1. NumPy in Python

What is NumPy?

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined using Numpy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

1. Arrays in NumPy: NumPy's main object is the homogeneous multidimensional array.

It is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers. In NumPy dimensions are called axes. The number of axes is rank. NumPy's array class is called ndarray. It is also known by the alias array.

```
# Python program to demonstrate
# basic array characteristics
import numpy as np

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

# Printing type of arr object
print("Array is of type: ", type(arr))

# Printing array dimensions (axes)
print("No. of dimensions: ", arr.ndim)

# Printing shape of array
print("Shape of array: ", arr.shape)

# Printing size (total number of elements) of array
print("Size of array: ", arr.size)
```

```
# Printing type of elements in array
print("Array stores elements of type: ", arr.dtype)
```

```
Array is of type:  <class 'numpy.ndarray'>
No. of dimensions:  2
Shape of array:  (2, 3)
Size of array:  6
Array stores elements of type:  int64
```

2. Array creation: There are various ways to create arrays in NumPy.

1. For example, you can create an array from a regular Python list or tuple using the array function. The type of the resulting array is deduced from the type of the elements in the sequences.
2. Often, the elements of an array are originally unknown, but its size is known. Hence, NumPy offers several functions to create arrays with initial placeholder content. These minimize the necessity of growing arrays, an expensive operation. For example: np.zeros, np.ones, np.full, np.empty, etc.
3. To create sequences of numbers, NumPy provides a function analogous to range that returns arrays instead of lists.
4. arange: returns evenly spaced values within a given interval. step size is specified.
5. linspace: returns evenly spaced values within a given interval. num no. of elements are returned.
6. Reshaping array: We can use reshape method to reshape an array. Consider an array with shape (a1, a2, a3, …, aN). We can reshape and convert it into another array with shape (b1, b2, b3, …, bM). The only required condition is: a1 x a2 x a3 … x aN = b1 x b2 x b3 … x bM . (i.e original size of array remains unchanged.)
7. Flatten array: We can use flatten method to get a copy of array collapsed into one dimension. It accepts order argument. Default value is ‘C’ (for row-major order). Use ‘F’ for column major order.

```
# Python program to demonstrate
# array creation techniques
import numpy as np

# Creating array from list with type float
a = np.array([[1, 2, 4], [5, 8, 7]], dtype = 'float')
print ("Array created using passed list:\n", a)

# Creating array from tuple
b = np.array((1 , 3, 2))
print ("\nArray created using passed tuple:\n", b)

# Creating a 3X4 array with all zeros
c = np.zeros((3, 4))
```

```
c = np.zeros((3, 4))
print ("\nAn array initialized with all zeros:\n", c)

# Create a constant value array of complex type
d = np.full((3, 3), 6, dtype = 'complex')
print ("\nAn array initialized with all 6s."
            "Array type is complex:\n", d)

# Create an array with random values
e = np.random.random((2, 2))
print ("\nA random array:\n", e)

# Create a sequence of integers
# from 0 to 30 with steps of 5
f = np.arange(0, 30, 5)
print ("\nA sequential array with steps of 5:\n", f)

# Create a sequence of 10 values in range 0 to 5
g = np.linspace(0, 5, 10)
print ("\nA sequential array with 10 values between"
                                        "0 and 5:\n", g)

# Reshaping 3X4 array to 2X2X3 array
arr = np.array([[1, 2, 3, 4],
                [5, 2, 4, 2],
                [1, 2, 0, 1]])

newarr = arr.reshape(2, 2, 3)

print ("\nOriginal array:\n", arr)
print ("Reshaped array:\n", newarr)

# Flatten array
arr = np.array([[1, 2, 3], [4, 5, 6]])
flarr = arr.flatten()

print ("\nOriginal array:\n", arr)
print ("Fattened array:\n", flarr)
```

```
Array created using passed list:
 [[1. 2. 4.]
 [5. 8. 7.]]

Array created using passed tuple:
 [1 3 2]

An array initialized with all zeros:
 [[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]

An array initialized with all 6s.Array type is complex:
 [[6.+0.j 6.+0.j 6.+0.j]
 [6.+0.j 6.+0.j 6.+0.j]
 [6.+0.j 6.+0.j 6.+0.j]]

A random array:
 [[0.70864301 0.1445599 ]
 [0.62385575 0.05495546]]

A sequential array with steps of 5:
 [ 0  5 10 15 20 25]

A sequential array with 10 values between0 and 5:
 [0.         0.55555556 1.11111111 1.66666667 2.22222222 2.77777778
 3.33333333 3.88888889 4.44444444 5.        ]

Original array:
 [[1 2 3 4]
 [5 2 4 2]
 [1 2 0 1]]
Reshaped array:
 [[[1 2 3]
  [4 5 2]]

 [[4 2 1]
  [2 0 1]]]

Original array:
 [[1 2 3]
 [4 5 6]]
Fattened array:
 [1 2 3 4 5 6]
```

