#Use of python libraries are their understand and prepare the data through EDA (exploratory data analysis) which is essential at step2 and step3 of CRISM DM Framework

DATA TOOLKITS :

numpy

pandas

matplotlib

library

is a collection of pre written code that is used to perform common task.

library contains python modules and packages.

MODULES:-

it is a single python file containing a python code (functions,classes or variables etc).

All the modules kept together is called as package. multiple packages together is called as library.

summary numpy was developed by travis oliphat in 2005, it was introduced to have a specialised library to perform mathematical operations and computation for python language.

```
# This is formatted as code
```

```
import numpy as np #very important statement
```

numpy array is a data structure used to store data of same data type numpy stores the data in numpy array

numpy stores homogenous data.

numpy is built using c language as c is one of the fastest language in programming world after c++ numpy computation is very fast since it stores only homogenous data that's why numpy array is faster. it has a continous memory location that's why numpy is faster

list since it stores heterogenous data its computation is slow in comparison to numpy array having homogenous data .

ndarray is the type class ie n dimensional array array.ndim gives us the number of dimensions an array consist of: 1D array contains only row 2D array contains rows and columns .

to understand the no. of dimensions observe the number of square brackets opened and closed .

```
np.matrix([1,2,3,4,5])
```

matriz is a specialised two dimensional array matrix will be always 2D

more ways to convert the array are as follows:-

other ways to generate an array

1. np.array

```
arr1 = np.array([1,2,3,4,5,6,7])
arr1
```

```
array([1, 2, 3, 4, 5, 6, 7])
```

2.asarray function is used to convert list or anything into an array

```
l = [1,2,3,4]
np.asarray(l)
```

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

1. np.asanyarray asanyarray will convert the input to an ndarray but passes ndarray subclasses through. if something is already an array or part of array (subclass) it will simply allow to pass through. ie nothing will be done to that array

```
l = [1,2,3,4]
np.asanyarray(l)
```

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

arrays are mutable

shallow copy : change in one array will lead to change in another both variables will have same memory location.

a = arr (will point to same memory location so changes in any will reflect on both)

deep copy : array.copy() change in one array wil not reflect in another

Multiple approaches to generate any array

np.fromfunction() constructs an array by executing a function over each coordinates.

```
arr1 = np.ones((3,4))
print(arr1)
np.fromfunction(lambda i,j: i==j,(3,3))
#no. of elements array.size
#shape of array array.shape
```

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

```
array([[ True, False, False],
       [False,  True, False],
       [False, False,  True]])
```

```
np.fromfunction(lambda i,j: i*j,(3,3))
```

```
array([[0., 0., 0.],
       [0., 1., 2.],
       [0., 2., 4.]])
```

```
for i in range(5):
  print(i)
```

```
0
1
2
3
4
```

```
[i for i in range(5)]
```

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

Both the above examples are same

np.fromiter() it creates a new 1D array from an iterable object.

np.fromstring() default dtype = float

np.fromstring() default dtype = float np.fromstring()

for string character you have to use split

```
str = "ajay , vijay , sanjay"
str.split(",")
np.array(str.split(","))
```

```
array(['ajay ', ' vijay ', ' sanjay'], dtype='<U7')
```

Another method of numpy to generate a sequence of number by using arange() [start:stop:step]

```
np.arange(1,100) # arange returns evenly specified values within a
given interval ny default step is 1.
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
34,
       35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
51,
       52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
68,
       69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
85,
       86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99])
```

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

```
array([1. , 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2. , 2.1,
2.2,
       2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 3. , 3.1, 3.2, 3.3, 3.4,
3.5,
       3.6, 3.7, 3.8, 3.9, 4. , 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7,
4.8,
       4.9, 5. , 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 6. ,
6.1,
       6.2, 6.3, 6.4, 6.5, 6.6, 6.7, 6.8, 6.9, 7. , 7.1, 7.2, 7.3,
7.4,
       7.5, 7.6, 7.7, 7.8, 7.9, 8. , 8.1, 8.2, 8.3, 8.4, 8.5, 8.6,
8.7,
       8.8, 8.9, 9. , 9.1, 9.2, 9.3, 9.4, 9.5, 9.6, 9.7, 9.8, 9.9])

#np.linspace()
np.linspace(1,5,10)# gives 10 no.s between 1 and 5

array([1.        , 1.44444444, 1.88888889, 2.33333333, 2.77777778,
       3.22222222, 3.66666667, 4.11111111, 4.55555556, 5.        ])
```

zeros function np.zeros() 1D array

default float

it returns a new array of given shape and type,filled with zeros.

```
np.zeros((2,3),dtype=int)

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

ones function default dtype = float np.ones

return a new array of given shape and type , filled with ones

```
np.ones((3,2),dtype=int)

array([[1, 1],
       [1, 1],
       [1, 1]])

np.array([[2,2,2],[2,2,2]]) #form all the two's or all elements same
of an array.

array([[2, 2, 2],
       [2, 2, 2]])
```

3D array Dimensions can be increased by passing an extra arguement.

index wise addition/subtraction

