**DATA SCIENCE LAB**

Name : Kevin Chacko Abraham Roll No : 13

Batch : R-MCA B

Date : 01/09/2022

# EXPERIMENT : 1

**AIM:**

To implement

(a) Matrix operations (using vectorization),

(b) transformation using python and

(c) SVD using Python.

**Program:**

1. Matrix operations using vectorization

A numpy array is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers. The number of dimensions is the rank of the array; the shape of an array is a tuple of integers giving the size of the array along each

dimension.

We can initialize numpy arrays from nested Python lists, and access elements using square brackets. Numpy also provides many functions to create arrays with all zero values, all ones, identity matrix, all random values.

Vectorization is used to speed up the Python code without using loop. Using such a function can help in minimizing the running time of code efficiently. Various operations are being performed over vector such as dot product of vectors which is also known as scalar product as it produces single output, outer products which results in square matrix of dimension equal to length X length of the vectors, Element wise multiplication which products the element of same indexes and dimension of the matrix remain unchanged.

**Code:**

import numpy as np  
  
a = np.array([1, 2, 3])

print("type: %s" %type(a))

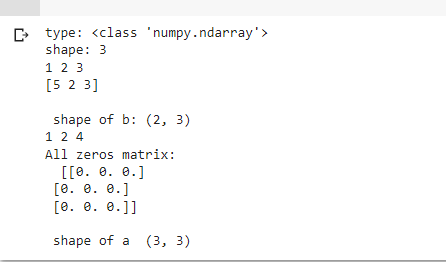
print("shape: %s" %a.shape)

print(a[0], a[1], a[2])

a[0] = 5   
print(a)   
  
b = np.array([[1,2,3],[4,5,6]])   
print("\n shape of b:",b.shape)   
print(b[0, 0], b[0, 1], b[1, 0])   
a = np.zeros((3,3))

print("All zeros matrix:\n %s" %a)   
print("\n shape of a ",a.shape)

**Output:**

****

in the observation book, it starts from here only.

#vectorized sum  
print("Vectorized sum example\n")  
x = np.array([[1,2],[3,4]])  
print("x:\n %s" %x)  
print("sum: %s"%np.sum(x))

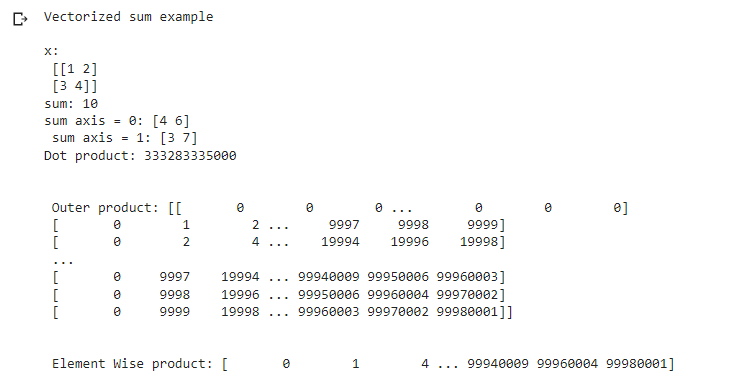
print("sum axis = 0: %s" %np.sum(x, axis=0))

print(" sum axis = 1: %s" %np.sum(x, axis=1))

#matrix dot product  
a = np.arange(10000)  
b = np.arange(10000)  
  
dp = np.dot(a,b)  
  
print("Dot product: %s\n" %dp)

#outer product  
op = np.outer(a,b)  
print("\n Outer product: %s\n" %op)  
  
#elementwise product  
  
ep = np.multiply(a, b)   
print("\n Element Wise product: %s \n" %ep)

**Output:**

****

(b) Matrix transformation

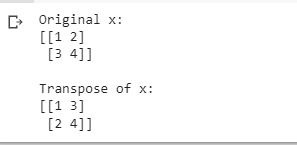
import numpy as np

x = np.array([[1,2], [3,4]])

print("Original x: \n%s " %x)

print("\nTranspose of x: \n%s" %x.T)

**Output:**



(c) SVD using python

Matrix decomposition, also known as matrix factorization, involves describing a given matrix using its constituent elements.

The Singular-Value Decomposition, or SVD for short, is a matrix decomposition method for reducing a matrix to its constituent parts in order to make certain subsequent matrix calculations simpler.

A = U . Sigma . V^T

Where A is the real m x n matrix that we wish to decompose, U is an m x m matrix, Sigma (often represented by the uppercase Greek letter Sigma) is an m x n diagonal matrix, and V^T is the transpose of an n x n matrix where T is a superscript.

