**Data Analysis with Python**

1.**concept overview-python**

1.1 **variable:**

Container to store values

Code:

A=5

1.2 **print:**

It is function which is used to display

Code:

a=5

print("i am",a,"years old")

Output:

i am 5 years old

1.3 **operators:**

Operators in general are used to perform operations on values

and variables.

1.3.1 **Arthemetic operators**:

Arithmetic Operators are used to perform mathematical operations like addition, subtraction, multiplication, and division.

Code:

#arthemetic

a=25

b=5

print(a+b)

print(a-b)

print(a/b)

print(a//b)

print(a\*\*b)

Output:

30

20

5.0

5

9765625

**1.3.2** **Relational operators:**Relational operators in python are used to compare the operand values on either side**.** The relational operators in Python return a boolean value i.e., either True or False, based on the value of operands.

Code:

#relational

a=5

b=7

print(a==b)

print(a>b)

print(a<b)

print(a!=b)

print(a>=b)

print(a<=b)

Output:

False

False

True

True

False

True

**1.3.3** **Logical operator:**Logical Operators in Python are used to perform logical operations on the values of variables. The value is either true or false.

Code:

a=34

b=7

print(a>b and a<b)

print(a>b or a<b)

Output:

False

True

**1.3.4** **Membership operator:**Membership operators are operators used to validate the membership of a value. The 'in' operator is used to check if a value exists in a sequence or not.

Code:

a="dp"

print("x" in a)

**Output:**

False

**1.4** **control flow-conditional statements**

It decide the direction (Control Flow) of the flow of program execution.

1.4.1 **Simple If:**

Code:

a=7

print("positive")

Output:

positive

1.4.2 **If Else:**

Code:

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

if(a>0):

print("positive")

else:

print("negative")

Output:

Enter a value 34

positive

1.4.3 **Else if ladder:**

Code:

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

if(a>0):

print("positive")

elif(a==0):

print("neutral")

else:

print("negative")

Output:

Enter a value 0

neutral

**1.5 Control flow-loop statements**

**1.5.1 for loop:**

It is used to repeatedly execute a group of statements

as long as the condition is satisfied.

Code:

#develop a program on 7th table using for loop

a=7

for i in range(1,10,1):

b=a\*i

print(b)

Output:

7

14

21

28

35

42

49

56

63

**1.5.2 while loop:**

It is used to repeatedly execute a block of code untill the

condition is satisfied.

**Code:**

**a=int(input("enter a value"))**

**i=1**

**while(i<=5):**

**b=a\*i**

**i=i+1**

**print(b)**

**Output:**

**enter a value2**

**2**

**4**

**6**

**8**

**10**

**1.6 Data Slicing:**

Slicing is the extraction of a part of a string,list or tuple.it enables the users to access the specific range of elements.

**Syntax:**

List\_name[start,stop,step]

**Code:**

a="sri vasavi engineering college"

print(a[::-1])

print(a[4:7:1])

Output:

egelloc gnireenigne ivasav irs

vas

**1.7 Type Casting:**

In Python, Type Casting is a process in which we convert a variables of one type to another.these conversions can be implicit or explicit.

**Code:**

**a=8**

**b=9.1**

**print(a+b)#implict call**

**print(int(a+b))#explicit call**

**Output:**

**17.1**

**17**

**1.8 Collections\_list:**

* List is a collection of elements.
* List is heterogenous.
* List is mutable.

Code:

l1=[56,"dp",10.8,3]

print(l1)

Output:

[56, 'dp', 10.8, 3]

* Append :

Code:

a=75

l1.append(a)

print(l1)

* Insert :

Code:

b="hai"

l1.insert(3,b)

print(l1)

Output:

[56, 'dp', 10.8, 'hai', 'b', 3, 75, 75, 75]

* Extend:

l2=["dp",45]

l1.extend(l2)

print(l1)

Output:

[56, 'dp', 10.8, 'hai', 'b', 3, 75, 75, 75, 'gopika', 45, 'gopika', 45]

* Pop :

Code:

l1.pop(7)

print(l1)

Output:

[56, 'dp', 10.8, 'hai', 'b', 3, 75, 75, 'gopika', 45, 'gopika', 45]

* Remove:

Code:

l1.remove(75)

print(l1)

Output:

[56, 'dp', 10.8, 'hai', 'b', 3, 75, 'gopika', 45, 'gopika', 45]

* list comprehension

1. iterations

2. applies some functions on every element

3.conitions

4. Output:list

#squaring the elements in the list

l1=[100,200,300,400]

l2=[i\*\*2 for i in l1]

print(l2)

OUTPUT:

[10000, 40000, 90000, 160000]

#squaring the elements in the list which is greater than 200

l1=[100,200,300,400]

l2=[i\*\*2 for i in l1 if i>200]

print(l2)

Output:

[90000, 160000]

#the salaries of 5 employees in a company is taken in a list.the tax is 10% if the salary is lessthan or equal to 50000 or it is 15% create a list with tax amounts #[67000,45000,89000,34000,50000]

list-name=[(body of if) (condition) else (body of else) itearator]

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

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

#using list comprehension with same example

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:**

Numpy is a python library used for working with numerical data in python.

There are different sub packages in numpy.

Install:

!pip install np

#importing:

import numpy as np

#import ram

#random module is subpackage of numpy

**#creating 1-D 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.

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

print(type(A))

Output:

<class 'numpy.ndarray'>

**#creating 2-D 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.

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

print(b)

Output:

[[2 3 4]

[4 5 6]]

**#creating 3-D 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.

C=np.array([[[2,3,4],[5,6,7]],[[1,8,9],[0,4,5]]])

print(C)

Output:

[[[2 3 4]

[5 6 7]]

[[1 8 9]

[0 4 5]]]

**#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

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

#groups,rows,columns,2D

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

print(e)

Output:

[[1. 1.]

[1. 1.]

[1. 1.]]

**#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

f=np.zeros((4,4))

print(f)

Output:

[[0. 0. 0. 0.]

[0. 0. 0. 0.]

[0. 0. 0. 0.]

[0. 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

b=np.arange(3,31,3)

print(b)

Output:

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

Linspace:It is a function that generates a sequence of evenly spaced

numbers over specified range

#linspace

r=np.linspace(24,10,3)

print(r)

Output:

[24. 17. 10.]

Reshape:This function is used to reshape an array into a given shape

without changing data.

y=np.arange(1,7).reshape(2,3)

print(y)

q=np.arange(9,15).reshape(2,3)

print(q)

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=1

g=np.sum((A,B),axis=1)#ROWS FIRST

print(g)

Output:

[[4 4]

[6 6]]

#sum function using axis=0

g=np.sum((A,B),axis=0)#columns first

print(g)

Output:

[[3 3]

[7 7]]

b=np.array(25,289,361,81)

#find square rooots and iterate through the results values. output:5 square is 25

b=np.array([25,289,361,81])

for i in b:

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

Output:

5.0 square is 25

17.0 square is 289

19.0 square is 361

9.0 square is 81

#array joins

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))) #columns

print("\n")

output:[[34 35 36]

[37 38 39]

[ 4 5 6]

[ 7 8 9]]

#array joins

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.hstack((a,b))) #rows

print("\n")

Output:

[[34 35 36 4 5 6]

[37 38 39 7 8 9]]

#array joins

a=np.arange(30).reshape(2,3,5)

print(a)

print("output of dstack")

print(np.dstack(a))

#noof rows becomes noof groups

#colums 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]]]

#creating an array of size(4,8)

b=np.arange(2,34).reshape(8,4)

print(b)

Output:

[[ 2 3 4 5]

[ 6 7 8 9]

[10 11 12 13]

[14 15 16 17]

[18 19 20 21]

[22 23 24 25]

[26 27 28 29]

[30 31 32 33]]

#random array

c=np.random.rand(8,4) #randrange is between 0 1nd 1

print(c)

Output:

[[0.41710006 0.2466779 0.42689413 0.37424698]

[0.00386473 0.75278467 0.64264913 0.53306426]

[0.37000206 0.49605354 0.70199896 0.30336513]

[0.33565892 0.42109433 0.76963036 0.55384553]

[0.39601839 0.72165219 0.87305478 0.37336139]

[0.80691092 0.20345904 0.57198638 0.32002396]

[0.35827043 0.66107693 0.97707299 0.35375784]

[0.63987243 0.06149391 0.00472188 0.13097827]]

c=10\*np.random.rand(8,4) #randrange is between 0 1nd 1

print(c)

#random array

c=10\*np.random.rand(8,4) #randrange is between 0 1nd 1

print(c)

Output:

[[4.26000300e+00 4.00481007e+00 4.97987841e+00 5.46368804e+00]

[1.65131660e+00 6.05527964e+00 3.76533188e+00 1.90646048e+00]

[1.69403655e+00 5.79061343e-03 9.63423375e+00 5.62560210e+00]

[4.45537268e+00 6.36689456e-01 1.76221251e+00 5.36279923e+00]

[6.27725604e+00 4.09478447e+00 4.43877068e+00 5.19779819e+00]

[5.43818322e+00 3.53624451e-01 5.58745598e+00 1.75436914e-02]

[5.06053128e+00 6.40999956e+00 1.02408020e+00 2.69838000e+00]

[6.28502095e+00 5.65419310e-01 3.70093726e+00 3.31217035e+00]]

#random array

c=np.floor(10\*np.random.rand(8,4)) #randrange is between 0 1nd 1

print(c)

Output:

[[4. 6. 7. 2.]

[3. 1. 5. 1.]

[2. 4. 9. 5.]

[9. 8. 1. 4.]

[7. 8. 2. 3.]

[8. 8. 6. 5.]

[5. 4. 5. 3.]

[6. 3. 6. 5.]]

