Data Analysis with Python

1. Concept Overview -Python

1.1. Variable:

Container to store values.

Code:

a=5

1.2. Print:

It is a function which used to display

Code:

```
a=5
print("Iam",a,"years old")
```

Output:

```
I am 5 years old
```

1.3. Operators:

operators are special symbols combination of symbols or key words that designates some type of computation.

Arithmetic operators:

An Arithmetic operator is a special type of mathematical function that performs a calculation on two operands.The different type of arithmetic operators include addition,subtraction,multiplication,division,modulus,exponentiation and floor division.

```
Addition,Subtraction,Multiplication,Division
```

Code:

```
a=5
b=25
print(a+b)
```

```
print(a-b)

print(a*b)

print(a/b)
```

Output:

30

-20

125

0.2

Power or Exponent

Code:

```
print(2**4)
```

Output:

16

Floor Division

Code:

```
print(2//5)
```

Output:

12

Relation Operator: A relation operator in python is a symbol or keyword used to compare two values and determine the relation between them .The different types of relational operators include are <(less than),>(greater than),<=(less than or equal to),>=(greater than or equal to),!=(not equal to)

Code:

```
a=6

b=6

print(a==b)
```

```
print(a>b)

print(a>b)

print(a<=b)

print(a>=b)

print(a!=b)
```

Output:

```
True

False

False

True

True

False
```

Logical Operator: Logical operators are used to combine multiple conditions together less than as a single boolean expression.There are three types of logical operators in python ‘and’,’or’,’not’.

Code:

```
a=6

b=4

print((a>b) and (a<b)) #true false
```

Output:

```
False
```

Code:

```
a=6

b=25

print((a<b) or (a<b))#true false
```

Output:

```
True
```

Membership operators: Checks whether the given value is a member of sequence such as strings,lists and tuples

Two primary types:'in'(return True if specified value is found within the sequence)and 'not in'(returns True if specified value is not in the given sequence).

Code:

```
a="lavanya"

print("l" in a)

print("r" in a)
```

Output:

```
True

False
```

1.4. **Control flow - Conditional Statements:**

Conditional if:

An if statement is a conditional statement used to check the condition,and executes if the condition is true.It is also a control flow statement which utilises decision-making to control the flow of execution.

```
write a program to get a number from the user and    check
weather it is positive or not
```

Code:

```
a=int(input("enter a number:"))

print(a)

if a>0:
```

```
print("positive")
```

Output:

```
enter a number:8

8

positive
```

Conditional -if-else:

The if-else statement used to execute both the true and the false part of a given condition.

Code:

```
a=int(input("enter a number:"))
print(a)
if a>0:
print("positive")
else:
print("negative")
```

Output:

```
enter a number:-2

-2

Negative
```

Conditional if-else-ladder: Common programming construct that is based upon nested if is the if-else-if ladder.

```
write a program to get a number from the user and check weather it is positive or not
```

Code:

```
a=int(input("enter a number:"))
```

```
print(a)

if a>0:

print("positive")

elif (a==0):

print("Neutral")

else:

print("negative")
```

Output:

```
enter a number:0

0

Neutral
```

1.5. **Control flow - Looping Statements:**

Table by using - For loop: A control flow statement that is used to repeatedly execute the group of statements.

Code:

```
n=int(input("Enter n value"))

for i in range(1,11):

rel=n*i

print(rel)
```

Output:

```
Enter n value8

8

16

24

32
```

```
40

48

56

64

72

80
```

Table using While loop

Code:

```
n=int(input("Enter a value:"))

i=1

while(i<=10):

a=n*i

print(a)

i=i+1
```

Output:

```
Enter a value:2

2

4

6

8

10

12

14

16

18

20
```

1.6. Data Slicing:

The slice() method extracts a section of data and returns it as a new data,without modifying it.This means users can take a specific range of elements without changing it.

Syntax: slice(start,stop,step)

Code:

```
a="Python is easy"

print(a[0:6]) #start:stop:step
```

Output:

```
Python
```

Reverse:

Code:

```
a="python is easy"

print(a[::-1])#start:stop:step
```

Output:

```
ysae si nohtyp
```

Slicing by step-2

Code:

```
a="python is easy"

print(a[::2])#start:stop:step
```

Output:

```
pto ses
```

1.7. Type Casting

It is a process that converts a variable's data type into another data type.

Implicit type casting: By interpreter

Code:

```
a=25.5 #float
b=6 #int
print(a*b)#interpreter automatically changes the data type
```

Output:

```
153.0
```

Explicit type casting: By user

Code:

```
a=25.5 #float
b=6 #int
print(int(a*b))#user
#explicit conversion
```

Output:

```
153
```

1.8. Collection List:

LIST:

List is a collection of elements

- **List is Heterogenous**
- **mutable**

Code:

```
#list collections
l1=[56,"lav",54.0,67]
for i in l1:
print(i)
```

Output:

```
56

lav

54.0

67
```

List append:

Code:

```
l3=[1,2,3,]

l4=[4,5,6]

l3.append(l4)

print(l3)
```

Output:

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

Insert at specific index:

Code:

```
l1=[1,2,3,]

l2="hiiii"

l1.insert(2,'hiii')#syntax is index number ,element

print(l1)
```

Output:

```
[1, 2, 'hiii', 3]
```

List adding:

Code:

```
print(l3+l4)
```

Output:

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

List Extend:

Code:

```
l3=[1,2,3,]

l4=[4,5,6]

l3.extend(l4)

print(l3)
```

Output:

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

List pop:

Code:

```
l1=[1,2,3,4,5,6,7,8]

l1.pop(0)#syntax pop(index number)

print(l1)
```

Output:

```
[2, 3, 4, 5, 6, 7, 8]
```

List max:

Code:

```
#max in list

l5=[12.8,34.6,98.2,9.4]

print(max(l5))
```

Output:

```
98.2
```

```
#min in lists
```

```
print(min(l5))
```

Output:

```
9.4
```

1.9.List Comprehensions:

- Iterates
- applies some function on every element
- conditions
- output:list

Code:

```
l1=[2,3,4]
l2=[i**2 for i in l1]#output iterator condition
print(l2)
```

Output:

```
[4, 9, 16]
```

Code:

```
l1=[2,3,4]
l2=[i**2 for i in l1 if i>3]#output iterator condition
Print(l2)
```

Output:

```
[16]
```

```
#type of list
type(l2)
#The salaries of 5 employees in a company is taken as a
list.the tax is 10% if the salary is less than or equal to
50000
```

```
#or it is 15%
#create a new list with tax amount
#[67000,45000,89000,34000,50000]
#list_name=[(body of if) if (condition) else (body of else
iterator]
```

Code:

```
sal =[67000,45000,89000,34000,50000]

tax =[]

for i in sal:

if i<=50000:

t=i*0.1

tax.append(t)

else :

t=i*0.15

tax.append(t)

print(tax)
```

Output:

```
[10050.0, 4500.0, 13350.0, 3400.0, 5000.0]
```

Code:

```
#using list

Sal=
[67000,45000,89000,34000,50000]#syntax---->list_name=[(body
of if) (condition) else (body of else) iterate]

tax=[i*0.1 if i<=50000 else i*0.15 for i in sal]

print(tax)
```

Output:

```
[10050.0, 4500.0, 13350.0, 3400.0, 5000.0]
```

2.0 Library Function :

2.1 Numpy:

Numpy is a python library used for working with arrays.

- Numpy stands for Numerical Python.
- It also has functions for the working domain of linear algebra,fourier transform and matrices.

Importing

```
import numpy as np
```

2.2 Arrays :

Creating 1D array #rows and columns

Code:

```
import numpy as np

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

print(type(a))
```

Output:

```
<class 'numpy.ndarray'>
```

Creating 2D array #rows and columns

Code:

```
#creating 2D array-rows and columns

import numpy as np

b=np.array([[2,3,4],[7,8,9]])

print(b)
```

Output:

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

```
[7 8 9]]

#creating a 3D array--rows,columns,groups

import numpy as np

c=np.array([[[1,2,3],[4,5,6]],[[6,9,0],[2,3,11]]])

print(c)
```

Output:

```
[[[ 1  2  3]

 [ 4  5  6]]



 [[ 6  9  0]

 [ 2  3 11]]
```

```
 #checking dimensions

 print(a.ndim)

 print(b.ndim)

 print(c.ndim)
```

Output:

```
 2

  2

  3
```

Code:

```
#ones
import numpy as np
d=np.ones((2,5))#2D#rows,columns
print(d)
```

Output:

```
[[1. 1. 1. 1. 1.]
```

```
 [1. 1. 1. 1. 1.]]
```

Code:

```
#ones
import numpy as np
e=np.ones((2,3,2))#3D groups ,rows ,columns
print(e)
```

Output:

```
[[[1. 1.]
 [1. 1.]
 [1. 1.]]


 [[1.1.]
  [1. 1.]
  [1. 1.]]]
```

```
#zeros
import numpy as np
f=np.zeros((3,2))#2D#rows ,columns
print(f)
```

Output:

```
[[0. 0.]
 [0. 0.]
 [0. 0.]]
```

Code:

```
#zeros
import numpy as np
```

```
g=np.zeros((2,3,2))#groups ,rows,columns #3D
print(g)
```

Output:

```
[[[0. 0.]
  [0. 0.]
  [0. 0.]]


 [[0. 0.]
  [0. 0.]
  [0. 0.]]]
```

Code:

```
#Eye
import numpy as np
h=np.eye(2)
print(h)
```

Output:

```
[[1. 0.]
 [0. 1.]]
```

2.3 Arange:

Arange: The arange function in python is used to create a sequence of

Numbers with a specified start,stop and step value

Code:

```
#arrange
```

```
import numpy as np

i=np.arange(3,31,3)##start,stop,step

print(i)
```

Output:

```
[ 3  6  9 12 15 18 21 24 27 30]
```

Code:

```
#5 table

import numpy as np

k=np.arange(5,1001,5)

print(k)
```

Output:

```
    [   5   10   15   20   25   30   35   40   45   50   55
60   65   70

  75   80   85   90   95  100  105  110  115  120  125  130  135  140

 145  150  155  160  165  170  175  180  185  190  195  200  205  210

 215  220  225  230  235  240  245  250  255  260  265  270  275  280

 285  290  295  300  305  310  315  320  325  330  335  340  345  350

 355  360  365  370  375  380  385  390  395  400  405  410  415  420

 425  430  435  440  445  450  455  460  465  470  475  480  485  490

 495  500  505  510  515  520  525  530  535  540  545  550  555  560

 565  570  575  580  585  590  595  600  605  610  615  620  625  630

 635  640  645  650  655  660  665  670  675  680  685  690  695  700

 705  710  715  720  725  730  735  740  745  750  755  760  765  770

 775  780  785  790  795  800  805  810  815  820  825  830  835  840

 845  850  855  860  865  870  875  880  885  890  895  900  905  910
```

```
 915  920  925  930  935  940  945  950  955  960  965  970  975  980

 985  990  995 1000]
```

Reshape: This function is used to reshape an array into a given shape without changing data.

Code:

```
L=np.arange(1,7).reshape(2,3)
print(L)
R=np.arange(9,15).reshape(2,3)
print(R)
```

Output:

```
 [[1 2 3]
 [4 5 6]]
[[ 9 10 11]
 [12 13 14]]
#sum of arrays
print(L+R)
```

Output:

```
[[10 12 14]
 [16 18 20]]
#sum function
G=np.sum((L,R))
print(G)
```

Output:

```
90
#sum function using axis=0
G=np.sum((L,R),axis=0)
print(G)
```

Output:

```
[[10 12 14]
 [16 18 20]]
```

```
#sum function using axis=1
G=np.sum((L,R),axis=1)
print(G)
```

Output:

```
[[ 5  7  9]
 [21 23 25]]
#linspace
M=np.linspace(1,2,6)
print(M)
```

Output:

```
[1.  1.2 1.4 1.6 1.8 2. ]
```

Code:

```
t=np.ones((4,2))

u=np.ones((4,2))

print(np.sum((t,u),axis=0))
```

Output:

```
[[2. 2.]

 [2. 2.]

 [2. 2.]

 [2. 2.]]
```

Code:

```
A=np.array([[1,1],[0,1]])
B=np.array([[2,0],[3,4]])
print(A*B)
```

Output:

```
[[2 0]
 [0 4]]
```

Code:

```
print(A@B)
```

Output:

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

#problem:

```
#b=np.array[25,289,361,81] find square root and iterate
through result values output 5 square is 25 by using np.sqrt
b=np.array( [25,289,361,81])
for i in b:
print(int(np.sqrt(i)),"square is",i)
```

Output:

```
5 square is 25
17 square is 289
19 square is 361
9 square is 8
```

2.4Array joining

```
a=np.array([34,35,36,37,38,39])
a.resize(2,3)
b=np.array([4,5,6,7,8,9])
b.resize(2,3)
print(np.vstack((a,b)))
print("\n")
print(np.hstack((a,b)))
```

Output:

```
[[34 35 36]
 [37 38 39]
 [ 4  5  6]
 [ 7  8  9]]


[[34 35 36  4  5  6]
 [37 38 39  7  8  9]]
```

Dstack:it is used to stack arrays in sequence depth wise.

```
a=np.arange(30).reshape(2,3,5)
print(a)
print("output of dstack")
print(np.dstack(a))
#no.of rows becomes no.of groups
#columns becomes rows
```

```
#group becomes columns
```

Output:

```
[[[ 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]]]
output of dstack
[[[ 0 15]
  [ 1 16]
  [ 2 17]
  [ 3 18]
  [ 4 19]]

 [[ 5 20]
  [ 6 21]
  [ 7 22]
  [ 8 23]
  [ 9 24]]

 [[10 25]
  [11 26]
  [12 27]
  [13 28]
  [14 29]]]
```

#Random module is subpackage of numpy

```
Example:
a=np.random.rand(1)
print(a)
```

Output:

[0.66463289]

Example:

```
a=np.random.rand(8,4) #rand-range is between 0 and 1
print(a)
```

Output:

```
[[0.86793957 0.20550871 0.56757905 0.85394866]
 [0.63239592 0.14391372 0.389903   0.34102883]
 [0.25517135 0.36435822 0.07377094 0.02046665]
 [0.24820571 0.67029435 0.5019085  0.31999846]
 [0.45923602 0.92441907 0.1682318  0.37349023]
 [0.77982353 0.11648227 0.9405154  0.18739525]
 [0.82948274 0.7246973  0.55465982 0.3635287 ]
 [0.98206701 0.74445092 0.43170092 0.70714976]]
```

Example:

```
a=10*np.random.rand(8,4)
print(a)
```

Output:

```
[[5.44601317 5.98726441 7.70631605 7.7068417 ]
 [1.67336756 6.97480975 1.03651707 3.82562451]
 [9.89885977 1.23724356 5.43990632 0.21200397]
 [5.88096494 4.2766098  2.78442444 8.19555105]
 [9.99290092 9.3625434  2.55097112 8.71467748]
 [9.88690419 9.827445   1.21206775 6.63063275]
 [7.60806905 5.18914903 3.45710376 6.21997742]
 [9.74972197 4.32029902 2.53673393 8.41650447]]
```

```
#floor function
a=np.floor(10*np.random.rand(8,4))
print(a)
```

Output:

```
a=np.floor(10*np.random.rand(8,4))

print(a)
```

output

```
[[0. 6. 4. 2.]
 [5. 2. 7. 2.]
 [5. 5. 6. 7.]
 [8. 3. 8. 6.]
 [9. 0. 0. 4.]
 [1. 0. 8. 0.]
 [7. 9. 5. 2.]
 [8. 5. 1. 5.]]
```

Example:

```
a=np.arange(1,33).reshape(8,4)
print(a)
```

Output:

```
[[ 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]]
```

2.5 Vsplit:

Vsplit function is used to split the array into two subarrays along the vertical axis(rows).

Example-1:

```
np.vsplit(a,4) #here 4 is the no of splits we require
```

Output:

```
[array([[1, 2, 3, 4],
        [5, 6, 7, 8]]),
 array([[ 9, 10, 11, 12],
        [13, 14, 15, 16]]),
 array([[17, 18, 19, 20],
        [21, 22, 23, 24]]),
 array([[25, 26, 27, 28],
        [29, 30, 31, 32]])]
```

Example-2:

```
print(np.vsplit(C,(2,5)))
```

Output:

```
[array([[5., 4., 5., 1.],
       [9., 8., 5., 5.]]), array([[5., 0., 1., 5.],
       [4., 9., 1., 1.],
       [0., 9., 0., 4.]]), array([[6., 4., 6., 6.],
       [3., 1., 4., 0.],
       [6., 7., 0., 8.]])]
```

Example-3:

```
np.vsplit(a,(3,5)) #split it after 3rd row and 5th row
```

Output:

```
[array([[ 1,  2,  3,  4],
        [ 5,  6,  7,  8],
        [ 9, 10, 11, 12]]),
 array([[13, 14, 15, 16],
```

```
        [17, 18, 19, 20]]),
 array([[21, 22, 23, 24],
        [25, 26, 27, 28],
        [29, 30, 31, 32]])]
```

2.6 hsplit:

hsplit function is used to split the array into two subarrays along the horizontal axis(columns).

Example-1:

```
a=np.arange(1,33).reshape(4,8)
print(a)
```

Output:

```
[[ 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]]
```

Example-2:

```
np.hsplit(a,4) #splitting takes place from left to right
#split divide the array into 4 parts
```

Output:

```
[array([[ 1,  2],
        [ 9, 10],
        [17, 18],
        [25, 26]]),
 array([[ 3,  4],
        [11, 12],
        [19, 20],
        [27, 28]]),
 array([[ 5,  6],
        [13, 14],
        [21, 22],
        [29, 30]]),
 array([[ 7,  8],
        [15, 16],
        [23, 24],
        [31, 32]])]
```

Example-3:

```
np.hsplit(a,(3,7)) #splitting takes place after 3rd column
and 7th column
```

Output:

```
[array([[ 1,  2,  3],
    [ 9, 10, 11],
```

```
      [17, 18, 19],
      [25, 26, 27]]),
 array([[ 4,  5,  6,  7],
      [12, 13, 14, 15],
      [20, 21, 22, 23],
      [28, 29, 30, 31]]),
 array([[ 8],
      [16],
      [24],
      [32]])]
```

Example-4:

```
np.hsplit(a,(2,5)) #split takes place after 2nd column and
5th column
```

Output:

```
[array([[ 1,  2],
        [ 9, 10],
        [17, 18],
               [25, 26]]),
 array([[ 3,  4,  5],
        [11, 12, 13],
        [19, 20, 21],
        [27, 28, 29]]),
 array([[ 6,  7,  8],
        [14, 15, 16],
        [22, 23, 24],
        [30, 31, 32]])]
```

2.7 Trigonometry:

Example-1:

```
np.pi
```

Output:

3.141592653589793

Example-2:

```
A=[np.pi/4,np.pi/3,np.pi/2,np.pi]
print(A)
```

Output:

```
[0.7853981633974483, 1.0471975511965976, 1.5707963267948966,
3.141592653589793]
```

Example-3:

```
B = np.rad2deg(A)
print(B)  #convert radians to degrees
```

Output:

```
[ 45.  60.  90. 180.]
```

Example-4:

```
np.deg2rad(B) #convert degree to radians
```

Output:

```
array([0.78539816, 1.04719755, 1.57079633, 3.14159265])
```

Example-5:

```
np.sin(1) #radians
```

Output:

```
0.8414709848078965
```

Example-6:

```
np.cos(1)
```

Output:

```
0.5403023058681398
```

Example-7:

```
np.tan(1)
```

Output:

```
1.5574077246549023
```

Statistics

Example-1:

```
ar=np.array([23,45,67,89,21,34])
np.mean(ar)
```

Output:

46.5

Example-2:

```
ar=np.array([23,45,67,89,21,34])
np.median(ar)
```

Output:

39.5

Example-3:

```
ar=np.array([23,45,67,89,21,34])
np.std(ar)
```

Output:

```
24.452334585202017
```

Example-4:

```
ar=np.array([23,45,67,89,21,34])
np.var(ar)
```

Output:

```
597.9166666666666
```

Inverse matrix:

```
C=np.arange(1,5).reshape(2,2)
print(C)
```
Output:
```
[[1 2]
 [3 4]]
```
#logic for inverse matrix
```
np.linalg.inv(C)
```
Output:
```
array([[-2. ,  1. ],
       [ 1.5, -0.5]])
```
Max function(): It is used to find the maximum element in the matrix.

