DATA ANALYSIS WITH PYTHON

1.Concept overview-python

1.1 variable

Containers to store value

Code

A=5

1.2 print

It is a function which used to display

Code:

#variable and print

```
a=18

print("my age is",a)

OUTPUT:
```

my age is 19

1.3 Operators

```
#arthematic

a=5

b=20

  print(a+b)

OUTPUT:

     5

#power

print(2**6)

OUTPUT:
```

```
            64
      #float division
            print(12//2)
            OUTPUT:
            6
      #relational
      a=2
      b=2
      print(a==b)
      print(a<b)
      print(a>=b)
      print(a<=b)
      print(a!=b)
      OUTPUT:
      TRUE
      FALSE
      TRUE
      TRUE
      FALSE
#logical
      a=6
      b=26
      print(a>b) and (a<b)
      OUTPUT:
      FALSE
```

```
** print((a>b) or (a<b))

OUTPUT:

FALSE

#membership operators

a="pirate"

print("a" not in a)

print("a" in a)

OUTPUT:

FALSE

TRUE
```

1.4 Control flow-Conditional statements

```
#write a program to get a number from the user and check whether
it is positive or negative

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

  print(type(a))

if(a>0):

  print("positive")

elif(a==0):

  print("neutral")

else:

  print("negative")
```

OUTPUT:

Enter a number:0

<class ‘int’>

Neutral

Enter a number:1

Positive

Enter a number:-2

negative

1.5 Control flow-Looping statements

```
n=6

for i in range(1,11,1):

  print(n,"*",i,"=",n*i)
```

OUTPUT:

6*1=6

6*2=12

6*3=18

6*4=24

6*5=30

6*6=36

6*7=42

6*8=48

6*9=54

6*10=60

```
#table using while loop
```

```
n=int(input("enter a value"))

i=1

while(i<=10):

  print(n,"*",i,"=",n*i)

  i=i+1
```

OUTPUT:

11*1=11

11*2=22

11*3=33

11*4=44

11*5=55

11*6=66

11*7=77

11*8=88

11*9=99

11*10=110

1.6 Data slicing

```
a="python is easy"

print(a[0:6])
```

OUTPUT:

Python

**a="python is easy"

```
print(a[::2])
```

OUTPUT:

Pto ses

1.7Type casting

```
#implicit

a=25.5

b=6

print(a*b)#interpreter automatically prints the values in the float
datatype
```

OUTPUT:

153.0

```
#explicit

a=25.5

b=6

print(int(a*b))#user conversion

OUTPUT:

153
```

1.8 Collection-list

collections list:is a collection of elements heterogeneous mutable

---

Double-click (or enter) to edit

```
list=[1,"sada",6.0]

print(list)
```

OUTPUT:

[1, ‘sada’,6.0]

```
for i in list:

  print(i)
```

OUTPUT:

1

Sada

6.0

#append

```
list.append(2)

print(list)
```

OUTPUT:

[1,’sada’,6.0,2]

```
#insert

list.insert(2,"hi")

print(list)
```

OUTPUT:

[1,’sada’ ‘hi’ ,6.0 ,2]

```
#extend

l1=[1,2,3,4,5]

l2=[6,7,8,9,10]

l1.extend(l2)

print(l1)

OUTPUT:

[1,2,3,4,5,6,7,8,9,10]

#REMOVE
```

```
l1.remove(5)

print(l1)



OUTPUT:

[1,2,3,4,6,7,8,9,10]

#pop

l1.pop(1)

print(l1)

OUTPUT:

[1,3,4,5,6,7,8,9,10]
```

LIST COMPREHENSION:

list comprehension -iterators -applies some functionality on every element -conditions -output

Code
Text

```
l1=[45,67,89,90]

l2=[i**2 for i in l1]#output iterator condition

print(l2)

OUTPUT:

[2025,4489, 7921,8100]

**

l2=[i**2 for i in l1 if i>50]

print(l2)

OUTPUT:

[4489,7921,8100]
```

EXAMPLE:

The salaries of an employee is in the company is taken as a list the tax is 10 percent is less than 50k or it is 15%

create a new list with tax amounts

[67k,45k,89k,34k,50k]

list_name=[(body of if) condition else (body of else) iterator]

[ ]

```
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]

**

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

NUMPY

Arrays: An array is a special variable, which can hold more than one value at a time. If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this: car1 = "Ford" car2 = "Volvo" car3 = "BMW"

```
 #importing

import numpy as np
```

ONE DIMENSIONAL ARRAY: The most basic type of array is a one dimensional array, in which each element is stored linearly and may be retrieved individually by providing its index value. A collection of elements with the same data type that are kept in a linear arrangement under a single variable name is referred to as a one dimensional array.

EXAMPLE:

```
#creating 1D array

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

print(type(a))
```

OUTPUT:

Class 'numpy.ndarray'

TWO DIMENSIONAL ARRAY: In Python, a 2D array is essentially a list of lists. The outer list represents the rows, and each inner list represents a row of elements, similar to how a row works in a matrix.

EXAMPLE:

```
#creating 2D array

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

```
print(b)
```

OUTPUT:

```
[[2 3 4]
 [7 8 9]
```

THREE DIMENSIONAL ARRAY:3-D arrays are referred to as multi-dimensional arrays. Multi-dimensional arrays are defined as an “array of arrays” that store data in a tabular form. Imagine this, an array list of data elements makes a 1-D (one-dimensional) array.

EXAMPLE:

```
#creating 3D ARRAY

c=np.array([[[2,3,4],[6,7,8]],[[6,9,0],[2,3,1]]])#2 groups:g1)r1:2,3,4 r2:6,7,8 g2)r1:6,9,0 r2:2,3,1

print(c)
```

OUTPUT:

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


[[6 9 0 ]
 [2 3 1 ]]]
```

CHECKING DIMENSIONS:Use ndim attribute available with the NumPy array as numpy_array_name. ndim to get the number of dimensions. Alternatively, we can use the shape attribute to get the size of each dimension and then use len() function for the number of dimensions

EXAMPLE:

```
print(a.ndim)

print(b.ndim)

print(c.ndim)
```

OUTPUT:

1

2

3

ONES:Python numpy. ones() function returns a new array of given shape and data type, where the element's value is set to 1. This function is very similar to numpy zeros() function

EXAMPLE:

```
#ones

D=np.ones((2,3,3))

print(D)
```

OUTPUT:

```
[[[ 1.1.1.]

  [1.1.1.]

  [1.1.1.]]

 [[ 1.1.1.]

  [ 1.1.1.]

  [ 1.1.1.]]
```

2)

```
#ones

e=np.ones((3,2))

print(e)
```

OUTPUT:

```
[[ 1. 1 ]

 [ 1. 1 ]

 [ 1. 1 ]]
```

3)

```
import numpy as np

a=np.ones((8,4))

print(a)
```

OUTPUT:

[[1. 1. 1. 1.]

[1. 1. 1. 1.]

[1. 1. 1. 1.]

[1. 1. 1. 1.]

[1. 1. 1. 1.]

[1. 1. 1. 1.]

[1. 1. 1. 1.]

[1. 1. 1. 1.]]

4)

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

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

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

OUTPUT:

[[2. 2.]

[2. 2.]

[2. 2.]

[2. 2.]]

ZEROS:To create an array of zeros in Python, you can use the np. zeros function from the numpy library. Here's a simple example: import numpy as np arr = np

EXAMPLE:

1)

```
#zeros

f=np.zeros((3,2))

print(f)
```

OUTPUT:

[[0 0]

[0 0]

```
 [ 0 0]]
```

2):

```
j=np.eye(4,3)

print(j)
```

OUTPUT:

```
[[ 1 0 0]

 [ 0 1 0]

 [ 0 0 1]

 [ 0 0 0]]
```

ARRANGE:arange() function in Python is a powerful tool that allows you to create arrays with evenly spaced values. It is a versatile function used in various scenarios, from simple arithmetic to complex mathematical operations. This blog will explore the various applications of numpy

EXAMPLE:

1)

```
#arange

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

print(h)
```

OUTPUT:

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

ARRANGE WITH RESHAPE:

arange() This is particularly suitable when you want to create a plot in Matplotlib. That's how you can obtain the ndarray instance with the elements [0, 1, 2, 3, 4, 5] and reshape it to a two-dimensional array.

EXAMPLE:

1)

```
#arange with reshape

h=np.arange(3,31,3).reshape(5,2)

print(h)
```

OUTPUT:

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

2)#5 table upto 1000

```
i=np.arange(5,1001,5)
print(i)
```

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]
```

3)

```
v=np.linspace(12,24,10)#divides the numbers in b/w 12 to 24 into 10 equal parts

print(v)
```

OUTPUT:

12. 13.33333333 14.66666667 16. 17.33333333 18.66666667

20. 21.33333333 22.66666667 24.]

4)

```
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]]
```

5)

```
print(1+r)
```

OUTPUT:

[[10 11 12]

[13 14 15]]

6)

```
g=np.sum((l,r))
print(g)
```

OUTPUT:

7)

```
g=np.sum((1,r),axis=0)

print(g)
```

OUTPUT:

[[10 11 12]

[13 14 15]]

8)

```
g=np.sum((l,r),axis=1)#rows

print(g)
```

OUTPUT:

[[5 7 9]

[21 23 25]]

9)

```
a=np.array([[1,1],[0,1]])

b=np.array([[2,0],[3,4]])

print(a*b)
```

OUTPUT:

[[2 0]

[0 4]]

10)

```
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]]
```

EXAMPLE:

b=np.array[25,289,361,81] find square roots and iterate through result value output:5 square is 25

```
b=[25,289,361,81][[2 0]

 [0 4]]



for i in b:

  print(np.sqrt(i),"square is",1)
```

OUTPUT:

5.0 square is 1

17.0 square is 1

19.0 square is 1

9.0 square is 1

ARRAY JOINTS:Joining means putting contents of two or more arrays in a single array. In SQL we join tables based on a key

EXAMPLE:

```
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(a)
print(b)
print("\n")
print(np.vstack((a,b)))#coloumns
print("\n")
print(np.hstack((a,b)))
OUTPUT:
```

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

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

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

RANDOM AND RAND:NumPy random. rand() function in Python is used to return random values from a uniform distribution in a specified shape. This function creates an array of the given shape and it fills with random samples from the uniform distribution.

EXAMPLE:

1)

```
A=np.random.rand(1)
print(A)
```

OUTPUT:

```
[0.46584502]
```

2)

```
A=np.random.rand(8,4)#rand range is b/w 0 and 1
print(A)
```

OUTPUT:

```
[[0.09907899 0.1598259  0.71504942 0.34729956]
 [0.84639197 0.14582284 0.68671143 0.42395321]
 [0.45224266 0.99970248 0.04310802 0.58339377]
 [0.542621   0.91284503 0.40066622 0.60989855]
 [0.19722872 0.21018408 0.32868864 0.53519215]
 [0.46477612 0.43323041 0.49110677 0.14027071]
 [0.62703084 0.12187961 0.10166665 0.30419094]
 [0.86395801 0.93728822 0.1887755  0.18869618]]
```

3)

```
a=np.floor (10*np.random.rand(8,4))#rand range is b/w 0 and 1
```

```
print(a)
```

OUTPUT:

```
[[6. 6. 8. 8.]

 [3. 6. 9. 4.]

 [7. 1. 7. 1.]

 [7. 7. 0. 4.]

 [2. 3. 1. 5.]

 [8. 1. 2. 7.]

 [3. 4. 7. 5.]

 [0. 6. 9. 0.]]
```

SPLITTING AN ARRAY:Splitting is reverse operation of Joining.Joining merges multiple arrays into one and Splitting breaks one array into multiple.We use `array_split()` for splitting arrays, we pass it the array we want to split and the number of splits

EXAMPLE:

1)

```
np.split(a,(2,7))#split after 2nd line and again split after 7th line
```

OUTPUT:

[array([[1, 2, 3, 4],

[5, 6, 7, 8]]),

array([[9, 10, 11, 12],

[13, 14, 15, 16],

[17, 18, 19, 20],

[21, 22, 23, 24],

[25, 26, 27, 28]]),

array([[29, 30, 31, 32]])]

VSPLIT:

vsplit. numpy. vsplit is a special case of split() function where axis is 1 indicating a vertical split regardless of the dimension of the input array.

EXAMPLE:

```
np.vsplit(a,4)#split can be done only when the given num is a multiple of number of rows
```

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 QUESTION:

creating an array with four rows and 8 coloumns:

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

print(b)
```

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]]
```

HSPLIT:

hsplit() function split an array into multiple sub-arrays horizontally (column-wise). hsplit is equivalent to split with axis=1, the array is always split along the second axis regardless of the array dimension

EXAMPLES:

1)

```
np.hsplit(b,4)
```

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]])]

2)

```
np.hsplit(b,(2,6))#splits after second and fifth
```

OUTPUT:

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

3)

```
np.hsplit(b,(2,3,6))#splits after 2nd and 3rd and 6th coloumns
```

OUTPUT:

```
[array([[ 1,  2],
        [ 9, 10],
        [17, 18],
        [25, 26]]),
 array([[ 3],
        [11],
        [19],
        [27]]),
 array([[ 4,  5,  6],
        [12, 13, 14],
        [20, 21, 22],
```

```
       [28, 29, 30]]),
 array([[ 7,  8],
       [15, 16],
       [23, 24],
       [31, 32]])]
```

NUMPY PI:

NumPy constants are the predefined fixed values used for mathematical calculations. For example, np. pi is a mathematical constant that returns the value of pi (π), i.e. 3.141592653589793. Using predefined constants makes our code concise and easier to read.

EXAMPLES:

1)

```
np.pi
```

OUTPUT:

3.141592653589793

2)

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

print(A)
```

OUTPUT:

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

RANDOM:

The random is a module present in the NumPy library. This module contains the functions which are used for generating random numbers. This module contains some simple random data generation methods, some permutation and distribution functions, and random generator functions.

DEGREE:

numpy. degrees() and rad2deg() in Python

- **Syntax : numpy.degrees(x[, out]) = ufunc 'degrees') Parameters :**
- **array : [array_like] elements are in radians. out : [ndaaray, optional] Output array of same shape as x. 2pi Radians = 360 degrees.**
- **Return : An array with degree values in place of radian values**

EXAMPLES:

1)

```
b=np.rad2deg(A)#converts radians to degree

print(b)
```

OUTPUT:

[0.7853981633974483, 1.0471975511965976, 1.5707963267948966, 3.141592653589793]

2)

```
b=np.rad2deg(A)#converts radians to degree
print(b)
```

OUTPUT:
[45. 60. 90. 180.]

3)

```
np.deg2rad(b)
```

OUTPUT:
array([0.78539816, 1.04719755, 1.57079633, 3.14159265])

SIN:

sin() in Python. numpy. sin(x[, out]) = ufunc 'sin') : This mathematical function helps user to calculate trigonometric sine for all x(being the array elements).

EXAMPLE:

```
np.sin(1)
```

OUTPUT:

0.8414709848078965

COS:

The numpy. cos() method in Python calculates the ratio of the length of the side nearest to the angle to the length of the hypotenuse. This function returns the cosine of a number. To be more precise, it returns the cosine of a radian.

EXAMPLE:

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

OUTPUT:

0.5403023058681398

TAN:

tan() in Python. numpy. tan(array[, out]) = ufunc 'tan') : This mathematical function helps user to calculate trigonometric tangent for all x(being the array elements).

EXAMPLE:

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

OUTPUT:

1.5574077246549023

MEAN:

NumPy array mean() function in Python is used to compute the arithmetic mean or average of the array elements along with the specified axis or multiple axis

EXAMPLE:

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

np.mean(st)
```

OUTPUT:

46.5

MEDIAN:

numpy. median(arr, axis=None, out=None)

- **arr: array_like This will be our input array to perform median method.**

- **axis: None or int or tuple of ints, optional Axis on which we perform the arithmetic median if specified. ...**
- **out: ndarray(optional) Used for defining an alternative output array in which the result is placed.**

EXAMPLE:

```
np.median(st)
```

OUTPUT:

39.5

SQUARE ROOT:

You can use NumPy to calculate the square root of elements in an array using the numpy. sqrt() function. This function is used to return the non-negative square root of an array element-wise (for each element of the array).

EXAMPLE:

```
np.std(st)#std=sqrt((sigma(xi-x)^2)/n) where xi=individual elementt and x=mean n= no of elements
```

OUTPUT:

24.452334585202017

VARIABLE(VAR):

var() in Python. numpy. var(arr, axis = None) : Compute the variance of the given data (array elements) along the specified axis(if any). This Result is Variance.

EXAMPLE:

1)

```
np.var(st)#var=(std)^2
```

OUTPUT:

597.9166666666666

2)

```
c=np.arange(1,5).reshape(2,2)
print(c)
```

OUTPUT:

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

LINERA ALZEBRA(linalg):

The NumPy linear algebra functions rely on BLAS and LAPACK to provide efficient low level implementations of standard linear algebra algorithms

EXAMPLE:

1)

```
np.linalg.inv(c)#linear alzebra is a subpackage in numpy
```

OUTPUT:

```
array([[-2. ,  1. ],
       [ 1.5, -0.5]])
```

ARGUMENT MAXIMUM NUMBER:

NumPy argmax() in Python is used to return the indices of the max elements of the given array along with the specified axis. Using this function gets the indices of maximum elements of single dimensions and multi-dimensional(row-wise or column-wise) of the given array

EXAMPLE:

1)

```
c=np.arange(1,25).reshape(6,4)
print(np.argmax(c))#argmax will provide u the index of highest number in each row or coloumn in the given matrix
```

OUTPUT:

```
23
```

2)

```
print(np.argmax(c))
```

OUTPUT:

```
23
```

3)

```
d=10*np.random.rand(24).reshape(6,4)
print(d)
```

OUTPUT:

```
[[4.07751998 8.72834794 8.97873355 5.95557456]
 [6.58553538 2.38379297 9.23355193 9.42305738]
 [8.50221714 3.01282204 2.4185026  2.45751048]
 [6.46190302 9.19893241 4.35219561 3.28589104]
 [9.8770008  4.53912116 5.75201058 4.9960637 ]
 [0.1713341  3.35536689 2.15546352 3.97426257]]
```

We use the function argmax() with the parameter axis that can be defined as follows:

- **axis=0 , the index of the max element per column will be returned.**
- **axis=1 the index of the max element per row will be returned**

EXAMPLE:

1)

```
print(np.argmax(d,axis=1))#argmax with axis1 will provide u the index of the highest number in each individual row in the given matrix
```

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

2)

```
print(np.argmax(d,axis=0))#argmax with axis 0 will provide u the index off the highest number in each individual coloumn in the given matrix
```

OUTPUT:

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

ARGUMENT MINIMUM NUMBER(argmin()):

- If axis = None , the array is flattened and the index of the flattened array is returned.
- If axis = 0, the index of the smallest element in each column is returned.
- If axis = 1, the index of the smallest element in each row is returned.

EXAMPLE:

1)

```
#we can usemin as like argmax

print(np.argmin(d,axis=1)) arg
```

OUTPUT:

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

2)

```
Print(np.argmin(d,axis=0))
```

OUTPUT:

```
[5 1 5 2]
```

FLOOR :

floor() function in NumPy is used to return the floor values of each element in an input array x such that for each of the elements of x , the floor values are less than or equal to it.

EXAMPLE:

1)

```
c=np.floor(10*np.random.rand(24)).reshape(6,4)
print(c)
```

OUTPUT:

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

SEARCHING:

numpy.searchsorted(): The function is used to find the indices into a sorted array arr such that, if elements are inserted before the indices,

EXAMPLE:

1)

```
#searching the indices based on the conditions
#finding indixes where the values are even
a=np.array([34,56,17,89,91])
print(np.where(a%2==0))
```

OUTPUT:

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

2)

```
b=np.array([24,16,7,17,54,60])

print(np.where(b%6==0))
```

OUTPUT:

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

SORTED LIST SEARCHING:

Search Sorted

There is a method called searchsorted() which performs a binary search in the array, and returns the index where the specified value would be inserted to maintain the search order. The searchsorted() method is assumed to be used on sorted arrays.

EXAMPLE:

1)

```
#searching is done only for sorted list
```

```
a=np.array([6,7,8,9,10])

x=np.searchsorted(a,10)

print(x)
```

OUTPUT:

4

2)

```
#unsortedlist

b=np.array([34,12,7])

x=np.searchsorted(b,7)

print(x)
```

OUTPUT:

0

SORTING ARRAYS:

The NumPy ndarray object has a function called sort() , that will sort a specified array.

- **ExampleGet your own Python Server. Sort the array: import numpy as np. ...**
- **Sort the array alphabetically: import numpy as np. ...**
- **Sort a boolean array: import numpy as np. ...**
- **Sort a 2-D array: import numpy as np.**

EXAMPLES:

1)

```
arr=np.array(['banana','cherry','apple'])

ar=np.array([True,False,True])#value for false=0 and value for true=1

print(np.sort(arr))

print(np.sort(ar))
```

OUTPUT:

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

```
[False  True  True]
```

2)

```
#2d arrayL:sorting happens within a row

arr=np.array([[3,2,4],[5,0,1]])

print(np.sort(arr))
```

OUTPUT:

```
[[2 3 4]

 [0 1 5]]
```

ARRAY FILTERS:

Getting some elements out of an existing array and creating a new array out of them is called filtering. In NumPy, you filter an array using a boolean index

EXAMPLES:

1)

```
#array filters

arr=np.array([40,42,50,44,67,78])

x=[True,False,True,False,True,True]#filter list

newarr=arr[x]

print(newarr)
```

OUTPUT:

```
[40 50 67 78]
```

2)

```
arr=np.array([40,43,50,44,67,78])

filt=np.where(arr%2==0)

print(filt)
```

OUTPUT:

```
(array([0, 2, 3, 5]),)
```

3)

```
arr[filt]
```

OUTPUT:

array([40, 50, 44, 78])

ITERATING THROUGH 2 ARRAYS:

Suppose the first operand is one dimensional and the second operand is two dimensional. The iterator will have three dimensions, so op_axes will have two 3-element lists. The first list picks out the one axis of the first operand, and is -1 for the rest of the iterator axes, with a final result of [0, -1, -1].

EXAMPLE:

1)

```
#iterating through two arrays

names=np.array([" harsha","sada","rohith","josh","abhi"])

initials=np.array(["b","m","u","v","l"])

for i,j in zip(initials,names):

  print(i,".",j)
```

OUTPUT:

```
b .  harsha

m . sada

u . rohith

v . josh

l . abhi
```

2)

```
a1=np.array([10,20,30,40,50,60])

a2=np.array([20,21,22,23,24,25])

n1=np.multiply(a1,a2)

print(n1)
```

OUTPUT:

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

3)

```
a1=np.array([10,20,30,40,50,60])

a2=np.array([20,21,22,23,24,25])

n1=np.divide(a1,a2)

print(n1)
```

OUTPUT:

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

4)

```
a1=np.array([10,20,30,40,50,60])

a2=np.array([20,21,22,23,24,25])

n1=np.mod(a1,a2)

print(n1)
```

OUTPUT:

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

5)

```
a1=np.array([10,20,30,40,50,60])

a2=np.array([20,21,22,23,24,25])

n1=np.divmod(a1,a2)

print(n1)
```

OUTPUT:

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

LOGORITHMS:

Python's numpy. log() is a mathematical function that computes the natural logarithm of an input array's elements. The natural logarithm is the inverse of the exponential function, such that log (exp(x)) = x.