Operations on single array: We can use overloaded arithmetic operators to do element-wise operation on array to create a new array. In case of +=, -=, *= operators, the exsisting array is modified.

```
# Python program to demonstrate
# basic operations on single array
import numpy as np

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

# add 1 to every element
print ("Adding 1 to every element:", a+1)

# subtract 3 from each element
print ("Subtracting 3 from each element:", a-3)

# multiply each element by 10
print ("Multiplying each element by 10:", a*10)

# square each element
print ("Squaring each element:", a**2)
```

```
# modify existing array
a *= 2
print ("Doubled each element of original array:", a)

# transpose of array
a = np.array([[1, 2, 3], [3, 4, 5], [9, 6, 0]])

print ("\nOriginal array:\n", a)
print ("Transpose of array:\n", a.T)
```

```
Adding 1 to every element: [2 3 6 4]
Subtracting 3 from each element: [-2 -1  2  0]
Multiplying each element by 10: [10 20 50 30]
Squaring each element: [ 1  4 25  9]
Doubled each element of original array: [ 2  4 10  6]

Original array:
 [[1 2 3]
 [3 4 5]
 [9 6 0]]
Transpose of array:
 [[1 3 9]
 [2 4 6]
 [3 5 0]]
```

Binary operators: These operations apply on array elementwise and a new array is created. You can use all basic arithmetic operators like +, -, /, , etc. In case of +=, -=, = operators, the exsisting array is modified.

```
# Python program to demonstrate
# binary operators in Numpy
import numpy as np

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

# add arrays
print ("Array sum:\n", a + b)

# multiply arrays (elementwise multiplication)
print ("Array multiplication:\n", a*b)

# matrix multiplication
print ("Matrix multiplication:\n", a.dot(b))
```

```
Array sum:
 [[5 5]
 [5 5]]
```

```
Array multiplication:
 [[4 6]
 [6 4]]
Matrix multiplication:
 [[ 8  5]
 [20 13]]
```

```
# Python program to demonstrate sorting in numpy
import numpy as np

a = np.array([[1, 4, 2],
                 [3, 4, 6],
              [0, -1, 5]])

# sorted array
print ("Array elements in sorted order:\n",
                    np.sort(a, axis = None))

# sort array row-wise
print ("Row-wise sorted array:\n",
                np.sort(a, axis = 1))

# specify sort algorithm
print ("Column wise sort by applying merge-sort:\n",
            np.sort(a, axis = 0, kind = 'mergesort'))

# Example to show sorting of structured array
# set alias names for dtypes
dtypes = [('name', 'S10'), ('grad_year', int), ('cgpa', float)]

# Values to be put in array
values = [('Hrithik', 2009, 8.5), ('Ajay', 2008, 8.7),
           ('Pankaj', 2008, 7.9), ('Aakash', 2009, 9.0)]

# Creating array
arr = np.array(values, dtype = dtypes)
print ("\nArray sorted by names:\n",
            np.sort(arr, order = 'name'))

print ("Array sorted by grauation year and then cgpa:\n",
                np.sort(arr, order = ['grad_year', 'cgpa']))
```

```
Array elements in sorted order:
 [-1  0  1  2  3  4  4  5  6]
Row-wise sorted array:
 [[ 1  2  4]
 [ 3  4  6]
 [-1  0  5]]
Column wise sort by applying merge-sort:
 [[ 0 -1  2]
 [ 1  4  5]
 [ 3  4  6]]
```

```
Array sorted by names:
 [(b'Aakash', 2009, 9. ) (b'Ajay', 2008, 8.7) (b'Hrithik', 2009, 8.5)
 (b'Pankaj', 2008, 7.9)]
Array sorted by grauation year and then cgpa:
 [(b'Pankaj', 2008, 7.9) (b'Ajay', 2008, 8.7) (b'Hrithik', 2009, 8.5)
 (b'Aakash', 2009, 9. )]
```

2. Difference between Pandas VS NumPy

Pandas: It is an open-source, BSD-licensed library written in Python Language. Pandas provide high performance, fast, easy to use data structures and data analysis tools for manipulating numeric data and time series. Pandas is built on the numpy library and written in languages like Python, Cython, and C. In pandas, we can import data from various file formats like JSON, SQL, Microsoft Excel, etc.

```
# Importing pandas library
import pandas as pd

# Creating and initializing a nested list
age = [['Aman', 95.5, "Male"], ['Sunny', 65.7, "Female"],
       ['Monty', 85.1, "Male"], ['toni', 75.4, "Male"]]

# Creating a pandas dataframe
df = pd.DataFrame(age, columns=['Name', 'Marks', 'Gender'])

# Printing dataframe
df
```

	Name	Marks	Gender
0	Aman	95.5	Male
1	Sunny	65.7	Female
2	Monty	85.1	Male
3	toni	75.4	Male

Numpy: It is the fundamental library of python, used to perform scientific computing. It provides high-performance multidimensional arrays and tools to deal with them. A numpy array is a grid of values (of the same type) that are indexed by a tuple of positive integers, numpy arrays are fast, easy to understand, and give users the right to perform calculations across arrays.

```
# Importing Numpy package
import numpy as np
```

```
# Creating a 3-D numpy array using np.array()
org_array = np.array([[23, 46, 85],
                      [43, 56, 99],
                      [11, 34, 55]])

# Printing the Numpy array
print(org_array)
```

```
[[23 46 85]
 [43 56 99]
 [11 34 55]]
```

Numpy numpy.resize()

With the help of Numpy numpy.resize(), we can resize the size of an array. Array can be of any shape but to resize it we just need the size i.e (2, 2), (2, 3) and many more. During resizing numpy append zeros if values at a particular place is missing.