Example-1:
```
C=np.floor(10*np.random.rand(24)).reshape(6,4)
print(C)
```
Output:
```
[[9. 5. 4. 6.]
 [2. 4. 1. 5.]
 [1. 4. 5. 5.]
 [3. 1. 9. 7.]
 [7. 3. 6. 9.]
 [0. 1. 5. 9.]]
```
Example for find max element
```
print(np.argmax(C))
```
Output:
```
0
```

Example-2:
```
c=10*np.random.rand(24).reshape(6,4)
print(c)
```
Output:
```
[[9.15519817 8.39266107 0.76805167 8.43663521]
 [9.28316067 5.63640124 5.04580318 9.51054732]
 [0.71929046 4.36124117 7.86106573 6.51879388]
 [1.16087787 9.53980816 8.89155739 0.36573456]
 [4.67095163 8.77191641 6.75421363 9.81088343]
 [4.69871256 1.8691479  0.13198657 0.63774885]]
```
Example for find max element
```
print(np.argmax(c))
```
Output:

19

Example for finding max element in rows:
```
print(np.argmax(c,axis=1))
```

Output:

```
[0 3 2 1 3 0]
```

Example for finding max element in columns:

```
print(np.argmax(c,axis=0))
```

Output:

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

Example for find the minimum element:

```
print(np.argmin(c))
```

Output:

```
22
```

Example for finding min element in columns:

```
print(np.argmin(c,axis=0))
```

Output:

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

Example for finding min element in rows:

```
print(np.argmin(c,axis=1))
```

Output:

```
[2 2 0 3 0 2]
```

Search

Example-1:

```
a=np.array([34,56,7,17,88,91])
print(np.where(a%2==0))
print(np.where(a%2==1))
```

Output:

(array([0, 1, 4]),)

(array([2, 3, 5]),)

Example-2:

```
a=np.array([24,16,7,17,54,60])
print(np.where(a%6==0))
```

Output:

```
(array([0, 4, 5]),)
```

Example-3:

```
a=np.array([2,4,7,3,6]) #searchsort
x=np.searchsorted(a,3)
print(x)
```

Output:

1

Example-4:

```
a=np.array([2,4,6,7,8])
x=np.searchsorted(a,7)
print(x)
```

Output:

```
3
```

Example-5:

```
a=np.array(['banana','apple','cherry']) #sorting
print(np.sort(a))
```

Output:

```
['apple' 'banana' 'cherry']
```

Example-6:

```
a=np.array(['true','false','true'])
print(np.sort(a))
```

Output:

```
['false' 'true' 'true']
```

Example-7:

```
b=np.array([[3,2,4],[5,0,1]])
print(np.sort(b))
```

Output:

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

2.8 Filtering:

filter() function is to process an iterable and extract those items that satisfy a given condition.

Example-1:

```
b=np.array([40,43,50,44,67,78])
flit=np.where(b%2==0)            #flit may be list or array
print(flit)
```

Output:

```
(array([0, 2, 3, 5]),)
b[flit]
Output:
array([40, 50, 44, 78])
```

Iterating through two arrays:

Example-1:

```
names=np.array(["valli","sudha","susmitha","priya"])
initials=np.array(["p","m","u","s"])
for i,j in zip(initials,names):
  print(i,".",j)
```

Output:

```
p . valli
m . sudha
u . susmitha
s . priya
```

Example-2:

```
a=np.array([10,20,30,40,50,60])
b=np.array([20,21,22,23,24,25])
n=np.multiply(a,b)      #multiplication
print(n)
```

Output:

```
[ 200  420  660  920 1200 1500]
```

Example for divide two arrays:

```
a=np.array([10,20,30,40,50,60])
b=np.array([20,21,22,23,24,25])
print(np.divide(a,b))
```

Output:

```
[0.5        0.95238095 1.36363636 1.73913043 2.08333333 2.4
]
```

Example for modulo division on two arrays:

```
a=np.array([10,20,30,40,50,60])
b=np.array([20,21,22,23,24,25])
print(np.mod(a,b))
```

Output:

```
[10 20  8 17  2 10]
```

Example:

```
a=np.array([10,20,30,40,50,60])
b=np.array([20,21,22,23,24,25])
print(np.divmod(a,b))
```

Output:

```
(array([0, 0, 1, 1, 2, 2]), array([10, 20,  8, 17,  2, 10]))
```

2.9 Logarithms

Example-1:

```
a1=np.arange(1,10)
```

```
print(a1)
print(np.log2(a1))
```

Output:

```
[1 2 3 4 5 6 7 8 9]
[0.         1.         1.5849625  2.         2.32192809
2.5849625
 2.80735492 3.         3.169925  ]
```

Example-2:

```
a=1.2
print(np.log(a)) #natural log e
```

Output:

```
0.1823215567939546
```

Example-3:

```
a=1.2
print(np.log2(a)) #log base 2
```

Output:

```
0.2630344058337938
```

Example-4:

```
a=1.2
print(np.log10(a)) #log base 10
```

Output:

```
0.07918124604762482
```

Example-5:

```
a=np.array([1,1.2,3,4])
print(np.log(a)) #natural log e
```

Output:

```
[0.         0.18232156 1.09861229 1.38629436]
```

Other mathematical functions

Example-1:

```
c=np.array([5,6,3,8,])
print(np.cumprod(c))
```

Output:

```
[  5  30  90 720]
```

Example-2:

```
c=np.array([5,6,3,8])
print(np.sum(c))
```

Output:

```
22
```

Example-3:

```
a=np.array([10,15,25,15])
```

```
n=np.diff(a)
print(n)
```

Output:

```
[  5  10 -10]
```

Example-4:

```
a=455
b=665
print(np.lcm(a,b))
```

Output:

8645

Example-5:

```
a=455
b=665
print(np.gcd(a,b))
```

Output:

35

Example-6:

```
#FINDING GCD
m1=455
m2=665
x=np.gcd(m1,m2)
print(x)
```

Output:

35

Example-7:

```
a=np.array([12,15,60])
gc=np.gcd.reduce(a) #takes multiple inputs and gives single
output
print(gc)
```

Output:

3

Example-8:

```
a=np.array([12,15,60])
lc=np.lcm.reduce(a)
print(lc)
```

Output:

60

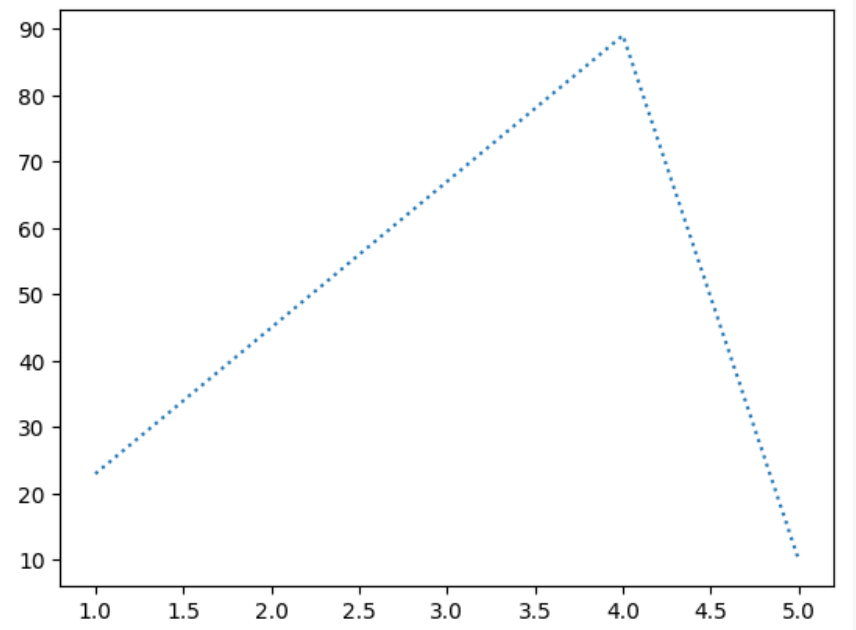
2.10 Matplotlib:

Matplotlib is a cross_platform,data visualisation and graphical plotting library for python and its numerical extension numpy.

Example-1:

```
a=[23,45,67,89,10]  #corona cases in first five days
b=[1,2,3,4,5]
plt.plot(b,a,linestyle=":") #plot (x,y) #
plt.show()
```

Output:

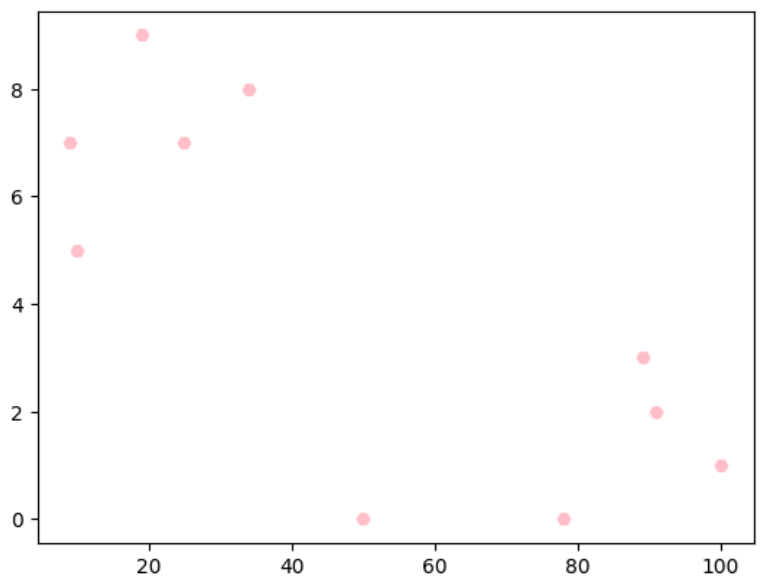


Example-2:

Runs scored by 10 players [100,50,91,78,89,25,34,19,9,10] wickets taken by the same 10 new players [1,0,2,0,3,7,8,9,7,5] from the clusters.

```
a=[100,50,91,78,89,25,34,19,9,10]
b=[1,0,2,0,3,7,8,9,7,5]
plt.scatter(a,b,color="pink",marker="H")
plt.show()
```

Output:

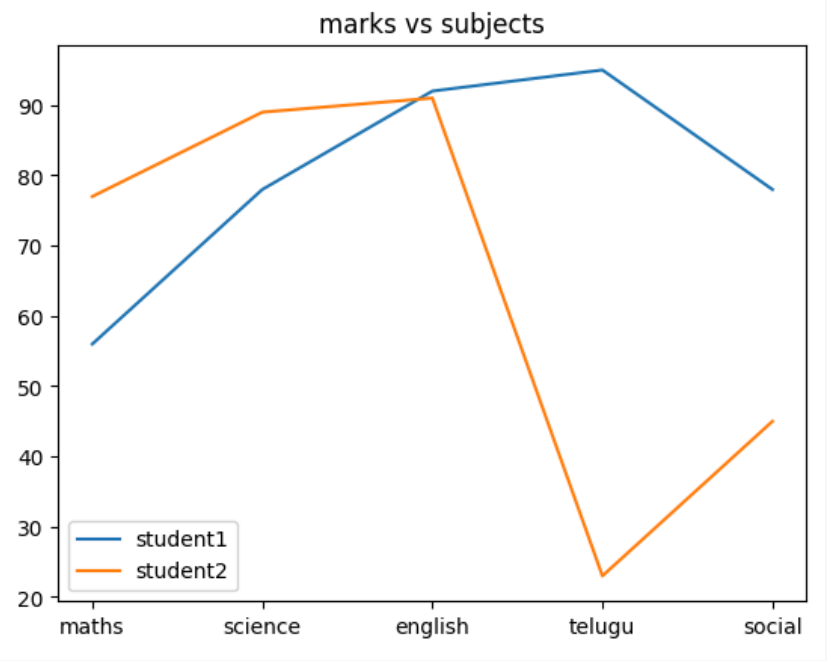


Example-3:

plot the scores of 2 students in 5 different subjects. subjects as x axis and marks as y axis.

```
stu1=[56,78,92,95,78]
stu2=[77,89,91,23,45]
sub=["maths","science","english","telugu","social"]
plt.plot(sub,stu1,label="student1")
plt.plot(sub,stu2,label="student2")
plt.title("marks vs subjects")
plt.legend()
```

Output:

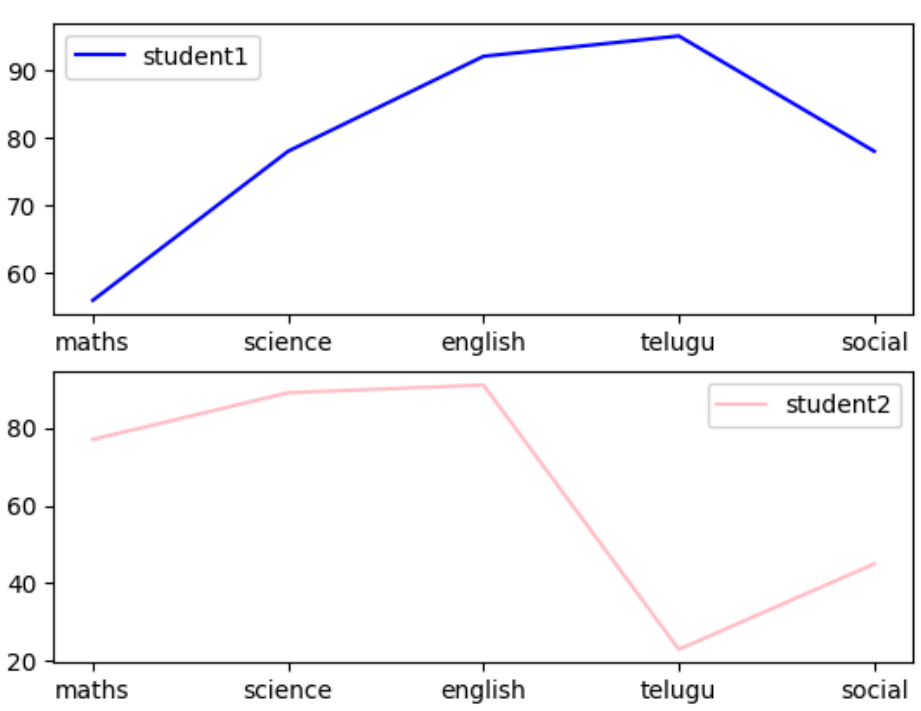


subplot (rows,columns,position)

Example:

```
stu1=[56,78,92,95,78]
stu2=[77,89,91,23,45]
sub=["maths","science","english","telugu","social"]
plt.subplot(2,1,1)
plt.plot(sub,stu1,label="student1",color="b")
plt.legend()
plt.subplot(2,1,2)
plt.plot(sub,stu2,label="student2",color="pink")
plt.legend()
```

Output:



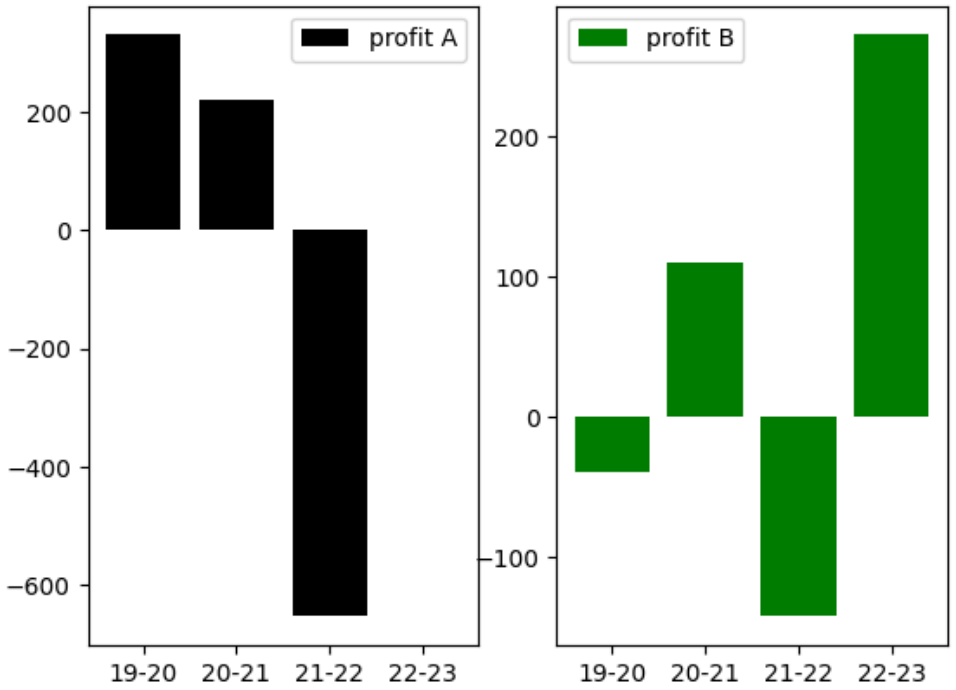
Example-2:

```
A=[230,560,780,127,128]
B=[200,160,270,127,400]
print("revenue of A",A)
print("revenue of B",B)
profA=np.diff(A)
print("profits of A",profA)
profB=np.diff(B)
print("profits of B",profB)
years=["19-20","20-21","21-22","22-23"]
plt.subplot(1,2,1)
plt.bar(years,profA,color="k",label="profit A")
plt.legend(loc = "best")
plt.subplot(1,2,2)
plt.bar(years,profB,color="g",label="profit B")
plt.legend(loc = "best")
```

Output:

```
revenue of A [230, 560, 780, 127, 128]

revenue of B [200, 160, 270, 127, 400]

profits of A [ 330  220 -653    1]

profits of B [ -40  110 -143  273]
```

<matplotlib.legend.Legend at 0x78b0fc5d6cb0>

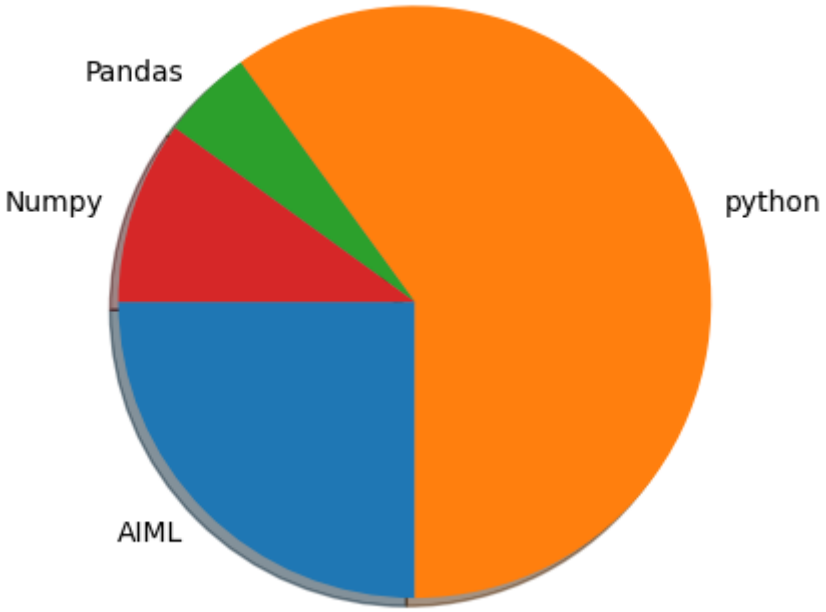


Pie Chart:

```
a=np.array([25,60,5,10])

v=["aiml","python","pandas","numpy"]

plt.pie(a,labels=v)

plt.show()
```

Output:

Output:

