Unit-III Introduction to NumPy, Pandas, Matplotlib. Exploratory Data Analysis (EDA), Data Science life cycle, Descriptive Statistics, Basic tools (plots, graphs and summary statistics) of EDA, Philosophy of EDA. Data Visualization: Scatter plot, bar chart, histogram, boxplot, heat maps, etc

Introduction to Data Science:

Data science is a generation of actionable knowledge or discover hidden patterns from the raw data directly from huge amount of complex data by applying concepts from statistics, mathematics and computer science. The goal of data science is to gain insights from both structured and unstructured data.

derive meaningful information, and make business decisions. Data science uses complex machine learning algorithms to build predictive models.

Why Data Science?

- Traditionally, the data that we had was mostly structured and small in size, which could be analyzed by using simple tools. Today most of the data is unstructured. More than 80 % of the data will be unstructured.

Structured data – all data which can be stored in a table with rows and columns. This data exists in a format of relational databases (RDBMSs). or analytical purposes, you can use data warehouses. DWs are central data storages used by companies for data analysis and reporting.

There is a special programming language used for handling relational databases and warehouses called SQL, which stands for Structured Query Language and was developed back in the 1970s by IBM.

This data can comprise both text and numbers, such as employee names, contacts, ZIP codes, addresses, credit card numbers, etc.

	A	B	C	D	E	F	G
1	Purchase ID	Last name	First name	Birthday	Country	Date of purchase	Amount of purchase
2	1	Davidson	Michael	04/03/1986	United States	10/12/2016	37
3	2	Vito	Jim	09/01/1994	United Kingdom	02/02/2016	85
4	3	Johnson	Tom	23/08/1972	France	02/11/2016	83
5	4	Lewis	Peter	18/10/1979	Germany	22/11/2016	27
6	5	Koenig	Edward	13/05/1983	Argentina	26/03/2015	43
7	6	Preston	Jack	16/06/1991	United States	06/11/2016	77
8	7	Smith	David	11/03/1965	Canada	15/11/2016	23
9	8	Brown	Luis	03/09/1997	Australia	03/07/2015	74
10	9	Miller	Thomas	07/01/1980	Germany	07/11/2016	13
11	10	Williams	Bill	26/07/1960	United States	20/11/2015	80
12	11	Gemini	Alexia	12/09/1995	Canada	11/03/2017	35
13	12	Bond	James	25/02/1975	United Kingdom	12/08/2017	40
14	13	Burgle	Patricia	01/12/1990	United States	18/01/2015	55
15	14	Reding	Michelle	07/04/1985	Canada	23/02/2017	28
16	15	Harvey	Billy	14/07/1971	United Kingdom	12/01/2016	41
17							

Structured data commonly exists in tables similar to Excel files and Google Docs spreadsheets.

Unstructured data doesn't have any pre-defined structure to it. with unstructured data is that traditional methods and tools can't be used to analyze and process it.

Ex: unstructured data such as email, text files, social media posts, video, images, audio, sensor data, and so on.

```
a=[1,2,3]

b=[4,5,6]

c=a+b

print(c)

result=[]

for i,j in zip(a,b):

    result.append(i+j)

print(result)
```

o/p:

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

[5, 7, 9]

Numpy:

NumPy (*Numerical Python)* is a Python library/package **used for doing scientific calculations. It** works with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. **NumPy** was created in 2005 by Travis Oliphant. It is an open source project and you can **use** it freely. **NumPy** is a commonly **used Python data analysis** package. It also has functions for working in domain of linear algebra, fourier transform, and matrices. Scientific languages available are MATLAB, FORTRAN,etc. Numpy s not part of basic python installation.

Installation of Python:

Pip install numpy

Once NumPy is installed, import it in your applications by adding the `import` keyword:

Import numpy

Version of numpy:

print(numpy.__version__)

Creation of ndarray:

> **Arrays in NumPy:** NumPy's main object is the homogeneous multidimensional array. Numpy creates arrays called ndarray. It works faster than lists. We can create a NumPy `ndarray` object by using the `array()` function. In NumPy, dimensions are called axes

```
import numpy
a = numpy.array([1, 2, 3, 4, 5])
print(a)
```

o/p:

[1 2 3 4 5]

NumPy is usually imported under the `np` alias

```
import numpy as np
a=np.array([1, 2, 3, 4, 5])
print(arr)
```

o/p: [1 2 3 4 5]

```
import numpy as np
a = np.array([1, 2, 3, 4, 5])
print(a)
print(type(a))
```

o/p:

[1 2 3 4 5]

<class 'numpy.ndarray'>

```
a.dtype
dtype('int32')
>>> a=np.array([1.5,2.3])
>>> a.dtype
dtype('float64')

>>> a=np.array([10,20,30])
>>> a[0]
10
>>> a[0]=12.5
>>> a
array([12, 20, 30])
```

0-D:

```
np.array(50)
```

1-d:

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

Or Array of 0D arrays

```
import numpy as np
a=np.array([1, 2, 3, 4, 5])
print(a)
```

O/P: [1 2 3 4 5]

```
>>> a.ndim
1
```

2-D:

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

```
import numpy as np
a = np.array([[1, 2, 3], [4, 5, 6]])
print(arr)
```

o/p:

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

3-D:

An array that has 2-D arrays (matrices) as its elements is called 3-D array. Array of 2D arrays

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

```
import numpy as np

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

print(arr)
O/P:
[[[1 2 3]
  [4 5 6]]

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

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

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

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

o/p:

o

1

2

3

>>> a.ndim

1

>>> a.shape

(3,)

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

>>> a.ndim

2

>>> a.shape

(2, 3)

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

>>> a.ndim

3

The shape of an array can be defined as the number of elements in each dimension.

```
>>> a.shape
(2, 2, 3)
```

The first number in the parenthesis represents the number of elements within the first axis/dimension; the second number the number of elements within the second axis/dimension, the third number the number of elements within the third axis/dimensions, and so on.

For instance, the (2, 2, 3) indicates 2 elements along the first axis, 2 elements along the second axis, and 3 elements along the third axis.

```
>>> import numpy as np
>>> a=np.array(60)
>>> a
array(60)
>>> a.ndim
0
>>> b=np.array([30,40,50])
>>> b
array([30, 40, 50])
>>> b.ndim
1
>>> c
array([[1, 2, 3],
       [5, 6, 7]])
>>> c.ndim
2
>>> d=np.array([[[1,2,3],[4,5,6],[7,8,9]],[[6,3,2],[1,2,3],[7,8,9]]])
>>> d
array([[[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]],

       [[6, 3, 2],
        [1, 2, 3],
        [7, 8, 9]]])
>>> d.ndim
3
>>> a.shape
()
>>> b.shape
(3,)
>>> c.shape
```

(2, 3)
>>> d.shape
(2, 3, 3)

To count the number of elements within an array type
>>> a.size
1
>>> b.size
3
>>> c.size
6
>>> d.size
18

List, tuples, sets can be used for creating arrays
>>> a=np.array((1,2,3))
>>> a
array([1, 2, 3])
>>> a
array([1, 2, 3])
>>> b=np.array([[1,2,3],[4,5,6]])
>>> b
array([[1, 2, 3],
[4, 5, 6]])
>>> c=np.array({'python','data','science'})
>>> c
array({'data', 'python', 'science'}, dtype=object)

```
zeros = np.zeros(5)

# ones
ones = np.ones((3, 3))

# arange
arange = np.arange(1, 10, 2)

# empty
empty =  np.empty([2, 2])

# linspace
linespace = np.linspace(-1.0, 1.0, num=10)

# full
full = np.full((3,3), -2)

# indices
```

```
indices =  np.indices((3,3))

Array of zeros:
[0. 0. 0. 0. 0.]

Array of ones:
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]

Array of empty entries:
[[4.67794427e-310 6.90921830e-310]
 [0.00000000e+000 0.00000000e+000]]

Evenly spaced array in a range:
[-1.         -0.77777778 -0.55555556 -0.33333333 -0.11111111  0.11111111
  0.33333333  0.55555556  0.77777778  1.        ]

Array with same number on each entry:
[[-2 -2 -2]
 [-2 -2 -2]
 [-2 -2 -2]]

Array from indices:
[[[0 0 0]
  [1 1 1]
  [2 2 2]]

 [[0 1 2]
  [0 1 2]
  [0 1 2]]]
```

- The `zero` method generates an array of zeros of shape defined by a tuple passed to the function
- The `ones` method generates an array of ones of shape defined by a tuple passed to the function
- The `empty` method generates an empty array (although very small numbers will be printed) of shape defined by a tuple passed to the function.

```
diagonal = np.diag([1, 2, 3], k=0)

# identity
identity = np.identity(3)

# eye
eye = np.eye(4, k=1)

# rand
rand = np.random.rand(3,2)
rand = np.random.rand(3,2)
```

- The `diagonal` function returns an array with the numbers in the diagonal and zeros elsewhere
- The `identity` function returns an identity matrix
- The `eye` function returns an array with ones on the diagonal and zeros elsewhere
- The `random.rand` function returns an array of random numbers sampled from a uniform distribution

```
Diagonal matrix from array-like structure:
[[1 0 0]
 [0 2 0]
 [0 0 3]]

Identity matrix:
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]

Diagonal matrix with ones and zeros elsewhere:
[[0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]
 [0. 0. 0. 0.]]

Array of random numbers sampled from a uniform distribution:
[[0.75060485 0.07962041]
 [0.36030122 0.11582055]
 [0.57917376 0.93888782]]
```

Reshaping arrays

Reshaping means changing the shape of an array.

The shape of an array is the number of elements in each dimension.

By reshaping we can add or remove dimensions or change number of elements in each dimension.

Convert the following 1-D array with 12 elements into a 2-D array.

The outermost dimension will have 4 arrays, each with 3 elements:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

newarr = arr.reshape(4, 3)
newarr
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

Reshape From 1-D to 3-D

Example

Convert the following 1-D array with 12 elements into a 3-D array.

The outermost dimension will have 2 arrays that contains 3 arrays, each with 2 elements:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

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

newarr

array([[[ 1,  2],

        [ 3,  4],

        [ 5,  6]],


       [[ 7,  8],

        [ 9, 10],

        [11, 12]]])
```

Can We Reshape Into any Shape?

Yes, as long as the elements required for reshaping are equal in both shapes.

We can reshape an 8 elements 1D array into 4 elements in 2 rows 2D array but we cannot reshape it into a 3 elements 3 rows 2D array as that would require 3x3 = 9 elements.

```
import numpy as np

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

newarr = arr.reshape(3, 3)

print(newarr)

Traceback (most recent call last):
  File "<pyshell#45>", line 1, in <module>
    newarr = arr.reshape(3, 3)
ValueError: cannot reshape array of size 8 into shape (3,3)
```

