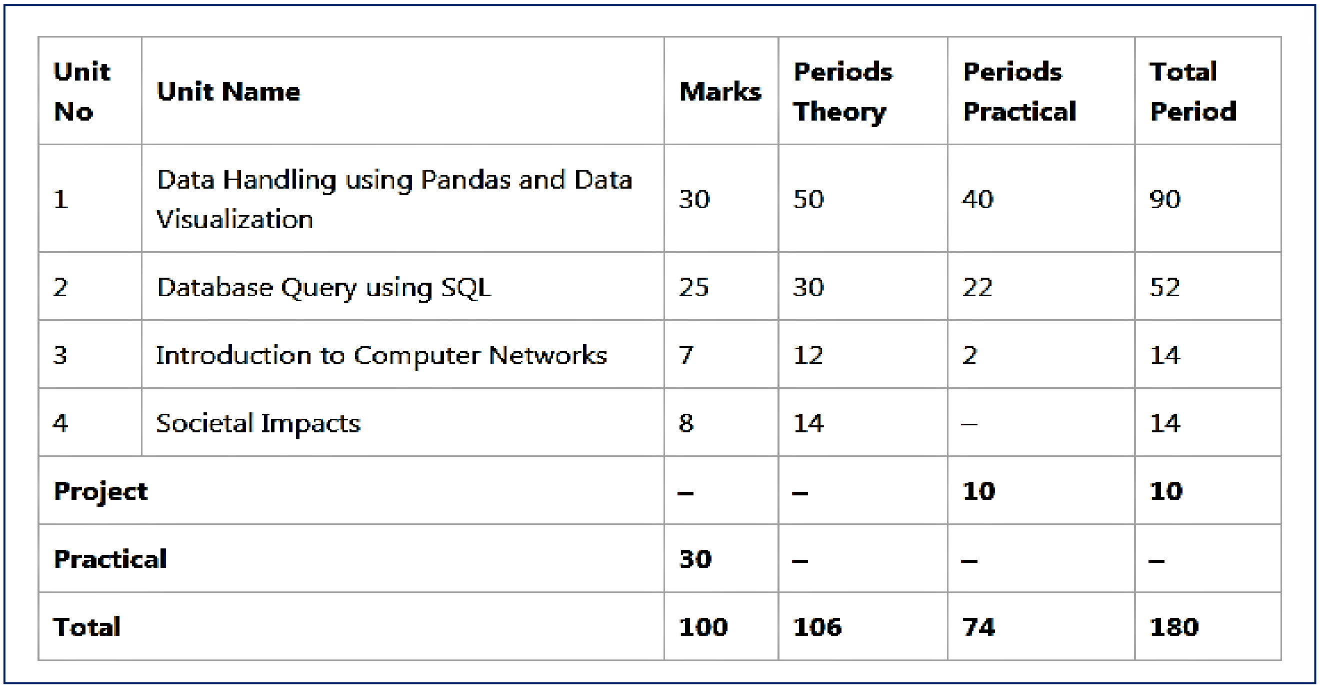
UNIT- I

Data Handling using Pandas and Data Visualization



Unit No	Unit Name	Marks	Periods Theory	Periods Practical	Total Period
1	Data Handling using Pandas and Data Visualization	30	50	40	90
2	Database Query using SQL	25	30	22	52
3	Introduction to Computer Networks	7	12	2	14
4	Societal Impacts	8	14	–	14
Project		**–**	**–**	**10**	**10**
Practical		**30**	**–**	**–**	**–**
Total		**100**	**106**	**74**	**180**

Syllabus:

i) **Data Handling using Pandas -I**

- Introduction to Python libraries - Pandas, Matplotlib.
- Data structures in Pandas - **Series** and **Data Frames**.
- Series: Creation of Series from – ndarray, dictionary, scalar value; mathematical operations; Head and Tail functions, Selection, Indexing and Slicing.
- Data Frames: creation - from dictionary of Series, list of dictionaries, Text/CSV files; display; iteration;
- Operations on rows and columns: add, select, delete, rename; Head and Tail functions; Indexing using Labels,
- Boolean Indexing; **Joining, Merging and Concatenation.**
- Importing/Exporting Data between CSV files and Data Frames.

ii) **Data handling using Pandas – II**

- **Descriptive Statistics: max, min, count, sum, mean, median, mode, quartile, Standard deviation, variance.**
- **DataFrame operations: Aggregation, group by, Sorting, Deleting and Renaming Index, Pivoting.**
- **Handling missing values – dropping and filling.**
- **Importing/Exporting Data between MySQL database and Pandas.**

iii) Data Visualization

- Purpose of plotting; drawing and saving following types of plots using Matplotlib – line plot, bar graph.
- Histogram, **pie chart, frequency polygon, box plot and scatter plot.**
- Customizing plots: **color, style (dashed, dotted), width;** adding label, title, and legend in plots.

NB. *RED marks have been excluded*

===

Pandas is an open-source Python Library providing high-performance Data Science for data manipulation and analysis tool using its powerful data structures. The name Pandas is derived from the word Panel Data – an Econometrics from Multidimensional data.

The **MatPotLib** Python library, developed by John Hunter and many other contributors, is used to create high-quality graphs, charts, and figures. The library is extensive and capable of changing very minute details of a figure. Some basic concepts and functions provided in matplotlib are:

- **Figure and axes:** The entire illustration is called a figure and each plot on it is an axes (do not confuse Axes with Axis).
- **Plotting:** The very first thing required to plot a graph is data. A dictionary of key-value pairs can be declared, with keys and values as the x and y values. After that, scatter(), bar(), and pie(), along with tons of other functions, can be used to create plot.
- **Axis:** The figure and axes obtained using subplots() can be used for modification. Properties of the x-axis and y-axis (labels, minimum and maximum values, etc.) can be changed using Axes.set().

Data Type of Pandas

- integer
- string
- float
- object

Data Structure of Pandas:

- **Series:** 1-D structure to store homogeneous (same data type) and mutable (can be modified/added) data, but size of the series is immutable.
- **DataFrame:** 2-D structure to store heterogeneous (multiple data type) and mutable data.
- **Panel:** It is 3-D way of storing data.

Series

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. For example, the following series is a collection of integers 10, 23, 56,…

10 23 56 17 52 61 73 90 26 72

Key Points:

- Homogeneous data
- Size Immutable
- Values of Data Mutable

A series can be created using various inputs like:

- List
- Dictionary
- NdArray
- Scalar value or constant

Create an Empty Series:

Example:

```
#import the pandas library and aliasing as pd
        import pandas as pd
        s = pd.Series()
        print (s)
```

Output:

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

Create a Series from List:

If data is a list, then index passed must be of the same length. If no index is passed, then by default index will be range(n) where n is array length, i.e., 0,1,2,3…. range(len(list))-1].

Example:

```
#import the pandas library and aliasing as pd
      x=[10, 20, 30, 40, 50] # List
      import pandas as pd
      s = pd.Series(x) # Series
      print (s)
```

Output:

```
0   10
1   20
2   30
3   40
4   50
   dtype: object
```

Create a Series from ndarray:

If data is a ndarray, then index passed must be of the same length. If no index is passed, then by default index will be range(n) where n is array length, i.e., *0,1,2,3…. range(len(array))-1].

Example: **With integer values in a list**

```
import pandas
x=[10,20,30,40]
print("The original list: ")
print(x)
s1=pandas.Series(x)
print("The Series: ")
print(s1)
print("With user given index: ")
s2=pandas.Series(x, index=[100,200,300,400])
print(s2)
```

Output:

```
The original list:
[10, 20, 30, 40]

The Series:
0        10
1        20
2        30
3        40
dtype: int64
With user given index:
100        10
200        20
300        30
400        40
dtype: int64
```

# With Text value:

```
import pandas as pd
x=['India', 'UK', 'USA', 'China','Russia']
print("The original list: ")
print(x)
```

```
s1=pd.Series(x)
print("The Series: ")
print(s1)
print("With user given index: ")
s2=pd.Series(x, index=[100,200,300,400,500])
print(s2)
```

Output:

```
The original list:
['India', 'UK', 'USA', 'China', 'Russia']
The Series:
0    India
1       UK
2      USA
3    China
4   Russia
            dtype: object
With user given index:
100    India
200       UK
300      USA
400    China
500   Russia
            dtype: object
```

```
x=['a','b','c','d'] # List
import pandas as pd import
numpy as np
data = np.array(x) # Array created by NumPy
s = pd.Series(data) # Series
print (s)
```

Output:

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

Array with defined indexes:

Example:

#import the pandas library and aliasing as pd

```
import pandas as pd
import numpy as np
data = np.array(['a','b','c','d'])
s = pd.Series(data, index=[100,101,102,103])
print (s)
```

Output:

```
100  a
101  b
102  c
103  d
        dtype: object
```

Creating series on a dictionary:

A dict can be passed as input and if no index is specified, then the dictionary keys are taken in a sorted order to construct index. If index is passed, the values in data corresponding to the labels in the index will be pulled out.

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

Output:

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

Observe − Dictionary keys are used to construct the index.

Example

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

Output:

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

Observe: Index order is persisted and the missing element is filled with NaN (Not a Number).

Example:

```
x={'a': “Africa”, 'b': ”Britain“, 'c': “Canada” , 'd': “Denmark”}
import pandas as pd
s = pd.Series(data) # Series
print (s)
```

Output:

```
a        Africa
b        Britain
c        Canada
d        Denmar
   dtype: string
```

Example:

```
import pandas as pd
data = {'a' : 0., 'b' : 1., 'c' : 2.}
s = pd.Series(data)
print (s)
s1=pd.Series(data, index=['a', 'c', 'd'])
print(s1)
s2=pd.Series(data, index=['b','c','a','x'])
print(s2)
print(s.index)
print(s.dtype)
print(s.shape)
```

Output:

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

a    0.0
c    2.0
d    NaN
dtype: float64

b    1.0
c    2.0
a    0.0
x    NaN
dtype: float64

Index(['a', 'b', 'c'], dtype='object')

float64

(3,)
```

Create a Series from Scalar:

If data is a scalar value, an index must be provided then the value will be repeated to match the length of index

```
import pandas as pd
```

```
s = pd.Series(15, index=[0, 1, 2, 3, 4])
print (s)
```

Output:

```
0       15
1       15
2       15
3       15
4       15
dtype: int64
```

Accessing Data from Series with Position:

Data in the series can be accessed similar to that in an ndarray.

Example

Retrieve/access the first element. As we already know, the counting starts from zero for the array, which means the first element is stored at zeroth position and so on.

```
import pandas as pd
s = pd.Series([1,2,3,4,5], index = ['a','b','c','d','e'])
#retrieve the first element
print (s[0])
```

Output:

```
1
```

Example

Retrieve/access/index the first three elements in the Series. If a: is inserted in front of it, all items from that index onwards will be extracted. If two parameters (with: between them) is used, items between the two indexes (not including the stop index)

```
import pandas as pd
s = pd.Series([1,2,3,4,5], index = ['a','b','c','d','e']) #retrieves the first three element
print (s[:3])
```

Output:

```
a       1
b       2
c       3

dtype: int64
```

Example

Retrieve (slicing) the last three elements.

```
import pandas as pd
s = pd.Series([1,2,3,4,5], index = ['a','b','c','d','e']) #retrieve the last three element
print (s[-3:])
```

Output:

```
c           3
d           4
e           5
dtype: int64
```

Retrieve Data Using Label (Index)

A Series is like a fixed-size dict in that you can get and set values by index label.

Example

Retrieve a single element using index label value.

```
import pandas as pd
s = pd.Series([1,2,3,4,5], index = ['a','b','c','d','e']) #retrieve a single element
print (s['a'])
```

Output:

```
1
```

Example

Retrieve multiple elements using a list of index label values.

```
import pandas as pd
s = pd.Series([1,2,3,4,5] ,index = ['a','b','c','d','e']) #retrieve multiple elements
print (s[ ['a','c','d'] ])
```

Output:

```
a           1
c           3
d           4
dtype: int64
```

Example 3

If a label is not contained, an exception is raised.

```
import pandas as pd
s = pd.Series([1,2,3,4,5], index = ['a','b','c','d','e'])
print (s['f']) # No output, as ‘f’ is not present
```

Output:

```
KeyError: 'f'
```

Creating Series using Mathematical Operations:

Example:

```
import numpy as np
import pandas as pd
s1=np.arange(10, 15) # Spelling is arange() [10,11,12,13,14]
print(s1)
s2=pd.Series(data=s1*2, index=s1)
print(s2)
```

Output:

```
[10, 11, 12, 13, 14]
dtype: int64

10     20
11     22
12     24
13     26
14     28
       dtype: int64
```

Example:

```
import pandas as pd
s = pd.Series(range(2,15,2))
print(s)
s1=pd.Series(range(2,15,2), index=[10,11,12,13,14,16,17])
print(s1)
```

Output:

```
0    2
1    4
2    6
3    8
4   10
5   12
6   14
dtype: int64

10    2
11    4
12    6
13    8
14   10
16   12
17   14
dtype: int64
```

# Example: Slicing using iloc and loc

```
    import pandas as pd
    s = pd.Series(range(2,10,2))
    print("#Prints the values of Series: ")
    print(s)
    s1=pd.Series(range(2,12,2), index=[10,11,12,13,14])
    print("#Prints the of Series with user index: ")
    print(s1)
    print("#Prints the values of Series with default index: ")
    print(s[1:4])
```

```
        print("#Prints the same values of Series with default index using iloc: ")
        print(s.iloc[1:4])
        print("#Can't print with iloc as 11:14 are not default indexes: ")
        print(s1.iloc[11:14])
        print("#Prints the values of Series with user index using loc: ")
        print(s1.loc[11:14])  # It prints all the values of the range (:)  of labels (user index)
```

```
  Output:
  3     8
  14    10
  dtype: int64

  #Prints the values of Series with default index:
  1    4
  2    6
  3    8
  dtype: int64

  #Prints the same values of Series with default index using iloc:
  1    4
  2    6
  3    8
  dtype: int64

  #Can't print with iloc as 11:14 are not default indexes:
  Series([], dtype: int64)

  #Prints the values of Series with user index using loc:
  11     4
  12     6
  13     8
  14    10
dtype: int64
```

# Example: Indexing and accessing using iloc and loc:

```
        import pandas as pd
        s=pd.Series([1, 2, 3, 4, 5] , index=['a', 'b','c','d','e'])
        print(s)
        print()
        print(s.iloc[1 : 4]) # for indexing or selecting based on position
        print()
        print(s.loc['b' : 'e']) # Rule of range of values doesn't work here
```

```
Output:
  a         1
  b         2
  c         3
  d         4
  e         5
```

```
            dtype: int64


    b       2
    c       3
    d       4
            dtype: int64


    b       2
    c       3
    d       4
    e       5
            dtype: int64
```

Difference between loc , iloc & slicing, range

```
import pandas as pd
s = pd.Series([11,22,33,44,55,66,77,88,99,100], index=[49,48,47,46,45, 1, 2, 3, 4, 5])
print(s.loc[:3])    # Prints the values till user's index
print()
print(s.loc[1:3])  # Prints the values till user's index
print()
print(s[:3])         # Prints the values till default index-1
print()
print(s[1:3])  # Prints the values till default index-1
print()
print(s.iloc[:3]) # Prints the values same as default index rules
print()
print(s.iloc[1:3])
```

Output:

```
-------
49    11
48    22
47    33
46    44
45    55
1     66
2     77
3     88
dtype: int64

1    66
2    77
3    88
dtype: int64

49    11
48    22
47    33
dtype: int64
```

```
49    11
48    22
47    33
dtype: int64
```

Head and Tail functions

To view a small sample of a Series or the DataFrame object, use the head() and the tail() methods.

head() returns the first n rows(observe the index values). The default number of elements to display is five, but you may pass a custom number.

Example:

```
import pandas as pd
s = pd.Series([10,20,30,40,50], index=[1,2,3,4,5])
print(s)
print("Head=>")
print(s.head()) # top 5 rows by default
print(s.head(3)) # to 3 rows
```

Output:

```
1          10
2          20
3          30
4          40
5          50
dtype: int64

Head=>
1           10
2           20
3           30
4           40
5           50

dtype: int64

1         10
2         20
3         30
dtype: int64
```

Example:

```
import pandas as pd
import numpy as np
#Create a series with 4 random numbers
s = pd.Series(np.random.randn(4))
print ("The original series is:")
print (s)
print ("The first two rows of the data series:")
print s.head(2)
```

Output:

```
The original series is:
0   0.720876
1  -0.765898
2   0.479221
3  -0.139547
            dtype: float64

The first two rows of the data series:
0   0.720876
1  -0.765898
            dtype: float64
```

tail() returns the last n rows(observe the index values). The default number of elements to display is five, but you may pass a custom number.

Example:

```
import pandas as pd
s = pd.Series([10,20,30,40,50], index=[1,2,3,4,5])
print(s)
print(“Tail=> “)
print(s.tail()) # By default print 5 lowermost rows
print(s.tail(3)) # Prints 3 rows from bottom of the series
```

Output:

```
1           10
2           20
3           30
4           40
5           50
dtype: int64