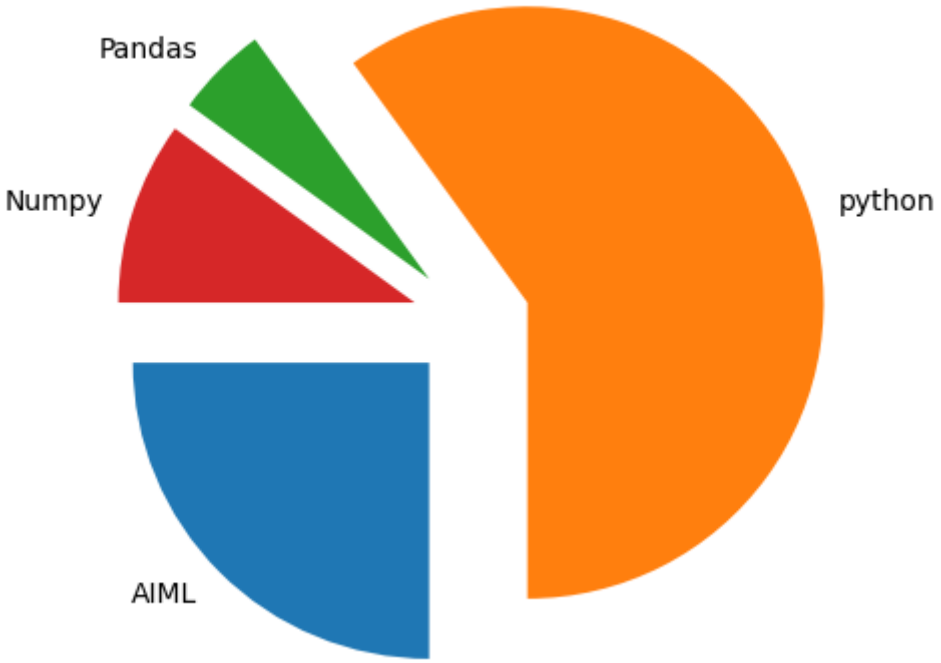


Code:

```
#exploding
#pie chart
a=np.array([25,60,5,10])
labe=["AIML","python","Pandas","Numpy"]
explo=[0.2,0.2,0.2,0.2]
```

```
plt.pie(a,labels=labe,explode=explo,startangle=180)

plt.show()
```

Output:



Pandas:

- Used for data manipulation
- Create dataframes from excel,csv,txt,DBs
- Dataframes(rows and columns readable by python)
- Data cleaning by dropping or replace with mean
- visualise the data

3.1 Importing:

```
import pandas as pd
```

Code:

```
import pandas as pd
a=["lavanya","prasanna","akhila","sasi"]
r=pd.Series(a,index=[67,43,44,89])#pd.Series()
print(r)
```

Output:

```
67     lavanya
43    prasanna
44      akhila
89        sasi
dtype: object
```

Importing files

- For csv and txt :read_csv
- For excel : read_excel

Code:

```
#Importing csv files
df=pd.read_csv("/content/diabetcsv.csv")
df.head(10)
```

Output:

	preg	plas	pres	skin	insu	mass	pedi	age	class
0	6	148	72	35	0	33.6	0.627	50	tested_positive
1	1	85	66	29	0	26.6	0.351	31	tested_negative
2	8	183	64	0	0	23.3	0.672	32	tested_positive
3	1	89	66	23	94	28.1	0.167	21	tested_negative
4	0	137	40	35	168	43.1	2.288	33	tested_positive
5	5	116	74	0	0	25.6	0.201	30	tested_negative
6	3	78	50	32	88	31.0	0.248	26	tested_positive
7	10	115	0	0	0	35.3	0.134	29	tested_negative
8	2	197	70	45	543	30.5	0.158	53	tested_positive
9	8	125	96	0	0	0.0	0.232	54	tested_positive

```
#Importing text files
dft=pd.read_csv("/content/grades.txt",sep=" ")
dft.head(10)
```

Output:

	Names	Initials	SEM1	SEM2	SEM3	Grade
0	Joe	K	9.8	10.0	9.9	A+
1	Rajesh	M	8.9	9.1	9.3	A
2	Kissan	V	9.9	9.3	9.2	A
3	Mary	N	7.7	8.0	7.1	B
4	Jeen	K	9.8	9.1	9.9	A+
5	Raj	M	8.9	9.1	9.3	A
6	Hassan	V	9.9	9.0	9.2	A
7	Mari	N	7.7	8.0	7.1	B
8	Jess	K	9.8	9.1	9.9	A+
9	Rajini	M	7.0	9.1	9.3	A

```
#Importing excel files
dfe=pd.read_excel("/content/diabetes.xlsx")
dfe.head(10)
```

Output:

	preg	plas	pres	skin	insu	mass	pedi	age	class
0	6	148	72	35	0	33.6	0.627	50	tested_positive
1	1	85	66	29	0	26.6	0.351	31	tested_negative
2	8	183	64	0	0	23.3	0.672	32	tested_positive
3	1	89	66	23	94	28.1	0.167	21	tested_negative
4	0	137	40	35	168	43.1	2.288	33	tested_positive
5	5	116	74	0	0	25.6	0.201	30	tested_negative
6	3	78	50	32	88	31.0	0.248	26	tested_positive
7	10	115	0	0	0	35.3	0.134	29	tested_negative
8	2	197	70	45	543	30.5	0.158	53	tested_positive
9	8	125	96	0	0	0.0	0.232	54	tested_positive

3.2 Describe:

```
print(dft.describe)
```

Output:

```
bound method NDFrame.describe of      Names Initials  SEM1  SEM2
SEM3 Grade
0       Joe        K   9.8  10.0   9.9    A+
1    Rajesh        M   8.9   9.1   9.3     A
2    Kissan        V   9.9   9.3   9.2     A
3      Mary        N   7.7   8.0   7.1     B
4      Jeen        K   9.8   9.1   9.9    A+
5       Raj        M   8.9   9.1   9.3     A
6    Hassan        V   9.9   9.0   9.2     A
7      Mari        N   7.7   8.0   7.1     B
8      Jess        K   9.8   9.1   9.9    A+
9    Rajini        M   7.0   9.1   9.3     A
10    Kiran        V   9.9   9.3   9.2     A
11     Maya        N   7.7   8.0   7.1     B
```

```
12   Jolin        K   9.8   9.1   9.9    A+
13    Riya        M   8.0   9.1   9.3     A
14    Sana        V   9.9   9.3   9.2     A
15    Mark        N   7.7   8.0   7.0     B>
```

```
print(dfe.describe)
```

Output:

```
<bound method NDFrame.describe of      preg  plas  pres  skin
insu  mass   pedi  age            class
0        6   148    72    35      0  33.6  0.627   50
tested_positive
1        1    85    66    29      0  26.6  0.351   31
tested_negative
2        8   183    64     0      0  23.3  0.672   32
tested_positive
3        1    89    66    23     94  28.1  0.167   21
tested_negative
4        0   137    40    35    168  43.1  2.288   33
tested_positive
..     ...   ...   ...   ...    ...   ...    ...   ...
...
763     10   101    76    48    180  32.9  0.171   63
tested_negative
764      2   122    70    27      0  36.8  0.340   27
tested_negative
765      5   121    72    23    112  26.2  0.245   30
tested_negative
766      1   126    60     0      0  30.1  0.349   47
tested_positive
767      1    93    70    31      0  30.4  0.315   23
tested_negative


[768 rows x 9 columns]>
```

Code:

```
#excel files
print(dfe.shape)
print(dfe.shape[0])# get the number of rows only
print(dfe.shape[1])#Get the number of columns only
```

Output:

```
(768, 9)
768
9
```

Code:

```
#text files
dft=pd.read_csv("/content/grades.txt",sep=" ")
print(dft.shape)
print(dft.shape[0])# get the number of rows only
print(dft.shape[1])#Get the number of columns only
```

Output:

```
 (16, 6)
16
6
```

```
#csv files
print(df.shape)
print(df.shape[0])# get the number of rows only
print(df.shape[1])#Get the number of columns only
```

Output:

```
(768, 9)
768
9
```

3.3 Accessing data

- **loc-accepts column names and index**
- **iloc-accepts only index**

```
print(dft[2:5])#to access rows
```

Output:

```
   Names Initials  SEM1  SEM2  SEM3 Grade
2  Kissan        V   9.9   9.3   9.2     A
3    Mary        N   7.7   8.0   7.1     B
4    Jeen        K   9.8   9.1   9.9    A+
```

```
print(dft.iloc[2:5, :3])#iloc[row range,column range]=>index
```

Output:

	Names	Initials	SEM1
2	Kissan	V	9.9
3	Mary	N	7.7
4	Jeen	K	9.8

```
#isnull()
dfn.isnull()
```

Output

	Names	Initials	SEM1	SEM2	SEM3	Grade	Placed
0	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False
3	False	False	False	False	True	False	False
4	False	False	False	False	False	False	False
5	False	False	False	False	False	False	False
6	False	False	False	False	False	False	False
7	False	False	False	False	False	False	False
8	False	False	True	False	False	False	False
9	False	False	True	False	False	False	False
10	False	False	True	False	False	False	False
11	False	False	False	False	False	False	False
12	False	False	False	False	False	False	False
13	False	False	False	False	False	False	False
14	False	False	False	False	False	False	False
15	False	False	False	False	False	False	False
16	False	False	False	False	False	False	False

```
#isnull()
dfn=pd.read_csv("/content/grades_withnulls.csv")
dfn.isnull().head(7)
```

Output:

	Names	Initials	SEM1	SEM2	SEM3	Grade	Placed
0	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False
3	False	False	False	False	True	False	False
4	False	False	False	False	False	False	False
5	False	False	False	False	False	False	False
6	False	False	False	False	False	False	False

```
dfn.isnull().sum()# to view how many nulls we have per column.
```

Output:

```
Names      0
Initials   0
SEM1       3
SEM2       0
SEM3       1
Grade      0
Placed     0
dtype: int64
```

Code:

```
dfn.isnull().sum().sum()# to view the total nulls
```

Output:

4

3.4 dropna() :

It is a function used to remove rows or columns with missing values (NaN) from a DataFrame

Code:

```
dfn.dropna()# dropping all the rows with nulls
```

Output:

	Names	Initials	SEM1	SEM2	SEM3	Grade	Placed
0	Joe	K	9.8	10.0	9.9	A+	1
1	Rajesh	M	8.9	9.1	9.3	A	1
2	Kissan	V	9.9	9.8	10.0	A	0
4	Jeen	K	9.8	9.1	9.9	A+	1
5	Raj	M	8.9	9.1	9.3	A	1
6	Hassan	V	9.9	9.0	9.2	A	1
7	Mari	N	7.7	8.0	7.1	B	1
11	Maya	N	7.7	8.0	7.1	B	0
12	Jolin	K	9.8	9.1	9.9	A+	1
13	Rajesh	M	8.9	9.1	9.3	A	1
14	Riya	M	9.3	9.9	10.0	A	1
15	Sana	V	9.9	9.3	9.2	A	0
16	Mark	N	7.7	8.0	7.0	B	0

3.5 fullna():

You can use the fillna() function to fill the null values in the dataset.

Code:

```
#fillna()
dfn.fillna(5)
```

Output:

	Names	Initials	SEM1	SEM2	SEM3	Grade	Placed
0	Joe	K	9.8	10.0	9.9	A+	1
1	Rajesh	M	8.9	9.1	9.3	A	1
2	Kissan	V	9.9	9.8	10.0	A	0
3	Mary	N	7.7	8.0	5.0	B	0
4	Jeen	K	9.8	9.1	9.9	A+	1
5	Raj	M	8.9	9.1	9.3	A	1
6	Hassan	V	9.9	9.0	9.2	A	1
7	Mari	N	7.7	8.0	7.1	B	1
8	Jess	K	5.0	9.1	9.9	A+	1
9	Rajini	M	5.0	9.1	9.3	A	0
10	Kiran	V	5.0	9.3	9.2	A	0
11	Maya	N	7.7	8.0	7.1	B	0
12	Jolin	K	9.8	9.1	9.9	A+	1
13	Rajesh	M	8.9	9.1	9.3	A	1
14	Riya	M	9.3	9.9	10.0	A	1
15	Sana	V	9.9	9.3	9.2	A	0
16	Mark	N	7.7	8.0	7.0	B	0

Code:

```
dfc1=dfn.fillna(5)
dfc1
```

Output:

	Names	Initials	SEM1	SEM2	SEM3	Grade	Placed
0	Joe	K	9.8	10.0	9.9	A+	1
1	Rajesh	M	8.9	9.1	9.3	A	1
2	Kissan	V	9.9	9.8	10.0	A	0
3	Mary	N	7.7	8.0	5.0	B	0
4	Jeen	K	9.8	9.1	9.9	A+	1
5	Raj	M	8.9	9.1	9.3	A	1
6	Hassan	V	9.9	9.0	9.2	A	1
7	Mari	N	7.7	8.0	7.1	B	1
8	Jess	K	5.0	9.1	9.9	A+	1
9	Rajini	M	5.0	9.1	9.3	A	0
10	Kiran	V	5.0	9.3	9.2	A	0
11	Maya	N	7.7	8.0	7.1	B	0
12	Jolin	K	9.8	9.1	9.9	A+	1
13	Rajesh	M	8.9	9.1	9.3	A	1
14	Riya	M	9.3	9.9	10.0	A	1
15	Sana	V	9.9	9.3	9.2	A	0
16	Mark	N	7.7	8.0	7.0	B	0

3.6 Cleaning with mean:

Code

```
m=dfn['SEM3'].mean()
print(m)
```

Output:

9.100000000000001

Code:

```
dfc2=dfn.fillna(m)
dfc2
```

Output:

	Names	Initials	SEM1	SEM2	SEM3	Grade	Placed
0	Joe	K	9.8	10.0	9.9	A+	1
1	Rajesh	M	8.9	9.1	9.3	A	1
2	Kissan	V	9.9	9.8	10.0	A	0
3	Mary	N	7.7	8.0	9.1	B	0
4	Jeen	K	9.8	9.1	9.9	A+	1
5	Raj	M	8.9	9.1	9.3	A	1
6	Hassan	V	9.9	9.0	9.2	A	1
7	Mari	N	7.7	8.0	7.1	B	1
8	Jess	K	9.1	9.1	9.9	A+	1
9	Rajini	M	9.1	9.1	9.3	A	0
10	Kiran	V	9.1	9.3	9.2	A	0
11	Maya	N	7.7	8.0	7.1	B	0
12	Jolin	K	9.8	9.1	9.9	A+	1
13	Rajesh	M	8.9	9.1	9.3	A	1
14	Riya	M	9.3	9.9	10.0	A	1
15	Sana	V	9.9	9.3	9.2	A	0
16	Mark	N	7.7	8.0	7.0	B	0

3.7 Dropping duplicates:

The drop_duplicates() method removes duplicate rows.

Code:

```
dfc3=dfc2.drop_duplicates()
dfc3
```

Output:

	Names	Initials	SEM1	SEM2	SEM3	Grade	Placed
0	Joe	K	9.8	10.0	9.9	A+	1
1	Rajesh	M	8.9	9.1	9.3	A	1
2	Kissan	V	9.9	9.8	10.0	A	0
3	Mary	N	7.7	8.0	9.1	B	0
4	Jeen	K	9.8	9.1	9.9	A+	1
5	Raj	M	8.9	9.1	9.3	A	1
6	Hassan	V	9.9	9.0	9.2	A	1
7	Mari	N	7.7	8.0	7.1	B	1
8	Jess	K	9.1	9.1	9.9	A+	1
9	Rajini	M	9.1	9.1	9.3	A	0
10	Kiran	V	9.1	9.3	9.2	A	0
11	Maya	N	7.7	8.0	7.1	B	0
12	Jolin	K	9.8	9.1	9.9	A+	1
14	Riya	M	9.3	9.9	10.0	A	1
15	Sana	V	9.9	9.3	9.2	A	0
16	Mark	N	7.7	8.0	7.0	B	0

Columns:

Code:

```
dfc3.rename( columns= {"Grade":"GPA"})
```

Output:

	Names	Initials	SEM1	SEM2	SEM3	Grade	Placed
0	Joe	K	9.8	10.0	9.9	A+	1
1	Rajesh	M	8.9	9.1	9.3	A	1
2	Kissan	V	9.9	9.8	10.0	A	0
3	Mary	N	7.7	8.0	9.1	B	0
4	Jeen	K	9.8	9.1	9.9	A+	1
5	Raj	M	8.9	9.1	9.3	A	1
6	Hassan	V	9.9	9.0	9.2	A	1
7	Mari	N	7.7	8.0	7.1	B	1
8	Jess	K	9.1	9.1	9.9	A+	1
9	Rajini	M	9.1	9.1	9.3	A	0
10	Kiran	V	9.1	9.3	9.2	A	0
11	Maya	N	7.7	8.0	7.1	B	0
12	Jolin	K	9.8	9.1	9.9	A+	1
14	Riya	M	9.3	9.9	10.0	A	1
15	Sana	V	9.9	9.3	9.2	A	0
16	Mark	N	7.7	8.0	7.0	B	0

```
Code:
dfc3.head()
```

Output:

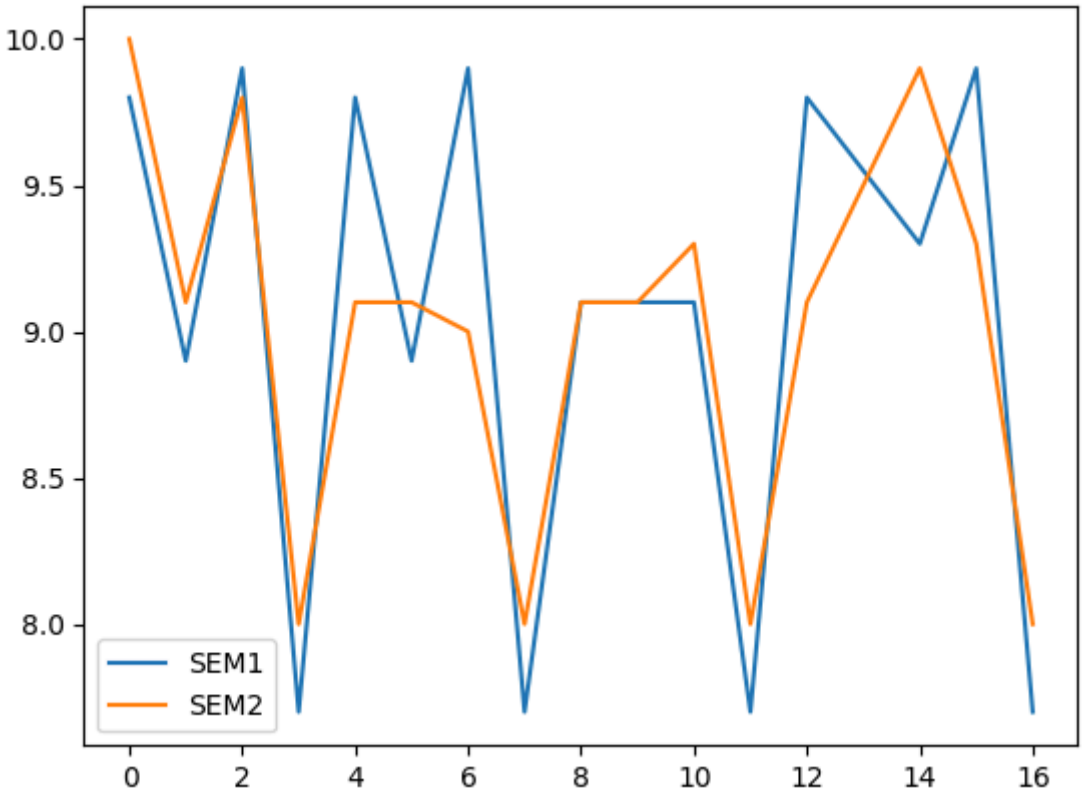
	Names	Initials	SEM1	SEM2	SEM3	Grade	Placed
0	Joe	K	9.8	10.0	9.9	A+	1
1	Rajesh	M	8.9	9.1	9.3	A	1
2	Kissan	V	9.9	9.8	10.0	A	0
3	Mary	N	7.7	8.0	9.1	B	0
4	Jeen	K	9.8	9.1	9.9	A+	1

3.8 Plotting:

Code:

```
dfc3[['SEM1','SEM2']].plot.line()
```

Output:

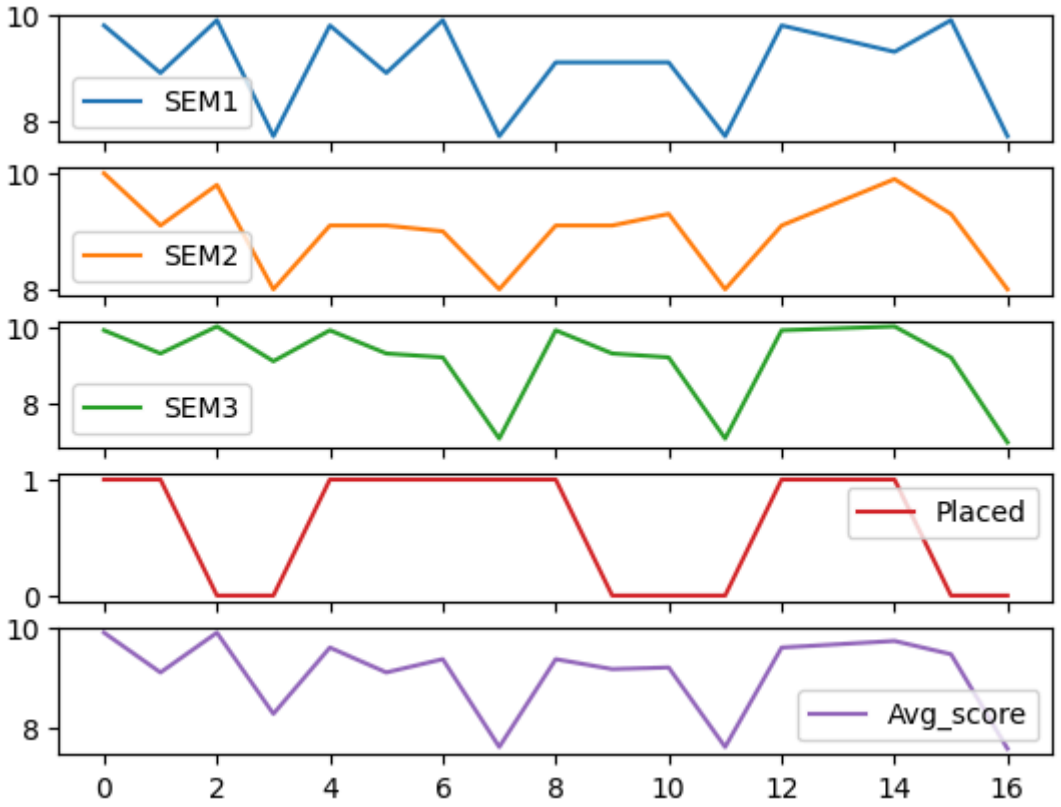


3.9 Subplots:

Code:

dfc3.plot.line(subplots=True)

Output:



Seaborn:

Seaborn is a Python data visualisation library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics**.**

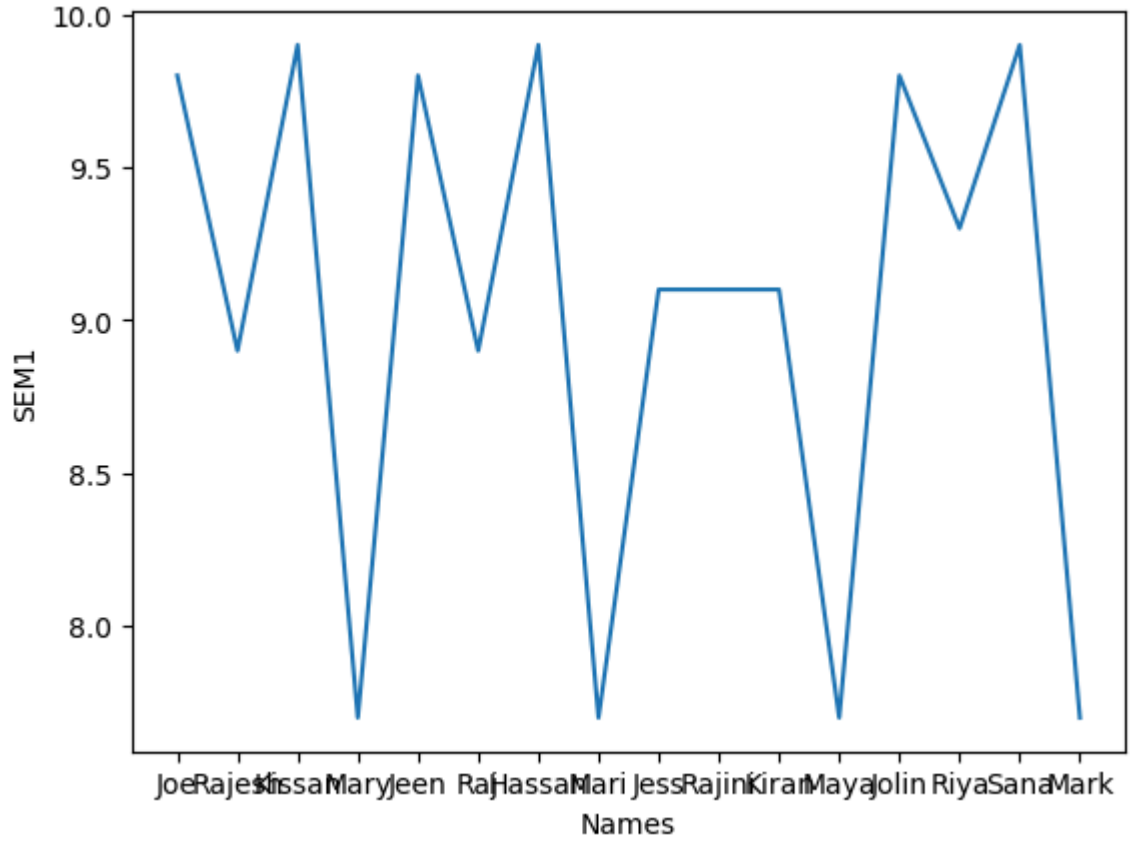
4.1 Importing:

```
import seaborn as sns
```

Code:

```
p1=sns.lineplot(x='Names',y='SEM1',data=dfc2)
```

Output:



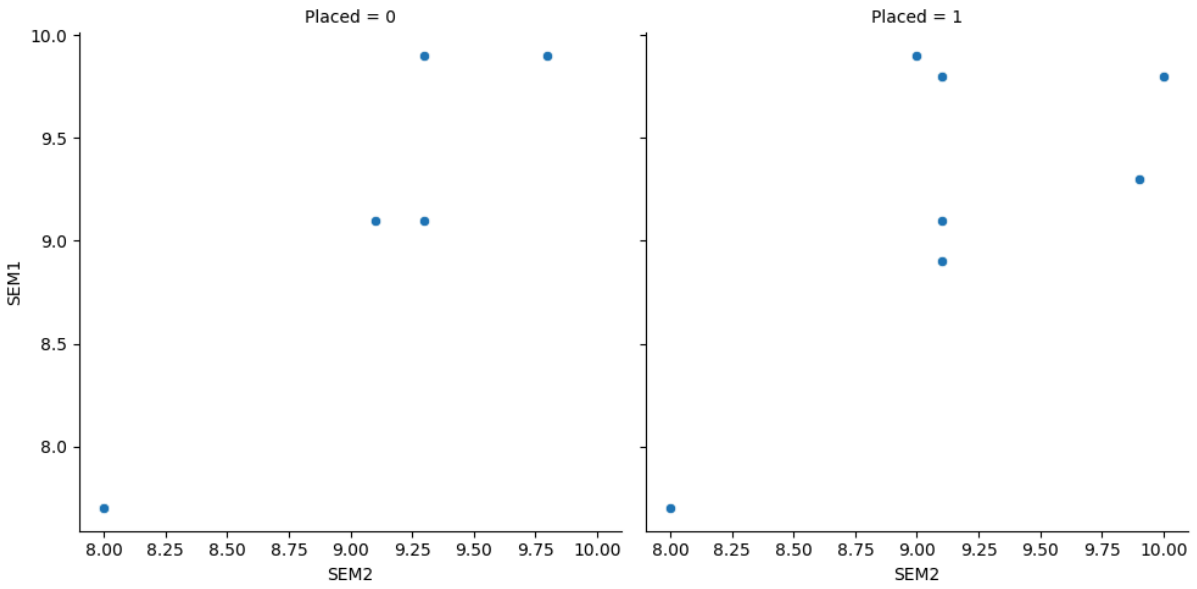
4.2 Relplot:

The Seaborn Relational Plot (relplot) allows us to visualise how variables within a dataset relate to each other.

Code:

```
sns.relplot(
  data=dfc3,
  x='SEM2',y='SEM1',col="Placed")
```

Output:



Load diabets.csv

- create relplot with age in the x axis and class as columns

Code:

```
df=pd.read_csv("/content/diabetcsv.csv")
df
```

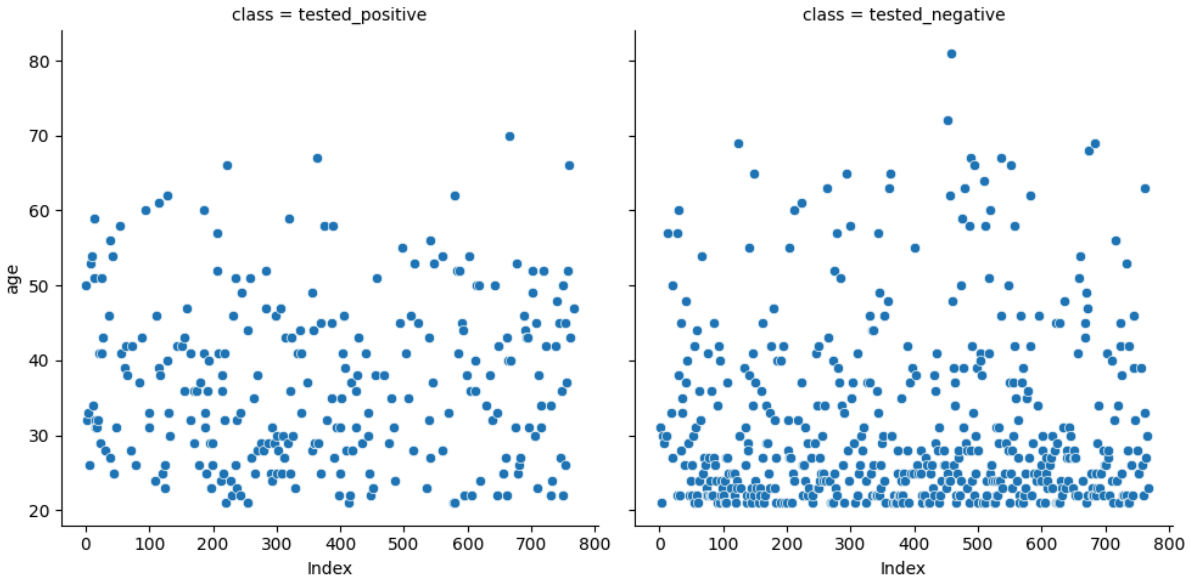
Output:

	preg	plas	pres	skin	insu	mass	pedi	age	class
0	6	148	72	35	0	33.6	0.627	50	tested_positive
1	1	85	66	29	0	26.6	0.351	31	tested_negative
2	8	183	64	0	0	23.3	0.672	32	tested_positive
3	1	89	66	23	94	28.1	0.167	21	tested_negative
4	0	137	40	35	168	43.1	2.288	33	tested_positive
5	5	116	74	0	0	25.6	0.201	30	tested_negative
6	3	78	50	32	88	31.0	0.248	26	tested_positive
7	10	115	0	0	0	35.3	0.134	29	tested_negative
8	2	197	70	45	543	30.5	0.158	53	tested_positive
9	8	125	96	0	0	0.0	0.232	54	tested_positive

Code:

```
df=pd.read_csv("/content/diabetcsv.csv")
df['Index']=(range(0,768))
sns.relplot(
    data=df,
    x="Index",y="age",col="class")
```

Output:



In-built datasets in seaborn

- Tips
- Dowjones
- Fmri
- Dots
- Health Exp
- To load datasets use:
- =>load_datasets("datasetname")

hue = different colour for diff category style=different colour for diff marker

Code:

```
import seaborn as sns
tips=sns.load_dataset("tips")
tips.head()
```

Output:

	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

Code:

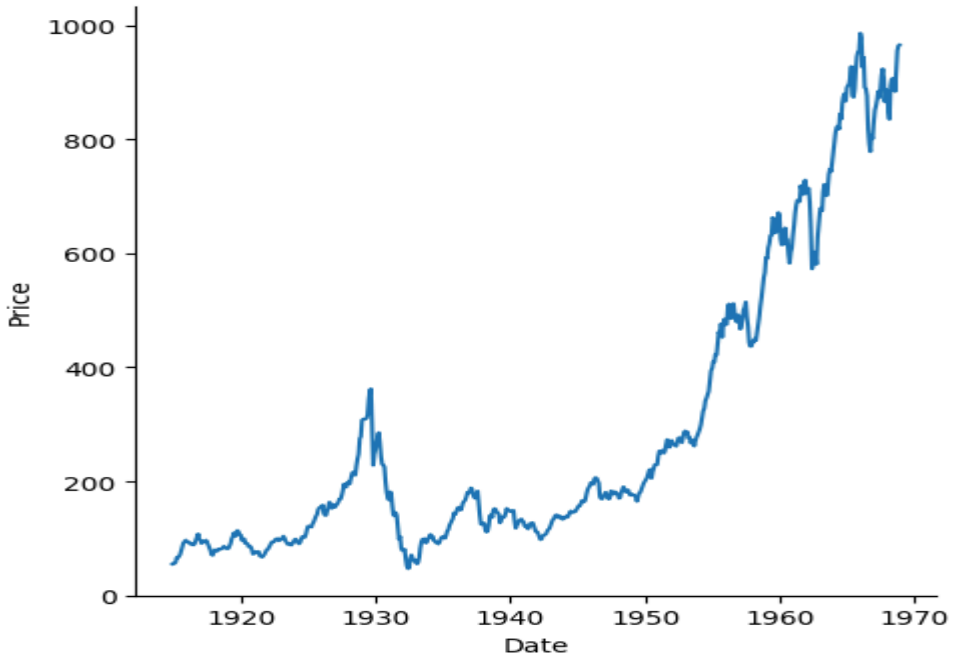
```
dowjones=sns.load_dataset("dowjones")
dowjones.head()
```

	Date	Price
0	1914-12-01	55.00
1	1915-01-01	56.55
2	1915-02-01	56.00
3	1915-03-01	58.30
4	1915-04-01	66.45

Code:

```
sns.relplot(data=dowjones,x='Date',y='Price',kind='line')
```

Output:



To access the fmri :

Code:

```
fmri=sns.load_dataset("fmri")
fmri.head()
```

Output:

	subject	timepoint	event	region	signal
0	s13	18	stim	parietal	-0.017552
1	s5	14	stim	parietal	-0.080883
2	s12	18	stim	parietal	-0.081033
3	s11	18	stim	parietal	-0.046134
4	s10	18	stim	parietal	-0.037970

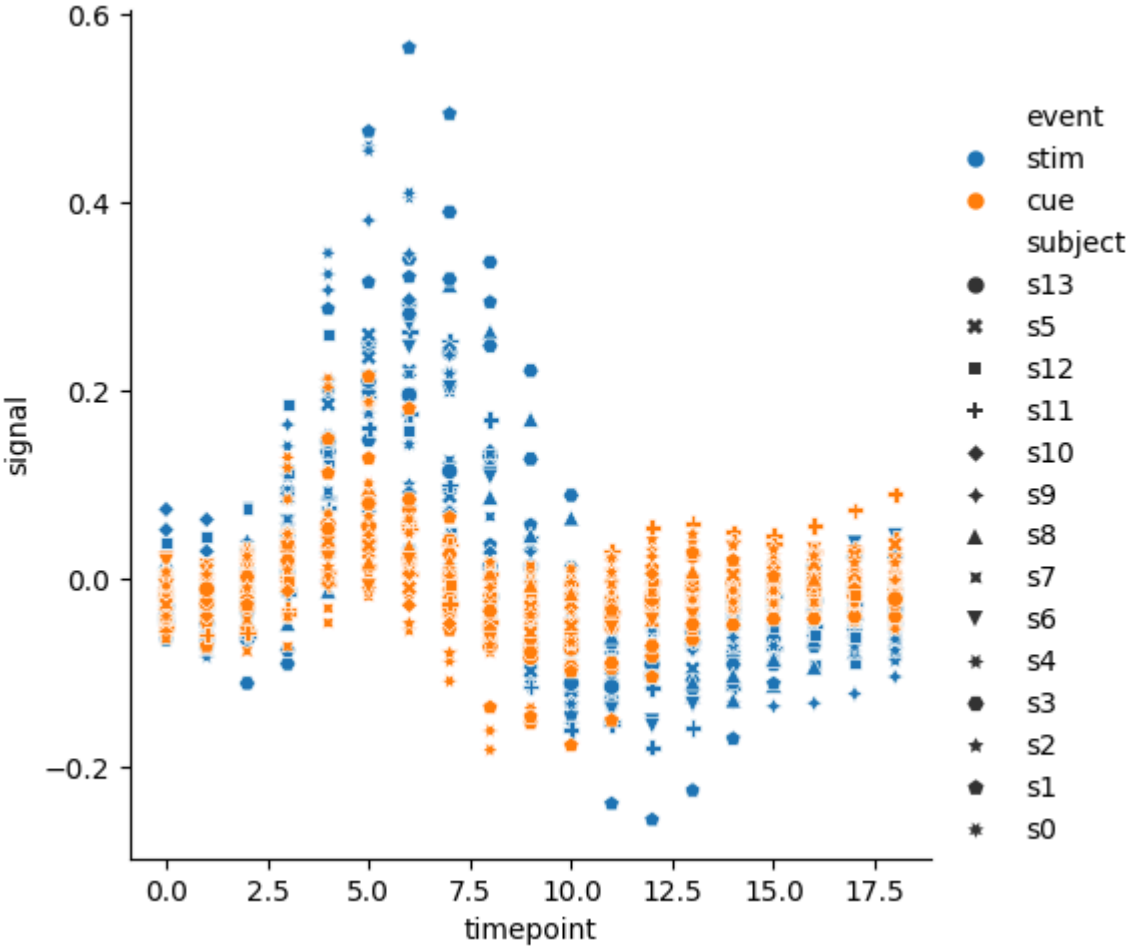
subject means patient number

- region=brain location

Code:

```
sns.relplot(data=fmri,x='timepoint',y='signal',hue='event',style='subject')
```