Example #1:

In this example we can see that with the help of .resize() method, we have changed the shape of an array from 1×6 to 2×3.

```
# importing the python module numpy
import numpy as np

# Making a random array
gfg = np.array([1, 2, 3, 4, 5, 6])

# Reshape the array permanently
gfg.resize(2, 3)

print(gfg)
```

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

Example #2:

In this example we can see that, we are trying to resize the array of that shape which is type of out of bound values. But numpy handles this situation to append the zeros when values are not existed in the array.

```
# importing the python module numpy
import numpy as np

# Making a random array
ga = np.array([1, 2, 3, 4, 5, 6])

# Required values 12, existing values 6
ga.resize(3, 4)

print(ga)
```

```
     [[1 2 3 4]
      [5 6 0 0]
      [0 0 0 0]]
```

Reshape() method in Numpy

Both the numpy.reshape() and numpy.resize() methods are used to change the size of a NumPy array. The difference between them is that the reshape() does not changes the original array but only returns the changed array, whereas the resize() method returns nothing and directly changes the original array.

Example 1: Using reshape()

```
# importing the module
import numpy as np

# creating an array
A = np.array([1, 2, 3, 4, 5, 6])
print("Original array:")
display(A)

# using reshape()
print("Changed array")
display(A.reshape(2, 3))

print("Original array:")
display(A)
```

```
Original array:
array([1, 2, 3, 4, 5, 6])
```

Example 2: Using resize()

```
array([1, 2, 3, 4, 5, 6])
```

```
# importing the module
import numpy as np

# creating an array
Aa = np.array([1, 2, 3, 4, 5, 6])
print("Original array:")
display(Aa)

# using resize()
print("Changed array")
# this will print nothing as None is returned
display(Aa.resize(2, 3))

print("Original array:")
display(Aa)
```

```
Original array:
array([1, 2, 3, 4, 5, 6])
Changed array
None
Original array:
array([[1, 2, 3],
       [4, 5, 6]])
```

```
# import the important module in python
import numpy as np

# make a matrix with numpy
B = np.matrix('[1, 2; 4, 5; 7, 8]')

print(B)

# applying matrix.reshape() method
new = B.reshape((2, 3))

print(new)
```

```
[[1 2]
 [4 5]
 [7 8]]
[[1 2 4]
 [5 7 8]]
```

Numpy.transpose()

numpy.transpose(), We can perform the simple function of transpose within one line by using numpy.transpose() method of Numpy. It can transpose the 2-D arrays on the other hand it has no effect on 1-D arrays. This method transpose the 2-D numpy array.

```
# importing python module named numpy
import numpy as np

# making a 3x3 array
Aa = np.array([[1, 2, 3],
               [4, 5, 6],
               [7, 8, 9]])

# before transpose
print(Aa, end ='\n\n')

# after transpose
print(Aa.transpose())
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]

[[1 4 7]
 [2 5 8]
 [3 6 9]]
```

```
# importing python module named numpy
import numpy as np

# making a 3x3 array
Ab = np.array([[1, 2],
               [4, 5],
               [7, 8]])

# before transpose
print(Ab, end ='\n\n')

# after transpose
print(Ab.transpose(1, 0))
```

```
[[1 2]
 [4 5]
 [7 8]]

[[1 4 7]
 [2 5 8]]
```

3. Convert NumPy Array to Pandas DataFrame

Step 1: Create a NumPy Array

```
import numpy as np

my_array = np.array([[11,22,33],[44,55,66]])

print(my_array)
print(type(my_array))
```

```
[[11 22 33]
 [44 55 66]]
<class 'numpy.ndarray'>
```

Step 2: Convert the NumPy Array to Pandas DataFrame

```
import numpy as np
import pandas as pd

my_array = np.array([[11,22,33],[44,55,66]])

df = pd.DataFrame(my_array, columns = ['Column_A','Column_B','Column_C'])

print(df)
print(type(df))
```

```
   Column_A  Column_B  Column_C
0        11        22        33
1        44        55        66
<class 'pandas.core.frame.DataFrame'>
```

Step 3 (optional): Add an Index to the DataFrame

What if you'd like to add an index to the DataFrame?

For instance, let's add the following index to the DataFrame

```
index = ['Item_1', 'Item_2']
```

So here is the complete code to convert the array to a DataFrame with an index:

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

```
my_array = np.array([[11,22,33],[44,55,66]])

df = pd.DataFrame(my_array, columns = ['Column_A','Column_B','Column_C'], index = ['Item_1',

print(df)
print(type(df))
```

```
        Column_A  Column_B  Column_C
Item_1        11        22        33
Item_2        44        55        66
<class 'pandas.core.frame.DataFrame'>
```

4. Array Contains a Mix of Strings and Numeric Data

```
import numpy as np

my_array = np.array([['Jon',25,1995,2016],['Maria',47,1973,2000],['Bill',38,1982,2005]], dtyp

print(my_array)
print(type(my_array))
print(my_array.dtype)
```

```
[['Jon' 25 1995 2016]
 ['Maria' 47 1973 2000]
 ['Bill' 38 1982 2005]]
<class 'numpy.ndarray'>
object
```

