio State	3425510.0

Pandas.apply()

Pandas.apply allow the users to pass a function and apply it on every single value of the Pandas series. It comes as a huge improvement for the pandas library as this function helps to segregate data according to the conditions required due to which it is efficiently used in data science and machine learning.

To read the csv file and squeezing it into a pandas series following commands are used:

```
import pandas as pd
```

```
s = pd.read_csv("stock.csv", squeeze=True)
```

Syntax:
`s.apply(func, convert_dtype=True, args=())`

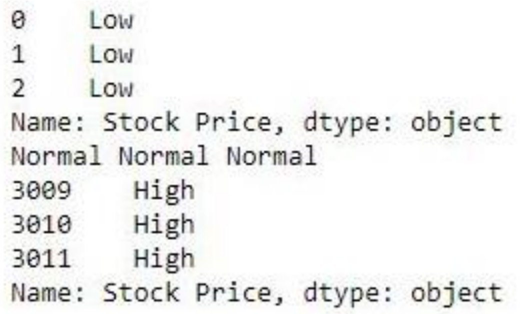
Parameters:
func: *.apply takes a function and applies it to all values of pandas series.*
convert_dtype: *Convert dtype as per the function's operation.*
args=(): *Additional arguments to pass to function instead of series.*
Return Type: *Pandas Series after applied function/operation.*

The following example passes a function and checks the value of each element in series and returns low, normal or High accordingly.

```
import pandas as pd



# reading csv

s = pd.read_csv("stock.csv", squeeze = True)



# defining function to check price

def fun(num):



    if num<200:

        return "Low"



    elif num>= 200 and num<400:

        return "Normal"



    else:

        return "High"
```

```
# passing function to apply and storing returned series in new

new = s.apply(fun)



# printing first 3 element

print(new.head(3))



# printing elements somewhere near the middle of series

print(new[1400], new[1500], new[1600])



# printing last 3 elements

print(new.tail(3))
```

Output:



```
0    Low
1    Low
2    Low
Name: Stock Price, dtype: object
Normal Normal Normal
3009    High
3010    High
3011    High
Name: Stock Price, dtype: object
```

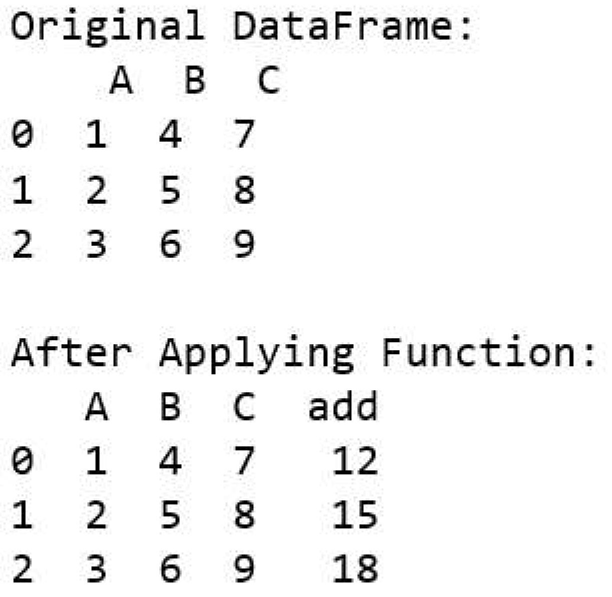
Apply function to every row in a Pandas DataFrame

Python is a great language for performing data analysis tasks. It provides with a huge amount of Classes and function which help in analyzing and manipulating data in an easier way. One can use apply() function in order to apply function to every row in given dataframe.

Example #1:

```
# Import pandas package
import pandas as pd

# Function to add
def add(a, b, c):
    return a + b + c

def main():

    # create a dictionary with
    # three fields each
    data = {
            'A':[1, 2, 3],
            'B':[4, 5, 6],
            'C':[7, 8, 9] }

    # Convert the dictionary into DataFrame
    df = pd.DataFrame(data)
    print("Original DataFrame:\n", df)

    df['add'] = df.apply(lambda row : add(row['A'],
                     row['B'], row['C']), axis = 1)

    print('\nAfter Applying Function: ')
    # printing the new dataframe
    print(df)

if __name__ == '__main__':
    main()
```

Output:

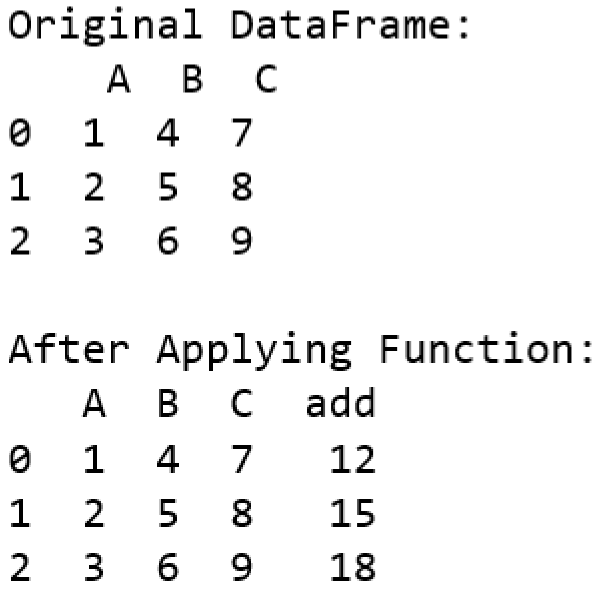


```
Original DataFrame:
     A  B  C
0  1  4  7
1  2  5  8
2  3  6  9

After Applying Function:
   A  B  C  add
0  1  4  7   12
1  2  5  8   15
2  3  6  9   18
```