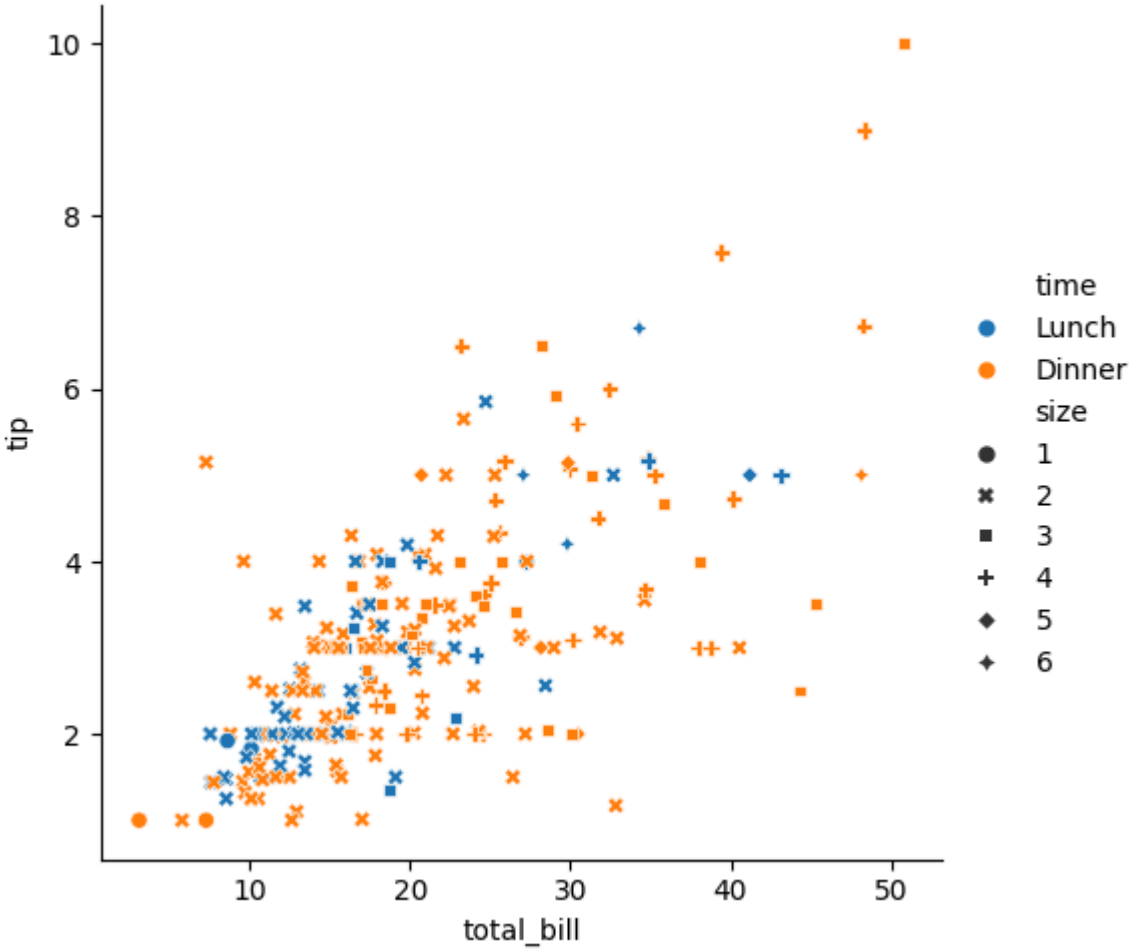
Output:



Code:

```
#hue -->Creating difference based on a column via colours
tips=sns.load_dataset("tips")
sns.relplot(data=tips,x='total_bill',y='tip',hue="time",style="size")
```

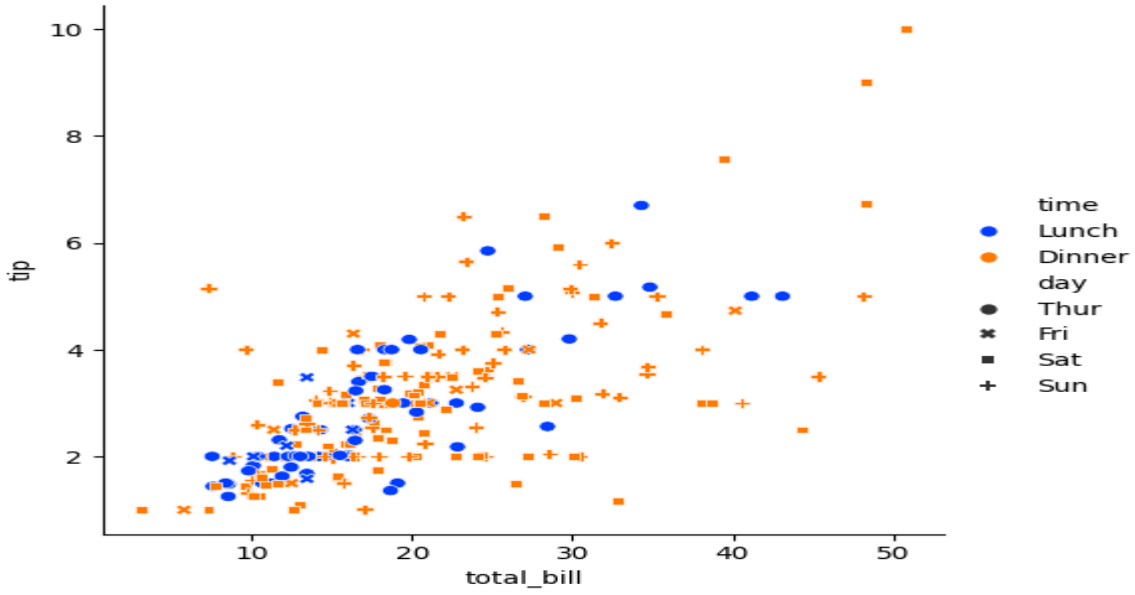
Output:



Code:

```
tips=sns.load_dataset("tips")
sns.relplot(data=tips,x='total_bill',y='tip',hue="time",style="day" ,palette="bright")
```

Output:



palette

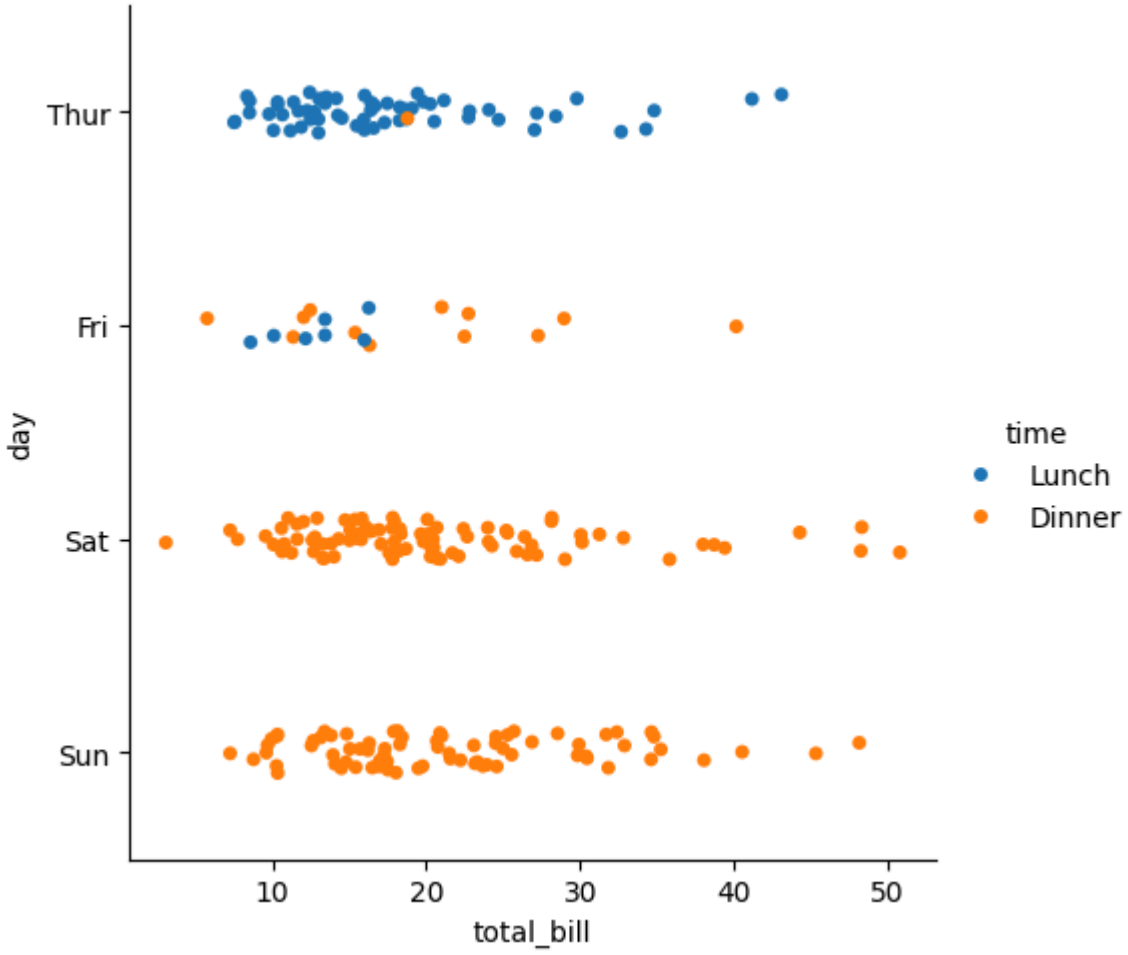
- pastel
- bright
- dark
- muted
- color blind
- Deep

Category plot:

The Seaborn catplot() function is used to create figure-level relational plots onto a Seaborn FacetGrid

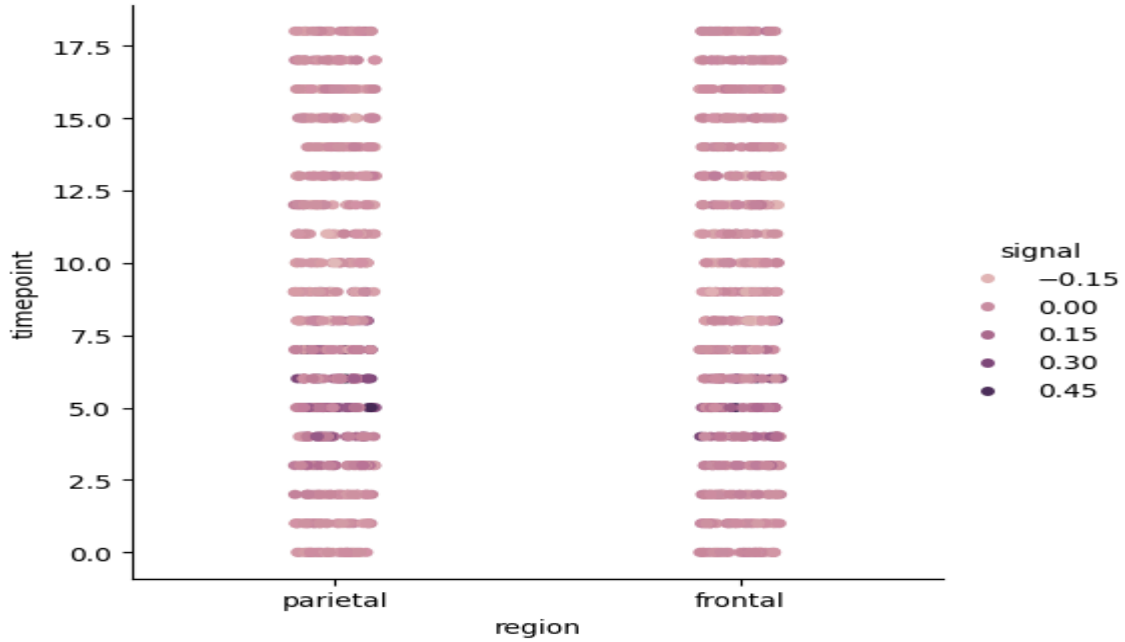
```
#category plot
sns.catplot(data=tips,x='total_bill',y='day',hue="time")
```

Output:




```
sns.catplot(data=fmri,x='region',y='timepoint',hue="signal")
```

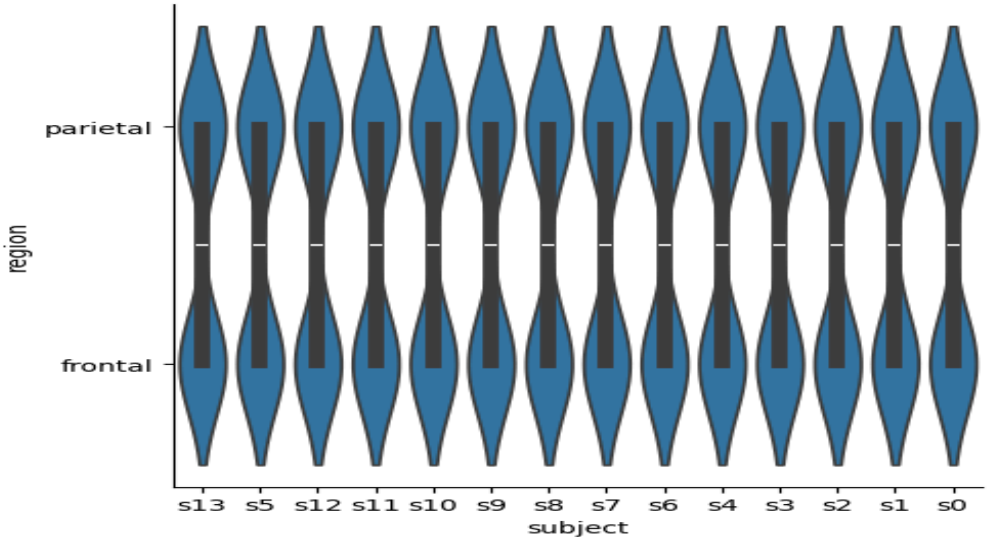
Output:



Code:

```
sns.catplot(data=fmri,x='subject',y='region',kind='violin')
```

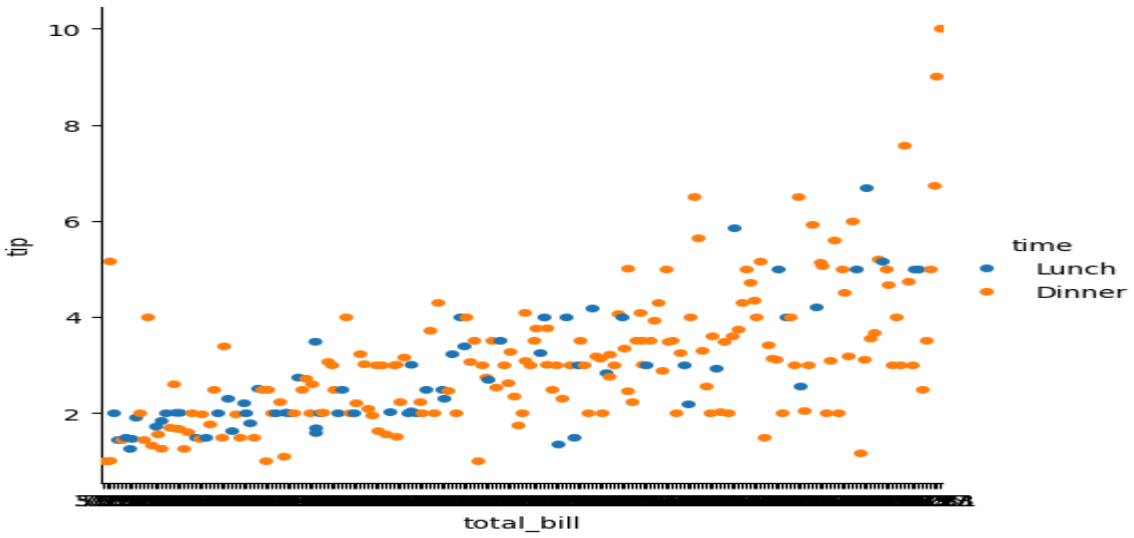
Output:



Code:

```
sns.catplot(data=tips,x='total_bill',y='tip',hue="time")
```

Output:



Linear fit

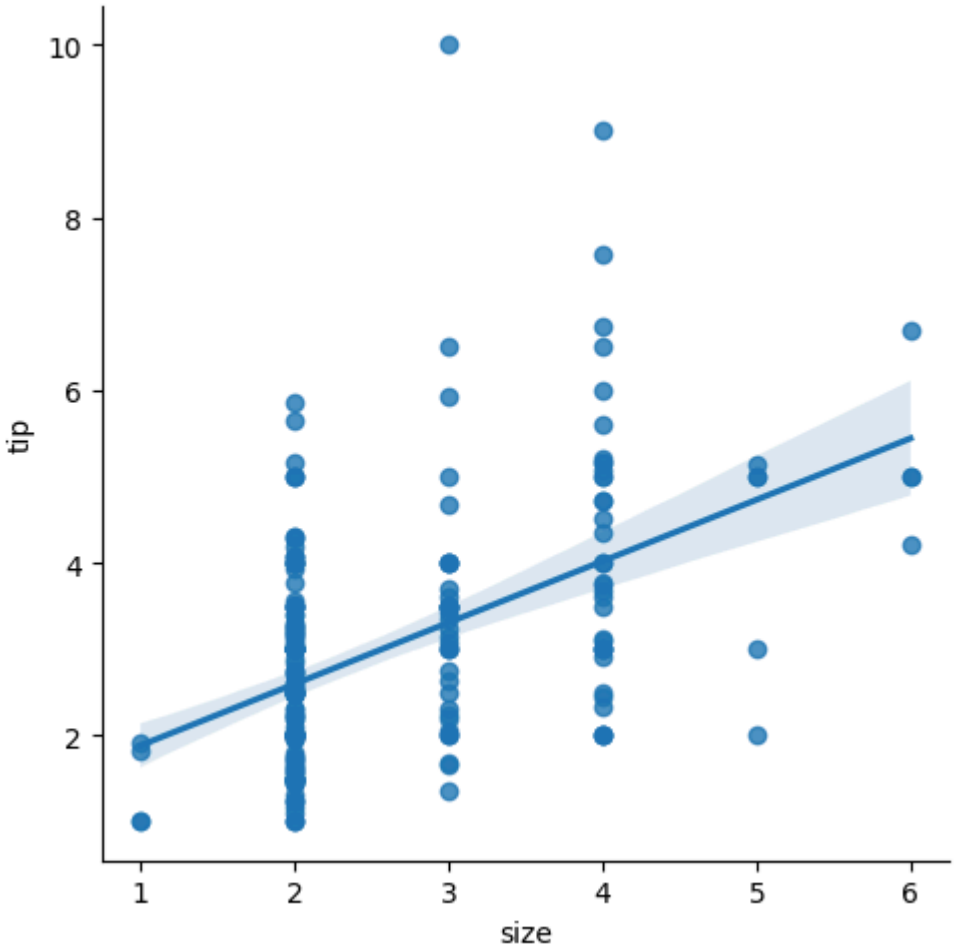
- lmplot()
- STEPS FOR LINEAR FIT
- plotting the dataset
- fitting the line
- predicting

lmplot() method is used to draw a scatter plot onto a FacetGrid.