Use the following syntax to convert the NumPy array to a DataFrame:

```
import numpy as np
import pandas as pd

my_array = np.array([['Jon',25,1995,2016],['Maria',47,1973,2000],['Bill',38,1982,2005]], dtyp

df = pd.DataFrame(my_array, columns = ['Name','Age','Birth Year','Graduation Year'])

print(df)
print(type(df))
```

```
    Name Age Birth Year Graduation Year
0    Jon  25       1995            2016
1  Maria  47       1973            2000
2   Bill  38       1982            2005
<class 'pandas.core.frame.DataFrame'>
```

Let’s check the data types of all the columns in the new DataFrame by adding df.dtypes to the code:

```
import numpy as np
import pandas as pd

my_array = np.array([['Jon',25,1995,2016],['Maria',47,1973,2000],['Bill',38,1982,2005]], dtyp

df = pd.DataFrame(my_array, columns = ['Name','Age','Birth Year','Graduation Year'])

print(df)
print(type(df))
```

```
        Name Age Birth Year Graduation Year
     0    Jon  25       1995            2016
     1  Maria  47       1973            2000
     2   Bill  38       1982            2005
     <class 'pandas.core.frame.DataFrame'>
```

Let’s check the data types of all the columns in the new DataFrame by adding df.dtypes to the code:

```
import numpy as np
import pandas as pd

my_array = np.array([['Jon',25,1995,2016],['Maria',47,1973,2000],['Bill',38,1982,2005]], dtyp

df = pd.DataFrame(my_array, columns = ['Name','Age','Birth Year','Graduation Year'])

print(df)
print(type(df))
print(df.dtypes)
```

```
        Name Age Birth Year Graduation Year
     0    Jon  25       1995            2016
     1  Maria  47       1973            2000
     2   Bill  38       1982            2005
     <class 'pandas.core.frame.DataFrame'>
     Name               object
     Age                object
     Birth Year         object
     Graduation Year    object
     dtype: object
```

Currently, all the columns under the DataFrame are objects/strings

For example, suppose that you’d like to convert the last 3 columns in the DataFrame to integers.

To achieve this goal, you can use astype(int) as captured below:

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

my_array = np.array([['Jon',25,1995,2016],['Maria',47,1973,2000],['Bill',38,1982,2005]])

df = pd.DataFrame(my_array, columns = ['Name','Age','Birth Year','Graduation Year'])

df['Age'] = df['Age'].astype(int)
df['Birth Year'] = df['Birth Year'].astype(int)
df['Graduation Year'] = df['Graduation Year'].astype(int)

print(df)
print(type(df))
print(df.dtypes)
```

Double-click (or enter) to edit

How to Union Pandas DataFrames using Concat

You can union Pandas DataFrames using contact:

pd.concat([df1, df2])

Step 1: Create the first DataFrame

```
import pandas as pd

clients1 = {'clientFirstName': ['Jon','Maria','Bruce','Lili'],
            'clientLastName': ['Smith','Lam','Jones','Chang'],
            'country': ['US','Canada','Italy','China']
           }

df1 = pd.DataFrame(clients1, columns= ['clientFirstName', 'clientLastName','country'])

print (df1)
```

```
  clientFirstName clientLastName country
0             Jon          Smith      US
1           Maria            Lam  Canada
2           Bruce          Jones   Italy
3            Lili          Chang   China
```

Step 2: Create the second DataFrame

```
import pandas as pd
```

```
clients2 = {'clientFirstName': ['Bill','Jack','Elizabeth','Jenny'],
            'clientLastName': ['Jackson','Green','Gross','Sing'],
            'country': ['UK','Germany','Brazil','Japan']
           }

df2 = pd.DataFrame(clients2, columns= ['clientFirstName', 'clientLastName','country'])

print (df2)
```

```
  clientFirstName clientLastName  country
0            Bill        Jackson       UK
1            Jack          Green  Germany
2       Elizabeth          Gross   Brazil
3           Jenny           Sing    Japan
```

Step 3: Union Pandas DataFrames using Concat

```
import pandas as pd

clients1 = {'clientFirstName': ['Jon','Maria','Bruce','Lili'],
            'clientLastName': ['Smith','Lam','Jones','Chang'],
            'country': ['US','Canada','Italy','China']
           }

df1 = pd.DataFrame(clients1, columns= ['clientFirstName', 'clientLastName','country'])


clients2 = {'clientFirstName': ['Bill','Jack','Elizabeth','Jenny'],
            'clientLastName': ['Jackson','Green','Gross','Sing'],
            'country': ['UK','Germany','Brazil','Japan']
           }

df2 = pd.DataFrame(clients2, columns= ['clientFirstName', 'clientLastName','country'])

union = pd.concat([df1, df2])
print (union)
```

```
  clientFirstName clientLastName  country
0             Jon          Smith       US
1           Maria            Lam   Canada
2           Bruce          Jones    Italy
3            Lili          Chang    China
0            Bill        Jackson       UK
1            Jack          Green  Germany
2       Elizabeth          Gross   Brazil
3           Jenny           Sing    Japan
```

You may then choose to assign the index values in an incremental manner once you concatenated the two DataFrames.

To do so, simply set ignore_index=True within the pd.concat brackets:

```
import pandas as pd

clients1 = {'clientFirstName': ['Jon','Maria','Bruce','Lili'],
            'clientLastName': ['Smith','Lam','Jones','Chang'],
            'country': ['US','Canada','Italy','China']
           }

df1 = pd.DataFrame(clients1, columns= ['clientFirstName', 'clientLastName','country'])


clients2 = {'clientFirstName': ['Bill','Jack','Elizabeth','Jenny'],
            'clientLastName': ['Jackson','Green','Gross','Sing'],
            'country': ['UK','Germany','Brazil','Japan']
           }

df2 = pd.DataFrame(clients2, columns= ['clientFirstName', 'clientLastName','country'])

union = pd.concat([df1, df2], ignore_index=True)
print (union)
```

```
  clientFirstName clientLastName  country
0             Jon          Smith       US
1           Maria            Lam   Canada
2           Bruce          Jones    Italy
3            Lili          Chang    China
4            Bill        Jackson       UK
5            Jack          Green  Germany
6       Elizabeth          Gross   Brazil
7           Jenny           Sing    Japan
```

Pandas.DataFrame.loc

```
df = pd.DataFrame([[1, 2], [4, 5], [7, 8]],
     index=['cobra', 'viper', 'sidewinder'],
     columns=['max_speed', 'shield'])
df
```

	max_speed	shield
cobra	1	2
viper	4	5
sidewinder	7	8

```
df.loc['viper']
```

```
     max_speed    4
     shield       5
     Name: viper, dtype: int64
```

```
df.loc[['viper', 'sidewinder']]
```

	max_speed	shield
viper	4	5
sidewinder	7	8

```
df.loc['cobra', 'shield']
```

```
     2
```

```
df.loc['cobra':'viper', 'max_speed']
```

```
     cobra    1
     viper    4
     Name: max_speed, dtype: int64
```

IF condition in Pandas DataFrame

(1) IF condition – Set of numbers

```
import pandas as pd

numbers = {'set_of_numbers': [1,2,3,4,5,6,7,8,9,10]}
df = pd.DataFrame(numbers,columns=['set_of_numbers'])

df.loc[df['set_of_numbers'] <= 4, 'equal_or_lower_than_4?'] = 'True'
df.loc[df['set_of_numbers'] > 4, 'equal_or_lower_than_4?'] = 'False'

print (df)
```

```
        set_of_numbers equal_or_lower_than_4?
     0               1                   True
     1               2                   True
     2               3                   True
     3               4                   True
     4               5                  False
     5               6                  False
     6               7                  False
     7               8                  False
     8               9                  False
     9              10                  False
```

(2) IF condition – set of numbers and lambda

how to get the same results as in case 1 by using lambada, where the conditions are:

If the number is equal or lower than 4, then assign the value of ‘True’ Otherwise, if the number is greater than 4, then assign the value of ‘False’

```
import pandas as pd

numbers = {'set_of_numbers': [1,2,3,4,5,6,7,8,9,10]}
df = pd.DataFrame(numbers,columns=['set_of_numbers'])

df['equal_or_lower_than_4?'] = df['set_of_numbers'].apply(lambda x: 'True' if x <= 4 else 'Fa

print (df)
```

```
        set_of_numbers equal_or_lower_than_4?
     0               1                   True
     1               2                   True
     2               3                   True
     3               4                   True
     4               5                  False
     5               6                  False
     6               7                  False
     7               8                  False
     8               9                  False
     9              10                  False
```

(3) IF condition – strings

```
import pandas as pd

names = {'First_name': ['Jon','Bill','Maria','Emma']}
df = pd.DataFrame(names,columns=['First_name'])

df.loc[df['First_name'] == 'Bill', 'name_match'] = 'Match'
df.loc[df['First_name'] != 'Bill', 'name_match'] = 'Mismatch'

print (df)
```

```
       First_name name_match
     0        Jon   Mismatch
     1       Bill      Match
     2      Maria   Mismatch
     3       Emma   Mismatch
```

(4) IF condition – strings and lambada

```
import pandas as pd

names = {'First_name': ['Jon','Bill','Maria','Emma']}
df = pd.DataFrame(names,columns=['First_name'])

df['name_match'] = df['First_name'].apply(lambda x: 'Match' if x == 'Bill' else 'Mismatch')

print (df)
```

```
  First_name name_match
0        Jon   Mismatch
1       Bill      Match
2      Maria   Mismatch
3       Emma   Mismatch
```

(5) IF condition with OR

```
import pandas as pd

names = {'First_name': ['Jon','Bill','Maria','Emma']}
df = pd.DataFrame(names,columns=['First_name'])

df.loc[(df['First_name'] == 'Bill') | (df['First_name'] == 'Emma'), 'name_match'] = 'Match'
df.loc[(df['First_name'] != 'Bill') & (df['First_name'] != 'Emma'), 'name_match'] = 'Mismatch

print (df)
```

```
  First_name name_match
0        Jon   Mismatch
1       Bill      Match
2      Maria   Mismatch
3       Emma      Match
```

Applying an IF condition under an existing DataFrame column

