DEPARTMENT OF COMPUTER APPLICATION

TKM COLLEGE OF ENGINEERING

KOLLAM – 691005

20MCA241 -DATA SCIENCE LAB

PRACTICAL RECORD BOOK

Third Semester MCA

2022-2023

Submitted by:

NAME: STEFI T

ROLL NO: TKM21MCA-2038

DEPARTMENT OF COMPUTER APPLICATION
TKM COLLEGE OF ENGINEERING
KOLLAM – 691005

Certificate

This is a bonafide record of the work done by **STEFI T (TKM21MCA-2038)** in the Third Semester in **DATA SCIENCE LAB** Course(20MCA241) towards the partial fulfillment of the degree of Master of Computer Applications during the academic year 2022-2023.

Staff Member in-charge

……..

Examiner

……..

CONTENTS

10	Program to implement k-means clustering technique using any standard dataset available in the public domain	56
	COURSE OUTCOME 4	
11	Programs on feedforward network to classify any standard dataset available in the public domain.	60
12	Programs on convolutional neural network to classify images from any standard dataset in the public domain.	72
	COURSE OUTCOME 5	
13	Web Data Mining • Implement a simple web crawler (ensure ethical conduct). • Implement a program to scrap the web page of any popular website – suggested python package is scrappy (ensure ethical conduct).	76
14	Natural Language Processing Problems may be designed for the following topics so that students can get hands on experience in using python for natural language processing: • Part of Speech tagging • N-gram	78

COURSE OUTCOME 1

PROGRAM NO:1

AIM: Review of python programming – Programs review the fundamentals of python

In [1]:
```
3+3
```

Out[1]:
```
6
```

In [2]:
```
4-3
```

Out[2]:
```
1
```

In [3]:
```
10*5
```

Out[3]:
```
50
```

In [4]:
```
8/2
```

Out[4]:
```
4.0
```

In [5]:
```
3**2
```

Out[5]:
```
9
```

In [6]:
```
9%2
```

Out[6]:
```
1
```

In [7]:
```
'hello world'
```

Out[7]:
```
'hello world'
```

In [8]:
```
x=13
y=20
z=x+y
print(z)
```

```
33
```

In [9]:
```
li=[1,2,3,4,5]
li.append(6)
li
```

Out[9]:
```
[1, 2, 3, 4, 5, 6]
```

In [10]:
```
li[2]
```

Out[10]:
```
3
```

In [11]:
```
li[0:2]
```

Out[11]:
```
[1, 2]
```

In [12]:
```
li[1:]
```

Out[12]:
```
[2, 3, 4, 5, 6]
```

In [13]:
```
d={'key1':'item1','key2':'item2','key3':'item3'}
d
```

Out[13]:
```
{'key1': 'item1', 'key2': 'item2', 'key3': 'item3'}
```

In [14]:
```
d['key1']
```

Out[14]:
```
'item1'
```

In [15]:
```
8>5
```

Out[15]:
```
True
```

In [16]:
```
3==5
```

Out[16]:
```
False
```

In [17]:
```
t=(1,2,3,4,5)
t
```

Out[17]:
```
(1, 2, 3, 4, 5)
```

In [18]:
```
t[2]
```

Out[18]:
```
3
```

In [19]:
```
s={1,2,3,4,5,6,7}
s
```

Out[19]:
```
{1, 2, 3, 4, 5, 6, 7}
```

In [21]:
```
a=[1,2,3,4,5,6,7,8,9,10]
for i in a:
    print(i)
```

```
1
2
3
4
5
6
7
8
9
10
```

In [22]:
```
i=1
while i<7 :
    print(i)
    i=i+1
```

```
1
2
3
4
5
6
```

In [23]:
```
(1>2)or(2<3)
```

Out[23]:
```
True
```

In [24]:
```
(3>4)and(4<5)
```

Out[24]:
```
False
```

In [25]:
```
if 2>3:
    print('false')
else:
    print('true')
```

```
true
```

In [26]:
```
if(1==2):
    print('equal')
elif(1<2):
    print("small")
else:
    print("large")
```

```
small
```

In [27]:
```
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

In [28]:
```
def fun(param1="default"):
    print(param1)
fun()
```

```
default
```

In [29]:
```
def square(a):
    print(a*a)
square(5)
```

```
25
```

In [30]:
```
def a(var):
    return var**2
a(5)
```

Out[30]:
```
25
```

RESULT: Program executed successfully and output is obtained

PROGRAM NO : 2

AIM : Matrix operations (using vectorization) and transformation using python and SVD using Python

PROGRAM CODE:

In [46]:
```
import numpy as np
```

In [47]:
```
list=[1,2,3,4]
```

In [48]:
```
np.array(list)
```

Out[48]:
```
array([1, 2, 3, 4])
```

In [49]:
```
list2=[[5,6,7],[6,9,2],[4,1,0]]
print(np.array(list2))
print(np.arange(0,10))
print(np.arange(0,11,2))
```

```
[[5 6 7]
 [6 9 2]
 [4 1 0]]
[0 1 2 3 4 5 6 7 8 9]
[ 0  2  4  6  8 10]
```

In [50]:
```
print(np.zeros(3))
print(np.zeros((3,3)))
```

```
[0. 0. 0.]
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

In [51]:
```
print(np.ones(3))
print(np.ones((3,3)))
print(np.eye(3))
```

```
[1. 1. 1.]
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

In [52]:
```
print(np.linspace(0,3,6))
print(np.linspace(0,10,100))
```

```
[0.  0.6 1.2 1.8 2.4 3. ]
[ 0.          0.1010101   0.2020202   0.3030303   0.4040404   0.50505051
  0.60606061  0.70707071  0.80808081  0.90909091  1.01010101  1.11111111
  1.21212121  1.31313131  1.41414141  1.51515152  1.61616162  1.71717172
  1.81818182  1.91919192  2.02020202  2.12121212  2.22222222  2.32323232
  2.42424242  2.52525253  2.62626263  2.72727273  2.82828283  2.92929293
  3.03030303  3.13131313  3.23232323  3.33333333  3.43434343  3.53535354
  3.63636364  3.73737374  3.83838384  3.93939394  4.04040404  4.14141414
  4.24242424  4.34343434  4.44444444  4.54545455  4.64646465  4.74747475
  4.84848485  4.94949495  5.05050505  5.15151515  5.25252525  5.35353535
  5.45454545  5.55555556  5.65656566  5.75757576  5.85858586  5.95959596
  6.06060606  6.16161616  6.26262626  6.36363636  6.46464646  6.56565657
  6.66666667  6.76767677  6.86868687  6.96969697  7.07070707  7.17171717
  7.27272727  7.37373737  7.47474747  7.57575758  7.67676768  7.77777778
```

```
  7.87878788  7.97979798  8.08080808  8.18181818  8.28282828  8.38383838
  8.48484848  8.58585859  8.68686869  8.78787879  8.88888889  8.98989899
  9.09090909  9.19191919  9.29292929  9.39393939  9.49494949  9.5959596
  9.6969697   9.7979798   9.8989899  10.        ]
```

In [53]:
```
print(np.random.rand(3))
print(np.random.rand(3,4))
```

```
[0.15234515 0.50232352 0.83823821]
[[0.04192374 0.82728732 0.38891065 0.73764902]
 [0.79475604 0.12035143 0.37720277 0.60935013]
 [0.46417229 0.60633732 0.99973196 0.24408305]]
```

In [54]:
```
print(np.random.randn(3))
print(np.random.randint(4,8,2))
```

```
[-0.03799443  1.10313584 -1.92235898]
[4 5]
```

In [55]:
```
l=np.arange(0,25)
print(l.reshape(5,5))
print(l.max())
print(l.min())
print(l.argmax())
print(l.argmin())
print(l.shape)
print(l.reshape(25,1))
print(l.dtype)
```

```
[[ 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]]
24
0
24
0
(25,)
[[ 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]]
int32
```

In [56]:
```
k=[[5,10,15],[20,25,30],[35,40,45]]
arr=np.array(k)
print(arr[1])
print(arr[1:3])
arr[1:2]=40
arr[:]=40
arr[1:3,0:2]
kk=arr>20
arr[kk]
```

```
[20 25 30]
[[20 25 30]
 [35 40 45]]
```

Out[56]:
```
array([40, 40, 40, 40, 40, 40, 40, 40, 40])
```

In [57]:
```
np.zeros(10)
```

Out[57]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

In [58]:
```
p=np.ones(10)
p*5
```

Out[58]:
```
array([5., 5., 5., 5., 5., 5., 5., 5., 5., 5.])
```

In [59]:
```
np.arange(10,51)
```

Out[59]:
```
array([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])
```

In [60]:
```
np.arange(10,51,2)
```

Out[60]:
```
array([10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42,
       44, 46, 48, 50])
```

In [61]:
```
v=np.arange(0,9)
v.reshape(3,3)
```

Out[61]:
```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

In [62]:
```
np.eye(3)
```

Out[62]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

In [63]:
```
np.random.rand(1)
```

Out[63]:
```
array([0.33406929])
```

In [64]:
```
np.random.randint(1,50,25)
```

Out[64]:
```
array([31,  2, 22, 10, 45, 40, 41,  7, 32, 10,  2, 18, 15, 39, 21, 41, 10,
       34, 18, 33, 41, 40, 27, 13, 46])
```

In [65]:
```
np.random.randn(1,50,25)
```

Out[65]:
```
array([[[-1.27020537, -2.50816124, -1.8123106 , ..., -0.44643465,
         -1.27732788, -0.07429487],
        [-0.58050628,  0.06997152,  0.05895656, ..., -0.19897518,
          0.39748334, -0.58418633],
        [-0.09690502, -1.02527908,  0.5135745 , ..., -1.17321249,
         -0.13731161,  0.33551079],
```

```
       ...,
       [-0.06085408,  0.23718939,  0.01660841, ...,  0.68693226,
         0.18994088,  0.95411884],
       [-0.37268881,  0.72918971, -0.14822244, ..., -0.0091523 ,
         0.16134298, -0.23245148],
       [ 0.96880241, -0.16401126, -0.11239686, ..., -0.72846091,
         0.00383111,  1.20346535]]])
```

In [66]:
```
np.random.randn(5,5)
```

Out[66]:
```
array([[ 1.91397306, -0.80778315, -1.26477433, -0.24550774, -0.25262497],
       [-0.08294887, -0.54768512, -0.50109335, -0.59469534,  2.01975351],
       [-0.53401172,  0.05154503,  0.61506534, -1.01315937,  2.29261458],
       [ 0.37342585, -1.04094139, -0.76481691,  0.4255788 ,  0.46166209],
       [ 0.26104523,  0.69180279, -0.12655302,  1.08715892, -0.92957177]])
```

In [67]:
```
ar=np.arange(1,101)
d=ar.reshape(10,10)
d/100
```

Out[67]:
```
array([[0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1 ],
       [0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19, 0.2 ],
       [0.21, 0.22, 0.23, 0.24, 0.25, 0.26, 0.27, 0.28, 0.29, 0.3 ],
       [0.31, 0.32, 0.33, 0.34, 0.35, 0.36, 0.37, 0.38, 0.39, 0.4 ],
       [0.41, 0.42, 0.43, 0.44, 0.45, 0.46, 0.47, 0.48, 0.49, 0.5 ],
       [0.51, 0.52, 0.53, 0.54, 0.55, 0.56, 0.57, 0.58, 0.59, 0.6 ],
       [0.61, 0.62, 0.63, 0.64, 0.65, 0.66, 0.67, 0.68, 0.69, 0.7 ],
       [0.71, 0.72, 0.73, 0.74, 0.75, 0.76, 0.77, 0.78, 0.79, 0.8 ],
       [0.81, 0.82, 0.83, 0.84, 0.85, 0.86, 0.87, 0.88, 0.89, 0.9 ],
       [0.91, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98, 0.99, 1.  ]])
```

In [68]:
```
l=np.arange(1,26).reshape(5,5)
l.sum(axis=0)
```

Out[68]:
```
array([55, 60, 65, 70, 75])
```

In [69]:
```
from numpy.linalg import svd
```

In [70]:
```
a=np.array([[1,-.8],[0,1],[1,0]])
```

In [71]:
```
print (a)
```

```
[[ 1.  -0.8]
 [ 0.   1. ]
 [ 1.   0. ]]
```

In [72]:
```
a.shape
```

Out[72]:
```
(3, 2)
```

In [73]:
```
U,S,VT=svd(a,full_matrices=True)
```

In [74]:
```
a.shape
```

Out[74]:
```
(3, 2)
```

In [75]:
```
print(U.shape,S.shape,VT.shape)
```

```
(3, 3) (2,) (2, 2)
```

In [76]:
```
U,S,VT=svd(a,full_matrices=False)
```

In [77]:
```
print(U.shape,S.shape,VT.shape)
```

```
(3, 2) (2,) (2, 2)
```

In [78]:
```
U
```

Out[78]:
```
array([[-7.88170109e-01,  1.87057766e-16],
       [ 3.84473224e-01, -7.80868809e-01],
       [-4.80591530e-01, -6.24695048e-01]])
```

In [79]:
```
S
```

Out[79]:
```
array([1.62480768, 1.        ])
```

In [80]:
```
VT
```

Out[80]:
```
array([[-0.78086881,  0.62469505],
       [-0.62469505, -0.78086881]])
```

MATRIX OPERATIONS

In [81]:
```
a=np.arange(0,10).reshape(5,2)
print(a)
```

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

In [82]:
```
b=np.arange(11,21).reshape(2,5)
print(b)
```

```
[[11 12 13 14 15]
 [16 17 18 19 20]]
```

In [83]:
```
np.dot(a,b)
```

Out[83]:
```
array([[ 16,  17,  18,  19,  20],
       [ 70,  75,  80,  85,  90],
       [124, 133, 142, 151, 160],
       [178, 191, 204, 217, 230],
       [232, 249, 266, 283, 300]])
```

In [84]:
```
array_1 = map(int, input().split())
```

```
22
```

In [85]:
```
array_1
```

Out[85]:
```
<map at 0x2a2f5b41840>
```

RESULT: Program is executed successfully and output is obtained

PROGRAM NO : 3

AIM: Programs using matplotlib / plotly / bokeh / seaborn for data visualisation

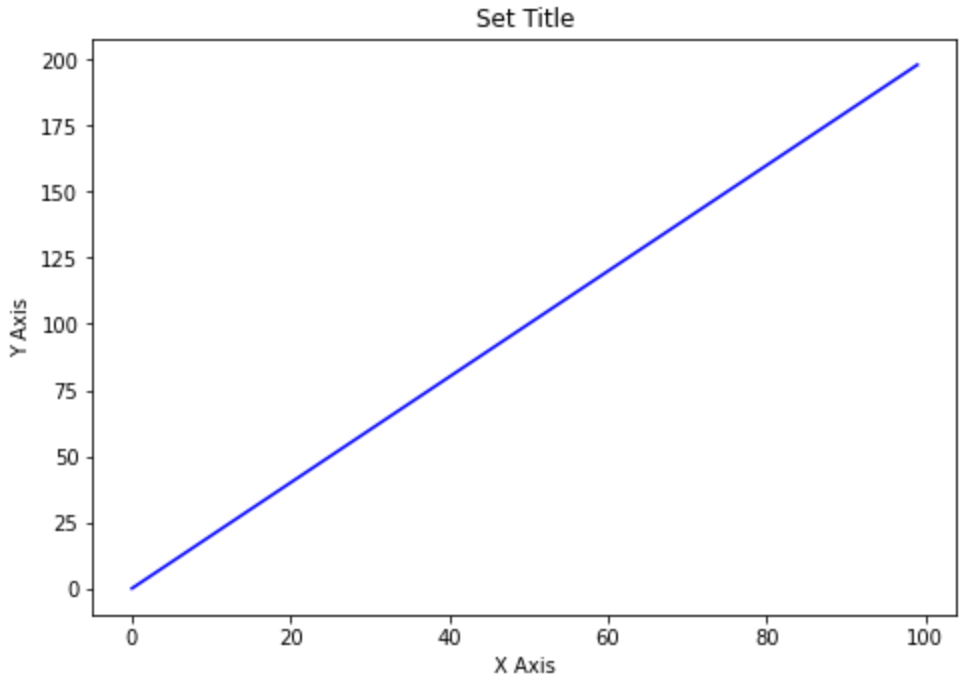
PROGRAM CODE:

In [3]:
```
import numpy as np
x = np.arange(0,100)
y = x*2
z = x**2
```

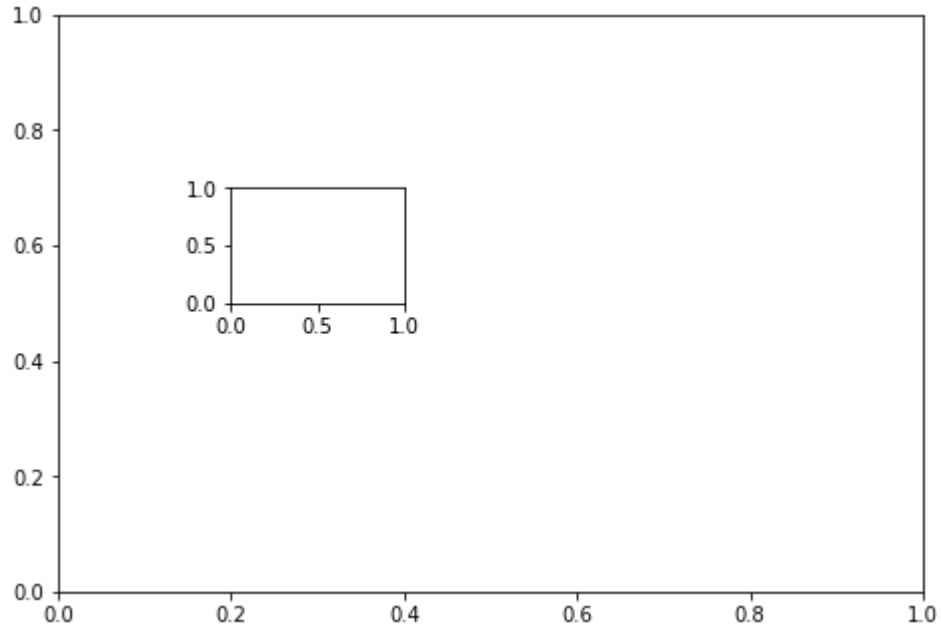
In [4]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

In [5]:
```
fig=plt.figure()
ax = fig.add_axes([0., 0., 1, 1])
ax.plot(x, y, 'b')
ax.set_xlabel('X Axis')
ax.set_ylabel('Y Axis')
ax.set_title('Set Title')
```

Out[5]:
```
Text(0.5, 1.0, 'Set Title')
```



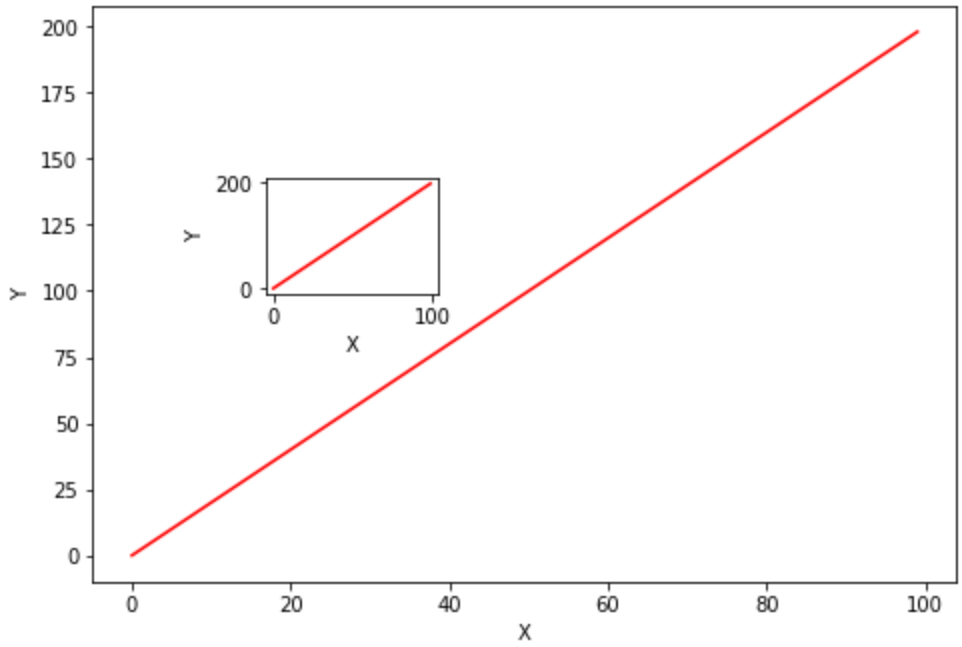
In [6]:
```
fig = plt.figure()
ax1 = fig.add_axes([0, 0, 1, 1])
ax2 = fig.add_axes([0.2, 0.5, 0.2, 0.2])
```



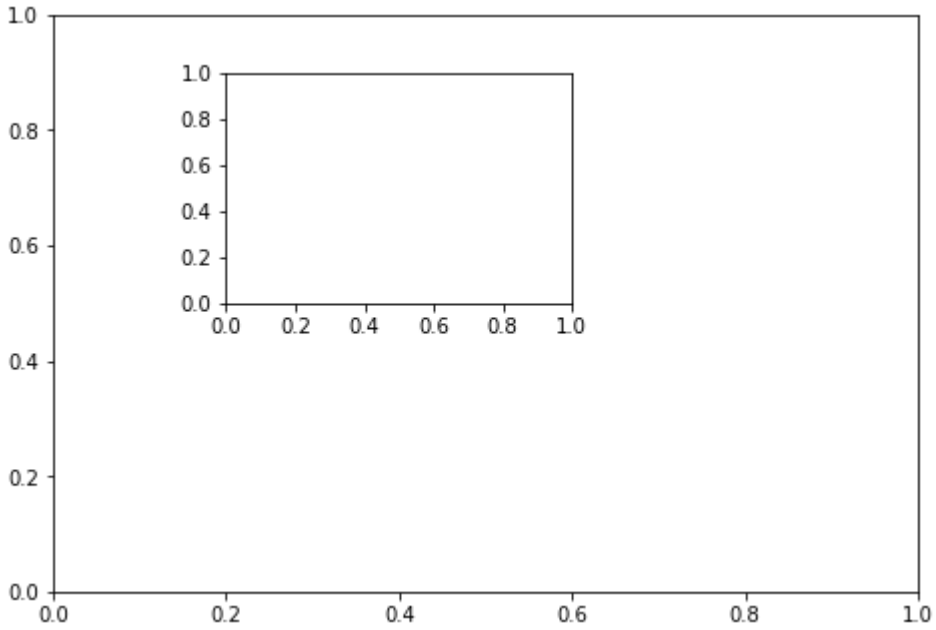
Now plot (x,y) on both axes. And call your figure object to show it.