Code:

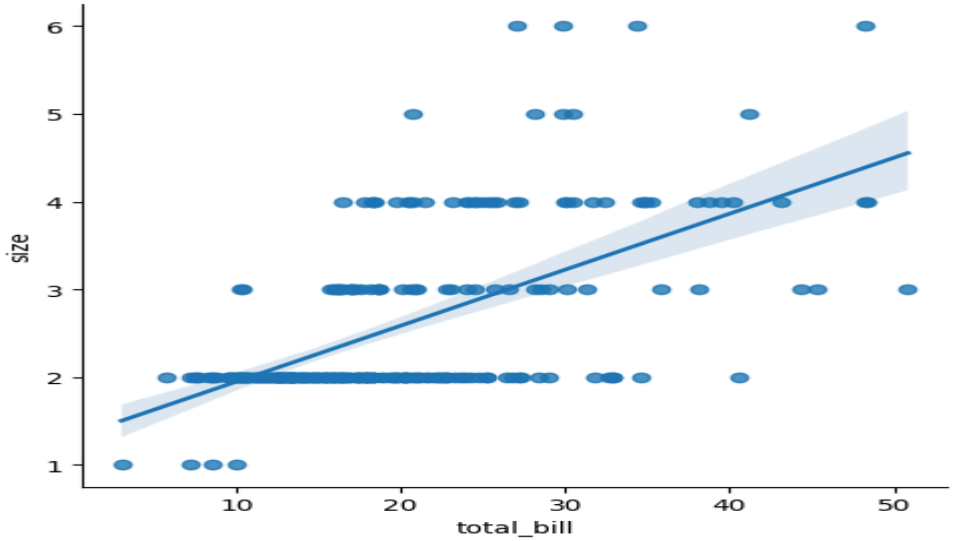
```
sns.lmplot(x='size',y='tip',data =tips)
```

Output:



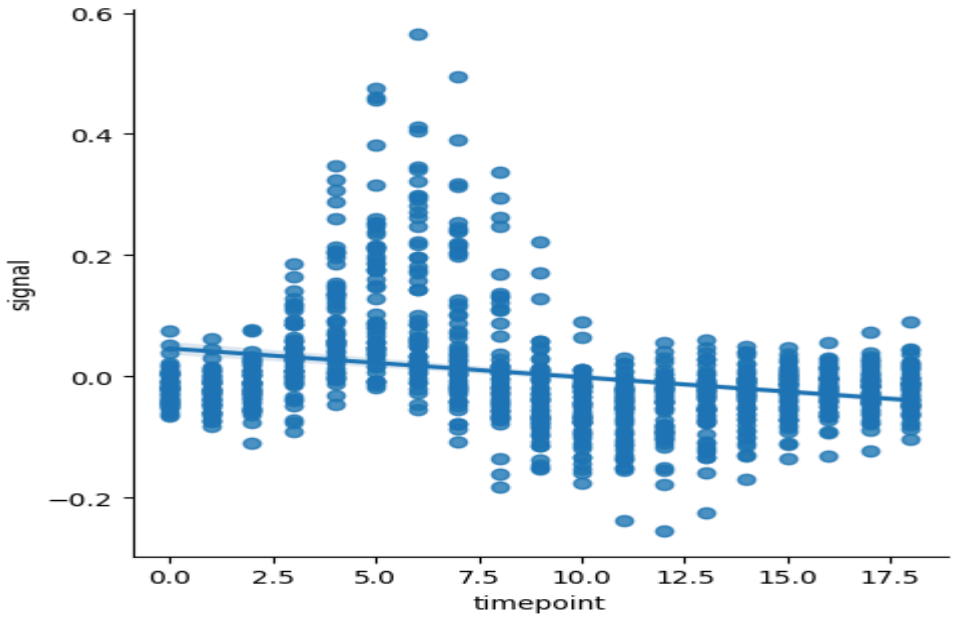

```
sns.lmplot(x='total_bill',y='size',data =tips)
```

Output:




```
sns.lmplot(x='timepoint',y='signal',data =fmri)
```

Output:



Web scraping:

- Loading data from websites
- Unstructured in HTML
- Convertible into sp

Scrapper

- Extract all the data on particular sites
- Specific data that a user wants

Process:

- URL->
- HTML code->
- Elements(CSS/JS)
- Scrapes the required data->
- Saves it in required format(CSV,xlsx,json)

Applications:

- Email Marketing
- Sentiment Analysis
- News Monitoring
- Market Research
- Price Monitoring

Libraries:

- Beautifulsoup- scoop out the html code-BS4(library)
- Requests to access the webpage
- Pandas-create dataframes from excel,csv,and txt files
- selenium
- Webdriver
- Webdriver_manager

To install any library:
!pip install library
!pip install–upgrade library #to upgrade to the recent version.
Importing subpackage:
from parent import child
Ex: from Bs4 import beautifulsoup

BeautifulSoup

- Used to scrape data from static website
- Package bs4:subpackage Beautifulsoup

Functions	**Purpose**	**Attributes**
Beautifulsoup()	To extract html code from a web page	.text html
find()	To find first element of a kind	(‘element_name)
findall()	To find all elements of a kind	(‘element_name)

Requests:

- Used to send the request to a web page
- ger(‘url’)

Project 1

Extracting Html code code of any website

Program flow

1. Import libraries
2. Save the url
3. Using requests.get(),access the web page
4. Using Beautifulsoup(),access the HTML code
5. Using find,access the table
6. Using find_all,access the rows of the table
7. Using.text ,extract only the text(removing html page)
8. Create a dataframe using pandas
9. Push all the extracted data into dataframe
10. Using to_csv,save the dataframe in csv format

Step-1

```
#Importing libraries
from bs4 import BeautifulSoup
import requests
import pandas as pd
```

Step-2

```
#save the url
url="https://www.instagram.com/"
```

Step-3

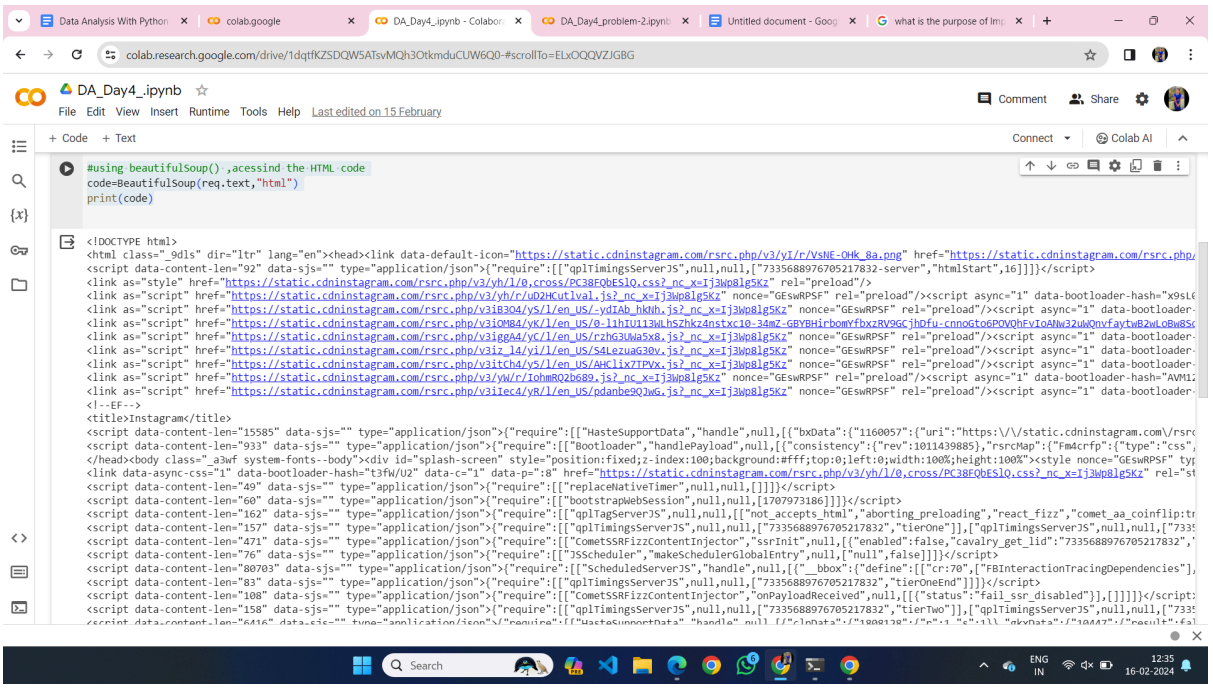
```
# using requests get,access the web page
req=requests.get(url)
```

Step-4

```
#using beautifulSoup() ,accessing the HTML code
code=BeautifulSoup(req.text,"html")
print(code)
```

Step-5

output:



Accessing elements:

- Go to webpage
- Right click—->Inspect
- Click on 
- Now hover on the element you want to access
- The html tags/code for the specific elements is highlighted
- Extract the elements name and import in your program using XPATH or BY ID

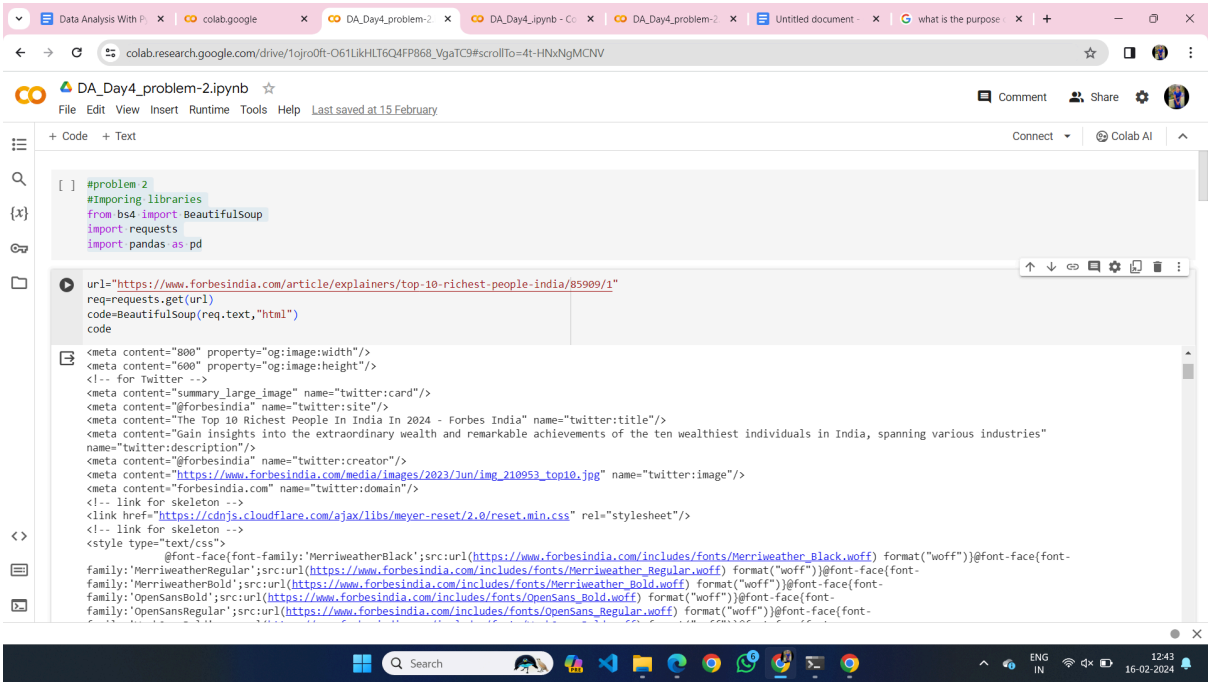
Problem-2

Step-1

```
#Importing libraries
from bs4 import BeautifulSoup
import requests
import pandas as pd
```

Step-2

```
url="https://www.forbesindia.com/article/explainers/top-10-richest-people-india/85909/1"
req=requests.get(url)
code=BeautifulSoup(req.text,"html")
code
```

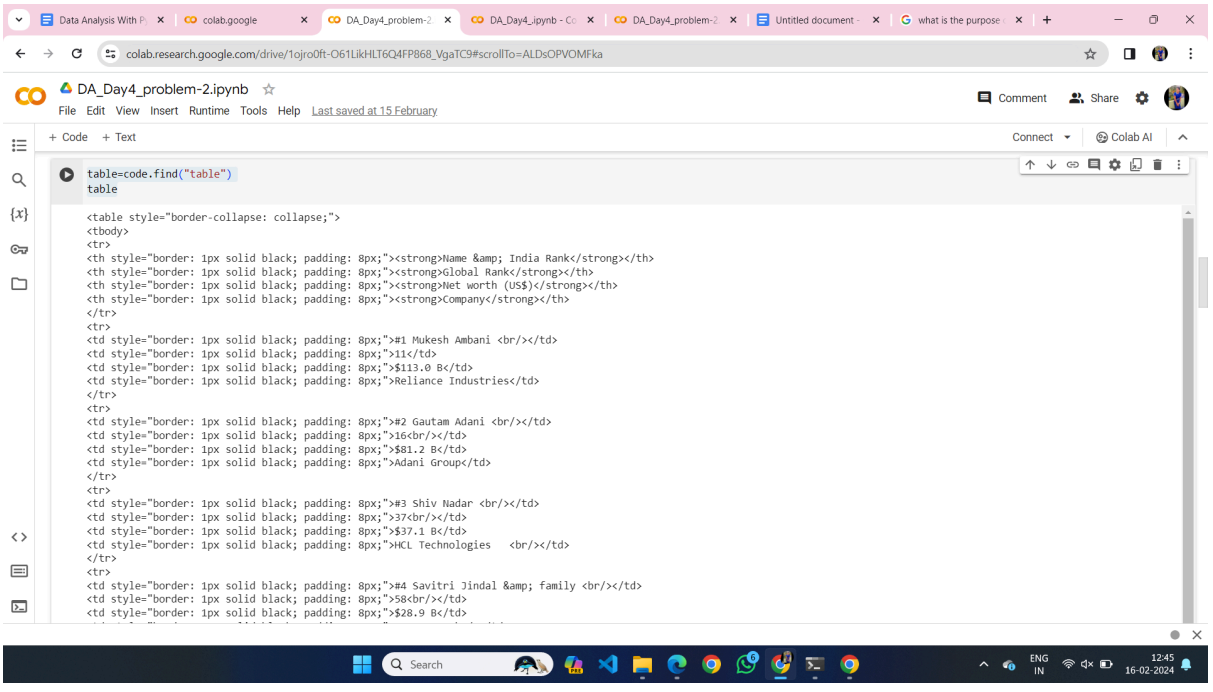


Step-2

```
table=code.find("table")
```

```
Table
```

Output:



Step-4

```
heading=table.find_all("th")
```

```
heading
```

Output:

[<th style="border: 1px solid black; padding: 8px;">Name & India Rank</th>,
 <th style="border: 1px solid black; padding: 8px;">Global Rank</th>,
 <th style="border: 1px solid black; padding: 8px;">Net worth (US$)</th>,
 <th style="border: 1px solid black; padding: 8px;">Company</th>]

Step-5

```
head=[i.text for i in heading]
head
```

Output:

['Name & India Rank', 'Global Rank', 'Net worth (US$)', 'Company']

Step-6

```
df=pd.DataFrame(columns=head)
print(df)
```

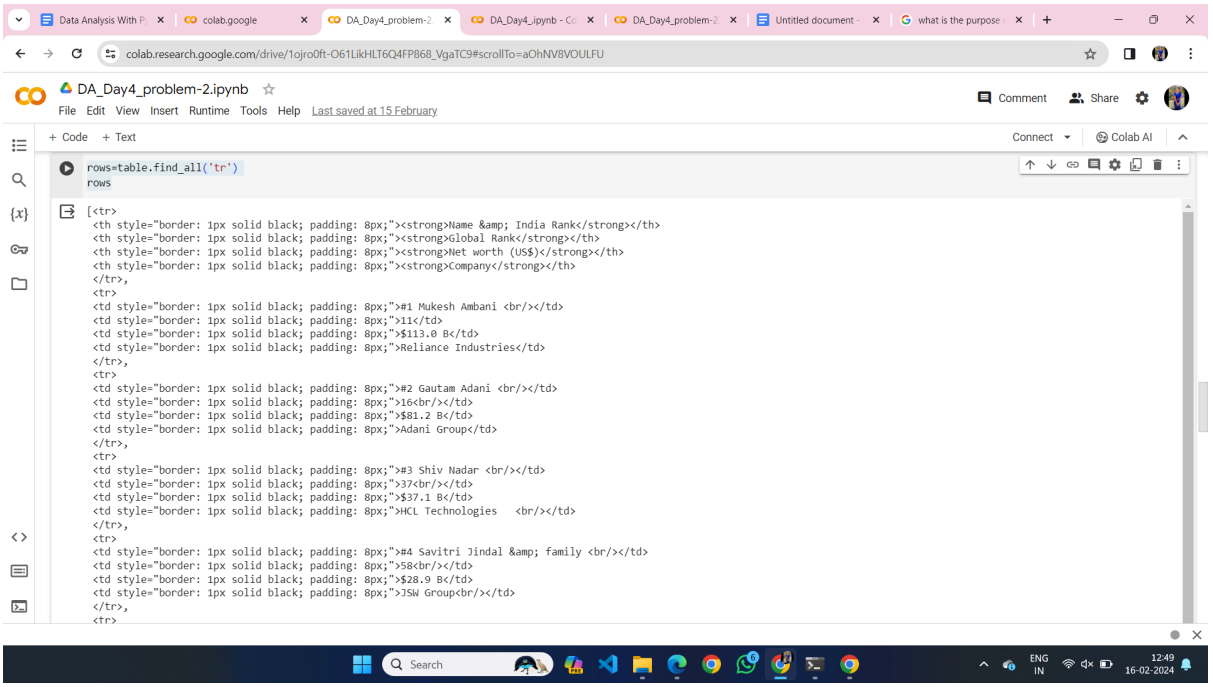
Output:

```
Empty DataFrame
Columns: [Name & India Rank, Global Rank, Net worth (US$), Company]
Index: []
```

Step-7

```
rows=table.find_all('tr')
rows
```

Output:



Step-8

```
for i in rows[1:9]:
  rd=i.find_all('td')
  eachrow=[i.text for i in rd]
```

```
  print(eachrow)
```

Output:

```
['#1 Mukesh Ambani ', '11', '$113.0 B', 'Reliance Industries']
['#2 Gautam Adani ', '16', '$81.2 B', 'Adani Group']
['#3 Shiv Nadar ', '37', '$37.1 B', 'HCL Technologies\xa0\xa0 ']
['#4 Savitri Jindal & family ', '58', '$28.9 B', 'JSW Group']
['#5 Cyrus Poonawalla ', '68', '$25.6 B', 'Serum Institute of India']
['#6 Dilip Shanghvi ', '69', '$25.5 B', 'Sun Pharmaceutical Industries
Ltd']
['#7 Kumar Birla ', '97', '$18.9 B', 'Aditya Birla Group']
['#8 Kushal Pal Singh', '98', '$18.9 B', 'DLF Limited']
```

Step-9

```
a=0
for i in rows[1:9]:
  rd=i.find_all('td')
  eachrow=[j.text for j in rd]
  df.loc[a]=eachrow
  a=a+1
df
```

Output:

```
a=0
for i in rows[1:9]:
  rd=i.find_all('td')
  eachrow=[j.text for j in rd]
  df.loc[a]=eachrow
  a=a+1
df
```

	#8 Kushal Pal Singh	98	$18.9 B	DLF Limited
0	#1 Mukesh Ambani	11	$113.0 B	Reliance Industries
1	#2 Gautam Adani	16	$81.2 B	Adani Group
2	#3 Shiv Nadar	37	$37.1 B	HCL Technologies
3	#4 Savitri Jindal & family	58	$28.9 B	JSW Group
4	#5 Cyrus Poonawalla	68	$25.6 B	Serum Institute of India
5	#6 Dilip Shanghvi	69	$25.5 B	Sun Pharmaceutical Industries Ltd
6	#7 Kumar Birla	97	$18.9 B	Aditya Birla Group
7	#8 Kushal Pal Singh	98	$18.9 B	DLF Limited

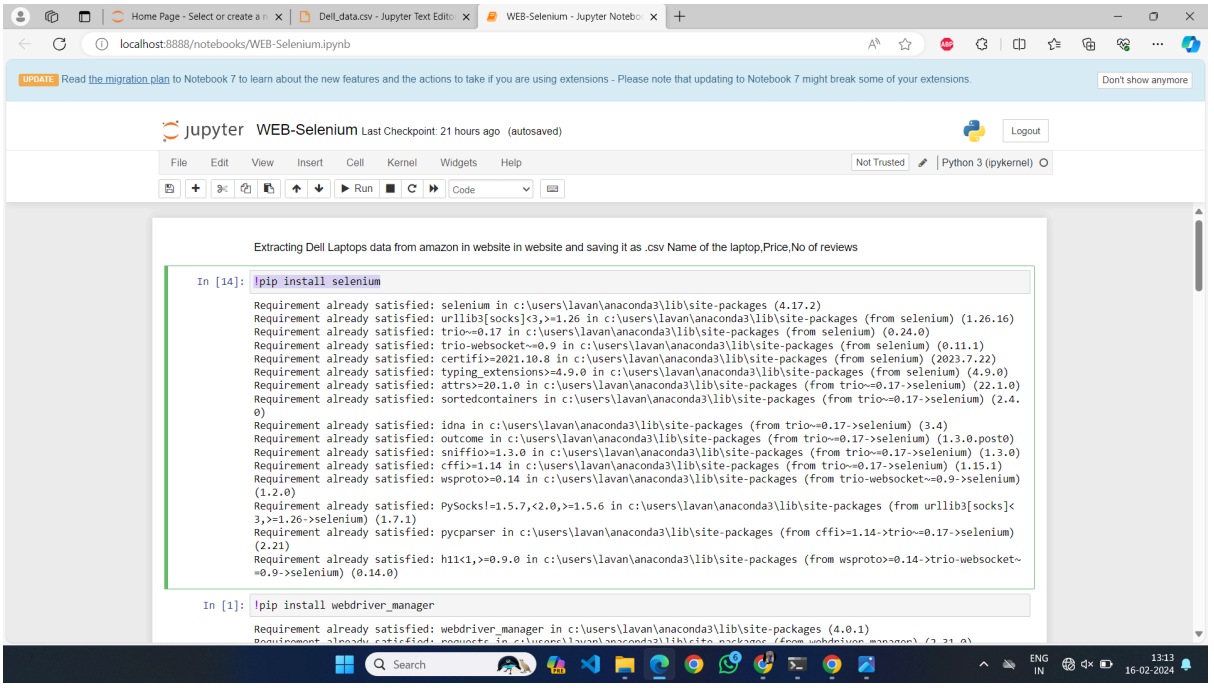
Step-10

```
df.to_csv("Richman.csv")
```