Array arithmetic

the top-left elements in each array are added together, the top-right elements of each array are added together, and so on. Subtraction, division, multiplication, exponentiation, logarithms, roots, and many other algebraic operations (or arithmetic depending on whom you ask), will be performed in the same manner.

```
import numpy as np
a = np.arange(1, 10).reshape((3,3))
print(a)
b = np.arange(10,19).reshape((3,3))
print(b)

add = a + b
print("addition",add)
sub = a - b
print("subtraction",sub)
mul = a * b
print("multiplication",mul)
division = a / b
print("true_division",division)
floor = a // b
print("floor_division",floor)
rem = np.remainder(a, b)
print("remainder",rem)
o/p:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[[10 11 12]
 [13 14 15]
 [16 17 18]]
addition [[11 13 15]
 [17 19 21]
 [23 25 27]]
subtraction [[-9 -9 -9]
 [-9 -9 -9]
 [-9 -9 -9]]
multiplication [[ 10  22  36]
 [ 52  70  90]
 [112 136 162]]
true_division [[0.1        0.18181818 0.25      ]
 [0.30769231 0.35714286 0.4       ]
 [0.4375     0.47058824 0.5       ]]
floor_division [[0 0 0]
 [0 0 0]
 [0 0 0]]
remainder [[1 2 3]
 [4 5 6]
 [7 8 9]]

>>> a=np.identity(3)
>>> a
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
>>> a.dtype
dtype('float64')
>>> a=np.identity(3,dtype='i')
>>> a
array([[1, 0, 0],
       [0, 1, 0],
       [0, 0, 1]], dtype=int32)
```

```
>>> a.dtype
dtype('int32')
>>> a=np.identity(3,dtype=int)
>>> a
array([[1, 0, 0],
       [0, 1, 0],
       [0, 0, 1]])
>>> a=np.eye(3)
>>> a
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
>>> a=np.eye(8)
>>> a
array([[1., 0., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1.]])
>>> a=np.eye(8,k=1)
>>> a
array([[0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 0., 0., 0., 0.]])
>>> a=np.eye(8,k=2)
>>> a
array([[0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0.]])
>>> a=np.eye(8,k=-1,dtype='i')
>>> a
array([[0, 0, 0, 0, 0, 0, 0, 0],
       [1, 0, 0, 0, 0, 0, 0, 0],
       [0, 1, 0, 0, 0, 0, 0, 0],
       [0, 0, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 1, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 0]], dtype=int32)
>>> a=np.eye(8,k=0)
>>> a
array([[1., 0., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1.]])
>>> a=np.eye(8)
>>> a
```

```
array([[1., 0., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1.]])
>>> (start,stop,step,dtype)
Traceback (most recent call last):
  File "<pyshell#23>", line 1, in <module>
    (start,stop,step,dtype)
NameError: name 'start' is not defined
>>> a=np.arange(3)
>>> a
array([0, 1, 2])
>>> a=np.arange(40)
>>> a
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
       34, 35, 36, 37, 38, 39])
>>> a.ndim
1
>>> a.shape=(5,8)
>>> a
array([[ 0,  1,  2,  3,  4,  5,  6,  7],
       [ 8,  9, 10, 11, 12, 13, 14, 15],
       [16, 17, 18, 19, 20, 21, 22, 23],
       [24, 25, 26, 27, 28, 29, 30, 31],
       [32, 33, 34, 35, 36, 37, 38, 39]])
>>> a.ndim
2
>>> a.shape
(5, 8)
>>> a=np.array(1,21)
Traceback (most recent call last):
  File "<pyshell#33>", line 1, in <module>
    a=np.array(1,21)
TypeError: Cannot interpret '21' as a data type
>>> a=np.arange(1,21)
>>> a
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20])
>>> a.shape=(4,5)
>>> a
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10],
       [11, 12, 13, 14, 15],
       [16, 17, 18, 19, 20]])
>>> a=np.arange(1,21,3)
>>> a
array([ 1,  4,  7, 10, 13, 16, 19])
>>> a=np.arange(1,21.1,3)
>>> a
array([ 1.,  4.,  7., 10., 13., 16., 19.])
>>> a=np.arange(1,21.1,3,dtype=int)
>>> a
array([ 1,  4,  7, 10, 13, 16, 19])

>>> a=np.random.rand(100)
>>> a
array([0.09625205, 0.66375609, 0.40027069, 0.09668508, 0.69563543,
       0.79675033, 0.98403482, 0.87273731, 0.17470161, 0.12475961,
       0.42195266, 0.48309578, 0.90233294, 0.12471774, 0.54954757,
```

```
       0.52243077, 0.44022385, 0.30461071, 0.80906925, 0.3398229 ,
       0.21902711, 0.68416111, 0.52792173, 0.73636642, 0.72739759,
       0.95089791, 0.49048233, 0.1598883 , 0.11011531, 0.29716097,
       0.21350247, 0.51686015, 0.09456273, 0.95730603, 0.67640728,
       0.05464754, 0.90547551, 0.46072303, 0.51021914, 0.51825393,
       0.53543517, 0.54798255, 0.47569338, 0.63552223, 0.07444923,
       0.06878207, 0.83899717, 0.00633528, 0.44504758, 0.96509376,
       0.40741983, 0.8561961 , 0.28148682, 0.24382749, 0.89142038,
       0.21062094, 0.90572327, 0.28456164, 0.00133873, 0.34841955,
       0.98618439, 0.70897179, 0.56600745, 0.60432564, 0.2639877 ,
       0.75488613, 0.67580114, 0.90945616, 0.6516998 , 0.03976116,
       0.12864848, 0.90474892, 0.61154226, 0.63130699, 0.54977776,
       0.60693429, 0.81442494, 0.33861608, 0.75987533, 0.07650644,
       0.27421804, 0.50519508, 0.00948162, 0.63853749, 0.67764656,
       0.21309345, 0.9205802 , 0.64797108, 0.23404973, 0.78935725,
       0.14723574, 0.4981408 , 0.20099304, 0.54870065, 0.48247184,
       0.75730997, 0.20561619, 0.41396026, 0.68749212, 0.18145768])
>>> a=np.random.rand(3,4)
>>> a
array([[0.58229451, 0.14834279, 0.59443151, 0.00360283],
       [0.17424312, 0.4117869 , 0.15241778, 0.16135185],
       [0.93578789, 0.49120737, 0.15527398, 0.43687062]])
>>> a=np.empty(3)
>>> a
array([1., 1., 1.])
>>> a=np.arange(1,21)
>>> a
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20])
>>> a.ndim
1
>>> a.shape
(20,)
>>> a[1:5]
array([2, 3, 4, 5])
                                                                    >>> a[:]
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20])
>>> a[4:]
array([ 5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20])
>>> a[:10]
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
>>> a[1:10:2]
array([ 2,  4,  6,  8, 10])
>>> a[-3:-1]
array([18, 19])
>>> a[-5:-2:-1]
array([], dtype=int32)



>>> import numpy as np
>>> import csv
>>> with open('winequality-red.csv', 'r') as f:
        wines = list(csv.reader(f, delimiter=';'))


>>> print(wines[:3])
[['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density', 'pH',
'sulphates', 'alcohol', 'quality'], ['7.4', '0.7', '0', '1.9', '0.076', '11',
'34', '0.9978', '3.51', '0.56', '9.4', '5'], ['7.8', '0.88', '0', '2.6',
'0.098', '25', '67', '0.9968', '3.2', '0.68', '9.8', '5']]
```

```
>>> wines=np.array(wines[1:],dtype=float)
>>> wines
array([[ 7.4  ,  0.7  ,  0.   , ...,  0.56 ,  9.4  ,  5.   ],
       [ 7.8  ,  0.88 ,  0.   , ...,  0.68 ,  9.8  ,  5.   ],
       [ 7.8  ,  0.76 ,  0.04 , ...,  0.65 ,  9.8  ,  5.   ],
       ...,
       [ 6.3  ,  0.51 ,  0.13 , ...,  0.75 , 11.   ,  6.   ],
       [ 5.9  ,  0.645,  0.12 , ...,  0.71 , 10.2  ,  5.   ],
       [ 6.   ,  0.31 ,  0.47 , ...,  0.66 , 11.   ,  6.   ]])
```

Matplotlib:

Matplotlib is one of the most popular Python packages used for data visualization. It is a library for making 2D plots from data in arrays.

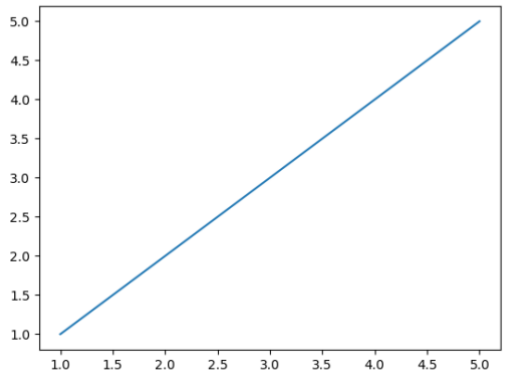
Matplotlib can be installed using pip. The following command is run in the command prompt to install Matplotlib.

```
pip install matplotlib
```

```
>>> import matplotlib

>>> matplotlib.__version__

'3.4.2'

import matplotlib.pyplot as plt

import numpy as np

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

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

plt.plot(a,b)

plt.show()
```

We pass two arrays as our input arguments to Pyplot's `plot()` method and use `show()` method to invoke the required plot. Here note that the first array

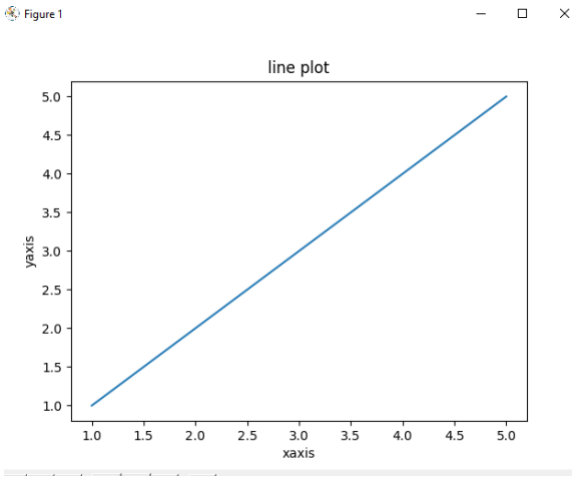
appears on the x-axis and second array appears on the y-axis of the



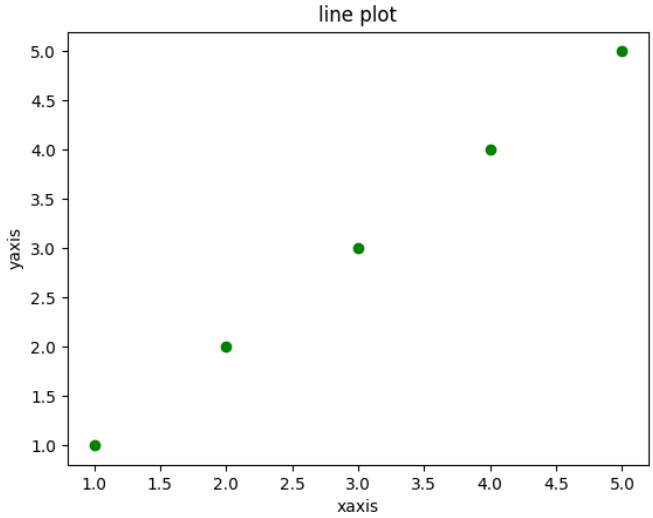
plot.

name x-axis and y-axis using methods `title()`, `xlabel()` and `ylabel()` respectively.

```
import matplotlib.pyplot as plt

import numpy as np

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

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

plt.plot(a,b)

plt.xlabel("xaxis")

plt.ylabel("yaxis")

plt.title("line plot")

plt.show()
```



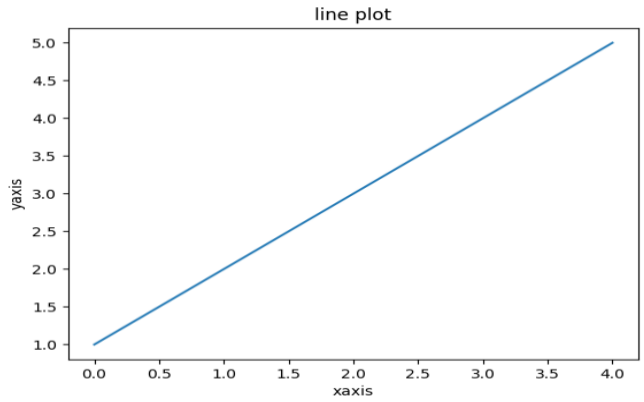
With every X and Y argument, you can also pass an optional third argument in the form of a string which indicates the colour and line type of the plot. The default format is **b-** which means a solid blue line. In the figure below we use **go** which means green circles.



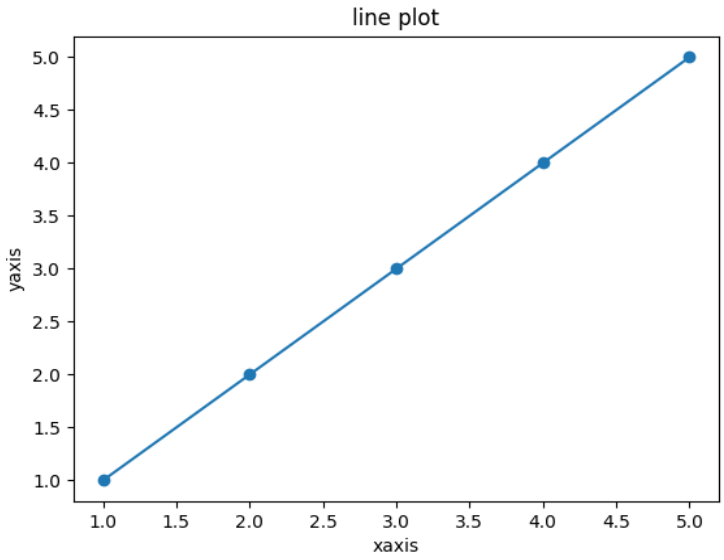
Default X-Points

If we do not specify the points in the x-axis, they will get the default values 0, 1, 2, 3, (etc. depending on the length of the y-points.

```
import matplotlib.pyplot as plt
import numpy as np
b=np.array([1,2,3,4,5])
plt.plot(b)
plt.xlabel("xaxis")
plt.ylabel("yaxis")
plt.title("line plot")
plt.show()
```



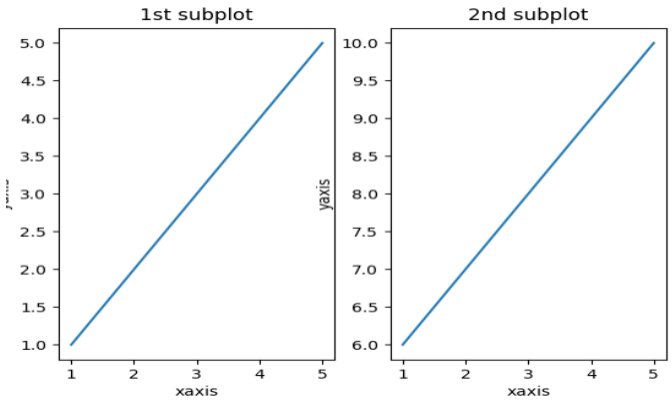

```
import matplotlib.pyplot as plt
import numpy as np
a=np.array([1,2,3,4,5])
b=np.array([1,2,3,4,5])
plt.plot(a,b,marker="o")
plt.xlabel("xaxis")
plt.ylabel("yaxis")
plt.title("line plot")
plt.show()
```



We can use `subplot()` method to add more than one plots in one figure. The `subplot()` method takes three arguments: they are `nrows`, `ncols` and `index`.