In [7]:
```
fig = plt.figure()
ax1 = fig.add_axes([0, 0, 1, 1])
ax2 = fig.add_axes([0.2, 0.5, 0.2, 0.2])
ax1.plot(x, y, 'r')
ax1.set_xlabel('X')
ax1.set_ylabel('Y')
ax2.plot(x, y, 'r')
ax2.set_xlabel('X')
ax2.set_ylabel('Y')
```

Out[7]:
```
Text(0, 0.5, 'Y')
```

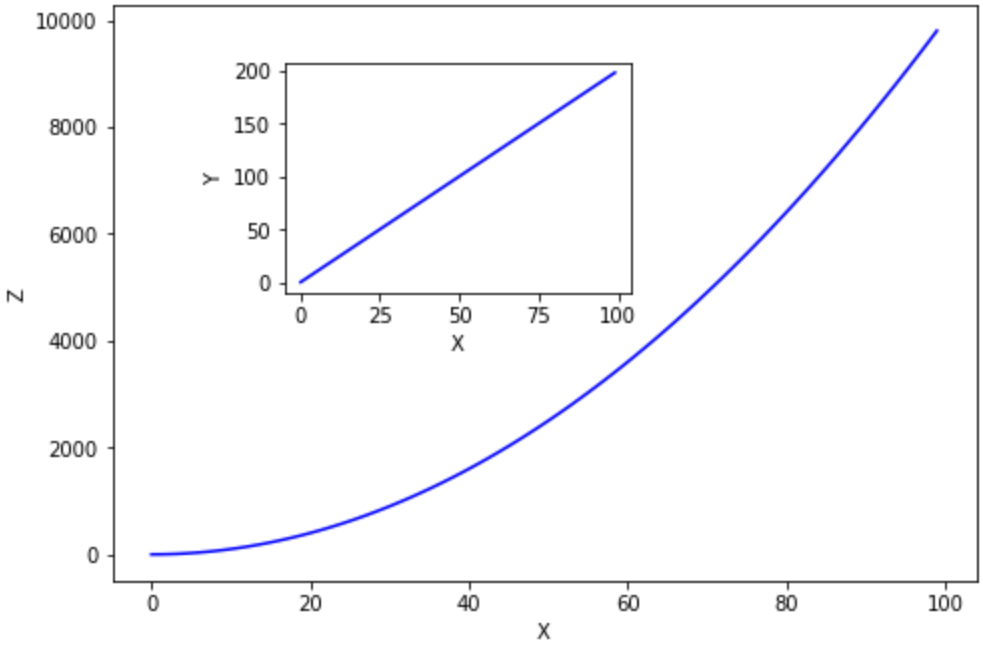


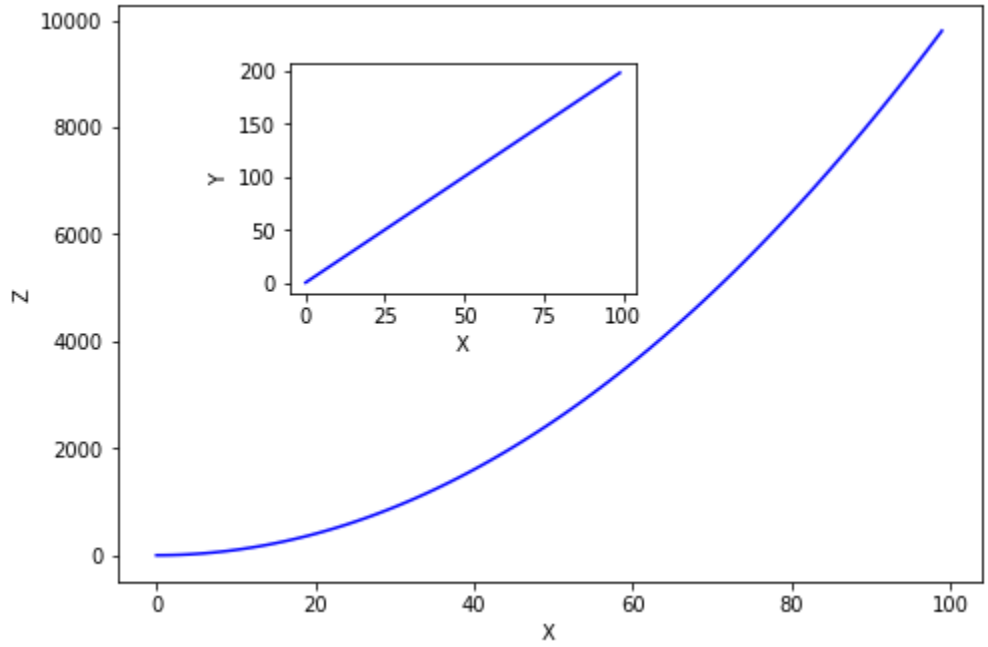
In [8]:
```
fig = plt.figure()
ax1 = fig.add_axes([0, 0, 1, 1])
ax2 = fig.add_axes([0.2, 0.5, 0.4, 0.4])
```



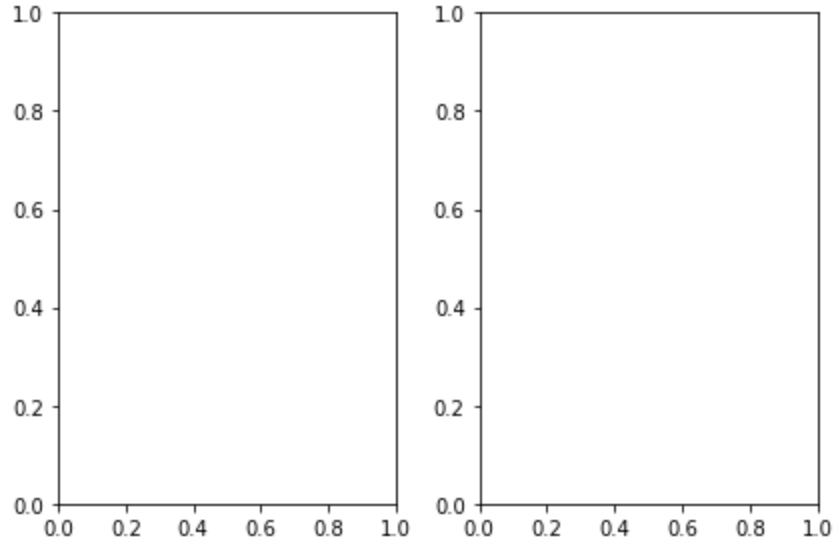
In [9]:
```
fig = plt.figure()
ax1 = fig.add_axes([0, 0, 1, 1])
ax2 = fig.add_axes([0.2, 0.5, 0.4, 0.4])
ax1.plot(x, z, 'b')
ax1.set_xlabel('X')
ax1.set_ylabel('Z')
ax2.plot(x, y, 'b')
ax2.set_xlabel('X')
ax2.set_ylabel('Y')
fig
```

Out[9]:

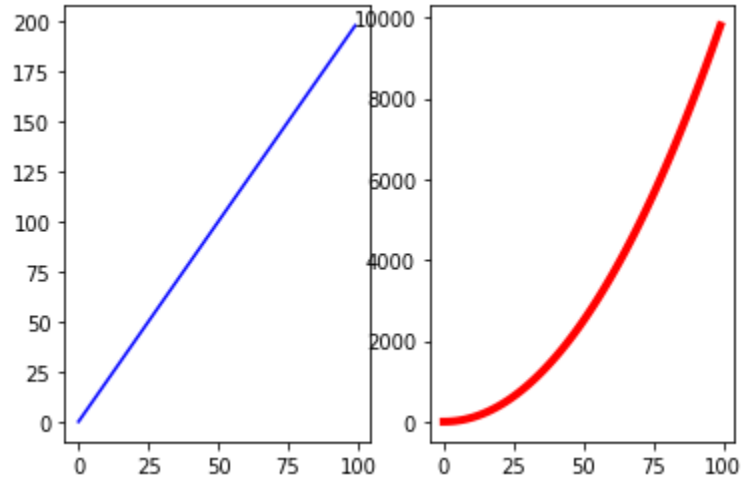




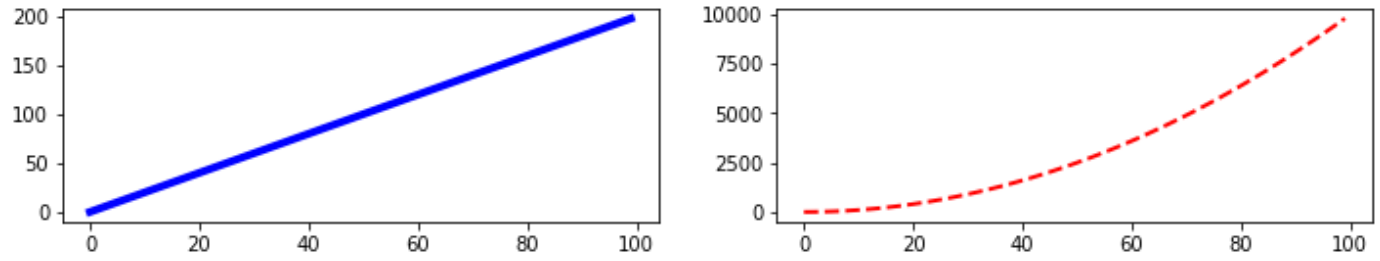
In [10]:
```
fig, axes = plt.subplots(nrows=1, ncols=2)
fig
plt.tight_layout()
```



In [11]:
```
plt.subplot(1,2,1)
plt.plot(x, y, 'b-')
plt.subplot(1,2,2)
plt.plot(x, z, 'r',  linewidth=4);
```

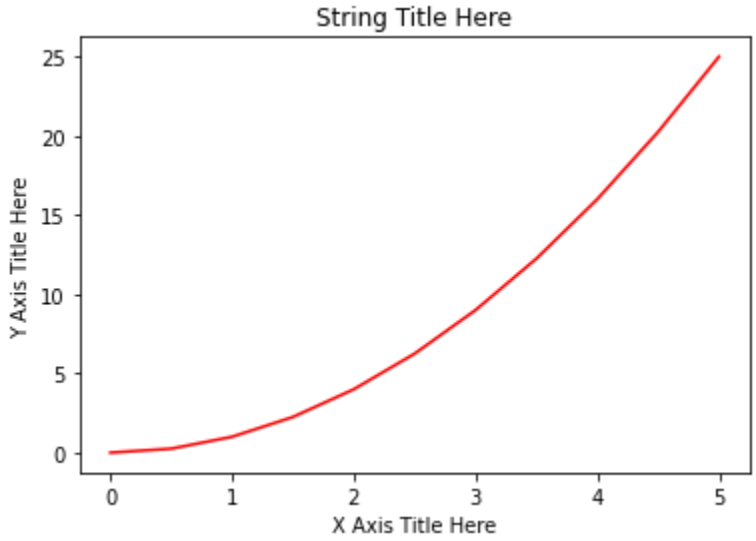


In [12]:
```
fig, axes = plt.subplots(1, 2, figsize=(12, 2))

plt.subplot(1,2,1)
plt.plot(x, y, 'b-',  linewidth=4)
plt.subplot(1,2,2)
plt.plot(x, z, 'r--', linewidth=2);
```

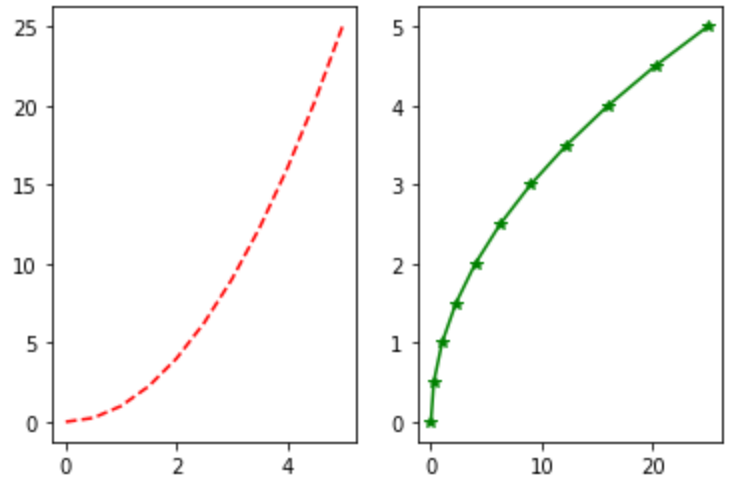


In [13]:
```
import numpy as np
x = np.linspace(0, 5, 11)
y = x ** 2
```

In [14]:
```
plt.plot(x, y, 'r') # 'r' is the color red
plt.xlabel('X Axis Title Here')
plt.ylabel('Y Axis Title Here')
plt.title('String Title Here')
plt.show()
```

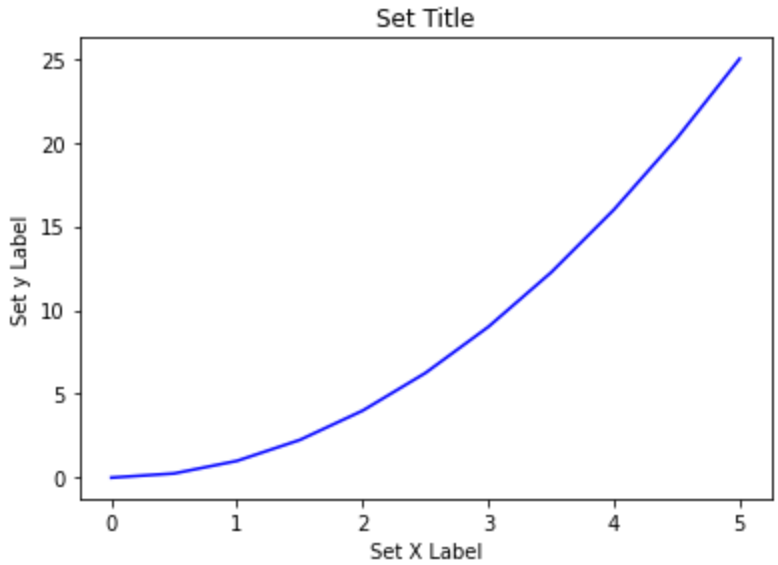


In [15]:
```
# plt.subplot(nrows, ncols, plot_number)
plt.subplot(1,2,1)
plt.plot(x, y, 'r--') # More on color options later
plt.subplot(1,2,2)
plt.plot(y, x, 'g*-');
```



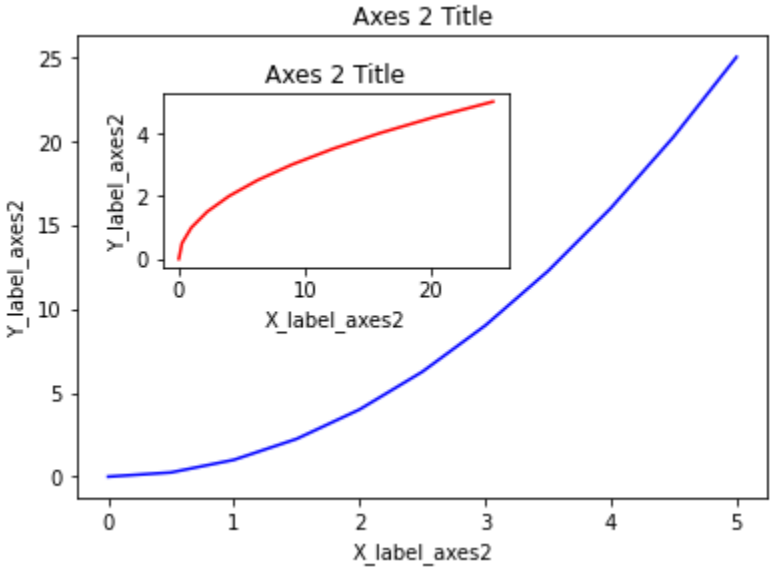
In [16]:
```
# Create Figure (empty canvas)
fig = plt.figure()

# Add set of axes to figure
axes = fig.add_axes([0.1, 0.1, 0.8, 0.8]) # left, bottom, width, height (range 0 to 1)

# Plot on that set of axes
axes.plot(x, y, 'b')
axes.set_xlabel('Set X Label') # Notice the use of set_ to begin methods
axes.set_ylabel('Set y Label')
axes.set_title('Set Title')
```

Out[16]:
```
Text(0.5, 1.0, 'Set Title')
```

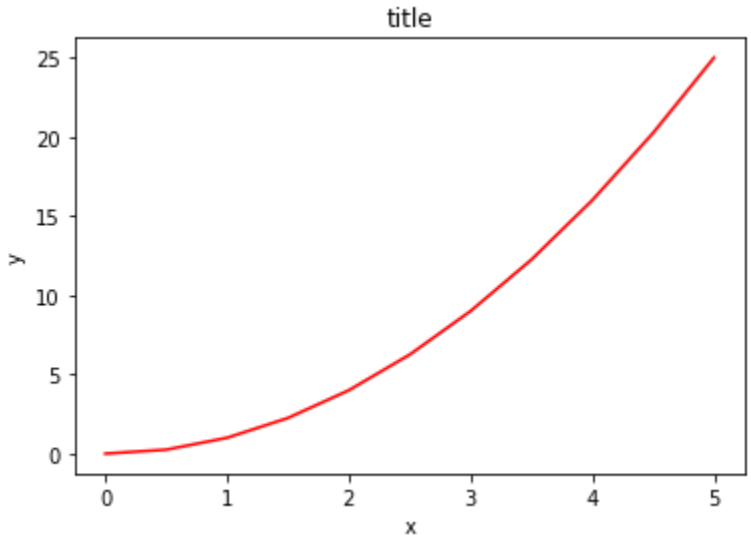


In [17]:
```
# Creates blank canvas
fig = plt.figure()

axes1 = fig.add_axes([0.1, 0.1, 0.8, 0.8]) # main axes
axes2 = fig.add_axes([0.2, 0.5, 0.4, 0.3]) # inset axes

# Larger Figure Axes 1
axes1.plot(x, y, 'b')
axes1.set_xlabel('X_label_axes2')
axes1.set_ylabel('Y_label_axes2')
axes1.set_title('Axes 2 Title')

# Insert Figure Axes 2
axes2.plot(y, x, 'r')
axes2.set_xlabel('X_label_axes2')
```

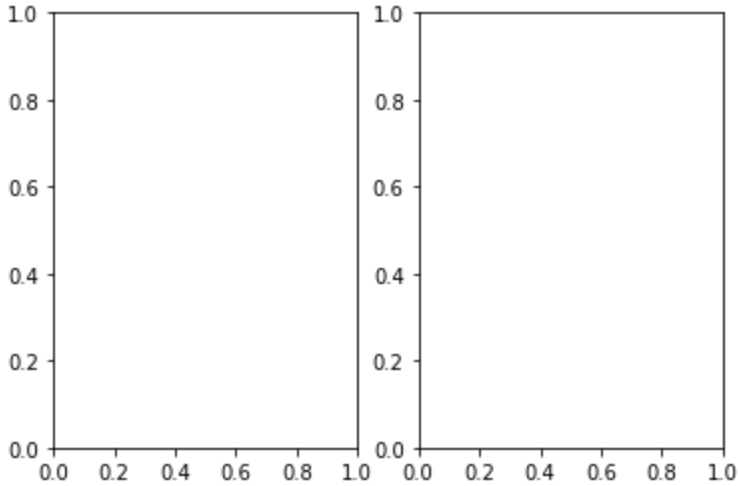
```
axes2.set_ylabel('Y_label_axes2')
axes2.set_title('Axes 2 Title');
```



In [18]:
```
# Use similar to plt.figure() except use tuple unpacking to grab fig and axes
fig, axes = plt.subplots()

# Now use the axes object to add stuff to plot
axes.plot(x, y, 'r')
axes.set_xlabel('x')
axes.set_ylabel('y')
axes.set_title('title');
```