Tail=>
1           10
2           20
3           30
4           40
5           50
dtype: int64

3           30
4           40
5           50
dtype: int64
```

Example:

```
import pandas as pd import
numpy as np
#Create a series with 4 random numbers
s = pd.Series(np.random.randn(4))
print ("The original series is:")
print (s)
print ("The last two rows of the data series:")
print s.tail(2)
```

Output:

```
The original series is:
0 -0.655091
1 -0.881407
2 -0.608592
3 -2.341413
        dtype: float64

The last two rows of the data series:
2 -0.608592
3 -2.341413
        dtype: float64
```

Access/Replace values with Condition (*Series.where()*):

Pandas Series.where() function replace values where the input condition is False for the given Series object. It takes another object as an input which will be used to replace the value from the original object.

Syntax: Series.where(cond, other=nan, inplace=False, axis=None, level=None, errors='raise', try_cast=False, raise_on_error=None)

Example: Print the series more than 50

```
import pandas as pd
s = pd.Series([10,20,30,40,50,60,70,80,90,100])
a=s.where(s > 50)
print(a)
```

Output:

```
0     NaN
1     NaN
2     NaN
3     NaN
4     NaN
5    60.0
6    70.0
7    80.0
8    90.0
9   100.0
dtype: float64
```

Example#2: Print the series has a value 50

```
import pandas as pd
s = pd.Series([10,20,30,40,50,60,70,80,90,100])
a=s.where(s == 50)
print(a)
```

Output:

```
0    NaN
1    NaN
2    NaN
3    NaN
4    50.0
5    NaN
6    NaN
7    NaN
8    NaN
9    NaN
dtype: float64
```

Example#3: Print the series less than 50

```
import pandas as pd
s = pd.Series([10,20,30,40,50,60,70,80,90,100])
a=s.where(s <= 50)
print(a)
```

Output:

```
0   10.0
1   20.0
2   30.0
3   40.0
4   50.0
5    NaN
6    NaN
7    NaN
8    NaN
9    NaN
dtype: float64
```

Example: Show/Filter the values with condition in 3 different ways

```
import pandas as pd
s = pd.Series(range(2,10,2))
print(s)
print(s>3) # Shows only Ture those meet the condition & rest False
print(s[s>3]) # Shows the values meet the condition
print(s.where(s>3)) # Shows the values those meet the condition & rest shows NaN
```

Output:

```
0    2
1    4
2    6
3    8
dtype: int64

0    False
1     True
2     True
3     True
dtype: bool

1    4
2    6
3    8
dtype: int64

0    NaN
1    4.0
2    6.0
3    8.0
dtype: float64
```

Example #4: **Use Series.where() function to replace values in the given Series object with some other value when the passed condition is not satisfied.**

```
import pandas as pd
# Creating the First Series
sr1 = pd.Series(['New York', 'Chicago', 'Toronto', 'Lisbon', 'Rio'])
sr1.index = ['City 1', 'City 2', 'City 3', 'City 4', 'City 5']
print(sr1)

# Creating the Second Series
sr2 = pd.Series(['New York', 'Bangkok', 'London', 'Lisbon', 'Brisbane'])
sr2.index = ['City 1', 'City 2', 'City 3', 'City 4', 'City 5']
print(sr2)

# Replace the values using Series.where()
a=sr1.where(sr1 == 'Rio', sr2)
print(a)
```

Output:

```
# Serites of sr1
City 1      New York
City 2      Chicago
City 3      Toronto
City 4      Lisbon
City 5      Rio
dtype: object
```

# Serites of sr2

```
City 1      New York
City 2      Bangkok
City 3      London
City 4      Lisbon
City 5      Brisbane
dtype: object
```

# ‘Brisbane’ of sr2 of index city5 has been replaced by ‘Rio’ of sr1 of same index

```
City 1      New York
City 2      Bangkok
City 3      London
City 4      Lisbon
City 5      Rio
dtype: object
```

Example #5 : **Use Series.where() function to replace values in the given Series object with some other value when the passed condition is not satisfied.**

```
import pandas as pd
sr1 = pd.Series([22, 18, 19, 20, 21], index = ['Student 1', 'Student 2', 'Student 3', 'Student 4', 'Student 5'])
print(sr1)
print()

# Creating the second Series
sr2 = pd.Series([19, 16, 22, 20, 18] , index = ['Student 1', 'Student 2', 'Student 3', 'Student 4', 'Student 5'])
print(sr2)
print()
print("#Replace the values with Series.where()")
b=sr1.where(sr1>20, sr2)
print(b)
```

# Replace the value(s) of sr2 by sr1 with Series.where()

```
b=sr1.where(sr1 >20, sr2)
print(b)
```

Output:

```
Student 1    22
Student 2    18
Student 3    19
Student 4    20
Student 5    21
dtype: int64

Student 1    19
Student 2    16
Student 3    22
Student 4    20
Student 5    18
dtype: int64
```

```
#Replacing the values of sr2 by sr1 where the corresponding value of sr1 is greater than 20
Student 1   22
Student 2   16
Student 3   22
Student 4   20
Student 5   21
dtype: int64
```

Mathematical Operations:

+	add()
-	sub(), subtract()
*	mul(), multiply()
/	div(), divide()
//	floordiv()
%	mod()
**	pow()

Example:

```
import pandas as pd
s1 = pd.Series([1,2,3,4])
s2 = pd.Series([10,20,30,40])
print (s1)
print (s2)
print”ADD:”, (s2+s1)        # print(s2.add(s1))
print(”SUB:”, s2-s1)        # print(s2.sub(s1))
print(”MUL:”, s2*s1)        # print(s2.multiply(s1))
print(”DIV:”, s2/s1)        # print(s2.div(s1))
print(”F.DIV:”, s2//s1)     # print(s2.floordiv(s1))
print(”MOD:”, s2%s1)        # print(s2.mod(s1))
print(”POW:”, s2**s1)       # print(s2.pow(s1))
```

Output:

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

0       10
1       20
2       30
3       40
dtype: int64

ADD:
0       11
1       22
2       33
3       44
dtype: int64

SUB:
0       9
1       18
```

```
2           27
3           36

dtype: int64


MUL:
0           11
1           22
2           33
3           44
dtype: int64

DIV:
0           10.0
1           10.0
2           10.0
3           10.0
dtype: float

F.DIV:
0           10
1           10
2           10
3           10
dtype: int64
```

Using range() and for loop:

```
    import pandas as pd
     s = pd.Series(range(1 , 15 , 3), index=(x for x in ‘abcde’)
     print (s)
```

Output:

```
a         1
b         4
c         7
d         10
e         13
          dtype: float64
```

###

DataFrame

DataFrame is a **two-dimensional array** with **heterogeneous** data, like a table with rows and columns.

A pandas DataFrame can be created using various inputs like

- Lists
- dict
- Series
- Numpy ndarrays
- Another DataFrame

Key Points of DataFrame:

- Heterogeneous data
- Size Mutable
- Data Mutable
- Can Perform Arithmetic operations on rows and columns

For example in the following table (DataStructure):

Name	Age	Gender	Rating
Raman	32	Male	3.45
Jayati	28	Female	4.6
Saurav	45	Male	3.9
Kritika	38	Female	2.78

The table represents the data of a team of an organization with their overall performance rating. The data is represented in rows and columns. Each column represents an attribute and each row represents a person.

Data Type of Columns:

The data types of the four columns are as follows −

Column	Type
Name	String
Age	Integer
Gender	String
Rating	Float

1. Create an Empty DataFrame

Example:

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

Output:

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

2. Create a DataFrame from Lists:

Example:

```
import pandas as pd
df= pd.DataFrame([10, 20, 30, 40, 50])
print (df)
```

Output:

```
    0
0  10
1  20
2  30
3  40
4  50
```

3. Create a DataFrame from Nested Lists:

Example:

```
import pandas as pd
df = pd.DataFrame([ [1, 2, 3, 4, 5] , [10, 20, 30, 40, 50] ])
print (df)
```

Output:

```
   0   1      2      3      4

0  1   2      3      4      5       # Comes from 1st List
1  10  20     30     40     50      # Comes from 2nd List
```

Example: **Create a DataFrame with Column names**

```
import pandas as pd
x = [[‘XII’,101] , [‘XI’,210] , [‘X’,301]]
df = pd.DataFrame(x , columnS=['Class' , '90% Score'])
print (df)
```

Output:

```
   Class 90% Score
0   XII     101
1   XI      201
2   X       301
```

Example:

```
import pandas as pd
data = [['Alex',10],['Bob',12],['Clarke',13]]
```

```
df = pd.DataFrame(data , columns=['Name' , 'Age'] , dtype=float)
print (df)
```

Output:

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

4. Create a DataFrame from Dictionary of ndarrays

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:

```
import pandas as pd
d = {'col1': [1, 2], 'col2': [3, 4]}
df = pd.DataFrame(d)
print (df)
```

Output:

```
  col1 col2
0    1    3
1    2    4
```

Example

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

Output:

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

Example:

```
import pandas as pd
nme = ["aparna", "pankaj", "sudhir", "Geeku"]
deg = ["MBA", "BCA", "M.Tech", "MBA"]
scr = [90, 40, 80, 98]
dict = {'Name': nme, 'Degree': deg, 'Score': scr}
df = pd.DataFrame(dict)
print(df)
```

Output:

```
           Name              Degree          Score
0          Aparna            MBA             90
1          Pankaj            BCA             40
2          Sudhir            M.Tech          80
3          Geeku             MBA             98
```

Example:

```
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)
```

Output:

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

4.1 Example (Changing Row indexes):

```
import pandas as pd
data = {    'Name': ['Jai', 'Princ', 'Gaurav', 'Anuj'] ,
            'Height': [5.1 , 6.2 , 5.1 , 5.2] ,
            'Qualification': ['Msc' , 'MA' , 'Msc' , 'Msc']
            }
df = pd.DataFrame(data)
print(df)
print("") # Print/insert a blank line
df1 = pd.DataFrame(data, index=['one', 'two', 'three', 'four']) # Assigning index
print(df1)
```

Output:

```
         Height    Name           Qualification
0        5.1       Jai            Msc
1        6.2       Princ          MA
2        5.1       Gaurav         Msc
3        5.2       Anuj           Msc

         Height    Name           Qualification
one      5.1       Jai            Msc
two      6.2       Princ          MA
three    5.1       Gaurav         Msc
four     5.2       Anuj           Msc
```

5. Sorting of data in DataFrame:

5.1 Example:

```
import pandas as pd
cars =      {'Brand': ['Honda Civic','Toyota Corolla','Ford Focus','Audi A4'],
             'Price': [22000,25000,27000,35000],
             'Year': [2015,2013,2018,2018]
            }
df = pd.DataFrame(cars, columns= ['Brand', 'Price','Year'])
print (df)
```

Output:

```
      Brand            Price     Year
 0    Honda Civic      22000     2015
 1    Toyota Corolla   25000     2013
 2    Ford Focus       27000     2018
 3    Audi A4          35000     2018
```

5.2 Example:

```
import pandas as pd
cars = {'Brand': ['Honda Civic','Toyota Corolla','Ford Focus','Audi A4'],
     'Price': [22000,25000,27000,35000],
     'Year': [2015,2013,2018,2018]
     }

df = pd.DataFrame(cars, columns= ['Brand', 'Price','Year']) sd=df.sort__valueS(by=['Brand'],
ascending=False) # Descending order
print (sd)
```

Output:

```
          Brand            Price     Year
1         Toyota Corolla   25000     2013
0         Honda Civic      22000     2015
2         Ford Focus       27000     2018
3         Audi A4          35000     2018
```

5.3 Example:

```
import pandas as pd
cars = {'Brand': ['Honda Civic' , 'Toyota Corolla', 'Ford Focus' , 'Audi A4'],
           'Price': [22000, 25000, 27000, 35000],
           'Year': [2015, 2013, 2018, 2018]
         }
df = pd.DataFrame(cars, columns= ['Brand', 'Price' , 'Year'])
sd=df.sort_values(by=['Price'], ascending=False) # Descending order
print (sd)
```

Output:

```
          Brand               Price      Year
3         Audi A4             35000      2018
2         Ford Focus          27000      2018
1         Toyota Corolla      25000      2013
0         Honda Civic         22000      2015
```

5.4 Example:

```
import pandas as pd
cars = {'Brand': ['Honda Civic','Toyota Corolla','Ford Focus','Audi A4'],
          'Price': [22000,25000,27000,35000],
           'Year': [2015,2013,2018,2018]
      }
df = pd.DataFrame(cars, columns= ['Brand', 'Price','Year'])
sd=df.sort_values(by=['Year'], ascending=True) # Ascending order
print (sd)
```

Output:

```
          Brand               Price       Year
1         Toyota Corolla      25000       2013
0         Honda Civic         22000       2015
2         Ford Focus          27000       2018
3         Audi A4             35000       2018
```

6. Renaming a column name in DataFrame

6.1 Example (Renaming one field):

```
import pandas as pd
L1=[10,30,50,70,90]
print(L1)                    # Prints List
df=pd.DataFrame(L1)
print(df)                    #Prints DataFrame
df.columns=['Code'] # Renaming column
print(df)
```

Output:

```
[10 , 30 , 50 , 70 , 90]

        0
0        10
1        30
2        50
3        70
4        90

        Code 10
0
1        30
2        50
3        70
4        90
```

6.2 Renaming column using function rename()

```
import pandas as pd
df = pd.DataFrame({"A": [1, 2, 3], "B": [4, 5, 6]})
print(df)
df.rename(columns={"A": "a", "B": "c"}, inplace=True)
print(df)
```

Output:

```
   a  c
0  1  4
1  2  5
2  3  6
```

Note: *When* ***inplace = True*** *, the data is modified in place, which means it will return nothing and the* ***dataframe is now updated****. When inplace = False , which is the default, then the operation is performed and it returns a copy of the object. You then need to save it to something.*

6.3 Example (Get new column names):

```
 import pandas as pd
 data = {    'Name': ['Jai', 'Princ', 'Gaurav', 'Anuj'] ,
             'Height': [5.1 , 6.2 , 5.1 , 5.2] ,
             'Qualification': ['Msc' , 'MA' , 'Msc' , 'Msc']}
df = pd.DataFrame(data, index=['one', 'two', 'three', 'four'])
print(df)
 df.columns=['N. Height' , 'N. Name', 'N. Qualification'] # New column names
 print(df)
```

Output:

```
        Height       Name        Qualification
one     5.1          Jai         Msc
two     6.2          Princ       MA
three   5.1          Gaurav      Msc
four    5.2          Anuj        Msc

        N. Hight     N. Name     N. Qualification
 one     5.1         Jai         Msc
 two     6.2         Princ       MA
 three    5.1        Gaurav      Msc
 four     5.2        Anuj        Msc
```

6.3 Example (Replace a specific column name):

```
 import pandas as pd
data = {          'Name': ['Jai', 'Princ', 'Gaurav', 'Anuj'] ,
                    'Height': [5.1 , 6.2 , 5.1 , 5.2] ,
                   'Qualification': ['Msc' , 'MA' , 'Msc' , 'Msc']}
df = pd.DataFrame(data)
```

```
df.rename(columns={'N. Qualification' : 'Degree'} , inplace=True) # Replacing a specific column
print(df)
```

Output:

```
         N. Hight     N. Name      Degree
one      5.1          Jai          Msc
two      6.2          Princ        MA
three    5.1          Gaurav       Msc
four     5.2          Anuj         Msc
```

7 Adding column to a DataFrame

7.1 Example:

```
import pandas as pd
data = {    'Name': ['Jai', 'Princ', 'Gaurav', 'Anuj'] ,
            'Height': [5.1 , 6.2 , 5.1 , 5.2] ,
            'Qualification': ['Msc' , 'MA' , 'Msc' , 'Msc']
            }
df = pd.DataFrame(data)
print(df)                        # Prints the DataFrame
print(“”)
addr = ['Delhi', 'Bangalore', 'Chennai', 'Patna'] # Declare a list
df['New Address'] = addr         # Prints the DataFrame with a new column
print(df)
```

Output:

```
          Name       Height     Qualification
0         Jai        5.1        Msc
1         Princ      6.2        MA
2         Gaurav     5.1        Msc
3         Anju       3.2        Msc

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

7.2 Example: **Addition of Series to DataFrame row & column wise**

```
import pandas as pd
x = pd.DataFrame({0: [1,2,3], 1: [4,5,6], 2: [7,8,9] })
print(x)
print()
y = pd.Series([1, 2, 3])
print(y)
print()
new_x = x.add(y, axis=0) # Adding series to DF row-wise for axis=0 on 0th col using add() function
```

```
print(new_x)
print()
new_y = x.add(y, axis=1) # Adding series to DF col-wise for axis=1 on 0th row using add() function
print(new_y)
```

output;

```
   0  1  2
0  1  4  7
1  2  5  8
2  3  6  9

0    1
1    2
2    3
dtype: int64

   0  1   2
0  2  5   8
1  4  7  10
2  6  9  12


   0  1   2
0  2  6  10
1  3  7  11
2  4  8  12
```

7.3 Example: **Binary operation of DataFrame with DtaFrame row /column wise: addition**