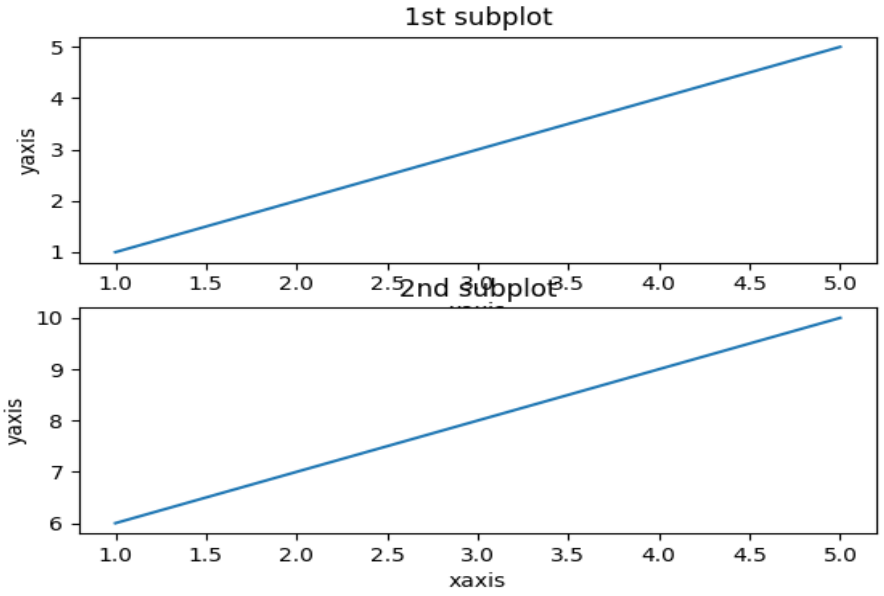
```
import matplotlib.pyplot as plt
import numpy as np
a=np.array([1,2,3,4,5])
```

```
b=np.array([1,2,3,4,5])
plt.subplot(1,2,1)
plt.plot(a,b)
plt.xlabel("xaxis")
plt.ylabel("yaxis")
plt.title("1st subplot")

a=np.array([1,2,3,4,5])
b=np.array([6,7,8,9,10])
plt.subplot(1,2,2)
plt.plot(a,b)
plt.xlabel("xaxis")
plt.ylabel("yaxis")
plt.title("2nd subplot")

plt.show()
```



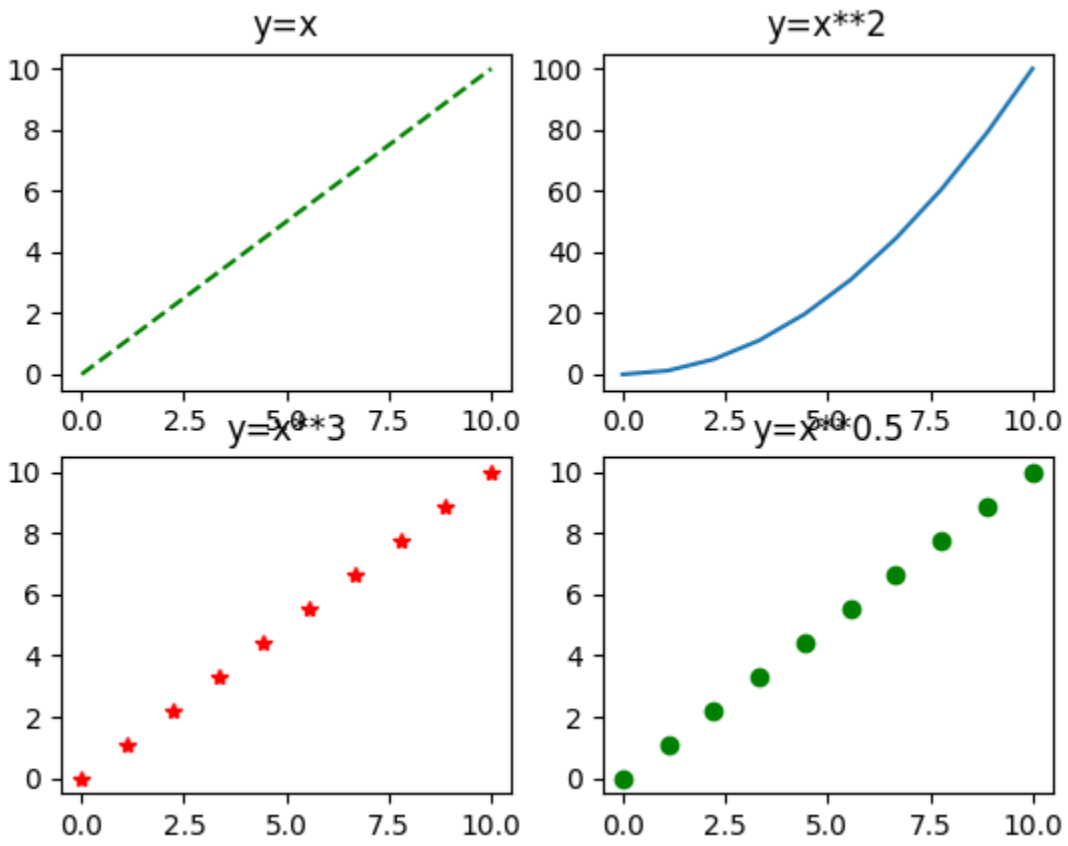

```
import matplotlib.pyplot as plt
import numpy as np
a=np.array([1,2,3,4,5])
b=np.array([1,2,3,4,5])
plt.subplot(2,1,1)
plt.plot(a,b)
plt.xlabel("xaxis")
plt.ylabel("yaxis")
plt.title("1st subplot")

a=np.array([1,2,3,4,5])
b=np.array([6,7,8,9,10])
plt.subplot(2,1,2)
```

```
plt.plot(a,b)
plt.xlabel("xaxis")
plt.ylabel("yaxis")
plt.title("2nd subplot")

plt.show()
```




```
import numpy as np
import matplotlib.pyplot as plt
x=np.linspace(0,10,10)
print(x)
y1=x
print(y1)
y2=x**2
print(y2)
y3=x**3
print(y3)
y4=np.sqrt(x)
print(y4)


plt.subplot(2,2,1)
plt.plot(x,y1,"g--")
plt.title("y=x")
```

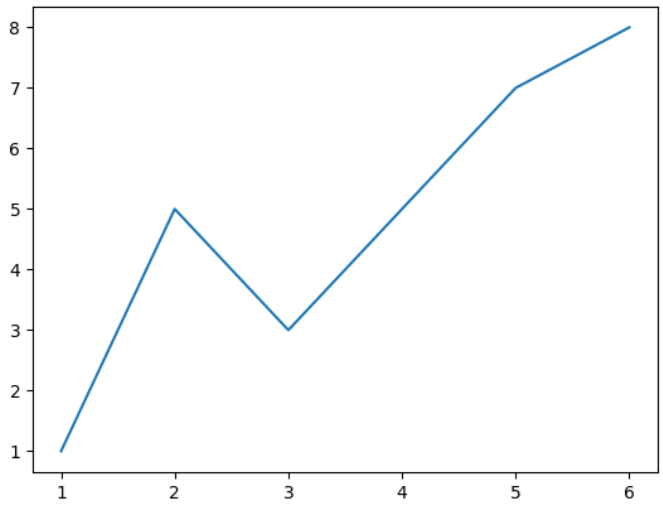
```
plt.subplot(2,2,2)
plt.plot(x,y2)
plt.title("y=x**2")

plt.subplot(2,2,3)
plt.plot(x,y1,"r*")
plt.title("y=x**3")

plt.subplot(2,2,4)
plt.plot(x,y1,"go")
plt.title("y=x**0.5")

plt.show()
```



Plot a Line Plot in Matplotlib

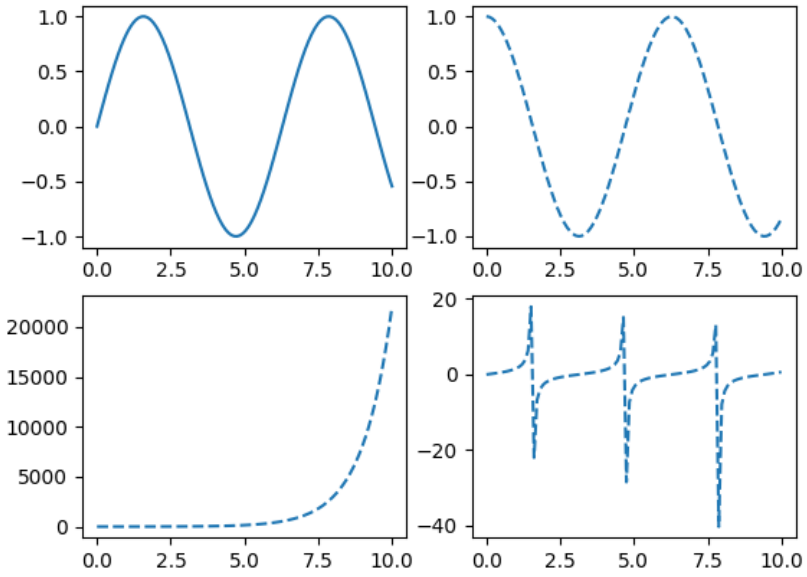
To plot a line plot in Matplotlib, you use the generic `plot()` function from the PyPlot instance. There's no specific `lineplot()` function - the generic one automatically plots using lines or markers.

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5, 6]

y = [1, 5, 3, 5, 7, 8]

plt.plot(x, y)

plt.show()
```



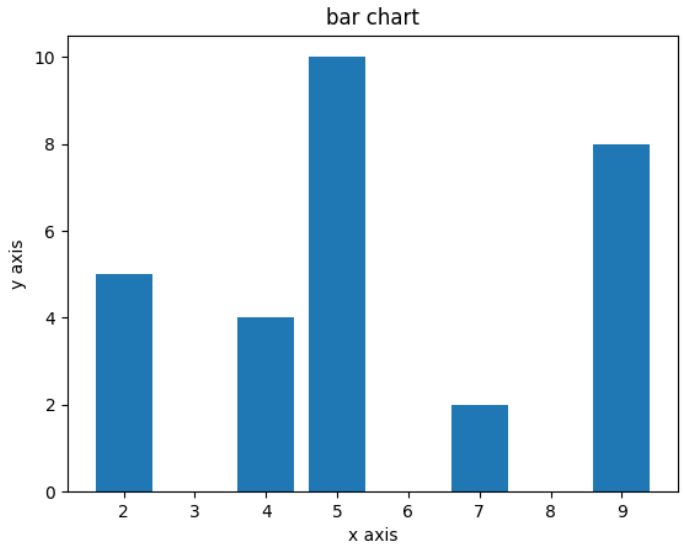

```
import numpy as np

import matplotlib.pyplot as plt

x = np.linspace(0, 10, 100)

plt.subplot(2,2,1)

plt.plot(x, np.sin(x), '-')


plt.subplot(2,2,2)
```

```
plt.plot(x, np.cos(x), '--')


plt.subplot(2,2,3)

plt.plot(x, np.exp(x), '--')


plt.subplot(2,2,4)

plt.plot(x, np.tan(x), '--')

plt.show()
```

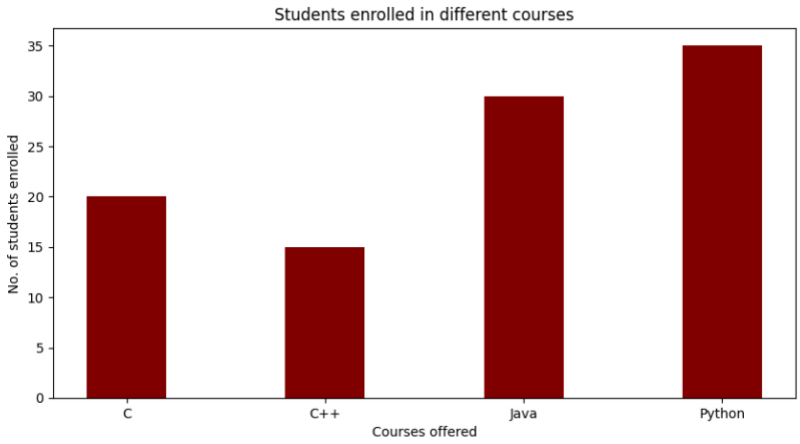


Bar plot:

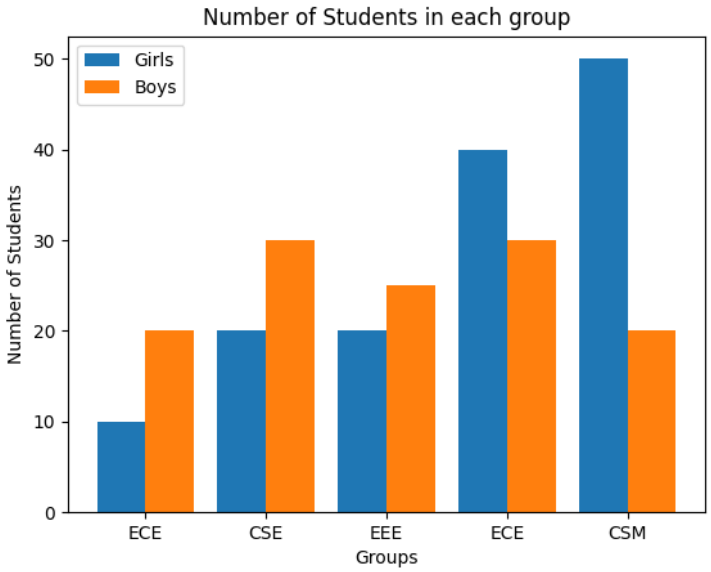
```
import numpy as np
import matplotlib.pyplot as plt
x = [5, 2, 9, 4, 7]
y = [10, 5, 8, 4, 2]
plt.xlabel("x axis")
plt.ylabel("y axis")
plt.title("bar chart")
plt.bar(x,y)
plt.show()
```




```
import numpy as np
import matplotlib.pyplot as plt


# creating the dataset
data = {'C':20, 'C++':15, 'Java':30,
        'Python':35}
courses = list(data.keys())
values = list(data.values())

fig = plt.figure(figsize = (10, 5))

# creating the bar plot
plt.bar(courses, values, color ='maroon',
        width = 0.4)

plt.xlabel("Courses offered")
plt.ylabel("No. of students enrolled")
plt.title("Students enrolled in different courses")
plt.show()
```

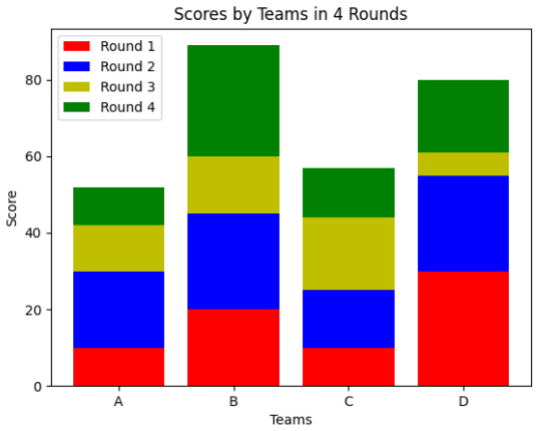