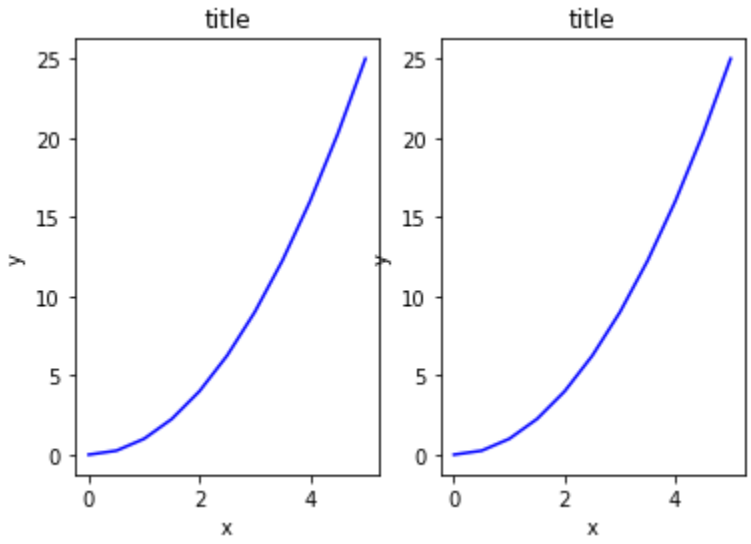


In [19]:
```
# Empty canvas of 1 by 2 subplots
fig, axes = plt.subplots(nrows=1, ncols=2)
```

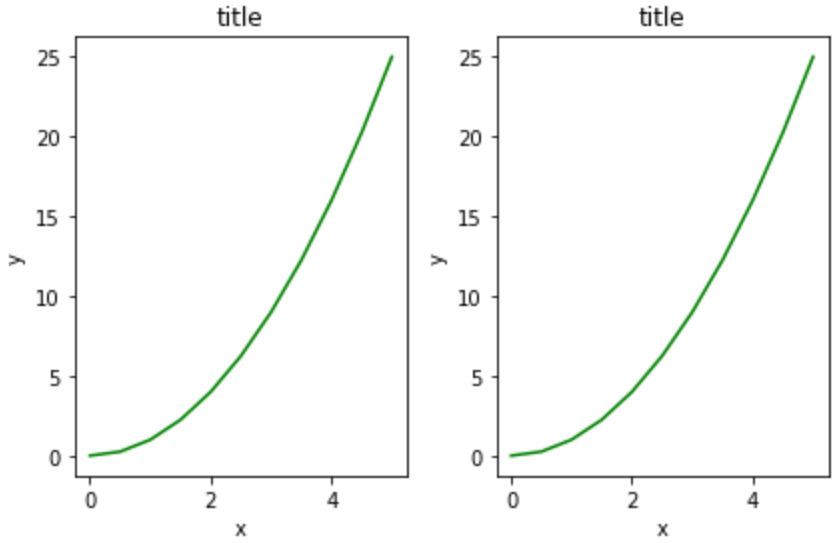


In [20]:
```
for ax in axes:
    ax.plot(x, y, 'b')
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_title('title')

# Display the figure object
fig
```

Out[20]:



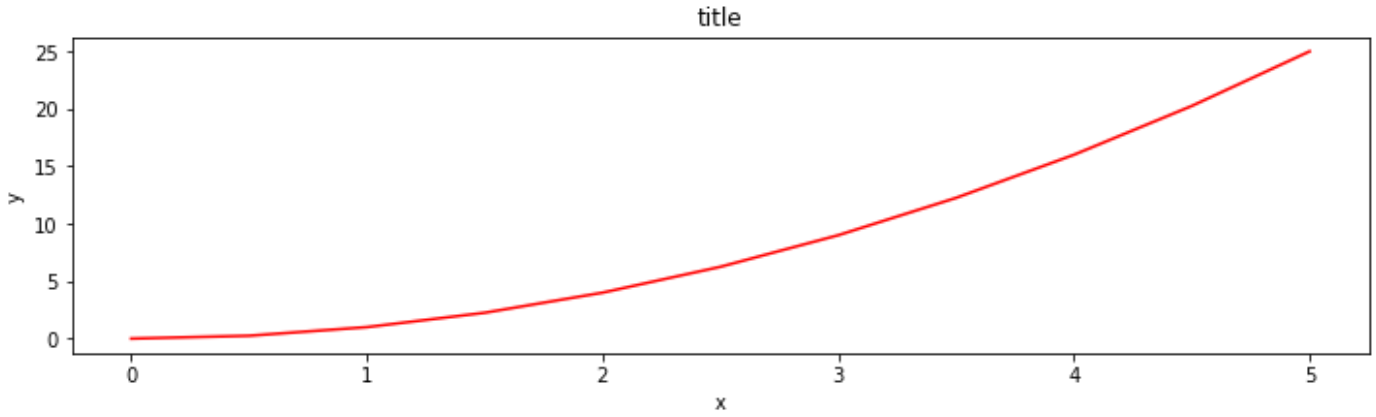
In [21]:
```
fig, axes = plt.subplots(nrows=1, ncols=2)

for ax in axes:
    ax.plot(x, y, 'g')
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_title('title')

fig
plt.tight_layout()
```



In [22]:
```
fig = plt.figure(figsize=(8,4), dpi=100)
```

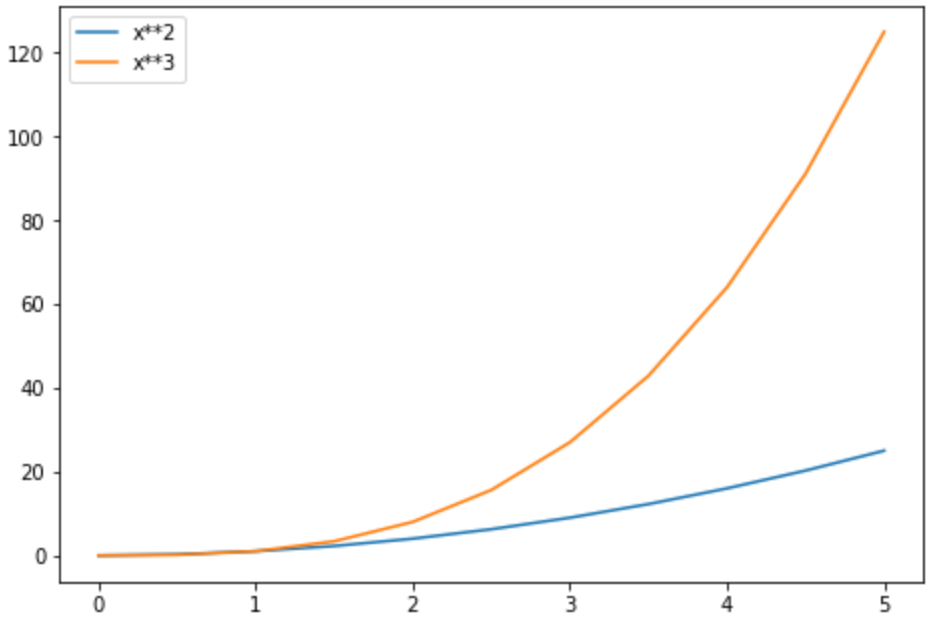
```
<Figure size 800x400 with 0 Axes>
```

In [23]:
```
fig, axes = plt.subplots(figsize=(12,3))

axes.plot(x, y, 'r')
axes.set_xlabel('x')
axes.set_ylabel('y')
axes.set_title('title');
```



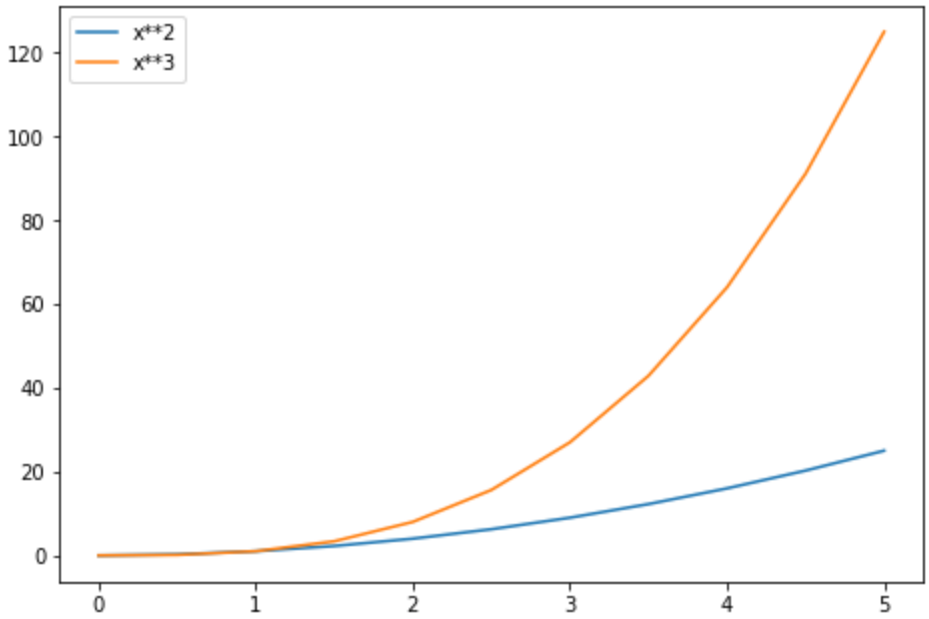
In [24]:
```
fig = plt.figure()

ax = fig.add_axes([0,0,1,1])

ax.plot(x, x**2, label="x**2")
ax.plot(x, x**3, label="x**3")
ax.legend()
```

Out[24]:
```
<matplotlib.legend.Legend at 0x20ce1ea6680>
```



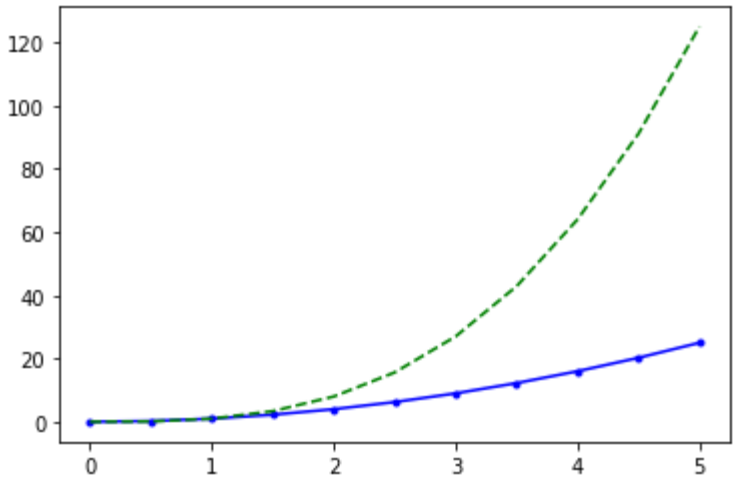
In [25]:
```
# Lots of options....

ax.legend(loc=1) # upper right corner
ax.legend(loc=2) # upper left corner
ax.legend(loc=3) # lower left corner
ax.legend(loc=4) # lower right corner

# .. many more options are available

# Most common to choose
ax.legend(loc=0) # let matplotlib decide the optimal location
fig
```

Out[25]:



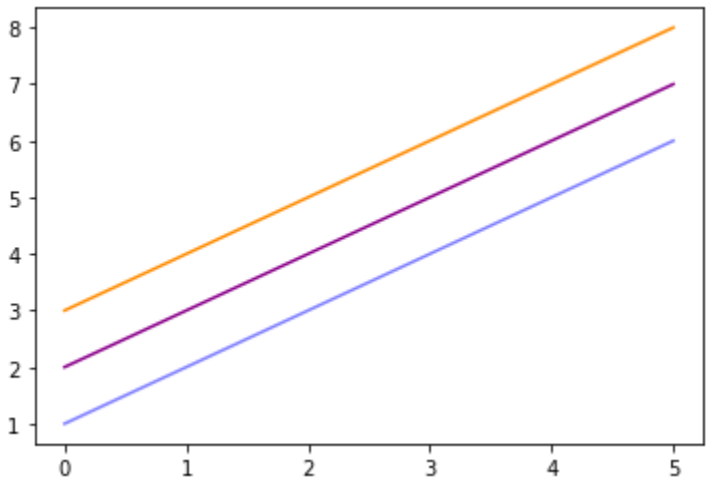
In [26]:
```
# MATLAB style line color and style
fig, ax = plt.subplots()
ax.plot(x, x**2, 'b.-') # blue line with dots
ax.plot(x, x**3, 'g--') # green dashed line
```

Out[26]:
```
[<matplotlib.lines.Line2D at 0x20ce0d22230>]
```



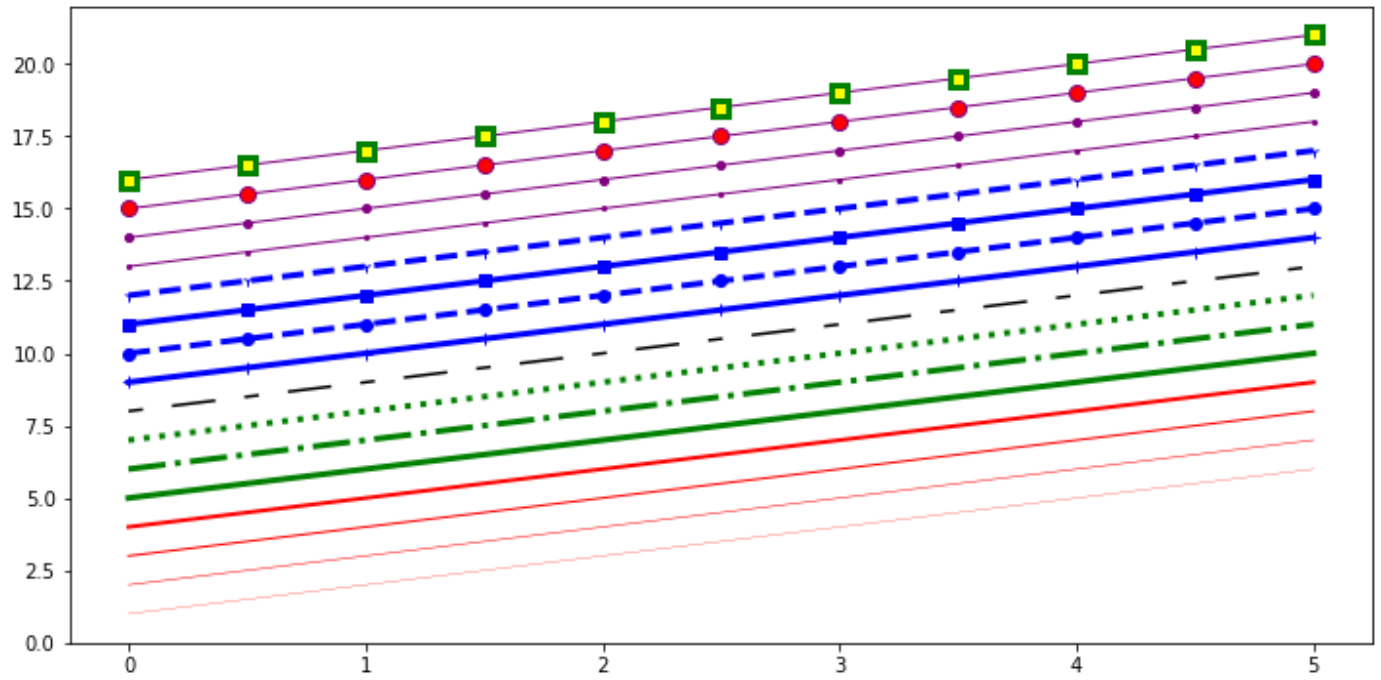
In [27]:
```
fig, ax = plt.subplots()

ax.plot(x, x+1, color="blue", alpha=0.5) # half-transparant
ax.plot(x, x+2, color="#8B008B")         # RGB hex code
ax.plot(x, x+3, color="#FF8C00")         # RGB hex code
```

Out[27]:
```
[<matplotlib.lines.Line2D at 0x20ce0cf96c0>]
```



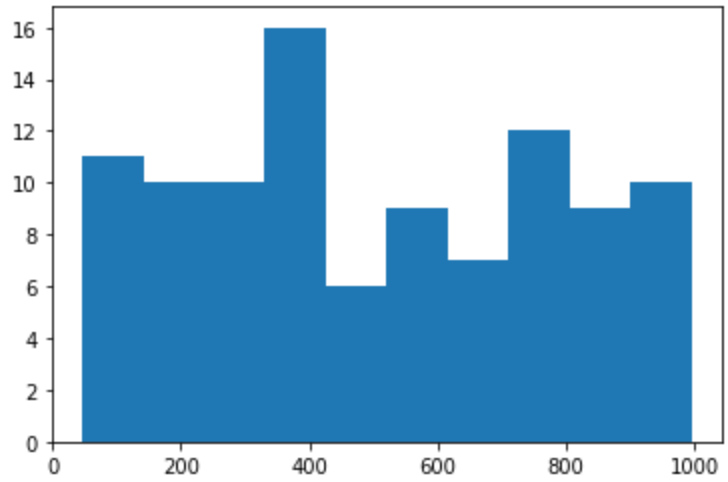
In [28]:
```
fig, ax = plt.subplots(figsize=(12,6))

ax.plot(x, x+1, color="red", linewidth=0.25)
ax.plot(x, x+2, color="red", linewidth=0.50)
ax.plot(x, x+3, color="red", linewidth=1.00)
ax.plot(x, x+4, color="red", linewidth=2.00)

# possible linestype options '-', '--', '-.', ':', 'steps'
ax.plot(x, x+5, color="green", lw=3, linestyle='-')
ax.plot(x, x+6, color="green", lw=3, ls='-.')
ax.plot(x, x+7, color="green", lw=3, ls=':')

# custom dash
line, = ax.plot(x, x+8, color="black", lw=1.50)
line.set_dashes([5, 10, 15, 10]) # format: line length, space length, ...

# possible marker symbols: marker = '+', 'o', '*', 's', ',', '.', '1', '2', '3', '4', ..
ax.plot(x, x+ 9, color="blue", lw=3, ls='-', marker='+')
ax.plot(x, x+10, color="blue", lw=3, ls='--', marker='o')
ax.plot(x, x+11, color="blue", lw=3, ls='-', marker='s')
ax.plot(x, x+12, color="blue", lw=3, ls='--', marker='1')

# marker size and color
ax.plot(x, x+13, color="purple", lw=1, ls='-', marker='o', markersize=2)
```

```
ax.plot(x, x+14, color="purple", lw=1, ls='-', marker='o', markersize=4)
ax.plot(x, x+15, color="purple", lw=1, ls='-', marker='o', markersize=8, markerfacecolor
ax.plot(x, x+16, color="purple", lw=1, ls='-', marker='s', markersize=8,
        markerfacecolor="yellow", markeredgewidth=3, markeredgecolor="green");
```

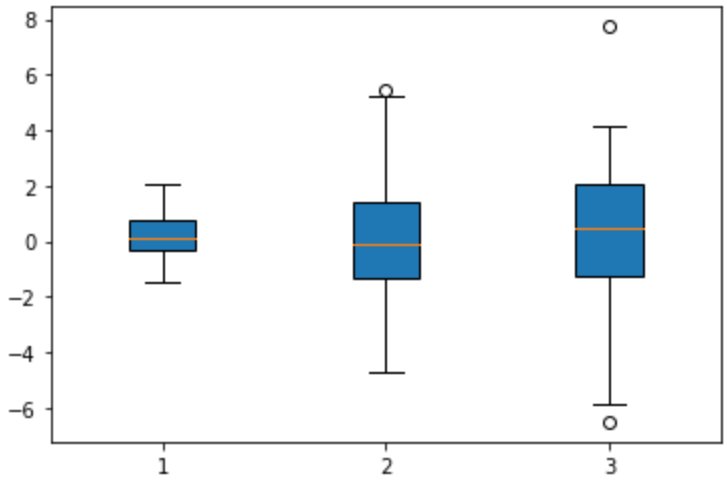


In [29]:
```
from random import sample
data = sample(range(1, 1000), 100)
plt.hist(data)
```

Out[29]:
```
(array([11., 10., 10., 16.,  6.,  9.,  7., 12.,  9., 10.]),
 array([ 47. , 141.7, 236.4, 331.1, 425.8, 520.5, 615.2, 709.9, 804.6,
        899.3, 994. ]),
 <BarContainer object of 10 artists>)
```



In [30]:
```
data = [np.random.normal(0, std, 100) for std in range(1, 4)]

# rectangular box plot
plt.boxplot(data,vert=True,patch_artist=True);
```



RESULT: Program is executed successfully and output is obtained

PROGRAM NO: 4

AIM: Programs to handle data using pandas.