```
import pandas as pd
x = pd.DataFrame({0: [1,2,3], 1: [4,5,6], 2: [7,8,9] })
y = pd.DataFrame({0: [1,2,3], 1: [4,5,6], 2: [7,8,9] })
print(x)
print()
print(y)
print()
x1 = x.add(y, axis=0)   # Adding series to DF row-wise as axis=0 on 0th col
print(x1)
print()
y1 = x.add(y, axis=1)   # Adding series to DF col-wise as axis=1 on 0th row
print(y1)
```

Output:

```
   0  1  2
0  1  4  7
1  2  5  8
2  3  6  9
dtype: int64
```

```
   0  1  2
0  1  4  7
1  2  5  8
2  3  6  9
dtype: int64

    0   1   2
0  2   8  14
1  4  10  16
2  6  12  18
dtype: int64

    0   1   2
0  2   8  14
1  4  10  16
2  6  12  18
dtype: int64
```

Example: **Binary mathematical operations of within DataFrames row wise**

```
import pandas as pd
x = pd.DataFrame({0: [1,2,3], 1: [4,5,6], 2: [7,8,9] })
y = pd.DataFrame({0: [1,2,3], 1: [4,5,6], 2: [7,8,9] })
print(x)
print()
print(y)
print("\n Addition: \n")
x1 = x.add(y, axis=0)   # Row wise addition
print(x1)
print("\n Subtraction: \n")
x2 = x.sub(y, axis=0)   # Row wise subtraction
print(x2)
print("\n Multiplication: \n")
x3 = x.mul(y, axis=0)   # Row wise multiplication
print(x3)
print("\n Division: \n")
x4 = x.div(y, axis=0)   # Row wise division
print(x4)
```

Output:

```
   0  1  2
0  1  4  7
1  2  5  8
2  3  6  9

   0  1  2
0  1  4  7
1  2  5  8
2  3  6  9
```

```
Addition:
   0   1   2
0  2   8  14
1  4  10  16
2  6  12  18

 Subtraction:
   0  1  2
0  0  0  0
1  0  0  0
2  0  0  0

 Multiplication:
   0   1   2
0  1  16  49
1  4  25  64
2  9  36  81

 Division:
     0    1    2
0  1.0  1.0  1.0
1  1.0  1.0  1.0
2  1.0  1.0  1.0
```

7.5 Example: **Binary mathematical operations of within DataFrames column wise**

```
import pandas as pd
x = pd.DataFrame({0: [1,2,3], 1: [4,5,6], 2: [7,8,9] })
y = pd.DataFrame({0: [1,2,3], 1: [4,5,6], 2: [7,8,9] })
print(x)
print()
print(y)
print("\n Addition: \n")
x1 = x.add(y, axis=1)      # Col wise addition
print(x1)
print("\n Subtraction: \n")
x2 = x.sub(y, axis=1)      # Col wise subtraction
print(x2)
print("\n Multiplication: \n")
x3 = x.mul(y, axis=1)      # Colwise multiplication
print(x3)
print("\n Division: \n")
x4 = x.div(y, axis=1)      # Col wise division
print(x4)
```

Output:

```
   0  1  2
0  1  4  7
1  2  5  8
2  3  6  9
```

Addition:

```
   0   1   2
0  2   8  14
1  4  10  16
2  6  12  18
```

Subtraction:

```
   0  1  2
0  0  0  0
1  0  0  0
2  0  0  0
```

Multiplication:

```
   0   1   2
0  1  16  49
1  4  25  64
2  9  36  81
```

Division:

```
     0    1    2
0  1.0  1.0  1.0
1  1.0  1.0  1.0
2  1.0  1.0  1.0
```

8 Selecting columns from DataFrame:

8.1 Example:

```
import pandas as pd
data = { 'Name': ['Jai', 'Princ', 'Gaurav', 'Anuj'],
            'Age': [27, 24, 22, 32],
                'Address': ['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
                'Qualification': ['Msc', 'MA', 'MCA', 'Phd']
                }
df = pd.DataFrame(data)
x1=df[df.columns[1 : 4]]
print(x1)                     # From 2nd col to 4th col
print(“”)
x2=df[df.columns[1:3] # From 2nd col to 3rd  col
print(x2)
x3=df[df.columns[:]]
print(x3)                     # All columns
print(“”)
x4=df[df.columns[:2]]
print(x4)                     # First 1st col to 2nd col
```

Output:

```
        Age     Address     Qualification
0       27      Delhi       Msc
1       24      Kanpur      MA
2       22      Allahabad   MCA
3       32      Kannauj     Phd

            Age     Address
    0        27     Delhi
    1       24      Kanpur
    2       22      Allahabad
    3       32      Kannauj

      Name        Age         Address        Qualification
0     Jai         27          Delhi          Msc
1     Princ       24          Kanpur         MA
2     Gaurav      22          Allahabad      MCA
3     Anuj        32          Kannauj        Phd

      Name              Age
0       Jai             27
1       Princ           24
2       Gaurav          22
3       Anuj            32
```

8.1 Selecting data from DataFrame with iteration:

```
import pandas as pd
# import numpy as np1
raw_data1 = { 'name': ['freya', 'mohak'],
        'age': [10, 1],
        'favorite_color': ['pink', 'blue'],
        'grade': [88, 92]}
df1 = pd.DataFrame(raw_data1, columns = ['name', 'age', 'favorite_color', 'grade'])
for index, row in df1.iterrows():          # Reads data with index row wise & returns to vriable row
    print (row["name"], row["age"])
```

Output:

```
freya 10
mohak 1
```

9 Adding a row to a DataFrame (1st way)

```
import pandas as pd
data = {   'Name': ['Jai', 'Princ', 'Gaurav', 'Anuj'] ,
           'Height': [5.1 , 6.2 , 5.1 , 5.2] ,
           'Qualification': ['Msc' , 'MA' , 'Msc' , 'Msc']
           }
df = pd.DataFrame(data)
print(df)                          # Prints the DataFrame
x = {'Name': 'Amit', 'Height': 5.5, 'Qualification': ‘PhD’}
df = df.append(x, ignore_index = True)
# ignore_index : bool, default False
# If True, do not use the index labels,  the resulting axis will be labeled 0, 1,..., n-1.
```

```
print(df)
```

Output:

```
          Name        Height       Qualification
0         Jai         5.1          Msc
1         Princ       6.2          MA
2         Gaurav      5.1          Msc
3         Anju        3.2          Msc

          Name        Height       Qualification
0         Jai         5.1          Msc
1         Princ       6.2          MA
2         Gaurav      5.1          Msc
3         Anju        3.2          Msc
4         Amit        5.5          PhD
```

9.1 Adding a row to a DataFrame (2nd way)

```
import pandas as pd
data = { 'Name': ['Jai', 'Princ', 'Gaurav', 'Anuj'] ,
                'Height': [5.1 , 6.2 , 5.1 , 5.2] ,
                'Qualification': ['Msc' , 'MA' , 'Msc' , 'Msc']
        }
df = pd.DataFrame(data)
print(df)        # Prints the DataFrame
print("")
df.loc[len(df.index)] = ['Amit', 5.5, 'PhD']
# df.loc locates the index to place values of the new row
# len(df.index) returens number of indexes = no. of rows+1 i.e. last index+1 i.e. new index
print(df)
```

```
Name            Height   Qualification
0       Jai             5.1       Msc
1       Princ   6.2        MA
2       Gaurav          5.1       Msc
3       Anuj            5.2       Msc

        Name            Height   Qualification
0       Jai             5.1       Msc
1       Princ   6.2        MA
2       Gaurav          5.1       Msc
3       Anuj            5.2       Msc
4       Amit            5.5       PhD
```

10 Select Row / Column using iloc (Index Location):

Syntax: df.iloc[range of rows(x:y) , range of columns(a:b)]

df.iloc[x:y , a:b]

11. Example:

```
    import pandas as pd
    data = { 'Name': ['Jai', 'Princ', 'Gaurav', 'Anuj'], 'Age': [27,
                 24, 22, 32],
                 'Address': ['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
                 'Qualification': ['Msc', 'MA', 'MCA', 'Phd']
                 }
  df = pd.DataFrame(data)
    x1=df.iloc[ : , 1 : 4]     # First part is range of rows , Second part is range of columns
    print(x1)
    x2=df.iloc[1:2 , 1:4]
    print(x2)
    x3=df.iloc[1: , 1:]
    print(x3)
    4=df.iloc[: , :]
    print(x4)
```

Output:

```
     Age   Address       Qualification
0       27   Delhi           Msc
1       24   Kanpur          MA
2       22   Allahabad       MCA
3       32   Kannauj         Phd

   Age 24  Address          Qualification
1          Kanpur
                            MA

      Age  Address          Qualification
1       24 Kanpur           MA
2       22 Allahabad        MCA
3       32 Kannauj          Phd

   Name         Age        Address      Qualification
0 Jai           27         Delhi        Msc
1 Princ         24         Kanpur       MA
2 Gaurav        22         Allahabad    MCA
3 Anuj          32         Kannauj      Phd
```

Crating DataFrame with Series and adding columns to DataFrame

```
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)
print()
df['three']=pd.Series([10,20,30],index=['a','b','c'])  # Adding a new column
print(df)
print()
df ['four']=df ['one']+df ['three']         # Adding a new column made by sum of other columns
print(df )
print()
print (df['one'])           # Displaying a column data
```

Output:

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

   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

   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

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

4 Deleting Row / Column:

10.1 Example:

```
import pandas as pd
data = { 'Name': ['Jai', 'Princ', 'Gaurav', 'Anuj'],
               'Age': [27, 24, 22, 32],
                 'Address': ['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
                 'Qualification': ['Msc', 'MA', 'MCA', 'Phd']
                 }
      df = pd.DataFrame(data)
      print(df)
```

Output:

```
   Name         Age    Address        Qualification
0 Jai           27     Delhi          Msc
1 Princ         24     Kanpur         MA
2 Gaurav        22     Allahabad      MCA
3 Anuj          32     Kannauj        Phd
```

Deleting a column using del:

```
      import pandas as pd
      data = { 'Name': ['Jai', 'Princ', 'Gaurav', 'Anuj'],
                 'Age': [27, 24, 22, 32],
                 'Address': ['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
                 'Qualification': ['Msc', 'MA', 'MCA', 'Phd']
                 }
      df = pd.DataFrame(data)
      del df['Name'] # Removes / deletes field
      print(df)
```

Output:

```
   Age   Address        Qualification
0 27      Delhi         Msc
1 24     Kanpur         MA
2 22     Allahabad      MCA
3 32     Kannauj        Phd
```

Deleting a column using pop():

```
      import pandas as pd
      data = { 'Name': ['Jai', 'Princ', 'Gaurav', 'Anuj'], 'Age':
                 [27, 24, 22, 32],
                 'Address': ['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
                 'Qualification': ['Msc', 'MA', 'MCA', 'Phd']
                 }
      df = pd.DataFrame(data)
      df.pop('Age')
      print(df)
```

Output:

```
   Name         Address        Qualification
0 Jai           Delhi          Msc
1 Princ         Kanpur         MA
2 Gaurav        Allahabad      MCA
3 Anuj          Kannauj        Phd
```

Deleting a columns using drop():

```
import pandas as pd
data = { 'Name': ['Jai', 'Princ', 'Gaurav', 'Anuj'],
         'Age': [27, 24, 22, 32],
         'Address': ['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
         'Qualification': ['Msc', 'MA', 'MCA', 'Phd']
          }
df = pd.DataFrame(data)
df1= df.drop("Address",  axis=1) # Axis=1 is column
print(df1)
```

Output:

```
   Name       Age     Qualification
0  Jai        27      Msc
1 Princ       24      MA
2 Gaurav      22      MCA
3 Anuj        32      Phd
```

Deleting a rows using drop():

```
import pandas as pd
data = { 'Name': ['Jai', 'Princ', 'Gaurav', 'Anuj'],
         'Age': [27, 24, 22, 32],
         'Address': ['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
         'Qualification': ['Msc', 'MA', 'MCA', 'Phd']
         }
df = pd.DataFrame(data)
df2= df.drop([1,2], axis=0) # Axis=0 is Row
print(df2)
```

Output:

```
        Name   Age     Address        Qualification
0       Jai    27      Delhi          Msc
3       Anuj   32      Kannauj        Phd
```

Joining DataFrames with append() function:

```
import pandas as pd
df1 = pd.DataFrame([[1, 2], [3, 4]], columns = ['a','b'])
df2 = pd.DataFrame([[5, 6], [7, 8]], columns = ['a','b'])
df1 = df1.append(df2)
print (df1)
```

Output;

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

```
1  7  8
```

Joining DataFrames concat() function:

11.1 Example (Default index):

```
import pandas as pd
data1 = { 'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],
          'Age':[27, 24, 22, 32],
          'Address':['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj'],
          'Qualification':['Msc', 'MA', 'MCA', 'Phd']}

data2 = { 'Name':['Abhi', 'Ayushi', 'Dhiraj', 'Hitesh'],
          'Age':[17, 14, 12, 52],
          'Address':['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj'],
          'Qualification':['Btech', 'B.A', 'Bcom', 'B.hons']}

df1 = pd.DataFrame(data1)
df2 = pd.DataFrame(data2)

res1 = pd.concat( [df1,df2], axis=0) #Works on Rows for each Column
print(res1)
print(“”)
res2 = pd.concat([df1,df2], axis=1) # Works on Columns for each Row
print(res2)
```

Output:

```
        Name      Age   Address      Qualification
0       Jai       27    Nagpur       Msc
1       Princi    24    Kanpur       MA
2       Gaurav    22    Allahabad    MCA
3       Anuj      32    Kannuaj      Phd
0       Abhi      17    Nagpur       Btech
1       Ayushi    14    Kanpur       B.A
2       Dhiraj    12    Allahabad    Bcom
3       Hitesh    52    Kannuaj      B.hons

[8 rows x 4 columns]
        Name Age      Address    Qualification      Name Age     Address      Qualification
0       Jai     27      Nagpur       Msc            Abhi    17   Nagpur       Btech
1       Princi  24    Kanpur        MA              Aydhi   14   Kanpur       B.A
2        Gaurav 22    Allahabad     MCA             Dhiraj  12   Allahabad    Bcom
3       Anuj    32    Kannuaj       Phd             Hitesh  52   Kannuaj      B.hons
[4 rows x 8 columns]
```

NOTE:

- axis=0 acts on all the ROWS in each COLUMN
- axis=1 acts on all the COLUMNS in each ROW
- by default axis=0

Merging data within DataFrames using merge():

```
import pandas as pd
x = pd.DataFrame({'id':[1,2],'Name': ['anil', 'vishal'],  'subject_id':['sub1','sub2']})
y = pd.DataFrame({'id':[1,2],'Name': ['sumer', 'salil'], 'subject_id':['sub2','sub4']})
print(pd.merge(x , y , on='id'))
```

Output:

```
   id   Name_x   subject_id_x     Name_y    subject_id_y
0   1  anil         sub1                  sumer            sub2
1   2  vishal      sub2                   salil               sub4
```

11. Boolean Indexing:

Boolean indexing helps us to select the data from the DataFrames using a boolean vector. We need a DataFrame with a boolean index to use the boolean indexing. Let's see how to achieve the Boolean. indexing.

- Create a dictionary of data.
- Convert it into a DataFrame object with a boolean index
- Now, access the data using boolean indexing.