EXAMPLES:

1)

```
a1=1.2

print(np.log(a1))#natural loge
```

OUTPUT:

```
0.2630344058337938
```

2)

```
a1=1.2

print(np.log10(a1))#log base 10
```

OUTPUT:

```
0.07918124604762482
```

3)

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

print(np.log(a))
```

OUTPUT:

```
[0.         0.69314718 1.09861229 1.38629436 1.60943791 1.79175947

 1.94591015]
```

CUMILATIVE PRODUCT:

- Overview. The numpy. cumprod() function in NumPy computes the cumulative product of the elements in a given input array over a given axis.
- Syntax. numpy.cumprod(a, axis=None, dtype=None, out=None)
- Parameter values. The numpy. ...
- Return value. The numpy. ...

- Code example.

EXAMPLES:

1)

```
#cummulative product

a1=np.array([5,6,7,10])

x=np.cumprod(a1)

print(x)
```

OUTPUT:

[5 30 210 2100]

2)

```
a1=np.array([5,6,7,10])
x=np.cumsum(a1)
print(x)
```

OUTPUT:

```
[ 5 11 18 28]
```

DIFFERENCE:

diff(arr[, n[, axis]]) function is used when we calculate the n-th order discrete difference along the given axis. The first order difference is given by out[i] = arr[i+1] – arr[i] along the given axis. If we have to calculate higher differences, we are using diff recursively.

EXAMPLES:

1)

```
#difference

a2=np.array([10,15,25,15])

new=np.diff(a2)

print(new)
```

OUTPUT:

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

LCM:

lcm() in Python. numpy. lcm(arr1, arr2, out = None, where = True, casting = 'same_kind', order = 'K', dtype = None) : This mathematical function helps user to calculate lcm value of |arr1| and |arr2| elements. Parameters : arr1 / arr2 : [array_like]Input array

EXAMPLE:

1)

```
#lcm

n1=455

n2=665

x=np.lcm(n1,n2)

print(x)
```

OUTPUT:

8645

GCD:

numpy. gcd() returns the greatest common divisor of the absolute values of the input. The return type is either ndarray or scaler , depending on the input type.

EXAMPLES:

1)

```
#gcd

n1=455

n2=665

x=np.gcd(n1,n2)

print(x)
```

OUTPUT:

35

2)

```
#finding gcd for arrays of elements

a=np.array([12,15,60])

gc=np.gcd.reduce(a)#takes multiple inputs and gives single outputs

print(gc)
```

OUTPUT:

3

3)

```
#finding gcd for arrays of elements

a=np.array([12,15,60])

gc=np.gcd.reduce(a)#takes multiple inputs and gives single outputs

print(gc)
```

60

PLOTTING

A line graph is a common way to display the relationship between two dependent datasets. Its general purpose is to show change over time. To plot a line graph from the numpy array, we can use matplotlib which is the oldest and most widely used Python library for plotting. Also, it can be easily integrated with numpy which makes it easy to create line graphs to represent trends and patterns in the given datasets.

MATPLOTLIB:

Matplotlib is a plotting library for Python. It is used along with NumPy to provide an environment that is an effective open source alternative for MatLab. It can also be used with graphics toolkits like PyQt and wxPython.

Matplotlib module was first written by John D. Hunter. Since 2012, Michael Droettboom is the principal developer. Currently, Matplotlib ver. 1.5.1 is the stable version available. The package is available in binary distribution as well as in the source code form on

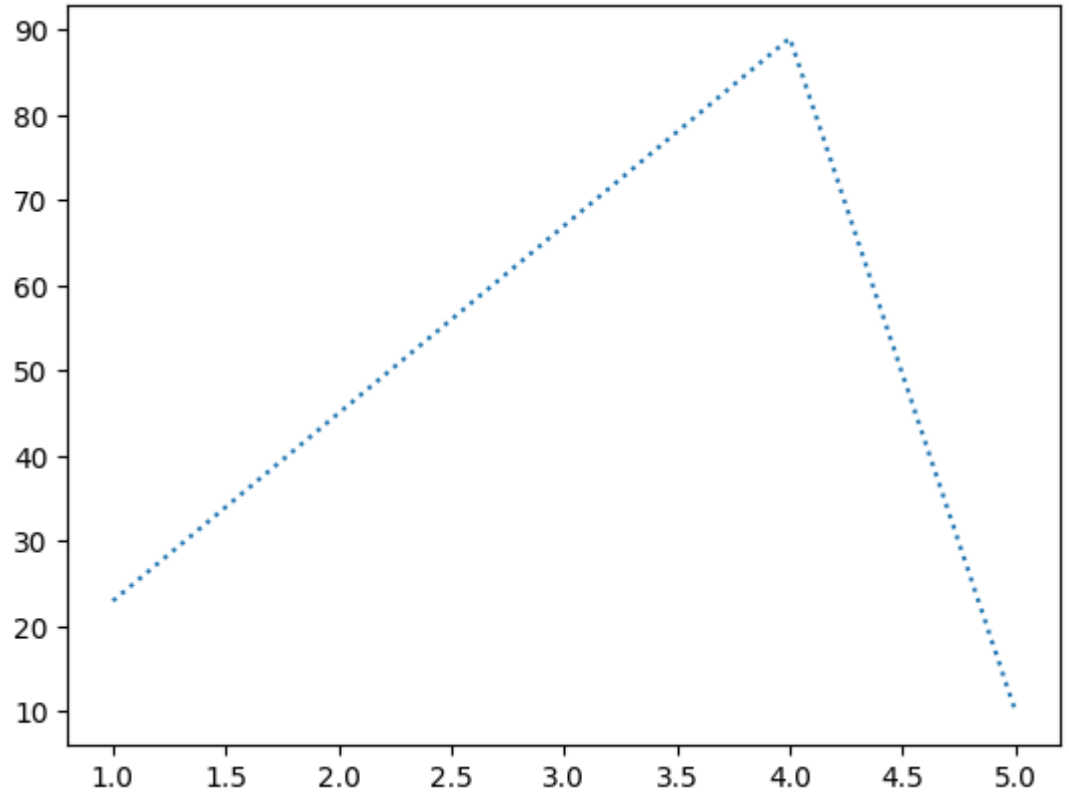
```
import matplotlib.pyplot as plt
import numpy as np
```

EXAMPLE:

1)

```
a=[23,45,67,89,10]
b=[1,2,3,4,5]
plt.plot(b,a,linestyle=":")
plt.show()
```

OUTPUT:



EXAMPLE:

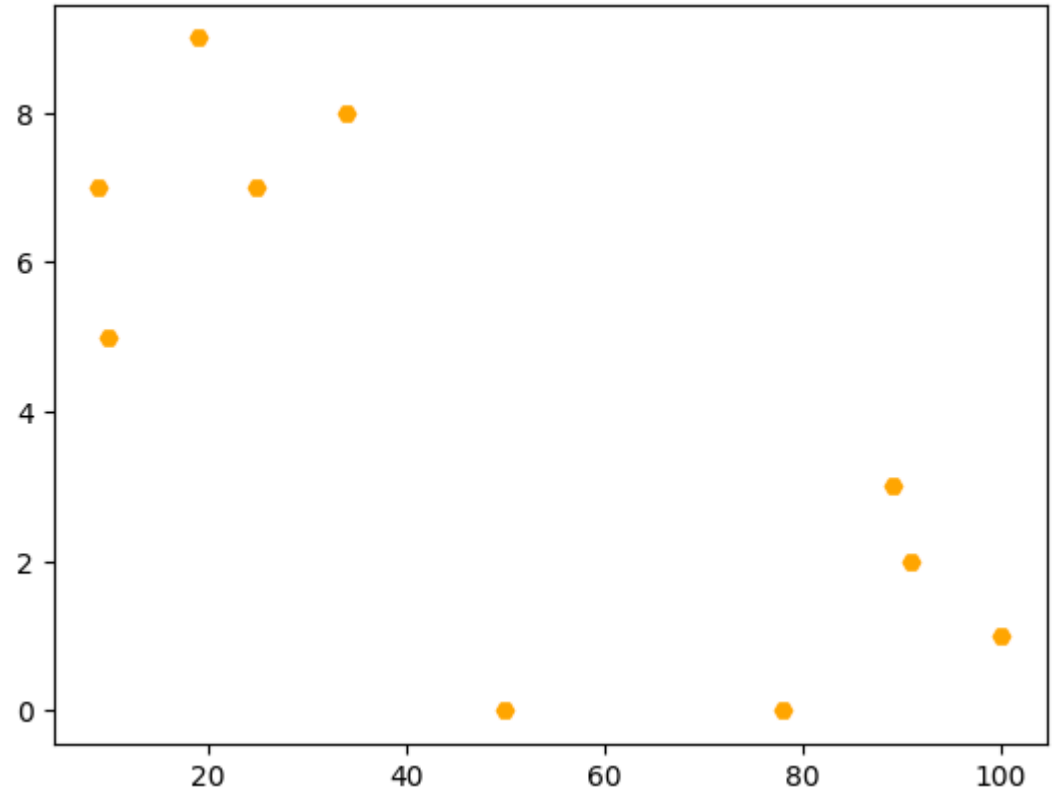
1Q)

Runs scored by 10 new players [100,50,91,78,89,25,34,19,9,10]wickets taken by same 10 new players [1,0,2,0,3,7,8,97,5] from clusters for batsmen and bowlers

*****

```
runs=np.array([100,50,91,78,89,25,34,19,9,10])
wickets=np.array([1,0,2,0,3,7,8,9,7,5])
plt.scatter(runs,wickets,color='orange',marker="H")
plt.show()
```

OUTPUT:



COLOURING INSTRUCTIONS:

colours allowed

*** R-red**

*** g-green**

*** B-blue**

*** C-cyan**

*** M-magenta**

*** Y-yellow**

*** k-black**

*** w-white**

MARKER INSTRUCTIONS:

Marker: *o circle

- **star**

- .point
- ,pixel
- x x
- X x(filled)
    - plus
- p plus(filled)
- s square
- D daimond
- d daimond(thin)
- p pentagon
- H hexagon
- h hexagon
- v triangle down
- ^ triangle up
- < triangle right

EXAMPLE:

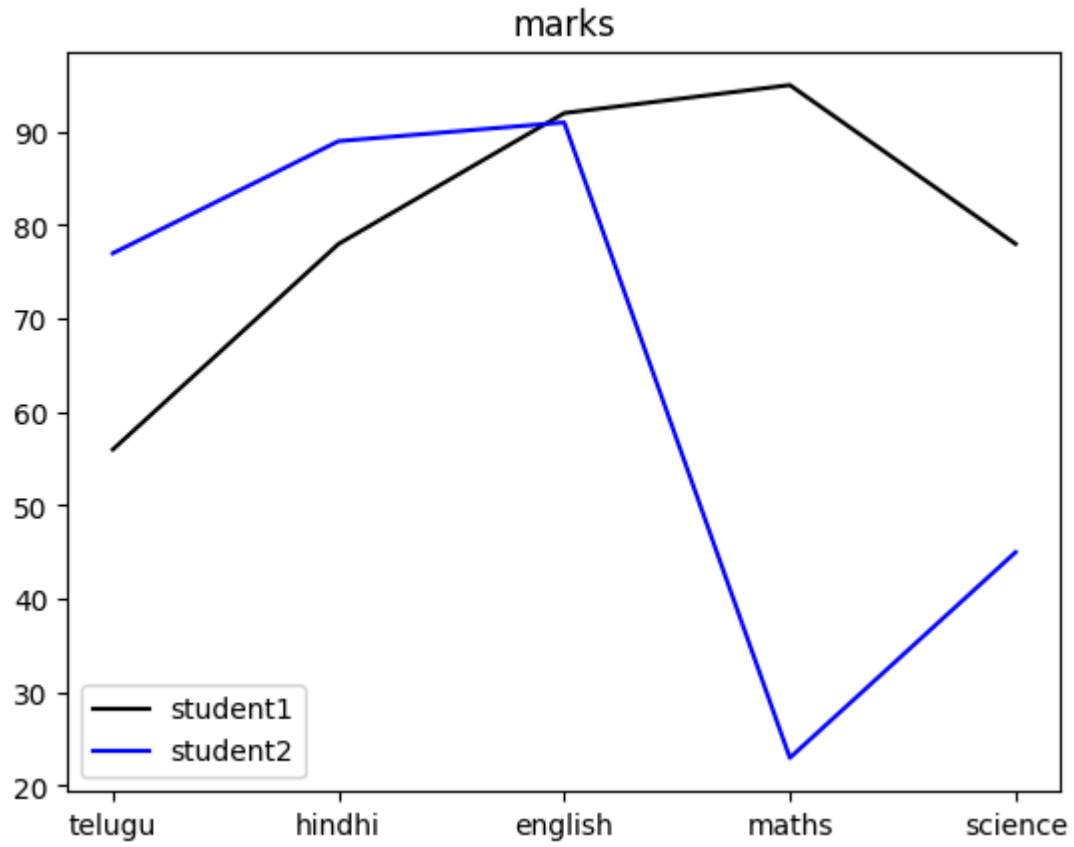
Q2)

score comparision of 2 students in 5 different subjects telugu,hindhi,english,maths,science as x axis and marks as y axis student 1[56,78,92,95,78] student2[77,89,91,23,45]

```
import numpy as np
import matplotlib.pyplot as plt
stu1=[56,78,92,95,78]
stu2=[77,89,91,23,45]
sub=["telugu","hindhi","english","maths","science"]
plt.plot(sub,stu1,label="student1",color='k')
plt.plot(sub,stu2,label="student2",color='b')
plt.title("marks")
plt.legend()
```

```
plt.show()
```

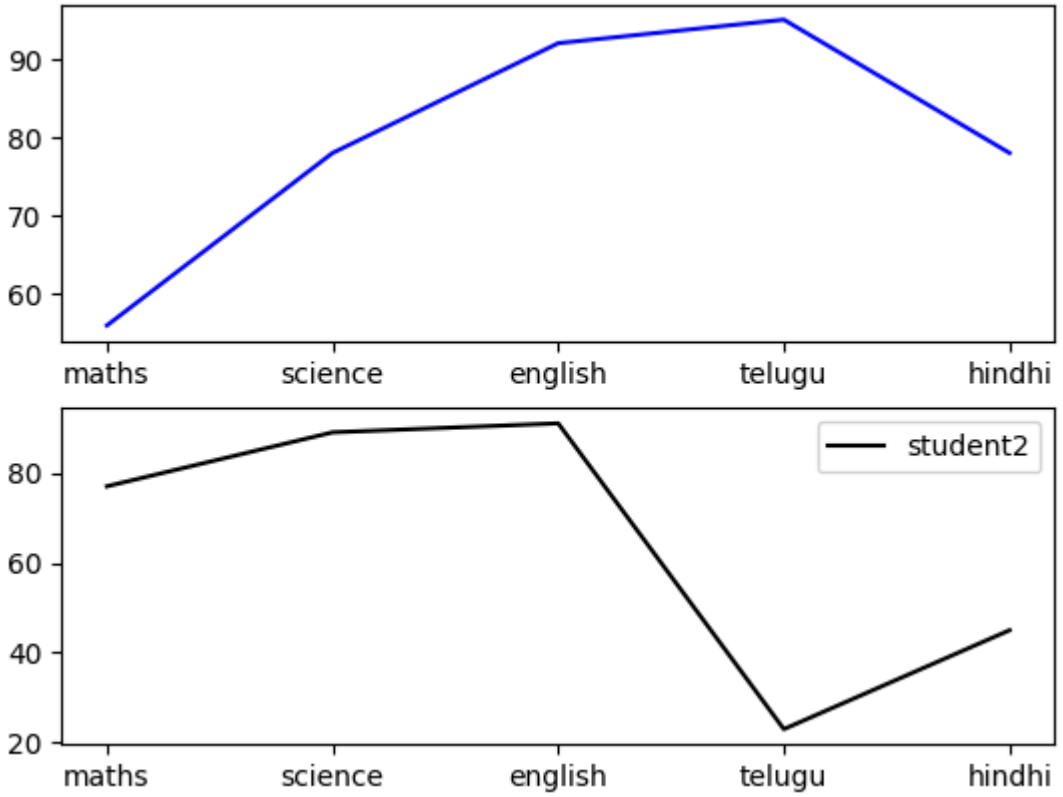
OUTPUT:



EXAMPLE:

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

OUTPUT:



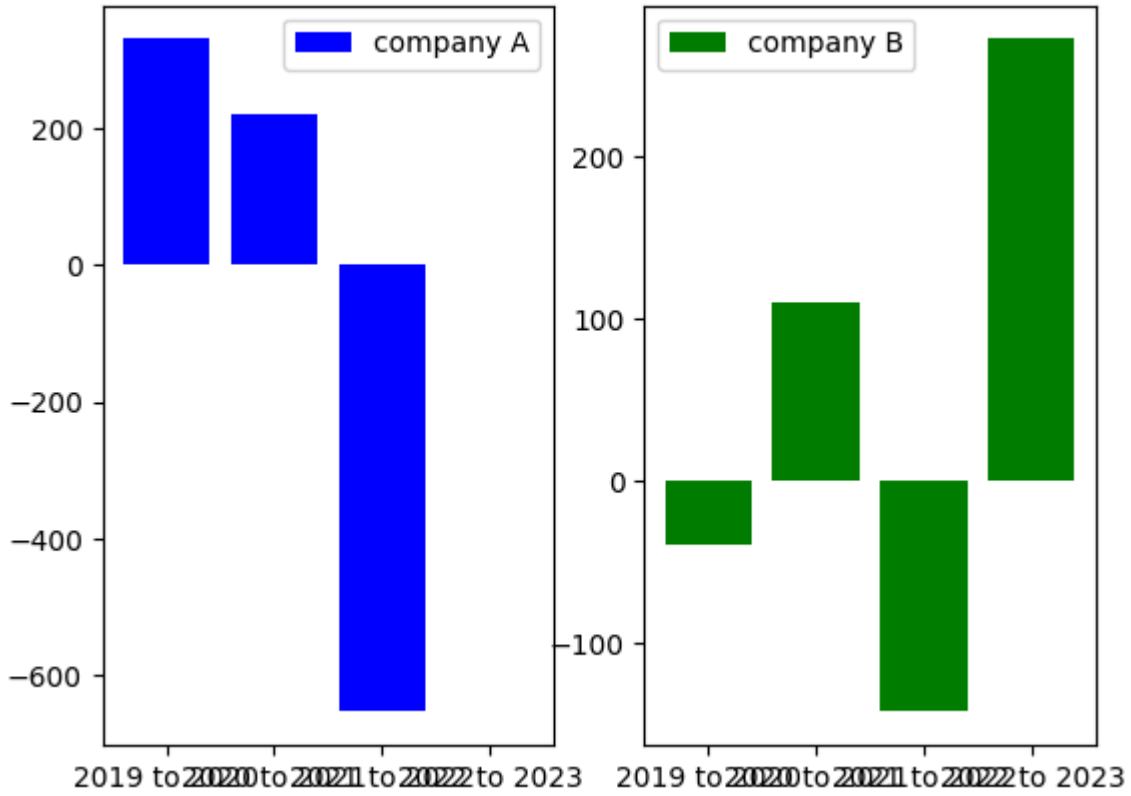
EXAMPLE:

```
year=np.array(["2019 to 2020","2020 to 2021","2021 to 2022","2022 to 2023"])
revenueofa=np.array([230,560,780,127,128])
revenueofb=np.array([200,160,270,127,400])
profita=np.diff(revenueofa)
profitb=np.diff(revenueofb)
print("revenue of company A",revenueofa)
print("revenue of company B",revenueofb)
print("profit of A",profita)
print("profit of B",profitb)
plt.subplot(1,2,1)
plt.bar(year,profita,label="company A",color='b')
plt.legend()
plt.subplot(1,2,2)
plt.bar(year,profitb,label="company B",color='g')
plt.legend()
```

```
plt.show()
```

OUTPUT:

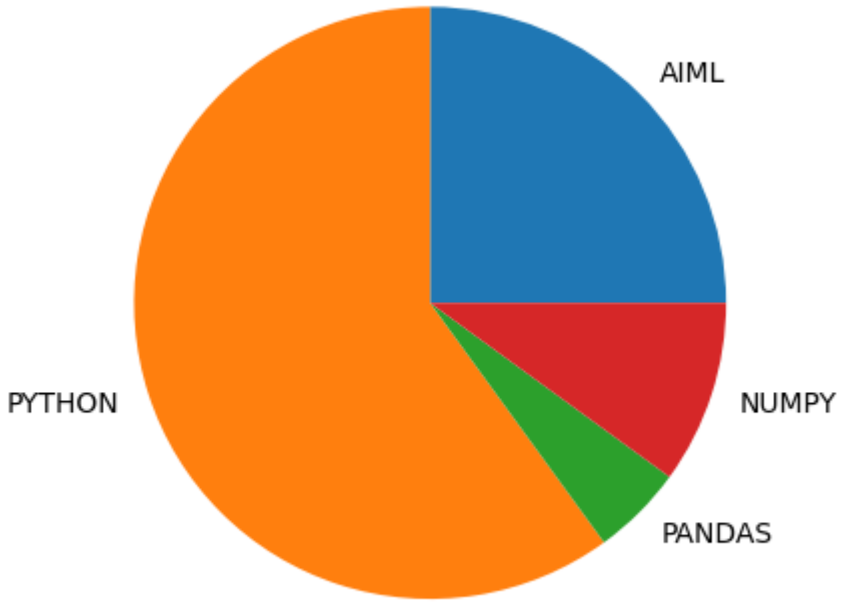
```
revenue of company A [230 560 780 127 128]

revenue of company B [200 160 270 127 400]

profit of A [ 330  220 -653    1]

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



EXAMPLE:

```
a=np.array([25,60,5,10])
labe=["AIML","PYTHON","PANDAS","NUMPY"]
plt.pie(a,labels=labe)
plt.show()
```

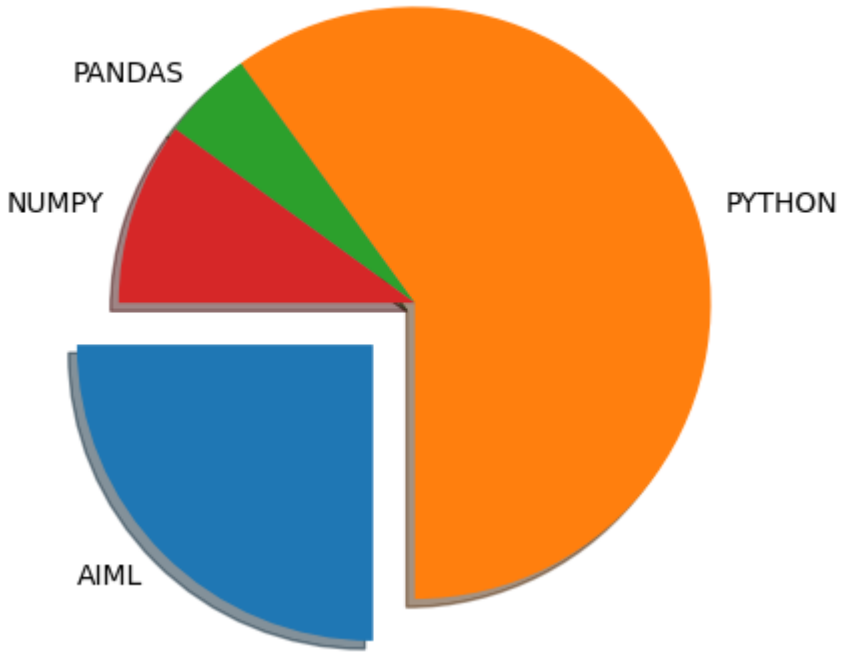
OUTPUT:



EXAMPLE:

```
a=np.array([25,60,5,10])
lab=["AIML","PYTHON","PANDAS","NUMPY"]
explo=[0.2,0,0,0]
plt.pie(a,labels=labe,explode=explo,startangle=180,shadow=True)
plt.show()
```

OUTPUT:



PANDAS

INTRODUCTION:

Pandas is a Python library used for working with data sets.

It has functions for analyzing, cleaning, exploring, and manipulating data.

The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

USES OF PANDAS:

Pandas allows us to analyze big data and make conclusions based on statistical theories.

Pandas can clean messy data sets, and make them readable and relevant.

Relevant data is very important in data science.

WHAT CAN PANDAS DO:

Pandas gives you answers about the data. Like:

- **Is there a correlation between two or more columns?**
- **What is average value?**
- **Max value?**
- **Min value?**

```
import numpy as np
#pandas is a subpackage in numpy
import pandas as pd
```

EXAMPLE:

1)

```
names=["harsha","sadhaf","rohith","lucky","loki"]
index=[48,42,42,43,45]
ser1=pd.Series(names,index)
print(ser1)
```

OUTPUT:

```
48    harsha
42    sadhaf
42    rohith
43     lucky
45      lokI
```

IMPORTING FILES:

Before reading a CSV file into a **pandas dataframe**, you should have some insight into what the data contains. Thus, it's recommended you skim the file before attempting to load it into memory: this will give you more insight into what columns are required and which ones can be discarded.

