*) shell commands can be executed in google colab. using the ! as prefix infront of the command.

*) to know the information about the cpu assigned to you on colab.

↓

we look in the proc folder, at the file cpuinfo.

↓

similarly for memory info → meminfo inside proc folder

*) sections can be used to write neat & tidy code in colab.

*) Latex can also be used in the markup version of a cell in colab for better documentation.

*) Adding '?' as suffix to any function will open colab help. and will give required information on how to operate the function.

help(fn_name)

↑

*) Inside functions, the text inside the triple quotes is used as the docstring & is used as reference when '?' is used with that function.

↓

Hence a good practice to write or provide a suitable docstring with each function

Colab pointers

Python pointers

*) just assigning a list to another variable leads to a shallow copy.

obj1 = list(obj)

obj2 = obj.copy

↓

deep copies

list → []

tuple → ()

set, dict → {}

↓

better to declare explicitly with name().

↓ defaults to dict.

*) + operator concatenates a list to another

*) list Change (5))

↓

returns a list of elements from 0, 4 (both inclusive)

*) map function → returns a list of the results after applying the given fn. to each element of the given iterable object.

↓ → important to typecast

b = list (map (lambda x : x**2, a))

↓

special kind of function that takes one argument and operates on it

↓

more like a one line function in python

parameter → defn.
arguments → actual sent data.

*) filter function → applies fn. sent as parameter on each element of the list, and returns the ones which return true.

↓

c = list (filter (lambda x : x%.2 == 0, b))

*) tuple : immutable as compared to list

↓

supposed to be faster than a list for computations.

*) time package can be employed for time based experiments.

↓

time.time() → returns current time in seconds since epoch

*) set → unique entities. (sorted as well)

↓

faster queries

↓

(element in iterable) → syntax to check for element in object

↓

returns bool.

↓

set.add( )

↘

to add new elements in set.

*) my-dict.keys(), my-dict.values()

↓

to access keys and values in a dictionary

↓

respective type of object for each

Can be typecasted to a list

*) my-dict.items()

↓

to access key, value pairs in a dictionary

*) python packages → import math as m

from math import factorial as fact

↓

this can be any .py file, and any function or variable / object can be imported.

*) File handling in python:

file = open('mobile_cleaned.csv', 'r')

↓ file name ↓ opening mode

↓

can be 'r' or 'w'

s = file.readline()

↓

reads one line in the file

*) string.split(delimeter)

↓

Splits the given string based on the delimiter and returns a list of separated elements

*) file.close()

↓

Lines can't be read after this is executed, until file is opened again

*) with open('mobile_cleaned.csv', 'r') as file:
print(file.readline())

↓

the file is scoped and not accessible outside this scope

*) with open('mobile_cleaned.csv', 'r') as file:
print(file.read())

↓

reads all lines.

---

for line in file:
print(line) → prints each line separately

*) with open('my_first_file_output.txt', 'w') as file:
file.write('hello world from py code')

*) class → keyword to create classes

•) github notes on oops.

•) class MobilePhone():

```
def __init__(self, name):
    self.name = name
```

> constructor

> instance of the class

↓

analogous to this

•) MobilePhone.__doc__

↓

to access doc string of the class

inherits all fns.

↑

•) inheritance → class iPhone(MobilePhone):

```
def __init__(self, name):
    MobilePhone.__init__(self, name, table, 4)
```

↓

calling superclass' init method.

explicit overwriting required.

MRO: Method Resolution order (DFS left to right)

•) __str__(self):

return str.object → accessed or used when print(obj) is done

*) Numpy → import numpy as np. → alias.

x = np.array(list)

type(x) → numpy.ndarray

x = np.asarray(l, float)

x.shape()

(3,) → 1D array

(3,1)

(2,3) → 2D array

x[0,0] → accessing elements in np array

x[0,0:2] → accessing using ranges.