Problem-3

Step-1

!pip install selenium



Step-2

!pip install webdriver_manager

Requirement already satisfied: webdriver_manager in c:\users\lavan\anaconda3\lib\site-packages (4.0.1)
Requirement already satisfied: requests in c:\users\lavan\anaconda3\lib\site-packages (from webdriver_manager) (2.31.0)
Requirement already satisfied: python-dotenv in c:\users\lavan\anaconda3\lib\site-packages (from webdriver_manager) (0.21.0)
Requirement already satisfied: packaging in c:\users\lavan\anaconda3\lib\site-packages (from webdriver_manager) (23.1)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\lavan\anaconda3\lib\site-packages (from requests->webdriver_manager) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:\users\lavan\anaconda3\lib\site-packages (from requests->webdriver_manager) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\lavan\anaconda3\lib\site-packages (from requests->webdriver_manager) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\lavan\anaconda3\lib\site-packages (from requests->webdriver_manager) (2023.7.22)

Step-3

from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from webdriver_manager.chrome import ChromeDriverManager

```
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
```

Step-4

```
#Define options and set browser capabilities
options=webdriver.ChromeOptions()
options.add_argument('--some-option')
#create webdriver instance with options
driver=webdriver.Chrome(options=options)
#access browser capabilities
browser_name=options.to_capabilities()["browserName"]
print(browser_name)
```

Output:

chrome

Step-5

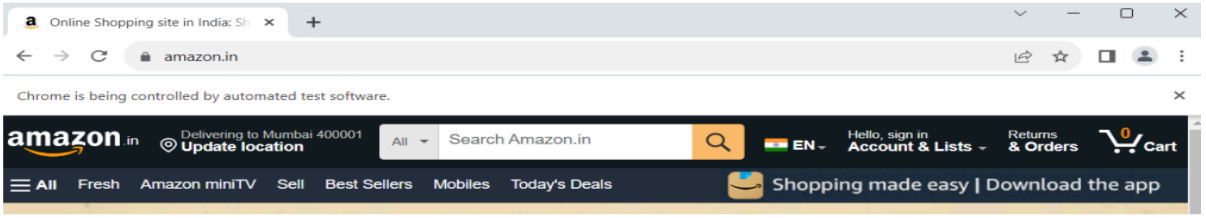
driver.get("https://www.amazon.in")

Step-6

search =driver.find_element(By.ID,"twotabsearchtextbox")

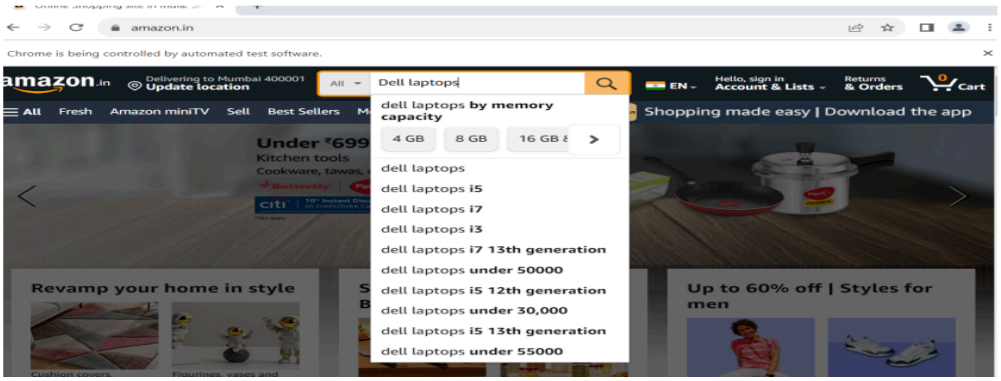
Step-7

search.send_keys("Dell laptops")



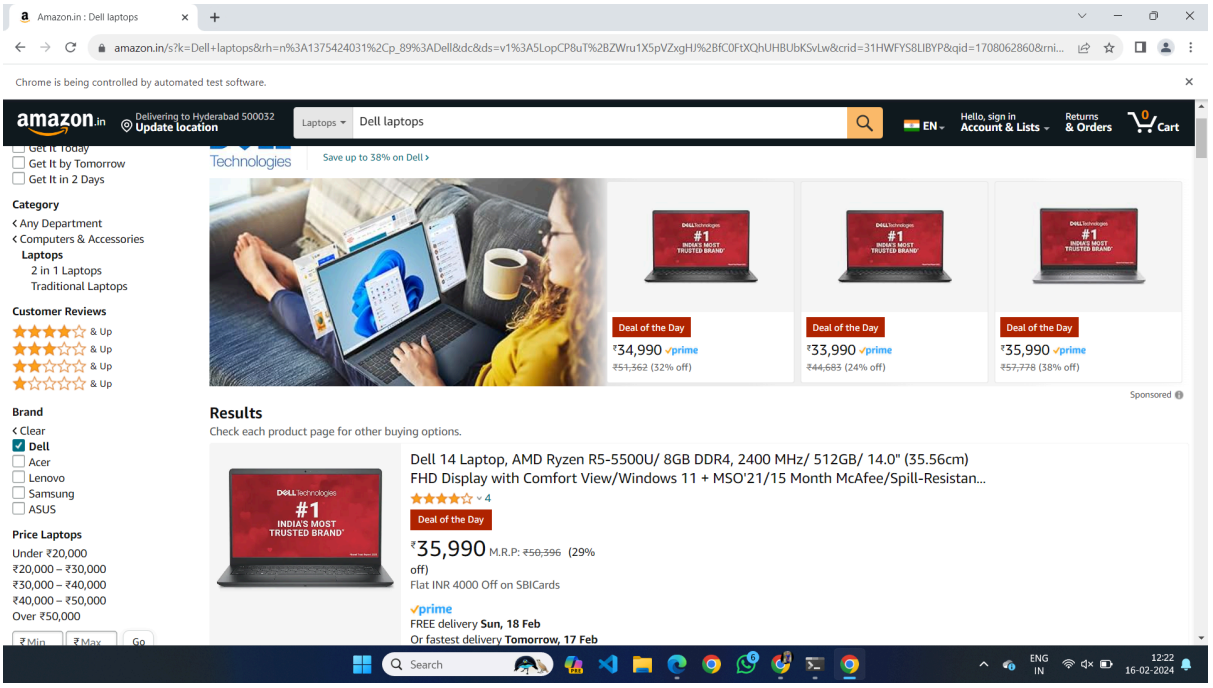
Step-5:

```
search=driver.find_element(By.ID,"twotabsearchtextbox")
```



Step-8

driver.find_element(By.ID,"nav-search-submit-button").click()

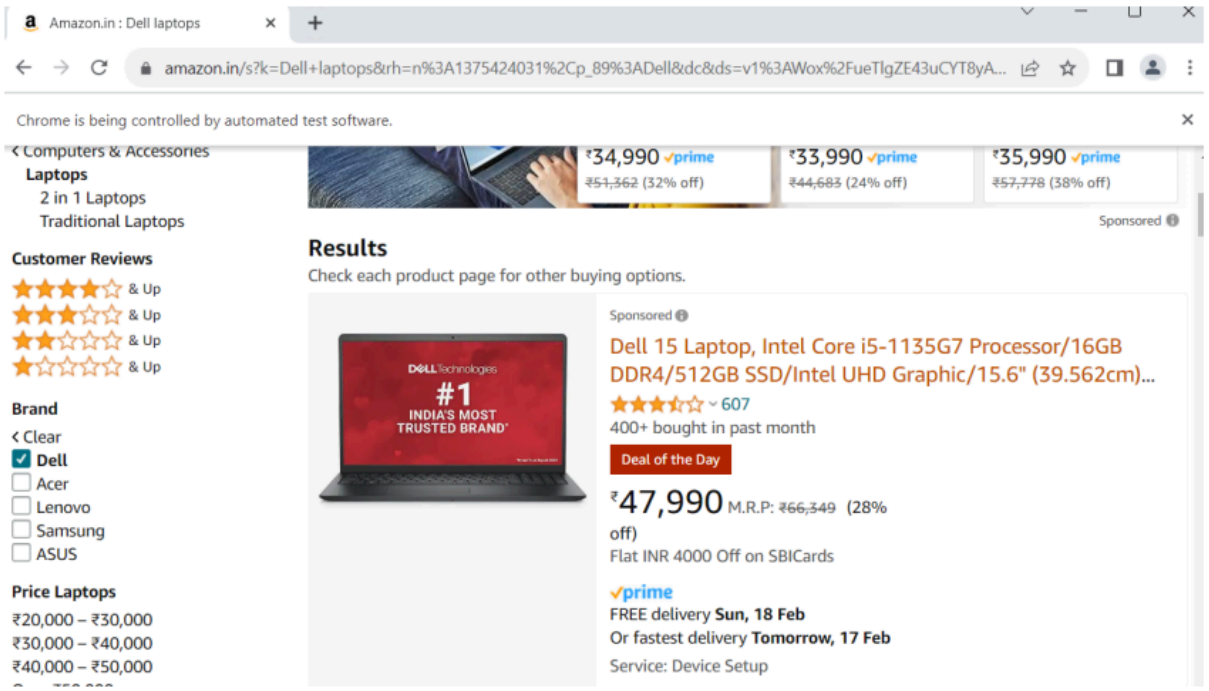


Step-9

driver.find_element(By.XPATH,"//span[text()='Dell']").click()

Output:

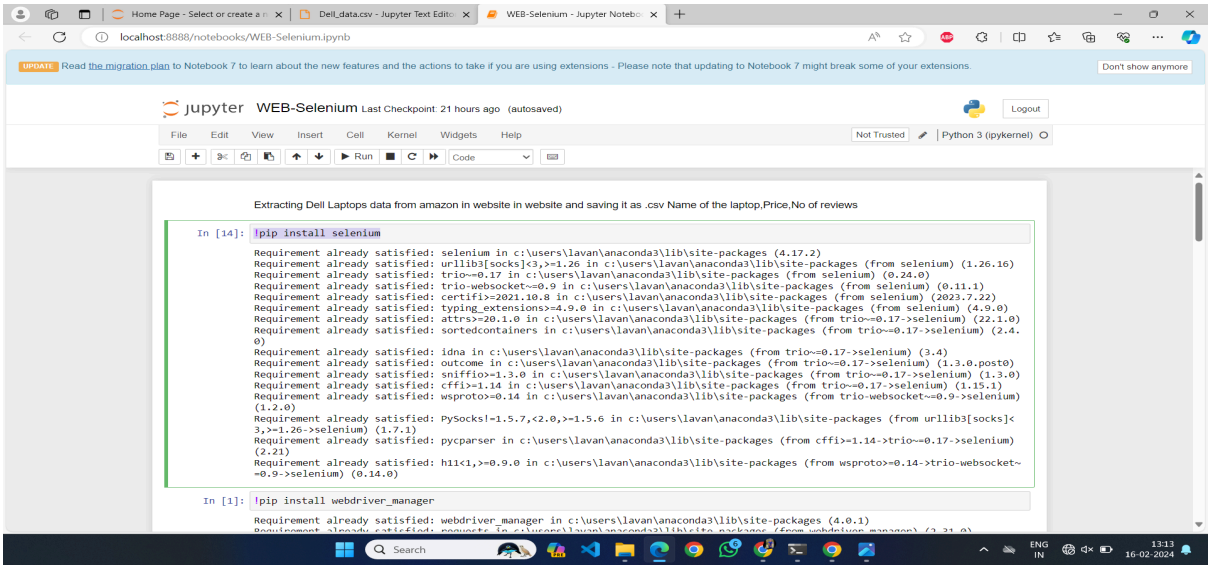
class name for product title a-size-medium a-color-base a-text-normal"



Step-10

```
names=driver.find_elements(By.XPATH,"//span[@class='a-size-medium a-color-base a-text-normal']")
l_names=[i.text for i in names]
print(l_names)
```

Output:



Step-11

```
print(len(l_names))
```

Output:

24

Step-12

```
price=driver.find_elements(By.XPATH,"//span[@class='a-price-whole']")
l_price=[i.text for i in price]
print(l_price)
```

Output:

['34,990', '33,990', '35,990', '35,990', '55,280', '38,990', '49,990', '35,990', '57,990', '44,990', '67,490', '75,990', '33,990', '83,490', '44,990', '23,649', '46,990', '30,630', '71,490', '98,990', '78,490', '50,490', '34,380', '81,490', '82,990', '1,08,599', '75,490']

Step-13

```
l_price.pop(0)
print(l_price)
```

['35,990', '55,280', '38,990', '49,990', '35,990', '57,990', '44,990', '67,490', '75,990', '33,990', '83,490', '44,990', '23,649', '46,990', '30,630', '71,490', '98,990', '78,490', '50,490', '34,380', '81,490', '82,990', '1,08,599', '75,490']

class name for number of reviews---->span a-size-bases-underline-text

```
reviews=driver.find_elements(By.XPATH,"//span[@class='a-size-base s-underline-text']")
l_reviews=[i.text for i in reviews]
print(l_reviews)
['4', '2', '4', '72', '239', '179', '607', '13', '517', '631', '82', '2', '506', '138', '1', '1', '6', '71', '1', '195', '11',
'27', '5', '11']

import pandas as pd
df=pd.DataFrame(columns=['Laptops','Price','Reviews'])
print(df)
Output:
Empty DataFrame
Columns: [Laptops, Price, Reviews]
Index: []
```

Step-13

```
df['Laptops']=l_names
df['Price']=l_price
df['Reviews']=l_reviews
```

Output:

```
0   Dell 14 Laptop, AMD Ryzen R5-5500U/ 8GB DDR4, ...    35,990       4
1   Dell Inspiron 3520 Laptop, Intel Core i5-1235U...    55,280       2
2   Dell Inspiron 3535 Laptop, AMD 7 Series Ryzen ...    38,990       4
3   Dell 14 Laptop, 12th Gen Intel Core i5-1235U P...    49,990      72
4   Dell 14 Laptop, 12th Gen Intel Core i3-1215U P...    35,990     239
5   Dell Inspiron 7430 2in1 Touch Laptop, 13th Gen...    57,990     179
6   Dell 15 Laptop, Intel Core i5-1135G7 Processor...    44,990     607
7   Dell Inspiron 3530 Laptop, 13th Gen Intel Core...    67,490      13
8   Dell G15 5520 Gaming Laptop, Intel i5-12500H/1...    75,990     517
9   Dell 15 Laptop, Intel Core i3-1115G4 Processor...    33,990     631
10  Dell Inspiron 5430 Laptop, 13th Gen Intel Core...    83,490      82
11  Dell 14 Laptop, Intel Core i5-1135G7 Processor...    44,990       2
12  (Renewed) DELL Latitude 5490 Core i5 8th Gen L...    23,649     506
13  Dell 14, Intel 12th Gen i5-1235U Laptop/8GB/51...    46,990     138
14  Dell New 14" Latitude 3420- i3 11th Gen || 8 G...    30,630       1
15  Dell Inspiron 3530 Laptop, Intel Core i7-1355U...    71,490       1
16  Dell Inspiron 5330 Laptop, Intel Evo Platform ...    98,990       6
17  Dell New G15-5515 Gaming Laptop, AMD Ryzen5-56...    78,490      71
18  Dell 14 Premium Metal Laptop, AMD Ryzen 5-5625...    50,490       1
19  Dell Inspiron 3520 Laptop, Intel Core i3-1215U...    34,380     195
20  Dell G15-5530 Gaming Laptop, Intel Core i5-134...    81,490      11
21  Dell Inspiron 7430 2in1 Touch Laptop, Intel Co...    82,990      27
22  Dell G15-5525 Gaming Laptop, AMD Ryzen 7-6800H...  1,08,599       5
23  Dell Inspiron 5320 Laptop, Intel Core i5-1240P...    75,490      11
```

```
df.to_csv("Dell_data.csv")
```

Application Program Interface:

Case study:Flight booking

Consider Ola app:
It has

- API from Login page
- API from Location
- API from Payment
- And so on

- All these small apps are connected together into a large app
- These small apps are called APIs and they are building blocks of so many Bigger apps,hence reusable.
- When 10000+ API's are used then it will crash.
- That's why API keys are created

One website dealing with one database is dealing with protocols,connection is straightforward'
One website dealing with different database belonging to different companies requires different API's

Public API—>accessible for everyone
Private API—>accessible for only subscribers.

Random Fox APIs

```
import requests
print(page.status_code)
```

Output:

```
200
```

```
print(page.text)
```

Output:

{"image":"https:\/\/randomfox.ca\/images\/79.jpg","link":"https:\/\/randomfox.ca\/?i=79"}

Project:

Extracting data from coin market cap API

Program Flow:

1. Go to the coin market cap website.
2. Go to Products→API and obtain API key.
3. Go to documentation to obtain API code.
4. Know your code.
5. Run it in any python environment.
6. Obtain data in the form of json.
7. Normalise the data into a dataframe.
8. Save it if you need it

Step 5:

```
#This example uses Python 2.7 and the python-request library.
```

```
from requests import Request, Session
from requests.exceptions import ConnectionError, Timeout,
TooManyRedirects
import json

url =
'https://sandbox-api.coinmarketcap.com/v1/cryptocurrency/listings
/latest'
parameters = {
  'start':'1',
  'limit':'5000',
  'convert':'USD'
}
headers = {
  'Accepts': 'application/json',
  'X-CMC_PRO_API_KEY': 'f7105291-a37f-486b-9d89-bde5fa563985',
}

session = Session()
session.headers.update(headers)

try:
  response = session.get(url, params=parameters)
  data = json.loads(response.text)
  print(data)
except (ConnectionError, Timeout, TooManyRedirects) as e:
  print(e)
```

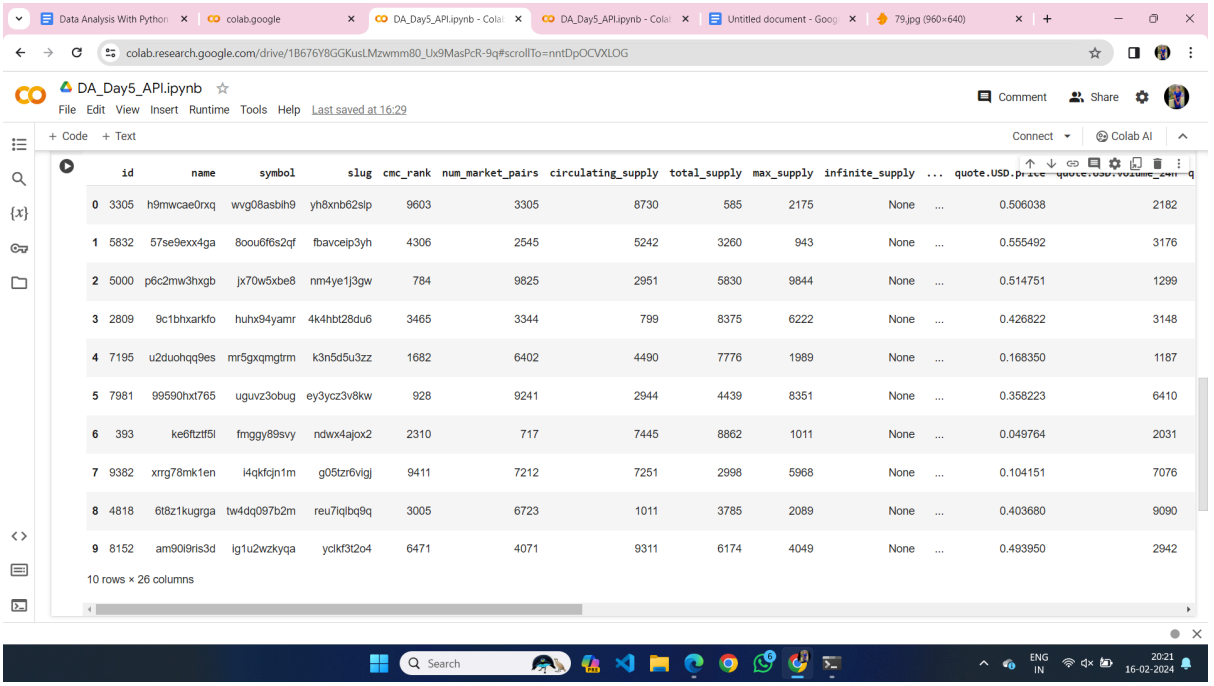
Output:

```
 'status': {'timestamp': '2024-02-16T09:46:47.898Z',
'error_code': 0, 'error_message': None, 'elapsed': 1,
'credit_count': 1, 'notice': None}, 'data': [{'id': 3305, 'name':
'h9mwcae0rxq'
```

Step 6:

```
import pandas as pd
norm=pd.json_normalize(data['data'])
norm
```

Output:



Machine learning:

- Like humans learn from their past experiences,machines learn from past data.
- Training a machine on various things is called machine learning.