SYNTAX:

for csv and txt:read_csv("file path")

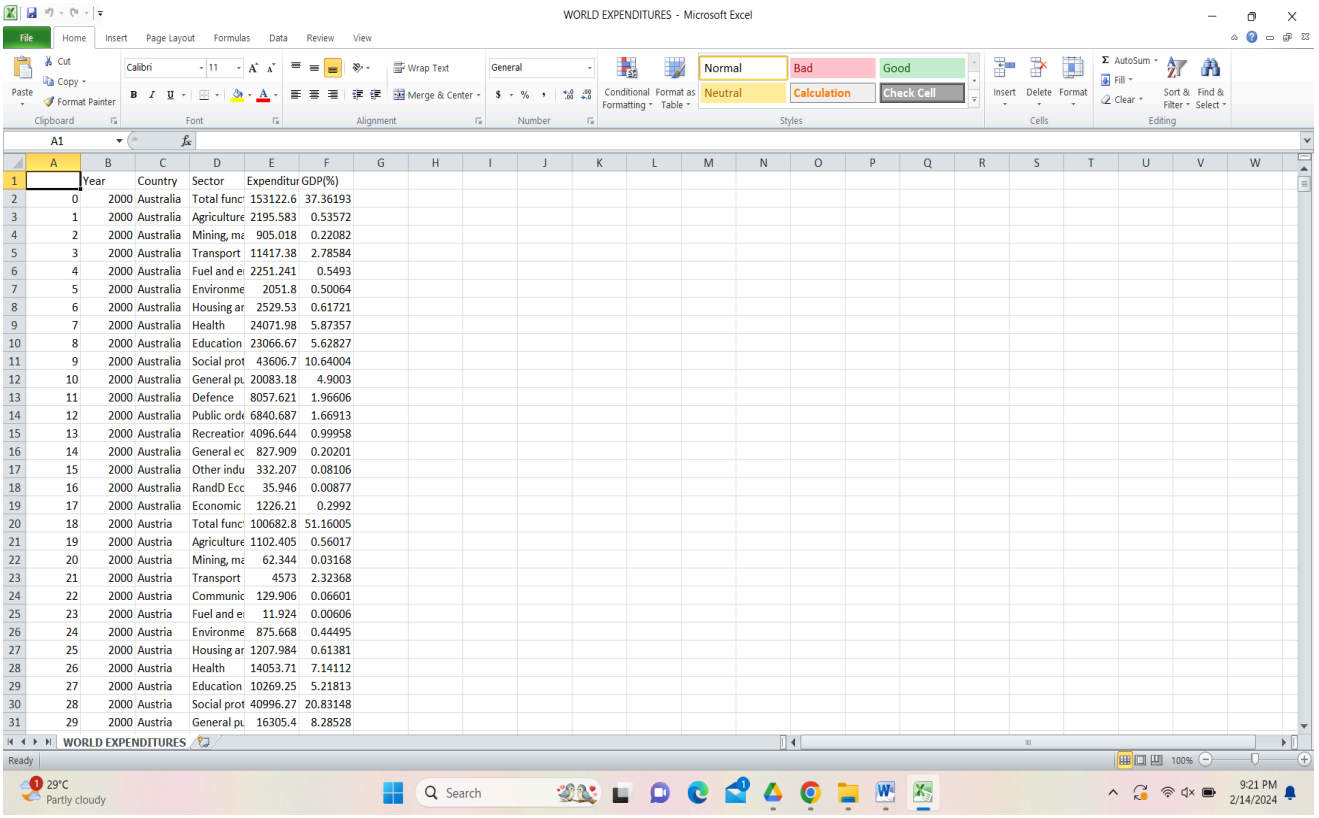
EXCEL:

Use pandas to_excel() function to write a DataFrame to an excel sheet with extension . xlsx. By default it writes a single DataFrame to an excel file, you can also write multiple sheets by using an ExcelWriter object with a target file name, and sheet name to write

SYNTAX:

***for excel:read_excel("file path")**

CSV FILE:



EXAMPLES:

Based on the above information:

```
df=pd.read_csv("/content/WORLD EXPENDITURES.csv")
df.head()
```

OUTPUT:

	Unnamed: 0	Year	Country	Sector	Expenditure(million USD)	GDP(%)
0	0	2000	Australia	Total function	153122.633	37.36193

1	1	2000	Australia	Agriculture, forestry, fishing and hunting	2195.583	0.53572
2	2	2000	Australia	Mining, manufacturing and construction	905.018	0.22082
3	3	2000	Australia	Transport	11417.379	2.78584
4	4	2000	Australia	Fuel and energy	2251.241	0.54930

2)

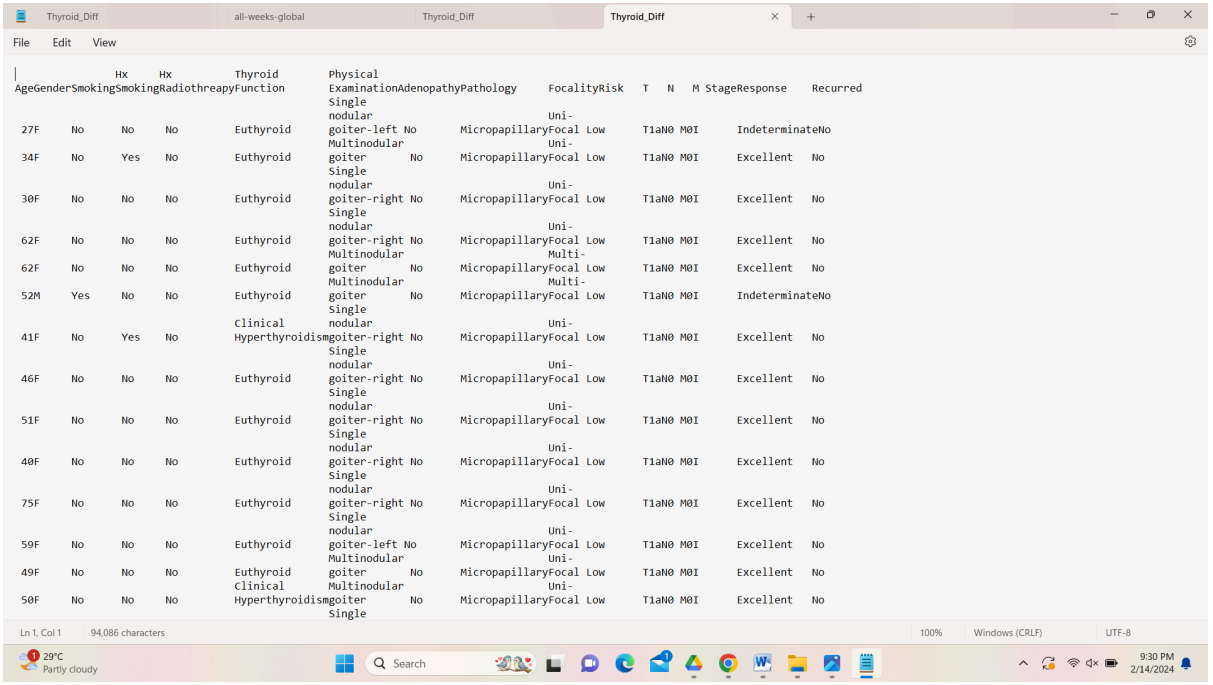
```
df.tail()
```

OUTPUT:

	Unnamed: 0	Year	Country	Sector	Expenditure(million USD)	GDP(%)
25728	25728	2021	United States of America	Social protection	2827585.06	12.07255
25729	25729	2021	United States of America	General public services	1291640.23	5.51474
25730	25730	2021	United States of America	Defence	759321.99	3.24197
25731	25731	2021	United States of America	Public order and safety	459973.19	1.96388
25732	25732	2021	United States of America	Recreation, culture and religion	62528.78	0.26697

EXAMPLE:

Based on the below text file information:



```
Age Gender Smoking Hx Smoking Hx Radiothreapy Thyroid Function Physical Examination Adenopathy Pathology Focality Risk T N M Stage Response Recurred
27 F No No No Euthyroid Single nodular goiter-left No Micropapillary Uni-Focal Low T1a N0 M0 I Indeterminate No
34 F No Yes No Euthyroid Multinodular goiter No Micropapillary Uni-Focal Low T1a N0 M0 I Excellent No
30 F No No No Euthyroid Single nodular goiter-right No Micropapillary Uni-Focal Low T1a N0 M0 I Excellent No
62 F No No No Euthyroid Single nodular goiter-right No Micropapillary Uni-Focal Low T1a N0 M0 I Excellent No
62 F No No No Euthyroid Multinodular goiter No Micropapillary Multi-Focal Low T1a N0 M0 I Excellent No
52 M Yes No No Euthyroid Multinodular goiter No Micropapillary Multi-Focal Low T1a N0 M0 I Indeterminate No
41 F No Yes No Clinical Hyperthyroidism Single nodular goiter-right No Micropapillary Uni-Focal Low T1a N0 M0 I Excellent No
46 F No No No Euthyroid Single nodular goiter-right No Micropapillary Uni-Focal Low T1a N0 M0 I Excellent No
51 F No No No Euthyroid Single nodular goiter-right No Micropapillary Uni-Focal Low T1a N0 M0 I Excellent No
40 F No No No Euthyroid Single nodular goiter-right No Micropapillary Uni-Focal Low T1a N0 M0 I Excellent No
75 F No No No Euthyroid Single nodular goiter-right No Micropapillary Uni-Focal Low T1a N0 M0 I Excellent No
59 F No No No Euthyroid Single nodular goiter-left No Micropapillary Uni-Focal Low T1a N0 M0 I Excellent No
49 F No No No Euthyroid Multinodular goiter No Micropapillary Uni-Focal Low T1a N0 M0 I Excellent No
50 F No No No Clinical Hyperthyroidism Multinodular goiter No Micropapillary Uni-Focal Low T1a N0 M0 I Excellent No
```

```
df=pd.read_csv("/content/Thyroid_Diff.txt",sep=",")
df.head(10)
```

OUTPUT:

	Unnamed: 0	Year	Country	Sector	Expenditure(million USD)	GDP(%)
0	0	2000	Australia	Total function	153122.633	37.36193
1	1	2000	Australia	Agriculture, forestry, fishing and hunting	2195.583	0.53572
2	2	2000	Australia	Mining, manufacturing and construction	905.018	0.22082
3	3	2000	Australia	Transport	11417.379	2.78584
4	4	2000	Australia	Fuel and energy	2251.241	0.54930
5	5	2000	Australia	Environment protection	2051.800	0.50064
6	6	2000	Australia	Housing and community amenities	2529.530	0.61721

7	7	2000	Australia	Health	24071.984	5.87357
8	8	2000	Australia	Education	23066.666	5.62827
9	9	2000	Australia	Social protection	43606.701	10.64004

DESCRIBE:

Pandas describe() is used to view some basic statistical details like percentile, mean, std, etc. of a data frame or a series of numeric values. When this method is applied to a series of strings, it returns a different output

EXAMPLES:

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

OUTPUT:

[ ]

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

output

<bound method NDFrame.describe of Hx Hx Thyroid Physical
0 AgeGenderSmokingSmokingRadiothreapyFunction ...
1 ...
2 ...
3 27F No No No Euthyroid ...
4 ...
... ...
1001 Clinical Multino...
1002 61M Yes Yes Yes Hyperthyroidismgoiter ...
1003 Multino...
1004 67M Yes No No Euthyroid goiter ...
1005

[1006 rows x 1 columns]>

Shape[0],shape[1]:

Usually, on a broader scale, the shape() method is used to fetch the dimensions of Pandas and NumPy type objects in python. Every value represented by the tuple corresponds to the actual dimension in terms of array or row/columns.

shape is a tuple that gives you an indication of the number of dimensions in the array. So in your case, since the index value of Y. shape[0] is 0, your are working along the first dimension of your array.

EXAMPLES:

```
Print(dft.shape)#get no of rows and coloumns
```

OUTPUT:

(1006 , 1)

2)

```
print(dft.shape[0])#get only no of rows
```

OUTPUT:

(1006)

3)

```
print(dft.shape[0])#get only no of rows
```

OUTPUT:

(1)

COLUMNS:

In Pandas, DataFrame. columns is an attribute that provides access to the column labels of a data frame. It returns an Index object representing the names of the columns in the DataFrame

EXAMPLES:

```
print(dft.columns)
```

OUTPUT:

```
Index(['                    Hx      Hx           Thyroid          Physical'],
dtype='object')
```

2)

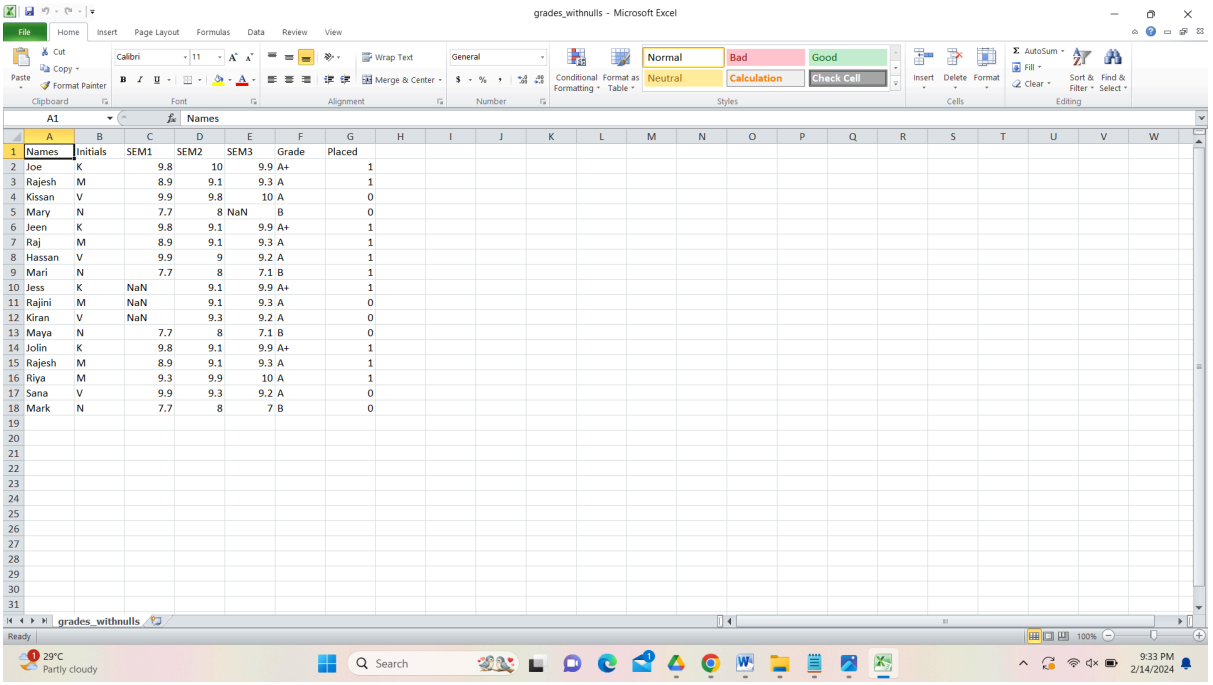
```
print(dft[2:5])
```

OUTPUT:

```
                    Hx      Hx           Thyroid          Physical
2                                                          ...
3    27F        No          No       No           Euthyroid  ...
4                                                          ...
```

EXAMPLE:

CSV FILE (GRADES):



Names	Initials	SEM1	SEM2	SEM3	Grade	Placed
Joe	K	9.8	10	9.9	A+	1
Rajesh	M	8.9	9.1	9.3	A	1
Kissan	V	9.9	9.8	10	A	0
Mary	N	7.7	8	NaN	B	0
Jeen	K	9.8	9.1	9.9	A+	1
Raj	M	8.9	9.1	9.3	A	1
Hassan	V	9.9	9	9.2	A	1
Mari	N	7.7	8	7.1	B	1
Jess	K	NaN	9.1	9.9	A+	1
Rajini	M	NaN	9.1	9.3	A	0
Kiran	V	NaN	9.3	9.2	A	0
Maya	N	7.7	8	7.1	B	0
Jolin	K	9.8	9.1	9.9	A+	1
Rajesh	M	8.9	9.1	9.3	A	1
Riya	M	9.3	9.9	10	A	1
Sana	V	9.9	9.3	9.2	A	0
Mark	N	7.7	8	7	B	0

```
dfe=pd.read_csv("/content/grades_withnulls.csv")
```

```
dfe.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	NaN	B	0
4	Jeen	K	9.8	9.1	9.9	A+	1

ACCESSING DATA:(LOC,ILOC)

Loc and iloc are two functions in Pandas that are used to slice a data set in a Pandas DataFrame. The function . loc is typically used for label indexing and can access multiple columns, while . iloc is used for integer indexing.

EXAMPLES:

1)

```
print(dfe.loc[2:5],["Names"])#rows of specified columns
```

```
 Names Initials  SEM1  SEM2  SEM3 Grade  Placed
2  Kissan        V   9.9   9.8  10.0     A       0
3    Mary        N   7.7   8.0   NaN     B       0
4    Jeen        K   9.8   9.1   9.9    A+       1
5     Raj        M   8.9   9.1   9.3     A       1 ['Names']
```

2)

```
print(dfe.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
```

3)

```
print(dfe.iloc[2:5])
```

OUTPUT:

```
   Names Initials  SEM1  SEM2  SEM3 Grade  Placed
2  Kissan        V   9.9   9.8  10.0     A       0
3    Mary        N   7.7   8.0   NaN     B       0
4    Jeen        K   9.8   9.1   9.9    A+
```

4)

```
print(dfe.loc[2:5,"Names"])
```

OUTPUT:

```
2    Kissan
3      Mary
4      Jeen
5       Raj
Name: Names, dtype: object
```

NULL:

o check for null values in a pandas DataFrame, we can use the isnull() method. The isnull() method returns a DataFrame of the same shape as the input DataFrame, but with boolean values indicating whether each cell is null or not

EXAMPLES:

1)

```
dfe.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

2)

```
dfe.isnull().sum()#to knew how many nulls we have per column
```

OUTPUT:

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

3)

```
dfe.isnull().sum().sum()#to view total nulls
```

OUTPUT:

```
4
```

DROOPNA:

The dropna() method removes the rows that contains NULL values. The dropna() method returns a new DataFrame object unless the inplace parameter

is set to True , in that case the dropna() method does the removing in the original DataFrame instead.

EXAMPLES:

1)

```
dfe.dropna()#drooping 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

2)

```
dfc=dfe.dropna()#drooping all the rows with nulls
```

```
dfc=dfe.dropna()#drooping 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

FILLNA(USAGE):

The `fillna()` method replaces the NULL values with a specified value.

The `fillna()` method returns a new DataFrame object unless the `inplace` parameter is set to `True`, in that case the `fillna()` method does the replacing in the original DataFrame instead.

```
dfc1=dfe.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

CLEANING DATA:(MEAN)

Data cleaning means fixing bad data in your data set. Bad data could be: Empty cells. Data in wrong format. Wrong data.

EXAMPLES:

1)

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

OUTPUT:

```
9.100000000000001
```

2)

```
dfc2=dfe.fillna(m)
print(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

RENAME COLUMN:

One way of renaming the columns in a Pandas Dataframe is by using the rename() function. This method is quite useful when we need to rename some selected columns because we need to specify information only for the columns which are to be renamed.

EXAMPLES:

1)

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

OUTPUT:

	Names	Initials	SEM1	SEM2	SEM3	GPA	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	NaN	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	NaN	9.1	9.9	A+	1
9	Rajini	M	NaN	9.1	9.3	A	0
10	Kiran	V	NaN	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

2)

```
dfe.rename(columns={"Grade" :"GPA"},inplace =True)
dfe.head()
```

OUTPUT:

Names	Initials	SEM1	SEM2	SEM3	GPA	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	NaN	B	0
4	Jeen	K	9.8	9.1	9.9	A+	1

AVERAGE:

The simplest way to compute row averages in pandas is to use the mean() method. This method calculates the mean (i.e., average) of a DataFrame or Series along a specified axis, which can be either rows or columns.

EXAMPLE:
```
dfe['Avg_score']=(dfe['SEM1']+ dfe['SEM2']+ dfe['SEM3'])/3#df['newcol]=values
dfe.head()
```

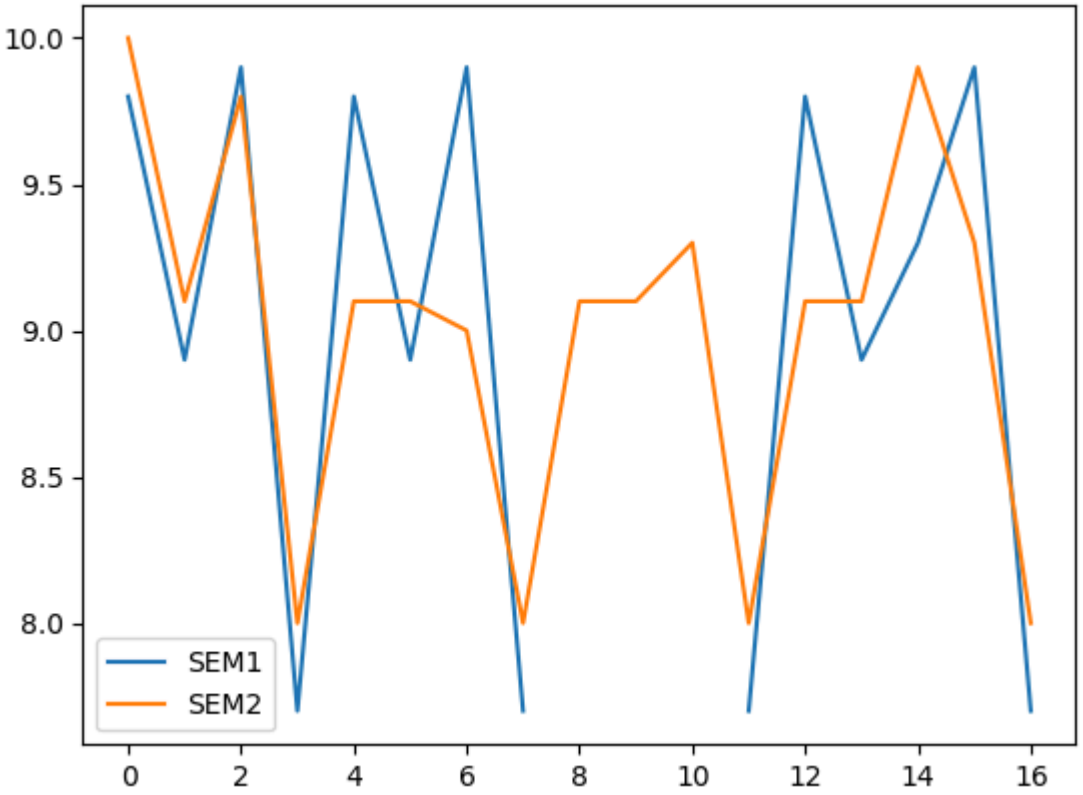
OUTPUT:

Names	Initials	SEM1	SEM2	SEM3	GPA	Placed	Avg_score	
0	Joe	K	9.8	10.0	9.9	A+	1	9.9
1	Rajesh	M	8.9	9.1	9.3	A	1	9.1
2	Kissan	V	9.9	9.8	10.0	A	0	9.9
3	Mary	N	7.7	8.0	NaN	B	0	NaN
4	Jeen	K	9.8	9.1	9.9	A+	1	9.6

PLOTTING WITH EXAMPLES:

1)
```
dfe[['SEM1','SEM2']].plot.line()
```

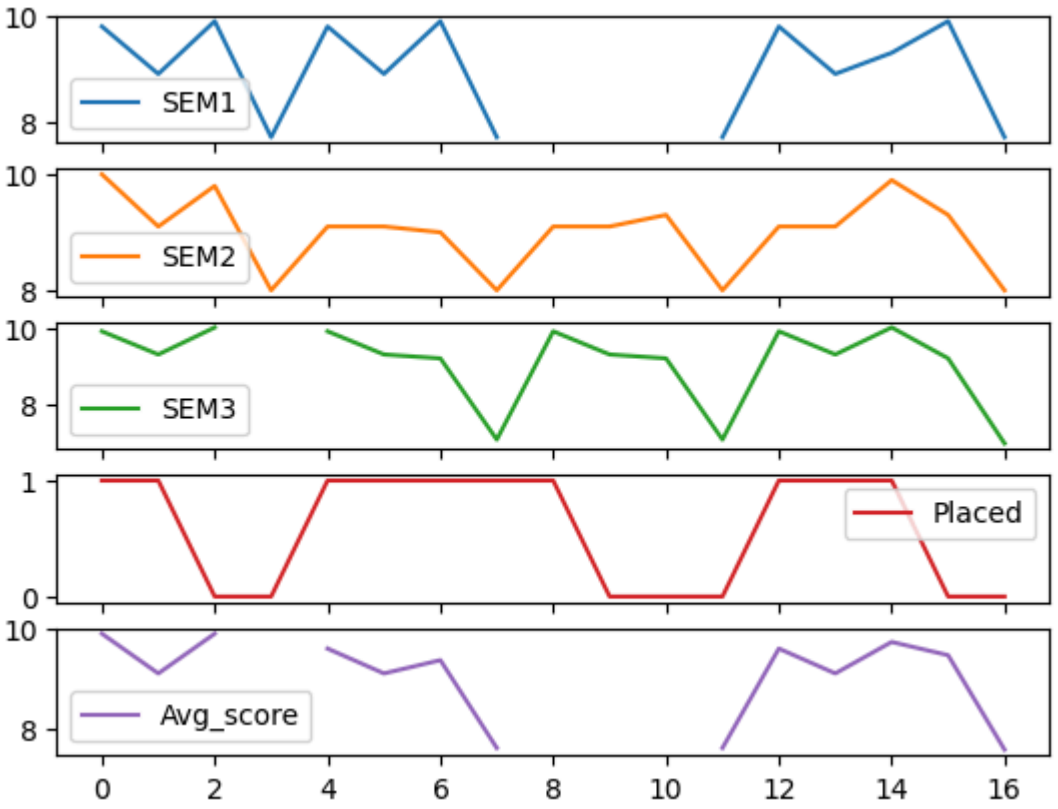
OUTPUT:



2)

```
dfe.plot.line(subplots=True)
```

OUTPUT:



SEABORN

DEFINITION:

Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. For a brief introduction to the ideas behind the library, you can read the introductory notes or the paper.