Vsplit:

split vertically cutting the array in vertical position

np.vsplit(c,4)

Output:

[array([[4., 6., 7., 2.],

[3., 1., 5., 1.]]),

array([[2., 4., 9., 5.],

[9., 8., 1., 4.]]),

array([[7., 8., 2., 3.],

[8., 8., 6., 5.]]),

array([[5., 4., 5., 3.],

[6., 3., 6., 5.]])]

np.vsplit(c,8)

Output:

[array([[4., 6., 7., 2.]]),

array([[3., 1., 5., 1.]]),

array([[2., 4., 9., 5.]]),

array([[9., 8., 1., 4.]]),

array([[7., 8., 2., 3.]]),

array([[8., 8., 6., 5.]]),

array([[5., 4., 5., 3.]]),

array([[6., 3., 6., 5.]])]

#splitting at particular rows 1st and 5th row

np.vsplit(c,(1,5))

Output:

[array([[4., 6., 7., 2.]]),

array([[3., 1., 5., 1.],

[2., 4., 9., 5.],

[9., 8., 1., 4.],

[7., 8., 2., 3.]]),

array([[8., 8., 6., 5.],

[5., 4., 5., 3.],

[6., 3., 6., 5.]])]

#create an array with 4 rows and 8 columns

d=np.arange(2,34).reshape(4,8)

print(d)

Output:

[[ 2 3 4 5 6 7 8 9]

[10 11 12 13 14 15 16 17]

[18 19 20 21 22 23 24 25]

[26 27 28 29 30 31 32 33]]

Horizontal split:

split vertically cutting the array in horizantal position.

np.hsplit(d,4)

Output:

[array([[ 2, 3],

[10, 11],

[18, 19],

[26, 27]]),

array([[ 4, 5],

[12, 13],

[20, 21],

[28, 29]]),

array([[ 6, 7],

[14, 15],

[22, 23],

[30, 31]]),

array([[ 8, 9],

[16, 17],

[24, 25],

[32, 33]])]

np.hsplit(d,2)

Output:

[array([[ 2, 3, 4, 5],

[10, 11, 12, 13],

[18, 19, 20, 21],

[26, 27, 28, 29]]),

array([[ 6, 7, 8, 9],

[14, 15, 16, 17],

[22, 23, 24, 25],

[30, 31, 32, 33]])]

#splitting at particular position

np.hsplit(d,(3,7))

Output:

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

[10, 11, 12],

[18, 19, 20],

[26, 27, 28]]),

array([[ 5, 6, 7, 8],

[13, 14, 15, 16],

[21, 22, 23, 24],

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

array([[ 9],

[17],

[25],

[33]])]

np.hsplit(d,(2,6))

Output:

[array([[ 2, 3],

[10, 11],

[18, 19],

[26, 27]]),

array([[ 4, 5, 6, 7],

[12, 13, 14, 15],

[20, 21, 22, 23],

[28, 29, 30, 31]]),

array([[ 8, 9],

[16, 17],

[24, 25],

[32, 33]])]

**Trigonometry Functions:**

**np.pi**

Output:

3.141592653589793

#creating radians

e=[np.pi/4,np.pi/4,np.pi]

print(e)

Output:

[0.7853981633974483, 0.7853981633974483, 3.141592653589793]

#converting radians into degree

np.rad2deg(e)

Output:

array([ 45., 45., 180.])

#converting radians into degree

e=([ 45., 45., 180.])

np.deg2rad(e)

Output:

array([0.78539816, 0.78539816, 3.14159265])

**Trigonometry values:**

**1.np.cos(1)**

**Output:**

**0.5403023058681398**

**2.np.sin(1)**

**Output:**

**0.8414709848078965**

**3.np.cos(45)**

**Output:**

**0.5253219888177297**

**4.np.sin(45)**

**Output:**

**0.8509035245341184**

**Statistics:**

**#creating an array**

**1.calculating mean**

**p=np.array([25,35,45,55,65,75])**

**np. mean(p)**

**Output:**

50.0

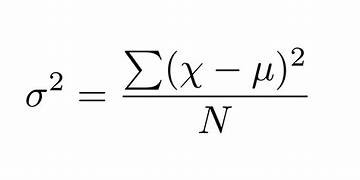
**2.calculating mean**

**p=np.array([25,35,45,55,65,75])**

**np.median(p)**

**3.calculating variance**

**formula:**

****

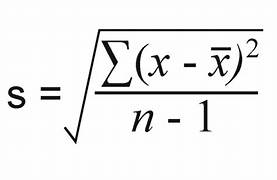
**np.var(p)**

**Output:**

**291.6666666666667**

**4.calculating standard deviation:**

**F**ormula:



np.std(p)

Output:

17.07825127659933

**Creating matrix:**

**g=np.arange(1,5).reshape(2,2)**

**print(g)**

**Output:**

**[[1 2]**

**[3 4]]**

**Inverse of matrix in linear algebra:**

**np.linalg.inv(g)**

Output:

**array([[-2. , 1. ],**

**[ 1.5, -0.5]])**

**#Create an array**

**q=np.arange(1,21).reshape(5,4)**

**print(q)**

**Output:**

**[[ 1 2 3 4]**

**[ 5 6 7 8]**

**[ 9 10 11 12]**

**[13 14 15 16]**

**[17 18 19 20]]**

**#find max number print the max number position**

**print(np.argmax(q))**

**Output:**

**19**

**#create an array**

**c=np.floor(10\*np.random.rand(8,4))**

**print(c)**

**Output:**

**[[6. 6. 5. 3.]**

**[6. 0. 0. 8.]**

**[1. 7. 2. 1.]**

**[1. 1. 7. 6.]**

**[7. 1. 3. 6.]**

**[0. 6. 0. 2.]**

**[4. 4. 9. 6.]**

**[5. 5. 3. 6.]]**

**#find max number print the max number position**

**print(np.argmax(c))**

Output:

26

#**create an array**

**c=10\*np.random.rand(8,4)**

**print(c)**

**Output:**

**[[9.71283685 4.96920185 0.89965147 5.63287243]**

**[1.41442099 7.61299749 1.79513709 4.35663633]**

**[5.78106129 7.96696479 0.21292152 7.34638472]**

**[1.81876759 5.99018431 6.61776614 4.47232379]**

**[1.35841125 2.05055714 1.33442422 2.57001196]**

**[4.46195763 0.79585853 6.65823337 8.55746628]**

**[7.26610273 2.29602706 3.61810029 2.64112403]**

**[3.68249237 3.08900181 3.60952006 2.56567894]]**

**#using argmax**

**1.print(np.argmax(c,axis=1))**

**# it will print every row highest number position**

**Output:**

**[0 1 1 2 3 3 0 0]**

**2.print(np.argmax(c,axis=0))**

**#print highest value column position**

**Output:**

**[0 2 5 5]**

**#using argmin**

**1.print(np.argmin(c,axis=0))**

**it will print every column lowest number position**

**Output:**

**[4 5 2 7]**

**2.print(np.argmin(c,axis=1))**

**it will print every row lowest number position**

**Output:**

**[2 0 2 0 2 1 1 3]**

**#using where function**

#creating an array

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

print(h)

Output:

[10 20 30 40 50 60]

1.print numbers greater than 20 using where function

print(np.where(h>20))

Output:

(array([2, 3, 4, 5]),)

2.#numbers divisible by 6

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

print(np.where(n%6==0))

Output:

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

#**search sort**

**#creating an array**

**h=np.array([10,20,30,40,50,60])**

**print(h)**

**Output:**

**[10 20 30 40 50 60]**

**1.search element**

**s=np.searchsorted(h,20)**

**print(s)**

**Output:**

**1**

**2.# for unsorted elements**

**s1=np.searchsorted(w,2)**

**print(s1)**

**Output:**

**0**

**#sorting:**

**1.sorting**

**o=np.array(['banana','cherry','apple'])**

**print(np.sort(o))**

**Output:**

**['apple' 'banana' 'cherry']**

**2.sorting with in the row 2d array**

**k=np.array([[2,6,4],[5,3,7]])**

**print(np.sort(k))**

**Output:**

**[[2 4 6]**

**[3 5 7]]**

**#using flit**

**1.l=np.array([32,55,72,89,36])**

**flit=np.where(l%2==0)**

**print(flit)**

**Output:**

**(array([0, 2, 4]),)**

**2.l[flit]**

**Output:**

**array([32, 72, 36])**

**Iterating through two arrays**

**names=np.array(['gopi','divya','vasavi','vyshu','neha'])**

**initials=np.array(['A','K','K','Y','M'])**

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

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

**Output:**

**A . gopi**

**K . divya**

**K . vasavi**

**Y . vyshu**

**M . neha**

**Logarithims:**

**1.log base 2**

**j=np.arange(1,10)**

**print(j)**

**print(np.log2(j))**

**Output:**

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

**[0. 1. 1.5849625 2. 2.32192809 2.5849625**

**2.80735492 3. 3.169925 ]**

**2.log base 10**

**i=np.arange(1,10)**

**print(i)**

**print(np.log10(i))**

**Output:**

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

**[0. 0.30103 0.47712125 0.60205999 0.69897 0.77815125**

**0.84509804 0.90308999 0.95424251]**

**3.log base e**

**x=np.arange(1,10)**

**print(x)**

**print(np.log(x))**

**Output:**

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

**[0. 0.69314718 1.09861229 1.38629436 1.60943791 1.79175947**

**1.94591015 2.07944154 2.19722458]**

**Other mathematical functions:**

**1.cumulative sum**

**e=np.array([4,5,6,7])**

**f=np.cumsum(e)**

**print(f)**

**Output:**

**[ 4 9 15 22]**

**2.cumulative product**

**g=np.cumprod(e)**

**print(g)**

**Output:**

**[ 4 20 120 840]**

**3.differance**

**#differance e2-e1**

**h=np.diff(e)**

**print(h)**

**Output:**

**[1 1 1]**

**4.lcm**

**#lcm**

**n1=567**

**n2=588**

**x=np.lcm(n1,n2)**

**print(x)**

**Output:**

**15876**

**5.gcd**

**#gcd**

**n1=567**

**n2=588**

**y=np.gcd(n1,n2)**

**print(y)**

**Output:**

**21**

**6.gcd with in array**

**#gcd with in array**

**v=np.array([3,6,8])**

**gc=np.gcd.reduce(v)**

**print(gc)**

**Output:**

**1**

**7.lcm with in array**

**#lcm within array**

**m=np.array([3,6,8])**

**lc=np.lcm.reduce(m)**

**print(lc)**

**Output:**

**24**

**Plotting:**

In Python, the matplotlib library is commonly used for plotting, and it provides various functions for creating different types of plots, such as line plots, bar charts, and scatter plots.