```
import numpy as np
import matplotlib.pyplot as plt

X = ['ECE','CSE','EEE','ECE','CSM']
girls = [10,20,20,40,50]
boys = [20,30,25,30,20]
w=0.4
'''x location'''

bar1 = np.arange(len(X))
bar2= [i+w for i in bar1]

plt.bar(bar1, girls, w, label = 'Girls')
plt.bar(bar2, boys, w, label = 'Boys')

plt.xticks(bar1+0.2,X)
plt.xlabel("Groups")
plt.ylabel("Number of Students")
plt.title("Number of Students in each group")
plt.legend()
plt.show()
```




```
import matplotlib.pyplot as plt

import numpy as np



x = ['A', 'B', 'C', 'D']

y1 = np.array([10, 20, 10, 30])

y2 = np.array([20, 25, 15, 25])

y3 = np.array([12, 15, 19, 6])

y4 = np.array([10, 29, 13, 19])


# plot bars in stack manner

plt.bar(x, y1, color='r')

plt.bar(x, y2, bottom=y1, color='b')

plt.bar(x, y3, bottom=y1+y2, color='y')

plt.bar(x, y4, bottom=y1+y2+y3, color='g')

plt.xlabel("Teams")
```

```
plt.ylabel("Score")

plt.legend(["Round 1", "Round 2", "Round 3", "Round 4"])

plt.title("Scores by Teams in 4 Rounds")

plt.show()
```

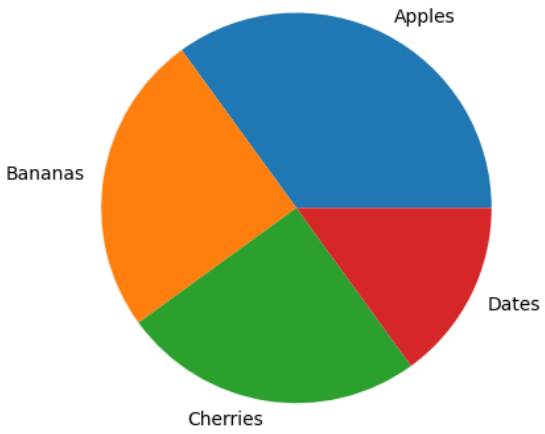


A **Pie Chart** is a circular statistical plot that can display only one series of data. The area of the chart is the total percentage of the given data. The area of slices of the pie represents the percentage of the parts of the data. The slices of pie are called wedges. Matplotlib API has `pie()` function in its `pyplot` module which create a pie chart representing the data in an array.

```
import matplotlib.pyplot as plt
import numpy as np
y = np.array([35, 25, 25, 15])
plt.pie(y)
plt.show()
```

By default the plotting of the first wedge starts from the x-axis and move *counterclockwise*.

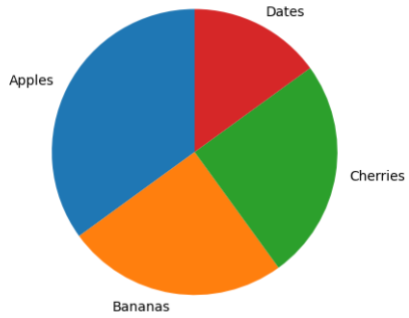
```
import matplotlib.pyplot as plt

import numpy as np

y = np.array([35, 25, 25, 15])

mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels)

plt.show()
```



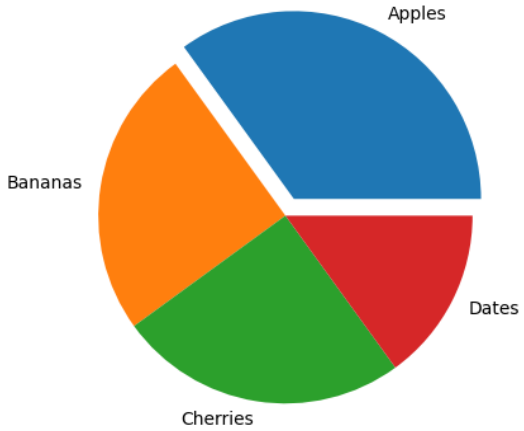
default start angle is at the x-axis, but you can change the start angle by specifying a `startangle` parameter.

The `startangle` parameter is defined with an angle in degrees, default angle is 0:

```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels, startangle = 90)
plt.show()
```




```
import matplotlib.pyplot as plt

import numpy as np

y = np.array([35, 25, 25, 15])

mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

myexplode = [0.1, 0, 0, 0]

plt.pie(y, labels = mylabels, explode = myexplode)

plt.show()
```

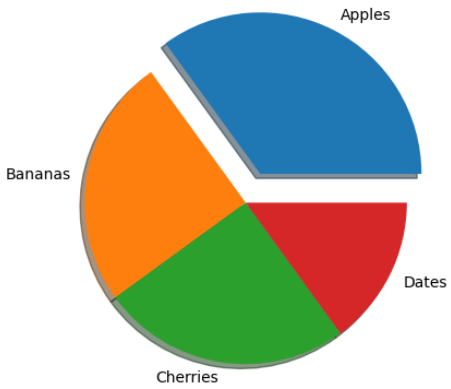



```
import matplotlib.pyplot as plt

import numpy as np

y = np.array([35, 25, 25, 15])

mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
```

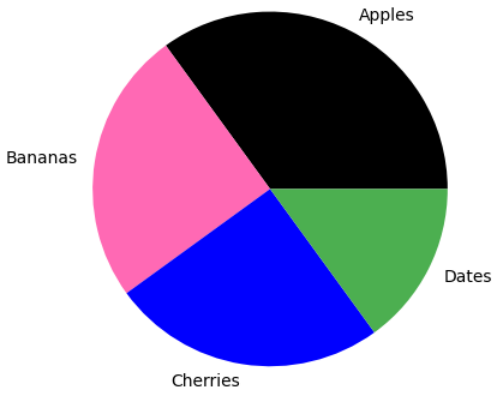
myexplode = [0.2, 0, 0, 0]

plt.pie(y, labels = mylabels, explode = myexplode, shadow = True)

plt.show()

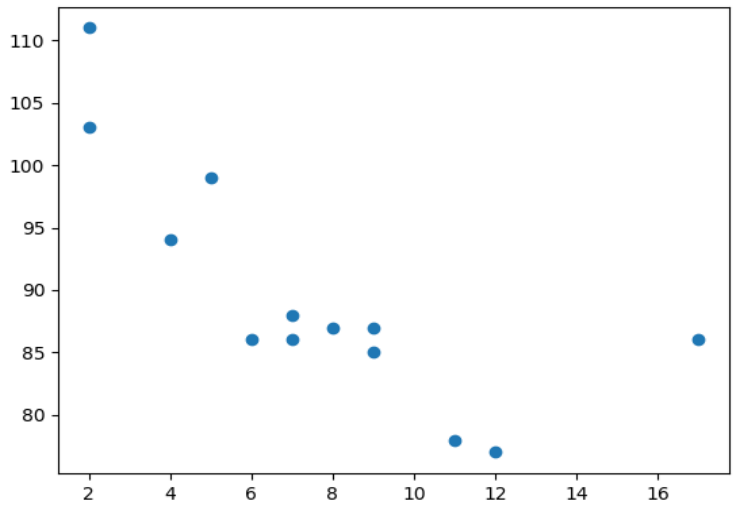



```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
mycolors = ["black", "hotpink", "b", "#4CAF50"]

plt.pie(y, labels = mylabels, colors = mycolors)
plt.show()
```



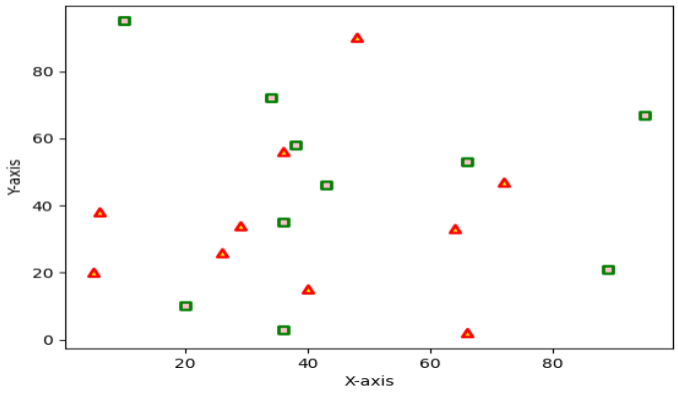
The `scatter()` function plots one dot for each observation. It needs two arrays of the same length, one for the values of the x-axis, and one for values on the y-axis:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])

plt.scatter(x, y)
plt.show()
```



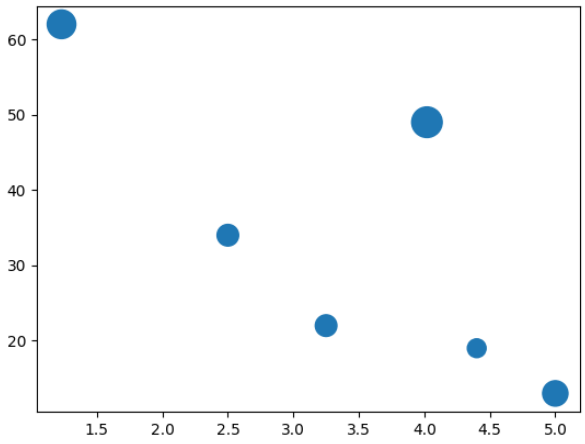

```
import matplotlib.pyplot as plt


# dataset-1

x1 = [89, 43, 36, 36, 95, 10,
      66, 34, 38, 20]


y1 = [21, 46, 3, 35, 67, 95,
      53, 72, 58, 10]


# dataset2

x2 = [26, 29, 48, 64, 6, 5,
```

```
        36, 66, 72, 40]

y2 = [26, 34, 90, 33, 38,
        20, 56, 2, 47, 15]

plt.scatter(x1, y1, c ="pink",
                        linewidths = 2,
                        marker ="s",
                        edgecolor ="green",
                        )
plt.scatter(x2, y2, c ="yellow",
                        linewidths = 2,
                        marker ="^",
                        edgecolor ="red",
                        )
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.show()
```

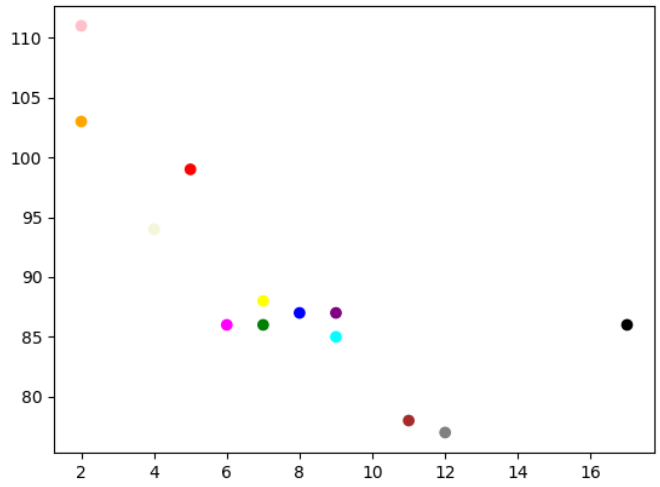


The different orange drinks he sells come from different suppliers and have different profit margins. You can show this additional information in the scatter plot by adjusting the size of the marker.

```
import matplotlib.pyplot as plt
import numpy as np
price = np.asarray([2.50, 1.23, 4.02, 3.25, 5.00, 4.40])
sales_per_day = np.asarray([34, 62, 49, 22, 13, 19])
profit_margin = np.asarray([20, 35, 40, 20, 27.5, 15])
plt.scatter(x=price, y=sales_per_day, s=profit_margin * 10)
plt.show()
```




```
import matplotlib.pyplot as plt

import numpy as np


x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])

y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])

colors =
np.array(["red","green","blue","yellow","pink","black","orange","purple","beige","brown","gray","cyan"
,"magenta"])

plt.scatter(x, y, c=colors)

plt.show()
```



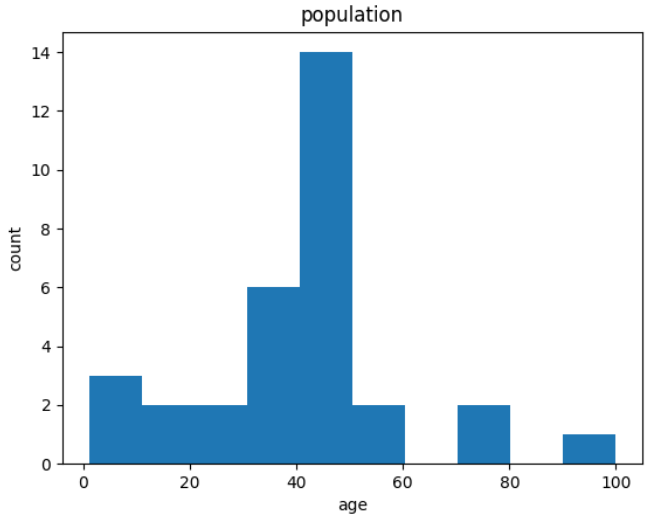
A histogram is basically used to represent data provided in a form of some groups.It is accurate method for the graphical representation of numerical data distribution.It is a type of bar plot where X-axis represents the bin ranges while Y-axis gives information about frequency.

To create a histogram the first step is to create bin of the ranges, then distribute the whole range of the values into a series of intervals, and the count the values which fall into each of the intervals.Bins are clearly identified as consecutive, non-overlapping intervals of variables.