# Singular-value decomposition

from numpy import array

from scipy.linalg import svd

# define a matrix

A = array([[1, 2], [3, 4], [5, 6]])

print("A: \n%s" %A)

# SVD

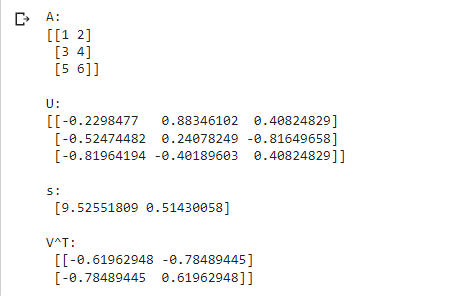
U, s, VT = svd(A)

print("\nU: \n%s" %U)

print("\ns: \n %s" %s)

print("\nV^T: \n %s" %VT)

**Output:**



**DATA SCIENCE LAB**

Name : Kevin Chacko Abraham Roll No : 13

Batch : R-MCA B

Date : 01/09/2022

# EXPERIMENT : 2

**AIM:**

Programs using matplotlib / plotly / bokeh / seaborn for data visualisation.

**Program:**

import matplotlib.pyplot as plt

x = [10, 20, 30, 40]

y = [20, 25, 35, 55]

plt.plot(x, y)

plt.show()

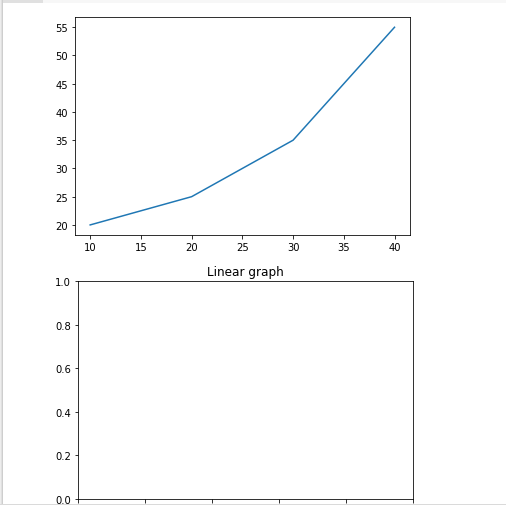
plt.title("Linear graph")

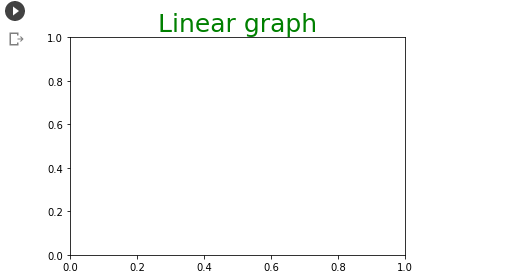
plt.show()

plt.title("Linear graph", fontsize=25, color="green")

plt.show()

**Output:**

****



plt.ylabel('Y-Axis')

plt.xlabel('X-Axis')

plt.show()

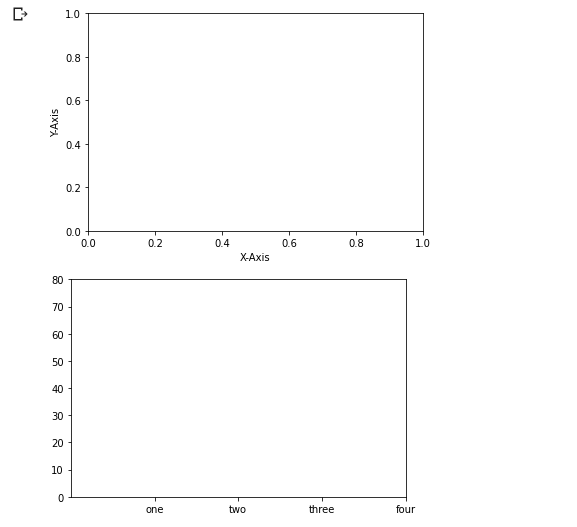
plt.ylim(0, 80)

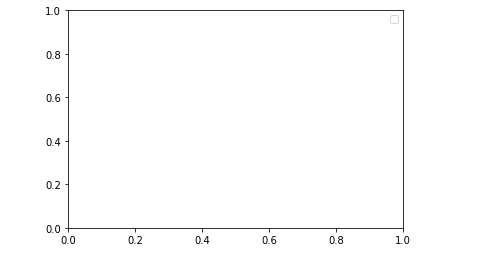
plt.xticks(x, labels=["one", "two", "three", "four"])

plt.show()

plt.legend(["GFG"])

plt.show()





import matplotlib

import numpy

fig = matplotlib.pyplot.figure()

# Generate line graph

x = numpy.arange(0, 1.414\*2, 0.05)

y1 = numpy.sin(x)

y2 = numpy.cos(x)