Case study:

A machine will suggest the songs by liked songs the high ratted song will be played to the listener

Types of Machine Learning:

- Supervised
- Unsupervised
- Reinforcement

Supervised learning:

Training the machine with some labelled data.

Example;

- Assuming 3 types of coins(1 rupee,1,euro,1 dirham)
- Takes a feature for every label.(features=weight,currency=Label)
- It takes inputs as a label gives output as any one of the labels.
- When you provide new coin to the machine it will look for the feature value
- If the feature value matches , and gives the label as output

Supervised Algorithm:

1. Linear regression
2. Logistic regression
3. Decision Tree
4. Random forest

Linear regression:

- It is also a type of machine learning -algorithm.
- Supervised machine learning algorithm.
- Learns from the labelled datasets and maps the data points to the most optimised linear functions.
- These points can be used for prediction on new datasets.

Dependent and Independent Variable:

- Independent:
  The independent Variable is the cause.Its value is independent of other variables in your study
- Dependent:
  The dependent variable is the effect .Its value depends on changes in the independent variable.

Case Study:

Consider measurements of a chemical reaction.
The mass of the product increases with time.
The observations are:

Time(m)	5	7	12	16	20
Mass(gms)	40	120	180	210	240

Time-Independent
Mass-Dependent

Find the mean of dependent and independent Variable

Regression line is y=a+bx

Regression Calculations:

Code:

```
from sklearn.linear_model import LinearRegression
LR=LinearRegression()
```

Code:

```
t=[[5],[7],[12],[16],[20]]# time -Independent-2D
m=[40,120,180,210,240]#mass-Dependent
LR.fit(t,m)
```

Output:

```
LinearRegression
LinearRegression()
```

Code:

```
LR.predict([[5.5]])
```

Output:

array([78.64935065])

Logistic Regression

- It is used for binary classification.
- Supervised learning Algorithm.

Binary Classification

Example Segregating Food:
Y=0, Class A-Healthy food
Y=1, Class B -Unhealthy food

Case Study:

- ☐ Let us take data from Football match.
- ☐ Based on the distance between the player and the goal post,we are going to predict whether it is a goal or not.!
- ☐ Let us plot some trails now.
- When distance =2m,goal is scored,Y=1
- When distance =4m,goal is scored ,Y=1
- For 5,7,10,20,22m,it is always a goal, Y=1
- When distance=23m,for 15 trails
  few are goals Y=1, few are failures Y=0

Sigmoid Function:

- Sigmoid function is a s-shaped function with peak at 1 and 0 at valley.
- When model is very confident, Narrow decision boarder
- When model is not confident ,Wide decision border

After plotting the distance vs goal between 20 and 30 metres the probability of goal gradually reduce from 1 to 0
But Logistic Regression accepts only two classes
So a threshold variable(0.5) is set.

Prediction is:

- P>0.5 , It's a **goal!** -Class A(Y=1)
- P=< 0.5 , It's a **Miss!** -Class B(Y=0)

Code:

```
import numpy as np
```

```
from sklearn.linear_model import LogisticRegression

# Distance and corresponding probability data
distances = np.array([1,2,5,10,15,20, 21, 22, 23, 24, 25, 26, 27, 28,
29, 30,35,40,41,47,50]).reshape(-1, 1)
probabilities = np.array([1,1,1,1,1,1,0.9, 0.85, 0.73, 0.67, 0.5, 0.47,
0.39, 0.31, 0.25, 0.15,0,0,0,0,0])
#convert probabilities to binary labels
threshold=0.5
binary_labels=(probabilities >threshold).astype(int)
#create and fit logistic regression model
logr=LogisticRegression()
logr.fit(distances,binary_labels)
```

Output:

```
LogisticRegression()
```

Code:

```
p=logr.predict([[10]])#distance
p
```

Output:
array([1])

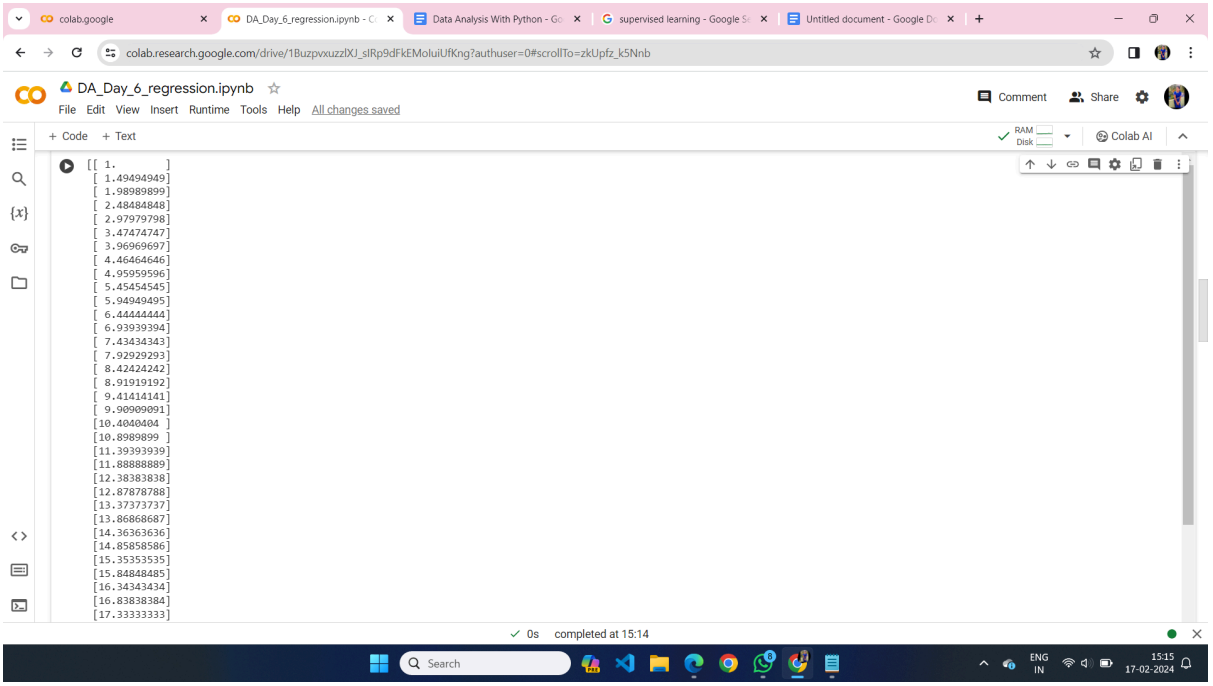
Code:

```
if p==[1]:
    print("Goal")
  else:
    print("Not Goal")
```

Output:
Goal

```
#predict 100 distances between 1 and 50
#Generate distances for prediction
dist=np.linspace(1,50,100).reshape(-1,1)
Print(dist)
#Make predictions using the model
prob=logr.predict_proba(dist)[:,1]
print(prob)
```

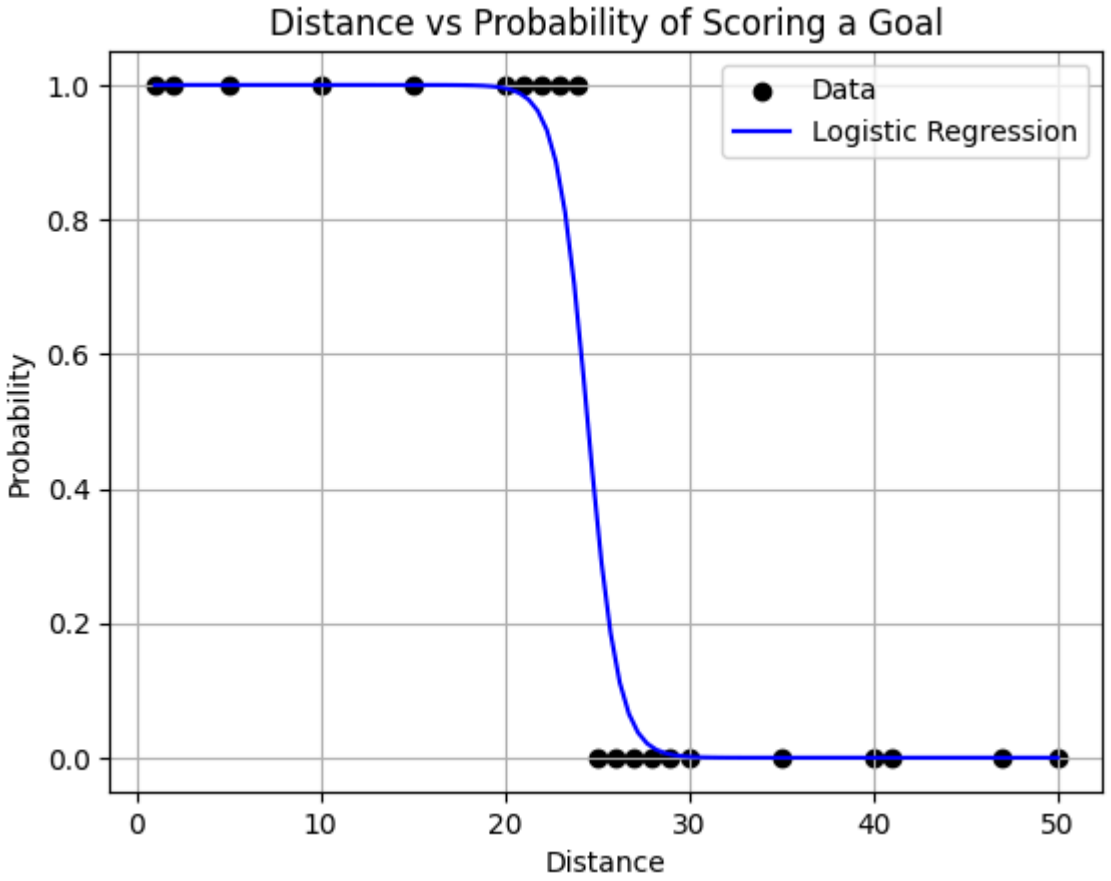
Output:



Code:

```
import matplotlib.pyplot as plt
#plotting actual data -train
plt.scatter(distances,binary_labels,color='black',label='Data')
#plotting test data with predictions -valid/test
plt.plot(dist,prob,color='blue',label='Logistic Regression')
plt.title('Distance vs Probability of Scoring a Goal')
plt.xlabel('Distance')
plt.ylabel('Probability')
plt.legend()
plt.grid(True)
plt.show()
```

Output:



Case Study:

- Taking a dataset of 26 states with features like literacy,Cleanliness,Crime Rate and targeting(predicting) Good or Bad state!
- Good is called Target variable here,it has values of 0s and 1s
- View**:**

Case Study:

- Taking a dataset of 26 states with features like literacy,Cleanliness,Crime Rate and targeting(predicting) Good or Bad state!
- Good is called Target variable here,it has values of 0s and 1s
- View**:**
-

state	**literacy**	**cleanliness**	**Crime Rate**	**Good**
A	92	90	54	0
B	56	67	50	1
C	78	85	62	0

D	63	72	48	1
E	85	79	55	0

Now let us create a Decision Tree

- Decision tree recurrently (continuously)splits the data until it gets pure leaves
- Let us view a DT based on crime Rate

Node	**Good**
Crime Rate <60	0
Crime rate <60	Cannot be determined
Crime Rate >50	1

Predict :

- Crime rate=63,good=0
- Crime rate =45 good =1

***Building Decision tree ***

- Node 1-: CR >60
- [True=c,q,x=0](pure leaf)
- False=[A,E,F,G,I,K,L,M,P,R,U,V=0],[B,D,H,I,M,O,S,T,W,Y,Z=1]
- Mixed leaf has target variable with both 1's and 0's.Hence the data is splitted once again.
- Node-2:CR>50
- True=[A,E,F,G,I,K,L,N,P,R,U,V]=0(Pure leaf)
- False=[B,D,H,J,M,O,S,T,W,Y,Z]=1(Pure leaf)

Random Forest:

Collection of decision tree
Keywords:bootstrapping,aggregation

Case Study:

Why we need RFs and Dts
Observe below data
Y-target
x0-x4–Features

Bootstrapping:
Splitting the main data set into a child data set having the same number of rows and should have different row combinations.

Convolutional Neural Networking:

- A convolutional NeuralNetwork (CNN) is a type of Deep Learning neural network architecture commonly used in Computer Vision
- It consists of Three layers

1. **Input Layers:**
   It's the layer in which we give input to our model.In CNN generally the input will be a image this layer holds the raw data
2. **Hidden layers:**

→Convolutional Layer:

- This is the layer ,which is used to extract features from the input dataset .It applies a set of learnable filters known as the kernel to input images.The filters/kernel are smaller matrices usually 2x2, 3x3and 5x5 shape, it slides over the input image data and compute the dot product between kernel weight and the corresponding input image patch.
- The output of this layer is referred to as featured maps.Suppose we use a total of 12 filters for this layer we will get an output volume of dimensions 32x32x12.

→Activation Layer :

- By adding an activation function to the output of the preceding layer,activation layers add non linearly to the network.It i will apply an element -wise activation function to the output to the convolutional
- The volume remains unchanged hence output volume will have dimensions 32*332*12.

→Pooling Layer:

- This layer is periodically inserted in the converts and its main function is to reduce the size of volume which makes the computation fast reduces memory and also
- If we use a max pool with 2x2 filters and stride 2,the resultant volume will be of dimension 16x16x12.

3. **Output layer:**
   The output from the fully connected layers is then fed into a logistic .
   classification tasks like sigmoid or softmax which converts the output

Activation Function:

- The activation function decides whether a neuron should be activated or not by calculating the weighted sum and further adding bias to it.The purpose of the activation function is to introduce non linearity into the output of a neuron.
- We know the neural network has neurons that work in correspondence with weight bias and their respective activation function. In a neural

network we would update the weights and biases of the neurons on the bias of the error at the output. This process is known as back propagation. Activation functions make the back propagation possible since the gradients are supplied along with the error to update the weights and biases.

- **Tanh :**
  =>Tanh Function
  1. Valin Range-1 to 0.
  2.

- **Sigmoid :**
  =>Sigmoid Function
  1. It's a function which is plotted as a ‘s’ shaped graph.
  2. Equation A=1/(1+ex)
  3. Nature: Non-linear that X values lie between -2 to 2, Y values are very steep. This means small changes in x would also bring about large changes in the value Y.
  4. Value Range: 0 to 1.

- **Relu :**
  1. It stands for Rectified linear unit.It is the most widely used activation function.Chiefly implemented in hidden layers of Neural network.
  2. Equation:-A(x)=max(0,x).It gives an output x if x is positive and 0 otherwise.
  3. Value range:-[0,inf)
  4. Nature:-Nonlinear which means we can easily backpropagate the errors and have multiple layers of neurons being activated by the Relu function.
  5. Uses:-Relu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations.At a time only a few neurons are activated making the network sparse making it efficient and easy for computation.
  6. In other words,RELU learns much faster than sigmoid and Tanh functions.
- **Softmax :**
  1. The softmax function is also a type of sigmoid function but id handy where we are trying to handle multi class classification problems.
  2. Nature: non-linear
  3. Uses:- The softmax function was commonly found in the output layer of the image classification problems.the softmax

function would squeeze the outputs for each class between 0 and 1 and would also dividenby the sum of the outputs.

4. Output:

The softmax function is ideally used in the output layer of the classifier where we are actually trying to attain the probabilities to define the class of each input.

If your output is for binary classification then,sigmoid function is a very natural choice for the output layer.

If your output is multi-class classification then ,Softmax is very useful to predict the probabilities of each .

BRAIN TUMOUR:

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.preprocessing.image import
ImageDataGenerator
#Define image size and batch size
IMG_SIZE=224
BATCH_SIZE=32


from google.colab import drive
drive.mount('/content/drive')
```

Output:

```
Mounted at /content/drive
```

```
#define data generators for train,validation and test sets
train_datagen=ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2
    )


train_generator=train_datagen.flow_from_directory(

'/content/drive/MyDrive/Brain_Tumor_Detection/Brain_Tumor_Detection/tra
in',
    target_size=(IMG_SIZE,IMG_SIZE),
    batch_size=BATCH_SIZE,
    class_mode='binary',
    subset='training'
)
Val_generator=train_datagen.flow_from_directory(
```

```
'/content/drive/MyDrive/Brain_Tumor_Detection/Brain_Tumor_Detection/train',
    target_size=(IMG_SIZE,IMG_SIZE),
    batch_size=BATCH_SIZE,
    class_mode='binary',
    subset='validation'
)
test_datagen=ImageDataGenerator(rescale=1./225)
test_generator=test_datagen.flow_from_directory(

"/content/drive/MyDrive/Brain_Tumor_Detection/Brain_Tumor_Detection/test",
    target_size=(IMG_SIZE,IMG_SIZE),
    batch_size=BATCH_SIZE,
    class_mode='binary'
)
```

Output:

```
Found 2408 images belonging to 2 classes.
Found 602 images belonging to 2 classes.
Found 60 images belonging to 1 classes.
```

```
#Define the model
model=keras.Sequential([

layers.Conv2D(32,(3,3),activation='relu',input_shape=(IMG_SIZE,IMG_SIZE,3)),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64,(3,3),activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(128,(3,3),activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Flatten(),
    layers.Dense(128,activation='relu'),
    layers.Dense(1,activation='sigmoid')
])
#compile the model
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
history=model.fit(train_generator,validation_data=Val_generator,epochs=5)
```

Output:

```
Epoch 1/5
76/76 [==============================] - 514s 7s/step - loss: 0.6041 - accuracy: 0.7367 - val_loss: 0.4049 - val_accuracy: 0.8439
```

```
Epoch 2/5
76/76 [==============================] - 283s 4s/step - loss: 0.3096 -
accuracy: 0.8738 - val_loss: 0.2158 - val_accuracy: 0.9186
Epoch 3/5
76/76 [==============================] - 325s 4s/step - loss: 0.1722 -
accuracy: 0.9369 - val_loss: 0.0963 - val_accuracy: 0.9618
Epoch 4/5
76/76 [==============================] - 304s 4s/step - loss: 0.0938 -
accuracy: 0.9697 - val_loss: 0.0495 - val_accuracy: 0.9867
Epoch 5/5
76/76 [==============================] - 316s 4s/step - loss: 0.0512 -
accuracy: 0.9859 - val_loss: 0.0244 - val_accuracy: 0.9917
```

```
model.save("Model.h5","label.txt")
```

Output

```
usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:310
3: UserWarning: You are saving your model as an HDF5 file via
`model.save()`. This file format is considered legacy. We recommend
using instead the native Keras format, e.g.
`model.save('my_model.keras')`.
  saving_api.save_model(
```

```
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import numpy as np

#load the saved model
model=load_model('/content/Model.h5')

#load and preprocess the test image
test_image_path='/content/drive/MyDrive/Brain_Tumor_Detection/Brain_Tum
or_Detection/train/yes/y1206.jpg +'
img=image.load_img(test_image_path,target_size=(224,224))
img_array=image.img_to_array(img)
img_array=np.expand_dims(img_array ,axis=0)  #batch dimension
img_array /=255.  #normalise the pixel values
#make predictions
prediction=model.predict(img_array)
#print the prediction
if prediction < 0.5:
  print("Prediction: No tumor(Probability:",prediction[0][0],")")
else:
  print("Prediction: Tumor present(Probability:",prediction[0][0],")")
```

Output:

```
1/1 [==============================] - 0s 172ms/step
Prediction: Tumour present(Probability: 0.9994598 )
```