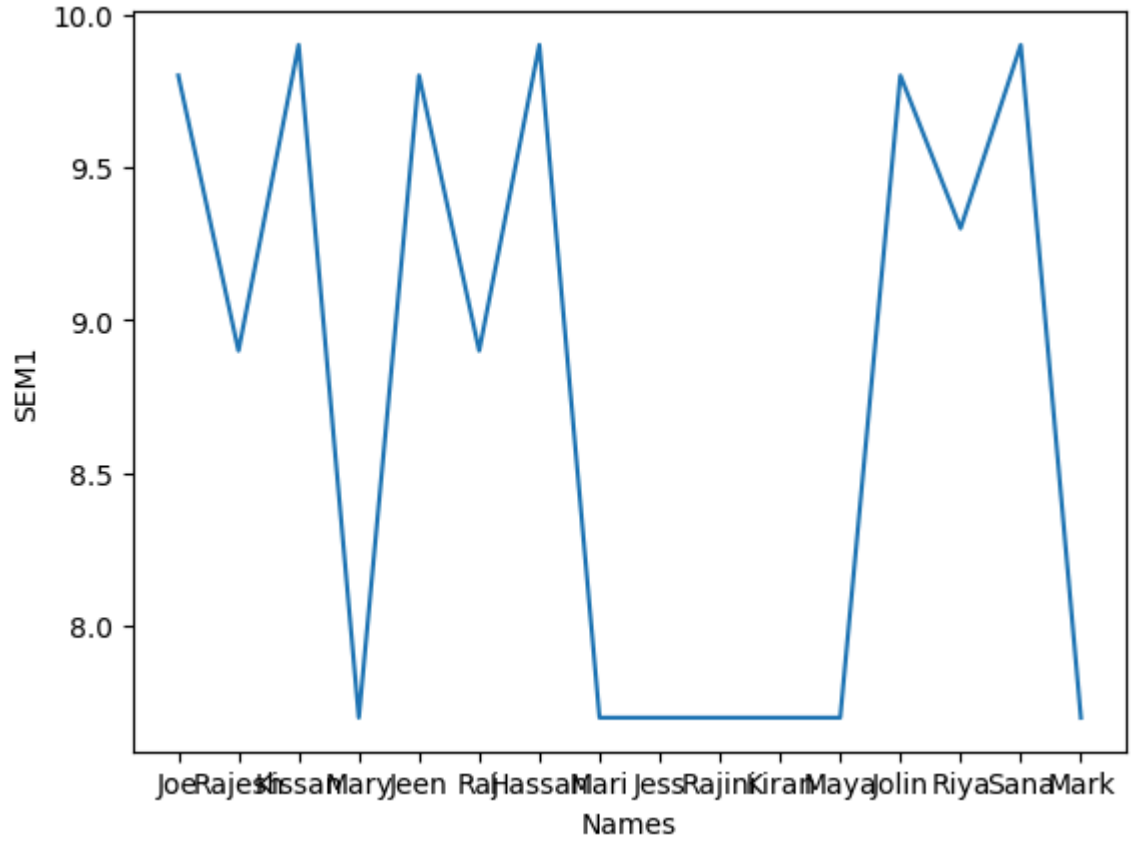
```
import seaborn as sns
```

EXAMPLES:

1)

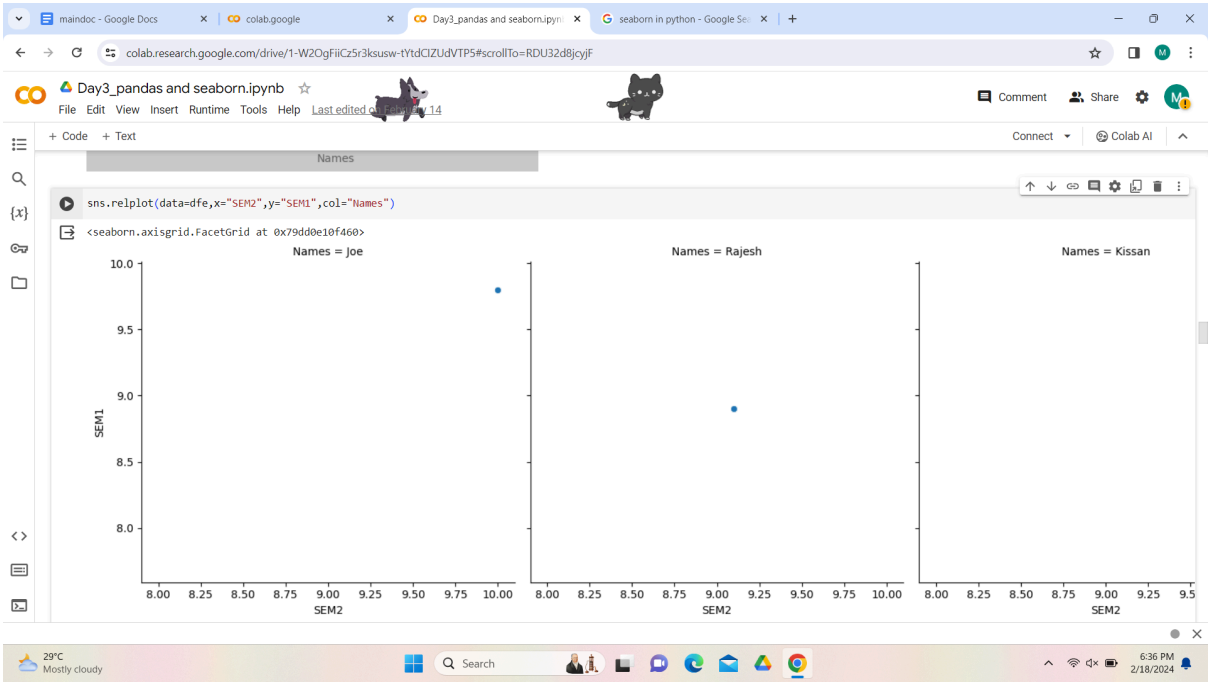
```
pl=sns.lineplot(x='Names',y='SEM1',data=dfe)
```

OUTPUT:



2)EXAMPLE:

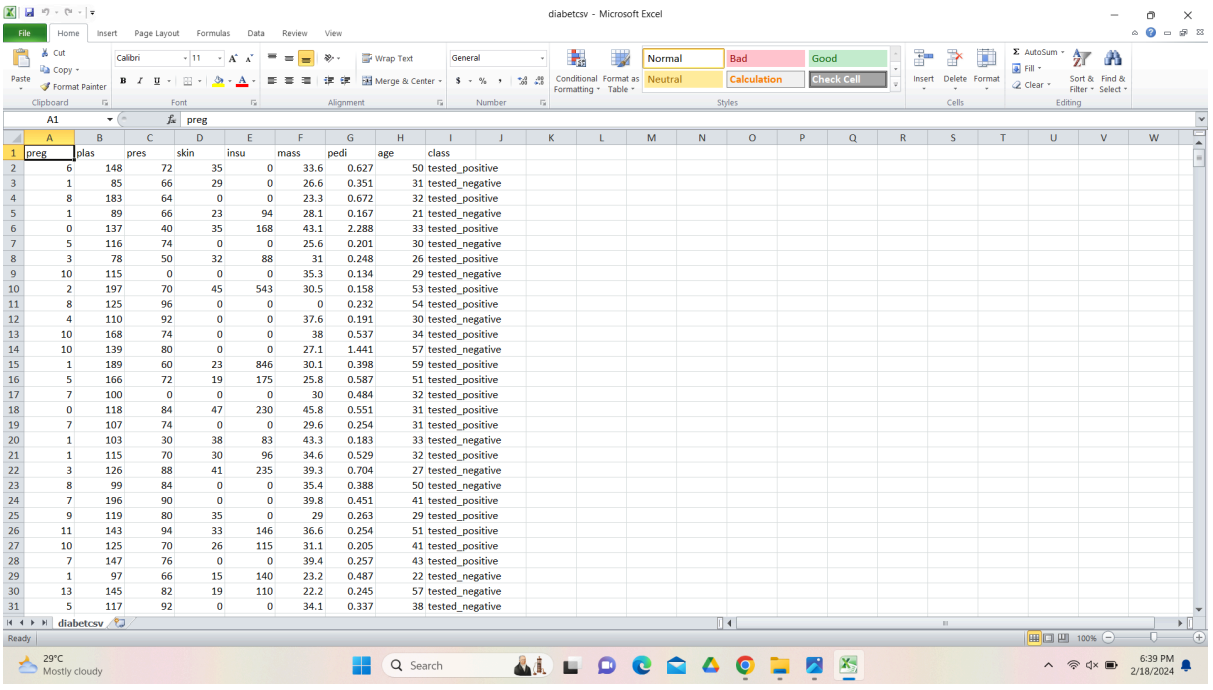
OUTPUT:



EXAMPLE QUESTION:

- **load diabeties**
- **create a replot age in x-axis and class as coloumns**

ANSWER:




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

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

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

INDEX AND RANGE FINDING:

You would need to make sure the string you provide to the x argument is actually a *column* in your dataframe. The easiest solution to achieve that is to reset the index of the dataframe to convert the index to a column.

EXAMPLE:

```
ds['index']=range(1,769)
ds
```

OUTPUT:

	preg	plas	pres	skin	insu	mass	pedi	age	class	index
0	6	148	72	35	0	33.6	0.627	50	tested_positive	1
1	1	85	66	29	0	26.6	0.351	31	tested_negative	2
2	8	183	64	0	0	23.3	0.672	32	tested_positive	3
3	1	89	66	23	94	28.1	0.167	21	tested_negative	4
4	0	137	40	35	168	43.1	2.288	33	tested_positive	5
...	...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	tested_negative	764
764	2	122	70	27	0	36.8	0.340	27	tested_negative	765
765	5	121	72	23	112	26.2	0.245	30	tested_negative	766
766	1	126	60	0	0	30.1	0.349	47	tested_positive	767
767	1	93	70	31	0	30.4	0.315	23	tested_negative	768

768 rows × 10 columns

RELPLOT:

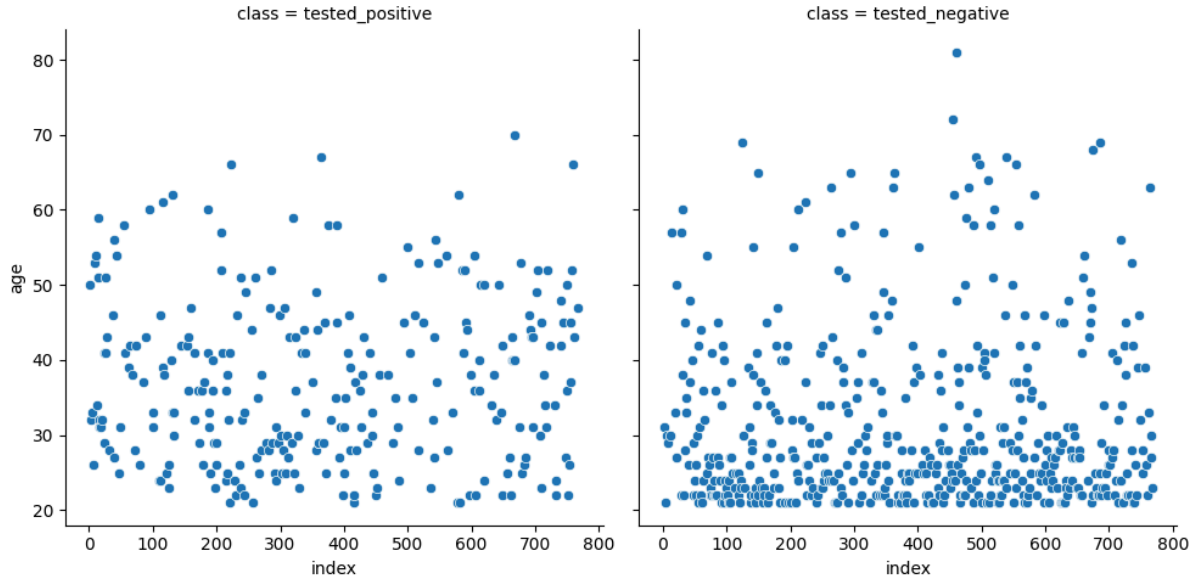
The Seaborn Relational Plot (relplot) allows us to visualise how variables within a dataset relate to each other. Data visualisation is an essential part of any data analysis or machine learning workflow. It allows to gain insights about our data. As the famous saying suggests, 'a picture is worth a thousand words'.

EXAMPLES:

1)

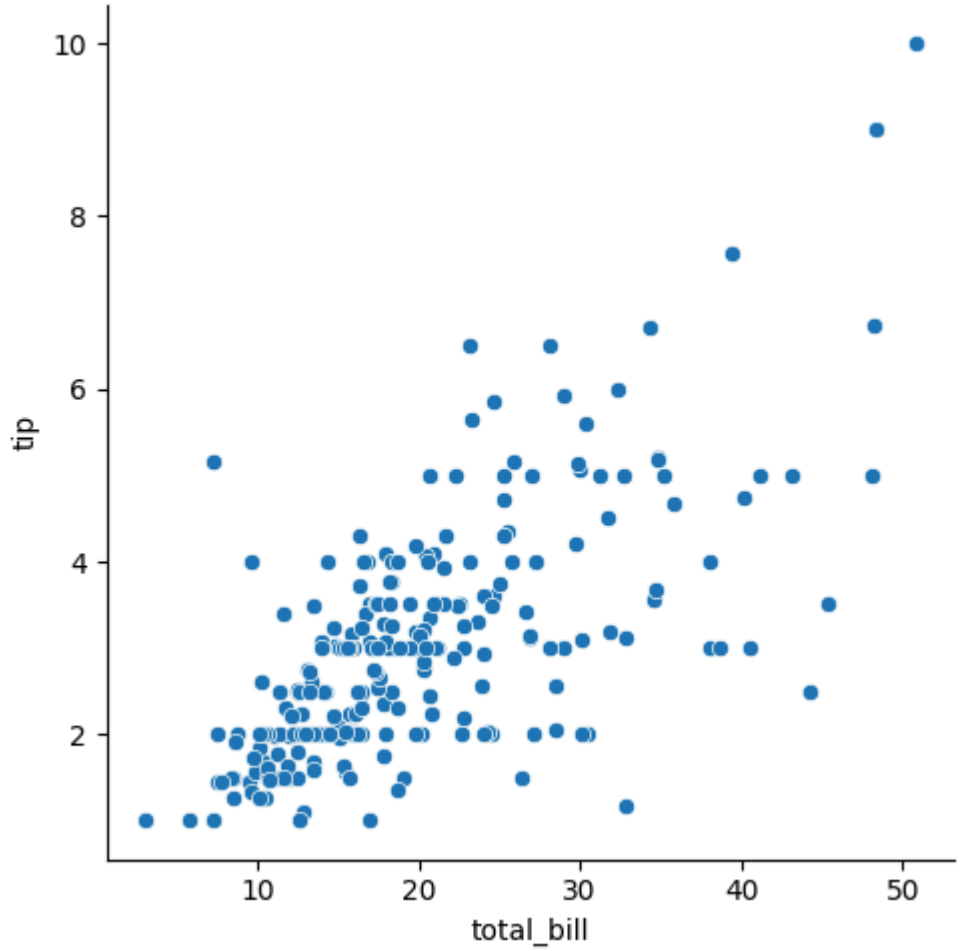
```
sns.relplot(data=ds,x="index",y="age",col="class")
```

OUTPUT:




```
sns.relplot(data=tips,x="total_bill",y="tip")
```

OUTPUT:



inbuilt datasets in seaborn:

- **tips**
- **dow jones**
- **fmri**
- **dots**
- **healthexp**
- **to load dataset use:**
- **load_0dataset("dowjones")**
- **hue=diff colour for diff category**
- **style=fiff markers for different**

  TIPS:

  The `tips` dataset, embedded within Seaborn, is a collection of records representing the tipping behavior of restaurant patrons. The dataset comprises: - `total_bill`: Total bill amount. - `tip`: Tip amount.

  EXAMPLES:

  1)

```
tips=sns.load_dataset("tips")
```

```
tips.head()
```

OUTPUT:

total_bill	sex	smoker	time
16.99	Female	No	Dinner
10.34	Male	No	Dinner
21.01	Male	No	Dinner
23.68	Male	No	Dinner

24.59	Female	No	Dinner

- **DOTS:**

  Seaborn is a library for making statistical graphics in Python. It builds on top of matplotlib and integrates closely with pandas data structures. Seaborn helps you explore and understand your data.

  EXAMPLE:

```
dots=sns.load_dataset("dots")

dots.head()
```

OUTPUT:

	align	choice	time	coherence	firing_rate
0	dots	T1	-80	0.0	33.189967
1	dots	T1	-80	3.2	31.691726
2	dots	T1	-80	6.4	34.279840
3	dots	T1	-80	12.8	32.631874
4	dots	T1	-80	25.6	35.060487

HEALTHEX:

EXAMPLE:

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

OUTPUT:

	Year	Country	Spending_USD	Life_Expectancy
0	1970	Germany	252.311	70.6
1	1970	France	192.143	72.2
2	1970	Great Britain	123.993	71.9
3	1970	Japan	150.437	72.0
4	1970	USA	326.961	70.9

FMRI:

It is a declarative open-source Python library built on top of Matplotlib. It is recommended to integrate the Seaborn library using the Jupyter/IPython interface in Matplotlib. In comparison to Matplotlib, Seaborn's functionality uses fewer syntax and specializes in statistics visualization beyond basic plotting

EXAMPLE:

```
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

DOW JONES:

In this Notebook we will have a quick look at the historic Dow Jones data-set, perform feature engineering and use plots to understand the results.

EXAMPLE:

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

OUTPUT:

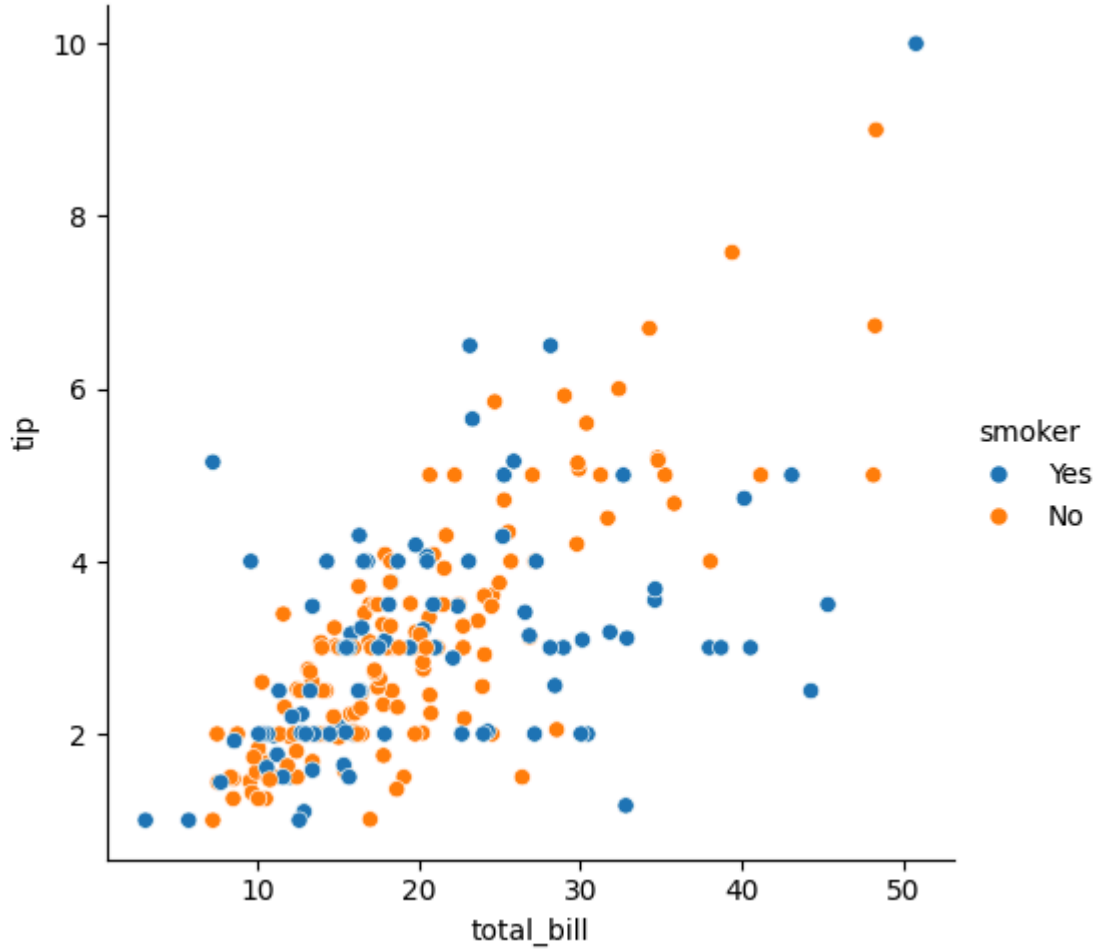
	Date	Price
0	1914-12-01	55.00
1	1915-01-01	56.55
2	1915-02-01	56.00
3	1915-03-01	58.30
4	1915-04-01	66.45

ALL EXAMPLES:

1)