Syntax:

import matplotlib.pyplot as plt

Example:

Plot the graph using corona cases in first 5 days

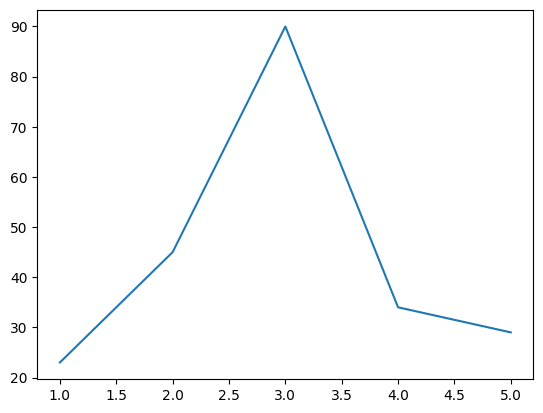
a=[23,45,90,34,29]

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

plt.plot(b,a) #plot(x,y)

plt.show()

Output:



problem1:

runs scored by 10 newspapers[100,50,91,68,89,25,34,19,9,10].

wickets taken by same 10 new players[1,0,2,0,3,7,8,9,7,5] from clusters from batsman and players.

Code:

#using scatter

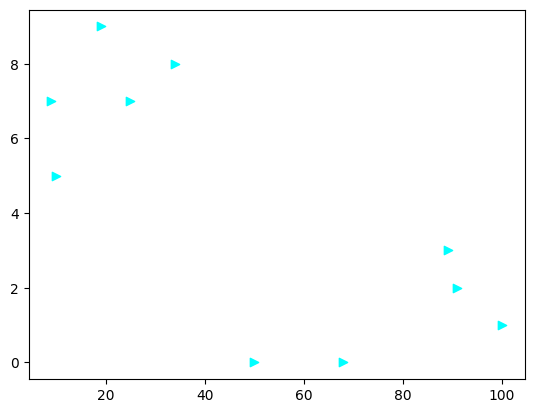
runs=np.array([100,50,91,68,89,25,34,19,9,10])

wic=np.array([1,0,2,0,3,7,8,9,7,5])

plt.scatter(runs,wic,color="cyan",marker=">")

plt.show()

Output:



#using lines

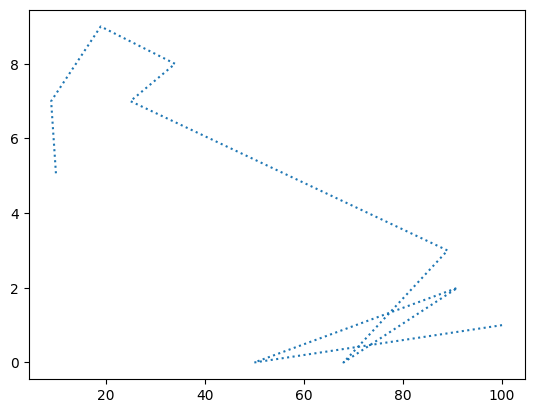
runs=np.array([100,50,91,68,89,25,34,19,9,10])

wic=np.array([1,0,2,0,3,7,8,9,7,5])

plt.plot(runs,wic,linestyle=":")

plt.show()

Output:



Problem2:

plot the score comparision of 2 students in 5 different subjects subjects as xaxis and marks as yaxis.

s1=[20,21,18,12,10]

s2=[11,31,52,67,45]

sub=["maths","english","telugu","social","science"]

plt.plot(sub,s1,label="s1",color="cyan")

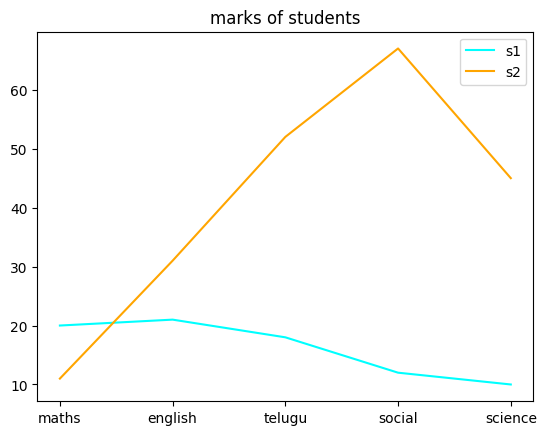
plt.plot(sub,s2,label="s2",color="orange")

plt.title("marks of students")

plt.legend()

plt.show()

Output:



**#using sub plots**

**s1=[20,21,18,12,10]**

**s2=[11,31,52,67,45]**

**sub=["maths","english","telugu","social","science"]**

**plt.subplot(2,2,1)**

**plt.plot(sub,s1,label="s1",color="cyan")**

**plt.legend()**

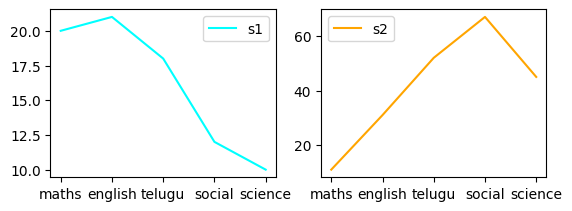
**plt.subplot(2,2,2)**

**plt.plot(sub,s2,label="s2",color="orange")**

**plt.legend()**

**plt.show()**

**Output:**

****

**Problem3:(using bar graph)**

**create subplots(bar) showing the profits of two components**

**given: revenue of company A and B**

plt.bar()

year A B

2019 23O 200

2020 560 160

2021 780 270

2022 127 127

2023 128 400

Code:

A=np.array([230,560,780,127,128])

B=np.array([200,160,270,127,400])

print("revenue of company A",A)

print("revenue of company B",B)

profA=np.diff(A)

print("profits of A",profA)

profB=np.diff(B)

print("profits of B",profB)

x=["19-20","20-21","21-22","22-23"]

plt.subplot(1,2,1)

plt.bar(x,profA,color="cyan",label="profit A")

plt.legend(loc="best")

plt.subplot(1,2,2)

plt.bar(x,profB,color="orange",label="profit B")

plt.legend(loc="best")

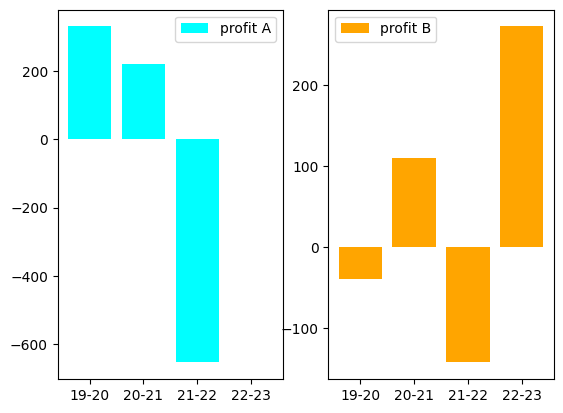
output:revenue of company A [230 560 780 127 128]

revenue of company B [200 160 270 127 400]

profits of A [ 330 220 -653 1]

profits of B [ -40 110 -143 273]

<matplotlib.legend.Legend at 0x7a649d2f7e20>



**Using pie chart:**

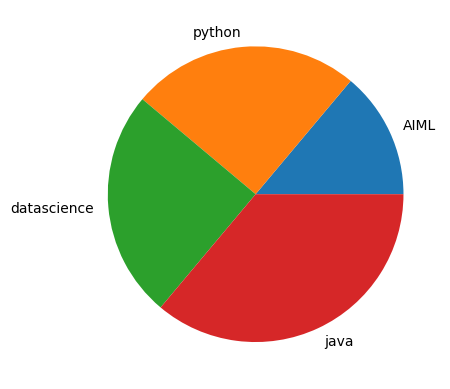
**a=np.array([25,45,45,65])**

**w=["AIML","python","datascience","java"]**

**plt.pie(a,labels=w)**

**plt.show()**

**Output:**

****

**2.**

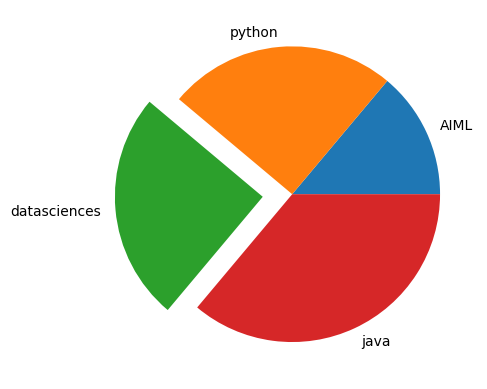
**a=np.array([25,45,45,65])**

**labe=["AIML","python","datasciences","java"]**

**explo=[0,0,0.2,0]**

**plt.pie(a,labels=labe,explode=explo,startangle=360)**

**plt.show()**

****

3.

a=np.array([25,45,45,65])

labe=["AIML","python","datasciences","java"]

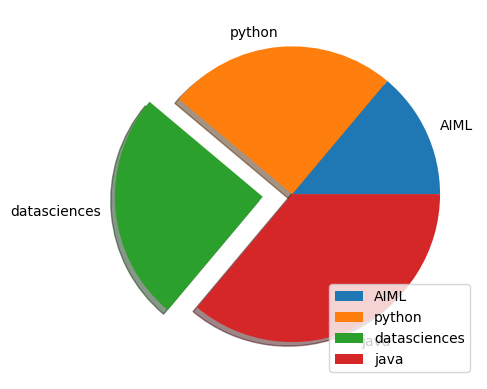
explo=[0,0,0.2,0]

plt.pie(a,labels=labe,explode=explo,shadow=True)

plt.legend()

plt.show()