*) inbuilt arrays → tuple entry

```
X = np.zeros((4,5))
```

```
X = np.eye(4,5)
```

↓

diagonal entries all 1, others marked with 0

```
X = np.random.random((4,5))
```

↓

random no. elements between 0 & 1

↓

apt. addition / subtraction and multiplication can be done for changing & altering range of random no.s

*) `z = X.T` (taking transpose of X)

*) reshaping np arrays

↓

```
Y = X.reshape(20,1)
```

*) `A = np.arange(5)`

↓

analogous to array from 0 to 4. (both inclusive)

*) A = np.arange(5)

B = np.arange(5)

C = A+B

D = A-B → point / element wise operation.

E = A*B

*) A+1 → scalar (1) gets broadcasted about the vector / np.array.

*) A = np.floor(np.random.random((2,3))*10)

↓

generating a random matrix of 2*3 with elements b/w 0 & 1.

↓

multiplied by 10

↓

taking floor of those values.

↓

integers between 0 & 10

(0 inclusive, 10 not inclusive)

*) A.sum() → sum all elements of an array

A.sum(axis=0) → sum elements row wise

A.sum(axis=1)

↳ sum elements column wise

*) axes are governed by dimensions in the array.

axis=0 (row)

axis=1 (cols)

axis=2 (obj. point to point)

axis=3 (obj to obj point to point) } reference object change.

*) np.inner( ) $(5,4) \cdot (3,4)$ → $(5,3)$

↓

product takes / picks elements of one array and multiplies with opp. ended row & then takes sum.

*) np.dot( ) → conventional dot product

*) np.outer( ) → without taking sum, each element for itself

$(5,4) \cdot (3,4)$

$(20, 12)$

*) Plotting (Matplotlib)

•) import matplotlib.pyplot as plt → to use matplot lib

↓

package needs to be imported.

•) plt.plot(x)

plt.show( )

↓

if x is a list or np.array, then the values act as y axis values and indices act as x axis values.

↓

these values are plotted and then show is used to print / show the graph

•) plt.plot(x, '*')

↓

scatters/plots x with the corresponding sign/symbol.

plt.plot(x, '*-')

↓

both line and symbol get plotted

•) x = np.linspace(0, 10, 100)

range → no. of elements in the given range

↓ returns a numpy array which returns given no. of elements in the given range

! pip install → installing packages with specific versions

seaborn >= 0.9.0

y = np.power(x, 0.5)

plt.plot(x, y) ⟶ plots x → y as per the given numpy array, which should indicate the sqrt distribution

plt.show()

*) use of an extension on top of matplotlib to make the plots look better

↓

seaborn.

•) import seaborn as sns → wraps around matplotlib and makes the plots look more presentable.

•) sns.set() → sets us properties of matplotlib for better plotting.

•) sns.lineplot(x, y)

plt.show()

sns.__version__

→ better dataframing package

*) pandas : import pandas as pd

data = pd.read_csv('mobile_cleaned.csv')

data.head()

•) ax → used as handles for axes.

ax = sns.scatterplot(x = "standby_time", y = "battery_capacity", data = data)

! better to have more dense plots.

hue can be used to get another feature as part of the plot

ax = sns.scatterplot(x = "stand_by_time", y = "battery_capacity", hue = "thickness", data = data)

•) histogram using sns. → distplot

ax = sns.distplot (data['stand-by-time'])

↓

bars + distribution

kde = False → no distribution

rug = True

↳ will show where exactly are the values.

bins = 20

↳ can be used to manipulate the no. of bins in a graph

columns with specific values

↑

•) boxplot → all possible values with a common plane for common range of values

ax = sns.boxplot (x = 'is_liked',

y = 'battery_capacity',

data = data)

ax = sns.boxplot (x = 'expandable_memory',

y = 'price', data = data)

↓

can be used to check the relation b/w two cols.

only 2d o/p.

↑

•) heatmap

uniform_data =

np.random.random ((10,12))

ax = sns.heatmap (uniform_data)

↓

values in 2D matrix mapped to a color map

↓

good method to visualize the performance of parameters

→ changing color of heatmap

```
ax = sns.heatmap(uniform_data, cmap="YlGnBu")
```

*) plotting images.

specific subsection of matplotlib to handle images

```
import matplotlib.image as mpimg
```

i) reading image

```
img = mpimg.imread('one_fourth_labs.png')
```

img → matrix → color patterns at different pixels

ii) `imgplot = plt.imshow(img)`

↳ plotting image