```
import pandas as pd
data = {
        'Name': ['Hafza', 'Srikanth', 'Rakesh'],
        'Age': [19, 20, 19]
        }
df = pd.DataFrame(data, index = [True, False, True]) # Creating a DataFrame with boolean index
print(df)
print(“”) #Prints a blank line
print(df.loc[True])
print(“”)
print(df.iloc[1])
```

Output:

```
           Name          Age
True       Hafza         19
False      Srikanth      20
True       Rakesh        19

           Name          Age
True       Hafza         19
True       Rakesh        19

Name Srikanth
Age 20
```

Selecting data from DataFrame with boolean indexing:

```
import pandas as pd
dict = {'name':['Mohak', "Freya", "Roshni"],
       'degree': ["MBA", "BCA", "M.Tech"],
        'score':[90, 40, 80]}
# creating a dataframe with boolean index
df = pd.DataFrame(dict, index = [True, False, True])  # accessing a dataframe using .loc[] function
print(df.loc[True])                                   #it will return rows of Mohak and Roshni only(matching true only)
```

Output:

```
         name     degree  score
True   Mohak      MBA      90
True   Roshni     M.Tech   80
```

Python | Pandas *Series.where()*

Python is a great language for doing data analysis, primarily because of the fantastic ecosystem of data-centric python packages. Pandas is one of those packages and makes importing and analyzing data much easier.

Pandas series is a One-dimensional ndarray with axis labels. The 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.

Pandas Series.where() function replace values where the input condition is False for the given Series object. It takes another object as an input which will be used to replace the value from the original object.

Syntax: Series.where(cond, other=nan, inplace=False, axis=None, level=None, errors='raise', try_cast=False, raise_on_error=None)

Parameters :
cond : boolean NDFrame, array-like, or callable
other : scalar, NDFrame, or callable
inplace : boolean, default False
axis : int, default None
level : int, default None
errors : str, {'raise', 'ignore'}, default raise
try_cast : boolean, default False

Example : Print the series more than 50

```
import pandas as pd
s = pd.Series([10,20,30,40,50,60,70,80,90,100])
a=s.where(s > 50)
print(a)
```

Output:

```
0      NaN
1      NaN
2      NaN
3      NaN
4      NaN
5     60.0
```

```
6     70.0
7     80.0
8     90.0
9    100.0
dtype: float64
```

Example : Print the series has a value 50

```
import pandas as pd
s = pd.Series([10,20,30,40,50,60,70,80,90,100])
a=s.where(s == 50)
print(a)
```

Output:

```
0     NaN
1     NaN
2     NaN
3     NaN
4     50.0
5     NaN
6     NaN
7     NaN
8     NaN
9     NaN
dtype: float64
```

Example#3: Print the series less than 50

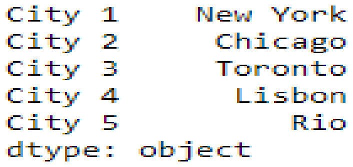
```
import pandas as pd
s = pd.Series([10,20,30,40,50,60,70,80,90,100])
a=s.where(s <= 50)
print(a)
```

Output:

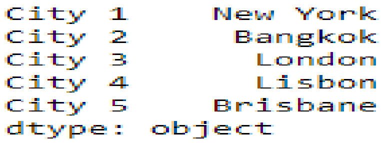
```
0    10.0
1    20.0
2    30.0
3    40.0
4    50.0
5     NaN
6     NaN
7     NaN
8     NaN
9     NaN
dtype: float64
```

Example #1: Use Series.where() function to replace values in the given Series object with some other value when the passed condition is not satisfied.

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

# Creating the First Series
sr1 = pd.Series(['New York', 'Chicago', 'Toronto', 'Lisbon', 'Rio'])

# Creating the row axis labels
sr1.index = ['City 1', 'City 2', 'City 3', 'City 4', 'City 5']

# Print the series
print(sr1)

# Creating the second Series
sr2 = pd.Series(['New York', 'Bangkok', 'London', 'Lisbon', 'Brisbane'])

# Creating the row axis labels
sr2.index = ['City 1', 'City 2', 'City 3', 'City 4', 'City 5']

# Print the series
print(sr2)
```

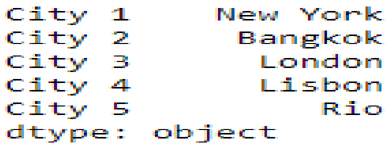


```
City 1    New York
City 2     Chicago
City 3     Toronto
City 4      Lisbon
City 5         Rio
dtype: object
```



```
City 1    New York
City 2     Bangkok
City 3      London
City 4      Lisbon
City 5    Brisbane
dtype: object
```

```
# replace the values
sr1.where(sr1 == 'Rio', sr2)
```

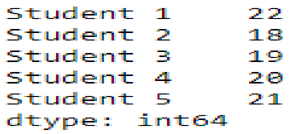


```
City 1    New York
City 2     Bangkok
City 3      London
City 4      Lisbon
City 5         Rio
dtype: object
```

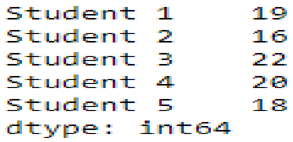
Example #2 : Use Series.where() function to replace values in the given Series object with some other value when the passed condition is not satisfied.

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

# Creating the First Series
sr1 = pd.Series([22, 18, 19, 20, 21])
```

```
# Creating the row axis labels
sr1.index = ['Student 1', 'Student 2', 'Student 3', 'Student 4', 'Student 5']

# Print the series
print(sr1)

# Creating the second Series
sr2 = pd.Series([19, 16, 22, 20, 18])

# Creating the row axis labels
sr2.index = ['Student 1', 'Student 2', 'Student 3', 'Student 4', 'Student 5']

# Print the series
print(sr2)
```

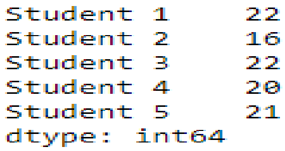


```
Student 1     22
Student 2     18
Student 3     19
Student 4     20
Student 5     21
dtype: int64
```



```
Student 1     19
Student 2     16
Student 3     22
Student 4     20
Student 5     18
dtype: int64
```

```
# replace the values
sr1.where(sr1 >20, sr2)
```



```
Student 1     22
Student 2     16
Student 3     22
Student 4     20
Student 5     21
dtype: int64
```

12.1 Accessing/importing/reading csv File in Pandas

A CSV is a **comma-separated values** file, which allows data to be saved in a tabular format. CSVs look like a garden-variety spreadsheet but with a **.**csv extension. CSV files can be used with most any spreadsheet program, such as Microsoft Excel or Google Spreadsheets.

emp1.xlsx

Emp ID	Emp Name	Emp Role
1	Pankaj Kumar	Admin
2	David Lee	Editor
3	Lisa Ray	Author

emp1.csv

Emp ID,   Emp Name,   Emp Role
1,   Pankaj Kumar,   Admin
2,   David Lee,   Editor 3,
3,   Lisa Ray,   Author

```
import pandas as pd
df = pd.read_csv(c:/mydata/class12/'emp1.csv')
print(df)
```

Output:

```
        Emp ID     Emp Name          Emp Role
0       1          Pankaj Kumar      Admin
1       2          David Lee         Editor
2       3          Lisa Ray          Author
```

```
import pandas as pd
# importing Data here from remote source of .csv file
df = pd.read_csv('emp1.csv')
x=[50000 , 55000, 60000]
df['Salary'] = x
print(df)
```

Output:

```
       Emp ID      Emp Name        Emp Role          Salary
0      1           Pankaj Kumar    Admin             50000
1      2           David Lee       Editor            55000
2      3           Lisa Ray        Author            60000
```

12.2 Exporting DataFrame to csv File

```
import pandas as pd


df.to_csv('emp2.csv', index=True)
df.to_csv('emp2.csv', index=False) # Without indexing
```

Output: (index=*True*)

```
     Emp ID                   Emp Name        Emp Role               Salary
0  0  1                       Pankaj Kumar    Admin                  50000
1  1  2                       David Lee       Editor                 55000
2  2  3                       Lisa Ray        Author                 60000
```

Output: (index=*False*)

```
    Emp ID                    Emp Name        Emp Role               Salary
0     1                       Pankaj Kumar    Admin                  50000
1     2                       David Lee       Editor                 55000
2     3                       Lisa Ray        Author                 60000
```

13. DataFrame Functions:

Example of a DataFrame:

```
import pandas as pd
x = { 'Name': ['Jai', 'Prince', 'Gaurav', 'Anuj'],
          'Age': [27, 24, 22, 32],
          'Address': ['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
                'Qualification': ['Msc', 'MA', 'MCA', 'Phd'] }
    df = pd.DataFrame(x)
     print(df)
```

Output:

```
    Name        Age     Address        Qualification
0 Jai           27      Delhi          Msc
1 Prince        24      Kanpur         MA
2 Gaurav        22      Allahabad      MCA
3   Anuj        32      Kannauj        Phd
```

13.1 Example: max()

```
     import pandas as pd
     x = { 'Name': ['Jai', 'Prince', 'Gaurav', 'Anuj'],
                'Age': [27, 24, 22, 32],
                'Address': ['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
                'Qualification': ['Msc', 'MA', 'MCA', 'Phd'] }
    df = pd.DataFrame(x)
     print(df['Age'].max())
```

Output:

```
32
```

13.2 Example: min()

```
     import pandas as pd
     x = { 'Name': ['Jai', 'Prince', 'Gaurav', 'Anuj'],
                'Age': [27, 24, 22, 32],
                'Address': ['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
                'Qualification': ['Msc', 'MA', 'MCA', 'Phd'] }
     df = pd.DataFrame(x)
     print(df['Age'].min())
```

Output:

```
22
```

13.3 Example: count() # It counts the number of values present in the column

```
     import pandas as pd
     x = { 'Name': ['Jai', 'Prince', 'Gaurav', 'Anuj'],
                'Age': [27, 24, 22, 32],
                'Address': ['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
                'Qualification': ['Msc', 'MA', 'MCA', 'Phd'] }
    df = pd.DataFrame(x)
     print(df['Age'].count())
```

Output:

```
4
```

13.4 Example: sum()   # Finds the total/addition of the values of the column

```
     import pandas as pd
     x = { 'Name': ['Jai', 'Prince', 'Gaurav', 'Anuj'],
                  'Age': [27, 24, 22, 32],
                  'Address': ['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
                  'Qualification': ['Msc', 'MA', 'MCA', 'Phd'] }
        df = pd.DataFrame(x)
         print(df['Age'].sum())
```

```
Output:
105
```

15.2 Altering / Renaming Series Label

```
import pandas as pd
import numpy as np
data = np.array([54,76,88,99,34])
s1 = pd.Series(data,index=['a','b','c','d','e'])
print (s1)
s2=s1.rename(index={'a':0, 'b':1})
print(“After reindexing: \n“, s2)
```

```
OUTPUT
a       54
b       76
c       88
d       99
e       34

dtype: int32

 0       54
 1       76
 c       88
 d       99
 e       34
         dtype: int32
```

15.3 Re-indexing Rows in Pandas Dataframe:

```
import pandas as pd
import numpy as np
table = {"name": ['vishal', 'anil', 'mayur', 'viraj','mahesh'],
'age':[15, 16, 15, 17,16],
'weight': [51, 48, 49, 51,48],
'height': [5.1, 5.2, 5.1, 5.3,5.1],
}
d = pd.DataFrame(table)
print("DATA OF DATAFRAME")
print(d)
print("DATA OF DATAFRAME AFTER REINDEX")
df=d.reindex([2,1, 0,4,3])
 print(df)
```

Output:

```
DATA OF DATAFRAME
         name     age  weight       height
0        vishal   15    51           5.1
1        anil     16    48           5.2
2        mayur    15    49           5.1
3        viraj    17    51           5.3
4        mahesh   16    48           5.1
```

```
DATA OF DATAFRAME AFTER REINDEX
         name     age weight  height
2        mayur   15    49    5.1
1        anil     16    48    5.2
0        vishal   15    51    5.1
4        mahesh   16    48    5.1
3        viraj    17    51    5.3
```

15.4 Re-Indexing Columns in Pandas DataFrame:

```
import pandas as pd
import numpy as np
table = {"name": ['vishal', 'anil', 'mayur', 'viraj','mahesh'],
'age':[15, 16, 15, 17,16],
'weight': [51, 48, 49, 51,48],
'height': [5.1, 5.2, 5.1, 5.3,5.1],
}
d = pd.DataFrame(table)
print("DATA OF DATAFRAME=>")
print(d)
print()
print("DATA OF DATAFRAME AFTER REINDEX=>")
df=d.reindex(columns=['name','weight','age'])
print(df)
```

Output:

```
DATA OF DATAFRAME=>
    name          age  weight       height
0   vishal        15    51           5.1
1   anil          16    48           5.2
2   mayur         15    49           5.1
3   viraj         17    51           5.3
4   mahesh        16    48           5.1
```

```
DATA OF DATAFRAME AFTER REINDEX=>
    name     weight age
0   vishal   51       15
1   ani      48       16
2   mayur    49       15
3   viraj    51       17
4  mahesh    48       16
```

17. Data Visualisation:

There are various types of Data Visualisation techniques, are as follows:

- Histogram
- Line plot
- Bar chart

17.1 Histogram:

A histogram is an accurate graphical representation of the distribution of numerical data. It is an estimate of the probability distribution of a continuous variable (quantitative variable) and was first introduced by Karl Pearson. It is a kind of bar graph. To construct a histogram, the first step is to “bin” the range of values — that is, divide the entire range of values into a series of intervals — and then count how many values fall into each interval. The bins are usually specified as consecutive, non-overlapping intervals of a variable. The bins (intervals) must be adjacent, and are often (but are not required to be) of equal size. A histogram shows the number of occurrences of different values in a dataset.

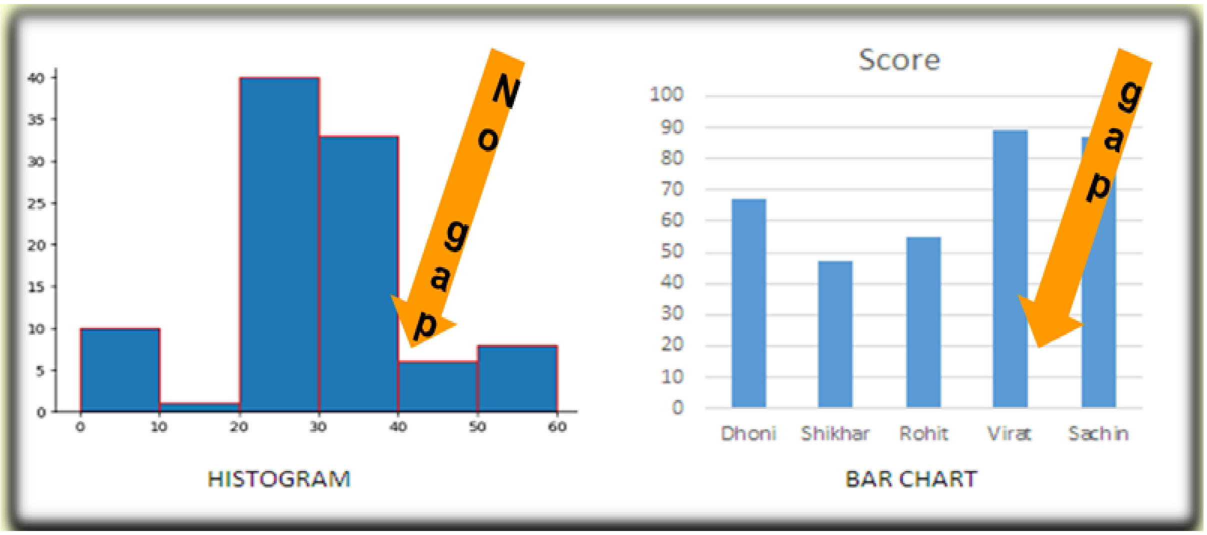
Histogram in Python:

Drawing a histogram in Python is very easy. All we have to do is code for 3-4 lines of code. But complexity is involved when we are trying to deal with live data for visualization.
To draw histogram in python following concepts must be clear.
Title: To display heading of the histogram.
Color : To show the colour of the bar.
Axis: y-axis and x-axis.
Data: The data can be represented as an array.
Height and width of bars. This is determined based on the analysis.
The width of the bar is called **bin** or intervals. Default size = 10
Border colour: To display border colour of the bar.

Matplotlib is the whole python package/ library used to create 2D graphs and plots by using python scripts. pyplot is a module in matplotlib, which supports a very wide variety of graphs and plots namely - histogram, bar charts, power spectra, error charts etc. It is used along with NumPy to provide an environment for MatLab.

Pyplot provides the state-machine interface to the plotting library in matplotlib.It means that figures and axes are implicitly and automatically created to achieve the desired plot. For example, calling plot from pyplot will automatically create the necessary figure and axes to achieve the desired plot. Setting a title will then automatically set that title to the current axes object.The pyplot interface is generally preferred for non-interactive plotting (i.e., scripting).

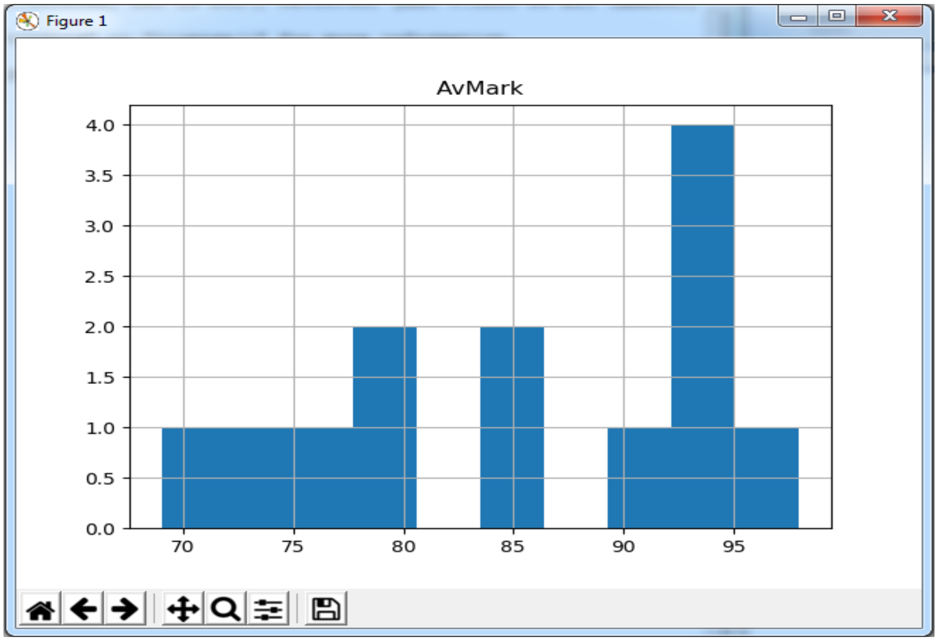
Difference between a histogram and a bar chart / graph:
A bar chart majorly represents categorical data (data that has some labels associated with it), they are usually represented using rectangular bars with lengths proportional to the values that they represent. While histograms on the other hand, is used to describe distributions. Given a set of data, are their distributions.



17.1.1 Program of Histogram in python:

```
import pandas as pd
import matplotlib.pyplot as plt # MAT-PLOT-LIB <DOT>PY-PLOT
x={"AvMark": [90,95,95, 93, 94,78,69,85,74,86,75,79,98] }
df=pd.DataFrame(x)
print(df)
df.hist() # hist() Converts the df in Histogram (Chart/Bar)
plt.show() # Prints the histogram
```

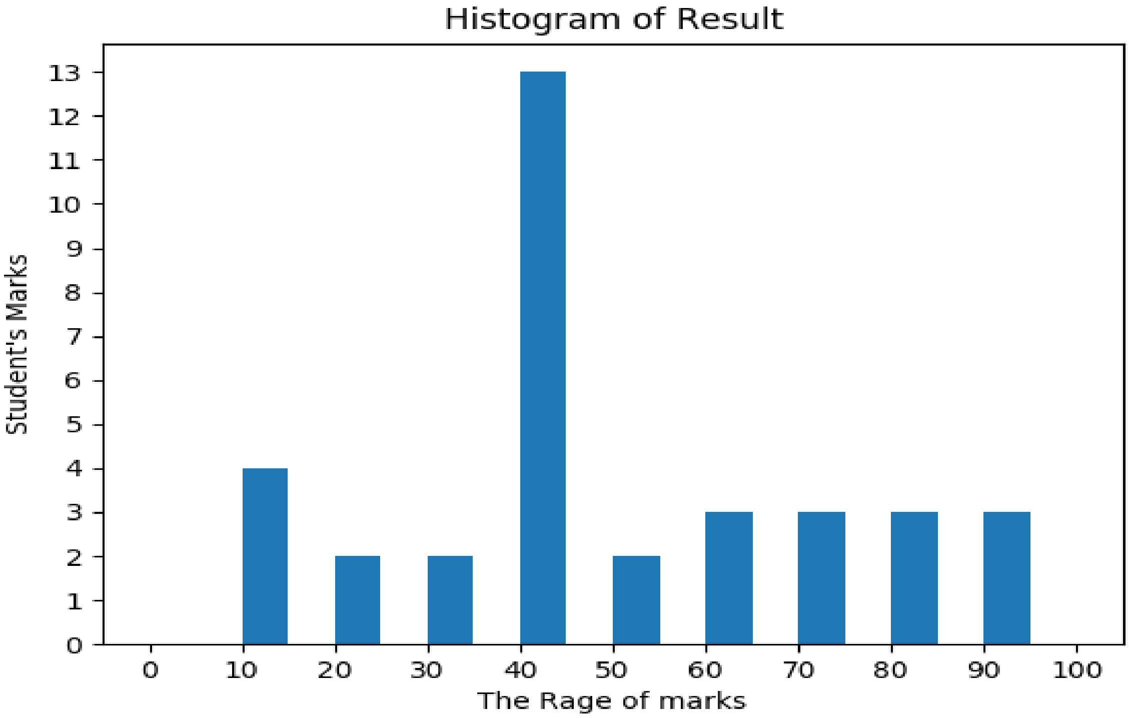
```
    AvMark
0       90
1       95
2       95
3       93
4       94
5       78
6       69
7       85
8       74
9       86
10      75
11      79
```



12 98

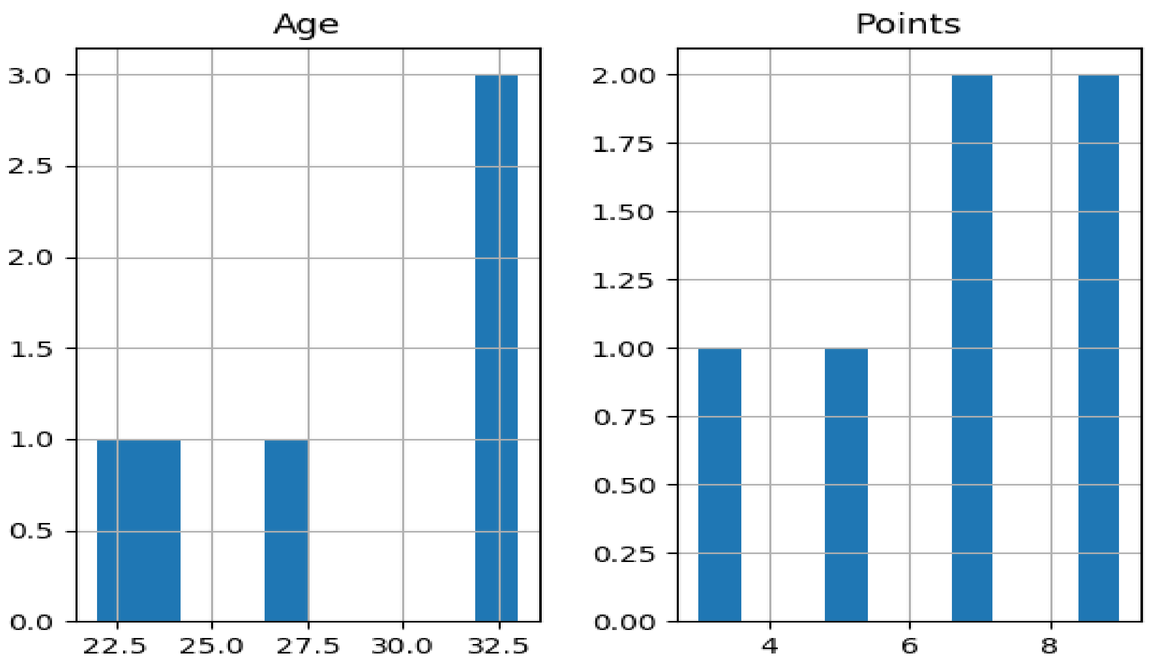
17.2.1a Example: *Plot a Histogram to show the various frequencies (bin with distribution) of given marks*

```
import matplotlib.pyplot as plt
marks=[90, 95, 80, 80, 80, 91, 75, 75, 77, 50, 65,65,55, 69, 74, 75, 85, 85 ]
range=[0,10,20,30,40,50,60,70,80,90,100]   # Works as Bin value
plt.xticks(range)     # Fix the values in x-axis
plt.yticks([0,1,2,3,4,5,6,7,8,9,10, 11,12,13,14,15])   # Fix the values in y-axis
plt.xlabel("The Rage of marks")
plt.ylabel("Student's Marks")
plt.title("Histogram of Result")
plt.hist(marks, range, width=5)     # Width of each bar has been reduced
plt.show()
```



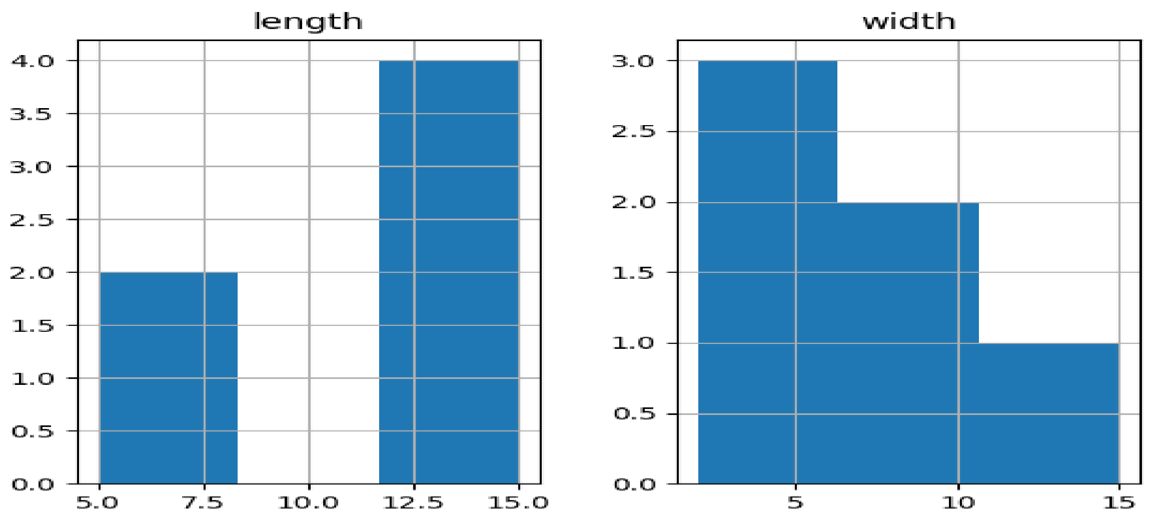
17.2.1b Example:

```
import pandas as pd
import matplotlib.pyplot as plt
x = {'Age': [27, 24, 22, 32, 33, 32], 'Points': [3,5,7, 9, 7, 9] }
df = pd.DataFrame(x)
hist = df.hist()
plt.show()
```



17.1.2 Example:

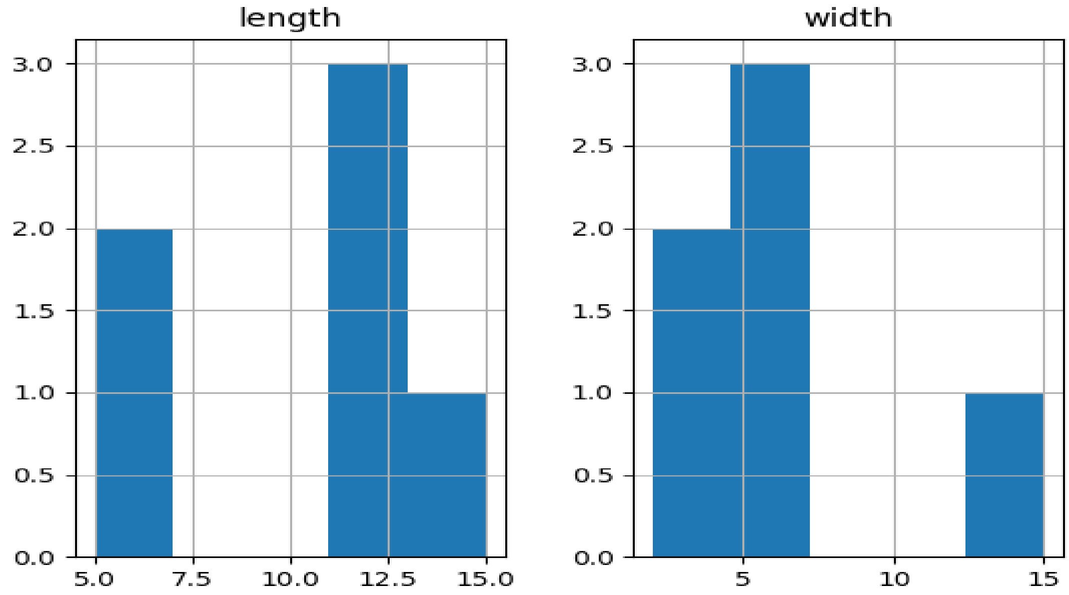
```
import pandas as pd
import matplotlib.pyplot as plt
data={ 'length': [15, 5, 12, 12, 12, 5],        'width': [7, 2, 15, 2, 5, 7]  }
df = pd.DataFrame(data)
hist = df.hist(bins=3) # Bin is 3
plt.show()
```



17.1.3 Example:

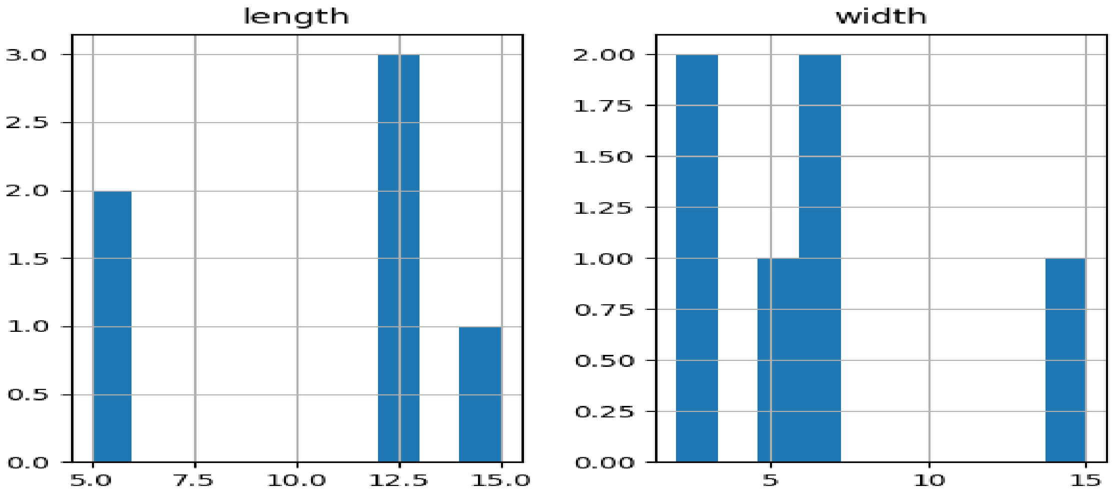
```
import pandas as pd
import matplotlib.pyplot as plt
data={'length': [15, 5, 12, 12, 12, 5],        'width': [7, 2, 15, 2, 5, 7]  }
df = pd.DataFrame(data)
hist = df.hist(bins=5)        # Bin is 5
```

```
plt.show()
```



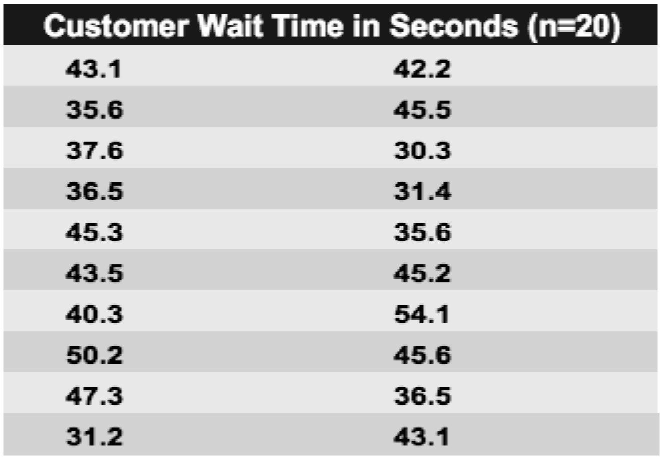
17.1.4 Example:

```
import pandas as pd
import matplotlib.pyplot as plt
data={'length': [15, 5, 12, 12, 12, 5], 'width': [7, 2, 15, 2, 5, 7] }
df = pd.DataFrame(data)
hist = df.hist(bins=10)      # Bin is 10
plt.show()
```



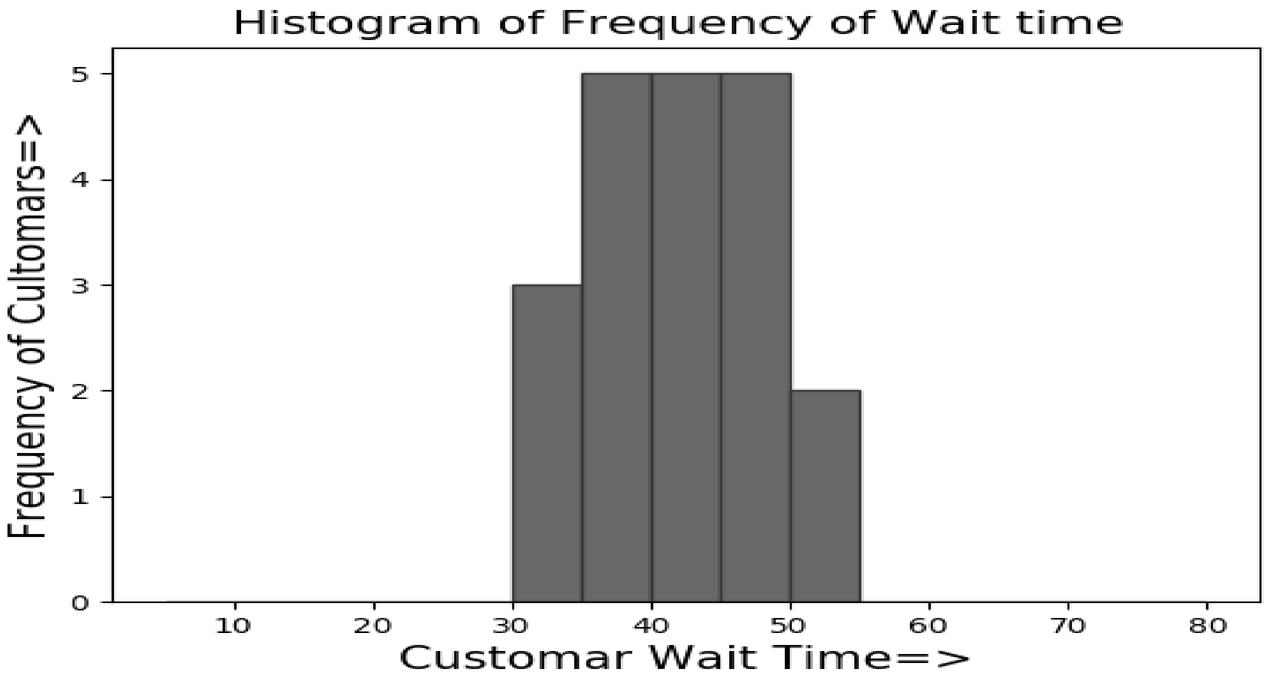
17.1.5 Example of a Histogram

Jeff is the branch manager at a local bank. Recently, Jeffs been receiving customer feedback saying that the wait times for a client to be served by a customer service representative are too long. Jeff decides to



Customer Wait Time in Seconds (n=20)	
43.1	42.2
35.6	45.5
37.6	30.3
36.5	31.4
45.3	35.6
43.5	45.2
40.3	54.1
50.2	45.6
47.3	36.5
31.2	43.1

observe and write down the time spent by each customer on waiting. Here are his findings from observing and writing down the wait times spent by 20 customers:



17.1.6 Histogram Data - Example

The corresponding histogram with 5-second bins (5-second intervals) would look as follows:

We can see that:

There are **3** customers waiting between 1 and 35 seconds
There are **5** customers waiting between 1 and 40 seconds
There are **5** customers waiting between 1 and 45 seconds
There are **5** customers waiting between 1 and 50 seconds
There are **2** customers waiting between 1 and 55 seconds
43.1 , 36.6, 37.6, 36.5, 45.3, 43.5, 40.3, 50.2, 47.3, 31.2, 42.2, 45.5, 30.3, 31.4, 35.6, 45.2, 54.1, 45.6, 36.5, 43.1

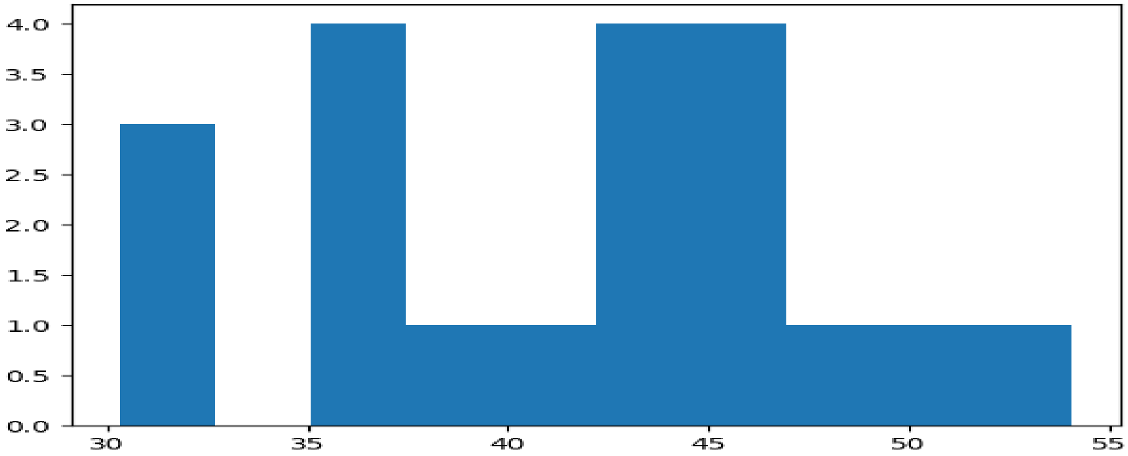
Syntax:

```
import matplotlib.pyplot as plt
x = [value1, value2, value3,. ..]
plt.hist(x, bins = number of bins)
plt.show()
```

17.1.7 Example:

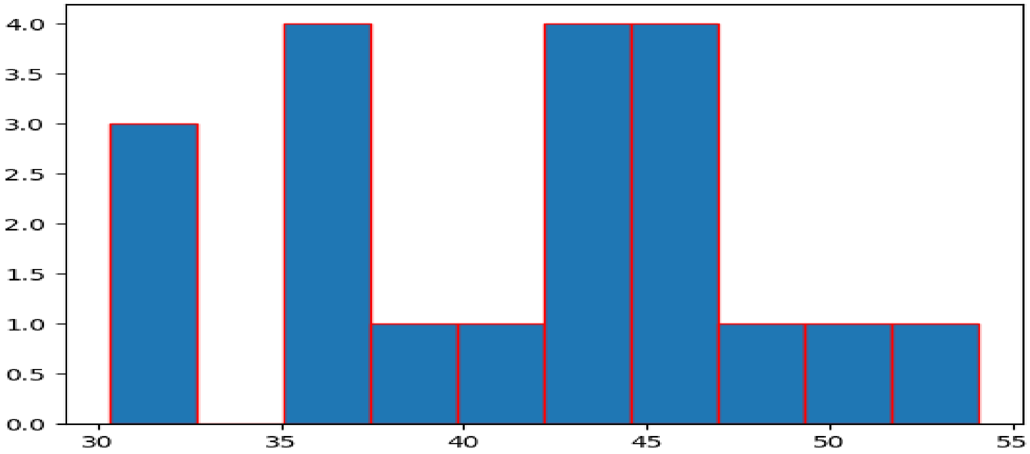
```
import matplotlib.pyplot as plt
wt=[43.1 , 36.6, 37.6, 36.5, 45.3, 43.5, 40.3, 50.2, 47.3, 31.2, 42.2, 45.5, 30.3, 31.4, 35.6,
45.2, 54.1, 45.6, 36.5, 43.1]
```

```
plt.hist(wt, 10)
plt.show()
```



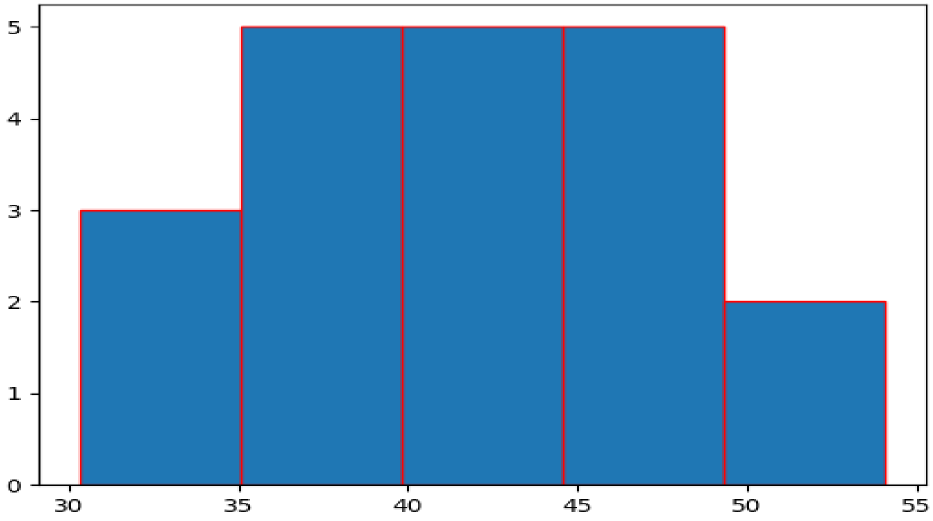
17.1.8 Example: (No bins i.e. default bins=10)

```
import matplotlib.pyplot as plt
wt=[43.1, 36.6, 37.6, 36.5, 45.3, 43.5, 40.3, 50.2, 47.3, 31.2, 42.2, 45.5, 30.3, 31.4, 35.6, 45.2,
    54.1, 45.6, 36.5, 43.1]
plt.hist(wt, edgecolor='red')
plt.show()
```



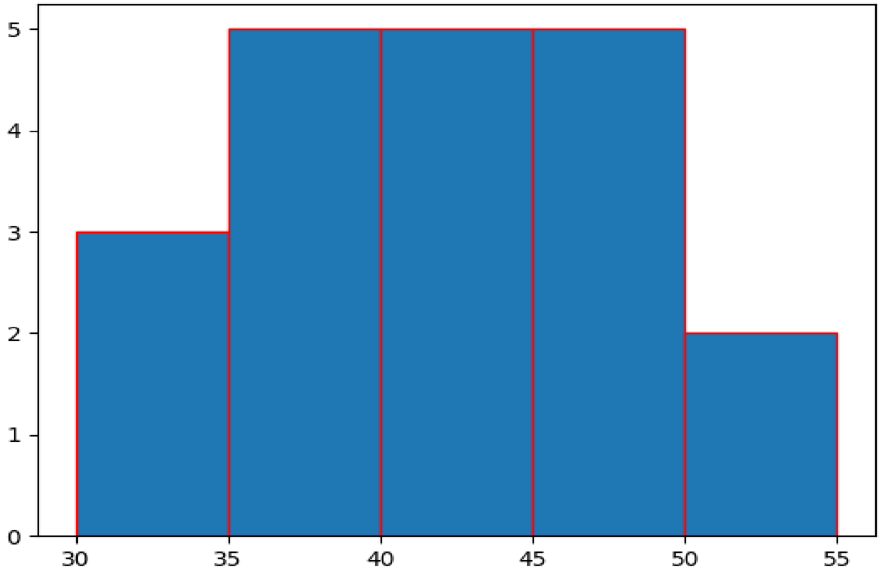
17.1.9 Example:

```
import matplotlib.pyplot as plt
wt=[43.1 , 36.6, 37.6, 36.5, 45.3, 43.5, 40.3, 50.2, 47.3, 31.2, 42.2, 45.5, 30.3, 31.4, 35.6, 45.2,
    54.1, 45.6, 36.5, 43.1]
plt.hist(wt, bins=5, edgecolor='red')
plt.show()
```



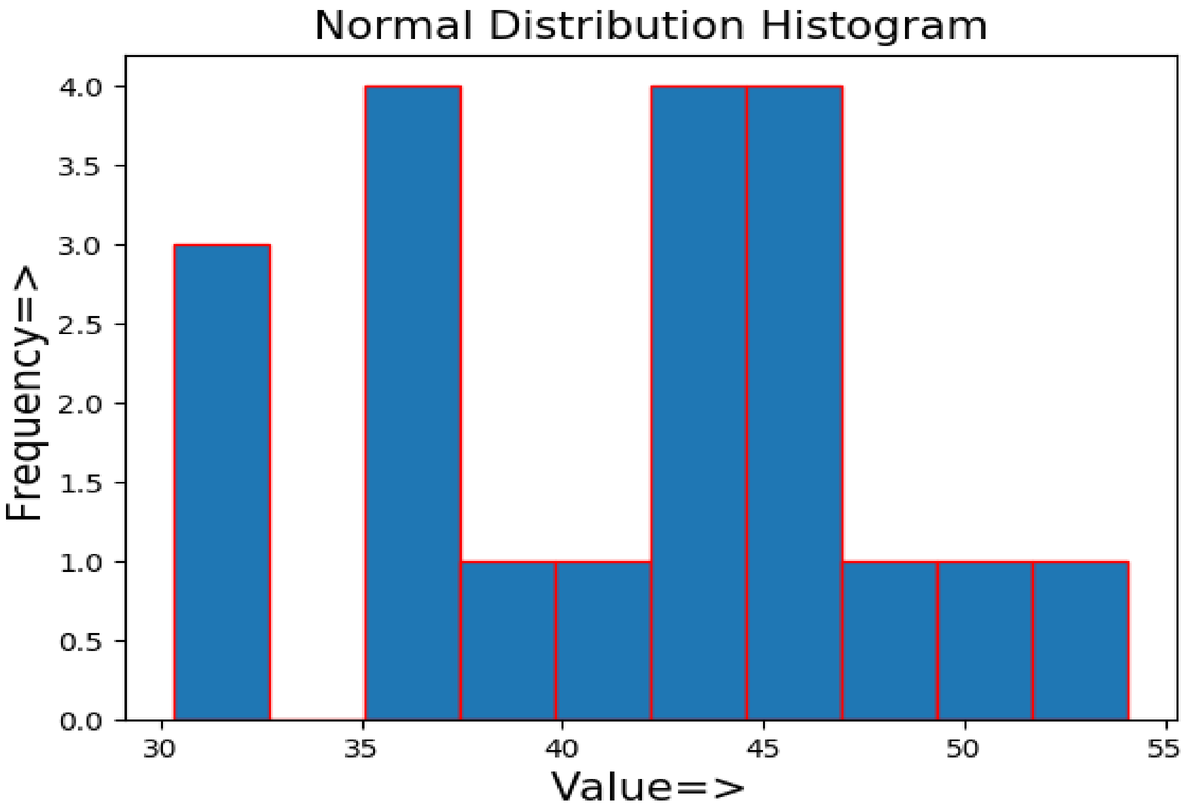
17.1.9 Example:

```
import matplotlib.pyplot as plt
wt=[43.1 , 36.6, 37.6, 36.5, 45.3, 43.5, 40.3, 50.2, 47.3, 31.2, 42.2, 45.5, 30.3, 31.4,
    35.6, 45.2, 54.1, 45.6, 36.5, 43.1]
bn=[30,35,40,45,50,55]
plt.hist(wt, bn, edgecolor='red')
plt.show()
```



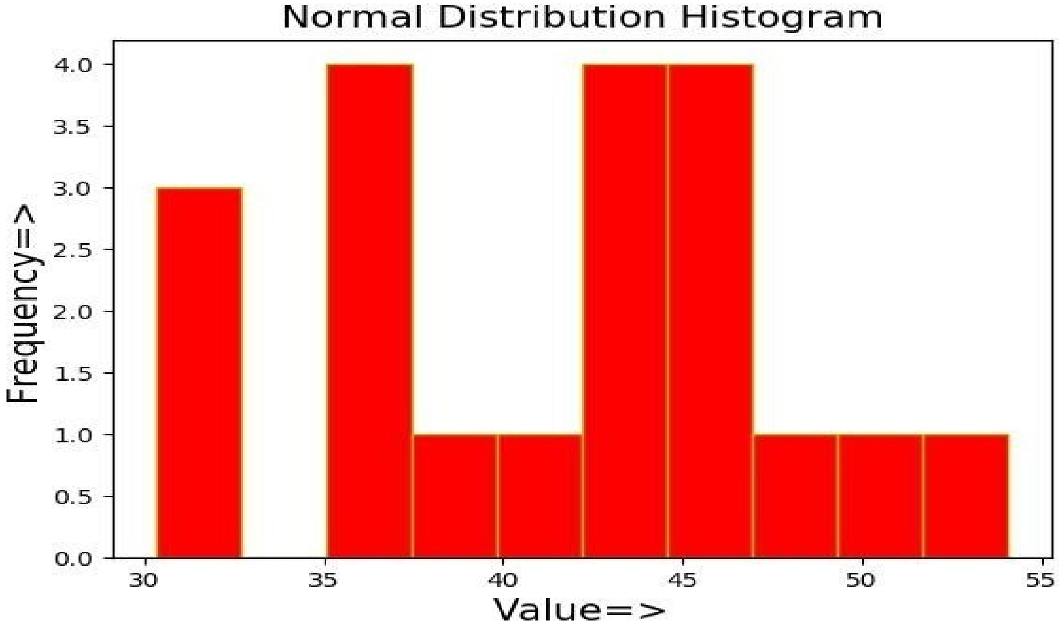
17.1.10 Example:

```
import matplotlib.pyplot as plt
wt=[43.1, 36.6, 37.6, 36.5, 45.3, 43.5, 40.3, 50.2,
47.3, 31.2, 42.2, 45.5, 30.3, 31.4, 35.6, 45.2, 54.1, 45.6, 36.5, 43.1]
plt.hist(wt, bins=10, edgecolor='red')
plt.xlabel('Value=>',fontsize=15)
plt.ylabel('Frequency=>',fontsize=15)
plt.title('Normal Distribution Histogram', fontsize=15)
plt.show()
```



Example: With customization (Label, Titile, Font size, Edge colour, Face colour.)