Output:



**pandas:**

* used for data manupulation(data cleaning,organizing files)
* creates dataframes fro excel,csv,txt,DBs
* data frames(rows and columns readable by python)
* data cleaning by dropping or replacing with mean
* visualize the data

Example:

names=["gopika","kusuma","prasanna"]

index=[67,70,72]

ser1=pd.Series(names,index)

print(ser1)

Output:

67 gopika

70 kusuma

72 prasanna

dtype: object

**CSV Files:**

**Example:**

**df=pd.read\_csv("/content/grades\_withnulls.csv")**

**df.head(10)**

**Output:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **index** | **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 |
| **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 |

Example2:

df.tail(5)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Names** | **Initials** | **SEM1** | **SEM2** | **SEM3** | **Grade** | **Placed** |  |
| **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 |

**Txt files:**

**dft=pd.read\_csv("/content/grades.txt")**

**dft.head(10)**

**Output:**

|  |  |
| --- | --- |
| **Names Initials SEM1 SEM2 SEM3 Grade** |  |
| **0** | **Joe K 9.8 10 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 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 9.2 A** |
| **7** | **Mari N 7.7 8 7.1 B** |
| **8** | **Jess K 9.8 9.1 9.9 A+** |
| **9** | **Rajini M 7 9.1 9.3 A** |

**Using separate command:**

**dft=pd.read\_csv("/content/grades.txt",sep=" ")**

**dft.head()**

**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+** |

**Excel files:**

**dfe=pd.read\_excel("/content/diabetes.xlsx")**

**dfe.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** |

**Using describe command:**

**df=pd.read\_csv("/content/grades\_withnulls.csv")**

**print(df.describe)**

**Output:**

**<bound method NDFrame.describe of 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**

**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>**

**get the noof rows and columns**

**print(df.shape)**

**Output:**

**(17, 7)**

**get the number of rows**

**print(df.shape[0])**

**Output:**

**17**

**get the number of columns**

**print(df.shape[1])**

**Output:**

**7**

**print(df.columns)**

**Output:**

**Index(['Names', 'Initials', 'SEM1', 'SEM2', 'SEM3', 'Grade', 'Placed'], dtype='object')**

**to access rows**

**print(df[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+ 1**

**Accessing data**

* **LOC-accepts column name and index**
* **ILOC-accepts only index**

**rows of specified columns**

**print(df.loc[2:5,'Names'])**

**Output:**

**2 Kissan**

**3 Mary**

**4 Jeen**

**5 Raj**

**Name: Names, dtype: object**

**rows of columns**

**print(df.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+ 1**

**iloc[row range,column range]=>index**

**print(df.iloc[2:5, :3])**

**Output:**

**Names Initials SEM1**

**2 Kissan V 9.9**

**3 Mary N 7.7**

**4 Jeen K 9.8**

**NULL Files:**

**dfn=pd.read\_csv("/content/grades\_withnulls.csv")**

**dfn.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** |

**2.dfn.isnull().head(7)**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **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** |

**to view how many null values in each column**

**dfn.isnull().sum()**

**Names 0**

**Initials 0**

**SEM1 3**

**SEM2 0**

**SEM3 1**

**Grade 0**

**Placed 0**

**dtype: int64**

**to view total nulls**

**dfn.isnull().sum().sum()**

**4**

**Dropping Nulls:**

**dropping ll the rows with null values**

**dfn.dropna()**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **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=dfn.dropna()**

**Dfc**

**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** |

**Using fill:**

**null values fill with 5**

**dfn.fillna(5)**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **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 with mean:**

**m=dfn['SEM1'].mean()**

**print(m)**

**Output:**

**8.992857142857144**

**2.dfc2=dfn.fillna(m)**

**print(dfc2)**

**Output:**

**Names Initials SEM1 SEM2 SEM3 Grade Placed**

**0 Joe K 9.800000 10.0 9.900000 A+ 1**

**1 Rajesh M 8.900000 9.1 9.300000 A 1**

**2 Kissan V 9.900000 9.8 10.000000 A 0**

**3 Mary N 7.700000 8.0 8.992857 B 0**

**4 Jeen K 9.800000 9.1 9.900000 A+ 1**

**5 Raj M 8.900000 9.1 9.300000 A 1**

**6 Hassan V 9.900000 9.0 9.200000 A 1**

**7 Mari N 7.700000 8.0 7.100000 B 1**

**8 Jess K 8.992857 9.1 9.900000 A+ 1**

**9 Rajini M 8.992857 9.1 9.300000 A 0**

**10 Kiran V 8.992857 9.3 9.200000 A 0**

**11 Maya N 7.700000 8.0 7.100000 B 0**

**12 Jolin K 9.800000 9.1 9.900000 A+ 1**

**13 Rajesh M 8.900000 9.1 9.300000 A 1**

**14 Riya M 9.300000 9.9 10.000000 A 1**

**15 Sana V 9.900000 9.3 9.200000 A 0**

**16 Mark N 7.700000 8.0 7.000000 B 0**

**. Dropping Duplicates:**

The drop\_duplicates() method removes duplicate rows.

Code:

#drop the duplicates from the above dfc2

dropped=dfc2.drop\_duplicates()

print(dropped)

Output:

Names Initials SEM1 SEM2 SEM3 Grade Placed

0 Joe K 9.800000 10.0 9.900000 A+ 1

1 Rajesh M 8.900000 9.1 9.300000 A 1

2 Kissan V 9.900000 9.8 10.000000 A 0

3 Mary N 7.700000 8.0 8.992857 B 0

4 Jeen K 9.800000 9.1 9.900000 A+ 1

5 Raj M 8.900000 9.1 9.300000 A 1

6 Hassan V 9.900000 9.0 9.200000 A 1

7 Mari N 7.700000 8.0 7.100000 B 1

8 Jess K 8.992857 9.1 9.900000 A+ 1

9 Rajini M 8.992857 9.1 9.300000 A 0

10 Kiran V 8.992857 9.3 9.200000 A 0

11 Maya N 7.700000 8.0 7.100000 B 0

12 Jolin K 9.800000 9.1 9.900000 A+ 1

14 Riya M 9.300000 9.9 10.000000 A 1

15 Sana V 9.900000 9.3 9.200000 A 0

16 Mark N 7.700000 8.0 7.000000 B 0

**Columns:**

Code:

dfc2.rename(columns={"Grade":"GPA"}) #replace grade column name with gpa

Output:

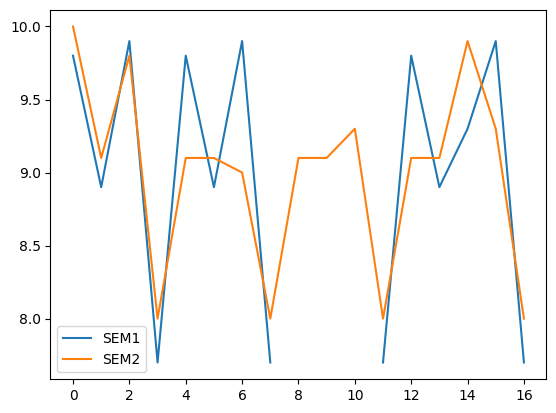
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Names** | **Initials** | **SEM1** | **SEM2** | **SEM3** | **GPA** | **Placed** |  |
| **0** | Joe | K | 9.800000 | 10.0 | 9.900000 | A+ | 1 |
| **1** | Rajesh | M | 8.900000 | 9.1 | 9.300000 | A | 1 |
| **2** | Kissan | V | 9.900000 | 9.8 | 10.000000 | A | 0 |
| **3** | Mary | N | 7.700000 | 8.0 | 8.992857 | B | 0 |
| **4** | Jeen | K | 9.800000 | 9.1 | 9.900000 | A+ | 1 |
| **5** | Raj | M | 8.900000 | 9.1 | 9.300000 | A | 1 |
| **6** | Hassan | V | 9.900000 | 9.0 | 9.200000 | A | 1 |
| **7** | Mari | N | 7.700000 | 8.0 | 7.100000 | B | 1 |
| **8** | Jess | K | 8.992857 | 9.1 | 9.900000 | A+ | 1 |
|  |  |  |  |  |  |  |  |

Code:

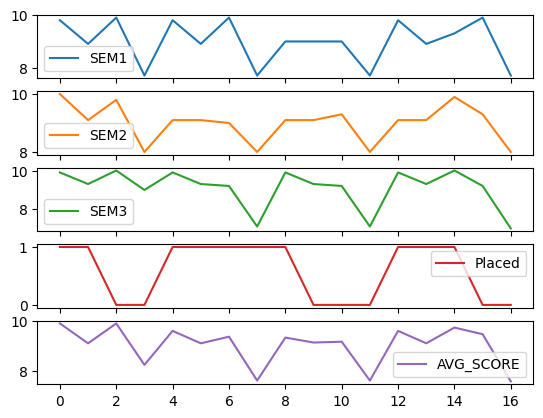
dfc2['AVG\_SCORE']=(dfc2['SEM1']+dfc2['SEM2']+dfc2['SEM3'])/3 #df['newcolumn']=values

dfn[['SEM1','SEM2']].plot.line()

Output:



dfc2.plot.line(subplots=True)



**Seaborn:**

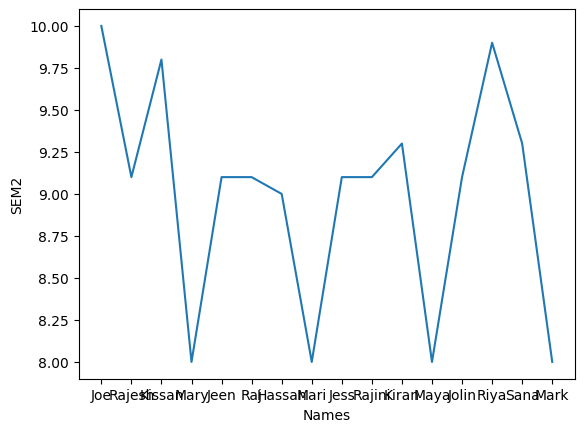
Seaborn is a Python data visualisation library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