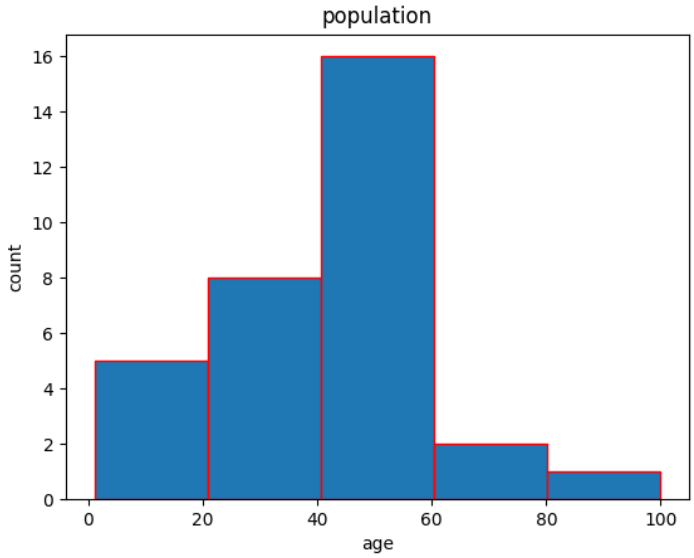
A histogram is a graph showing *frequency* distributions.

height of 250 people.

```
import matplotlib.pyplot as plt

import numpy as np

x = np.array((1,12,22,21,31,2,32,40,14,33,50,44,45,46,45,44,46,47,43,

    42,41,40,43,45,100,3,73,80,55,56,34,45))

plt.hist(x)

plt.xlabel("age")

plt.ylabel("count")

plt.title("population")

plt.show()
```




```
import matplotlib.pyplot as plt
import numpy as np
x = np.array((1,12,22,21,31,2,32,40,14,33,50,44,45,46,45,44,46,47,43,
    42,41,40,43,45,100,3,73,80,55,56,34,45))
plt.hist(x,5,ec="red")
plt.xlabel("age")
plt.ylabel("count")
plt.title("population")
plt.show()
```

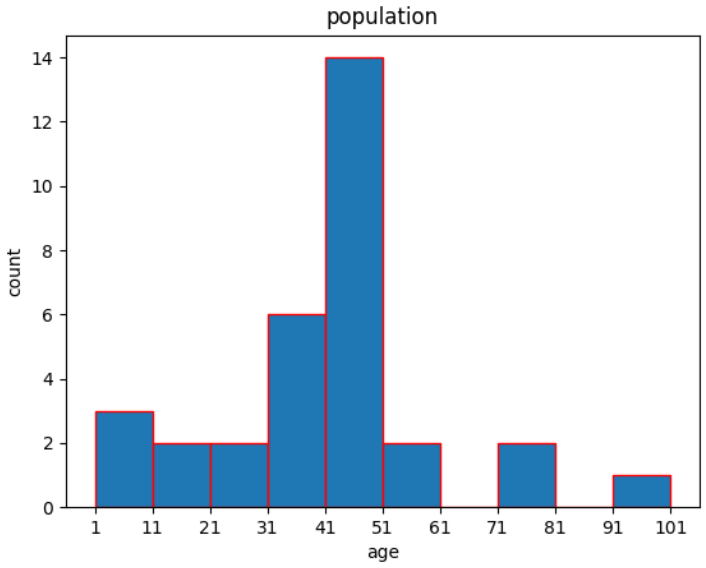


Bins can be integer, sequence, strings

Sequence:

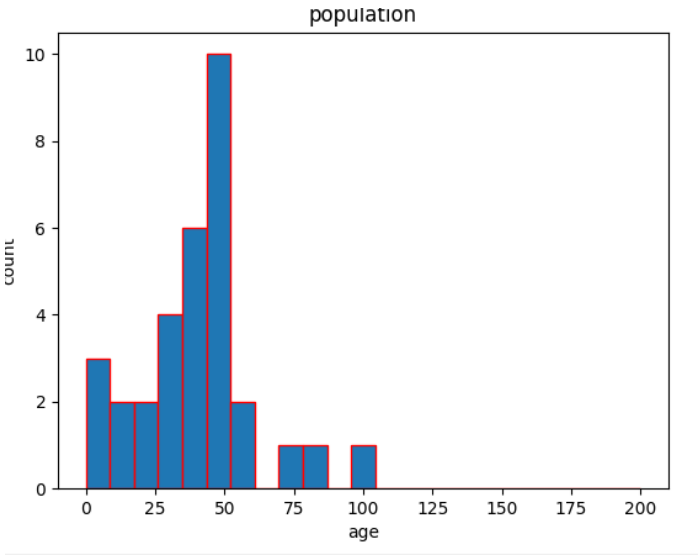
```
import matplotlib.pyplot as plt
```

```
import numpy as np

x = np.array((1,12,22,21,31,2,32,40,14,33,50,44,45,46,45,44,46,47,43,
    42,41,40,43,45,100,3,73,80,55,56,34,45))
y=np.array((1,11,21,31,41,51,61,71,81,91,101))
plt.hist(x,y,ec="red")
plt.xlabel("age")
plt.ylabel("count")
plt.title("population")
plt.xticks(y)
plt.show()
```




```
import matplotlib.pyplot as plt

import numpy as np


x = np.array((1,12,22,21,31,2,32,40,14,33,50,44,45,46,45,44,46,47,43,

    42,41,40,43,45,100,3,73,80,55,56,34,45))

#y=np.array((1,11,21,31,41,51,61,71,81,91,101))

plt.hist(x,"auto",(0,200),ec="red")

plt.xlabel("age")

plt.ylabel("count")
```

```
plt.title("population")

plt.show()
```



Exploratory Data Analysis(EDA):

Exploratory data analysis is a data exploration technique to understand the various aspects of data. *summarizing their main characteristics and plotting them visually.*

Objective of EDA is to filter the data from redundancies.

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

DataFrame. **DataFrame** is a 2-dimensional labeled data structure with columns of potentially different types. You can think of it like a spreadsheet

Pip install pandas

```
import pandas as pd
lst = ['Geeks', 'For', 'Geeks', 'is',
          'portal', 'for', 'Geeks']
 df = pd.DataFrame(lst)
print(df)
o/p:
       0
0   Geeks
1     For
```

```
2   Geeks
3      is
4  portal
5     for
6   Geeks
```

```
import pandas as pd
lst = ['Geeks', 'For', 'Geeks', 'is',
           'portal', 'for', 'Geeks']
df = pd.DataFrame(lst,index=['a','b','c','d','e','f','g'])
print(type(df))
print(df)
```

o/p:

```
<class 'pandas.core.frame.DataFrame'>
        0
a   Geeks
b     For
c   Geeks
d      is
e  portal
f     for
g   Geeks
```

Series can only contain single list with index, whereas **dataframe** can be made of more than one **series** or we can say that a **dataframe** is a collection of **series** that can be used to analyse the data

```
import pandas as pd
lst = ['Geeks', 'For', 'Geeks', 'is',
           'portal', 'for', 'Geeks']
df = pd.Series(lst,index=['a','b','c','d','e','f','g'])
print(type(df))
print(df)
```

O/P:

```
<class 'pandas.core.series.Series'>
a     Geeks
b       For
c     Geeks
d        is
e    portal
f       for
g     Geeks
dtype: object
```

```
import pandas as pd
data={'a':10,'b':20,'c':30}
df=pd.Series(data,index=['b','c','d','a'])
print(df)
```

o/p:

```
b    20.0
c    30.0
d     NaN
a    10.0
```

```
import pandas as pd
data={'a':10,'b':20,'c':30}
df=pd.Series(data,index=['b','c','d','a'])
print(df)
print(df[0])
```

o/p:

```
b    20.0
c    30.0
d     NaN
a    10.0
dtype: float64
20.0
```

Data frame:

It is 2 dimensional data structure. It consists of rows and coloumns.it is similar to excel.

```
import pandas as pd
data = {'Name':['Tom', 'nick', 'krish', 'jack'],
        'Age':[20, 21, 19, 18]}
df = pd.DataFrame(data)
print(df)
```

o/p:

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

```
import pandas as pd
```

```
# Define a dictionary containing employee data
data = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Age':[27, 24, 22, 32],
        'Address':['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
        'Qualification':['Msc', 'MA', 'MCA', 'Phd']}

# Convert the dictionary into DataFrame
df = pd.DataFrame(data)

# select two columns
print(df[['Name', 'Qualification']])
o/p:
Name Qualification
0     Jai          Msc
1  Princi           MA
2  Gaurav          MCA
3    Anuj          Phd
```

import pandas as pd

iris=pd.read_csv(r"C:\Users\SIREESHA\Desktop\Iris.csv")

pd.DataFrame(iris)

print(iris.head())

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	**1**	**5.1**	**3.5**	**1.4**	**0.2**	**Iris-setosa**
1	**2**	**4.9**	**3.0**	**1.4**	**0.2**	**Iris-setosa**
2	**3**	**4.7**	**3.2**	**1.3**	**0.2**	**Iris-setosa**
3	**4**	**4.6**	**3.1**	**1.5**	**0.2**	**Iris-setosa**
4	**5**	**5.0**	**3.6**	**1.4**	**0.2**	**Iris-setosa**

print(iris.tail())

```
Id  SepalLengthCm  ...  PetalWidthCm        Species
145  146          6.7  ...          2.3  Iris-virginica
146  147          6.3  ...          1.9  Iris-virginica
147  148          6.5  ...          2.0  Iris-virginica
148  149          6.2  ...          2.3  Iris-virginica
149  150          5.9  ...          1.8  Iris-virginica
```

```
print(iris.describe())
```

```
       Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
count  150.000000     150.000000    150.000000     150.000000    150.000000
mean    75.500000       5.843333      3.054000       3.758667      1.198667
std     43.445368       0.828066      0.433594       1.764420      0.763161
min      1.000000       4.300000      2.000000       1.000000      0.100000
25%     38.250000       5.100000      2.800000       1.600000      0.300000
50%     75.500000       5.800000      3.000000       4.350000      1.300000
75%    112.750000       6.400000      3.300000       5.100000      1.800000
max    150.000000       7.900000      4.400000       6.900000      2.500000
```

```
print(iris.iloc[0:3,0:2])
```

```
Id  SepalLengthCm
0   1            5.1
1   2            4.9
2   3            4.7
```

```
a=iris.drop('SepalLengthCm',axis=1)
```

print(a.head())

	Id	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	3.5	1.4	0.2	Iris-setosa
1	2	3.0	1.4	0.2	Iris-setosa
2	3	3.2	1.3	0.2	Iris-setosa
3	4	3.1	1.5	0.2	Iris-setosa
4	5	3.6	1.4	0.2	Iris-setosa

b=iris.drop([1,2,3],axis=0)

print(b.head())

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
5	6	5.4	3.9	1.7	0.4	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa
7	8	5.0	3.4	1.5	0.2	Iris-setosa

print(iris.mean())

print(iris.min())

Id 75.500000

SepalLengthCm 5.843333

SepalWidthCm 3.054000

PetalLengthCm 3.758667

PetalWidthCm 1.198667

dtype: float64

Id 1

SepalLengthCm 4.3

SepalWidthCm 2.0

PetalLengthCm 1.0

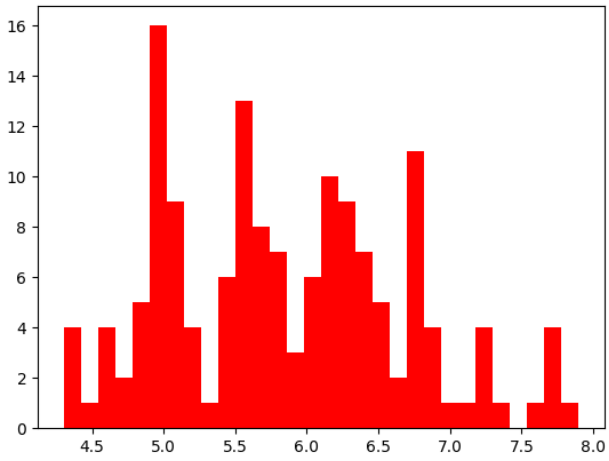
PetalWidthCm 0.1

Species Iris-setosa

dtype: object

plt.hist(iris['SepalLengthCm'],bins=30,color='red')

plt.show()



The Steps In Exploratory Data Analysis In Python

- Description of data
- Handling missing data
- Handling outliers
- Understanding relationships and new insights through plots

In Pandas, we can apply describe() on a DataFrame which helps in generating descriptive statistics that summarize the central tendency, dispersion, and shape of a dataset's distribution, excluding NaN values

There are 3 types of EDA

1.Univariate

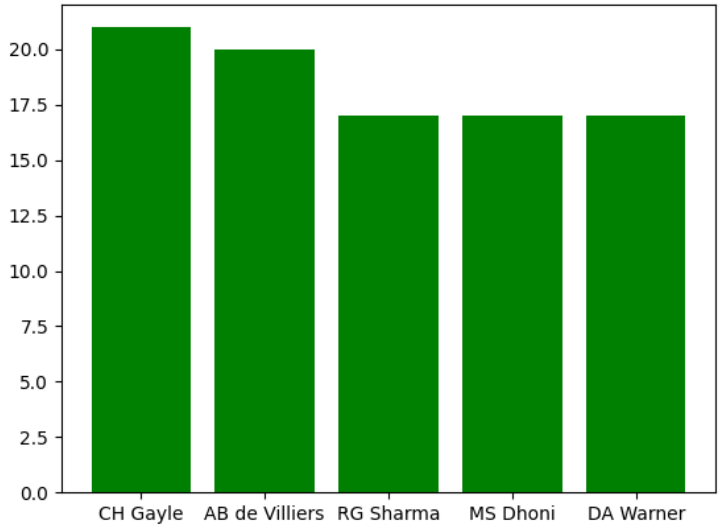
2. bivariate
3.multivariate
It handles with under fitting and over fitting of data.
Efficient will be almost zero if we didn't handle