# Creating two axes

# add\_axes([xmin,ymin,dx,dy])

axes1 = fig.add\_axes([0, 0, 1, 1])

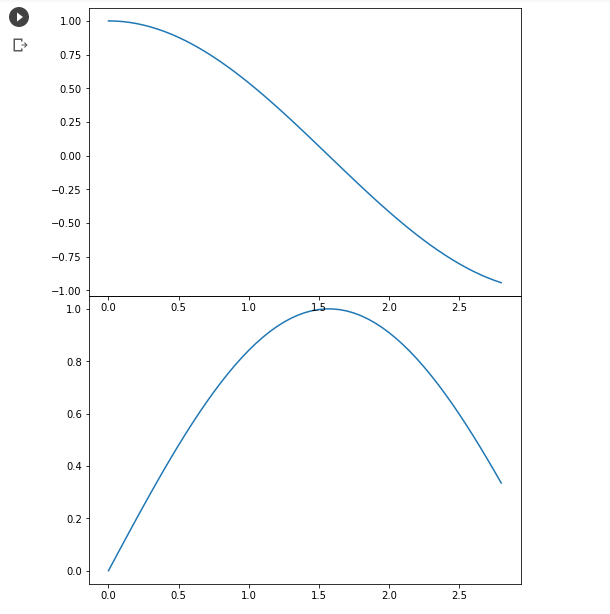
axes1.plot(x, y1)

axes2 = fig.add\_axes([0, 1, 1, 1])

axes2.plot(x, y2)

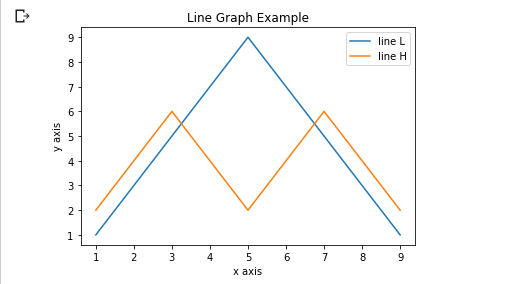
# Show plot

plt.show()



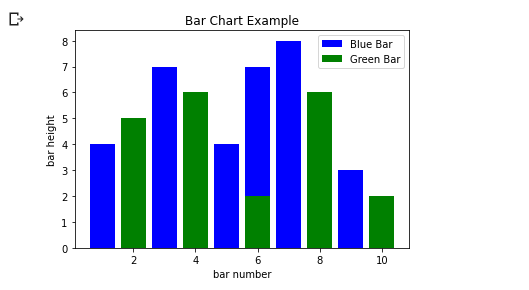
import matplotlib.pyplot as plt  
   
x = [1, 2, 3, 4, 5, 6, 7, 8, 9]  
y1 = [1, 3, 5, 7, 9, 7, 5, 3, 1]  
y2 = [2, 4, 6, 4, 2, 4, 6, 4, 2]  
plt.plot(x, y1, label="line L")  
plt.plot(x, y2, label="line H")  
plt.plot()  
   
plt.xlabel("x axis")  
plt.ylabel("y axis")  
plt.title("Line Graph Example")  
plt.legend()  
plt.show()

**Output:**



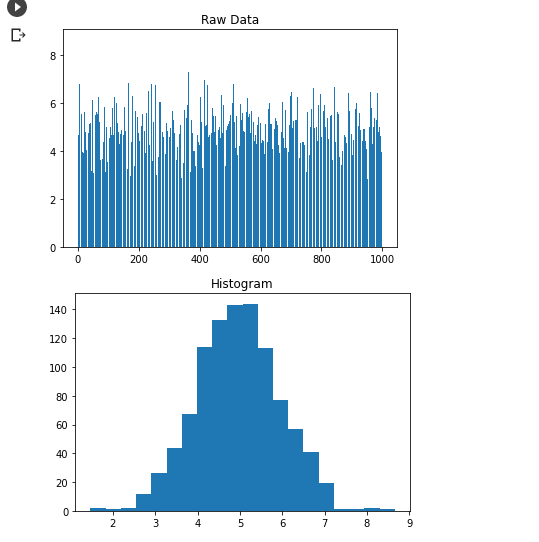
import matplotlib.pyplot as plt  
 x1 = [1, 3, 4, 5, 6, 7, 9]  
y1 = [4, 7, 2, 4, 7, 8, 3]  
   
x2 = [2, 4, 6, 8, 10]  
y2 = [5, 6, 2, 6, 2]  
  
plt.bar(x1, y1, label="Blue Bar", color='b')  
plt.bar(x2, y2, label="Green Bar", color='g')  
plt.plot()  
plt.xlabel("bar number")  
plt.ylabel("bar height")  
plt.title("Bar Chart Example")  
plt.legend()  
plt.show()