Imporing:

import seaborn as sns

Plotting:

pl=sns.lineplot(x="Names",y="SEM2",data=dfc2)

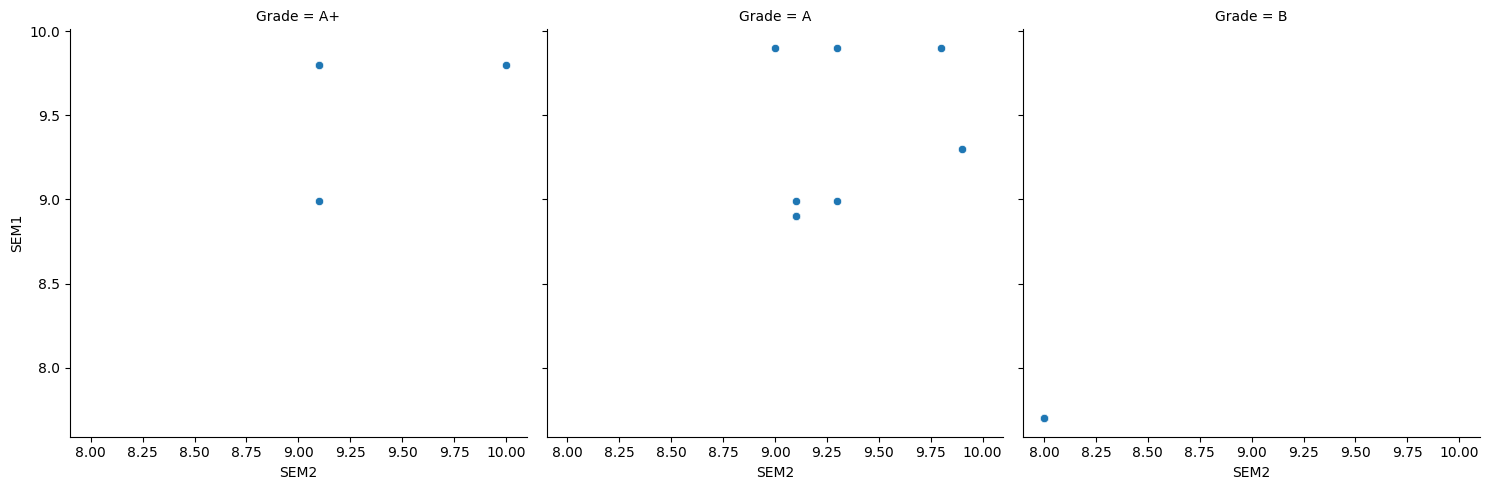


sns.relplot(

data=dfc2,

x="SEM2",y="SEM1",col="Grade")

Output:



prob: 1.load diabaties.csv 2.create rel plot with age in y-axis and class as columns

df=pd.read\_csv("/content/diabetcsv.csv")

df.head()

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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 |

df.tail(5)

Output:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **preg** | **plas** | **pres** | **skin** | **insu** | **mass** | **pedi** | **age** | **class** |  |
| **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 |

import numpy as np

df['index']=np.arange(0,768,1)

df.head()

Output:

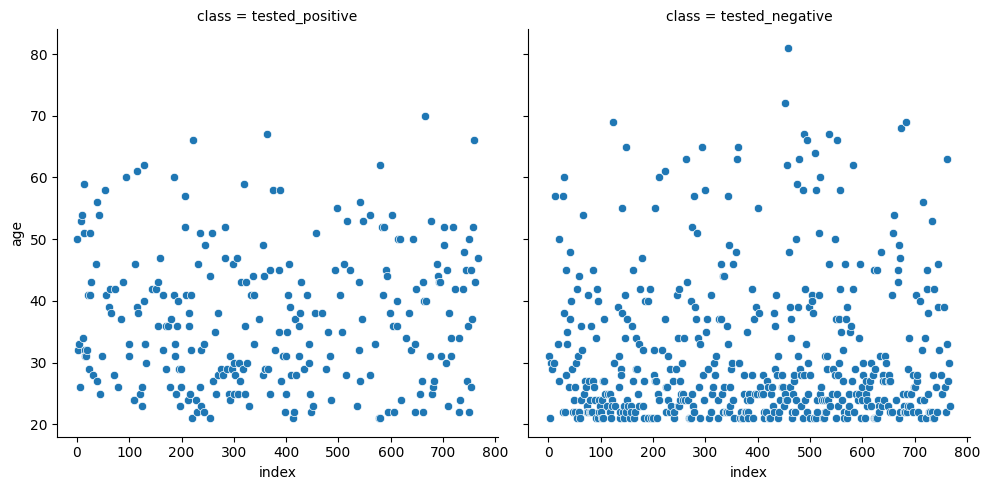
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **preg** | **plas** | **pres** | **skin** | **insu** | **mass** | **pedi** | **age** | **class** | **index** |
| **0** | 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 | tested\_positive | 0 |
| **1** | 1 | 85 | 66 | 29 | 0 | 26.6 | 0.351 | 31 | tested\_negative | 1 |
| **2** | 8 | 183 | 64 | 0 | 0 | 23.3 | 0.672 | 32 | tested\_positive | 2 |
| **3** | 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | 21 | tested\_negative | 3 |
| **4** | 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 | tested\_positive | 4 |

sns.relplot(

data=df,

x="index",y="age",col="class")

Output:



Inbuilt data sets sea born

* tips
* dowjones
* FMRI
* dots
* healthexp  
  to load this dataset
  + syntax:load\_dataset("datasetname")

tips=sns.load\_dataset("tips")

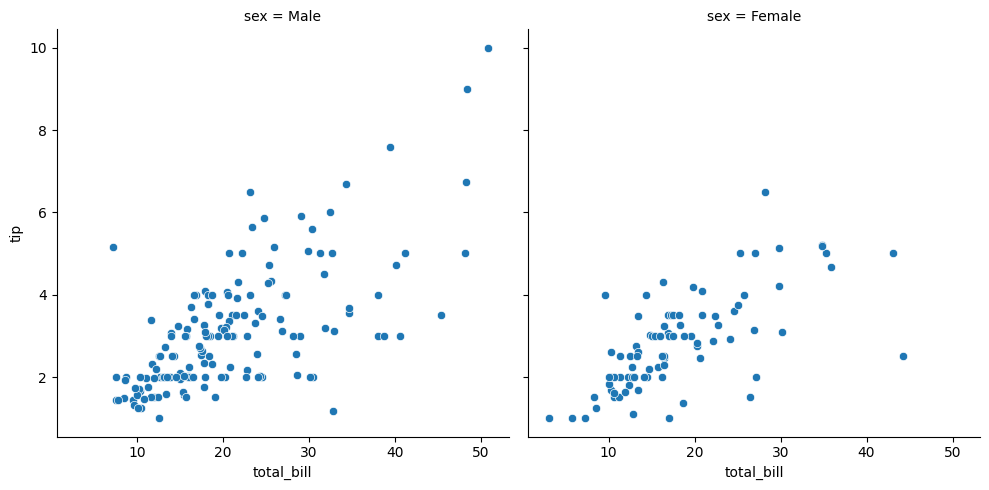
tips.head()

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **total\_bill** | **tip** | **sex** | **smoker** | **day** | **time** | **size** |  |
| **0** | 16.99 | 1.01 | Female | No | Sun | Dinner | 2 |
| **1** | 10.34 | 1.66 | Male | No | Sun | Dinner | 3 |
| **2** | 21.01 | 3.50 | Male | No | Sun | Dinner | 3 |
| **3** | 23.68 | 3.31 | Male | No | Sun | Dinner | 2 |
| **4** | 24.59 | 3.61 | Female | No | Sun | Dinner | 4 |

sns.relplot(

data=tips,

x="total\_bill",y="tip",col="sex")



**Dowjones:**

dowjones=sns.load\_dataset("dowjones")

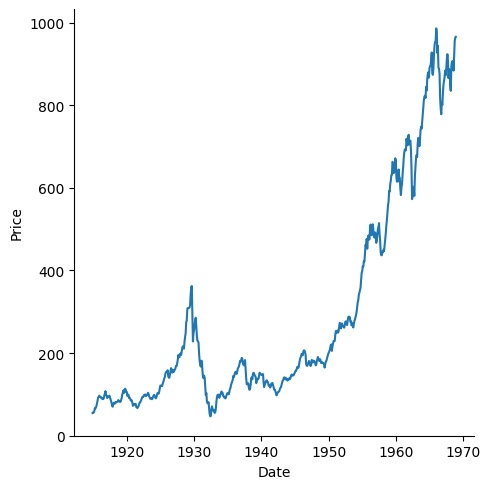
dowjones.head()

|  |  |  |
| --- | --- | --- |
|  | **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 |

sns.relplot(

data=dowjones,

x="Date",y="Price",hue="",kind="line")



**FMRI:**

**fmri=sns.load\_dataset("fmri")**

**#subjects=no.of patient,region=brain location,**

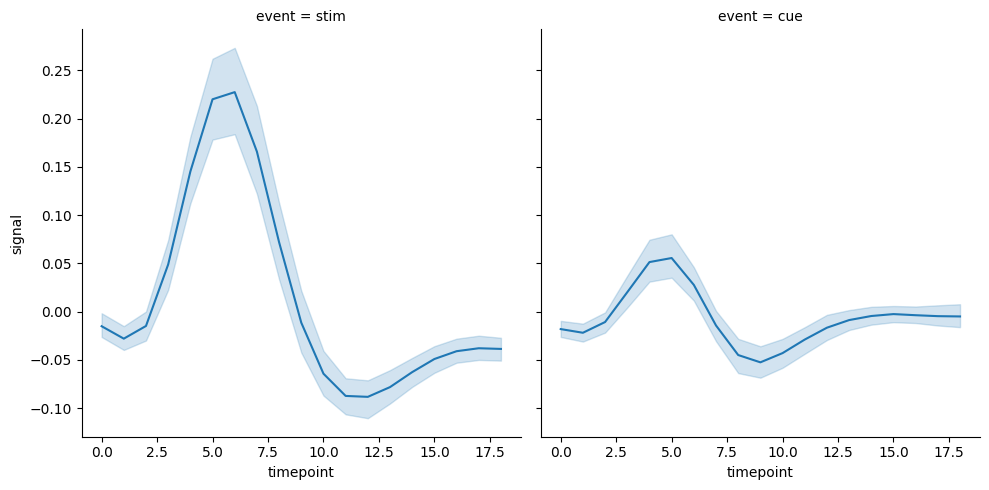
**fmri.head()**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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** |