PROGRAM CODE:

In [1]:
```
import numpy as np
import pandas as pd
```

In [2]:
```
labels=['a','b','c']
my_list=[10,20,30]
arr = np.array([10,20,30])
d = {'a':10,'b':20,'c':30}
```

In [3]:
```
pd.Series(data=my_list)
```

Out[3]:
```
0    10
1    20
2    30
dtype: int64
```

In [4]:
```
pd.Series(data=my_list,index=labels)
```

Out[4]:
```
a    10
b    20
c    30
dtype: int64
```

In [5]:
```
pd.Series(my_list,labels)
```

Out[5]:
```
a    10
b    20
c    30
dtype: int64
```

In [6]:
```
pd.Series(arr)
```

Out[6]:
```
0    10
1    20
2    30
dtype: int32
```

In [7]:
```
pd.Series(arr,labels)
```

Out[7]:
```
a    10
b    20
c    30
dtype: int32
```

In [8]:
```
pd.Series(d)
```

Out[8]:
```
a    10
b    20
c    30
dtype: int64
```

In [9]:
```
pd.Series(data=labels)
```

Out[9]:
```
0    a
1    b
2    c
dtype: object
```

In [10]:
```
pd.Series([sum,print,len])
```

Out[10]:
```
0      <built-in function sum>
1    <built-in function print>
2      <built-in function len>
dtype: object
```

In [11]:
```
ser1 = pd.Series([1,2,3,4],index = ['USA', 'Germany','USSR', 'Japan'])
```

In [12]:
```
ser1
```

Out[12]:
```
USA        1
Germany    2
USSR       3
Japan      4
dtype: int64
```

In [13]:
```
ser2 = pd.Series([1,7,4,6],index = ['USA', 'Germany','USSR', 'Japan'])
```

In [14]:
```
ser1['USA']
```

Out[14]:
```
1
```

In [15]:
```
ser1 + ser2
```

Out[15]:
```
USA         2
Germany     9
USSR        7
Japan      10
dtype: int64
```

Dataframes

In [16]:
```
from numpy.random import randn
np.random.seed(101)
df = pd.DataFrame(randn(5,4),index='A B C D E'.split(),columns='W X Y Z'.split())
```

In [17]:
```
df
```

Out[17]:

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

In [18]:
```
df['W']
```

Out[18]:
```
A    2.706850
B    0.651118
C   -2.018168
D    0.188695
E    0.190794
Name: W, dtype: float64
```

In [19]:
```
df[['W','Z']]
```

Out[19]:

	W	Z

A	2.706850	0.503826
B	0.651118	0.605965
C	-2.018168	-0.589001
D	0.188695	0.955057
E	0.190794	0.683509

In [20]:
```
type(df['W'])
```

Out[20]:
```
pandas.core.series.Series
```

In [21]:
```
df['new'] = df['W'] + df['Y']
```

In [22]:
```
df
```

Out[22]:

	W	X	Y	Z	new
A	2.706850	0.628133	0.907969	0.503826	3.614819
B	0.651118	-0.319318	-0.848077	0.605965	-0.196959
C	-2.018168	0.740122	0.528813	-0.589001	-1.489355
D	0.188695	-0.758872	-0.933237	0.955057	-0.744542
E	0.190794	1.978757	2.605967	0.683509	2.796762

In [23]:
```
df.drop('new',axis=1)
```

Out[23]:

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

In [24]:
```
# Not inplace unless specified!
df
```

Out[24]:

	W	X	Y	Z	new
A	2.706850	0.628133	0.907969	0.503826	3.614819
B	0.651118	-0.319318	-0.848077	0.605965	-0.196959
C	-2.018168	0.740122	0.528813	-0.589001	-1.489355
D	0.188695	-0.758872	-0.933237	0.955057	-0.744542
E	0.190794	1.978757	2.605967	0.683509	2.796762

In [25]:
```
df.drop('new',axis=1,inplace=True)
df
```

Out[25]:

	W	X	Y	Z

A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

In [26]:
```
df.drop('E',axis=0)
```

Out[26]:

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057

In [27]:
```
df.loc['A']
```

Out[27]:
```
W    2.706850
X    0.628133
Y    0.907969
Z    0.503826
Name: A, dtype: float64
```

In [28]:
```
df.iloc[2]
```

Out[28]:
```
W   -2.018168
X    0.740122
Y    0.528813
Z   -0.589001
Name: C, dtype: float64
```

In [29]:
```
df.loc['B','Y']
```

Out[29]:
```
-0.8480769834036315
```

In [30]:
```
df.loc[['A','B'],['W','Y']]
```

Out[30]:

	W	Y
A	2.706850	0.907969
B	0.651118	-0.848077

In [31]:
```
df>0
```

Out[31]:

	W	X	Y	Z
A	True	True	True	True
B	True	False	False	True
C	False	True	True	False
D	True	False	False	True
E	True	True	True	True

In [32]:
```
df[df>0]
```

Out[32]:

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	NaN	NaN	0.605965
C	NaN	0.740122	0.528813	NaN
D	0.188695	NaN	NaN	0.955057
E	0.190794	1.978757	2.605967	0.683509

In [33]:
```
df[df['W']>0]
```

Out[33]:

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

In [34]:
```
df[df['W']>0]['Y']
```

Out[34]:
```
A    0.907969
B   -0.848077
D   -0.933237
E    2.605967
Name: Y, dtype: float64
```

In [35]:
```
df[df['W']>0][['Y','X']]
```

Out[35]:

	Y	X
A	0.907969	0.628133
B	-0.848077	-0.319318
D	-0.933237	-0.758872
E	2.605967	1.978757

Index Details

In [36]:
```
df
```

Out[36]:

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

In [37]:
```
df.reset_index()
```

Out[37]:

	index	W	X	Y	Z
0	A	2.706850	0.628133	0.907969	0.503826
1	B	0.651118	-0.319318	-0.848077	0.605965
2	C	-2.018168	0.740122	0.528813	-0.589001
3	D	0.188695	-0.758872	-0.933237	0.955057
4	E	0.190794	1.978757	2.605967	0.683509

In [38]:
```
newind = 'CA NY WR OR CO'.split()
```

In [39]:
```
df['States']=newind
```

In [40]:
```
df
```

Out[40]:

	W	X	Y	Z	States
A	2.706850	0.628133	0.907969	0.503826	CA
B	0.651118	-0.319318	-0.848077	0.605965	NY
C	-2.018168	0.740122	0.528813	-0.589001	WR
D	0.188695	-0.758872	-0.933237	0.955057	OR
E	0.190794	1.978757	2.605967	0.683509	CO

In [41]:
```
df.set_index('States')
```

Out[41]:

	W	X	Y	Z
States				
CA	2.706850	0.628133	0.907969	0.503826
NY	0.651118	-0.319318	-0.848077	0.605965
WR	-2.018168	0.740122	0.528813	-0.589001
OR	0.188695	-0.758872	-0.933237	0.955057
CO	0.190794	1.978757	2.605967	0.683509

RESULT: Program is executed sucessfully and output is obtained

COURSE OUTCOME 2

PROGRAM NO: 5

AIM: Program to implement k-NN classification using any standard dataset available in the public domain and find the accuracy of the algorithm.

Dataset: sonar_csv

ALGORITHM:

1. Import Libraries
2. Read the data
3. Standardize the Variables
4. Import KNeighboursClassifier
5. Split the dataset into training and testing sets
6. Fit the model to the KNeigbours Classifier
7. Evaluate the model by predicting the test values
8. Find the accuracy of the model by printing the classification report and confusion matrix
9. Analyze better k value through iterations

PROGRAM CODE:

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Step 1: Import Libraries

Step 2: Load dataset

Step 3: Preprocessing the dataset if necessary

Step 4: Split the dataset into two train and test

Step 5: Build the MODEL

Step 6: Predict the test data

Step 7: Calculate the Accuracy using Confusion Matrix and Classification Report

In [4]:
```
dataset=pd.read_csv('sonar data.csv')
```

In [5]:
```
dataset
```

Out[5]:

	0.0200	0.0371	0.0428	0.0207	0.0954	0.0986	0.1539	0.1601	0.3109	0.2111	...	0.0027	0.0065	0.0159
0	0.0453	0.0523	0.0843	0.0689	0.1183	0.2583	0.2156	0.3481	0.3337	0.2872	...	0.0084	0.0089	0.0048
1	0.0262	0.0582	0.1099	0.1083	0.0974	0.2280	0.2431	0.3771	0.5598	0.6194	...	0.0232	0.0166	0.0095
2	0.0100	0.0171	0.0623	0.0205	0.0205	0.0368	0.1098	0.1276	0.0598	0.1264	...	0.0121	0.0036	0.0150
3	0.0762	0.0666	0.0481	0.0394	0.0590	0.0649	0.1209	0.2467	0.3564	0.4459	...	0.0031	0.0054	0.0105
4	0.0286	0.0453	0.0277	0.0174	0.0384	0.0990	0.1201	0.1833	0.2105	0.3039	...	0.0045	0.0014	0.0038
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
202	0.0187	0.0346	0.0168	0.0177	0.0393	0.1630	0.2028	0.1694	0.2328	0.2684	...	0.0116	0.0098	0.0195
203	0.0323	0.0101	0.0298	0.0564	0.0760	0.0958	0.0990	0.1018	0.1030	0.2154	...	0.0061	0.0093	0.0135
204	0.0522	0.0437	0.0180	0.0292	0.0351	0.1171	0.1257	0.1178	0.1258	0.2529	...	0.0160	0.0029	0.0051
205	0.0303	0.0353	0.0490	0.0608	0.0167	0.1354	0.1465	0.1123	0.1945	0.2354	...	0.0086	0.0046	0.0126
206	0.0260	0.0363	0.0136	0.0272	0.0214	0.0338	0.0655	0.1400	0.1843	0.2354	...	0.0146	0.0129	0.0047

207 rows × 61 columns

In [6]:
```
from sklearn.preprocessing import StandardScaler
```

In [7]:
```
scaler=StandardScaler()
```

In [8]:
```
scaler.fit(dataset.drop('R',axis=1))
```

Out[8]:
```
▾ StandardScaler
StandardScaler()
```

In [9]:
```
dataset.columns[:-1]
```

Out[9]:
```
Index(['0.0200', '0.0371', '0.0428', '0.0207', '0.0954', '0.0986', '0.1539',
       '0.1601', '0.3109', '0.2111', '0.1609', '0.1582', '0.2238', '0.0645',
       '0.0660', '0.2273', '0.3100', '0.2999', '0.5078', '0.4797', '0.5783',
       '0.5071', '0.4328', '0.5550', '0.6711', '0.6415', '0.7104', '0.8080',
       '0.6791', '0.3857', '0.1307', '0.2604', '0.5121', '0.7547', '0.8537',
       '0.8507', '0.6692', '0.6097', '0.4943', '0.2744', '0.0510', '0.2834',
       '0.2825', '0.4256', '0.2641', '0.1386', '0.1051', '0.1343', '0.0383',
       '0.0324', '0.0232', '0.0027', '0.0065', '0.0159', '0.0072', '0.0167',
       '0.0180', '0.0084', '0.0090', '0.0032'],
      dtype='object')
```

In [10]:
```
scaled_features=scaler.transform(dataset.drop('R',axis=1))
```

In [11]:
```
scaled_features
```

Out[11]:
```
array([[ 0.70018948,  0.42042142,  1.0529498 , ..., -0.4709383 ,
        -0.44268846, -0.42246083],
       [-0.13089402,  0.59942737,  1.71912994, ...,  1.30656071,
         0.25299833,  0.25405324],
       [-0.83579208, -0.64754631,  0.48045125, ..., -0.54822087,
        -0.63683361,  1.03005467],
       ...,
       [ 1.00042384,  0.15949749, -0.67235266, ...,  0.90469137,
        -0.0382194 , -0.68112798],
       [ 0.0475061 , -0.09535845,  0.13434985, ..., -0.00724291,
        -0.70154866, -0.34287094],
```

```
       [-0.13959647, -0.06501846, -0.78685237, ..., -0.67187297,
        -0.2970796 ,  0.99025973]])
```

In [12]:
```
new_dataset=pd.DataFrame(scaled_features,columns=dataset.columns[:-1])
```

In [13]:
```
new_dataset
```

Out[13]:

	0.0200	0.0371	0.0428	0.0207	0.0954	0.0986	0.1539	0.1601	0.3109	0.2111	..
0	0.700189	0.420421	1.052950	0.319501	0.777810	2.600518	1.522475	2.506911	1.324632	0.587392	.
1	-0.130894	0.599427	1.719130	1.167351	0.401466	2.087862	1.967839	2.847551	3.240336	3.058831	.
2	-0.835792	-0.647546	0.480451	-0.722021	-0.983262	-1.147115	-0.190961	-0.083126	-0.996071	-0.608898	.
3	2.044717	0.854283	0.110929	-0.315311	-0.289998	-0.671681	-0.011196	1.315846	1.516965	1.768058	.
4	-0.026465	0.208041	-0.419933	-0.788730	-0.660939	-0.094732	-0.024152	0.571138	0.280782	0.711633	.
...	...	...	...	...	...	...	...	...	...	...	.
202	-0.457236	-0.116596	-0.703580	-0.782275	-0.644733	0.988106	1.315179	0.407865	0.469725	0.447527	.
203	0.134531	-0.859926	-0.365285	0.050513	0.016119	-0.148874	-0.365867	-0.386178	-0.630046	0.053227	.
204	1.000424	0.159497	-0.672353	-0.534806	-0.720362	0.211508	0.066541	-0.198238	-0.436866	0.332213	.
205	0.047506	-0.095358	0.134350	0.145196	-1.051689	0.521132	0.403398	-0.262842	0.145217	0.202020	.
206	-0.139596	-0.065018	-0.786852	-0.577844	-0.967056	-1.197873	-0.908401	0.062527	0.058794	0.202020	.

207 rows × 60 columns

In [14]:
```
from sklearn.model_selection import train_test_split
X=new_dataset
y=dataset['R']
```

In [15]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=42)
```

In [16]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [17]:
```
knn=KNeighborsClassifier(n_neighbors=1)
```

In [18]:
```
knn.fit(X_train,y_train)
```

Out[18]:
```
KNeighborsClassifier(n_neighbors=1)
```

In [19]:
```
pred=knn.predict(X_test)
```

In [20]:
```
from sklearn.metrics import classification_report,confusion_matrix
```

In [21]:
```
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

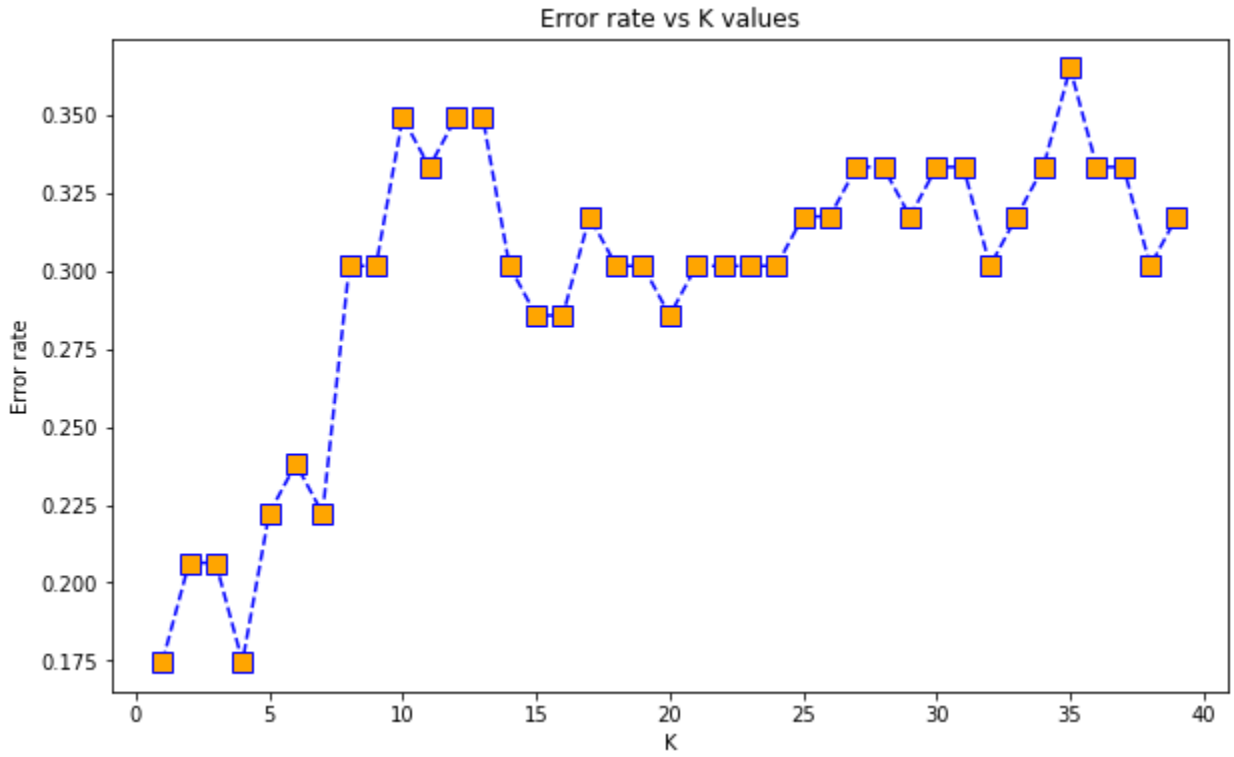
```
[[30  6]
 [ 5 22]]
              precision    recall  f1-score   support

           M       0.86      0.83      0.85        36
           R       0.79      0.81      0.80        27
```

```
    accuracy                           0.83        63
   macro avg       0.82      0.82      0.82        63
weighted avg       0.83      0.83      0.83        63
```

In [22]:
```
#Analyzing better k value through iterations
error_rate=[]

for i in range(1,40):
  knn=KNeighborsClassifier(n_neighbors=i)
  knn.fit(X_train,y_train)
  pred_i=knn.predict(X_test)
  error_rate.append(np.mean(pred_i !=y_test))
```

In [23]:
```
plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue',linestyle='dashed',
         marker='s',markerfacecolor='orange',markersize=10)
plt.title('Error rate vs K values')
plt.xlabel('K')
plt.ylabel('Error rate')
```

Out[23]:
```
Text(0, 0.5, 'Error rate')
```



In [24]:
```
knn=KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train,y_train)
pred=knn.predict(X_test)
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
[[31  5]
 [ 8 19]]
              precision    recall  f1-score   support

           M       0.79      0.86      0.83        36
           R       0.79      0.70      0.75        27

    accuracy                           0.79        63
   macro avg       0.79      0.78      0.79        63
```

```
weighted avg       0.79      0.79      0.79        63
```

RESULT: The program is executed successfully and output is obtained

PROGRAM NO: 6

AIM: Program to implement Naïve Bayes Algorithm using any standard dataset available in the public domain and find the accuracy of the algorithm

Dataset: Iris.csv

ALGORITHM:

1. Import Libraries
2. Read the data
3. Import StandardScaler and fit the numerical data in it
4. Split the dataset into training and testing sets
5. Import GaussianNB from sklearn.naive_bayes
6. Fit the model to the GaussianNB and predict the values
7. Compare the predicted and test values and find the accuracy score
8. Use confusion matrix and accuracy score to find the accuracy

PROGRAM CODE:

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```
dataset=pd.read_csv('Iris.csv')
dataset
```

Out[3]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

In [4]:
```
from sklearn.preprocessing import StandardScaler
```

In [5]:
```
scaler=StandardScaler()
```

In [6]:
```
scaler.fit(dataset.drop('Species',axis=1))
```

Out[6]:
```
▾ StandardScaler
StandardScaler()
```

In [7]:
```
scaled_features=scaler.transform(dataset.drop('Species',axis=1))
```

In [8]:
```
new_dataset=pd.DataFrame(scaled_features,columns=dataset.columns[:-1])
```

In [9]:
```
new_dataset
```

Out[9]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	-1.720542	-0.900681	1.032057	-1.341272	-1.312977
1	-1.697448	-1.143017	-0.124958	-1.341272	-1.312977
2	-1.674353	-1.385353	0.337848	-1.398138	-1.312977
3	-1.651258	-1.506521	0.106445	-1.284407	-1.312977
4	-1.628164	-1.021849	1.263460	-1.341272	-1.312977
...	...	...	...	...	...
145	1.628164	1.038005	-0.124958	0.819624	1.447956
146	1.651258	0.553333	-1.281972	0.705893	0.922064
147	1.674353	0.795669	-0.124958	0.819624	1.053537
148	1.697448	0.432165	0.800654	0.933356	1.447956
149	1.720542	0.068662	-0.124958	0.762759	0.790591

150 rows × 5 columns

In [10]:
```
from sklearn.model_selection import train_test_split
X=new_dataset
y=dataset['Species']
```

In [11]:
```
 X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=42)
```

In [12]:
```
from sklearn.naive_bayes import GaussianNB, BernoulliNB, CategoricalNB
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
scores = []
#GuassianNB
#Guassian Naive Bayes is used in cases when all our features are cases are continuous
#its features can have different values in data set as width and length can vary

classifier = GaussianNB()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
scores.append(accuracy_score(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[12]:
```
array([[19,  0,  0],
       [ 0, 13,  0],
       [ 0,  0, 13]], dtype=int64)
```

In [13]:
```
print(scores)
```

```
[1.0]
```

In [14]:
```
classifier1 = BernoulliNB()
classifier1.fit(X_train, y_train)
y_pred = classifier1.predict(X_test)
scores.append(accuracy_score(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[14]:
```
array([[19,  0,  0],
       [ 0,  9,  4],
       [ 0,  1, 12]], dtype=int64)
```

In [15]:
```
print(scores)
```

```
[1.0, 0.8888888888888888]
```

RESULT: The program is executed successfully and output is obtained

PROGRAM NO: 7

AIM: Program to implement linear and multiple regression techniques using any standard dataset available in the public domain and evaluate its performance

Dataset: Salary_Data.csv

ALGORITHM:

1. Import Libraries
2. Read the data
3. Split the dataset into training and testing sets
4. Import LinearRegression from sklearn.linear_model
5. Fit the model to LinearRegression
6. Predict the Test set results
7. Compare the predicted and actual values
8. Visualize Test set results

PROGRAM CODE:

In [13]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import metrics
from sklearn.metrics import mean_absolute_error
```

In [2]:
```
# Importing the dataset
dataset = pd.read_csv('Salary_Data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values
dataset
```

Out[2]:

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0
5	2.9	56642.0
6	3.0	60150.0
7	3.2	54445.0
8	3.2	64445.0
9	3.7	57189.0
10	3.9	63218.0
11	4.0	55794.0
12	4.0	56957.0
13	4.1	57081.0

14	4.5	61111.0
15	4.9	67938.0
16	5.1	66029.0
17	5.3	83088.0
18	5.9	81363.0
19	6.0	93940.0
20	6.8	91738.0
21	7.1	98273.0
22	7.9	101302.0
23	8.2	113812.0
24	8.7	109431.0
25	9.0	105582.0
26	9.5	116969.0
27	9.6	112635.0
28	10.3	122391.0
29	10.5	121872.0