```
sns.relplot(data=tips,x="total_bill",y="tip",hue="smoker")#create a differentiation based on a column via colours
```

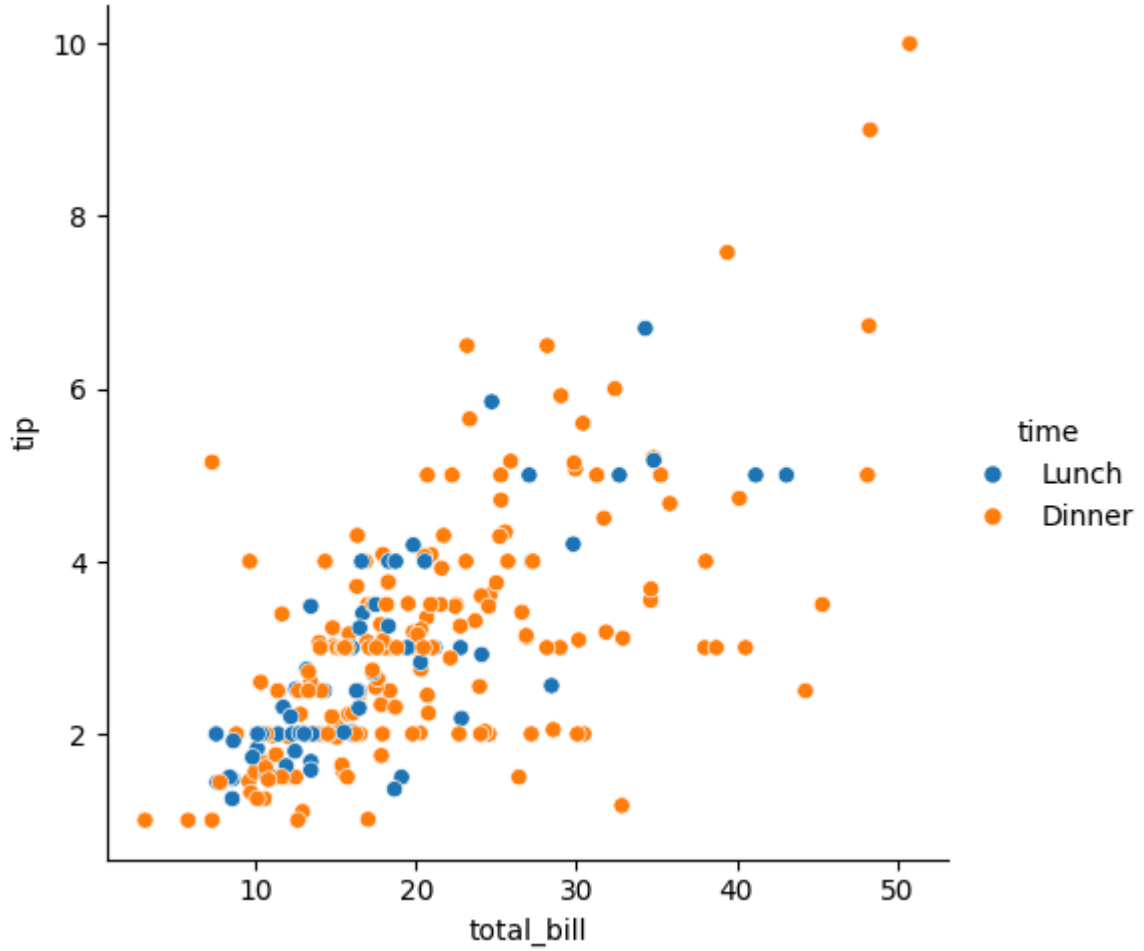
OUTPUT:



2)

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

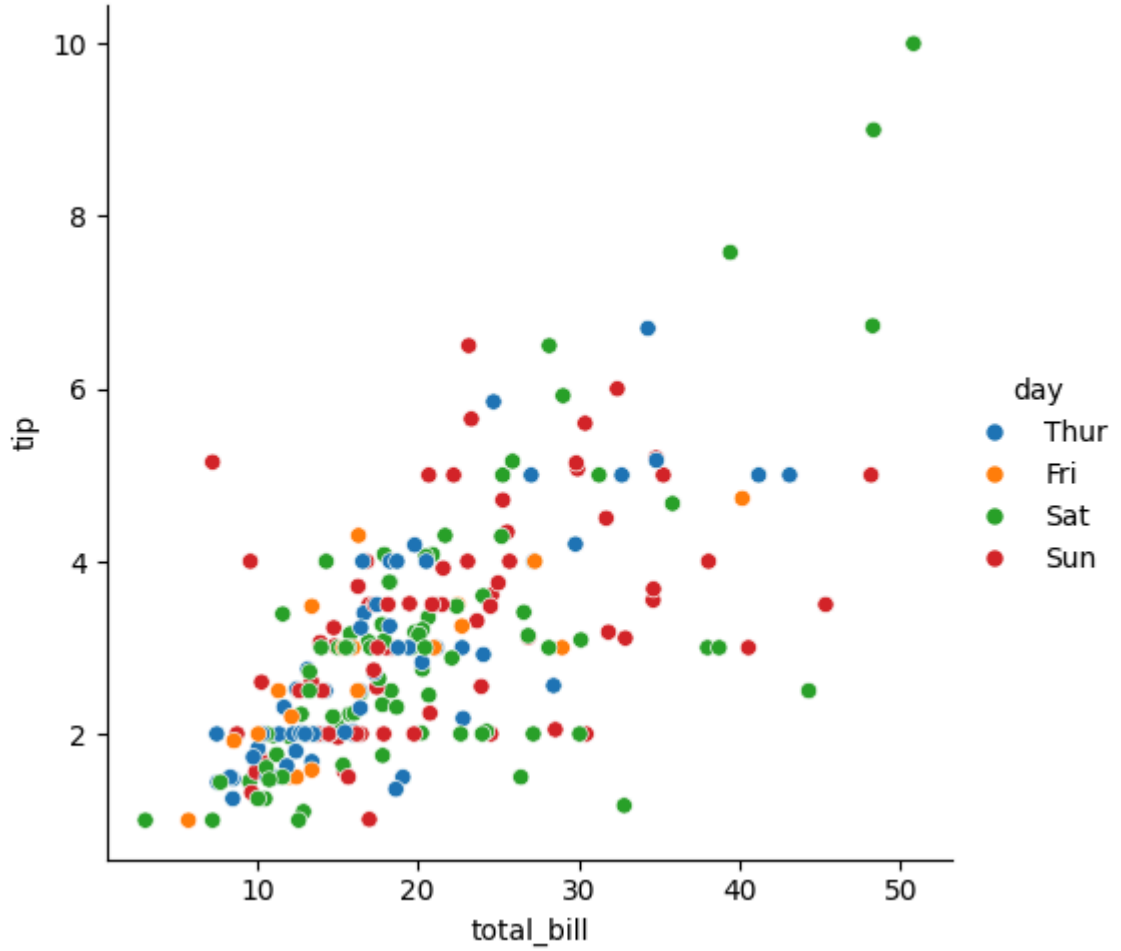
OUTPUT:



3)

```
sns.relplot(data=tips,x="total_bill",y="tip",hue="day")
```

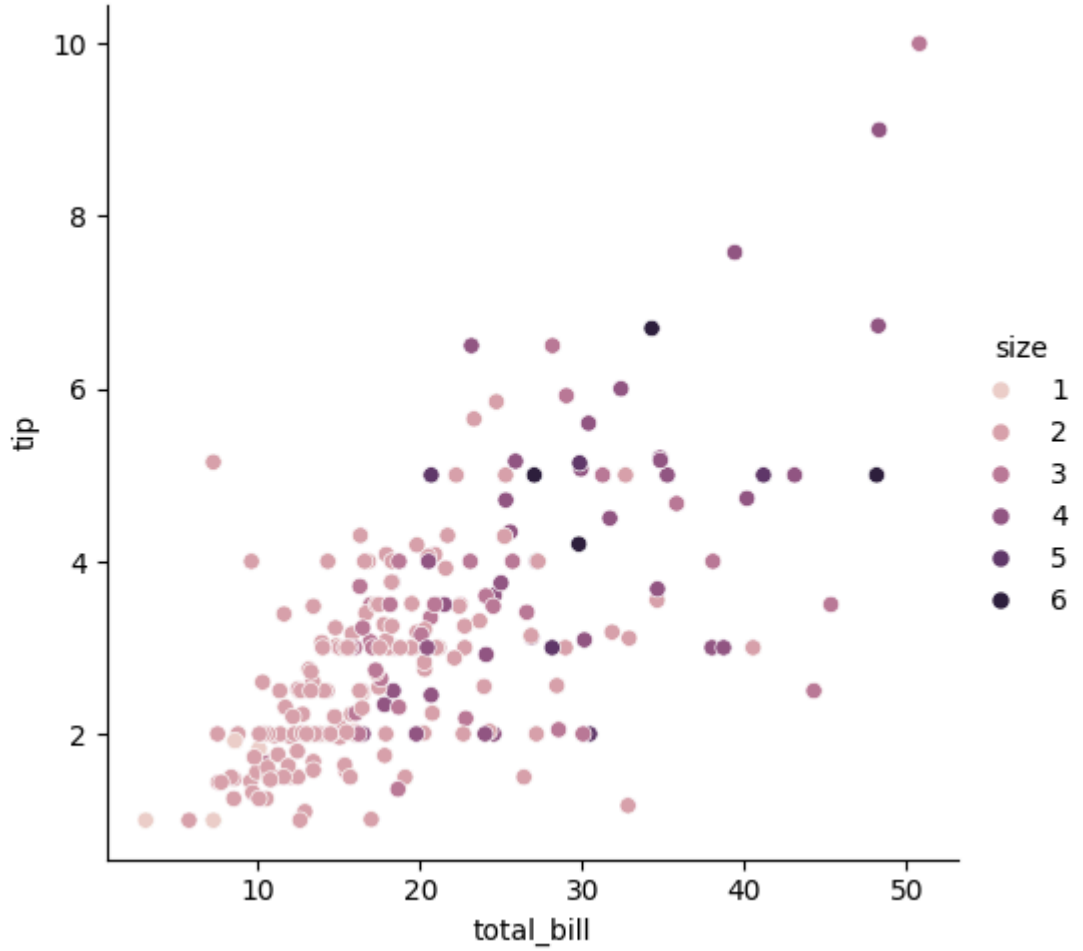
OUTPUT:



4)

```
sns.relplot(data=tips,x="total_bill",y="tip",hue="size")
```

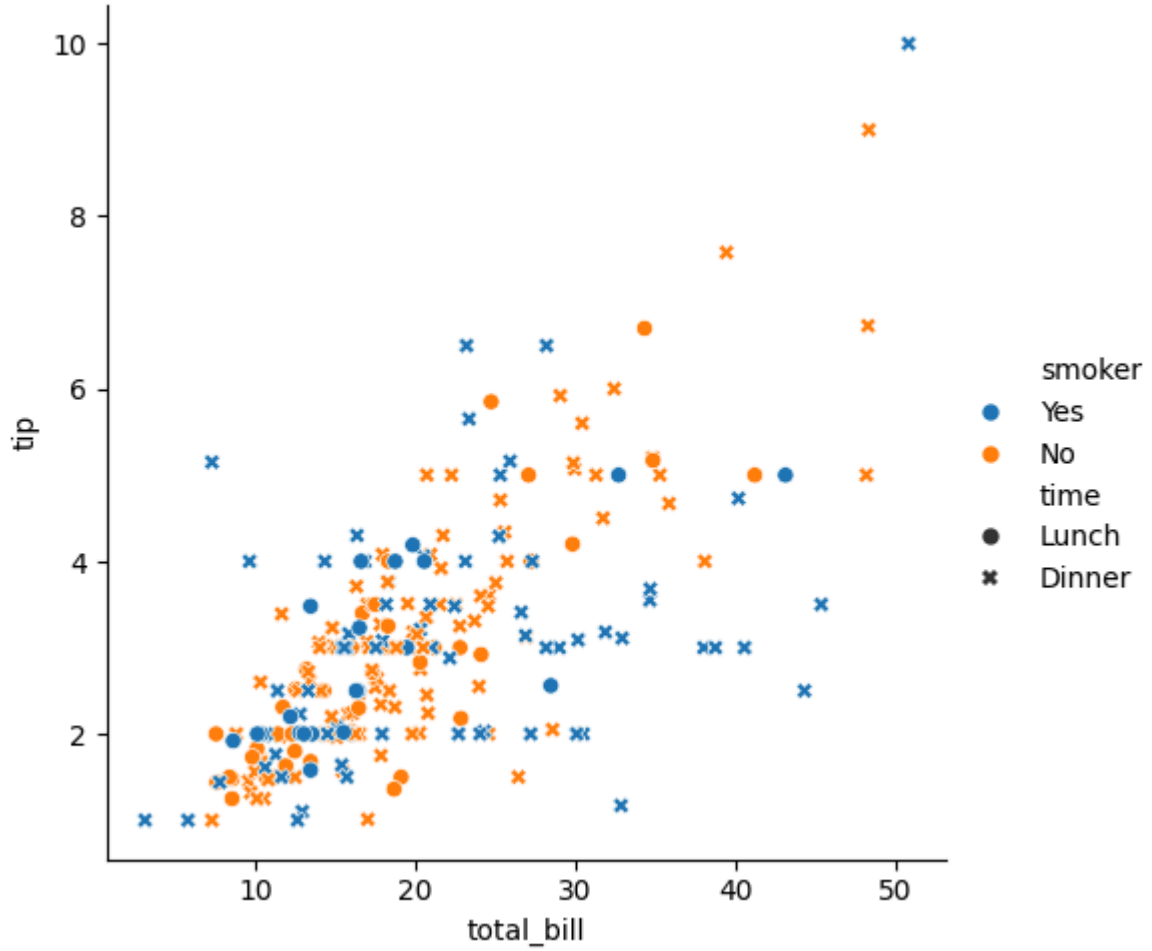
OUTPUT:



5)

```
sns.relplot(data=tips,x="total_bill",y="tip",hue="smoker",style="time")
```

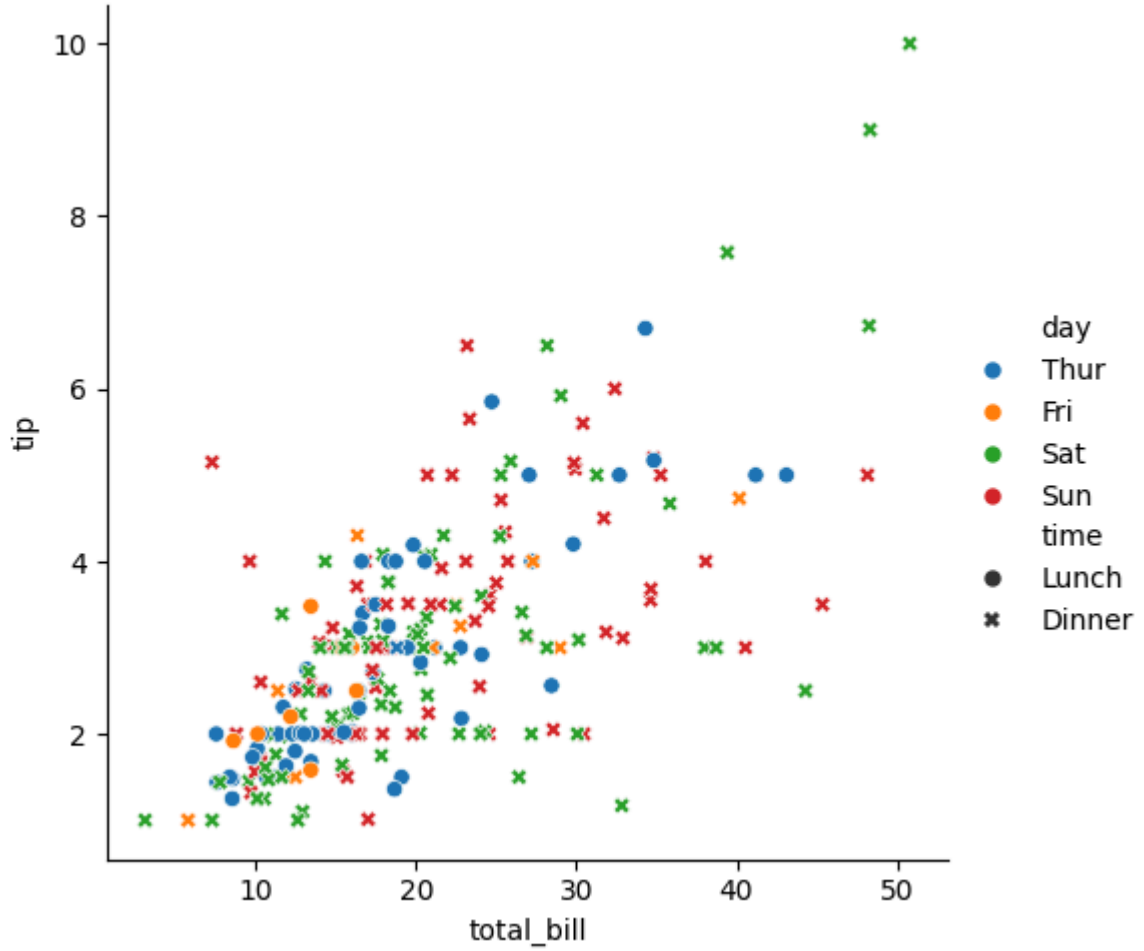
OUTPUT:



6)

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

OUTPUT:



SOME OTHER PROPERTIES TO PLOTTING

***pallets:**

Seaborn Default Color Palette

Not only does this function allow you the ability to use Seaborn default colors, but also any of Seaborn's other styling techniques. Seaborn has six variations of its default color palette: deep , muted , pastel , bright , dark , and colorblind .

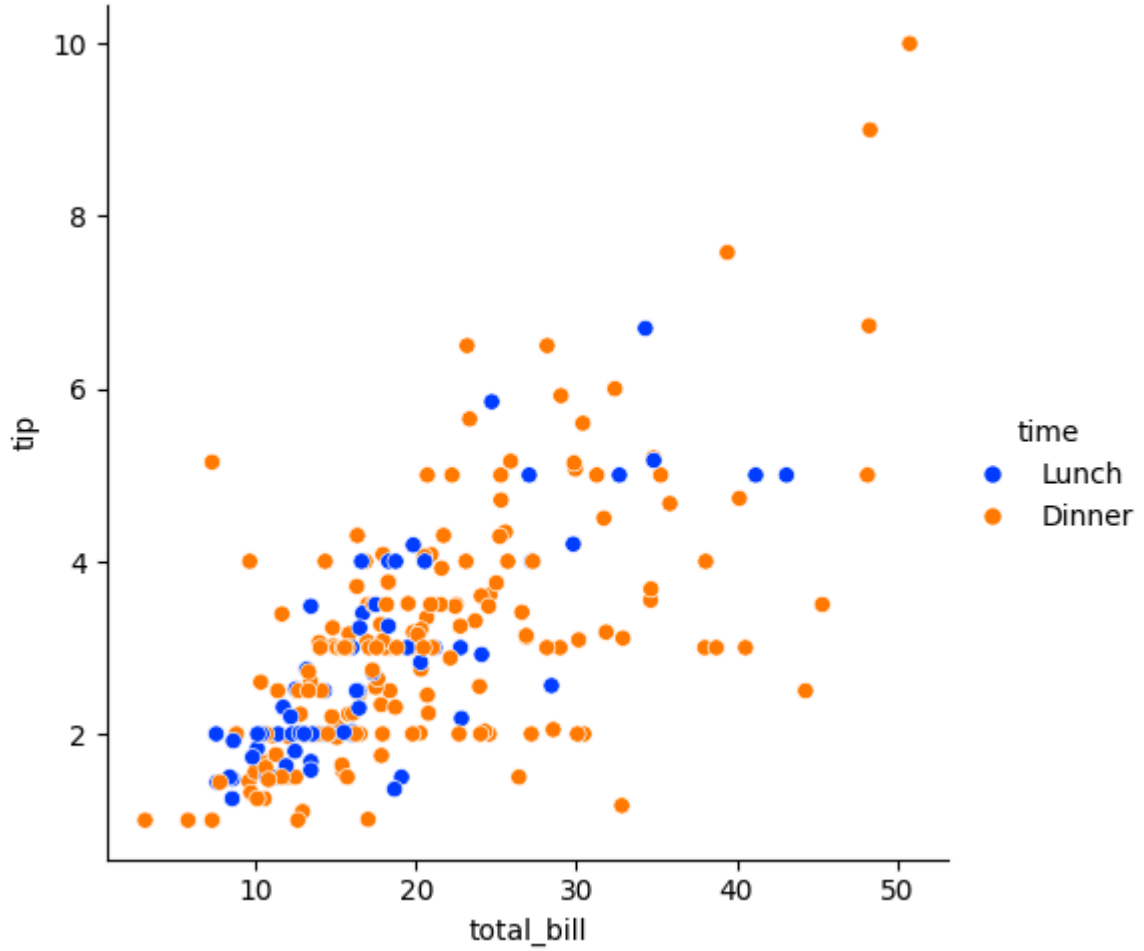
***bright:**

EXAMPLES:

1)

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

OUTPUT:

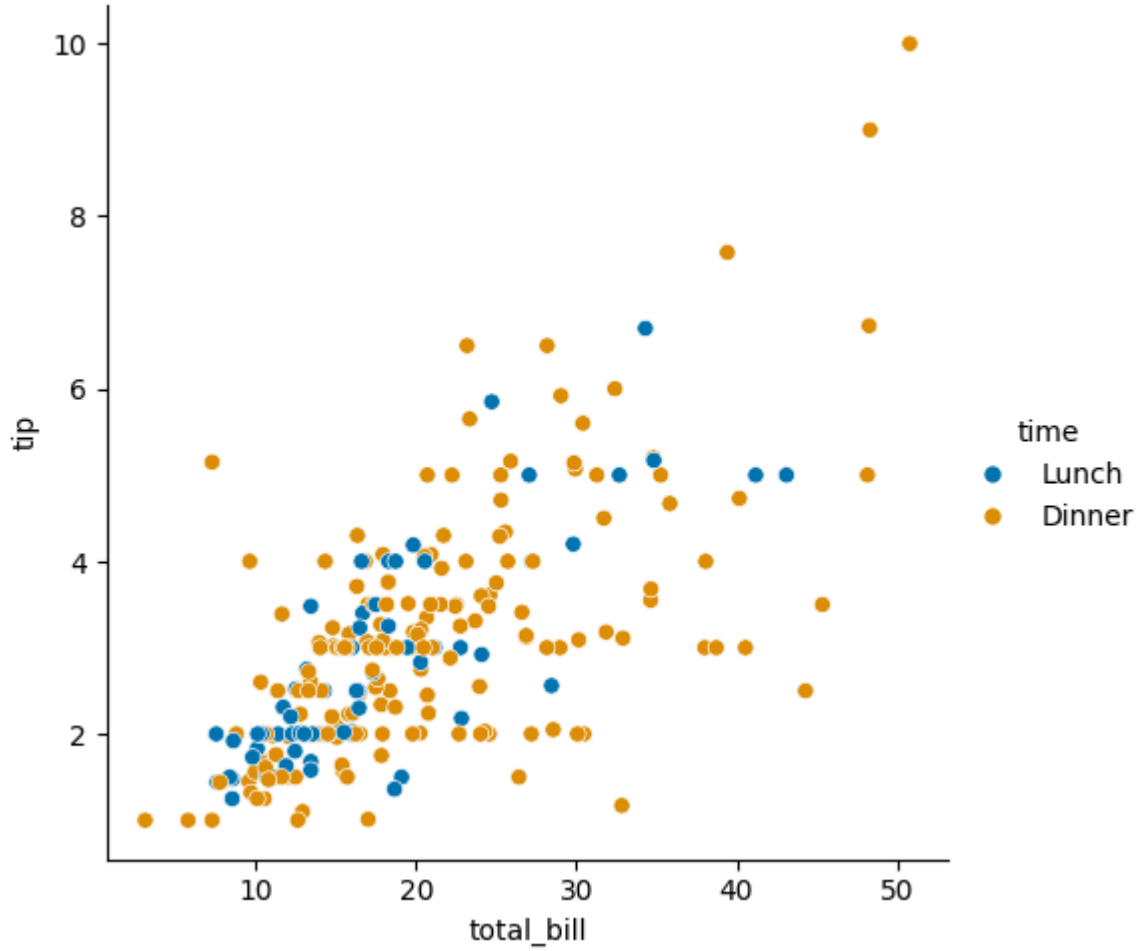


2)COLORBLIND:

EXAMPLE:

```
sns.relplot(data=tips,x="total_bill",y="tip",hue="time",palette="colorblind")
```

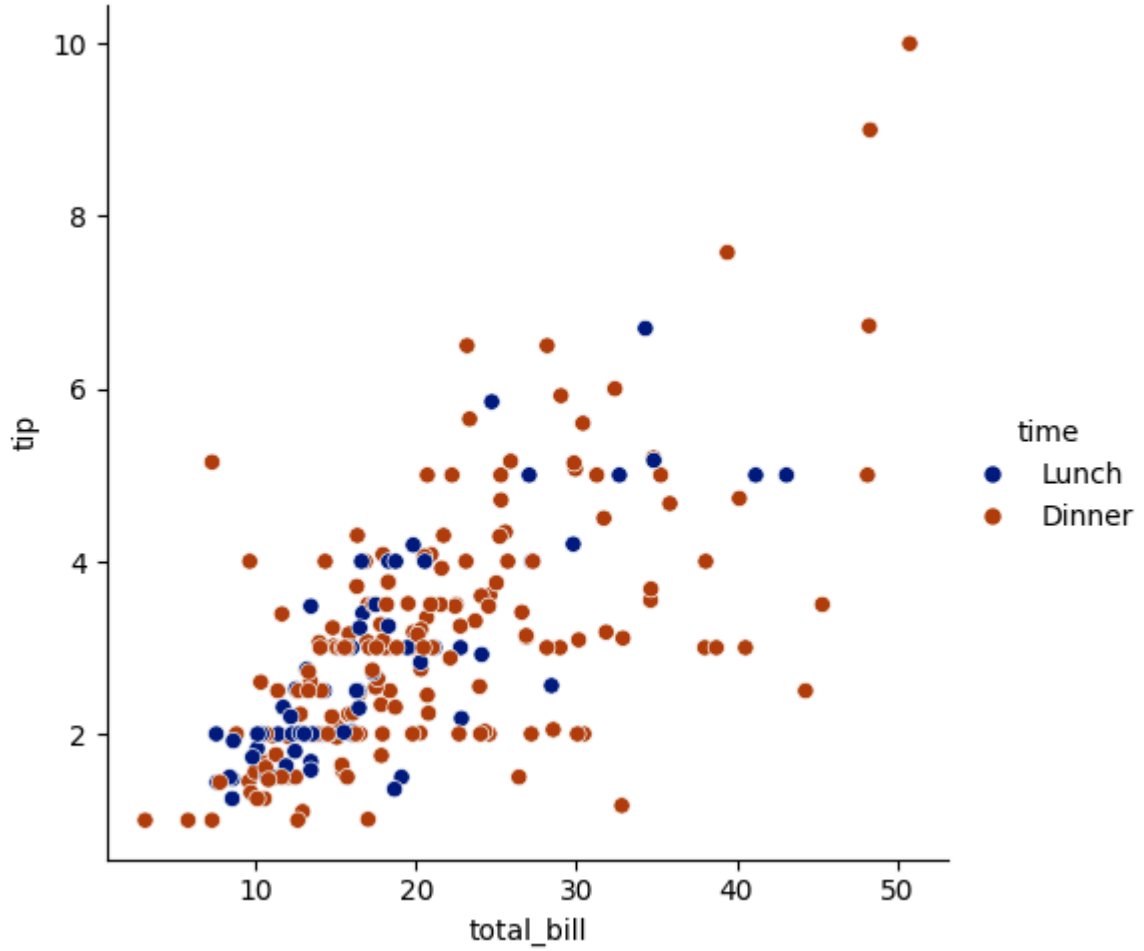
OUTPUT:



3)