```
import pandas as pd

numbers = {'set_of_numbers': [1,2,3,4,5,6,7,8,9,10,0,0]}
df = pd.DataFrame(numbers,columns=['set_of_numbers'])
print (df)

df.loc[df['set_of_numbers'] == 0, 'set_of_numbers'] = 999
df.loc[df['set_of_numbers'] == 5, 'set_of_numbers'] = 555

print (df)
```

```
        set_of_numbers
    0                1
    1                2
    2                3
    3                4
    4                5
    5                6
    6                7
    7                8
    8                9
    9               10
    10               0
    11               0
        set_of_numbers
    0                1
    1                2
    2                3
    3                4
    4              555
    5                6
    6                7
    7                8
    8                9
    9               10
    10             999
    11             999
```

```
import pandas as pd
import numpy as np

numbers = {'set_of_numbers': [1,2,3,4,5,6,7,8,9,10,np.nan,np.nan]}
df = pd.DataFrame(numbers,columns=['set_of_numbers'])
print (df)

df.loc[df['set_of_numbers'].isnull(), 'set_of_numbers'] = 0
print (df)
```

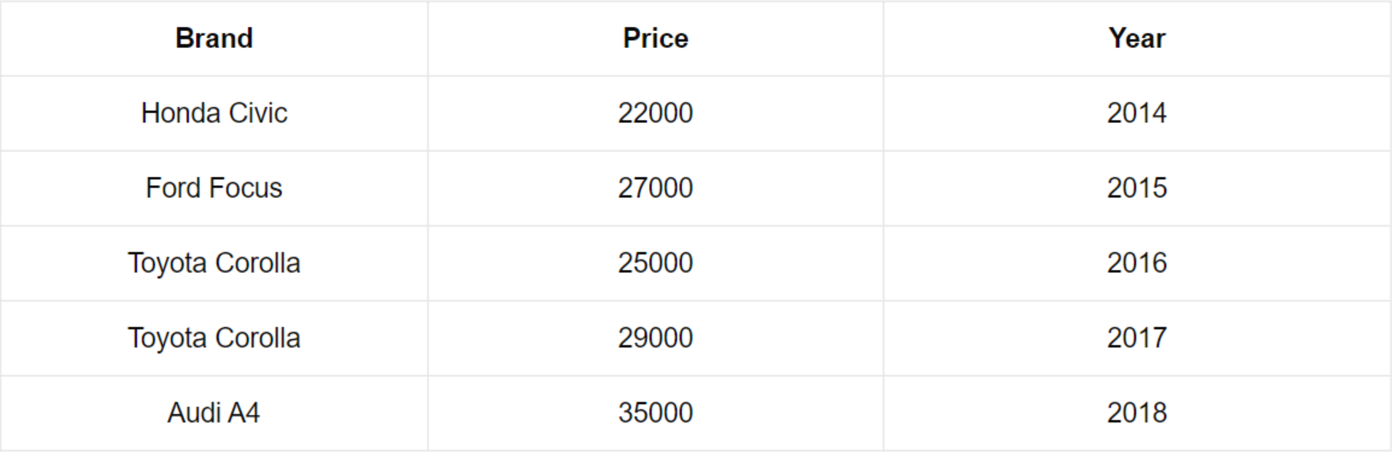
```
        set_of_numbers
    0              1.0
    1              2.0
    2              3.0
    3              4.0
    4              5.0
    5              6.0
    6              7.0
    7              8.0
    8              9.0
    9             10.0
    10             NaN
    11             NaN
        set_of_numbers
    0              1.0
    1              2.0
    2              3.0
```

```
3             4.0
4             5.0
5             6.0
6             7.0
7             8.0
8             9.0
9            10.0
10            0.0
11            0.0
```

Descriptive Statistics for Pandas DataFrame

Steps to Get the Descriptive Statistics for Pandas DataFrame

Step 1: Collect the Data



Brand	Price	Year
Honda Civic	22000	2014
Ford Focus	27000	2015
Toyota Corolla	25000	2016
Toyota Corolla	29000	2017
Audi A4	35000	2018

Step 2: Create the DataFrame

```
from pandas import DataFrame

Cars = {'Brand': ['Honda Civic','Ford Focus','Toyota Corolla','Toyota Corolla','Audi A4'],
        'Price': [22000,27000,25000,29000,35000],
         'Year': [2014,2015,2016,2017,2018]
        }

df = DataFrame(Cars, columns= ['Brand', 'Price','Year'])
print (df)
```

```
            Brand  Price  Year
0     Honda Civic  22000  2014
1      Ford Focus  27000  2015
2  Toyota Corolla  25000  2016
```

```
3  Toyota Corolla  29000  2017
4         Audi A4  35000  2018
```

Step 3: Get the Descriptive Statistics for Pandas DataFrame