Ipl data set

```
>>> ipl=pd.read_csv(r"C:\Users\SIREESHA\Desktop\matches.csv")
>>> ipl.head()
  id  season       city  ...        umpire1        umpire2 umpire3
0   1    2017  Hyderabad  ...     AY Dandekar       NJ Llong     NaN
1   2    2017       Pune  ...  A Nand Kishore         S Ravi     NaN
2   3    2017     Rajkot  ...     Nitin Menon      CK Nandan     NaN
3   4    2017     Indore  ...    AK Chaudhary  C Shamshuddin     NaN
4   5    2017  Bangalore  ...             NaN            NaN     NaN

[5 rows x 18 columns]
>>> ipl.shape
(756, 18)
>>> ipl.describe()
                id       season  dl_applied  win_by_runs  win_by_wickets
count    756.000000   756.000000  756.000000   756.000000      756.000000
mean    1792.178571  2013.444444    0.025132    13.283069        3.350529
std     3464.478148     3.366895    0.156630    23.471144        3.387963
min        1.000000  2008.000000    0.000000     0.000000        0.000000
25%      189.750000  2011.000000    0.000000     0.000000        0.000000
50%      378.500000  2013.000000    0.000000     0.000000        4.000000
75%      567.250000  2016.000000    0.000000    19.000000        6.000000
max    11415.000000  2019.000000    1.000000   146.000000       10.000000
>>> ipl.head()
  id  season       city  ...        umpire1        umpire2 umpire3
0   1    2017  Hyderabad  ...     AY Dandekar       NJ Llong     NaN
1   2    2017       Pune  ...  A Nand Kishore         S Ravi     NaN
2   3    2017     Rajkot  ...     Nitin Menon      CK Nandan     NaN
3   4    2017     Indore  ...    AK Chaudhary  C Shamshuddin     NaN
4   5    2017  Bangalore  ...             NaN            NaN     NaN

[5 rows x 18 columns]
>>> pd.DataFrame(ipl)
      id  season  ...        umpire2              umpire3
0      1    2017  ...       NJ Llong                  NaN
1      2    2017  ...         S Ravi                  NaN
```

```
2       3    2017  ...       CK Nandan                    NaN
3       4    2017  ...   C Shamshuddin                    NaN
4       5    2017  ...             NaN                    NaN
..    ...     ...  ...             ...                    ...
751  11347    2019  ...        O Nandan                 S Ravi
752  11412    2019  ...     Nitin Menon              Ian Gould
753  11413    2019  ...             NaN                    NaN
754  11414    2019  ...  Bruce Oxenford  Chettithody Shamshuddin
755  11415    2019  ...       Ian Gould             Nigel Llong

[756 rows x 18 columns]

>>> pd.set_option('max_columns', None)
>>> pd.set_option('max_columns', 18)
>>> ipl.head()
   id  season       city        date                 team1  \
0   1    2017  Hyderabad  2017-04-05         Sunrisers Hyderabad
1   2    2017       Pune  2017-04-06              Mumbai Indians
2   3    2017     Rajkot  2017-04-07               Gujarat Lions
3   4    2017     Indore  2017-04-08       Rising Pune Supergiant
4   5    2017  Bangalore  2017-04-08  Royal Challengers Bangalore

                     team2                toss_winner toss_decision  \
0  Royal Challengers Bangalore  Royal Challengers Bangalore         field
1      Rising Pune Supergiant       Rising Pune Supergiant         field
2       Kolkata Knight Riders        Kolkata Knight Riders         field
3            Kings XI Punjab             Kings XI Punjab         field
4           Delhi Daredevils  Royal Challengers Bangalore           bat

   result  dl_applied                     winner  win_by_runs  \
0  normal           0         Sunrisers Hyderabad           35
1  normal           0       Rising Pune Supergiant            0
2  normal           0        Kolkata Knight Riders            0
3  normal           0             Kings XI Punjab            0
4  normal           0  Royal Challengers Bangalore           15

   win_by_wickets player_of_match                                      venue  \
0               0    Yuvraj Singh  Rajiv Gandhi International Stadium, Uppal
1               7      SPD Smith    Maharashtra Cricket Association Stadium
2              10         CA Lynn     Saurashtra Cricket Association Stadium
```

```
3          6     GJ Maxwell              Holkar Cricket Stadium
4          0      KM Jadhav               M Chinnaswamy Stadium

          umpire1        umpire2 umpire3
0     AY Dandekar       NJ Llong     NaN
1  A Nand Kishore         S Ravi     NaN
2     Nitin Menon      CK Nandan     NaN
3    AK Chaudhary  C Shamshuddin     NaN
4             NaN           NaN     NaN
#man of matches
>>> ipl['player_of_match'].value_counts()
CH Gayle          21
AB de Villiers    20
RG Sharma         17
MS Dhoni          17
DA Warner         17
                  ..
PD Collingwood     1
NV Ojha            1
AC Voges           1
J Theron           1
S Hetmyer          1

ipl['player_of_match'].value_counts()[0:5]
CH Gayle          21
AB de Villiers    20
RG Sharma         17
MS Dhoni          17
DA Warner         17

>>> list(ipl['player_of_match'].value_counts()[0:5].keys())
['CH Gayle', 'AB de Villiers', 'RG Sharma', 'MS Dhoni', 'DA Warner']
>>> plt.bar(list(ipl['player_of_match'].value_counts()[0:5].keys()),
list(ipl['player_of_match'].value_counts()[0:5]), color='g')
<BarContainer object of 5 artists>
>>> plt.show()
```




```
>>> ipl['result'].value_counts()
normal       743
tie            9
no result      4
>>> ipl['toss_winner'].value_counts()
Mumbai Indians                98
Kolkata Knight Riders         92
Chennai Super Kings           89
Royal Challengers Bangalore   81
Kings XI Punjab               81
Delhi Daredevils              80
Rajasthan Royals              80
Sunrisers Hyderabad           46
Deccan Chargers               43
Pune Warriors                 20
```

```
Gujarat Lions                15
Delhi Capitals               10
Kochi Tuskers Kerala          8
Rising Pune Supergiants       7
Rising Pune Supergiant        6
Name: toss_winner, dtype: int64
>>> battle_first=ipl[ipl['win_by_runs']!=0]
>>> ipl.head()
   id  season       city        date                    team1  \
0   1    2017  Hyderabad  2017-04-05         Sunrisers Hyderabad
1   2    2017       Pune  2017-04-06              Mumbai Indians
2   3    2017     Rajkot  2017-04-07               Gujarat Lions
3   4    2017     Indore  2017-04-08       Rising Pune Supergiant
4   5    2017  Bangalore  2017-04-08  Royal Challengers Bangalore

                     team2                toss_winner toss_decision  \
0  Royal Challengers Bangalore  Royal Challengers Bangalore         field
1       Rising Pune Supergiant       Rising Pune Supergiant         field
2        Kolkata Knight Riders        Kolkata Knight Riders         field
3             Kings XI Punjab              Kings XI Punjab         field
4           Delhi Daredevils  Royal Challengers Bangalore           bat

   result  dl_applied                     winner  win_by_runs  \
0  normal           0         Sunrisers Hyderabad           35
1  normal           0       Rising Pune Supergiant           0
```

```
2  normal           0        Kolkata Knight Riders            0
3  normal           0             Kings XI Punjab             0
4  normal           0  Royal Challengers Bangalore           15

   win_by_wickets player_of_match                                venue  \
0               0    Yuvraj Singh  Rajiv Gandhi International Stadium, Uppal
1               7       SPD Smith    Maharashtra Cricket Association Stadium
2              10         CA Lynn     Saurashtra Cricket Association Stadium
3               6      GJ Maxwell                   Holkar Cricket Stadium
4               0       KM Jadhav                    M Chinnaswamy Stadium

         umpire1        umpire2 umpire3
0     AY Dandekar       NJ Llong     NaN
1  A Nand Kishore         S Ravi     NaN
2     Nitin Menon      CK Nandan     NaN
3    AK Chaudhary  C Shamshuddin     NaN
4             NaN            NaN     NaN
>>> battle_first.head()
    id  season       city        date                 team1  \
0    1    2017  Hyderabad  2017-04-05         Sunrisers Hyderabad
4    5    2017  Bangalore  2017-04-08  Royal Challengers Bangalore
8    9    2017       Pune  2017-04-11            Delhi Daredevils
13  14    2017    Kolkata  2017-04-15        Kolkata Knight Riders
14  15    2017      Delhi  2017-04-15            Delhi Daredevils
```

	team2	toss_winner	toss_decision	\
0	Royal Challengers Bangalore	Royal Challengers Bangalore	field	
4	Delhi Daredevils	Royal Challengers Bangalore	bat	
8	Rising Pune Supergiant	Rising Pune Supergiant	field	
13	Sunrisers Hyderabad	Sunrisers Hyderabad	field	
14	Kings XI Punjab	Delhi Daredevils	bat	

	result	dl_applied	winner	win_by_runs	\
0	normal	0	Sunrisers Hyderabad	35	
4	normal	0	Royal Challengers Bangalore	15	
8	normal	0	Delhi Daredevils	97	
13	normal	0	Kolkata Knight Riders	17	
14	normal	0	Delhi Daredevils	51	

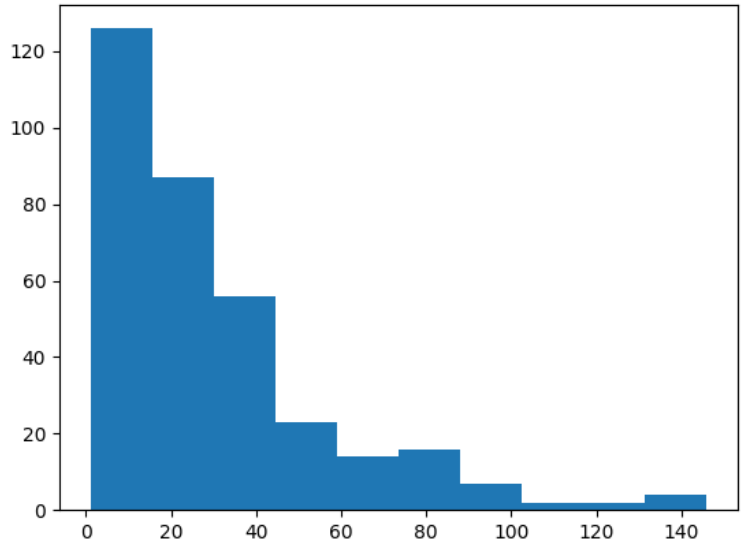
	win_by_wickets	player_of_match	venue	\
0	0	Yuvraj Singh	Rajiv Gandhi International Stadium, Uppal	
4	0	KM Jadhav	M Chinnaswamy Stadium	
8	0	SV Samson	Maharashtra Cricket Association Stadium	
13	0	RV Uthappa	Eden Gardens	
14	0	CJ Anderson	Feroz Shah Kotla	

	umpire1	umpire2	umpire3
0	AY Dandekar	NJ Llong	NaN
4	NaN	NaN	NaN
8	AY Dandekar	S Ravi	NaN

13 AY Dandekar NJ Llong NaN

14 YC Barde Nitin Menon NaN

```
>>> plt.hist(battle_first['win_by_runs'])
(array([126.,  87.,  56.,  23.,  14.,  16.,   7.,   2.,   2.,   4.]), array([  1. ,  15.5,  30. ,  44.5,  59. ,
73.5,  88. , 102.5, 117. ,
       131.5, 146. ]), <BarContainer object of 10 artists>)
>>> plt.show()
```




```
>>> battle_first['winner'].value_counts()
```

Mumbai Indians 57

Chennai Super Kings 52

Kings XI Punjab 38

Kolkata Knight Riders 36

Royal Challengers Bangalore 35

Sunrisers Hyderabad 30

Rajasthan Royals 27

Delhi Daredevils 25

Deccan Chargers 18

Pune Warriors 6

Rising Pune Supergiant 5

Delhi Capitals 3

Kochi Tuskers Kerala 2

Rising Pune Supergiants 2

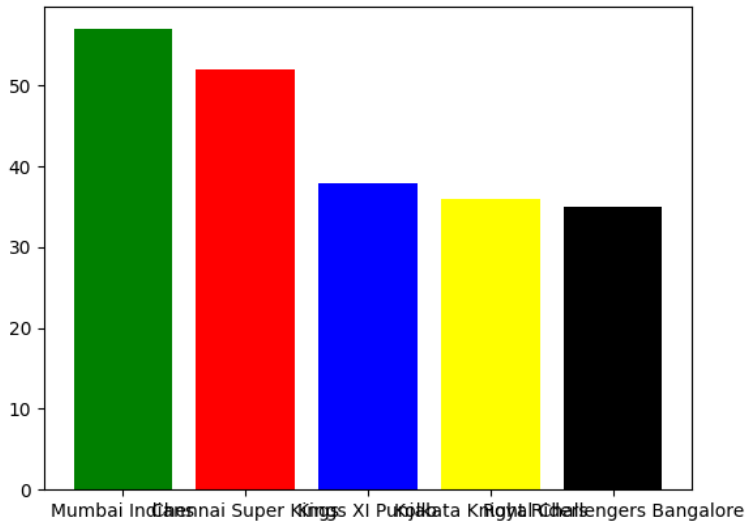
Gujarat Lions 1

Name: winner, dtype: int64

>>>
plt.bar(list(battle_first['winner'].value_counts()[0:5].keys()),list(battle_first['winner'].value_counts()[0:5]),color=['green','red','blue','yellow','black'])

<BarContainer object of 5 artists>

>>> plt.show()



>>> ipl['season'].value_counts()