```
sns.relplot(data=tips,x="total_bill",y="tip",hue="time",palette="dark")
```

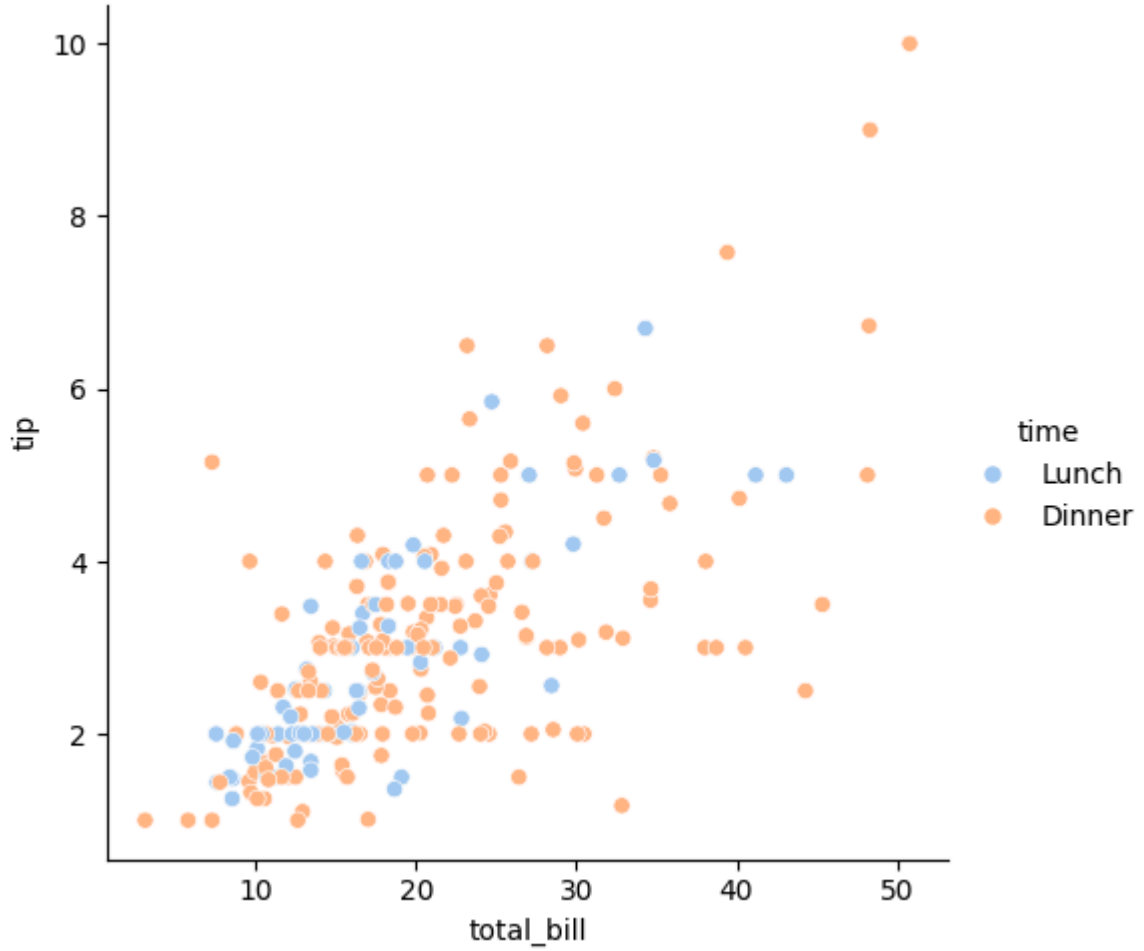
OUTPUT:



3)PASTEL:

```
sns.relplot(data=tips,x="total_bill",y="tip",hue="time",palette="pastel
")
```

OUTPUT:



LINE:

Creating a single seaborn line plot

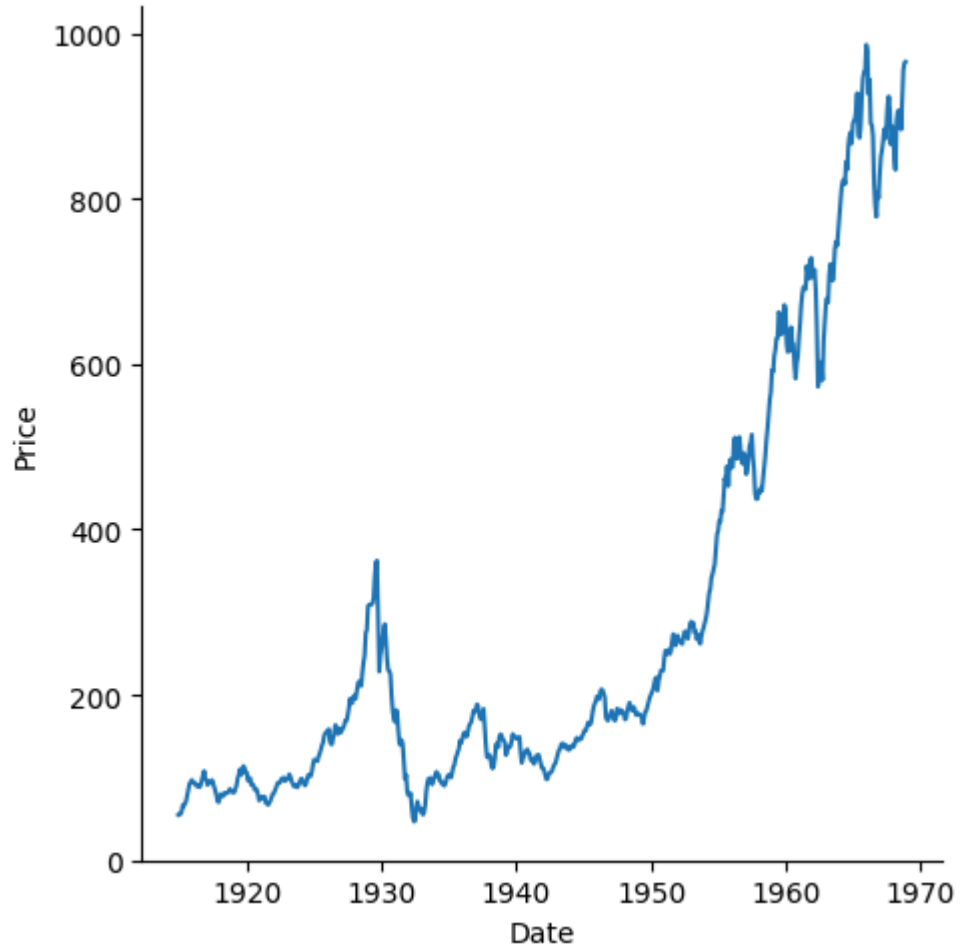
We defined the variables to plot on the x and y axes (the x and y parameters) and the dataframe (data) to take these variables from. In this case, we passed in one more argument specific to the relplot() function: kind='line'. By default, this function creates a scatter plot.

EXAMPLES:

1)

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

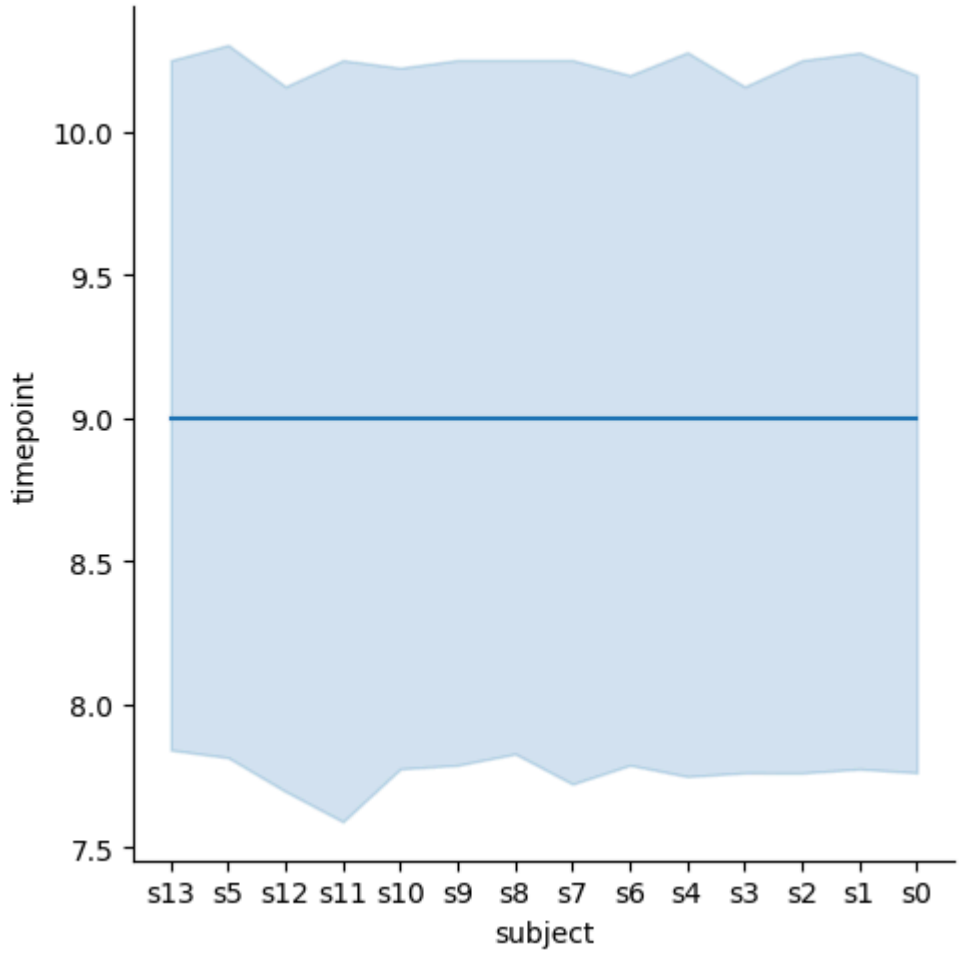
OUTPUT:



2)

```
sns.relplot(data=fmri,x="subject",y="timepoint",kind="line")
#subject means patience ,time point means no of seconds,region means
location
```

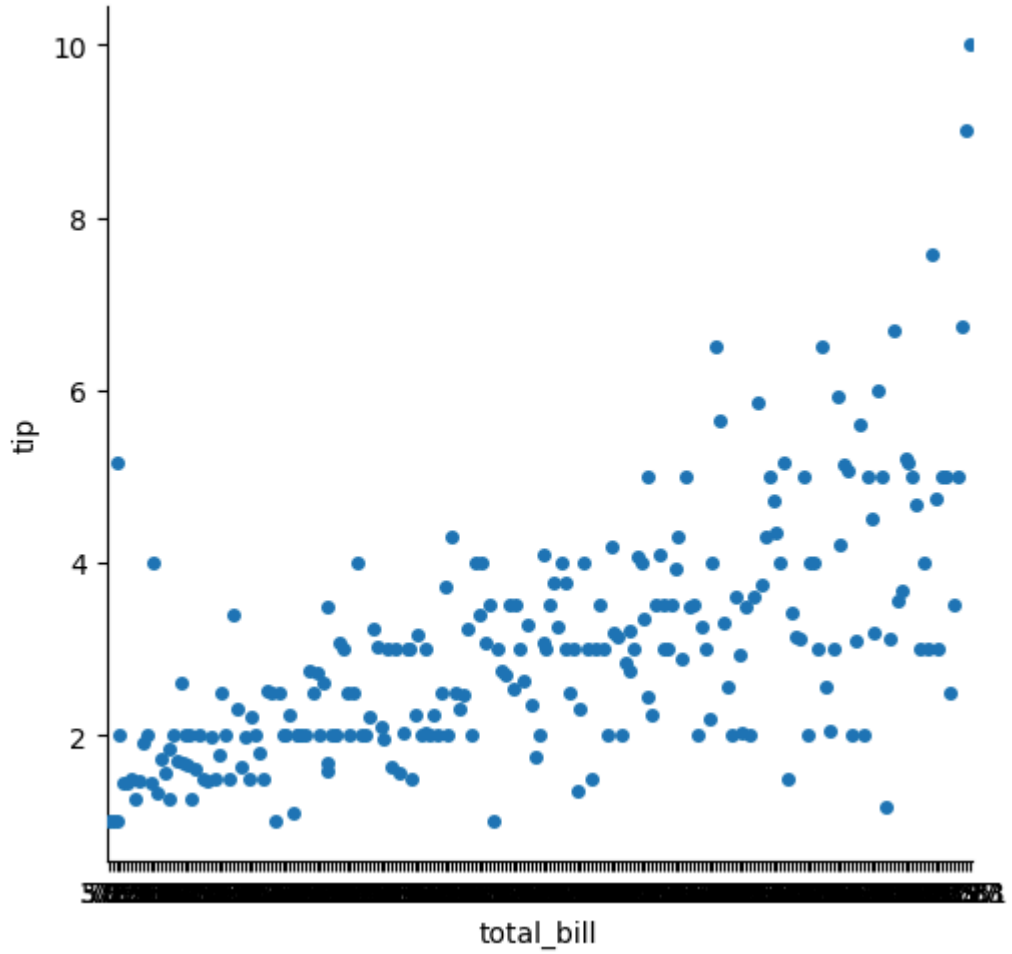
OUTPUT:



3)

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

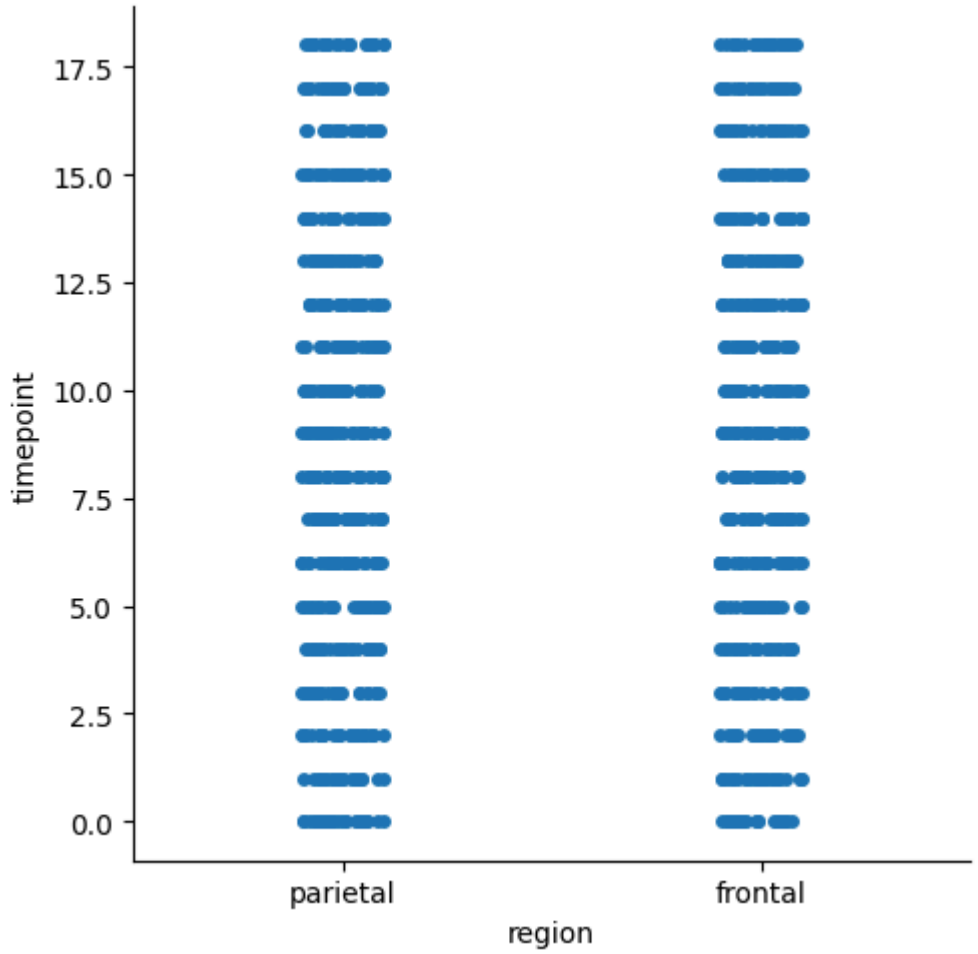
OUTPUT:



4)

```
sns.catplot(data=fmri,x="region",y="timepoint")#catplot it gives
category vise
```

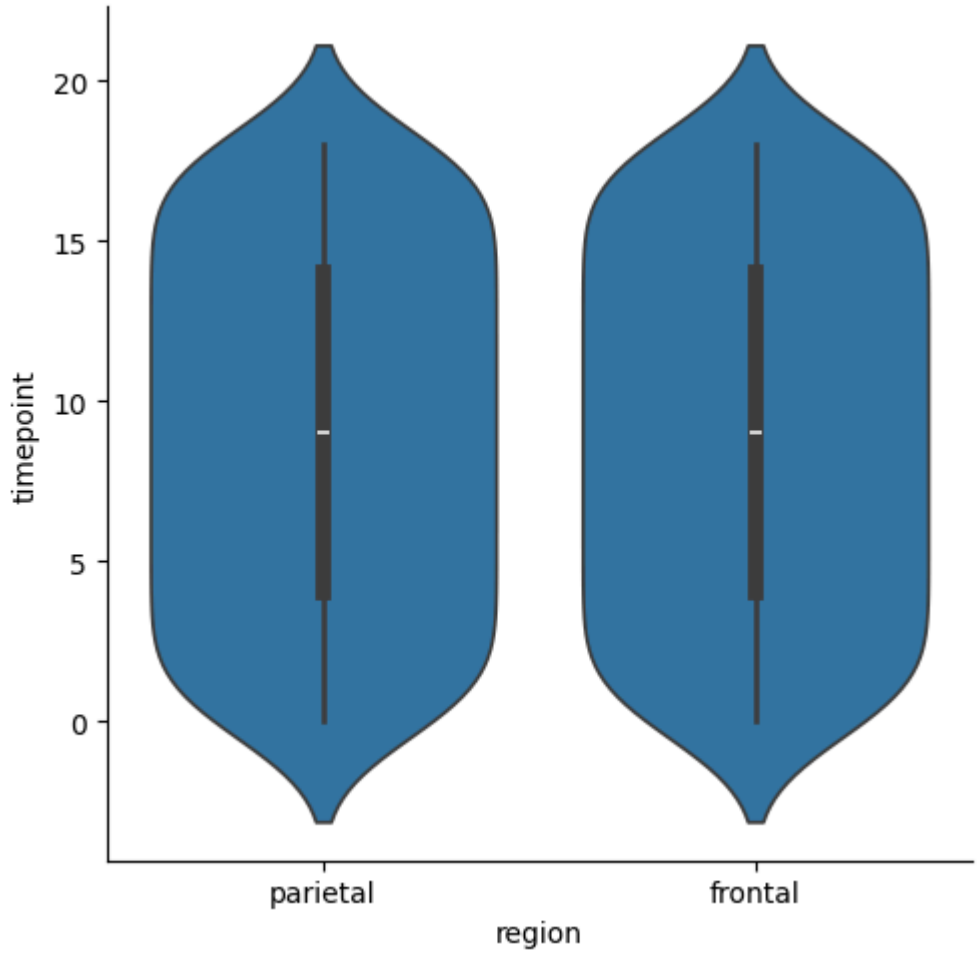
OUTPUT:



5)

```
sns.catplot(data=fmri,x="region",y="timepoint",kind="violin")
```

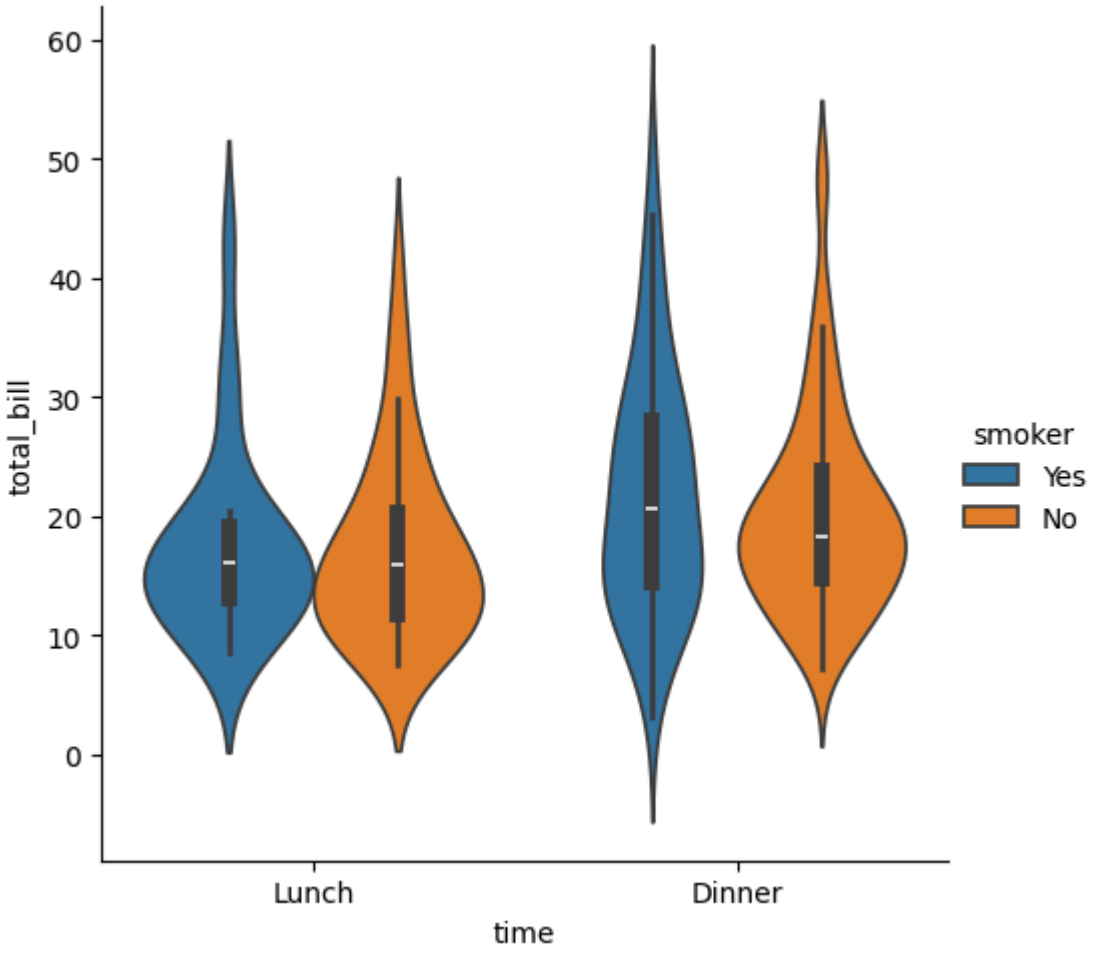
OUTPUT:



6)