```
arr = np.zeros((1,3,4))
arr

array([[[0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.]]])

arr2 = np.zeros((2,3))
arr3 = arr2 + 5
arr3

array([[5., 5., 5.],
       [5., 5., 5.]])
```

np.eye() function

Return a 2D array with one on the diagonal and rest elements be zero.

similar to identity matrix

```
np.eye(3)

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

np.empty() return a new array of given shape and type , without initializing entries.

it gives any random value to the array.

```
np.empty(5) #it gives random values to the array.
#helps getting random values

array([0.  , 0.25, 0.5 , 0.75, 1.  ])
```

random module of python

```
import random
random.choice((1,2,3,4,5))

3

random.choice("ajay")

{"type":"string"}

random.randrange(1,10)

9
```

ANY NUMBER TO BE GENERATED RANDOM NO.

random.random()

a random no. generated between 0 (included) and 1 (excluded)

shuffle function

random.shuffle()

uniform distribution where the data points which will be the output or the probability of getting any of the data point is equal .

```
random.uniform(7, 14)
```

```
10.522420304451582
```

it gives us random nos. in an array.

random no. related concept in numpy

np.random.random_sample()

np.random.rand()

```
np.random.random_sample(3)
```

```
array([0.21765383, 0.885346  , 0.08007244])
```

```
np.random.randn()# returns a sample (or samples) from the standard normal distribution.
# a distribution which has bell shaped curve is called standard normal distribution.
```

```
np.random.randint(1,19,size=(3,3))
```

```
array([[16,  6,  6],
       [17,  5, 18],
       [10, 14,  1]])
```

arr.ndim for

```
np.random.rand(3,3)
```

```
array([[0.44675858, 0.7619203 , 0.17846089],
       [0.90504729, 0.46938026, 0.19741774],
       [0.93174381, 0.84798206, 0.48912468]])
```

RESHAPING array

array.reshape() arr.reshape(2,6)

returns an array containing the same data with a new shape.

a condition here is that the size of the array should be equal to the size of original array.

ie arrays size will never change always,reshape parameter should be multiple of original array.

#case

when you dont know in reshape what parameter should be given in row or column count in that case any negative value for rows or columns.

```
arr = np.random.randint(1,88,size=(3,3))
print(arr)
arr.reshape(1,9)
```

```
[[ 2 49 53]
 [19 26 41]
 [82 14 29]]
```

```
array([[ 2, 49, 53, 19, 26, 41, 82, 14, 29]])
```

```
arr = np.random.randint(1,88,size=(3,3))
print(arr)
arr.reshape(1,1,9)# 3D array created through reshaping
```

```
[[66 56 60]
 [21 34 56]
 [64 54 31]]
```

```
array([[[66, 56, 60, 21, 34, 56, 64, 54, 31]]])
```

```
arra =arr.reshape(9,-1)
print(arra)
arr.reshape(9,1).base
```

```
[[66]
 [56]
 [60]
 [21]
 [34]
 [56]
 [64]
 [54]
 [31]]
```

```
array([[66, 56, 60],
       [21, 34, 56],
       [64, 54, 31]])
```

To see back the original array

array.reshape().base

conditions on array

arr1 = np.random.randint(1,88,(5,6))

values greater than 30 will give True

arr[arr>30]

select all the elements of arr1 when the condition is true.

arr[arr>30]

```
arr1 = np.random.randint(1,88,(5,6))
print(arr1)
arr1>30
```

```
[[67 16  1 36 22 32]
 [25 14 63 11 20 18]
 [17 42 65 86 39  4]
 [80 68 60 86 32 18]
 [ 6 45 76 27 26 69]]
```

```
array([[ True, False, False,  True, False,  True],
       [False, False,  True, False, False, False],
       [False,  True,  True,  True,  True, False],
       [ True,  True,  True,  True,  True, False],
       [False,  True,  True, False, False,  True]])
```

```
arr1[arr1>30]
```

```
array([67, 36, 32, 63, 42, 65, 86, 39, 80, 68, 60, 86, 32, 45, 76,
69])
```

#indexing to excess element in an array

arr1[0] to excess 0th row

arr1[1][2] for excessing particular element from the 1st index row.

from 0th row to 2nd row arr[0:3] 0th row,1st row and 2nd row

0th to 2nd and 0 and 2nd column

arr[0:3,[0,2]]

```
print(arr1)
c=arr1[2,4]
print(c)
d=arr1[0:3,[2,2]]
print(d)
```

```
[[67 16  1 36 22 32]
 [25 14 63 11 20 18]
 [17 42 65 86 39  4]
 [80 68 60 86 32 18]
 [ 6 45 76 27 26 69]]
```

```
39
[[ 1  1]
 [63 63]
 [65 65]]
```

#slicing on both rows and columns

```
arr1[2:6,0:3]

array([[17, 42, 65],
       [80, 68, 60],
       [ 6, 45, 76]])
```