Example #2:
You can use the numpy function as the parameters to the dataframe as well.

```
import pandas as pd
import numpy as np


def main():

    # create a dictionary with
    # five fields each
    data = {
            'A':[1, 2, 3],
            'B':[4, 5, 6],
            'C':[7, 8, 9] }

    # Convert the dictionary into DataFrame
    df = pd.DataFrame(data)
    print("Original DataFrame:\n", df)

    # applying function to each row in the dataframe
    # and storing result in a new column
    df['add'] = df.apply(np.sum, axis = 1)

    print('\nAfter Applying Function: ')
    # printing the new dataframe

    print(df)



if __name__ == '__main__':

    main()
```

Output:

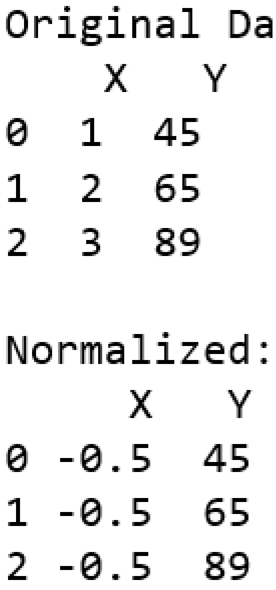


```
Original DataFrame:
    A  B  C
0  1  4  7
1  2  5  8
2  3  6  9

After Applying Function:
   A  B  C  add
0  1  4  7   12
1  2  5  8   15
2  3  6  9   18
```

Example #3: Normalising Data

```
# Import pandas package

import pandas as pd



def normalize(x, y):

    x_new = ((x - np.mean([x, y])) /

             (max(x, y) - min(x, y)))
    # print(x_new)
    return x_new


def main():

    # create a dictionary with three fields each
    data = {
        'X':[1, 2, 3],
        'Y':[45, 65, 89] }

    # Convert the dictionary into DataFrame
    df = pd.DataFrame(data)
    print("Original DataFrame:\n", df)

    df['X'] = df.apply(lambda row : normalize(row['X'],
                                  row['Y']), axis = 1)

    print('\nNormalized:')
    print(df)

if __name__ == '__main__':
    main()
```

Output:

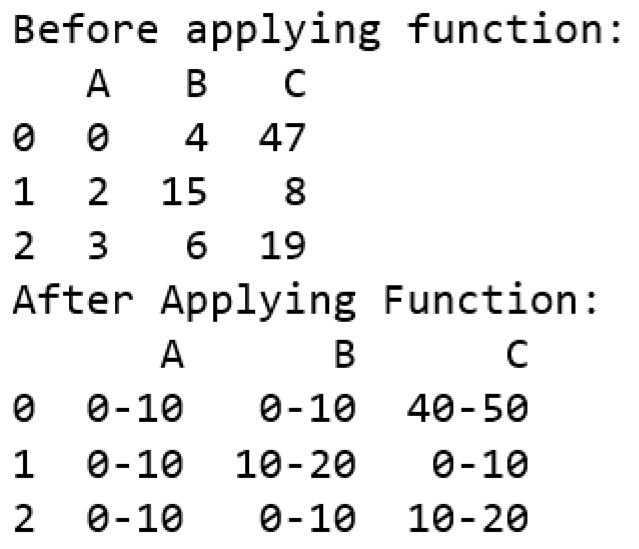


```
Original DataFrame:
   X   Y
0  1  45
1  2  65
2  3  89

Normalized:
     X   Y
0 -0.5  45
1 -0.5  65
2 -0.5  89
```

Example #4: Generate range

```
import pandas as pd
import numpy as np


pd.options.mode.chained_assignment = None


# Function to generate range
def generate_range(n):


    # printing the range for eg:
    # input is 67 output is 60-70
    n = int(n)


    lower_limit = n//10 * 10
    upper_limit = lower_limit + 10



    return str(str(lower_limit) + '-' + str(upper_limit))



def replace(row):

    for i, item in enumerate(row):



        # updating the value of the row

        row[i] = generate_range(item)

    return row





def main():

    # create a dictionary with

    # three fields each

    data = {

            'A':[0, 2, 3],

            'B':[4, 15, 6],

            'C':[47, 8, 19] }



    # Convert the dictionary into DataFrame

    df = pd.DataFrame(data)
    print('Before applying function: ')

    print(df)



    # applying function to each row in

    # dataframe and storing result in a new column

    df = df.apply(lambda row : replace(row))




    print('After Applying Function: ')

    # printing the new dataframe

    print(df)



if __name__ == '__main__':

    main()
```

Output:



```
Before applying function:
   A   B   C
0  0   4  47
1  2  15   8
2  3   6  19
After Applying Function:
      A      B      C
0  0-10   0-10  40-50
1  0-10  10-20   0-10
2  0-10   0-10  10-20
```