In [3]:
```
X
```

Out[3]:
```
array([[ 1.1],
       [ 1.3],
       [ 1.5],
       [ 2. ],
       [ 2.2],
       [ 2.9],
       [ 3. ],
       [ 3.2],
       [ 3.2],
       [ 3.7],
       [ 3.9],
       [ 4. ],
       [ 4. ],
       [ 4.1],
       [ 4.5],
       [ 4.9],
       [ 5.1],
       [ 5.3],
       [ 5.9],
       [ 6. ],
       [ 6.8],
       [ 7.1],
       [ 7.9],
       [ 8.2],
       [ 8.7],
       [ 9. ],
       [ 9.5],
       [ 9.6],
       [10.3],
       [10.5]])
```

In [4]:
```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3, random_state
```

In [5]:
```
# Feature Scaling
""" do feature scaling if necessary"""
```

Out[5]:
```
' do feature scaling if necessary'
```

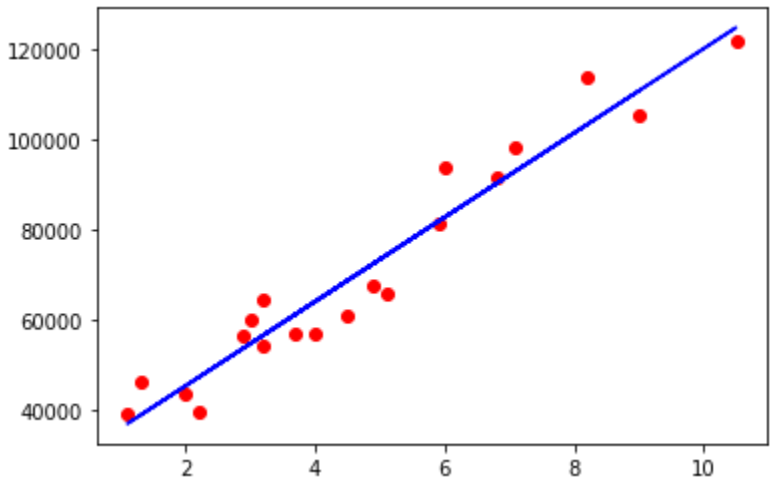
In [6]:
```
# Fitting Simple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Out[6]:
```
▾ LinearRegression
LinearRegression()
```

In [7]:
```
# Predicting the Test set results
y_pred = regressor.predict(X_test)
```

In [8]:
```
# Visualising the Training set results
plt.scatter(X_train, y_train, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color = 'blue')
```

Out[8]:
```
[<matplotlib.lines.Line2D at 0x20b8186d870>]
```



In [9]:
```
# Visualising the Test set results
plt.scatter(X_test, y_test, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color = 'blue')
plt.title('Salary vs Experience (Test set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```

In [14]:
```
#Step 6: Predict and evaluate
Y_pred = regressor.predict(X_test)
Mae_error = mean_absolute_error(y_test, Y_pred)
print("MAE error is ", Mae_error)
```

```
MAE error is  3426.4269374307078
```

In [15]:
```
print(regressor.intercept_)
```

```
26816.192244031183
```

In [16]:
```
print('MAE:', metrics.mean_absolute_error(y_test, Y_pred))
print('MSE:', metrics.mean_squared_error(y_test, Y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, Y_pred)))
```

```
MAE: 3426.4269374307078
MSE: 21026037.329511296
RMSE: 4585.4157204675885
```

Multiple Regression

In [22]:
```
df=pd.read_csv('USA_Housing.csv')
```

In [23]:
```
df.head()
```

Out[23]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address
0	79545.458574	5.682861	7.009188	4.09	23086.800503	1.059034e+06	208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1	79248.642455	6.002900	6.730821	3.09	40173.072174	1.505891e+06	188 Johnson Views Suite 079\nLake Kathleen, CA...
2	61287.067179	5.865890	8.512727	5.13	36882.159400	1.058988e+06	9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3	63345.240046	7.188236	5.586729	3.26	34310.242831	1.260617e+06	USS Barnett\nFPO AP 44820
4	59982.197226	5.040555	7.839388	4.23	26354.109472	6.309435e+05	USNS Raymond\nFPO AE 09386

In [24]:
```
df.drop(columns = "Address", inplace=True)
```

In [39]:
```
x = df.drop('Price',axis=1)
```

In [41]:
```
y = df['Price']
```

In [43]:
```
x_train,x_test,y_train,y_test=train_test_split(x, y, test_size=0.3, random_state=45)
```

In [44]:
```
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[44]:
```
▾ LinearRegression
LinearRegression()
```

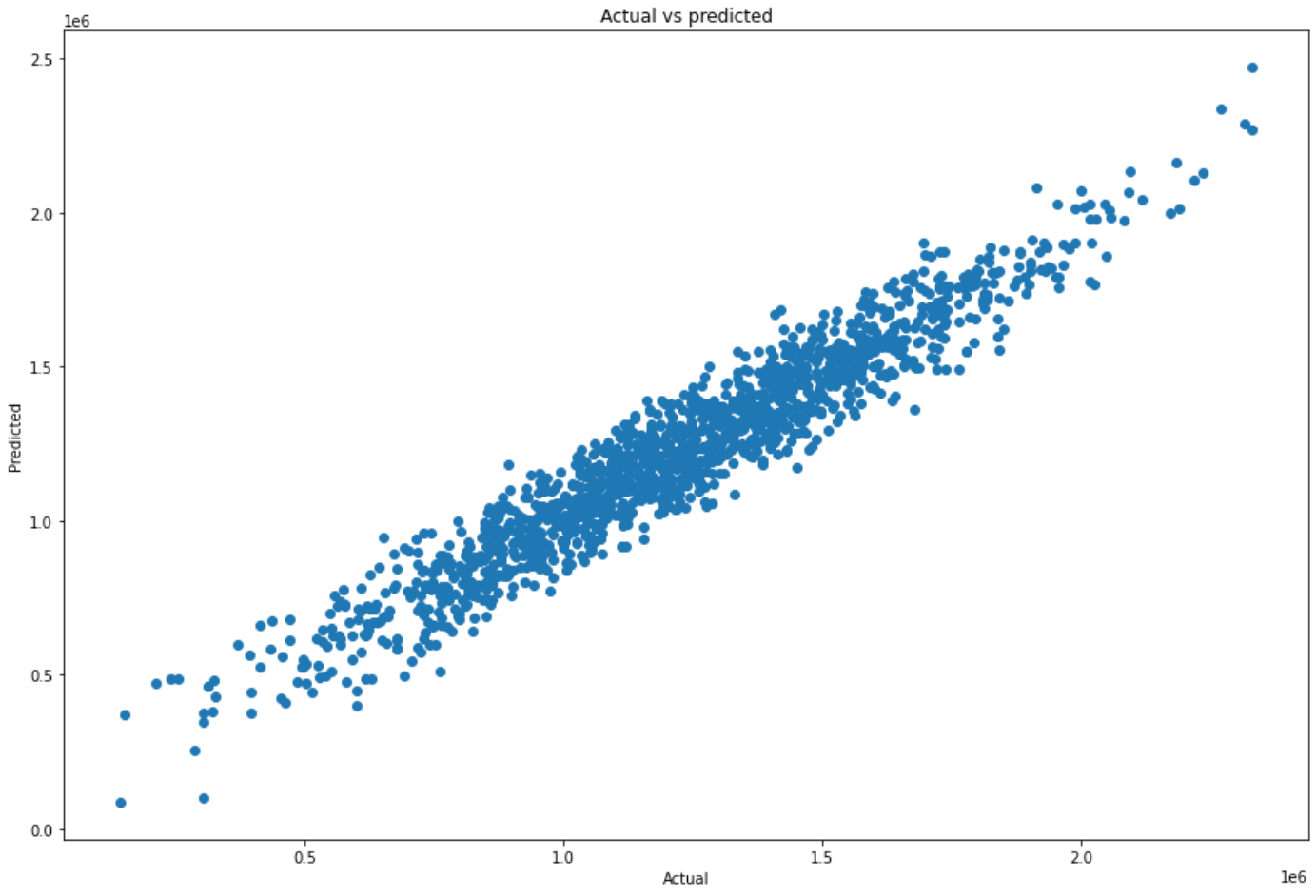
In [45]:
```
y_pred = lr.predict(x_test)
```

In [46]:
```
from sklearn.metrics import r2_score
r2_score(y_test,y_pred)
```

Out[46]:
```
0.9215354394118033
```

In [47]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(15,10))
plt.scatter(y_test,y_pred)
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Actual vs predicted')
```

Out[47]:
```
Text(0.5, 1.0, 'Actual vs predicted')
```



In [48]:
```
pred_df=pd.DataFrame({'Actual Value':y_test,'Predicted value':y_pred})
pred_df
```

Out[48]:

	Actual Value	Predicted value
4988	1.275143e+06	1.357724e+06
1827	9.268809e+05	9.392281e+05
2478	1.440909e+06	1.261011e+06
3116	1.185735e+06	1.073385e+06
3033	9.368647e+05	8.622679e+05
...	...	...
2282	1.170705e+06	1.181054e+06
424	1.275319e+06	1.186336e+06
2022	1.729062e+06	1.666521e+06
4797	1.335831e+06	1.483670e+06
3681	1.275497e+06	1.045619e+06

1500 rows × 2 columns

RESULT: The program is executed successfully and output is obtained

COURSE OUTCOME 3

PROGRAM NO: 8

AIM : Program to implement text classification using Support vector machine.

Dataset: SMSSpamCollection

ALGORITHM:

1. **Import Libraries**
2. **Read the data**
3. **Write a Function to remove punctuation & stopwords and return list of clean text words**
4. **Tokenise the data using the above function**
5. **Convert token into vectors using Count Vectorizer**
6. **Use .transform on our Bag-of-Words (bow) transformed object and transform the entire DataFrame of messages**
7. **Calculate term frequency-inverse document frequency**
8. **Split the dataset into training and testing sets**
9. **Fit the model to MultinomialNB**
10. **Create a Data Pipeline**
11. **Use SVM classifier algorithm and fit the training dataset on the classifier**

PROGRAM CODE:

In [ ]:

```
import nltk
import pandas as pd
```

In [ ]:

```
nltk.download_shell()
```

```
NLTK Downloader
---------------------------------------------------------------------------
    d) Download   l) List    u) Update   c) Config   h) Help   q) Quit
---------------------------------------------------------------------------
Downloader> d

Download which package (l=list; x=cancel)?
  Identifier> l
Packages:
  [ ] abc................. Australian Broadcasting Commission 2006
  [ ] alpino.............. Alpino Dutch Treebank
  [ ] averaged_perceptron_tagger Averaged Perceptron Tagger
  [ ] averaged_perceptron_tagger_ru Averaged Perceptron Tagger (Russian)
  [ ] basque_grammars..... Grammars for Basque
  [ ] biocreative_ppi..... BioCreAtIvE (Critical Assessment of Information
                           Extraction Systems in Biology)
```

```
  [ ] cess_cat.......... CESS-CAT Treebank
  [ ] cess_esp.......... CESS-ESP Treebank
  [ ] chat80............ Chat-80 Data Files
  [ ] city_database..... City Database
  [ ] cmudict........... The Carnegie Mellon Pronouncing Dictionary (0.6)
  [ ] comparative_sentences Comparative Sentence Dataset
  [ ] comtrans.......... ComTrans Corpus Sample
  [ ] conll2000......... CONLL 2000 Chunking Corpus
  [ ] conll2002......... CONLL 2002 Named Entity Recognition Corpus
Hit Enter to continue:
  [ ] conll2007......... Dependency Treebanks from CoNLL 2007 (Catalan
                         and Basque Subset)
  [ ] crubadan.......... Crubadan Corpus
  [ ] dependency_treebank. Dependency Parsed Treebank
  [ ] dolch............. Dolch Word List
  [ ] europarl_raw...... Sample European Parliament Proceedings Parallel
                         Corpus
  [ ] floresta.......... Portuguese Treebank
  [ ] framenet_v15...... FrameNet 1.5
  [ ] framenet_v17...... FrameNet 1.7
  [ ] gazetteers........ Gazeteer Lists
  [ ] genesis........... Genesis Corpus
  [ ] gutenberg......... Project Gutenberg Selections
  [ ] ieer.............. NIST IE-ER DATA SAMPLE
  [ ] inaugural......... C-Span Inaugural Address Corpus
  [ ] indian............ Indian Language POS-Tagged Corpus
  [ ] jeita............. JEITA Public Morphologically Tagged Corpus (in
                         ChaSen format)
  [ ] kimmo............. PC-KIMMO Data Files
  [ ] knbc.............. KNB Corpus (Annotated blog corpus)
  [ ] large_grammars..... Large context-free and feature-based grammars
                         for parser comparison
Hit Enter to continue:
  [ ] lin_thesaurus..... Lin's Dependency Thesaurus
  [ ] mac_morpho........ MAC-MORPHO: Brazilian Portuguese news text with
                         part-of-speech tags
  [ ] machado........... Machado de Assis -- Obra Completa
  [ ] masc_tagged....... MASC Tagged Corpus
  [ ] maxent_ne_chunker... ACE Named Entity Chunker (Maximum entropy)
  [ ] maxent_treebank_pos_tagger Treebank Part of Speech Tagger (Maximum entropy)
  [ ] moses_sample...... Moses Sample Models
  [ ] movie_reviews..... Sentiment Polarity Dataset Version 2.0
  [ ] mte_teip5......... MULTEXT-East 1984 annotated corpus 4.0
  [ ] mwa_ppdb.......... The monolingual word aligner (Sultan et al.
                         2015) subset of the Paraphrase Database.
  [ ] names............. Names Corpus, Version 1.3 (1994-03-29)
  [ ] nombank.1.0....... NomBank Corpus 1.0
  [ ] nonbreaking_prefixes Non-Breaking Prefixes (Moses Decoder)
  [ ] nps_chat.......... NPS Chat
  [ ] omw-1.4........... Open Multilingual Wordnet
  [ ] omw............... Open Multilingual Wordnet
  [ ] opinion_lexicon..... Opinion Lexicon
  [ ] panlex_swadesh..... PanLex Swadesh Corpora
  [ ] paradigms......... Paradigm Corpus
Hit Enter to continue:
  [ ] pe08.............. Cross-Framework and Cross-Domain Parser
                         Evaluation Shared Task
  [ ] perluniprops...... perluniprops: Index of Unicode Version 7.0.0
                         character properties in Perl
  [ ] pil............... The Patient Information Leaflet (PIL) Corpus
  [ ] pl196x............ Polish language of the XX century sixties
  [ ] porter_test....... Porter Stemmer Test Files
  [ ] ppattach.......... Prepositional Phrase Attachment Corpus
  [ ] problem_reports..... Problem Report Corpus
  [ ] product_reviews_1... Product Reviews (5 Products)
  [ ] product_reviews_2... Product Reviews (9 Products)
  [ ] propbank.......... Proposition Bank Corpus 1.0
  [ ] pros_cons......... Pros and Cons
```

```
                         version
  [ ] rslp............... RSLP Stemmer (Removedor de Sufixos da Lingua
                         Portuguesa)
  [ ] rte................ PASCAL RTE Challenges 1, 2, and 3
Hit Enter to continue:
  [ ] sample_grammars..... Sample Grammars
  [ ] semcor.............. SemCor 3.0
  [ ] senseval............ SENSEVAL 2 Corpus: Sense Tagged Text
  [ ] sentence_polarity... Sentence Polarity Dataset v1.0
  [ ] sentiwordnet........ SentiWordNet
  [ ] shakespeare......... Shakespeare XML Corpus Sample
  [ ] sinica_treebank..... Sinica Treebank Corpus Sample
  [ ] smultron............ SMULTRON Corpus Sample
  [ ] snowball_data....... Snowball Data
  [ ] spanish_grammars.... Grammars for Spanish
  [ ] state_union......... C-Span State of the Union Address Corpus
  [ ] stopwords........... Stopwords Corpus
  [ ] subjectivity........ Subjectivity Dataset v1.0
  [ ] swadesh............. Swadesh Wordlists
  [ ] switchboard......... Switchboard Corpus Sample
  [ ] tagsets............. Help on Tagsets
  [ ] timit............... TIMIT Corpus Sample
  [ ] toolbox............. Toolbox Sample Files
  [ ] treebank............ Penn Treebank Sample
  [ ] twitter_samples..... Twitter Samples
  [ ] udhr2............... Universal Declaration of Human Rights Corpus
                         (Unicode Version)
Hit Enter to continue:
  [ ] udhr................ Universal Declaration of Human Rights Corpus
  [ ] unicode_samples..... Unicode Samples
  [ ] universal_tagset.... Mappings to the Universal Part-of-Speech Tagset
  [ ] universal_treebanks_v20 Universal Treebanks Version 2.0
  [ ] vader_lexicon....... VADER Sentiment Lexicon
  [ ] verbnet3............ VerbNet Lexicon, Version 3.3
  [ ] verbnet............. VerbNet Lexicon, Version 2.1
  [ ] webtext............. Web Text Corpus
  [ ] wmt15_eval.......... Evaluation data from WMT15
  [ ] word2vec_sample..... Word2Vec Sample
  [ ] wordnet2021......... Open English Wordnet 2021
  [ ] wordnet31........... Wordnet 3.1
  [ ] wordnet............. WordNet
  [ ] wordnet_ic.......... WordNet-InfoContent
  [ ] words............... Word Lists
  [ ] ycoe................ York-Toronto-Helsinki Parsed Corpus of Old
                         English Prose

Collections:
  [ ] all-corpora......... All the corpora
  [ ] all-nltk............ All packages available on nltk_data gh-pages
                         branch
Hit Enter to continue:
  [ ] all................. All packages
  [ ] book................ Everything used in the NLTK Book
  [ ] popular............. Popular packages
  [ ] tests............... Packages for running tests
  [ ] third-party......... Third-party data packages

([*] marks installed packages)

Download which package (l=list; x=cancel)?
  Identifier> x

---------------------------------------------------------------------------
    d) Download   l) List    u) Update   c) Config   h) Help   q) Quit
---------------------------------------------------------------------------
Downloader> q
```

```
In [ ]:
```

In [ ]:

```
print(len(messages))
```

```
5574
```

In [ ]:

```
messages[0]
```

Out[ ]:

```
'ham\tGo until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...'
```

In [ ]:

```
for mess_no,message in enumerate(messages[:10]):
  print(mess_no,message)
  print('\n')
```

```
0 ham Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...


1 ham Ok lar... Joking wif u oni...


2 spam Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 871 21 to receive entry question(std txt rate)T&C's apply 08452810075over18's


3 ham U dun say so early hor... U c already then say...


4 ham Nah I don't think he goes to usf, he lives around here though


5 spam FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like some f un you up for it still? Tb ok! XxX std chgs to send, £1.50 to rcv


6 ham Even my brother is not like to speak with me. They treat me like aids patent.


7 ham As per your request 'Melle Melle (Oru Minnaminunginte Nurungu Vettam)' has been set as your callertune for all Callers. Press *9 to copy your friends Callertune


8 spam WINNER!! As a valued network customer you have been selected to receivea £900 priz e reward! To claim call 09061701461. Claim code KL341. Valid 12 hours only.


9 spam Had your mobile 11 months or more? U R entitled to Update to the latest colour mob iles with camera for Free! Call The Mobile Update Co FREE on 08002986030
```

In [ ]:

```
messages=pd.read_csv('/content/sample_data/SMSSpamCollection',sep='\t',names=['label','me ssage'])
```

In [ ]:

```
messages.head()
```

Out[ ]:

	label	message
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

In [ ]:

```
import string
```

In [ ]:

```
msg="sample message! Notice: It has punctuation."
```

In [ ]:

```
string.punctuation
```

Out[ ]:

```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

In [ ]:

```
#removing punctuations and stopwords
nopunc= [c for c in msg if c not in string.punctuation]
```

In [ ]:

```
nopunc
```

Out[ ]:

```
['s',
 'a',
 'm',
 'p',
 'l',
 'e',
 ' ',
 'm',
 'e',
 's',
 's',
 'a',
 'g',
 'e',
 ' ',
 'N',
 'o',
 't',
 'i',
 'c',
 'e',
 ' ',
 'I',
 't',
 ' ',
 'h',
 'a',
 's',
 ' ',
 'p',
 'u',
 'n',
 'c',
```

```
 'i',
 'o',
 'n']
```

In [ ]:

```
nopunc="".join(nopunc)
```

In [ ]:

```
nopunc
```

Out[ ]:

```
'sample message Notice It has punctuation'
```

In [ ]:

```
#for removing stopwords , we need to download stopword corpus
import nltk

from nltk.corpus import stopwords
```

In [ ]:

```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

Out[ ]:

```
True
```

In [ ]:

```
nopunc.split()
```

Out[ ]:

```
['sample', 'message', 'Notice', 'It', 'has', 'punctuation']
```

In [ ]:

```
clean_mess=[word for word in nopunc.split() if word.lower() not in stopwords.words("engl
ish")]
```

In [ ]:

```
clean_mess
```

Out[ ]:

```
['sample', 'message', 'Notice', 'punctuation']
```

In [ ]:

```
#Applying above function in given dataset
#1.Remove punctuation
#2.Remove stopwords
#3.Return list of clean text words
def text_process(mess):
  nopunc=[char for char in mess if char not in string.punctuation]
  nopunc="".join(nopunc)
  return [word for word in nopunc.split() if word.lower() not in stopwords.words("englis
h")]
```

In [ ]:

	label	message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

In [ ]:

```
#tokenise using above function for cleaned version
messages['message'].head(5).apply(text_process)
```