#MATHEMATICAL OPERATIONS ON TWO ARRAYS .

```
arr1 = np.random.randint(1,12,(3,3))
arr2 = np.random.randint(1,10,(3,3))
print(arr1)
print(arr2)
arr3 = arr1 + arr2 #adds the elements index wise
print(arr3)# same goes with subtract , multiplication ,division.

[[10  8  7]
 [ 9  2  8]
 [ 5  1  2]]
[[9 6 9]
 [2 8 9]
 [7 6 8]]
[[19 14 16]
 [11 10 17]
 [12  7 10]]
```

In python if we divide any integer with 0 it gives error.

but in numpy it gives infinity

#matrix multiplication matrix is a specialised 2D array is different from index wise multiplication.

arr1@arr2 and mp.matmul() and np.dot()

these are the ways to multiply matrix.

```
arr1@arr2

array([[155, 166, 218],
       [141, 118, 163],
       [ 61,  50,  70]])

np.matmul(arr1,arr2)
```

```
array([[155, 166, 218],
       [141, 118, 163],
       [ 61,  50,  70]])

np.dot(arr1,arr2)

array([[155, 166, 218],
       [141, 118, 163],
       [ 61,  50,  70]])
```

conditions for matrix multiplication no. of columns of first array or matrix should be equal to no. of rows of another matrix.

axb = bxc

b = b

axc matrix will be created .

```
arr1 = np.zeros ((3,3))
arr1+5

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

arr1 = np.random.randint(1,100,size=(3,3))
print(arr1)
arr1.T # transpose of a matrix transpose function

[[90 28 18]
 [58 78 36]
 [91 34 76]]

array([[90, 58, 91],
       [28, 78, 34],
       [18, 36, 76]])

arr1 = np.random.randint(1,12,(3,3))
arr2 = np.random.randint(1,10,(3,3))
print(arr1)
print(arr2)
arr1.flatten()#returns a copy of array collapsed into 1D.

[[ 2 11  6]
 [ 5 11  1]
 [ 8  2  3]]
[[9 6 7]
 [7 5 9]
 [4 9 2]]

array([ 2, 11,  6,  5, 11,  1,  8,  2,  3])
```

#expand direction

np.expand_dims()

increase the dimension by 1 in the provided axis 0 means rows (x axis) 1 means columns (yaxis)

```
np.expand_dims(arr1,axis=1)
```

```
array([[[ 2, 11,  6]],

       [[ 5, 11,  1]],

       [[ 8,  2,  3]]])
```

```
#squeezing the dimension
np.squeeze(arr1)# removes axes of length one from "a"
```

```
array([[ 2, 11,  6],
       [ 5, 11,  1],
       [ 8,  2,  3]])
```

```
#Repeat function
np.repeat(arr1,2,axis=1)
```

```
array([[ 2,  2, 11, 11,  6,  6],
       [ 5,  5, 11, 11,  1,  1],
       [ 8,  8,  2,  2,  3,  3]])
```

```
#roll function
np.roll(arr1,-1)#shift places within the elements of the array.
```

```
array([[11,  6,  5],
       [11,  1,  8],
       [ 2,  3,  2]])
```

```
#bitwise negation
~arr1
```

```
array([[ -3, -12,  -7],
       [ -6, -12,  -2],
       [ -9,  -3,  -4]])
```

```
fruits = ["mango","apple","orange"]
np.char.upper(fruits)
```

```
array(['MANGO', 'APPLE', 'ORANGE'], dtype='<U6')
```

#mathematical functions

np.exp()

np.log()

np.power()

np.multiply()

np.subtract()

np.mean()

np.max()

np.std()

np.var()

np.mod()# returs element wise remainder

np.sqrt()

#sorting and searching

```
np.sort(arr1)#returns a sorted copy of an array.
```

```
array([[ 2,  6, 11],
       [ 1,  5, 11],
       [ 2,  3,  8]])
```

np.searchsorted()

find indices where elements should be inserted to maintain order.

np.count_nonzero()/np.where()/np.extract()

all are used for similar purpose for eg to find no. of elements which are greater or less than zero etc.

```
np.count_nonzero(arr1)
```

```
9
```

arr.byteswap()

swap bytes of the array element

```
np.where(arr1)
```

```
(array([0, 0, 0, 1, 1, 1, 2, 2, 2]), array([0, 1, 2, 0, 1, 2, 0, 1,
2]))
```

```
import numpy.matlib as nm
nm.zeros(5)
```

```
matrix([[0., 0., 0., 0., 0.]])
```

```
arr1 = np.random.randint(1,100,(3,3))
arr1
```

```
array([[93, 35, 52],
       [72,  1,  3],
       [39, 35, 59]])

#numpy linear algebra function determination concept
np.linalg.det(arr1)

np.float64(-19851.000000000044)
```

inmatrix you cannot divide with another matrix we have to use inverse ()

```
np.linalg.inv(arr1)

array([[ 0.00231726,  0.01234195, -0.00266989],
       [ 0.20810035, -0.17424815, -0.1745504 ],
       [-0.12498111,  0.09520931,  0.12226084]])

a = np.array([[1,2],[3,4]])
b = np.array([[5,6],[7,8]])
np.linalg.solve(a,b)

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