2013 76

2012 74

```
2011    73
2010    60
2014    60
2016    60
2018    60
2019    60
2017    59
2015    59
2008    58
2009    57
>>> a=ipl['city'].value_counts()
>>> a.head()
Mumbai       101
Kolkata       77
Delhi         74
Bangalore     66
Hyderabad     64
>>> import numpy as np
>>> np.sum(ipl['toss_winner']==ipl['winner'])
393
>>> 325/686
0.4737609329446064
deliveries=pd.read_csv(r"C:\Users\SIREESHA\Desktop\deliveries.csv")
>>> pd.set_option("max_columns",21)
>>> deliveries.head()
```

	match_id	inning	batting_team	bowling_team	over	\
0	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	
1	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	
2	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	
3	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	
4	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	

	ball	batsman	non_striker	bowler	is_super_over	wide_runs	bye_runs	\
0	1	DA Warner	S Dhawan	TS Mills	0	0	0	
1	2	DA Warner	S Dhawan	TS Mills	0	0	0	
2	3	DA Warner	S Dhawan	TS Mills	0	0	0	
3	4	DA Warner	S Dhawan	TS Mills	0	0	0	
4	5	DA Warner	S Dhawan	TS Mills	0	2	0	

	legbye_runs	noball_runs	penalty_runs	batsman_runs	extra_runs	\
0	0	0	0	0	0	
1	0	0	0	0	0	
2	0	0	0	4	0	
3	0	0	0	0	0	
4	0	0	0	0	2	

	total_runs	player_dismissed	dismissal_kind	fielder
0	0	NaN	NaN	NaN
1	0	NaN	NaN	NaN
2	4	NaN	NaN	NaN

```
3          0          NaN          NaN     NaN
4          2          NaN          NaN     NaN
>>>>>> deliveries.shape
(150460, 21)
>>> deliveries['match_id'].unique()
array([  1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,  13,
        14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,  26,
        27,  28,  29,  30,  31,  32,  33,  34,  35,  36,  37,  38,  39,
        40,  41,  42,  43,  44,  45,  46,  47,  48,  49,  50,  51,  52,
        53,  54,  55,  56,  57,  58,  59,  60,  61,  62,  63,  64,  65,
        66,  67,  68,  69,  70,  71,  72,  73,  74,  75,  76,  77,  78,
        79,  80,  81,  82,  83,  84,  85,  86,  87,  88,  89,  90,  91,
        92,  93,  94,  95,  96,  97,  98,  99, 100, 101, 102, 103, 104,
       105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117,
       118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130,
       131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143,
       144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156,
       157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169,
       170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182,
       183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195,
       196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208,
       209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221,
       222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234,
       235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247,
       248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260,
```

261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273,
274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286,
287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299,
300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312,
313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325,
326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338,
339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351,
352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364,
365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377,
378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390,
391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403,
404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416,
417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429,
430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442,
443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455,
456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468,
469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481,
482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494,
495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507,
508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520,
521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533,
534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546,
547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559,
560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572,
573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585,

586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598,

599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611,

612, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623, 624,

625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636],

```
>>> match_1=deliveries[deliveries['match_id']==1]
>>> match_1.head()
```

	match_id	inning	batting_team	bowling_team	over	\
0	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	
1	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	
2	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	
3	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	
4	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	

	ball	batsman	non_striker	bowler	is_super_over	wide_runs	bye_runs	\
0	1	DA Warner	S Dhawan	TS Mills	0	0	0	
1	2	DA Warner	S Dhawan	TS Mills	0	0	0	
2	3	DA Warner	S Dhawan	TS Mills	0	0	0	
3	4	DA Warner	S Dhawan	TS Mills	0	0	0	
4	5	DA Warner	S Dhawan	TS Mills	0	2	0	

	legbye_runs	noball_runs	penalty_runs	batsman_runs	extra_runs	\
0	0	0	0	0	0	
1	0	0	0	0	0	
2	0	0	0	4	0	
3	0	0	0	0	0	

```
4          0          0          0          0          2


   total_runs player_dismissed dismissal_kind fielder
0          0              NaN           NaN     NaN
1          0              NaN           NaN     NaN
2          4              NaN           NaN     NaN
3          0              NaN           NaN     NaN
4          2              NaN           NaN     NaN
>>> match_1.shape
(248, 21)
>>> srh=match_1[match_1['inning']==1]
>>> srh.head()
   match_id  inning         batting_team                bowling_team  over  \
0         1       1  Sunrisers Hyderabad  Royal Challengers Bangalore     1
1         1       1  Sunrisers Hyderabad  Royal Challengers Bangalore     1
2         1       1  Sunrisers Hyderabad  Royal Challengers Bangalore     1
3         1       1  Sunrisers Hyderabad  Royal Challengers Bangalore     1
4         1       1  Sunrisers Hyderabad  Royal Challengers Bangalore     1


   ball    batsman non_striker    bowler  is_super_over  wide_runs  bye_runs  \
0     1  DA Warner    S Dhawan  TS Mills              0          0         0
1     2  DA Warner    S Dhawan  TS Mills              0          0         0
2     3  DA Warner    S Dhawan  TS Mills              0          0         0
3     4  DA Warner    S Dhawan  TS Mills              0          0         0
4     5  DA Warner    S Dhawan  TS Mills              0          2         0
```

```
   legbye_runs  noball_runs  penalty_runs  batsman_runs  extra_runs  \
0            0            0             0             0           0
1            0            0             0             0           0
2            0            0             0             4           0
3            0            0             0             0           0
4            0            0             0             0           2

   total_runs player_dismissed dismissal_kind fielder
0           0              NaN            NaN     NaN
1           0              NaN            NaN     NaN
2           4              NaN            NaN     NaN
3           0              NaN            NaN     NaN
4           2              NaN            NaN     NaN
>>> srh['batsman_runs'].value_counts()
1    57
0    32
4    17
6     9
2     9
3     1
Name: batsman_runs, dtype: int64
```

1-single-57

0-dot ball-32

```
>>> srh['dismissal_kind'].value_counts()
```

caught 3

bowled 1

```
>>> rc=match_1[match_1['inning']==2]
>>> rc.head()
    match_id  inning             batting_team        bowling_team  over  \
125         1       2  Royal Challengers Bangalore  Sunrisers Hyderabad     1
126         1       2  Royal Challengers Bangalore  Sunrisers Hyderabad     1
127         1       2  Royal Challengers Bangalore  Sunrisers Hyderabad     1
128         1       2  Royal Challengers Bangalore  Sunrisers Hyderabad     1
129         1       2  Royal Challengers Bangalore  Sunrisers Hyderabad     1

    ball       batsman    non_striker   bowler  is_super_over  wide_runs  \
125     1      CH Gayle  Mandeep Singh  A Nehra              0          0
126     2  Mandeep Singh      CH Gayle  A Nehra              0          0
127     3  Mandeep Singh      CH Gayle  A Nehra              0          0
128     4  Mandeep Singh      CH Gayle  A Nehra              0          0
129     5  Mandeep Singh      CH Gayle  A Nehra              0          0

    bye_runs  legbye_runs  noball_runs  penalty_runs  batsman_runs  \
125         0            0            0             0             1
126         0            0            0             0             0
127         0            0            0             0             0
128         0            0            0             0             2
129         0            0            0             0             4
```

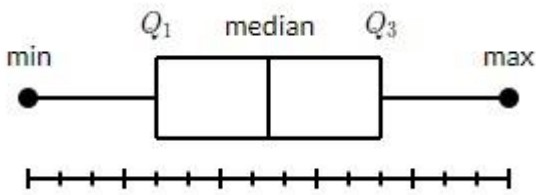
	extra_runs	total_runs	player_dismissed	dismissal_kind	fielder
125	0	1	NaN	NaN	NaN
126	0	0	NaN	NaN	NaN
127	0	0	NaN	NaN	NaN
128	0	2	NaN	NaN	NaN
129	0	4	NaN	NaN	NaN

```
>>> rc['batsman_runs'].value_counts()
0    49
1    44
4    15
6     8
2     7
>>> rc['dismissal_kind'].value_counts()
caught     6
bowled     2
run out    2
```

box plot: 5 number summary

min, max,median,Q1,Q2,Q3

A box plot which is also known as a whisker plot displays a summary of a set of data containing the minimum, first quartile, median, third quartile, and maximum. In a box plot, we draw a box from the first quartile to the third quartile. A vertical line goes through the box at the median. The whiskers go from each quartile to the minimum or maximum.



22,25,17,19,33,64,23,17,20,18

ASCENDING ORDER

17 17 18 19 20 22 23 25 33 64

MEDIAN=20+22/2=21

Q2=21

Q1=FIRST QUARTILE =25%=18

Q3=75%=25

OUTLIER

HIGHER OUTLIER=Q3+1.5*IQR

IQR=Q3-Q1=25-18=7

=25+1.5*7=35.5

LOWER OUTLIER=Q1-1.5*IQR

=18-1.5*7

=7.5

OUTLIERS IN THE DATA?

RIGHT SIDE>35.5

LEFT SIDE<7.5

ONE OUTLIER

MAX=33

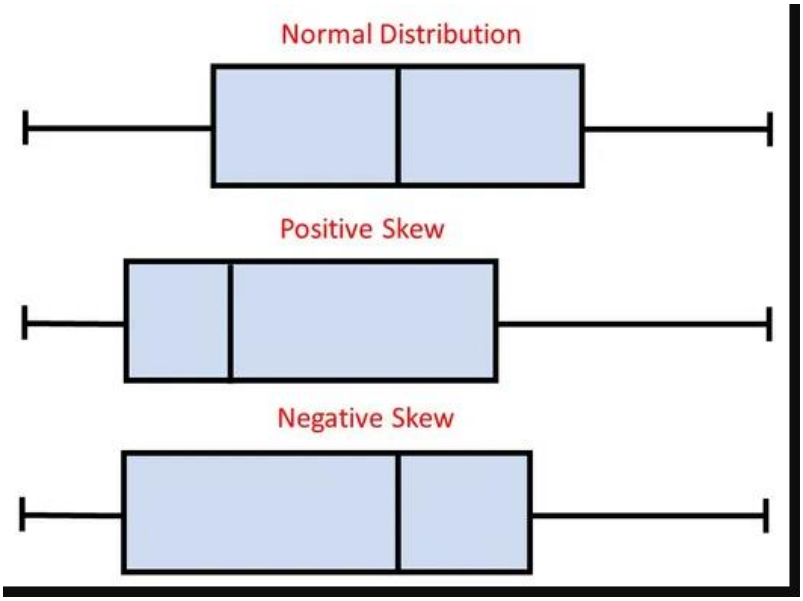
MIN=17

An outlier is an observation that is numerically distant from the rest of the data. When reviewing a box plot, an outlier is defined as a **data point** that is located outside the whiskers of the box plot.

type of chart often used in explanatory data analysis. Box plots visually show the distribution of numerical data and skewness through displaying the data quartiles (or percentiles) and averages.

Box plots show the five-number summary of a set of data: including the minimum score, first (lower) quartile, median, third (upper) quartile, and maximum score.

The box plot shape will show if a statistical data set is normally distributed or skewed.



When the median is in the middle of the box, and the whiskers are about the same on both sides of the box, then the distribution is symmetric.

When the median is closer to the bottom of the box, and if the whisker is shorter on the lower end of the box, then the distribution is positively skewed (skewed right).

When the median is closer to the top of the box, and if the whisker is shorter on the upper end of the box, then the distribution is negatively skewed (skewed left).

An outlier is an observation that is numerically distant from the rest of the data.

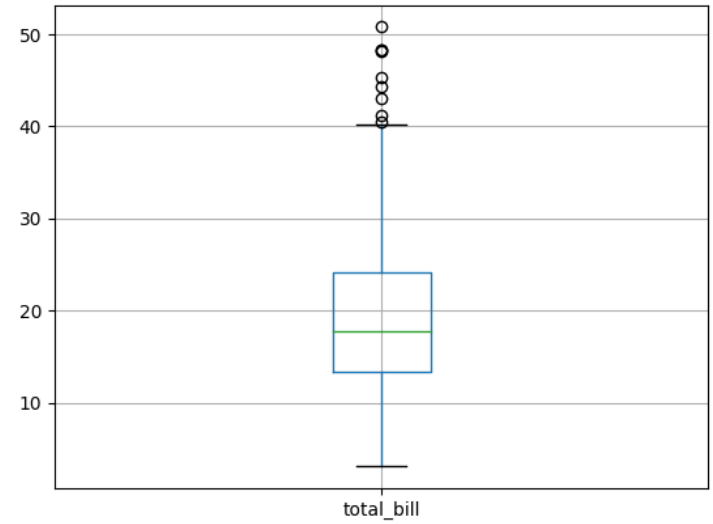
When reviewing a box plot, an outlier is defined as a data point that is located outside the whiskers of the box plot.

```
import matplotlib.pyplot as plt

import numpy as np

>>> data=pd.read_csv("C:\Users\SIREESHA\Desktop\tips.csv")

data.head()

   total_bill   tip     sex smoker  day    time  size

0       16.99  1.01  Female     No  Sun  Dinner     2

1       10.34  1.66    Male     No  Sun  Dinner     3

2       21.01  3.50    Male     No  Sun  Dinner     3

3       23.68  3.31    Male     No  Sun  Dinner     2

4       24.59  3.61  Female     No  Sun  Dinner     4
```

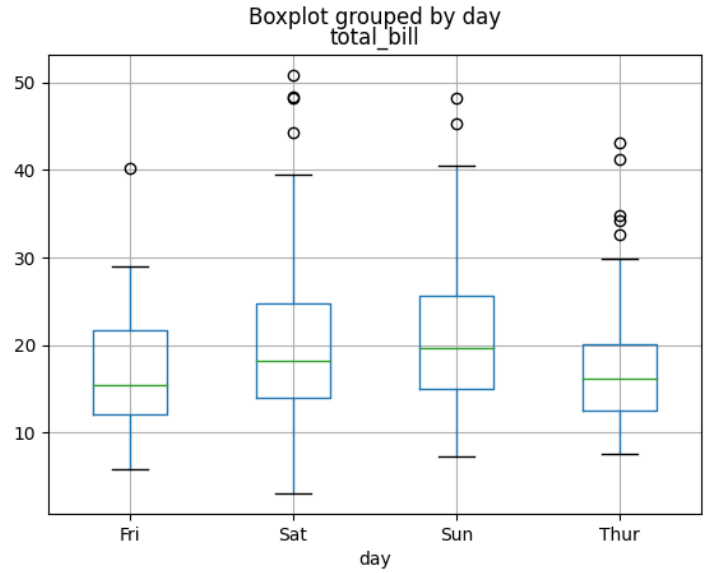
>>> data.boxplot('total_bill')

<AxesSubplot:>

>>> plt.show()