Out[ ]:

```
0    [Go, jurong, point, crazy, Available, bugis, n...
1                        [Ok, lar, Joking, wif, u, oni]
2    [Free, entry, 2, wkly, comp, win, FA, Cup, fin...
3        [U, dun, say, early, hor, U, c, already, say]
4    [Nah, dont, think, goes, usf, lives, around, t...
Name: message, dtype: object
```

In [ ]:

```
#convert tokens into vectors so that ML model can understand
from sklearn.feature_extraction.text import CountVectorizer
#Count vector is used to transform a given text into a vector on the basis of count of ea
ch word in the entire text
```

In [ ]:

```
bow_transformer=CountVectorizer(analyzer=text_process).fit(messages['message'])
```

In [ ]:

```
print(len(bow_transformer.vocabulary_))
```

```
11425
```

In [ ]:

```
msgs=messages['message'][6]
```

In [ ]:

```
print(msgs)
```

```
Even my brother is not like to speak with me. They treat me like aids patent.
```

In [ ]:

```
bow=bow_transformer.transform([msgs])
```

In [ ]:

```
print(bow)
```

```
  (0, 1802) 1
  (0, 4590) 1
  (0, 5193) 1
  (0, 7800) 2
  (0, 8761) 1
  (0, 9971) 1
  (0, 10629) 1
```

```
bow_transformer.get_feature_names()[7800]
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

Out[ ]:

```
'like'
```

In [ ]:

```
#apply this transformation for the whole message column in dataset
messages_bow=bow_transformer.transform(messages['message'])
```

In [ ]:

```
print('Shape of Sparse Matrix:',messages_bow.shape)
```

```
Shape of Sparse Matrix: (5572, 11425)
```

In [ ]:

```
#check non zero occurences
messages_bow.nnz
```

Out[ ]:

```
50548
```

TERM FREQUENCY-INVERSE DOCUMENT FREQUENCY

In [ ]:

```
#difference between TF and DF
#TF is frequency counter for a term t in document d
#DF is the count of occurences of term t in the document set N
from sklearn.feature_extraction.text import TfidfTransformer
```

In [ ]:

```
tfidf_transformer=TfidfTransformer().fit(messages_bow)
```

In [ ]:

```
tfidf1=tfidf_transformer.transform(bow)
```

In [ ]:

```
print(tfidf1)
```

```
  (0, 10629) 0.3352766696931058
  (0, 9971) 0.3268691780062757
  (0, 8761) 0.43700993321905807
  (0, 7800) 0.41453906826037096
  (0, 5193) 0.33843411088434017
  (0, 4590) 0.43700993321905807
  (0, 1802) 0.3352766696931058
```

In [ ]:

```
#converting the whole bag of words into tfidf
messages_tfidf=tfidf_transformer.transform(messages_bow)
```

In [ ]:

```
from sklearn.naive_bayes import MultinomialNB
```

```
spam_detect_model=MultinomialNB().fit(messages_tfidf,messages['label'])
```

In [ ]:

```
all_pred=spam_detect_model.predict(messages_tfidf)
```

In [ ]:

```
all_pred
```

Out[ ]:

```
array(['ham', 'ham', 'spam', ..., 'ham', 'ham', 'ham'], dtype='<U4')
```

In [ ]:

```
from sklearn.model_selection import train_test_split
```

In [ ]:

```
msg_train,msg_test,label_train,label_test=train_test_split(messages['message'],messages['
label'],test_size=.3)
```

In [ ]:

```
spam_detect_model=MultinomialNB().fit(messages_tfidf,messages['label'])
```

In [ ]:

```
predict=spam_detect_model.predict(messages_tfidf)
```

In [ ]:

```
from sklearn.metrics import classification_report
print(classification_report(messages['label'],predict))
```

```
              precision    recall  f1-score   support

         ham       0.98      1.00      0.99      4825
        spam       1.00      0.85      0.92       747

    accuracy                           0.98      5572
   macro avg       0.99      0.92      0.95      5572
weighted avg       0.98      0.98      0.98      5572
```

In [ ]:

```
#TRAIN TEST SPLIT
from sklearn.model_selection import train_test_split
msg_train,msg_test,label_train,label_test=train_test_split(messages['message'],messages['
label'],test_size=0.2)
```

creating a data pipeline

In [ ]:

```
from sklearn.pipeline import Pipeline

pipeline=Pipeline([
                   ('bow',CountVectorizer(analyzer=text_process)),#strings to token inte
ger counts
                   ('tfidf',TfidfTransformer()),#integer counts to weighted TF_IDF score
s
                   ('classifier',MultinomialNB()),#train on TF_IDF vectors with Naive Ba
yes Classifier
])
```

```
#now we pass message directly and the pipeline will do our pre processing
pipeline.fit(msg_train,label_train)
```

Out[ ]:

```
Pipeline(steps=[('bow',
                 CountVectorizer(analyzer=<function text_process at 0x7f7a42788f80>)),
                ('tfidf', TfidfTransformer()),
                ('classifier', MultinomialNB())])
```

In [ ]:

```
predictions=pipeline.predict(msg_test)
```

In [ ]:

```
print(classification_report(predictions,label_test))
```

```
              precision    recall  f1-score   support

         ham       1.00      0.96      0.98       993
        spam       0.75      1.00      0.86       122

    accuracy                           0.96      1115
   macro avg       0.87      0.98      0.92      1115
weighted avg       0.97      0.96      0.97      1115
```

SVM classifier

In [ ]:

```
from sklearn import model_selection,naive_bayes,svm
from sklearn.metrics import accuracy_score
```

In [ ]:

```
#classifier algo-SVM
#fit the training dataset on the classifier
pipeline1=Pipeline([
                    ('bow',CountVectorizer(analyzer=text_process)),#strings to token int
eger counts
                    ('tfidf',TfidfTransformer()),#integer counts to weighted TF_IDF scor
es
                    ('classifier',svm.SVC(C=0.1,kernel='linear',degree=3,gamma='auto'))#
train on TF_IDF vectors with Naive Bayes Classifier
])
```

In [ ]:

```
pipeline1.fit(msg_train,label_train)
```

Out[ ]:

```
Pipeline(steps=[('bow',
                 CountVectorizer(analyzer=<function text_process at 0x7f7a42788f80>)),
                ('tfidf', TfidfTransformer()),
                ('classifier', SVC(C=0.1, gamma='auto', kernel='linear'))])
```

In [ ]:

```
predictions1=pipeline1.predict(msg_test)
```

In [ ]:

```
print(classification_report(predictions1,label_test))
```

```
              precision    recall  f1-score   support
```

```
    accuracy                           0.90      1115
   macro avg       0.66      0.95      0.72      1115
weighted avg       0.97      0.90      0.92      1115
```

RESULT: The program is executed successfully and output is obtained

PROGRAM NO: 9

AIM : Program to implement decision trees using any standard dataset available in the public domain and find the accuracy of the algorithm.

Dataset: iris dataset

ALGORITHM:

1. **Import Libraries**
2. **Read the data**
3. **Split the dataset into training and testing sets**
4. **Fit the model to DecisionTreeClassifier**
5. **Predict the testing values and find the accuracy score**
6. **Visualize the decision tree using export_graphviz**

PROGRAM CODE:

In [ ]:

```
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris

data=load_iris()
```

In [ ]:

```
data.data.shape
```

Out[ ]:

```
(150, 4)
```

In [ ]:

```
print('Classes to Predict: ',data.target_names)
print('Features: ',data.feature_names)
```

```
Classes to Predict:  ['setosa' 'versicolor' 'virginica']
Features:  ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (c
m)']
```

In [ ]:

```
x=data.data
y=data.target
display(x.shape,y.shape)
```

```
(150, 4)
```

```
(150,)
```

In [ ]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=50,test_size=0.25)
```

In [ ]:

```
classifier=DecisionTreeClassifier() #default criterion - gini
classifier.fit(x_train,y_train)
```

Out[ ]:

```
DecisionTreeClassifier()
```

In [ ]:

```
y_pred=classifier.predict(x_test)
```

In [ ]:

```
from sklearn.metrics import accuracy_score
print('Accuracy on train data using Gini: ',accuracy_score(y_train,classifier.predict(x_t
rain)))
print('Accuracy on test data using Gini: ',accuracy_score(y_test,y_pred))
```

```
Accuracy on train data using Gini:  1.0
Accuracy on test data using Gini:  0.9473684210526315
```

In [ ]:

```
classifier_entropy=DecisionTreeClassifier(criterion='entropy')
classifier_entropy.fit(x_train,y_train)
y_pred_entropy=classifier_entropy.predict(x_test)
from sklearn.metrics import accuracy_score
print('Accuracy on train data using Entropy: ',accuracy_score(y_train,classifier_entropy.
predict(x_train)))
print('Accuracy on test data using Entropy: ',accuracy_score(y_test,y_pred_entropy))
```

```
Accuracy on train data using Entropy:  1.0
Accuracy on test data using Entropy:  0.9473684210526315
```

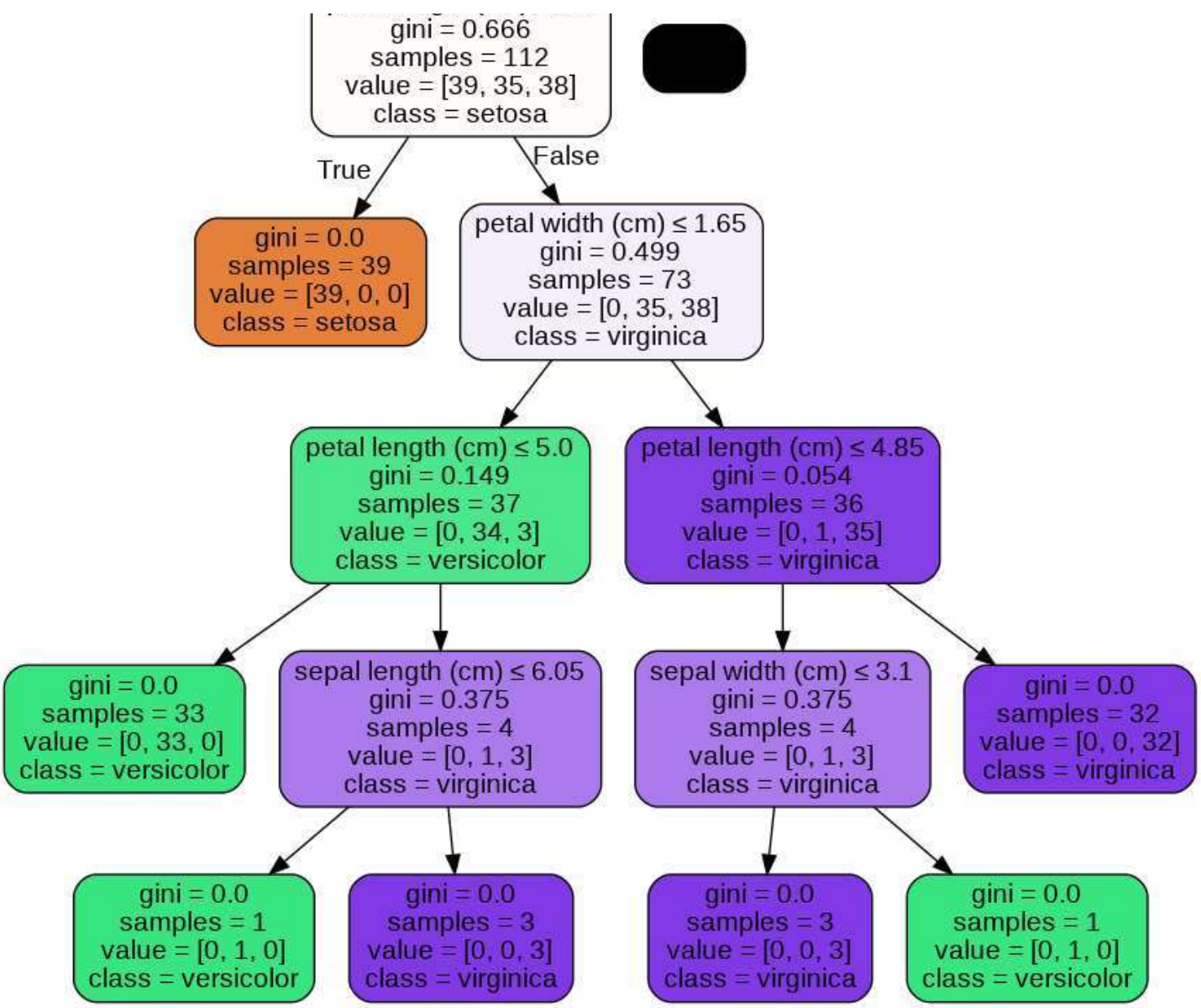
In [ ]:

```
classifier_entropy1=DecisionTreeClassifier(criterion='entropy',min_samples_split=50) #min
imum no. of samples required to split an internal node
classifier_entropy1.fit(x_train,y_train)
y_pred_entropy1=classifier_entropy1.predict(x_test)
from sklearn.metrics import accuracy_score
print('Accuracy on train data using Entropy with min samples 50: ',accuracy_score(y_train
,classifier_entropy1.predict(x_train)))
print('Accuracy on test data using Entropy with min samples 50: ',accuracy_score(y_test,y
_pred_entropy1))
```

```
Accuracy on train data using Entropy with min samples 50:  0.9642857142857143
Accuracy on test data using Entropy with min samples 50:  0.9473684210526315
```

In [ ]:

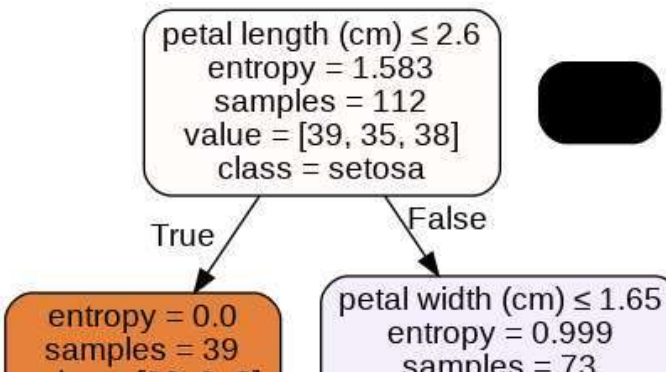
```
from sklearn.tree import export_graphviz #for visualization
from six import StringIO  #when stringIO object is created, it is initialised by passing
string to constructor.If no string is passed stringIO will start empty

from IPython.display import Image #IPython is an interactive shell built with python
import pydotplus #python interface to Graphviz's Dot Language

dot_data=StringIO()

export_graphviz(classifier, out_file=dot_data, filled=True, rounded=True, special_charact
ers=True, feature_names=data.feature_names,class_names=data.target_names)
graph=pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

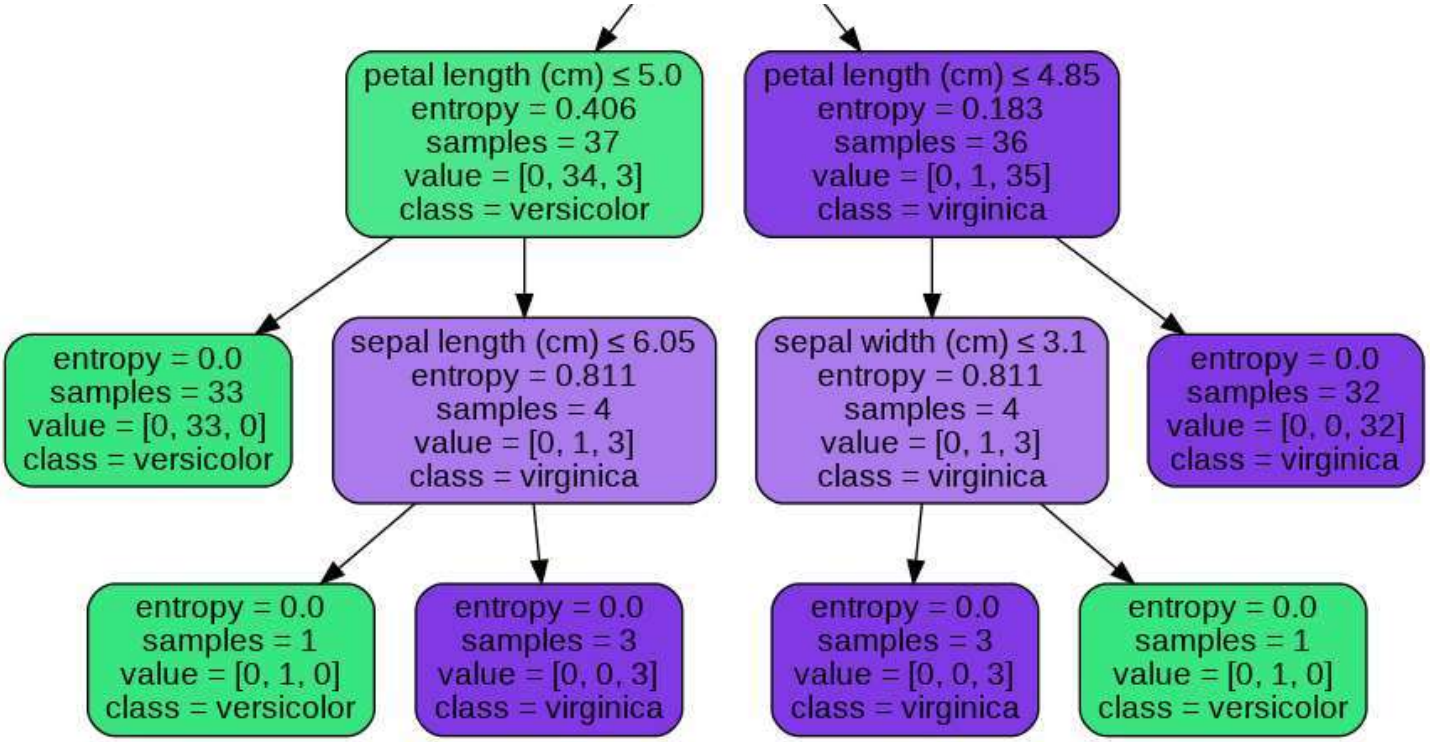


In [ ]:

```
from sklearn.tree import export_graphviz
from six import StringIO

from IPython.display import Image
import pydotplus

dot_data=StringIO()

export_graphviz(classifier_entropy, out_file=dot_data, filled=True, rounded=True, special
_characters=True, feature_names=data.feature_names,class_names=data.target_names)
graph1=pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph1.create_png())
```

Out[ ]:

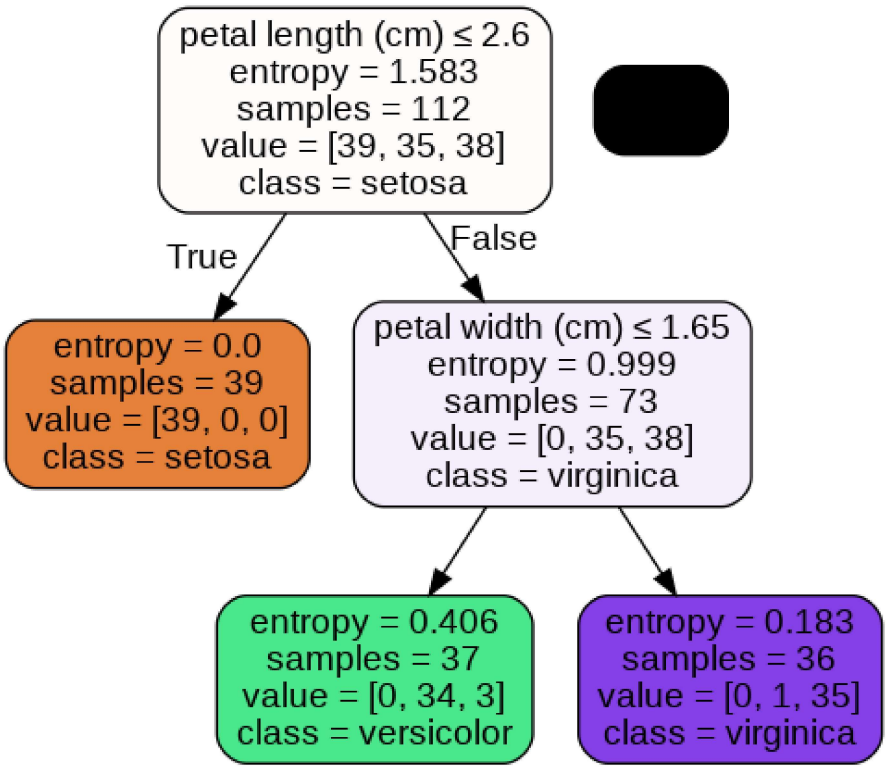




In [ ]:

```
from sklearn.tree import export_graphviz
from six import StringIO

from IPython.display import Image
import pydotplus

dot_data=StringIO()

export_graphviz(classifier_entropy1, out_file=dot_data, filled=True, rounded=True, specia
l_characters=True, feature_names=data.feature_names,class_names=data.target_names)
graph2=pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph2.create_png())
```

Out[ ]:



PROGRAM NO: 10

AIM : Program to implement k-means clustering technique using any standard dataset available in the public domain

---

Dataset : College_Data.csv

ALGORITHM:

1. **Import Libraries**
2. **Read the data**
3. **Visualize the data by creating a scatter plot**
4. **Create Clusters using KMean()**
5. **Create a confusion matrix and classification report**

PROGRAM CODE:

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [ ]:

```
df=pd.read_csv('/content/sample_data/College_Data',index_col=0)
```

In [ ]:

```
df.head()
```

Out[ ]:

	Private	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstate	Room.Board	Books	P
Abilene Christian University	Yes	1660	1232	721	23	52	2885	537	7440	3300	450	
Adelphi University	Yes	2186	1924	512	16	29	2683	1227	12280	6450	750	
Adrian College	Yes	1428	1097	336	22	50	1036	99	11250	3750	400	
Agnes Scott College	Yes	417	349	137	60	89	510	63	12960	5450	450	
Alaska Pacific University	Yes	193	146	55	16	44	249	869	7560	4120	800	

```
<class 'pandas.core.frame.DataFrame'>
Index: 777 entries, Abilene Christian University to York College of Pennsylvania
Data columns (total 18 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Private      777 non-null    object 
 1   Apps         777 non-null    int64  
 2   Accept       777 non-null    int64  
 3   Enroll       777 non-null    int64  
 4   Top10perc    777 non-null    int64  
 5   Top25perc    777 non-null    int64  
 6   F.Undergrad  777 non-null    int64  
 7   P.Undergrad  777 non-null    int64  
 8   Outstate     777 non-null    int64  
 9   Room.Board   777 non-null    int64  
 10  Books        777 non-null    int64  
 11  Personal     777 non-null    int64  
 12  PhD          777 non-null    int64  
 13  Terminal     777 non-null    int64  
 14  S.F.Ratio    777 non-null    float64
 15  perc.alumni  777 non-null    int64  
 16  Expend       777 non-null    int64  
 17  Grad.Rate    777 non-null    int64  
dtypes: float64(1), int64(16), object(1)
memory usage: 131.5+ KB
```

In [ ]:

```
df.describe()
```

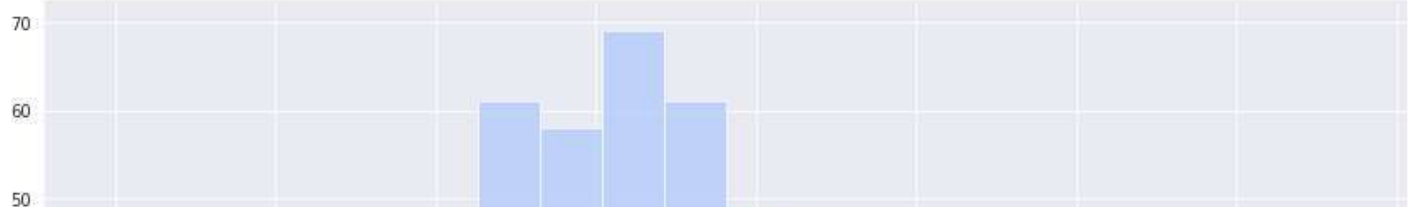
Out[ ]:

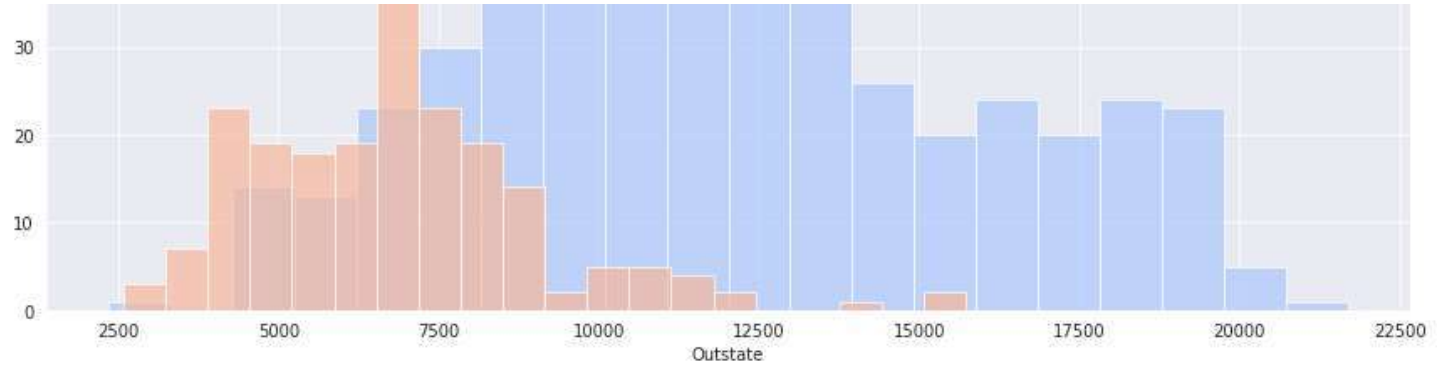
	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstate	Room.Bo
count	777.000000	777.000000	777.000000	777.000000	777.000000	777.000000	777.000000	777.000000	777.000
mean	3001.638353	2018.804376	779.972973	27.558559	55.796654	3699.907336	855.298584	10440.669241	4357.526
std	3870.201484	2451.113971	929.176190	17.640364	19.804778	4850.420531	1522.431887	4023.016484	1096.696
min	81.000000	72.000000	35.000000	1.000000	9.000000	139.000000	1.000000	2340.000000	1780.000
25%	776.000000	604.000000	242.000000	15.000000	41.000000	992.000000	95.000000	7320.000000	3597.000
50%	1558.000000	1110.000000	434.000000	23.000000	54.000000	1707.000000	353.000000	9990.000000	4200.000
75%	3624.000000	2424.000000	902.000000	35.000000	69.000000	4005.000000	967.000000	12925.000000	5050.000
max	48094.000000	26330.000000	6392.000000	96.000000	100.000000	31643.000000	21836.000000	21700.000000	8124.000

In [ ]:

```
#create a scatterplot of Grid rate vs RoomBoard where the points are colored by Private c
olumn
sns.set_style('darkgrid')
g=sns.FacetGrid(df,hue="Private",palette='coolwarm',size=6,aspect=2)
g=g.map(plt.hist,'Outstate',bins=20,alpha=0.7)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/axisgrid.py:337: UserWarning: The `size` p
arameter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```





K means cluster creation

In [ ]:

```
from sklearn.cluster import KMeans
```

In [ ]:

```
kmeans=KMeans(n_clusters=2)
```

In [ ]:

```
kmeans.fit(df.drop('Private',axis=1))
```

Out[ ]:

```
KMeans(n_clusters=2)
```

In [ ]:

```
kmeans.cluster_centers_
#"cluster center" is the arithmetic mean of all the points belonging to the cluster.
```

Out[ ]:

```
array([[1.03631389e+04, 6.55089815e+03, 2.56972222e+03, 4.14907407e+01,
        7.02037037e+01, 1.30619352e+04, 2.46486111e+03, 1.07191759e+04,
        4.64347222e+03, 5.95212963e+02, 1.71420370e+03, 8.63981481e+01,
        9.13333333e+01, 1.40277778e+01, 2.00740741e+01, 1.41705000e+04,
        6.75925926e+01],
       [1.81323468e+03, 1.28716592e+03, 4.91044843e+02, 2.53094170e+01,
        5.34708520e+01, 2.18854858e+03, 5.95458894e+02, 1.03957085e+04,
        4.31136472e+03, 5.41982063e+02, 1.28033632e+03, 7.04424514e+01,
        7.78251121e+01, 1.40997010e+01, 2.31748879e+01, 8.93204634e+03,
        6.51195815e+01]])
```

Evaluation

In [ ]:

```
#create a new column for df called 'Cluster' which is 1 for private and 0 for public
def converter(cluster):
  if cluster=='Yes':
    return 1
  else:
    return 0
```

In [ ]:

```
df['Cluster']=df['Private'].apply(converter)
```

In [ ]:

```
df.head()
```

	Private	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstate	Room.Board	Books	P
Abilene Christian University	Yes	1660	1232	721	23	52	2885	537	7440	3300	450	
Adelphi University	Yes	2186	1924	512	16	29	2683	1227	12280	6450	750	
Adrian College	Yes	1428	1097	336	22	50	1036	99	11250	3750	400	
Agnes Scott College	Yes	417	349	137	60	89	510	63	12960	5450	450	
Alaska Pacific University	Yes	193	146	55	16	44	249	869	7560	4120	800	

In [ ]:

```
#create a confusion matrix and classification report
from sklearn.metrics import confusion_matrix,classification_report
print(confusion_matrix(df['Cluster'],kmeans.labels_))
print(classification_report(df['Cluster'],kmeans.labels_))
```

```
[[ 74 138]
 [ 34 531]]
              precision    recall  f1-score   support

           0       0.69      0.35      0.46       212
           1       0.79      0.94      0.86       565

    accuracy                           0.78       777
   macro avg       0.74      0.64      0.66       777
weighted avg       0.76      0.78      0.75       777
```

RESULT: The program is executed successfully and output is obtained

COURSE OUTCOME 4

PROGRAM NO: 11

AIM : Programs on feedforward network to classify any standard dataset available in the public domain

---

Dataset : mnist dataset

ALGORITHM :

1. **Import required libraries**
2. **Split the dataset into training and testing sets**
3. **For preprocessing the data , reshape and reduce the x values of testing and training data**
4. **Convert the testing and training data in y to categorical velues**
5. **Assign sequential() model to a variable**
6. **For the first hidden layer, Add 64 dense layers to the model with sigmoid activation function 7.For the second hidden layer, Add 10 dense layers to the model with softmax activation function**
7. **Compile the model using least square method and optimize it using SGD**
8. **Fit the training dataset to the model and find the accuracy for n epochs**

PROGRAM CODE:

In [ ]:

```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense
from tensorflow.keras.optimizers import SGD
from matplotlib import pyplot as plt
```

In [ ]:

```
(x_train,y_train),(x_valid,y_valid)=mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.n
pz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```

In [ ]:

```
type(x_train)
```

Out[ ]:

```
numpy.ndarray
```

In [ ]:

```
x_train.shape
```

In [ ]:

```
y_train.shape
```

Out[ ]:

```
(60000,)
```

In [ ]:

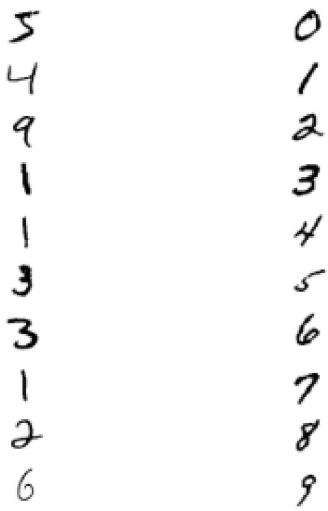
```
y_train[0:12]
```

Out[ ]:

```
array([5, 0, 4, 1, 9, 2, 1, 3, 1, 4, 3, 5], dtype=uint8)
```

In [ ]:

```
plt.figure(figsize=(5,5))
for k in range(20):
  plt.subplot(10,2,k+1)
  plt.imshow(x_train[k],cmap='Greys')
  plt.axis('off')
plt.show()
```

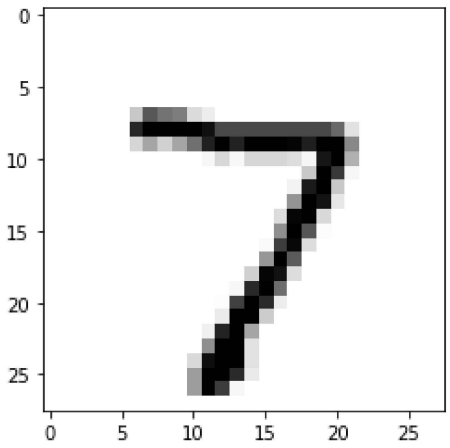


In [ ]:

```
plt.imshow(x_valid[0],cmap='Greys')
```

Out[ ]:

```
<matplotlib.image.AxesImage at 0x7f4caca394d0>
```



In [ ]:

```
array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,  84, 185, 159, 151,  60,  36,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0, 222, 254, 254, 254, 254, 241, 198,
        198, 198, 198, 198, 198, 198, 198, 170,  52,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,  67, 114,  72, 114, 163, 227, 254,
        225, 254, 254, 254, 250, 229, 254, 254, 140,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  17,  66,
         14,  67,  67,  67,  59,  21, 236, 254, 106,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,  83, 253, 209,  18,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,  22, 233, 255,  83,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0, 129, 254, 238,  44,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,  59, 249, 254,  62,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0, 133, 254, 187,   5,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   9, 205, 248,  58,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0, 126, 254, 182,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,  75, 251, 240,  57,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         19, 221, 254, 166,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   3,
        203, 254, 219,  35,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  38,
        254, 254,  77,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  31, 224,
        254, 115,   1,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
```

```
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  61, 242, 254,
         254,  52,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
           0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 121, 254, 254,
         219,  40,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
           0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 121, 254, 207,
          18,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
           0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
           0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
           0,   0]], dtype=uint8)
```

preprocess data

In [ ]:

```
x_train = x_train.reshape(60000,784).astype('float32')
x_valid = x_valid.reshape(10000,784).astype('float32')
```

In [ ]:

```
x_train/=255
x_valid/=255
```

In [ ]:

```
x_valid[0]
```

Out[ ]:

```
array([0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
```

```
0.        , 0.        , 0.32941177, 0.7254902 , 0.62352943,
0.5921569 , 0.23529412, 0.14117648, 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.87058824, 0.99607843, 0.99607843, 0.99607843, 0.99607843,
0.94509804, 0.7764706 , 0.7764706 , 0.7764706 , 0.7764706 ,
0.7764706 , 0.7764706 , 0.7764706 , 0.7764706 , 0.6666667 ,
0.20392157, 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.2627451 , 0.44705883,
0.28235295, 0.44705883, 0.6392157 , 0.8901961 , 0.99607843,
0.88235295, 0.99607843, 0.99607843, 0.99607843, 0.98039216,
0.8980392 , 0.99607843, 0.99607843, 0.54901963, 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.06666667, 0.25882354, 0.05490196, 0.2627451 ,
0.2627451 , 0.2627451 , 0.23137255, 0.08235294, 0.9254902 ,
0.99607843, 0.41568628, 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.3254902 , 0.99215686, 0.81960785, 0.07058824,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.08627451, 0.9137255 ,
1.        , 0.3254902 , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.5058824 , 0.99607843, 0.93333334, 0.17254902,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.23137255, 0.9764706 ,
0.99607843, 0.24313726, 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.52156866, 0.99607843, 0.73333335, 0.01960784,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.03529412, 0.8039216 ,
0.972549  , 0.22745098, 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.49411765, 0.99607843, 0.7137255 , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.29411766, 0.9843137 ,
0.9411765 , 0.22352941, 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
```

```
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.01176471, 0.79607844, 0.99607843,
       0.85882354, 0.13725491, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.14901961, 0.99607843, 0.99607843, 0.3019608 , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.12156863, 0.8784314 , 0.99607843,
       0.4509804 , 0.00392157, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.52156866, 0.99607843, 0.99607843, 0.20392157, 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.23921569, 0.9490196 , 0.99607843,
       0.99607843, 0.20392157, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.4745098 , 0.99607843, 0.99607843, 0.85882354, 0.15686275,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.4745098 , 0.99607843,
       0.8117647 , 0.07058824, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        ], dtype=float32)
```

In [ ]:

```
from keras import utils as np_utils
n_classes=10
y_train=keras.utils.np_utils.to_categorical(y_train,n_classes)
y_valid=keras.utils.np_utils.to_categorical(y_valid,n_classes)
```

In [ ]:

```
y_valid[0]
```

Out[ ]:

```
array([0., 0., 0., 0., 0., 0., 0., 1., 0., 0.], dtype=float32)
```

In [ ]:

```
x_valid[3]
```

Out[ ]:

```
array([0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
```

```
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.04313726,
0.5882353 , 0.99215686, 0.7921569 , 0.12156863, 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.14509805, 0.9843137 , 0.9843137 ,
0.99215686, 0.41960785, 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.08235294,
0.77254903, 0.9843137 , 0.9843137 , 0.99215686, 0.41960785,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.43137255, 0.74509805, 0.9843137 , 0.9843137 ,
0.9843137 , 0.99215686, 0.6627451 , 0.42745098, 0.24313726,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.99215686,
0.9843137 , 0.9843137 , 0.9843137 , 0.9843137 , 0.99215686,
0.9843137 , 0.9843137 , 0.8627451 , 0.2       , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.7137255 , 1.        , 0.99215686, 0.99215686,
0.99215686, 0.99215686, 0.91764706, 0.87058824, 0.99215686,
0.99215686, 0.99215686, 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.24705882, 0.8666667 ,
0.99215686, 0.9843137 , 0.9843137 , 0.9843137 , 0.5764706 ,
0.3019608 , 0.24313726, 0.5019608 , 0.9843137 , 0.9843137 ,
0.4117647 , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.1254902 , 0.90588236, 0.9843137 , 0.99215686, 0.9843137 ,
0.8627451 , 0.5372549 , 0.03921569, 0.        , 0.        ,
0.12156863, 0.9019608 , 0.9843137 , 0.9529412 , 0.44313726,
0.01960784, 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.14509805, 0.9843137 ,
0.9843137 , 0.99215686, 0.7372549 , 0.07843138, 0.        ,
0.        , 0.        , 0.        , 0.        , 0.42745098,
0.9843137 , 0.99215686, 0.9843137 , 0.13725491, 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
```

```
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.14509805,
0.99215686, 0.99215686, 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.1254902 , 0.7921569 , 1.        , 0.99215686, 0.6431373 ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.54901963, 0.9843137 , 0.9843137 ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.42745098, 0.9843137 ,
0.99215686, 0.9843137 , 0.13725491, 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.8509804 , 0.9843137 , 0.9843137 , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.08235294,
0.24705882, 0.90588236, 0.9843137 , 0.99215686, 0.9019608 ,
0.11764706, 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.8509804 , 0.9843137 ,
0.9843137 , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.5647059 , 0.9843137 , 0.9843137 ,
0.9843137 , 0.8666667 , 0.23921569, 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.8509804 , 0.9843137 , 0.9843137 , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.7137255 ,
0.8666667 , 0.9843137 , 0.9843137 , 0.9843137 , 0.7058824 ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.85490197,
0.99215686, 0.99215686, 0.28627452, 0.28627452, 0.89411765,
0.99215686, 0.99215686, 1.        , 0.99215686, 0.99215686,
0.99215686, 0.99215686, 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.44313726, 0.9843137 , 0.9843137 ,
0.99215686, 0.9843137 , 0.9843137 , 0.9843137 , 0.9843137 ,
0.99215686, 0.9843137 , 0.9843137 , 0.9843137 , 0.5764706 ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.12156863, 0.9019608 , 0.9843137 , 0.99215686, 0.9843137 ,
0.9843137 , 0.9843137 , 0.9843137 , 0.99215686, 0.9019608 ,
0.7411765 , 0.13725491, 0.03921569, 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.24313726,
0.5568628 , 0.99215686, 0.9843137 , 0.9843137 , 0.9843137 ,
0.9843137 , 0.99215686, 0.41960785, 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.28235295,
0.68235296, 0.9843137 , 0.6784314 , 0.2784314 , 0.28235295,
0.11764706, 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
```

```
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        ], dtype=float32)
```

In [ ]:

```
model=Sequential()
```

neurons in dense layers receive output from neurons of previous layer

In [ ]:

```
model.add(Dense(64, activation='sigmoid',input_shape=(784,)))
```

In [ ]:

```
model.add(Dense(10,activation='softmax'))
```

In [ ]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                50240

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 50,890
Trainable params: 50,890
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
model.compile(loss='mean_squared_error',optimizer=SGD(learning_rate=0.01),metrics=['accuracy'])
```

In [ ]:

```
history=model.fit(x_train,y_train,batch_size=125,epochs=75,verbose=1)
```

```
Epoch 1/100
600/600 [==============================] - 3s 4ms/step - loss: 0.0564 - accuracy: 0.6581
Epoch 2/100
600/600 [==============================] - 2s 4ms/step - loss: 0.0558 - accuracy: 0.6628
Epoch 3/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0553 - accuracy: 0.6675
Epoch 4/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0548 - accuracy: 0.6720
Epoch 5/100
600/600 [==============================] - 2s 4ms/step - loss: 0.0543 - accuracy: 0.6761
Epoch 6/100
600/600 [==============================] - 2s 4ms/step - loss: 0.0538 - accuracy: 0.6809
Epoch 7/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0533 - accuracy: 0.6844
Epoch 8/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0528 - accuracy: 0.6880
Epoch 9/100
```