```
import matplotlib.pyplot as plt
wt=[43.1, 36.6, 37.6, 36.5, 45.3, 43.5, 40.3, 50.2, 47.3, 31.2, 42.2, 45.5, 30.3, 31.4, 35.6, 45.2, 54.1, 45.6,
36.5, 43.1]
plt.hist(wt, bins=10, edgecolor='y', facecolor='r')
plt.xteicks()
plt.xlabel('Value=>', fontsize=15)
plt.ylabel('Frequency=>', fontsize=15)
plt.title('Normal Distribution Histogram', fontsize=15)
plt.savefig('Wait_time.jpeg') # savefig() func. Saves the graph in picture format (jpeg, png…)
plt.show()
```

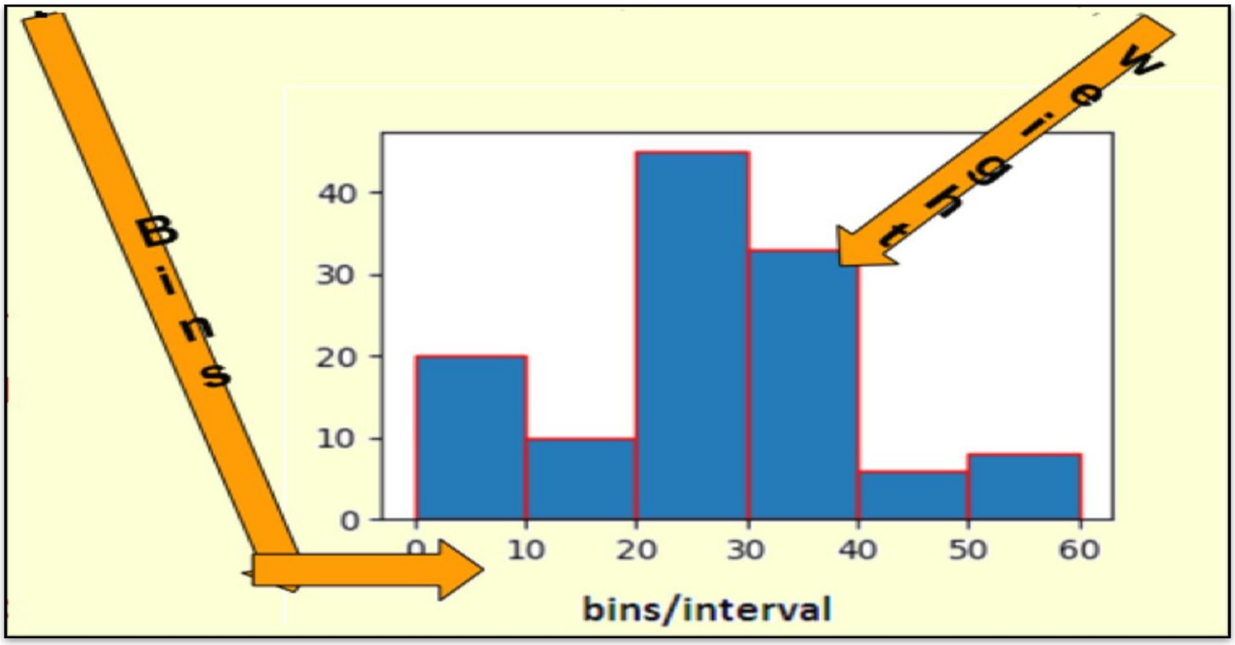


17.1.11 Example:

```
import numpy as np
import matplotlib.pyplot as plt
plt.hist([5,15,25,35,45,55], bins=[0,10,20,30,40,50, 60], weights=[20,10,45,33,6,8], edgecolor="red")
plt.show()
```

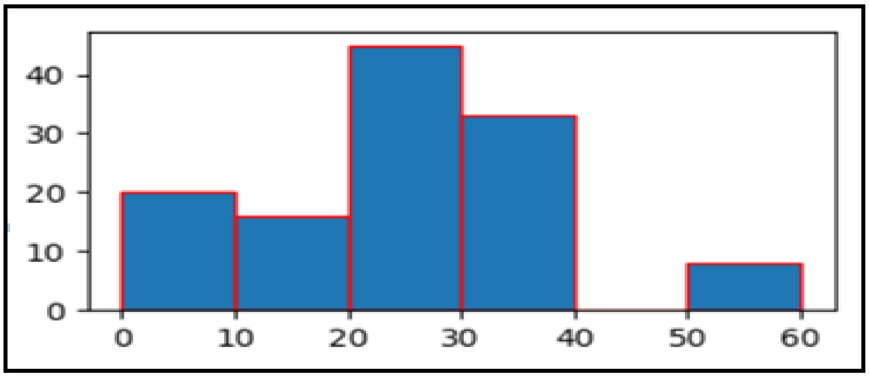
# First argument of hist() method is position (x , y Coordinate) of weight, where weight is to be displayed. No of coordinates must match No of weight otherwise error will generate
# Second argument is interval / Bins
# Third argument is weight for bars
# Fourth argument is the Border-colour of each bar

Creating Histogram on array of data



For better understanding we develop the same program with minor change.

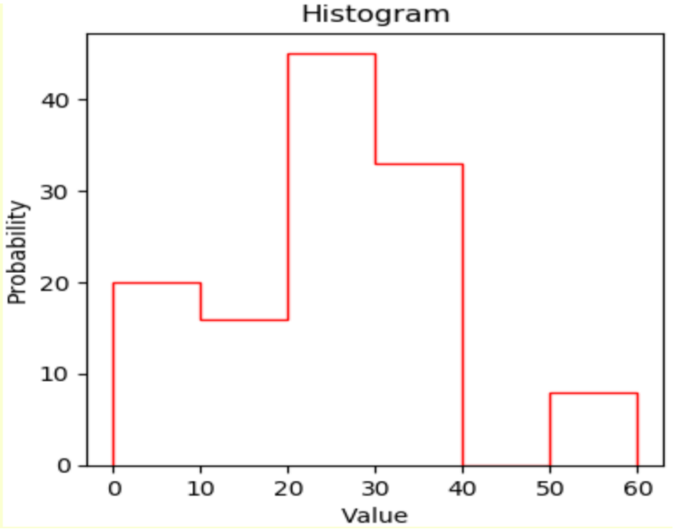
```
import matplotlib.pyplot as plt
plt.hist([5,15,25,35,15, 55], bins=[0,10,20,30,40,50, 60], weights=[20,10,45,33,6,8], edgecolor="red")
plt.show()
```



# At interval (bin) 40 to 50 no bar because we have not mentioned position from 40 to 50 in first argument(list) of hist method. Where as in interval 10 to 20 width is being Displayed as 16 (10+6 both weights are added) because 15 is twice In first argument.

Example (Histogram Type „step'):

```
import numpy as np
import matplotlib.pyplot as plt
data = [1,11,21,31,41]
plt.hist([5,15,25,35,15, 55], bins=[0,10,20,30,40,50, 60], weights=[20,10,45,33,6,8],
edgecolor="red", histtype='step') #plt.hist(data, bins=20, histtype='step')
plt.xlabel('Value')
plt.ylabel('Probability')
plt.title('Histogram')
plt.show()
```

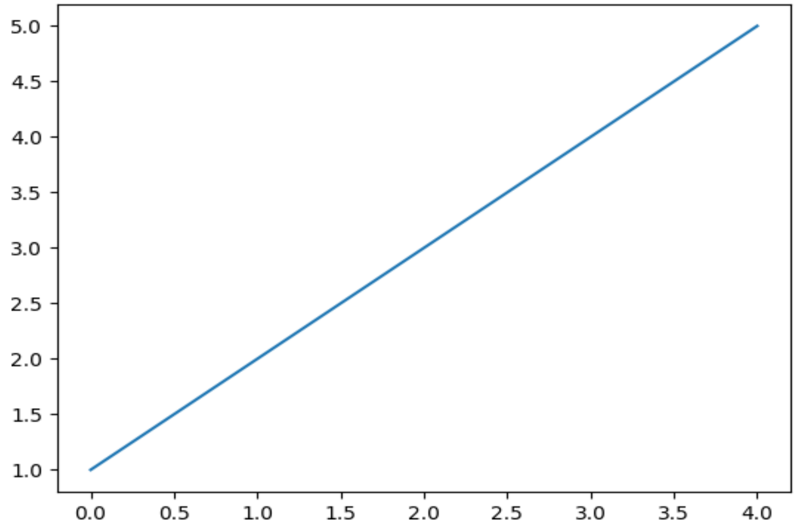


Note: Bin: *In histogram total range of data set (minimum to maximum) is divided into 8 to 15 equal parts. These equal parts are known as Bins or class-intervals.*

17.2 PyPlot Application to display a Line Chart:

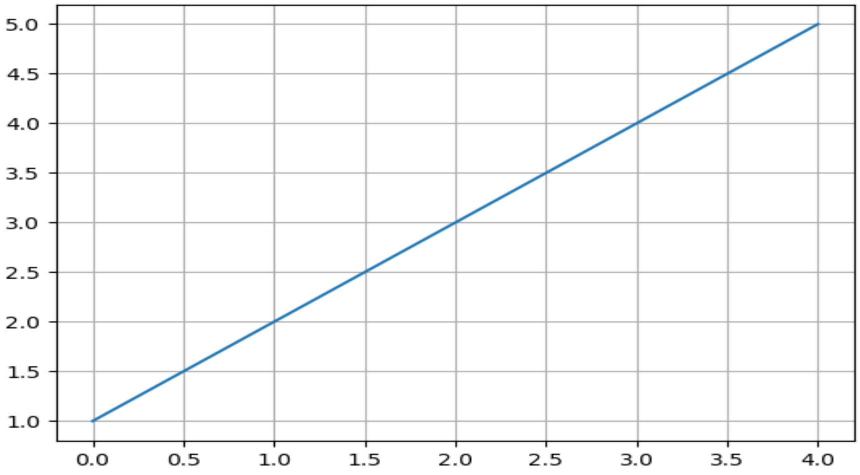
17.2.1 Example:

```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4,5])       # Generates or co-ordinates of the values in reference to Y-Axis
plt.show()
```



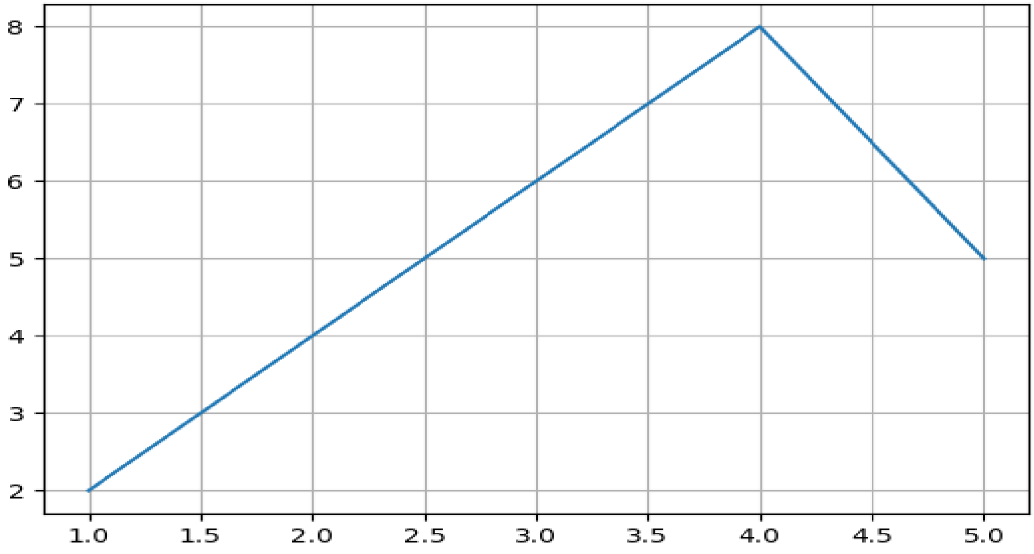
Example:

```
import matplotlib.pyplot as plt
plt.grid()                  # Reformat the layout with grid
plt.plot([1,2,3,4,5])       # Generates or co-ordinates of the values in reference to Y-Axis
plt.show()
```



17.2.2 Example:

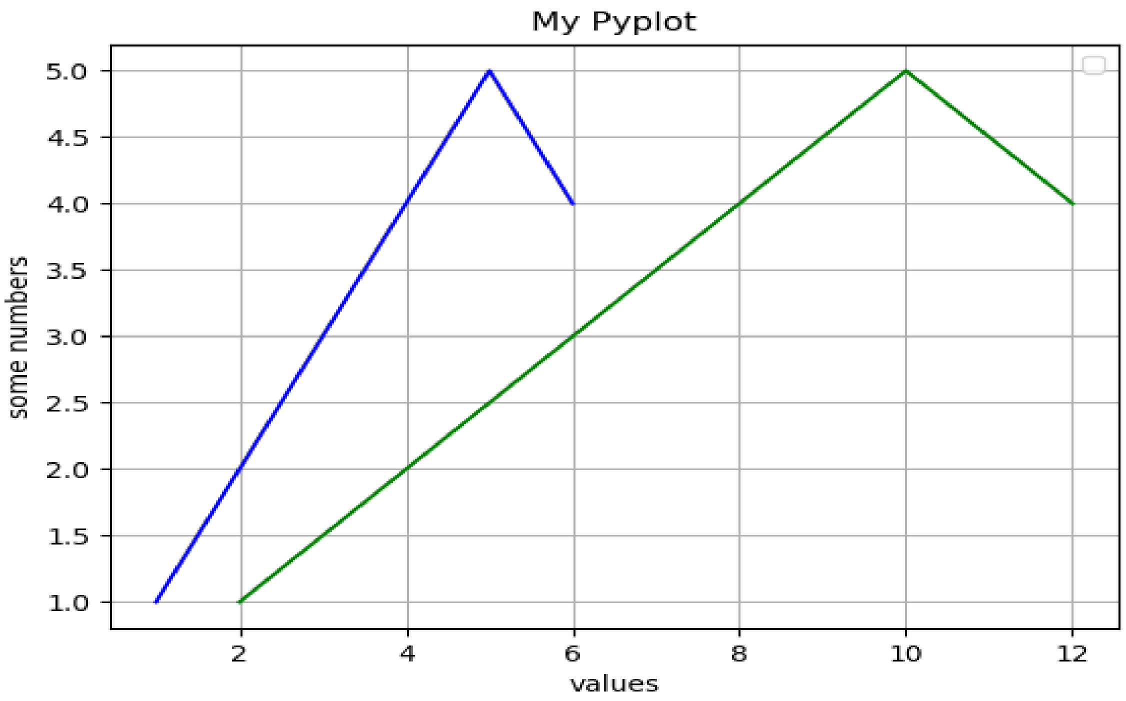
```
import matplotlib.pyplot as plt
plt.grid()           # Reformat the layout with grid
plt.plot([1,2,3,4,5] , [2,4,6,8,5]) # Generates or co-ordinates of the values in reference to Y-Axis
plt.show()
```



17.2.3 PyPlot application to display multiple lines

Example:

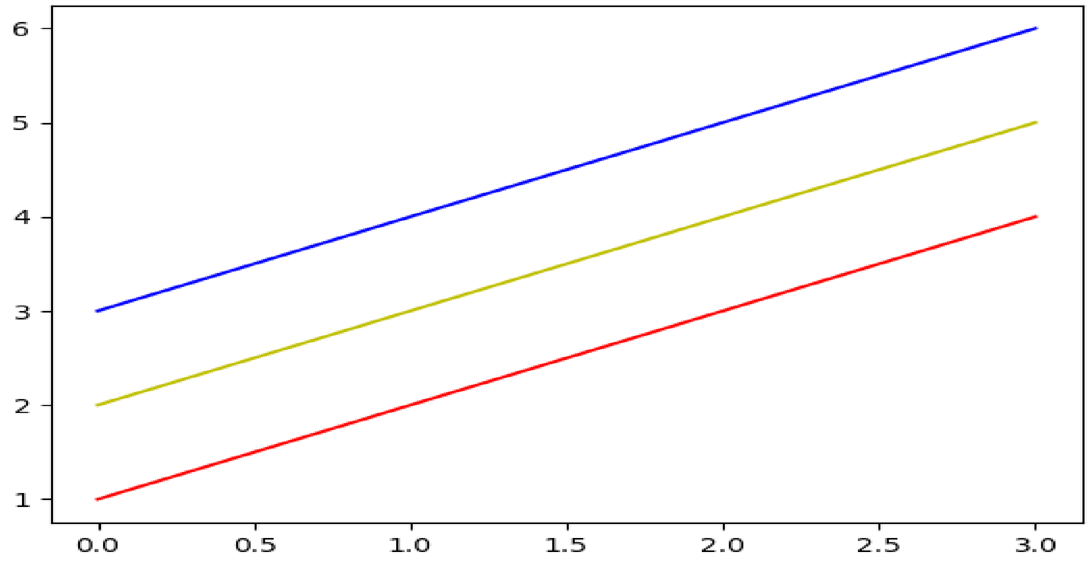
```
import matplotlib.pyplot as plt
plt.plot([2,4,6,8,10,12],[1, 2, 3, 4, 5, 4], 'g') # Creates a line in ref to X-Axis & Y-Axis respectability of Green colour('g')
plt.plot([1,2,3,4,5,6], [1, 2, 3, 4, 5, 4], 'b') # Creates a line in ref to X-Axis & Y-Axis respectability of Blue colour('b')
plt.grid()                                   # Reformat the layout with grid
plt.title('My Pyplot')                       # Displays title of the layout at the top
plt.ylabel('some numbers')                   # Displays label of Y-Axis
plt.xlabel('values')                         # Displays label of X-Axis
plt.show()
```



17.2.4 Application of arrange() function:

Example:

```
import numpy as np
import matplotlib.pyplot as plt
x=np.arange(1,5,1)
plt.plot(x, 'r')        # 'r' makes red colour of the line generated according to value of x
plt.plot(x+1, 'y')      # 'y' makes yellow colour of the line according to value of x+1
plt.plot(x+2, 'b')      # 'b' makes blue colour of the line according to value of x+2
plt.show()
```



X=[1, 2, 3, 4]
X+1 = [2 , 3, 4, 5]
X+2= [3 , 4, 5, 6]

<u>Note</u>:
arange(a,b,c): arange() function generates values from starting value(a) up to before stop value (b) incremented by third value(c) which is optional. In the above example, [**1, 2, 3, 4**] values will be generated for x variable, where initial value is 1, final is 5 and increment value is 1.

17.3 BAR CHART:

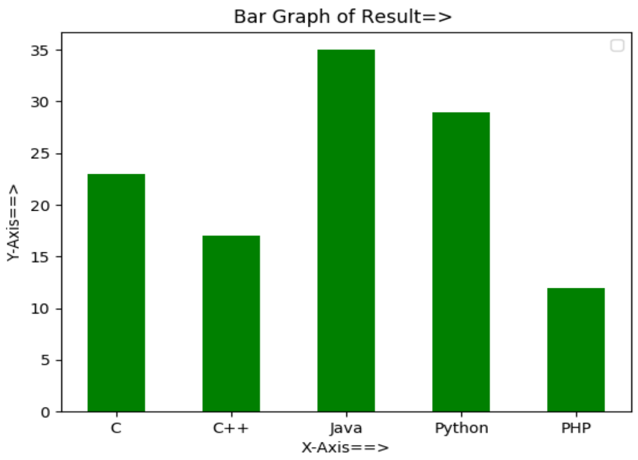
A bar chart or bar graph is a chart or graph that presents categorical data with rectangular bars with heights or lengths proportional to the values that they represent. The bars can be plotted vertically or horizontally.

A bar graph shows comparisons among discrete categories. One axis of the chart shows the specific categories being compared, and the other axis represents a measured value.

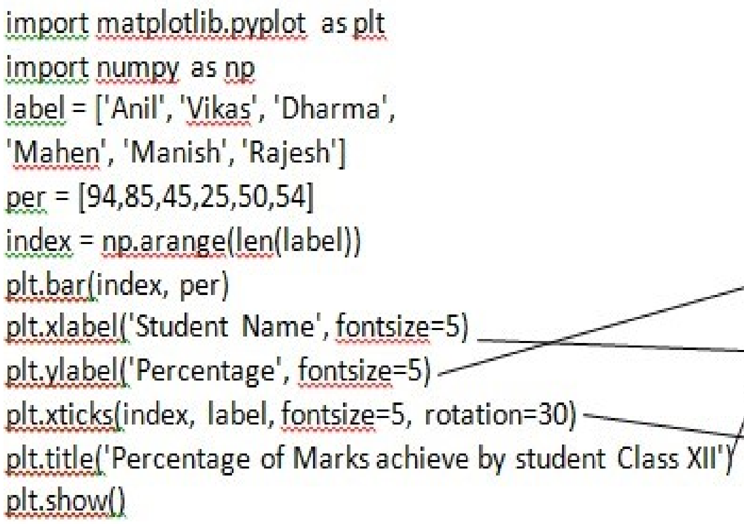
Matplotlib API provides the bar() function that can be used in the MATLAB style use as well as object oriented API.

17.3.1 Example:

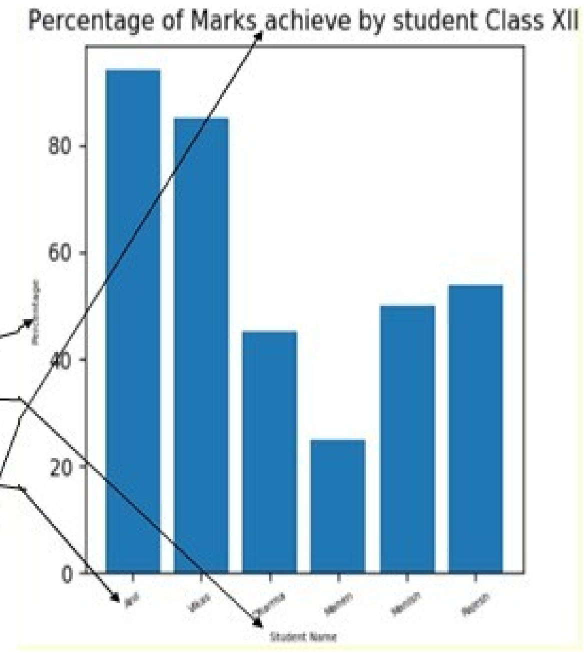
```
import matplotlib.pyplot as plt
langs = ['C', 'C++', 'Java', 'Python', 'PHP']
students = [23,17,35,29,12]
plt.bar(langs, students, width=.5, color='g')
plt.xlabel('X-Axis==>')
plt.ylabel('Y-Axis==>')
plt.title('Bar Graph of Result=> ')
plt.show()
```



17.3.2 Example

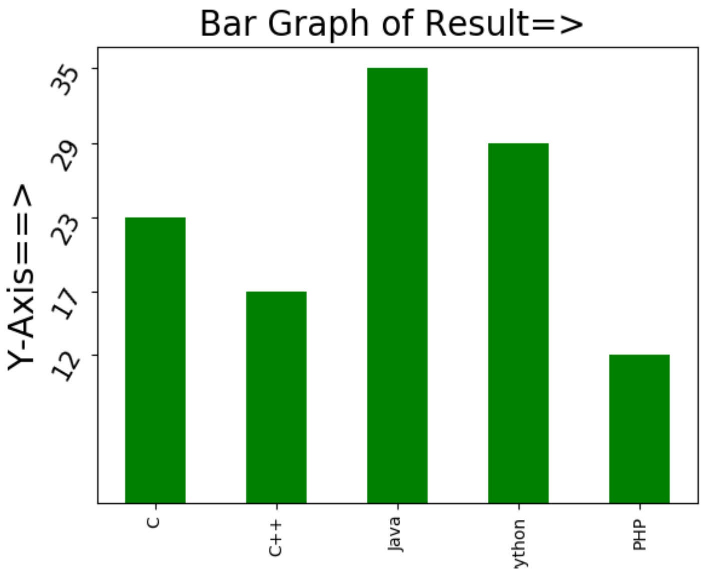


```
import matplotlib.pyplot  as plt
import numpy as np
label = ['Anil', 'Vikas', 'Dharma',
'Mahen', 'Manish', 'Rajesh']
per = [94,85,45,25,50,54]
index = np.arange(len(label))
plt.bar(index, per)
plt.xlabel('Student  Name', fontsize=5)
plt.ylabel('Percentage', fontsize=5)
plt.xticks(index, label, fontsize=5, rotation=30)
plt.title('Percentage of Marks achieve by student Class XII')
plt.show()
```



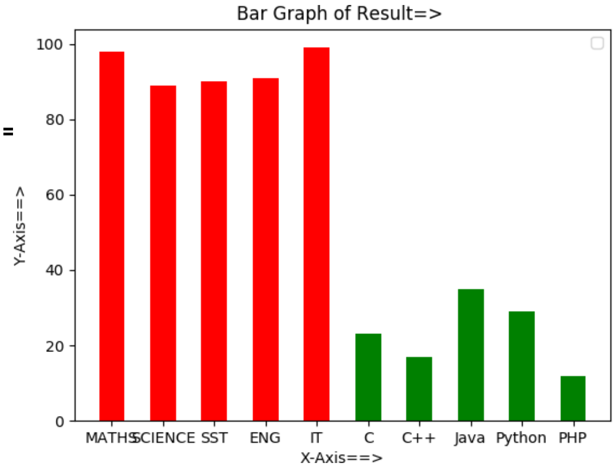
17.3.3 Example:

```
import matplotlib.pyplot as plt
langs = ['C', 'C++', 'Java', 'Python', 'PHP']
students = [23,17,35,29,12]
plt.bar(langs , students,  width=.5, color='g')
plt.xlabel('X-Axis==>', fontsize=20)
plt.ylabel('Y-Axis==>', fontsize=20)
plt.xticks(langs, fontsize=10, rotation=90)
plt.yticks(students, fontsize=15, rotation=60)
plt.title('Bar Graph of Result=> ', fontsize=20)
plt.show()
```



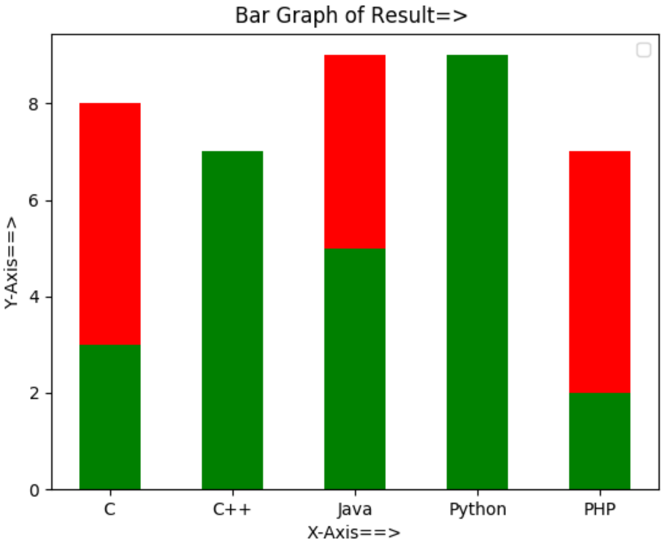
17.3.4 Example:

```
import matplotlib.pyplot as plt
x=['MATHS', 'SCIENCE', 'SST', 'ENG', 'IT'] y=[98,
89,90,91,99]
langs = ['C', 'C++', 'Java', 'Python', 'PHP'] students =
[23,17,35,29,12]
plt.bar(x , y, width=0.5 , color='r')
plt.bar(langs,students, width=.5, color='g')
plt.xlabel('X-Axis==>')
plt.ylabel('Y-Axis==>')
plt.title('Bar Graph of Result=> ')
plt.show()
```



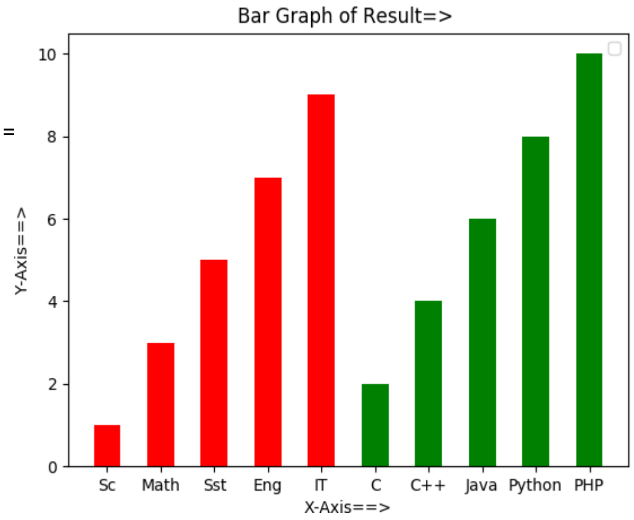
17.3.5 Example:

```
import matplotlib.pyplot as plt
x=['MATHS', 'SCIENCE', 'SST', 'ENG', 'IT']
y=[98, 89,90,91,99]
x=['C', 'C++', 'Java', 'Python', 'PHP']
y=[8, 5,9,6,7]
langs = ['C', 'C++', 'Java', 'Python', 'PHP']
students = [3,7,5,9,2]
plt.bar(x , y, width=0.5 , color='r')
plt.bar(langs,students, width=.5, color='g')
plt.xlabel('X-Axis==>')
plt.ylabel('Y-Axis==>')
plt.title('Bar Graph of Result=> ')
plt.show()
```



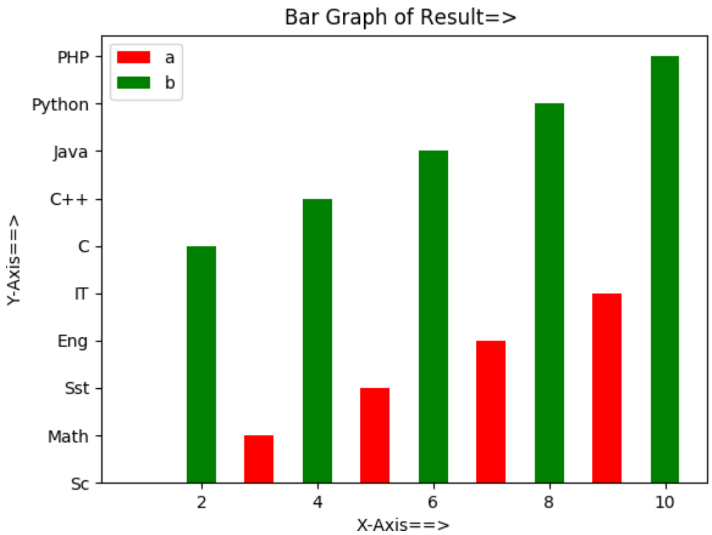
17.3.6 Example:

```
import matplotlib.pyplot as plt
x=['Sc', 'Math', 'Sst', 'Eng', 'IT']
y=[1,3,5,7,9]
langs = ['C', 'C++', 'Java', 'Python', 'PHP'] students =
[2,4,6,8,10]
plt.bar(x , y, width=0.5 , color='r')
plt.bar(langs,students, width=.5, color='g')
plt.xlabel('X-Axis==>')
plt.ylabel('Y-Axis==>')
plt.title('Bar Graph of Result=> ')
plt.show()
```



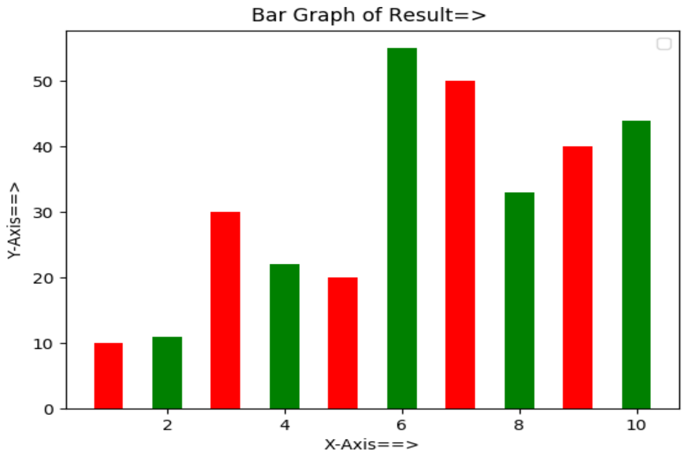
17.3.7 Example: Using legend()

```
import matplotlib.pyplot as plt
x=[1,3,5,7,9]
y=['Sc', 'Math', 'Sst', 'Eng', 'IT']
x1 = [2,4,6,8,10]
y1 = ['C', 'C++', 'Java', 'Python', 'PHP']
a=plt.bar(x , y, width=0.5 , color='r')
b=plt.bar(x1,y1, width=.5, color='g')
plt.xlabel('X-Axis==>')
plt.ylabel('Y-Axis==>')
plt.title('Bar Graph of Result=> ')
plt.legend([‘a’,’b’])   # Legend
plt.show()
```



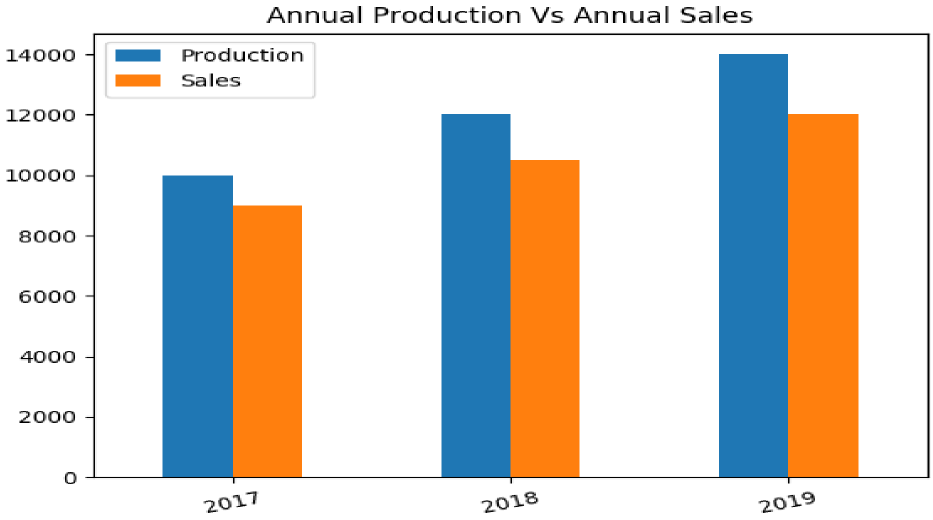
17.3.8 Example:

```
import matplotlib.pyplot as plt
y=[10,30,20,50,40]
x=[1,3,5,7,9]
y1 = [11,22,55, 33, 44]
x1 = [2,4,6,8,10]
plt.bar(x , y, width=0.5 , color='r')
plt.bar(x1,y1, width=.5, color='g')
plt.xlabel('X-Axis==>')
plt.ylabel('Y-Axis==>')
plt.title('Bar Graph of Result=> ')
#plt.legend()
plt.show()
```



17.3.9 Example: Using Panda's DataFrame:

```
import pandas as pd
import matplotlib.pyplot as plot
data = {"Production":[10000, 12000, 14000],
        "Sales":[9000, 10500, 12000]
       }
index= ["2017", "2018", "2019"]
df = pd.DataFrame(data=data, index=index) # data variable has two sets of values as X-axis
df.plot.bar(rot=15, title="Annual Production Vs Annual Sales")
plot.show()
```


Annual Production Vs Annual Sales
Production
Sales