```
from pandas import DataFrame

Cars = {'Brand': ['Honda Civic','Ford Focus','Toyota Corolla','Toyota Corolla','Audi A4'],
        'Price': [22000,27000,25000,29000,35000],
         'Year': [2014,2015,2016,2017,2018]
        }

df = DataFrame(Cars, columns= ['Brand', 'Price','Year'])

stats_numeric = df['Price'].describe()
print (stats_numeric)
```

```
count        5.000000
mean     27600.000000
std       4878.524367
min      22000.000000
25%      25000.000000
50%      27000.000000
75%      29000.000000
max      35000.000000
Name: Price, dtype: float64
```

You'll notice that the output contains 6 decimal places. You may then add the syntax of astype (int) to the code to get integer values.

This is how the code would look like:

```
from pandas import DataFrame

Cars = {'Brand': ['Honda Civic','Ford Focus','Toyota Corolla','Toyota Corolla','Audi A4'],
        'Price': [22000,27000,25000,29000,35000],
         'Year': [2014,2015,2016,2017,2018]
        }

df = DataFrame(Cars, columns= ['Brand', 'Price','Year'])

stats_numeric = df['Price'].describe().astype (int)
print (stats_numeric)
```

```
count        5
mean     27600
std       4878
min      22000
25%      25000
50%      27000
75%      29000
```

```
max      35000
Name: Price, dtype: int64
```

Descriptive Statistics for Categorical Data

```
from pandas import DataFrame

Cars = {'Brand': ['Honda Civic','Ford Focus','Toyota Corolla','Toyota Corolla','Audi A4'],
        'Price': [22000,27000,25000,29000,35000],
         'Year': [2014,2015,2016,2017,2018]
        }

df = DataFrame(Cars, columns= ['Brand', 'Price','Year'])

stats_categorical = df['Brand'].describe()
print (stats_categorical)
```

```
count                  5
unique                 4
top       Toyota Corolla
freq                   2
Name: Brand, dtype: object
```

*Descriptive Statistics for the Entire Pandas DataFrame *

```
from pandas import DataFrame

Cars = {'Brand': ['Honda Civic','Ford Focus','Toyota Corolla','Toyota Corolla','Audi A4'],
        'Price': [22000,27000,25000,29000,35000],
         'Year': [2014,2015,2016,2017,2018]
        }

df = DataFrame(Cars, columns= ['Brand', 'Price','Year'])

stats = df.describe(include='all')
print (stats)
```

```
                 Brand         Price         Year
count                5      5.000000     5.000000
unique               4           NaN          NaN
top     Toyota Corolla           NaN          NaN
freq                 2           NaN          NaN
mean               NaN  27600.000000  2016.000000
std                NaN   4878.524367     1.581139
min                NaN  22000.000000  2014.000000
25%                NaN  25000.000000  2015.000000
```

```
50%               NaN  27000.000000  2016.000000
75%               NaN  29000.000000  2017.000000
max               NaN  35000.000000  2018.000000
```

Breaking Down the Descriptive Statistics that are :

1. Count
2. Mean
3. Standard deviation
4. Minimum
5. 0.25 Quantile
6. 0.50 Quantile (Median)
7. 0.75 Quantile
8. Maximum

```
from pandas import DataFrame

Cars = {'Brand': ['Honda Civic','Ford Focus','Toyota Corolla','Toyota Corolla','Audi A4'],
        'Price': [22000,27000,25000,29000,35000],
         'Year': [2014,2015,2016,2017,2018]
        }

df = DataFrame(Cars, columns= ['Brand', 'Price','Year'])

count1 = df['Price'].count()
print('count: ' + str(count1))

mean1 = df['Price'].mean()
print('mean: ' + str(mean1))

std1 = df['Price'].std()
print('std: ' + str(std1))

min1 = df['Price'].min()
print('min: ' + str(min1))

quantile1 = df['Price'].quantile(q=0.25)
print('25%: ' + str(quantile1))

quantile2 = df['Price'].quantile(q=0.50)
print('50%: ' + str(quantile2))

quantile3 = df['Price'].quantile(q=0.75)
print('75%: ' + str(quantile3))

max1 = df['Price'].max()
print('max: ' + str(max1))
```

How to Plot a DataFrame using Pandas

How to plot a DataFrame using Pandas follow the complete steps to plot:

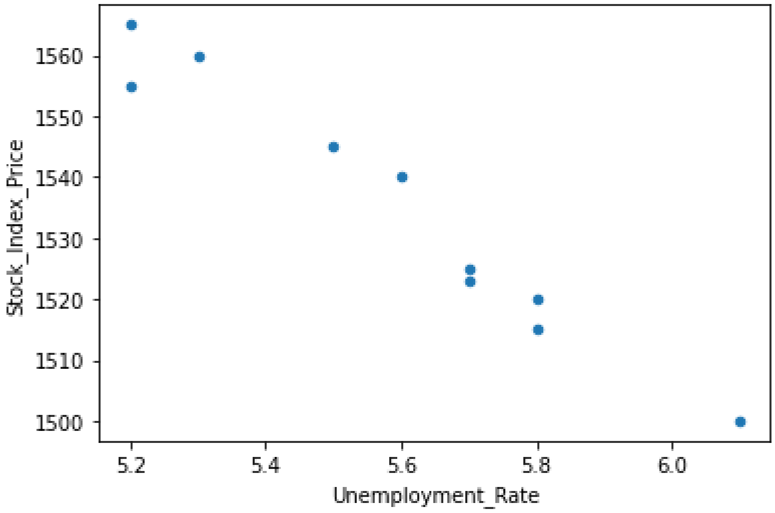
1. Scatter diagram
2. Line chart
3. Bar chart
4. Pie chart

1. Plot a Scatter Diagram using Pandas