**sns.relplot(**

**data=fmri,**

**x="timepoint",y="signal",col="event",palette="pastel",kind="line")**

****

**Dots:**

**dots=sns.load\_dataset("dots")**

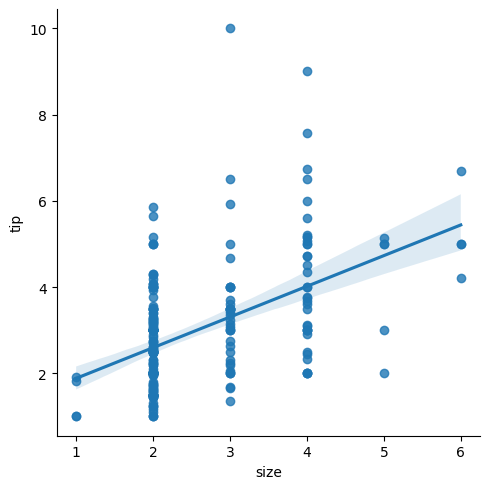
**dots.head()**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **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** |

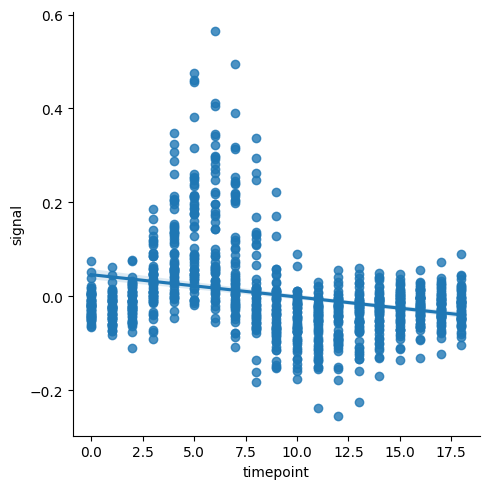
**Lineafit-lmplot()**

* **plotting the data set**
* **fitting the line**
* **prediction**

sns.lmplot(x="size",y="tip",data=tips);



**sns.lmplot(x="timepoint",y="signal",data=fmri);**

****

**web scrapping:**

* definition: loading a data from the website.
* unstructed in html
* convertible into spreadsheets/DB
* major websites have their API'S for webscrappinng

SCRAPPER:

* extract all the data from the 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:

* Beautiful soup(it scoop out the HTML code,package bs4-subpackage-beautifulsoup)  
  function
  + beautiful soup()
  + find()
  + find\_all()
* request(to control out the webpage)
* selenium
* pandas
* webdriver\_manager

to install any library

* !pip install library
* !pip install\_upgrade library #to upgrade the recent version importing subpackage
* from parent import the child
* importing from bs4 to beautiful soup

problem1:

Extracting HTML code of any website

#import libraries

f#rom bs4 import BeautifulSoup

#import requests

import requests

import pandas as pd

#import libraries

from bs4 import BeautifulSoup

url="https://www.kaggle.com/"

req=requests.get(url)

#using the beautifulsoup(),access the HTML code

a=BeautifulSoup(req.text,'html')

print(a)

problem2:

extracting table from a webpage

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

req=requests.get(v)

b=BeautifulSoup(req.text,'html')

print(b)

#td table data

#tr table row

#th table head

#to find the table from the code

table=b.find('table')

print(table)

# to find the headings from the table

heading=table.find\_all('th')

print(heading)

Output:

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

# to find only heading names in table

nl=[i.text for i in heading]

print(nl)

# to find only heading names in table

nl=[i.text for i in heading]

print(nl)

# to find rows

row=table.find\_all('tr')

print(row)

r=[i.text for i in row]

r

r=[i.text for i in row]

r

output

['\nName & India Rank\nGlobal Rank\nNet worth (US$)\nCompany\n',

'\n#1 Mukesh Ambani \n11\n$113.0 B\nReliance Industries\n',

'\n#2 Gautam Adani \n16\n$81.2 B\nAdani Group\n',

'\n#3 Shiv Nadar \n37\n$37.1 B\nHCL Technologies\xa0\xa0 \n',

'\n#4 Savitri Jindal & family \n58\n$28.9 B\nJSW Group\n',

'\n#5 Cyrus Poonawalla \n68\n$25.6 B\nSerum Institute of India\n',

'\n#6 Dilip Shanghvi \n69\n$25.5 B\nSun Pharmaceutical Industries Ltd\n',

'#7 Kumar Birla \n97\n$18.9 B\nAditya Birla Group\n',

'\n#8 Kushal Pal Singh\n98\n$18.9 B\nDLF Limited\n',

'\n',

'\n#9 Lakshmi Mittal \n102\n$17.2 B\nArcelorMittal\n',

'#10 Radhakishan Damani \n105\n$16.7 B\nDMart, Avenue Supermarts\n']

for i in row[1:9]:

er=i.find\_all('td')

eachrow=[i.text for i in er]

print(eachrow)

['#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']

c=0

for i in row[1:9]:

rd=i.find\_all('td')

erd=[i.text for i in rd]

df.loc[c]=erd

c=c+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 |

SELENIUM:

· Used to scrape data from dynamic website

· Subpackage:webdiver

|  |  |  |
| --- | --- | --- |
| **Function** | **Purpose** | **Attributes** |
| Chromeoptions() | Creates 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 web page |  |

problem:

extracting the Dell laptops data from amazon.in website and saving it as .csv name of the laptop,price,No.of Reviews

!pip install selenium

import webdriver\_manager

import pandas as pd

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

#defining 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

#navigate to a website

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

**Amazon Access:**

· Amazon Search box id:twotabsearchtextbox

· Searching id:nav-search-submit-button

# Project 3

· Extracting dell laptops data from amazon.in website and saving it as.csv Name of the laptop,Price,Number of Reviews

#Program flow

Phase 1-launching

· 1.install libraries-selenium,web\_manager

· 2

Phase 2

· 1.find the search box

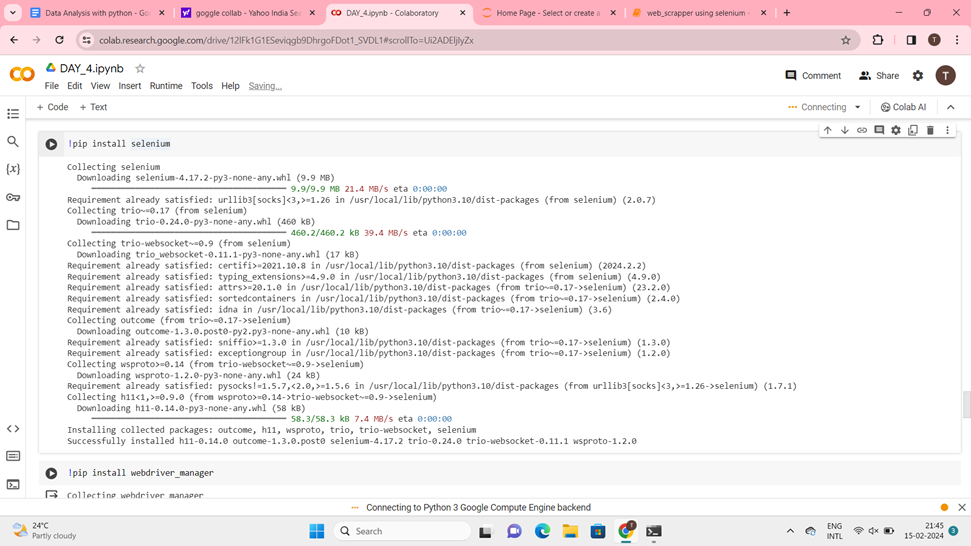
· 2.Push the text ‘dell laptops’

· 3.Click the search button

· 4.Select only ‘’dell”**Code:**

!pip install selenium

Output:



Code:

!pip install webdriver\_manager

Output:

Collecting webdriver\_manager

Downloading webdriver\_manager-4.0.1-py2.py3-none-any.whl (27 kB)

Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from webdriver\_manager) (2.31.0)

Collecting python-dotenv (from webdriver\_manager)

Downloading python\_dotenv-1.0.1-py3-none-any.whl (19 kB)

Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from webdriver\_manager) (23.2)

Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->webdriver\_manager) (3.3.2)

Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->webdriver\_manager) (3.6)

Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->webdriver\_manager) (2.0.7)

Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->webdriver\_manager) (2024.2.2)

Installing collected packages: python-dotenv, webdriver\_manager

Successfully installed python-dotenv-1.0.1 webdriver\_manager-4.0.1

Code:

import selenium

import webdriver\_manager

import pandas as pd

Code:

from selenium import webdriver

from time import sleep

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

Code:

#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

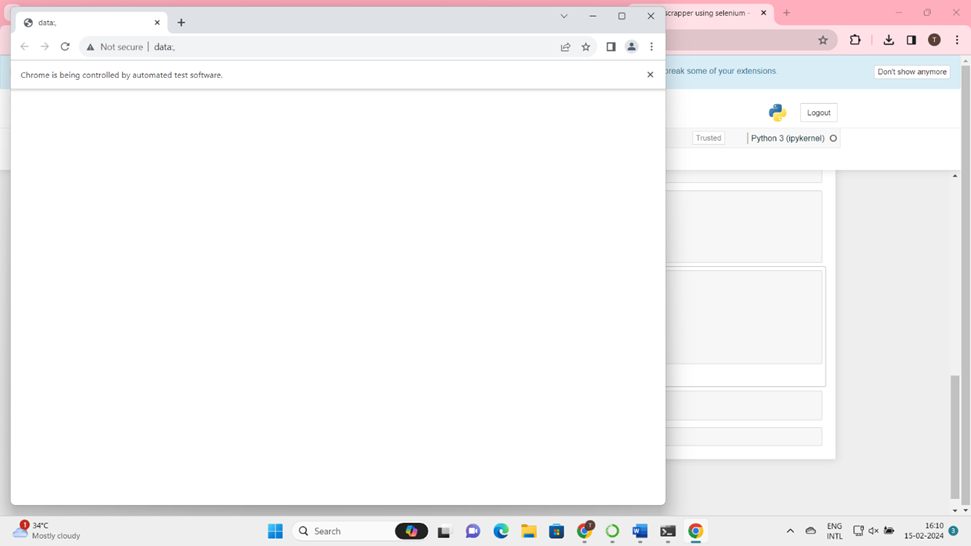
browser\_name=options.to\_capabilities()["browserName"]

print(browser\_name)

Output:

Chrome

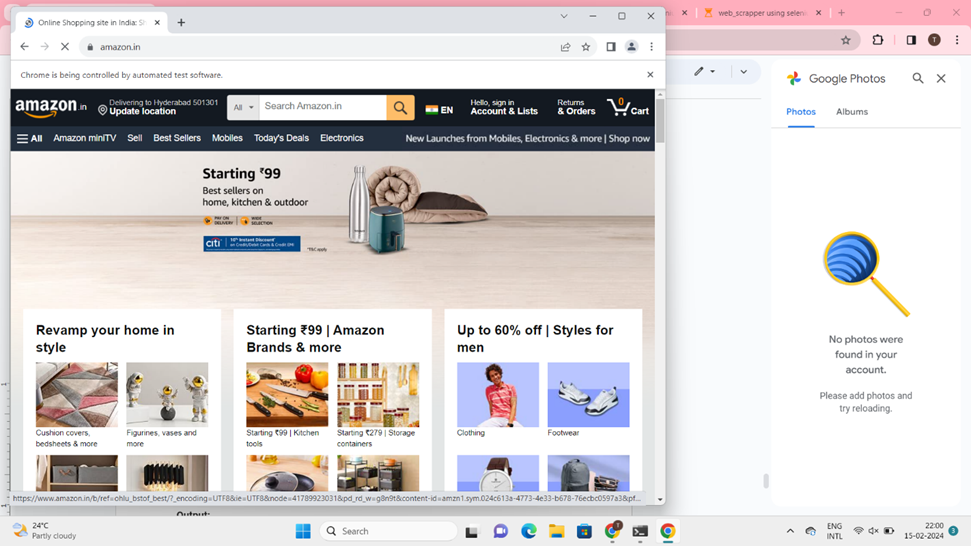
Code:



#navigate to a website

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

Output:



Code:

search=driver.find\_element(By.ID,"twotabsearchtextbox")

Output:

Code:

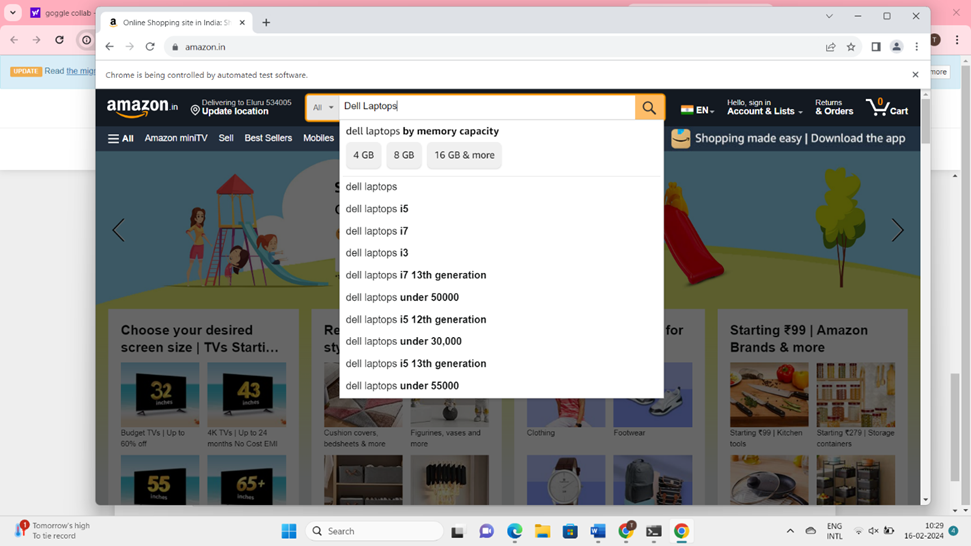
search.send\_keys("Dell Laptops")

Output:

Code:

driver.find\_element(By.ID,"nav-search-submit-button").click()

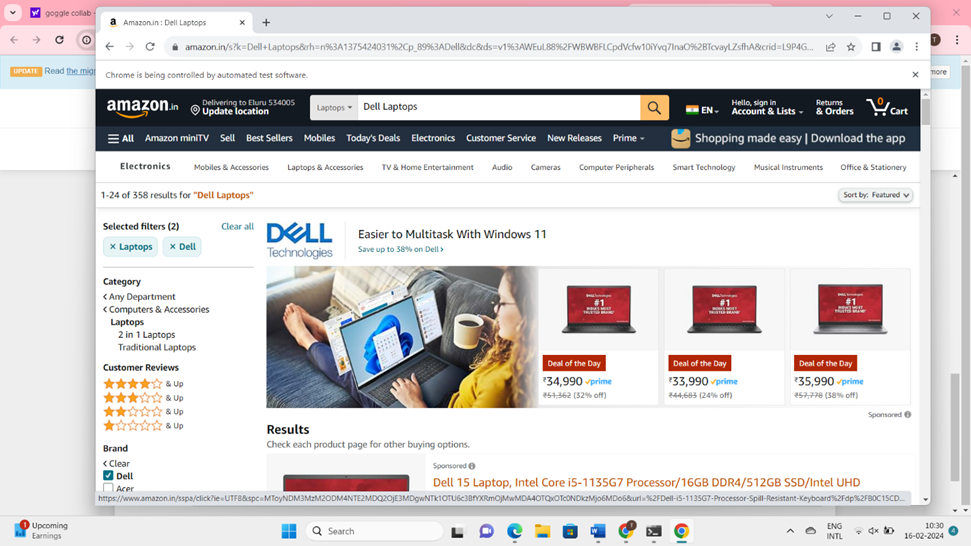
Output:



Code:

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

Output:



**Code:**

prices=driver.find\_elements(By.XPATH,".//span[@class='a-price-whole']")

l1\_prices=[i.text for i in prices]

print(l\_prices)

print(len(l1\_prices))

l1\_prices.pop(0)

l1\_prices.pop(0)

l1\_prices.pop(0)

print(l1\_prices)

print(len(l1\_prices))

**Output:**

['34,990', '33,990', '35,990', '35,990', '55,280', '35,990', '49,990', '38,990', '44,990', '23,649', '57,990', '67,490', '33,990', '44,990', '83,490', '46,990', '75,990', '30,630', '19,999', '71,490', '19,890', '22,499', '37,817', '20,999', '98,990', '36,970', '34,380']

27

['35,990', '55,280', '35,990', '49,990', '38,990', '44,990', '23,649', '57,990', '67,490', '33,990', '44,990', '83,490', '46,990', '75,990', '30,630', '19,999', '71,490', '19,890', '22,499', '37,817', '20,999', '98,990', '36,970', '34,380']

24

**Code:**

reviews=driver.find\_elements(By.XPATH,".//span[@class='a-size-base s-underline-text']")

l\_reviews=[i.text for i in names]

print(l\_reviews)

print(len(l\_reviews))

**Output:**

['4', '2', '239', '72', '4', '607', '506', '179', '13', '631', '2', '82', '138', '517', '1', '1,534', '1', '283', '76', '151', '176', '6', '186', '195']

24

**Code:**

headings=["laptop names","prices","reviews"]

Code:

df=pd.DataFrame(columns=headings)

print(df)

**Output:**

Empty DataFrame

Columns: [laptop names, prices, reviews]

Index: []

**Code:**

df["laptop names"]=l\_names

df["prices"]=l1\_prices

df["reviews"]=l\_reviews

print(df)

**Output:**

laptop names prices reviews

0 4 35,990 4

1 2 55,280 2

2 239 35,990 239

3 72 49,990 72

4 4 38,990 4

5 607 44,990 607

6 506 23,649 506

7 179 57,990 179

8 13 67,490 13

9 631 33,990 631

10 2 44,990 2

11 82 83,490 82

12 138 46,990 138

13 517 75,990 517

14 1 30,630 1

15 1,534 19,999 1,534

16 1 71,490 1

17 283 19,890 283

18 76 22,499 76

19 151 37,817 151

20 176 20,999 176

21 6 98,990 6

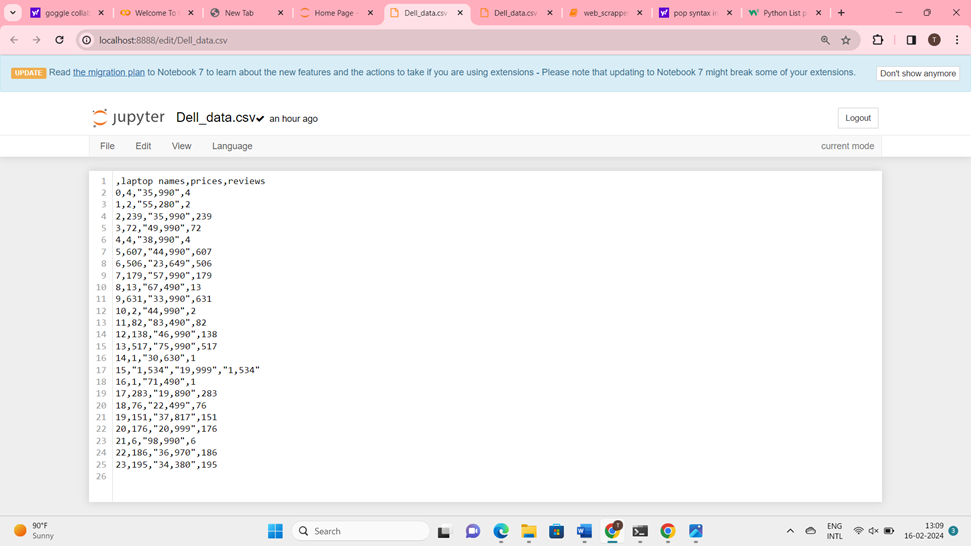
22 186 36,970 186

23 195 34,380 195

**Code:**

df.to\_csv("Dell\_data.csv")

**Output:**it the creates dell data file in home page



**What is API?**

\*API-application programming interface

\*Connecting small apps together create bigger apps

\*API is a building part of any bigger applications

\*one website accessing data from same data bases it works straight forward

\*one website accessing data from different data bases belong to different companies we need API’s for connection

**Case Study:**Flight booking

**Ex:**ola

\*App1=login

\*App2=Location

\*App3=Payment

\*And so on….