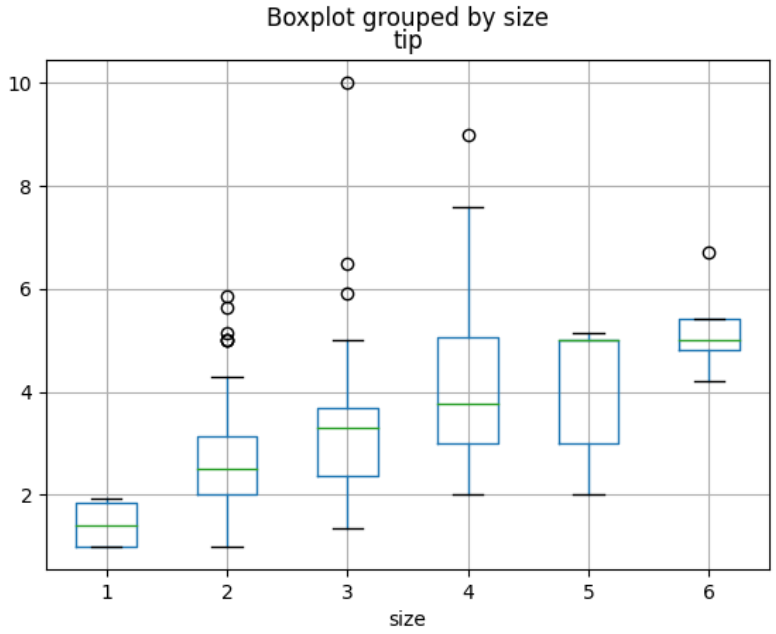
data.boxplot(by='day',columns=['total_bill'])



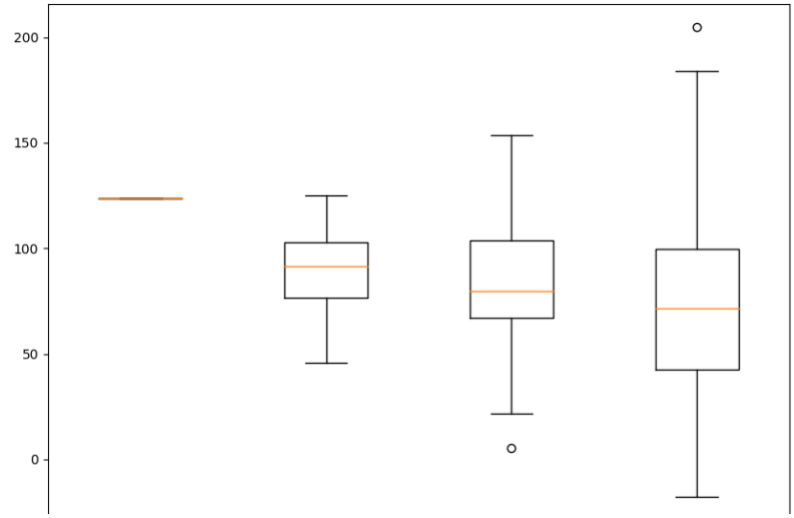
data.boxplot(by='size',column=['tip'])

<AxesSubplot:title={'center':'tip'}, xlabel='size'>

>>> plt.show()




```
data_1 = np.random.normal(100,10, 200)

data_2 = np.random.normal(90, 20, 200)

data_3 = np.random.normal(80, 30, 200)

data_4 = np.random.normal(70, 40, 200)

data = [data_1, data_2, data_3, data_4]

fig = plt.figure(figsize =(10, 7))

bp = plt.boxplot(data)

# show plot

plt.show()
```



1. Outliers can be removed from the data using statistical methods of IQR, Z-Score and Data Smoothing.

There are Two Methods for Outlier Treatment

1. Interquartile Range(IQR) Method
2. Z Score method

6.1 — IQR Method

Using IQR we can find outlier.
Data point that falls outside of 1.5 times of an Interquartile range above the 3rd quartile (Q3) and below the 1st quartile (Q1)

```
import pandas as pd

>>> data=pd.read_csv(r"C:\Users\SIREESHA\Desktop\mtcars.csv")

>>> data.head()

data.columns

Index(['model', 'mpg', 'cyl', 'disp', 'hp', 'drat', 'wt', 'qsec', 'vs', 'am',

       'gear', 'carb'],

      dtype='object')
```

```
>>> data.shape
(32, 12)
import matplotlib.pyplot as plt
plt.boxplot(data.hp)
plt.show()
>>> Q1=data['hp'].quantile(0.25)
>>> Q1
96.5
>>> Q3=data['hp'].quantile(0.75)
>>> Q3
180.0
>>> IQR=Q3-Q1
Lower_Whisker = Q1-1.5*IQR
>>> Upper_Whisker = Q3+1.5*IQR
>>> print(Lower_Whisker, Upper_Whisker)
-28.75 305.25
>>> data = data[data['hp']< Upper_Whisker]
>>> data.shape
(31, 12)
plt.boxplot(data.hp)
plt.show()
```

Heatmap:

A heatmap is a two-dimensional graphical representation of data where the individual values that are contained in a matrix are represented as colors. The seaborn python package allows the creation of annotated heatmaps. To create a heatmap in Python, we can use the seaborn library.

The seaborn library is built on top of Matplotlib. Seaborn library provides a high-level data visualization.

A correlation matrix is a tabular data representing the 'correlations' between pairs of variables in a given data. Each row and column represents a variable, and each value in this matrix is the correlation coefficient between the variables represented by the corresponding row and column.

The Correlation matrix is an important data analysis metric that is computed to summarize data to understand the relationship between various variables and make decisions accordingly.

It is also an important pre-processing step in Machine Learning pipelines to compute and analyze the correlation matrix where dimensionality reduction is desired on a high-dimension data.

We mentioned how each cell in the correlation matrix is a '**correlation coefficient**' between the two variables corresponding to the row and column of the cell.

A correlation coefficient is a number that denotes the strength of the relationship between two variables.

There are several types of correlation coefficients, but the most common of them all is the Pearson's coefficient

It is defined as the covariance between two variables divided by the product of the standard deviations of the two variables.

$$\rho(X,Y) = \frac{COV(X,Y)}{\sigma_X \sigma_Y}$$

Where the covariance between X and Y COV(X, Y) is further defined as the 'expected value of the product of the deviations of X and Y from their respective means'.
The formula for covariance would make it clearer.

$$COV(X,Y) = E[(X - \mu_X)(Y - \mu_Y)]$$

So the formula for Pearson's correlation would then become:

$$\rho(X,Y) = \frac{E[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y}$$

The value of ρ lies between -1 and +1.
Values nearing +1 indicate the presence of a strong positive relation between X and Y,

whereas those nearing -1 indicate a strong negative relation between X and Y.
Values near to zero mean there is an absence of any relationship between X and Y.

Finding the correlation matrix of the given data

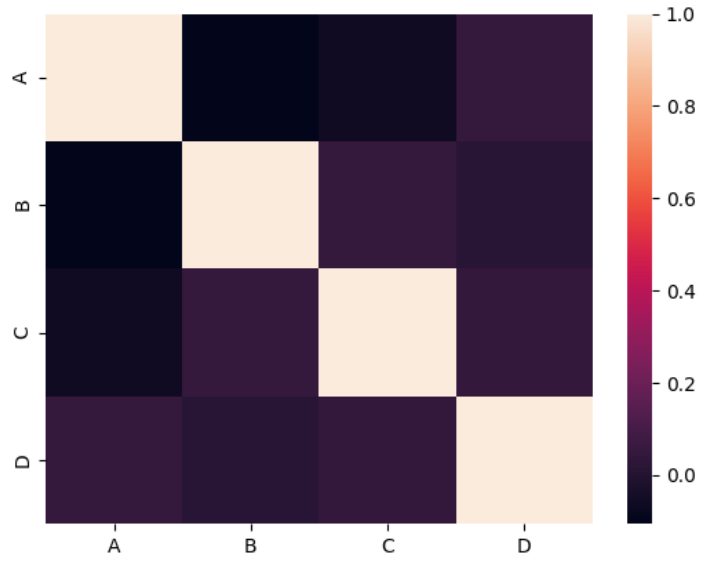
Let us generate random data for two variables and then construct the correlation matrix for them.

```
>>> import numpy as np

>>> X = np.random.randn(10)

>>> Y = np.random.randn(10)

>>> X

array([ 0.89797329,  0.58604905,  0.19492189, -1.40820353, -0.28923645,

        0.93423768, -1.57343296, -0.16874151,  1.75162599, -0.88760521])

>>> Y

array([ 1.76468401,  1.04163625,  0.6822118 ,  1.25972939, -0.82160922,

       -0.1723042 ,  0.53830215,  0.99552717, -0.1263286 ,  0.4108083 ])

>>> C = np.corrcoef(X,Y)

>>> C

array([[ 1.        , -0.13272714],

       [-0.13272714,  1.        ]])

>>> df=pd.DataFrame(data=np.random.randint(0,100,size=(50,4)),columns=['A','B','C','D'])

>>> corr=df.corr()

>>> corr

          A         B         C         D

A  1.000000 -0.105610 -0.059742  0.047276

B -0.105610  1.000000  0.046084  0.013553
```

C -0.059742 0.046084 1.000000 0.044677

D 0.047276 0.013553 0.044677 1.000000

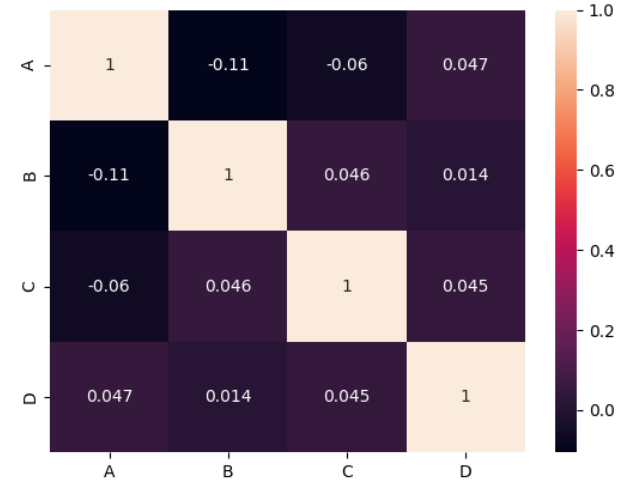
>>> sns.heatmap(corr)



>>> sns.heatmap(corr,annot=True)

<AxesSubplot:>

>>> plt.show()



>>> df=pd.read_csv(r"C:\Users\SIREESHA\Desktop\data.csv")

```
>>> df.head()
         id diagnosis  ...  fractal_dimension_worst  Unnamed: 32
0    842302         M  ...                  0.11890          NaN
1    842517         M  ...                  0.08902          NaN
2  84300903         M  ...                  0.08758          NaN
3  84348301         M  ...                  0.17300          NaN
4  84358402         M  ...                  0.07678          NaN

[5 rows x 33 columns]
>>> df.columns
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32'],
      dtype='object')
>>> df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                  Non-Null Count  Dtype
```

0	id	569 non-null	int64
1	diagnosis	569 non-null	object
2	radius_mean	569 non-null	float64
3	texture_mean	569 non-null	float64
4	perimeter_mean	569 non-null	float64
5	area_mean	569 non-null	float64
6	smoothness_mean	569 non-null	float64
7	compactness_mean	569 non-null	float64
8	concavity_mean	569 non-null	float64
9	concave points_mean	569 non-null	float64
10	symmetry_mean	569 non-null	float64
11	fractal_dimension_mean	569 non-null	float64
12	radius_se	569 non-null	float64
13	texture_se	569 non-null	float64
14	perimeter_se	569 non-null	float64
15	area_se	569 non-null	float64
16	smoothness_se	569 non-null	float64
17	compactness_se	569 non-null	float64
18	concavity_se	569 non-null	float64
19	concave points_se	569 non-null	float64
20	symmetry_se	569 non-null	float64
21	fractal_dimension_se	569 non-null	float64
22	radius_worst	569 non-null	float64
23	texture_worst	569 non-null	float64

```
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
>>> data=df.drop(['Unnamed: 32'], axis = 1)
>>> data.describe()
                id  radius_mean  ...  symmetry_worst  fractal_dimension_worst
count  5.690000e+02   569.000000  ...      569.000000               569.000000
mean   3.037183e+07    14.127292  ...        0.290076                 0.083946
std    1.250206e+08     3.524049  ...        0.061867                 0.018061
min    8.670000e+03     6.981000  ...        0.156500                 0.055040
25%    8.692180e+05    11.700000  ...        0.250400                 0.071460
50%    9.060240e+05    13.370000  ...        0.282200                 0.080040
75%    8.813129e+06    15.780000  ...        0.317900                 0.092080
max    9.113205e+08    28.110000  ...        0.663800                 0.207500

[8 rows x 31 columns]
>>> data.corr()
                          id  ...  fractal_dimension_worst
```

id	1.000000	...	-0.029866
radius_mean	0.074626	...	0.007066
texture_mean	0.099770	...	0.119205
perimeter_mean	0.073159	...	0.051019
area_mean	0.096893	...	0.003738
smoothness_mean	-0.012968	...	0.499316
compactness_mean	0.000096	...	0.687382
concavity_mean	0.050080	...	0.514930
concave points_mean	0.044158	...	0.368661
symmetry_mean	-0.022114	...	0.438413
fractal_dimension_mean	-0.052511	...	0.767297
radius_se	0.143048	...	0.049559
texture_se	-0.007526	...	-0.045655
perimeter_se	0.137331	...	0.085433
area_se	0.177742	...	0.017539
smoothness_se	0.096781	...	0.101480
compactness_se	0.033961	...	0.590973
concavity_se	0.055239	...	0.439329
concave points_se	0.078768	...	0.310655
symmetry_se	-0.017306	...	0.078079
fractal_dimension_se	0.025725	...	0.591328
radius_worst	0.082405	...	0.093492
texture_worst	0.064720	...	0.219122
perimeter_worst	0.079986	...	0.138957
area_worst	0.107187	...	0.079647

```
smoothness_worst         0.010338  ...                0.617624
compactness_worst       -0.002968  ...                0.810455
concavity_worst          0.023203  ...                0.686511
concave points_worst     0.035174  ...                0.511114
symmetry_worst          -0.044224  ...                0.537848
fractal_dimension_worst -0.029866  ...                1.000000

>>> import seaborn as sns
>>> sns.heatmap(data.corr())
<AxesSubplot:>
>>> plt.show()
```