```
import pandas as pd
import matplotlib.pyplot as plt

data = {'Unemployment_Rate': [6.1,5.8,5.7,5.7,5.8,5.6,5.5,5.3,5.2,5.2],
        'Stock_Index_Price': [1500,1520,1525,1523,1515,1540,1545,1560,1555,1565]
       }

df = pd.DataFrame(data,columns=['Unemployment_Rate','Stock_Index_Price'])
df.plot(x ='Unemployment_Rate', y='Stock_Index_Price', kind = 'scatter')
plt.show()
```

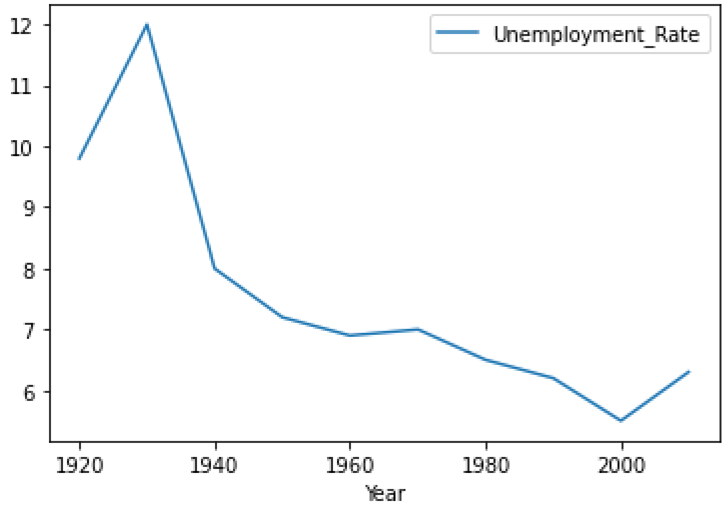


2. Plot a Line Chart using Pandas

```
import pandas as pd
import matplotlib.pyplot as plt
```

```
data = {'Year': [1920,1930,1940,1950,1960,1970,1980,1990,2000,2010],
        'Unemployment_Rate': [9.8,12,8,7.2,6.9,7,6.5,6.2,5.5,6.3]
       }

df = pd.DataFrame(data,columns=['Year','Unemployment_Rate'])
df.plot(x ='Year', y='Unemployment_Rate', kind = 'line')
plt.show()
```

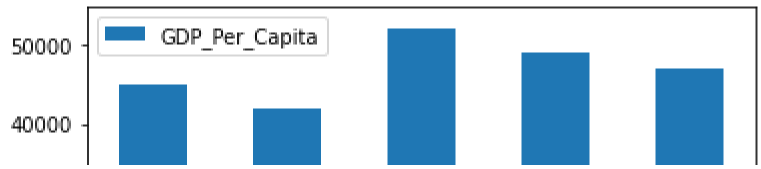


3. Plot a Bar Chart using Pandas

```
import pandas as pd
import matplotlib.pyplot as plt

data = {'Country': ['USA','Canada','Germany','UK','France'],
        'GDP_Per_Capita': [45000,42000,52000,49000,47000]
       }

df = pd.DataFrame(data,columns=['Country','GDP_Per_Capita'])
df.plot(x ='Country', y='GDP_Per_Capita', kind = 'bar')
plt.show()
```

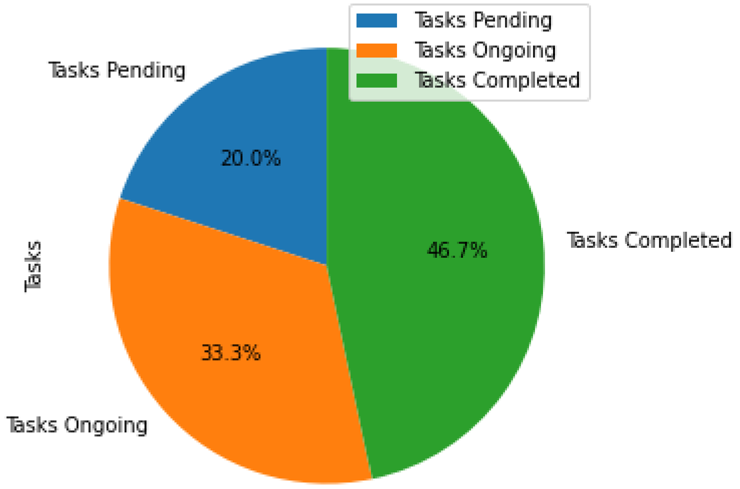


4. Plot a Pie Chart using Pandas

```
import pandas as pd
import matplotlib.pyplot as plt

data = {'Tasks': [300,500,700]}
df = pd.DataFrame(data,columns=['Tasks'],index = ['Tasks Pending','Tasks Ongoing','Tasks Comp

df.plot.pie(y='Tasks',figsize=(5, 5),autopct='%1.1f%%', startangle=90)
plt.show()
```


Tasks Pending
Tasks Ongoing
Tasks Completed
Tasks Pending
Tasks
Tasks Completed
Tasks Ongoing