· all these small apps are connected together into a large app called ola

· these small apps are called API’s and they are building blocks of so many bigger apps,hence reusable

· so if 1000+ apps use these APIs,what happens!

· API crashes…….

· hence API keys are creayed!

**Random Fox API:**

**Code:**

import requests

page=requests.get("https://randomfox.ca/floof")

**Code:**

print(page.status\_code)

Output:

200

**Code:**

print(page.text)

**Output:**

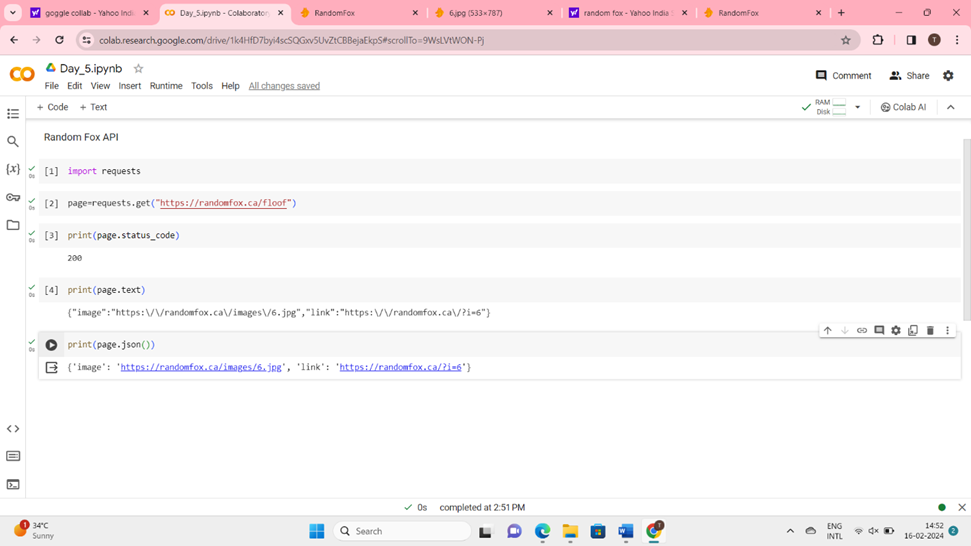
{"image":"https:\/\/randomfox.ca\/images\/6.jpg","link":"https:\/\/randomfox.ca\/?i=6"}

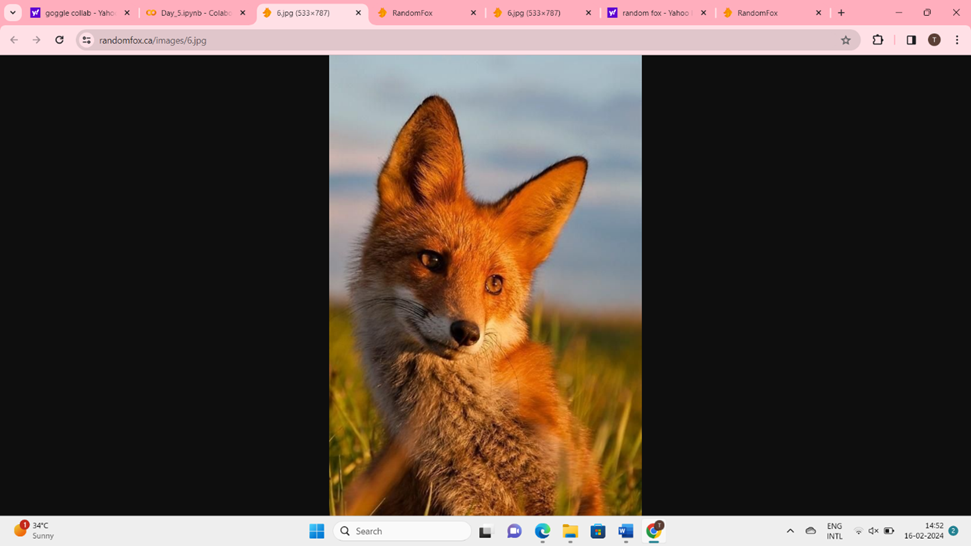
**Code:**

print(page.json())

**Output:**

{'image': '<https://randomfox.ca/images/6.jpg>', 'link': '<https://randomfox.ca/?i=6>'}





search **coinmarketcap** and enter

* And scroll down that page at the bottom there is a[Crypto API](https://coinmarketcap.com/api/) in products
* Go to[Crypto API](https://coinmarketcap.com/api/)

· Next **login**

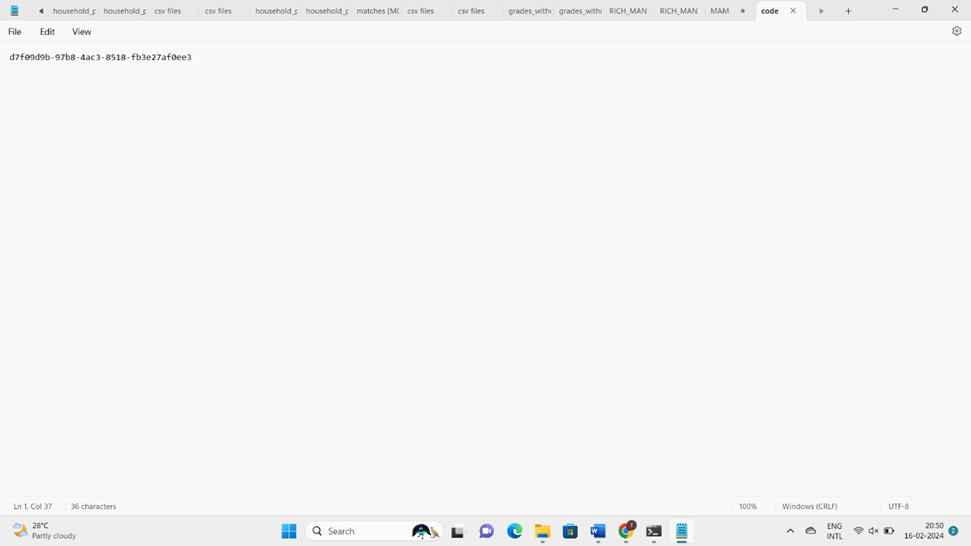
· After login to that page,you get the **key value**

· That key value save into the **note pad**

· After that,go to **“API Documentation”**

· Then click the **python** and copy that code and pate that code in **goggle collub**

· Then replace the key with your **original key** given below,

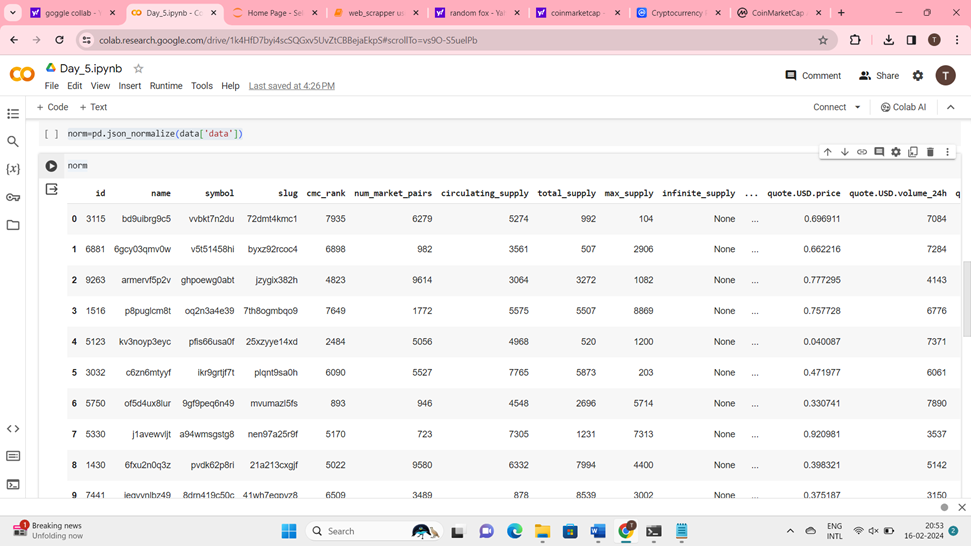


import pandas as pd

norm=pd.json\_normalize(data['data'])

norm

Output:



**Program**

# extracting data from coinMarket cap API:

program flow:

1. go to coinmarket 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 the data in the form of json
7. normalize the data into a dataframe
8. save if you needed

**Machine Learning:**

* like human learns from their past experience,machines learn from past data!
* train a machine on various things is called Machine Learning!

**Types Of Machine Learning:**

1. Supervised Machine Learning
2. Unsupervised Machine Learning
3. Reinforcement Learning

**Case Study:**

* based on the tempo,machine will suggest you the songs which contain same tempo as previous songs that you heared and liked