```
sns.catplot(data=tips,x="time",y="total_bill",kind="violin",hue="smoker
")
```

OUTPUT:



WEBSCRAPPING

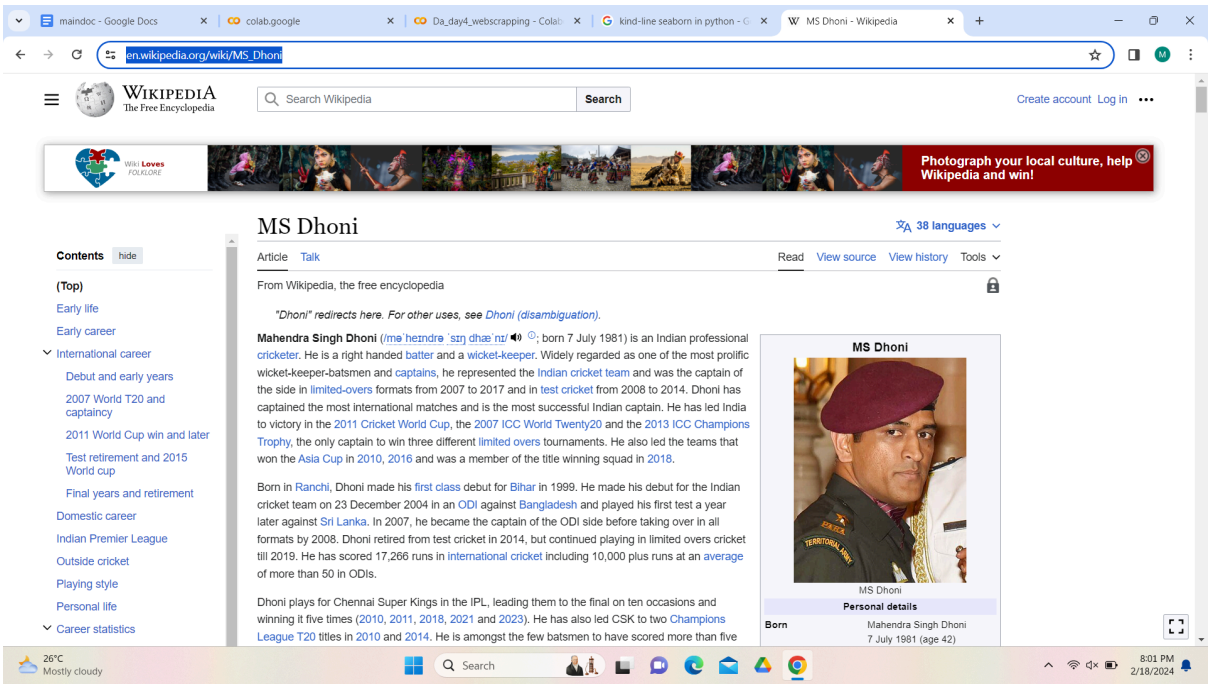
Python is popular for web scraping owing to the abundance of third-party libraries that can scrap complex HTML structures, parse text, and interact with HTML form. Here, we've listed some top Python web scraping libraries.

IMPORTING LIBRARIES:

Urllib3: is a powerful HTTP client library for Python. This makes it easy to perform HTTP requests programmatically. It handles HTTP headers, retries, redirects, and other low-level details, making it an excellent library for web scraping. It also supports SSL verification, connection pooling, and proxying

BeautifulSoup: allows you to parse HTML and XML documents. Using API, you can easily navigate through the HTML document tree and extract tags, meta titles, attributes, text, and other content. BeautifulSoup is also known for its robust error handling.

WEBSCRAPPING: PROBLEM1: EXTRACTING HTMLL CODE OF ANY WEBSITE:




```
#import libraries
from bs4 import BeautifulSoup
import requests
url=("https://en.wikipedia.org/wiki/MS_Dhoni")#save the url
```

Requests: is a simple yet powerful Python library for making HTTP requests. It is designed to be easy to use and intuitive, with a clean and consistent API. With Requests, you can easily send GET and POST requests, and handle cookies, authentication, and other HTTP features. It is also widely used in web scraping due to its simplicity and ease of use.

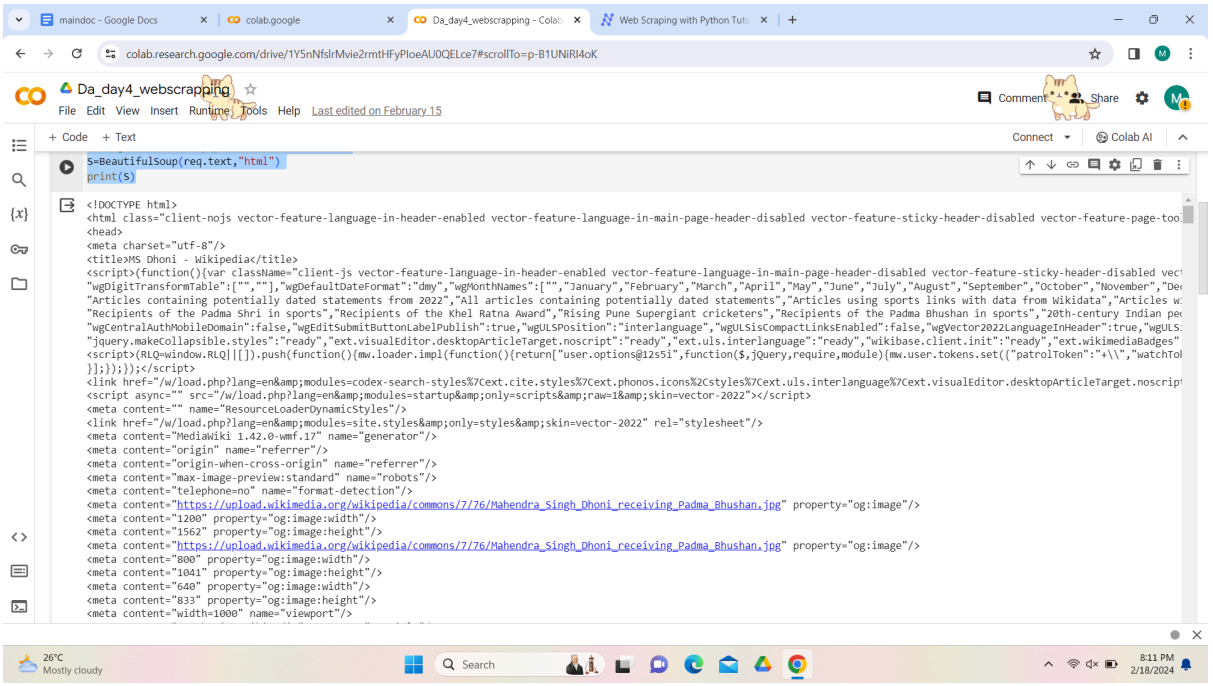
```
req=requests.get(url)#using this we can access the webpage
```

EXAMPLE:

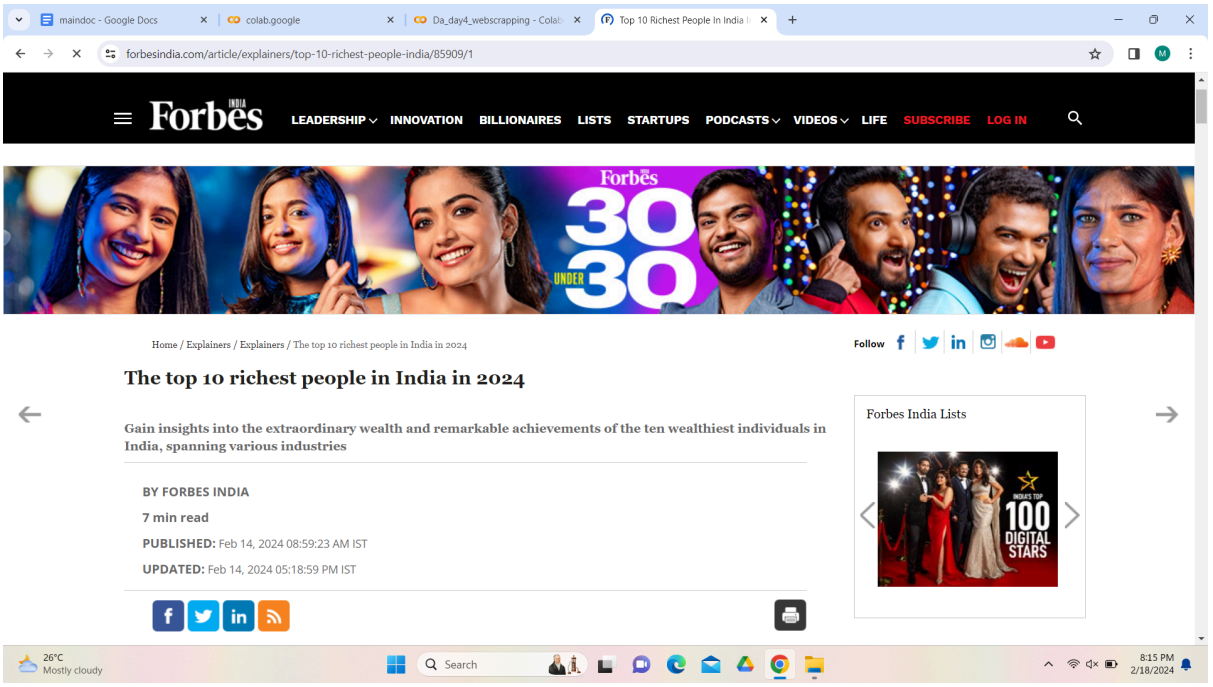
1)

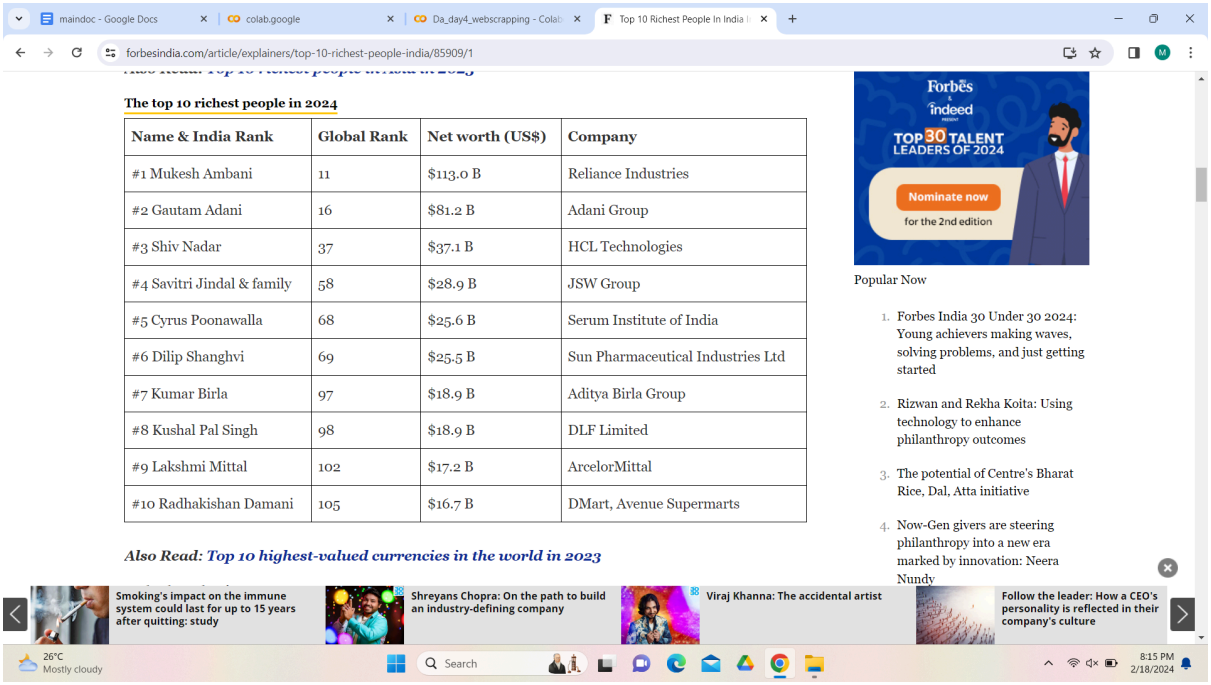
```
#using BeautifulSoup(),access the html code

S=BeautifulSoup(req.text,"html")

print(S)
```

OUTPUT:



EXAMPLE:

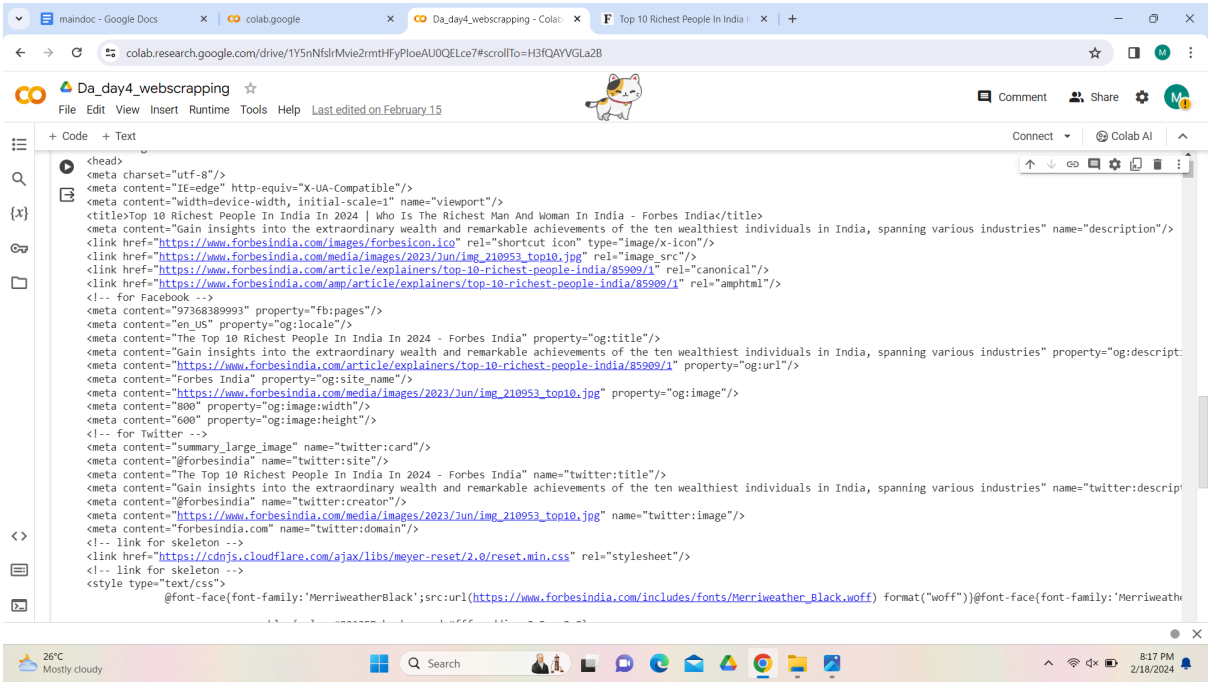




2)

```
url2=("https://www.forbesindia.com/article/explainers/top-10-richest-people-india/85909/1")

req2=requests.get(url2)

a=BeautifulSoup(req2.text,"html")

print(a)
```

OUTPUT:



PROBLEM 2):

TEXT:

Let's take a look at the step-by-step process of using Python to scrape website data.

Step 1: Choose the Website and Webpage URL. ...

Step 2: Inspect the website. ...

Step 3: Installing the important libraries. ...

Step 4: Write the Python code. ...

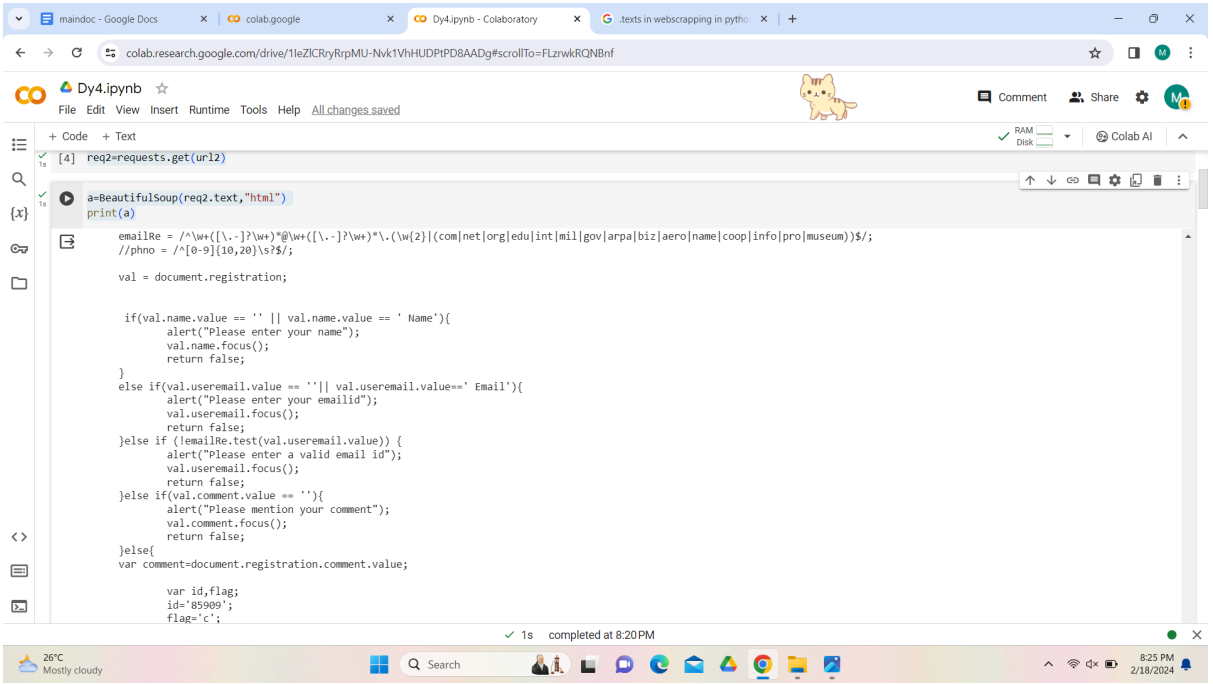
Step 5: Exporting the extracted data. ...

Step 6: Verify the extracted data.

```
#import libraries

from bs4 import BeautifulSoup
```

```
import requests


import pandas as pd




#save the url


url2=("https://www.forbesindia.com/article/explainers/top-10-richest-people-india/85909/1")


req2=requests.get(url2)


a=BeautifulSoup(req2.text,"html")


print(a)
```

OUTPUT:



th=table

tr=table row

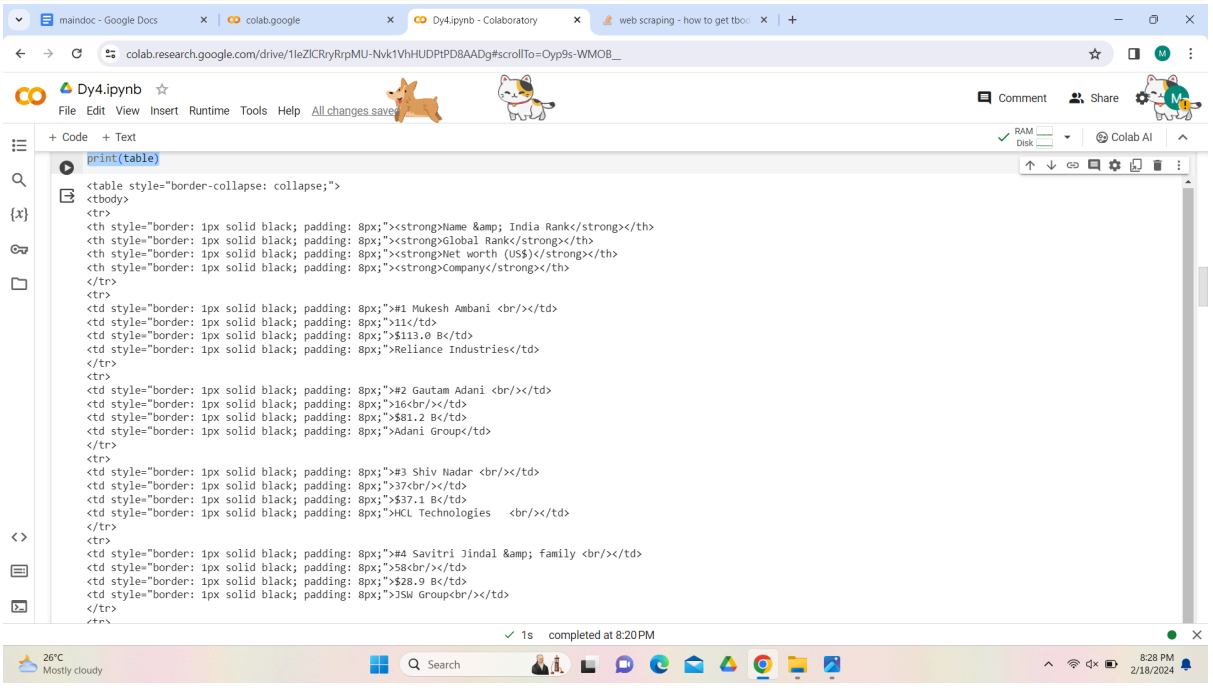
tbody=table body

using,find access the table

EXAMPLE:

```
table=a.find('table')

print(table)
```

OUTPUT:



2)

```
#using .text extract only the text(removing html tags)

headings=table.find_all('th')

head=[i.text for i in headings]

print(head)
```

OUTPUT:

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

3)

```
head=[i.text for i in headings]
#creating dataframes using pandas
df=pd.DataFrame(columns=head)
print(df)
```

OUTPUT:

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

4)

```
print(df.iloc[1:8])
```

OUTPUT:

```
Empty DataFrame

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

5)

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

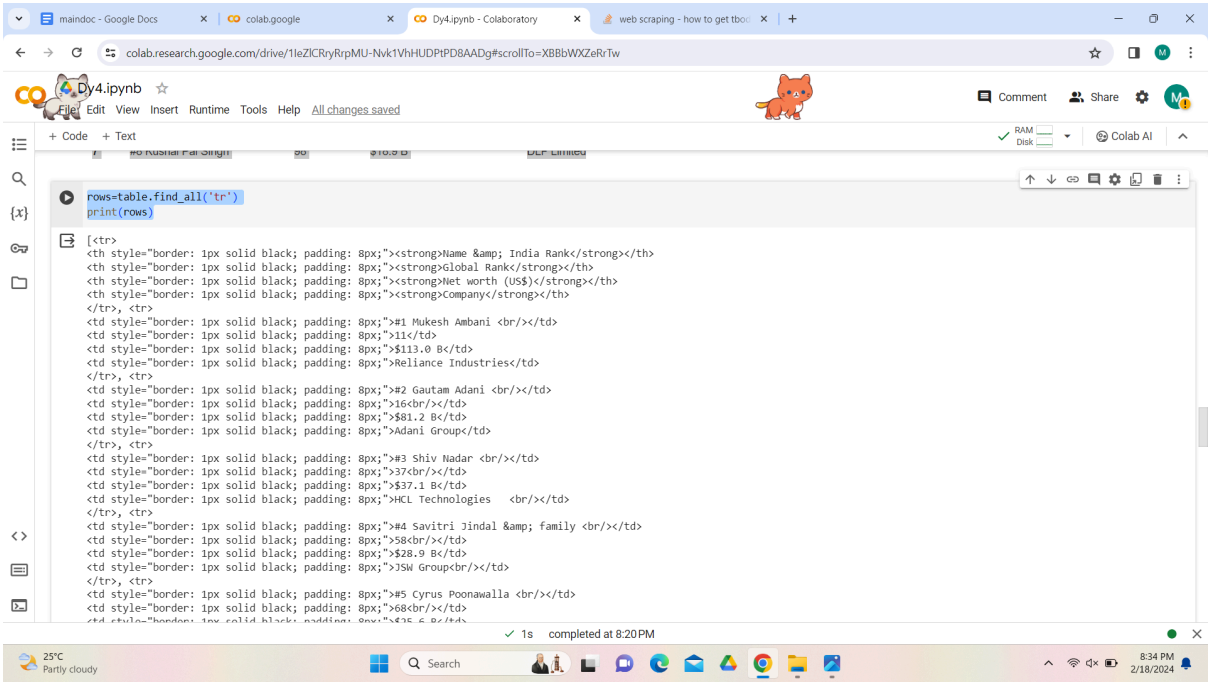
OUTPUT:

	Name & India Rank	Global Rank	Net worth (US$)	Company	
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	

6)

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

OUTPUT:



7)

```
rows=table.find_all('tr')
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']8)
```

SELINIUM

ACCESSING ELEMENTS:

*Go too webpage

* right click→inspect

*click on *now hover on the element you want to access

*tha html tags/code for that specific elements in highlighted

*extract te element names and import in your program using xpath or by id

PROJECT:

Extracting a table from html code of any website and saving it as csv file:

Program flow:

1)import libraries

2)save the url
3)using requests.get(),access the webpage
4)using beautifulsoup(,access the html code
5)using find,access the table
6)using find_all, access the rows o the table
7)using.text,extract only the text(removing html tags)
8)create a dataframes using pandas
9)push all extracted data into dataframes
10) using to _csv,save the data frame in .csv format

SELINIUM:

*used to scrap data from dynamic website
*subpackage:webdriver

FUNCTION	PURPOSE	ATTRIBUTES
Chrome options()	Create an instance of chrome	
.get	Access webpage	'url'
find_element	To find first element of a kind	By.ID By.XPATH
.click()	To click a button in a webpage	

EXAMPLES:

1)

project 3 extacting dell laptops data from amazon.in website and saving its as.csv

```
In [1]: !pip install selenium
```

```
Requirement already satisfied: selenium in c:\users\hp\anaconda3\lib\site-packages (4.17.2)
Requirement already satisfied: urllib3[socks]<3,>=1.26 in c:\users\hp\anaconda3\lib\site-packages (from selenium) (1.26.16)
Requirement already satisfied: trio~=0.17 in c:\users\hp\anaconda3\lib\site-packages (from selenium) (0.24.0)
Requirement already satisfied: trio-websocket~=0.9 in c:\users\hp\anaconda3\lib\site-packages (from selenium) (0.11.1)
Requirement already satisfied: certifi>=2021.10.8 in c:\users\hp\anaconda3\lib\site-packages (from selenium) (2023.7.22)
Requirement already satisfied: typing_extensions>=4.9.0 in c:\users\hp\anaconda3\lib\site-packages (from selenium) (4.9.0)
Requirement already satisfied: attrs>=20.1.0 in c:\users\hp\anaconda3\lib\site-packages (from trio~=0.17->selenium) (22.1.0)
Requirement already satisfied: sortedcontainers in c:\users\hp\anaconda3\lib\site-packages (from trio~=0.17->selenium) (2.4.0)
Requirement already satisfied: idna in c:\users\hp\anaconda3\lib\site-packages (from trio~=0.17->selenium) (3.4)
Requirement already satisfied: outcome in c:\users\hp\anaconda3\lib\site-packages (from trio~=0.17->selenium) (1.3.0.post0)
Requirement already satisfied: sniffio>=1.3.0 in c:\users\hp\anaconda3\lib\site-packages (from trio~=0.17->selenium) (1.3.0)
Requirement already satisfied: cffi>=1.14 in c:\users\hp\anaconda3\lib\site-packages (from trio~=0.17->selenium) (1.15.1)
Requirement already satisfied: wsproto>=0.14 in c:\users\hp\anaconda3\lib\site-packages (from trio-websocket~=0.9->selenium)
(1.2.0)
Requirement already satisfied: PySocks!=1.5.7,<2.0,>=1.5.6 in c:\users\hp\anaconda3\lib\site-packages (from urllib3[socks]<3,>=
1.26->selenium) (1.7.1)
Requirement already satisfied: pycparser in c:\users\hp\anaconda3\lib\site-packages (from cffi>=1.14->trio~=0.17->selenium) (2.
21)
Requirement already satisfied: h11<1,>=0.9.0 in c:\users\hp\anaconda3\lib\site-packages (from wsproto>=0.14->trio-websocket~=0.
9->selenium) (0.14.0)
```

2)

INSTALLATION OF WEBDRIVER:

```
In [2]: !pip install webdriver_manager
```

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

3)INSTALLATION OF PANDAS:

```
In [3]: !pip install pandas
```

```
Requirement already satisfied: pandas in c:\users\hp\anaconda3\lib\site-packages (2.0.3)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\hp\anaconda3\lib\site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\hp\anaconda3\lib\site-packages (from pandas) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in c:\users\hp\anaconda3\lib\site-packages (from pandas) (2023.3)
Requirement already satisfied: numpy>=1.21.0 in c:\users\hp\anaconda3\lib\site-packages (from pandas) (1.24.3)
Requirement already satisfied: six>=1.5 in c:\users\hp\anaconda3\lib\site-packages (from python-dateutil>=2.8.2->pandas) (1.16.
0)
```

EXAMPLES:

1)

```
In [9]: import selenium
```

```
In [10]: from selenium import webdriver
```

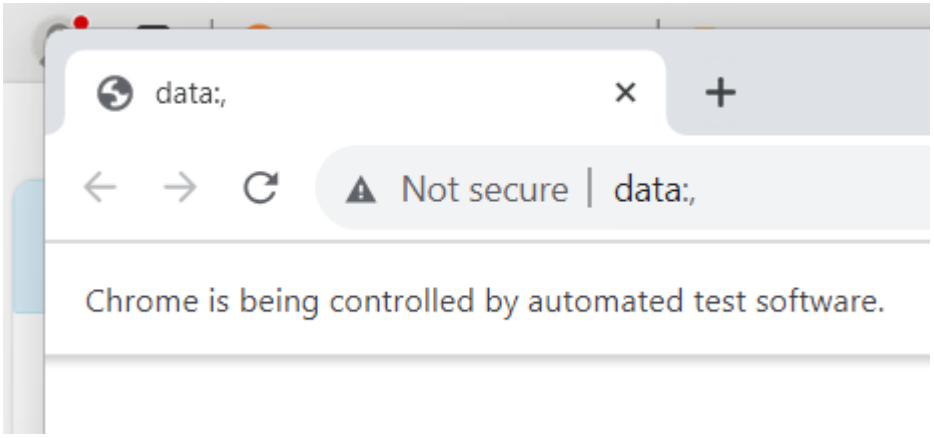
```
In [11]: from selenium.webdriver.chrome.options import Options
         from webdriver_manager.chrome import ChromeDriverManager
         from selenium.webdriver.common.by import By
         from selenium.webdriver.common.keys import Keys
```

2)

```
In [12]: #define options and sewt browser capabilities
         options=webdriver.ChromeOptions()
         options.add_argument('--some-option')
         #create webdriver.chrome(options=options)
         driver=webdriver.Chrome(options=options)
         #access browser capabilities
         browser_name=options.to_capabilities()["browserName"]
         print(browser_name)
```

```
chrome
```

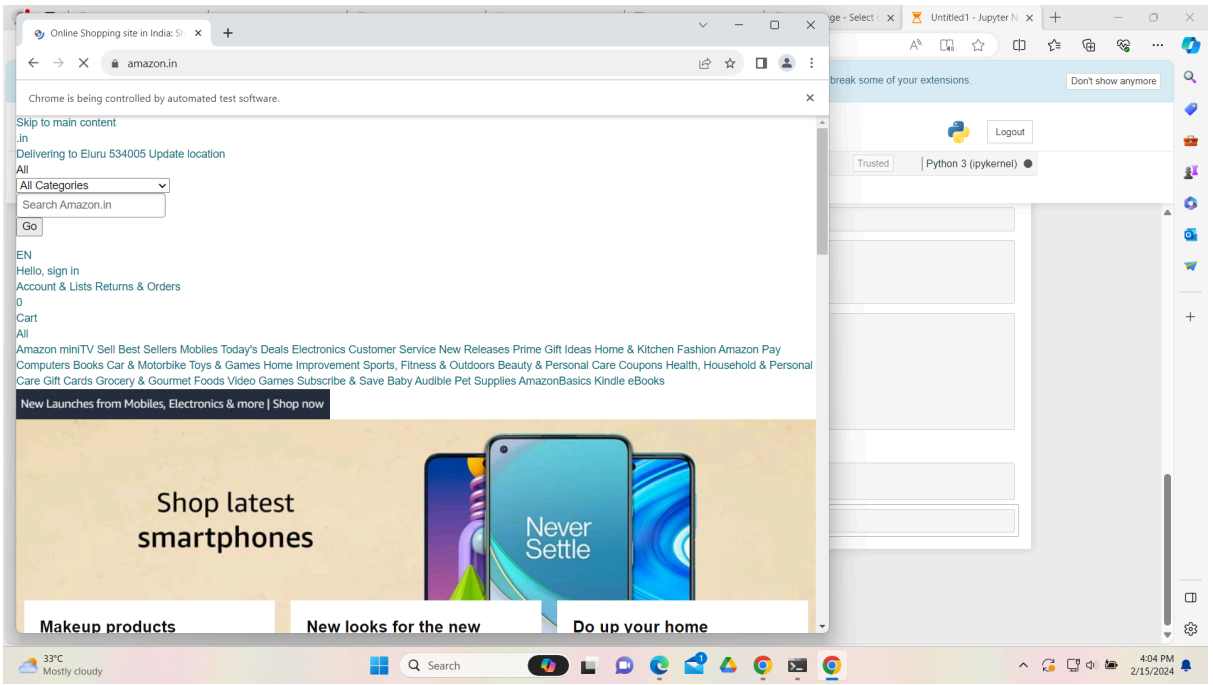
OUTPUT:



3)

```
In [17]: #Navigate to a website
         driver.get('https://www.amazon.in/')

In [18]: search=driver.find_element(By.ID,"twotabsearchtextbox") i
```

OUTPUT:

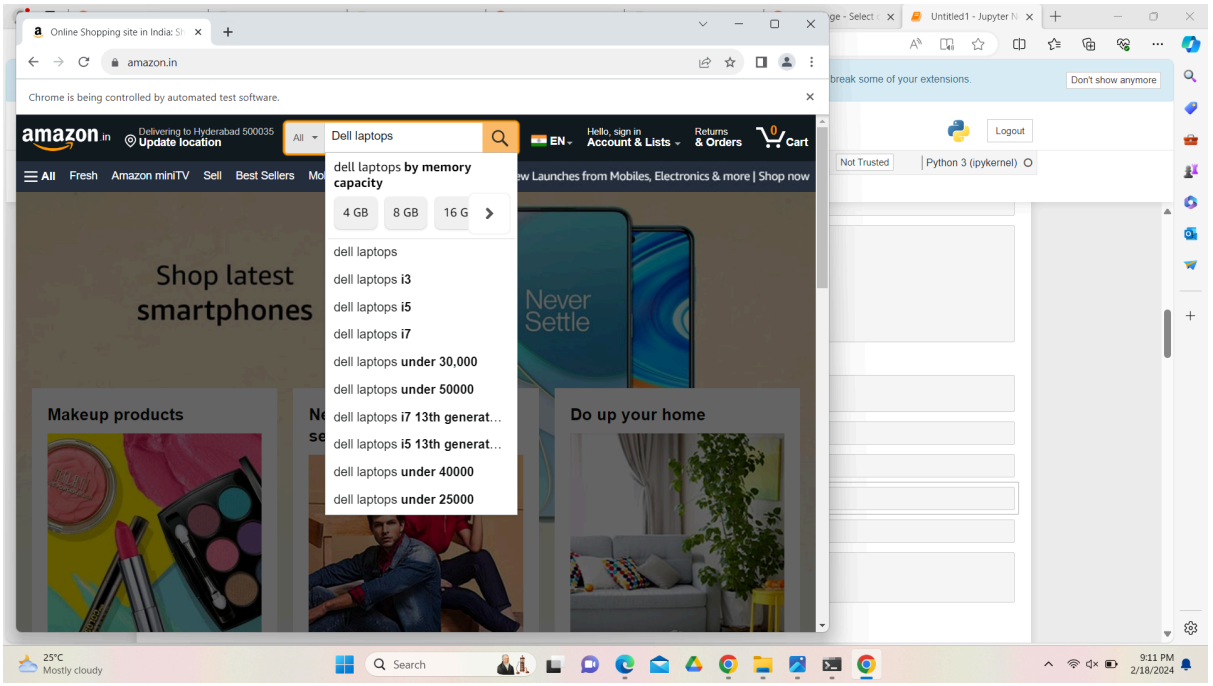


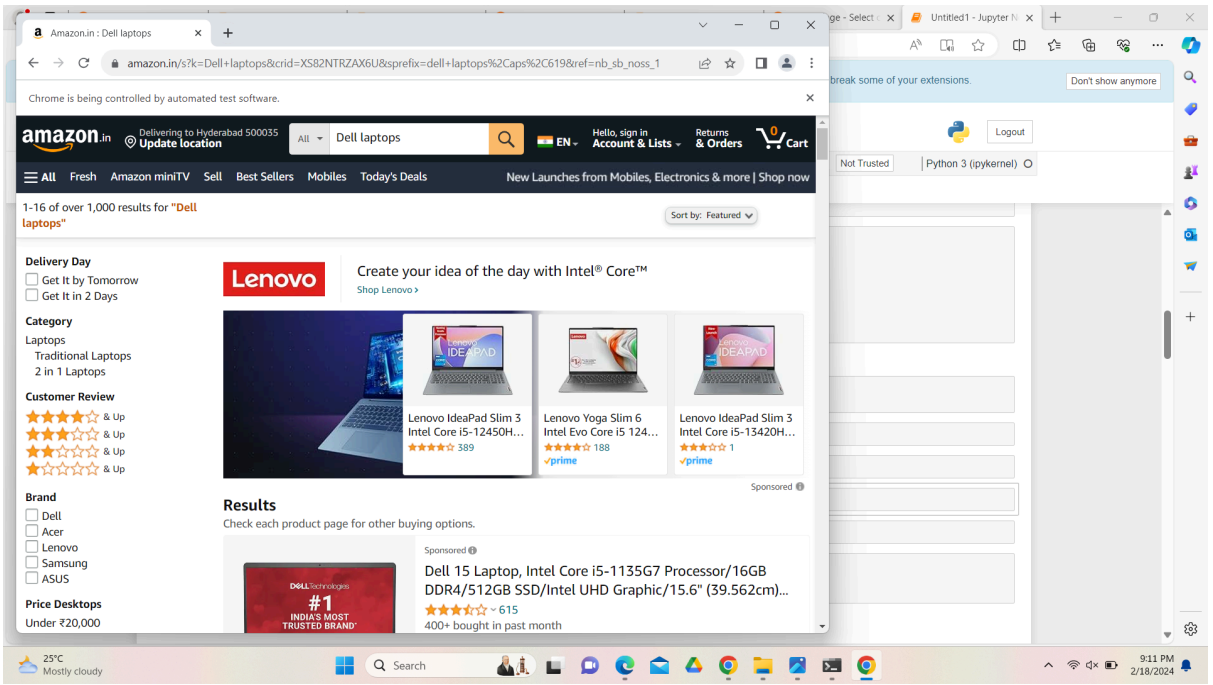
4)

```
In [18]: search=driver.find_element(By.ID,"twotabsearchtextbox") i

In [19]: search.send_keys("Dell laptops")

In [21]: driver.find_element(By.ID,"nav-search-submit-button").click()
```

```
<input type="text" id="twotabsearchtextbox" value="" name="field-keywords"
autocomplete="off" placeholder="Search Amazon.in" class="nav-input
nav-progressive-attribute" dir="auto" tabindex="0" aria-label="Search Amazon.in"
spellcheck="false">

<input id="nav-search-submit-button" type="submit" class="nav-input
nav-progressive-attribute" value="Go" tabindex="0">
```

OUTPUT:





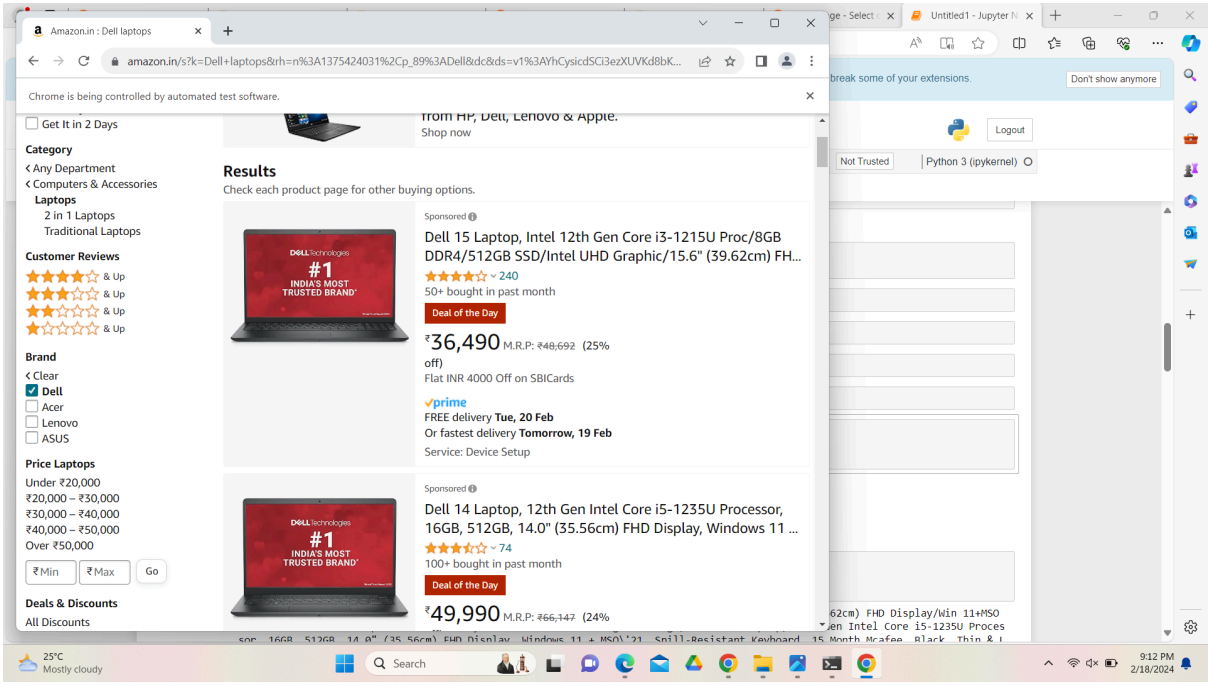
5)

```
In [26]: driver.find_element(By.XPATH,"//span[text()='Dell']").click()

In [29]: laptop_name=[]
         laptop_prices=[]
         review=[]
```

Type *Markdown* and LaTeX: α^2

Type *Markdown* and LaTeX: α^2

OUTPUT:

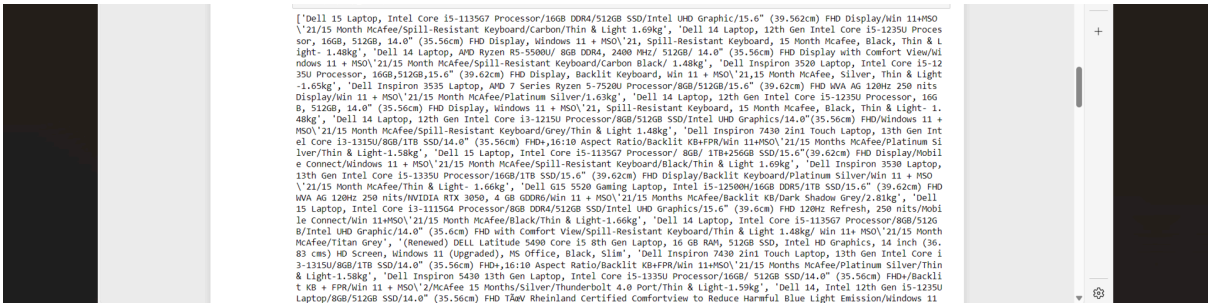


6)

Type *Markdown* and LaTeX: α^2

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

OUTPUT:



```
['Dell 15 Laptop, Intel Core i5-1135G7 Processor/16GB DDR4/512GB SSD/Intel UHD Graphic/15.6" (39.562cm) FHD Display/Win 11+MSO
\'21/15 Month McAfee/Spill-Resistant Keyboard/Carbon/Thin & Light 1.69kg', 'Dell 14 Laptop, 12th Gen Intel Core i5-1235U Proces
sor, 16GB, 512GB, 14.0" (35.56cm) FHD Display, Windows 11 + MSO\'21, Spill-Resistant Keyboard, 15 Month Mcafee, Black, Thin & L
ight- 1.48kg', 'Dell 14 Laptop, AMD Ryzen R5-5500U/ 8GB DDR4, 2400 MHz/ 512GB/ 14.0" (35.56cm) FHD Display with Comfort View/Wi
ndows 11 + MSO\'21/15 Month McAfee/Spill-Resistant Keyboard/Carbon Black/ 1.48kg', 'Dell Inspiron 3520 Laptop, Intel Core i5-12
35U Processor, 16GB,512GB,15.6" (39.62cm) FHD Display, Backlit Keyboard, Win 11 + MSO\'21,15 Month McAfee, Silver, Thin & Light
-1.65kg', 'Dell Inspiron 3535 Laptop, AMD 7 Series Ryzen 5-7520U Processor/8GB/512GB/15.6" (39.62cm) FHD WVA AG 120Hz 250 nits
Display/Win 11 + MSO\'21/15 Month McAfee/Platinum Silver/1.63kg', 'Dell 14 Laptop, 12th Gen Intel Core i5-1235U Processor, 16G
B, 512GB, 14.0" (35.56cm) FHD Display, Windows 11 + MSO\'21, Spill-Resistant Keyboard, 15 Month Mcafee, Black, Thin & Light- 1.
48kg', 'Dell 14 Laptop, 12th Gen Intel Core i3-1215U Processor/8GB/512GB SSD/Intel UHD Graphics/14.0"(35.56cm) FHD/Windows 11 +
MSO\'21/15 Month McAfee/Spill-Resistant Keyboard/Grey/Thin & Light 1.48kg', 'Dell Inspiron 7430 2in1 Touch Laptop, 13th Gen Int
el Core i3-1315U/8GB/1TB SSD/14.0" (35.56cm) FHD+,16:10 Aspect Ratio/Backlit KB+FPR/Win 11+MSO\'21/15 Months McAfee/Platinum Si
lver/Thin & Light-1.58kg', 'Dell 15 Laptop, Intel Core i5-1135G7 Processor/ 8GB/ 1TB+256GB SSD/15.6"(39.62cm) FHD Display/Mobil
e Connect/Windows 11 + MSO\'21/15 Month McAfee/Spill-Resistant Keyboard/Black/Thin & Light 1.69kg', 'Dell Inspiron 3530 Laptop,
13th Gen Intel Core i5-1335U Processor/16GB/1TB SSD/15.6" (39.62cm) FHD Display/Backlit Keyboard/Platinum Silver/Win 11 + MSO
\'21/15 Month McAfee/Thin & Light- 1.66kg', 'Dell G15 5520 Gaming Laptop, Intel i5-12500H/16GB DDR5/1TB SSD/15.6" (39.62cm) FHD
WVA AG 120Hz 250 nits/NVIDIA RTX 3050, 4 GB GDDR6/Win 11 + MSO\'21/15 Months McAfee/Backlit KB/Dark Shadow Grey/2.81kg', 'Dell
15 Laptop, Intel Core i3-1115G4 Processor/8GB DDR4/512GB SSD/Intel UHD Graphics/15.6" (39.6cm) FHD 120Hz Refresh, 250 nits/Mobi
le Connect/Win 11+MSO\'21/15 Month McAfee/Black/Thin & Light-1.66kg', 'Dell 14 Laptop, Intel Core i5-1135G7 Processor/8GB/512G
B/Intel UHD Graphic/14.0" (35.6cm) FHD with Comfort View/Spill-Resistant Keyboard/Thin & Light 1.48kg/ Win 11+ MSO\'21/15 Month
McAfee/Titan Grey', '(Renewed) DELL Latitude 5490 Core i5 8th Gen Laptop, 16 GB RAM, 512GB SSD, Intel HD Graphics, 14 inch (36.
83 cms) HD Screen, Windows 11 (Upgraded), MS Office, Black, Slim', 'Dell Inspiron 7430 2in1 Touch Laptop, 13th Gen Intel Core i
3-1315U/8GB/1TB SSD/14.0" (35.56cm) FHD+,16:10 Aspect Ratio/Backlit KB+FPR/Win 11+MSO\'21/15 Months McAfee/Platinum Silver/Thin
& Light-1.58kg', 'Dell Inspiron 5430 13th Gen Laptop, Intel Core i5-1335U Processor/16GB/ 512GB SSD/14.0" (35.56cm) FHD+/Backli
t KB + FPR/Win 11 + MSO\'2/McAfee 15 Months/Silver/Thunderbolt 4.0 Port/Thin & Light-1.59kg', 'Dell 14, Intel 12th Gen i5-1235U
Laptop/8GB/512GB SSD/14.0" (35.56cm) FHD TÃ¤V Rheinland Certified Comfortview to Reduce Harmful Blue Light Emission/Windows 11
```