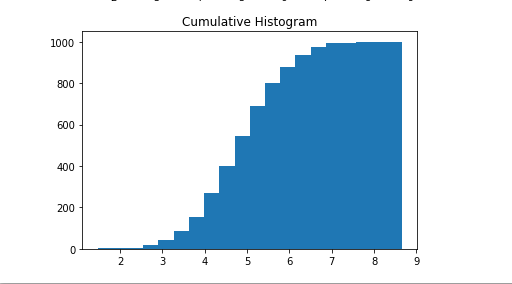
**Output:**



import matplotlib.pyplot as plt  
import numpy as np  
n = 5 + np.random.randn(1000)  
m = [m for m in range(len(n))]  
plt.bar(m, n)  
plt.title("Raw Data")  
plt.show()  
plt.hist(n, bins=20)  
plt.title("Histogram")  
plt.show()  
 plt.hist(n, cumulative=True, bins=20)  
plt.title("Cumulative Histogram")  
plt.show()

**Output:**





import matplotlib.pyplot as plt

x1 = [2, 3, 4]

y1 = [5, 5, 5]

x2 = [1, 2, 3, 4, 5]

y2 = [2, 3, 2, 3, 4]

y3 = [6, 8, 7, 8, 7]

plt.scatter(x1, y1)

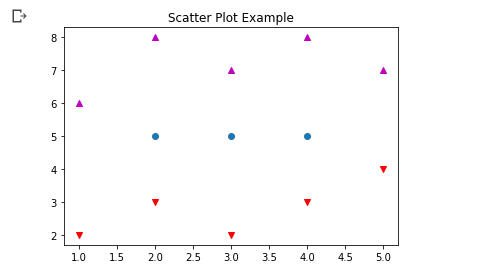
plt.scatter(x2, y2, marker='v', color='r')

plt.scatter(x2, y3, marker='^', color='m')

plt.title('Scatter Plot Example')

plt.show()

**Output:**



#Bubble chart

import plotly.express as px

df = px.data.iris()

fig = px.scatter(df, x="sepal\_width", y="sepal\_length",

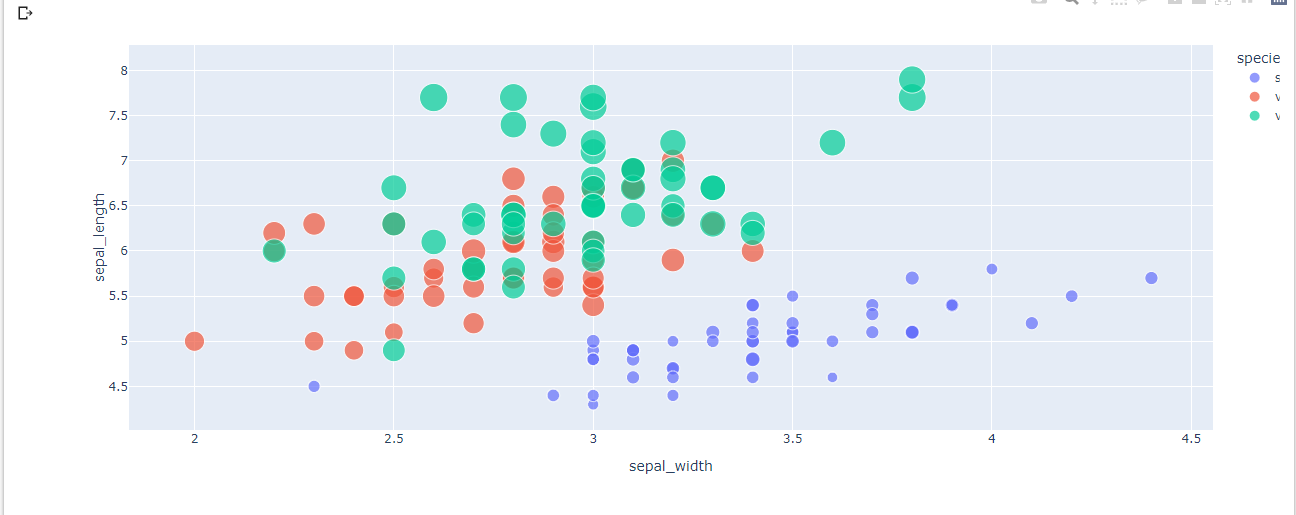
        color="species",

        size='petal\_length',

        hover\_data=['petal\_width'])

fig.show()

**Output:**



#distribution chart

import seaborn as sns

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

sns.displot(penguins, x="flipper\_length\_mm")

**Output:**