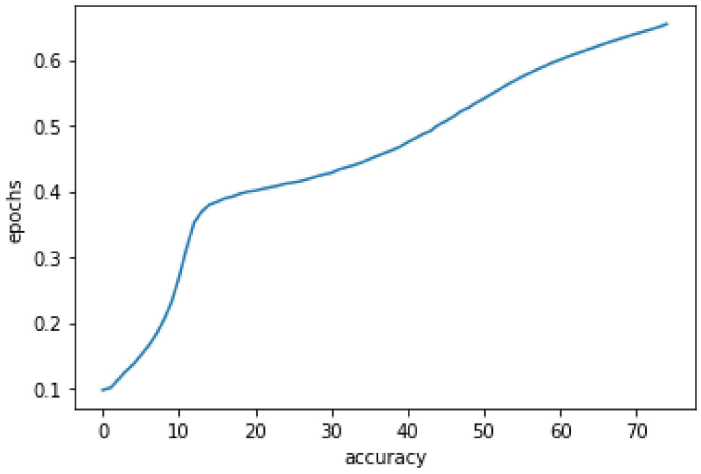
600/600 [==============================] - 2s 4ms/step - loss: 0.0513 - accuracy: 0.7006
Epoch 12/100
600/600 [==============================] - 2s 4ms/step - loss: 0.0509 - accuracy: 0.7039
Epoch 13/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0504 - accuracy: 0.7080
Epoch 14/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0500 - accuracy: 0.7117
Epoch 15/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0495 - accuracy: 0.7149
Epoch 16/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0491 - accuracy: 0.7184
Epoch 17/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0487 - accuracy: 0.7217
Epoch 18/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0483 - accuracy: 0.7247
Epoch 19/100
600/600 [==============================] - 2s 4ms/step - loss: 0.0478 - accuracy: 0.7282
Epoch 20/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0474 - accuracy: 0.7318
Epoch 21/100
600/600 [==============================] - 2s 4ms/step - loss: 0.0470 - accuracy: 0.7345
Epoch 22/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0466 - accuracy: 0.7378
Epoch 23/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0463 - accuracy: 0.7408
Epoch 24/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0459 - accuracy: 0.7433
Epoch 25/100
600/600 [==============================] - 2s 4ms/step - loss: 0.0455 - accuracy: 0.7461
Epoch 26/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0451 - accuracy: 0.7485
Epoch 27/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0448 - accuracy: 0.7504
Epoch 28/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0444 - accuracy: 0.7527
Epoch 29/100
600/600 [==============================] - 2s 4ms/step - loss: 0.0441 - accuracy: 0.7552
Epoch 30/100
600/600 [==============================] - 2s 4ms/step - loss: 0.0437 - accuracy: 0.7572
Epoch 31/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0434 - accuracy: 0.7594
Epoch 32/100
600/600 [==============================] - 2s 4ms/step - loss: 0.0430 - accuracy: 0.7611
Epoch 33/100
600/600 [==============================] - 2s 4ms/step - loss: 0.0427 - accuracy: 0.7633
Epoch 34/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0424 - accuracy: 0.7651
Epoch 35/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0421 - accuracy: 0.7672
Epoch 36/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0417 - accuracy: 0.7690
Epoch 37/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0414 - accuracy: 0.7710
Epoch 38/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0411 - accuracy: 0.7727
Epoch 39/100
600/600 [==============================] - 2s 4ms/step - loss: 0.0408 - accuracy: 0.7742
Epoch 40/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0405 - accuracy: 0.7759
Epoch 41/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0402 - accuracy: 0.7787
Epoch 42/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0399 - accuracy: 0.7807
Epoch 43/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0397 - accuracy: 0.7824
Epoch 44/100
600/600 [==============================] - 2s 4ms/step - loss: 0.0394 - accuracy: 0.7843
Epoch 45/100

600/600 [==============================] - 2s 3ms/step - loss: 0.0385 - accuracy: 0.7905
Epoch 48/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0383 - accuracy: 0.7928
Epoch 49/100
600/600 [==============================] - 2s 4ms/step - loss: 0.0380 - accuracy: 0.7951
Epoch 50/100
600/600 [==============================] - 2s 4ms/step - loss: 0.0378 - accuracy: 0.7970
Epoch 51/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0375 - accuracy: 0.7987
Epoch 52/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0372 - accuracy: 0.8004
Epoch 53/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0370 - accuracy: 0.8028
Epoch 54/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0367 - accuracy: 0.8049
Epoch 55/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0365 - accuracy: 0.8064
Epoch 56/100
600/600 [==============================] - 2s 4ms/step - loss: 0.0363 - accuracy: 0.8090
Epoch 57/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0360 - accuracy: 0.8106
Epoch 58/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0358 - accuracy: 0.8122
Epoch 59/100
600/600 [==============================] - 2s 4ms/step - loss: 0.0355 - accuracy: 0.8141
Epoch 60/100
600/600 [==============================] - 2s 4ms/step - loss: 0.0353 - accuracy: 0.8158
Epoch 61/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0351 - accuracy: 0.8178
Epoch 62/100
600/600 [==============================] - 2s 4ms/step - loss: 0.0349 - accuracy: 0.8196
Epoch 63/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0346 - accuracy: 0.8212
Epoch 64/100
600/600 [==============================] - 2s 4ms/step - loss: 0.0344 - accuracy: 0.8227
Epoch 65/100
600/600 [==============================] - 2s 4ms/step - loss: 0.0342 - accuracy: 0.8245
Epoch 66/100
600/600 [==============================] - 2s 4ms/step - loss: 0.0340 - accuracy: 0.8257
Epoch 67/100
600/600 [==============================] - 2s 4ms/step - loss: 0.0338 - accuracy: 0.8269
Epoch 68/100
600/600 [==============================] - 2s 4ms/step - loss: 0.0336 - accuracy: 0.8286
Epoch 69/100
600/600 [==============================] - 2s 4ms/step - loss: 0.0334 - accuracy: 0.8299
Epoch 70/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0332 - accuracy: 0.8314
Epoch 71/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0330 - accuracy: 0.8326
Epoch 72/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0328 - accuracy: 0.8333
Epoch 73/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0326 - accuracy: 0.8348
Epoch 74/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0324 - accuracy: 0.8359
Epoch 75/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0322 - accuracy: 0.8370
Epoch 76/100
600/600 [==============================] - 2s 4ms/step - loss: 0.0320 - accuracy: 0.8380
Epoch 77/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0318 - accuracy: 0.8393
Epoch 78/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0316 - accuracy: 0.8406
Epoch 79/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0314 - accuracy: 0.8416
Epoch 80/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0313 - accuracy: 0.8426
Epoch 81/100

```
600/600 [==============================] - 2s 3ms/step - loss: 0.0307 - accuracy: 0.8457
Epoch 84/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0306 - accuracy: 0.8465
Epoch 85/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0304 - accuracy: 0.8472
Epoch 86/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0302 - accuracy: 0.8480
Epoch 87/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0301 - accuracy: 0.8487
Epoch 88/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0299 - accuracy: 0.8497
Epoch 89/100
600/600 [==============================] - 2s 4ms/step - loss: 0.0297 - accuracy: 0.8503
Epoch 90/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0296 - accuracy: 0.8511
Epoch 91/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0294 - accuracy: 0.8523
Epoch 92/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0293 - accuracy: 0.8529
Epoch 93/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0291 - accuracy: 0.8534
Epoch 94/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0290 - accuracy: 0.8542
Epoch 95/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0288 - accuracy: 0.8548
Epoch 96/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0287 - accuracy: 0.8555
Epoch 97/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0285 - accuracy: 0.8561
Epoch 98/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0284 - accuracy: 0.8569
Epoch 99/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0282 - accuracy: 0.8574
Epoch 100/100
600/600 [==============================] - 2s 3ms/step - loss: 0.0281 - accuracy: 0.8582
```

In [ ]:

```
plt.plot(history.history['accuracy'])
plt.xlabel('accuracy')
plt.ylabel('epochs')
plt.show()
```



RESULT: The program is executed successfully and output is obtained

PROGRAM NO: 12

AIM: Programs on convolutional neural network to classify images from any standard dataset in the public domain

---

Dataset: cifar10

ALGORITHM:

1. **Import required libraries**
2. **Split the dataset into training and testing sets**
3. **Normalise the x values of testing and training data**
4. **For feature extraction from the model, we use 2D convolution with the relu activation function and pooling layers.**
5. **For classification, we flatten the model and add dense layers with activation function such as relu and softmax**
6. **Compile the model using least square method and optimize it using adam**
7. **Fit the training dataset to the model and find the accuracy for n epochs**
8. **Predict the accuracy and print the classification report**

PROGRAM CODE:

In [ ]:

```
import tensorflow as tf
from tensorflow.keras import datasets,layers,models
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import classification_report
```

In [ ]:

```
(x_train,y_train),(x_test,y_test)=datasets.cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 4s 0us/step
170508288/170498071 [==============================] - 4s 0us/step
```

In [ ]:

```
x_train.shape
```

Out[ ]:

```
(50000, 32, 32, 3)
```

In [ ]:

```
x_test.shape
```

Out[ ]:

```
(10000, 32, 32, 3)
```

```
]
```

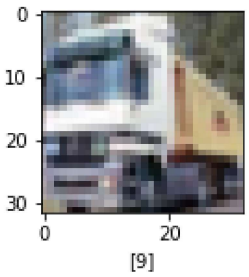
In [ ]:

```
def plot_sample(x,y,index):
  plt.figure(figsize=(14,2))
  plt.imshow(x[index])
  plt.xlabel(y[index])
```

In [ ]:

```
plot_sample(x_train,y_train,1)
```

```
/usr/local/lib/python3.7/dist-packages/matplotlib/text.py:1165: FutureWarning: elementwis
e comparison failed; returning scalar instead, but in the future will perform elementwise
comparison
  if s != self._text:
```



In [ ]:

```
#normalize
x_train=x_train/255
x_test=x_test/255
```

In [ ]:

```
#model
cnn=models.Sequential([
                       #feature extraction
                       layers.Conv2D(filters=32,activation='relu',kernel_size=(3,3),inpu
t_shape=(32,32,3)),
                       layers.MaxPool2D((2,2)),

                       layers.Conv2D(filters=32,activation='relu',kernel_size=(3,3),inpu
t_shape=(32,32,3)),
                       layers.MaxPool2D((2,2)),

                       #classification
                       layers.Flatten(),
                       layers.Dense(64,activation='relu'),
                       layers.Dense(10,activation='softmax')


])
```

In [ ]:

```
cnn.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
```

In [ ]:

```
cnn.fit(x_train,y_train,epochs=20)
```

```
Epoch 1/20
1563/1563 [==============================] - 52s 33ms/step - loss: 1.4914 - accuracy: 0.4
629
Epoch 2/20
```

```
387
Epoch 4/20
1563/1563 [==============================] - 53s 34ms/step - loss: 0.9614 - accuracy: 0.6
653
Epoch 5/20
1563/1563 [==============================] - 53s 34ms/step - loss: 0.9017 - accuracy: 0.6
883
Epoch 6/20
1563/1563 [==============================] - 52s 33ms/step - loss: 0.8543 - accuracy: 0.7
032
Epoch 7/20
1563/1563 [==============================] - 52s 33ms/step - loss: 0.8155 - accuracy: 0.7
169
Epoch 8/20
1563/1563 [==============================] - 52s 33ms/step - loss: 0.7773 - accuracy: 0.7
292
Epoch 9/20
1563/1563 [==============================] - 52s 33ms/step - loss: 0.7451 - accuracy: 0.7
412
Epoch 10/20
1563/1563 [==============================] - 52s 33ms/step - loss: 0.7124 - accuracy: 0.7
524
Epoch 11/20
1563/1563 [==============================] - 52s 33ms/step - loss: 0.6859 - accuracy: 0.7
615
Epoch 12/20
1563/1563 [==============================] - 52s 33ms/step - loss: 0.6593 - accuracy: 0.7
690
Epoch 13/20
1563/1563 [==============================] - 52s 33ms/step - loss: 0.6325 - accuracy: 0.7
780
Epoch 14/20
1563/1563 [==============================] - 51s 33ms/step - loss: 0.6090 - accuracy: 0.7
845
Epoch 15/20
1563/1563 [==============================] - 51s 33ms/step - loss: 0.5830 - accuracy: 0.7
950
Epoch 16/20
1563/1563 [==============================] - 52s 33ms/step - loss: 0.5643 - accuracy: 0.8
006
Epoch 17/20
1563/1563 [==============================] - 52s 33ms/step - loss: 0.5435 - accuracy: 0.8
082
Epoch 18/20
1563/1563 [==============================] - 52s 33ms/step - loss: 0.5266 - accuracy: 0.8
140
Epoch 19/20
1563/1563 [==============================] - 52s 33ms/step - loss: 0.5091 - accuracy: 0.8
206
Epoch 20/20
1563/1563 [==============================] - 52s 33ms/step - loss: 0.4902 - accuracy: 0.8
281
```

Out[ ]:

```
<keras.callbacks.History at 0x7fac364784d0>
```

In [ ]:

```
y_pred=cnn.predict(x_test)
```

In [ ]:

```
cnn.evaluate(x_test,y_test)
```

```
313/313 [==============================] - 4s 11ms/step - loss: 1.0884 - accuracy: 0.6832
```

Out[ ]:

```
y_test=y_test.reshape(-1)
y_pred=cnn.predict(x_test)
```

In [ ]:

```
y_classes=[np.argmax(element) for element in y_pred]
print('Classification report: \n',classification_report(y_test,y_classes))
```

```
Classification report:
               precision    recall  f1-score   support

           0       0.65      0.78      0.71      1000
           1       0.80      0.78      0.79      1000
           2       0.60      0.54      0.57      1000
           3       0.50      0.48      0.49      1000
           4       0.66      0.66      0.66      1000
           5       0.62      0.55      0.58      1000
           6       0.69      0.82      0.75      1000
           7       0.76      0.72      0.74      1000
           8       0.84      0.72      0.77      1000
           9       0.73      0.79      0.76      1000

    accuracy                           0.68     10000
   macro avg       0.68      0.68      0.68     10000
weighted avg       0.68      0.68      0.68     10000
```

RESULT: The program is executed successfully and output is obtained

COURSE OUTCOME 5

PROGRAM NO: 13

AIM: Web Data Mining

1.Implement a simple web crawler (ensure ethical conduct). 2.Implement a program to scrap the web page of any popular website – suggested python package is scrappy (ensure ethical conduct).

PROGRAM CODE:

Simple web Crawler:

In [ ]:
```
from bs4 import BeautifulSoup
import requests
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [ ]:
```
webpage = requests.get("https://www.lenskart.com/eyeglasses/promotions.html?utm_source=g
```

In [ ]:
```
webpage
```

Out[ ]:
```
<Response [200]>
```

In [ ]:
```
soup = BeautifulSoup(webpage.content,"html.parser")
```

In [ ]:
```
ProductSoup = soup.find_all("div",class_="product-name")
pricesoup = soup.find_all("span",class_="text-color-dark-blue fw700")
ratingsoup = soup.find_all("span", class_="rating-value")
```

In [ ]:
```
ProductSoup
```

Out[ ]:
```
[<div class="product-name">Aqualens</div>,
 <div class="product-name">Aqualens</div>,
 <div class="product-name">John Jacobs</div>,
 <div class="product-name">Vincent Chase</div>,
 <div class="product-name">Vincent Chase</div>,
 <div class="product-name">Lenskart Air</div>,
 <div class="product-name">Lenskart Air</div>,
 <div class="product-name">John Jacobs</div>,
 <div class="product-name">Aqualens</div>]
```

In [ ]:
```
lens_names = []
for i in range(0, len(ProductSoup)):
  lens_names.append(ProductSoup[i].get_text())
```

In [ ]:
```
lens_names
```

Out[ ]:
```
['Aqualens',
 'Aqualens',
 'John Jacobs',
 'Vincent Chase',
 'Vincent Chase',
 'Lenskart Air',
 'Lenskart Air',
 'John Jacobs',
 'Aqualens']
```

In [ ]:
```
pricesoup
```

Out[ ]:
```
[<span class="text-color-dark-blue fw700"><span>₹</span>240</span>,
 <span class="text-color-dark-blue fw700"><span>₹</span>123</span>,
 <span class="text-color-dark-blue fw700"><span>₹</span>3000</span>,
 <span class="text-color-dark-blue fw700"><span>₹</span>1199</span>,
 <span class="text-color-dark-blue fw700"><span>₹</span>1199</span>,
 <span class="text-color-dark-blue fw700"><span>₹</span>1199</span>,
 <span class="text-color-dark-blue fw700"><span>₹</span>1199</span>,
 <span class="text-color-dark-blue fw700"><span>₹</span>3000</span>,
 <span class="text-color-dark-blue fw700"><span>₹</span>440</span>]
```

In [ ]:
```
lens_prices = []
for i in range(0, len(pricesoup)):
  lens_prices.append(pricesoup[i].get_text())
```

In [ ]:
```
lens_prices
```

Out[ ]:
```
['₹240', '₹123', '₹3000', '₹1199', '₹1199', '₹1199', '₹1199', '₹3000', '₹440']
```

In [ ]:
```
lens_ratings = []
for i in range(0, len(ratingsoup)):
  lens_ratings.append(ratingsoup[i].get_text())
```

In [ ]:
```
lens_ratings
```

Out[ ]:
```
['4.8', '4.8', '4.4', '4.6', '4.6', '4.6', '4.6', '4.4', '4.7']
```

In [ ]:
```
print(len(lens_names), len(lens_prices), len(lens_ratings))
```

```
9 9 9
```

In [ ]:
```
import pandas as pd
```

In [ ]:
```
final = pd.DataFrame({"Product_name":lens_names,"Price":lens_prices,"Ratings":lens_ratin
```

In [ ]:
```
final
```

Out[ ]:

	Product_name	Price	Ratings
0	Aqualens	₹240	4.8
1	Aqualens	₹123	4.8
2	John Jacobs	₹3000	4.4
3	Vincent Chase	₹1199	4.6
4	Vincent Chase	₹1199	4.6
5	Lenskart Air	₹1199	4.6
6	Lenskart Air	₹1199	4.6
7	John Jacobs	₹3000	4.4
8	Aqualens	₹440	4.7

In [ ]:
```
final.to_csv('product.csv',index=False)
```

RESULT: The program is executed successfully and output is obtained

PROGRAM NO: 14

AIM: Natural Language Processing

- Part of Speech tagging
- N-gram and smoothening
- Chunking

Dataset: Reviews.csv

PROGRAM CODE:

In [ ]:
```
import nltk
from nltk import word_tokenize
nltk.download('punkt') #punkt is used for tokenising sentences and
                       # averaged_perceptron_tagger is used for tagging words with their
nltk.download('averaged_perceptron_tagger')
nltk.download('tagsets')#The process of classifying words into their parts of speech and
                         # labeling them accordingly is known as part-of-speech tagging,
                         # Parts of speech are also known as word classes or lexical cate
                         # The collection of tags used for a particular task is known as
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data] Downloading package tagsets to /root/nltk_data...
[nltk_data]   Unzipping help/tagsets.zip.
```

Out[ ]:
```
True
```

In [ ]:
```
nltk.help.upenn_tagset()
```

```
$: dollar
    $ -$ --$ A$ C$ HK$ M$ NZ$ S$ U.S.$ US$
'': closing quotation mark
    ' ''
(: opening parenthesis
    ( [ {
): closing parenthesis
    ) ] }
,: comma
    ,
--: dash
    --
.: sentence terminator
    . ! ?
:: colon or ellipsis
    : ; ...
CC: conjunction, coordinating
    & 'n and both but either et for less minus neither nor or plus so
    therefore times v. versus vs. whether yet
CD: numeral, cardinal
    mid-1890 nine-thirty forty-two one-tenth ten million 0.5 one forty-
    seven 1987 twenty '79 zero two 78-degrees eighty-four IX '60s .025
    fifteen 271,124 dozen quintillion DM2,000 ...
DT: determiner
    all an another any both del each either every half la many much nary
    neither no some such that the them these this those
```

EX: existential there
    there
FW: foreign word
    gemeinschaft hund ich jeux habeas Haementeria Herr K'ang-si vous
    lutihaw alai je jour objets salutaris fille quibusdam pas trop Monte
    terram fiche oui corporis ...
IN: preposition or conjunction, subordinating
    astride among uppon whether out inside pro despite on by throughout
    below within for towards near behind atop around if like until below
    next into if beside ...
JJ: adjective or numeral, ordinal
    third ill-mannered pre-war regrettable oiled calamitous first separable
    ectoplasmic battery-powered participatory fourth still-to-be-named
    multilingual multi-disciplinary ...
JJR: adjective, comparative
    bleaker braver breezier briefer brighter brisker broader bumper busier
    calmer cheaper choosier cleaner clearer closer colder commoner costlier
    cozier creamier crunchier cuter ...
JJS: adjective, superlative
    calmest cheapest choicest classiest cleanest clearest closest commonest
    corniest costliest crassest creepiest crudest cutest darkest deadliest
    dearest deepest densest dinkiest ...
LS: list item marker
    A A. B B. C C. D E F First G H I J K One SP-44001 SP-44002 SP-44005
    SP-44007 Second Third Three Two * a b c d first five four one six three
    two
MD: modal auxiliary
    can cannot could couldn't dare may might must need ought shall should
    shouldn't will would
NN: noun, common, singular or mass
    common-carrier cabbage knuckle-duster Casino afghan shed thermostat
    investment slide humour falloff slick wind hyena override subhumanity
    machinist ...
NNP: noun, proper, singular
    Motown Venneboerger Czestochwa Ranzer Conchita Trumplane Christos
    Oceanside Escobar Kreisler Sawyer Cougar Yvette Ervin ODI Darryl CTCA
    Shannon A.K.C. Meltex Liverpool ...
NNPS: noun, proper, plural
    Americans Americas Amharas Amityvilles Amusements Anarcho-Syndicalists
    Andalusians Andes Andruses Angels Animals Anthony Antilles Antiques
    Apache Apaches Apocrypha ...
NNS: noun, common, plural
    undergraduates scotches bric-a-brac products bodyguards facets coasts
    divestitures storehouses designs clubs fragrances averages
    subjectivists apprehensions muses factory-jobs ...
PDT: pre-determiner
    all both half many quite such sure this
POS: genitive marker
    ' 's
PRP: pronoun, personal
    hers herself him himself hisself it itself me myself one oneself ours
    ourselves ownself self she thee theirs them themselves they thou thy us
PRP$: pronoun, possessive
    her his mine my our ours their thy your
RB: adverb
    occasionally unabatingly maddeningly adventurously professedly
    stirringly prominently technologically magisterially predominately
    swiftly fiscally pitilessly ...
RBR: adverb, comparative
    further gloomier grander graver greater grimmer harder harsher
    healthier heavier higher however larger later leaner lengthier less-
    perfectly lesser lonelier longer louder lower more ...
RBS: adverb, superlative
    best biggest bluntest earliest farthest first furthest hardest
    heartiest highest largest least less most nearest second tightest worst
RP: particle

```
    aboard about across along apart around aside at away back before behind
    by crop down ever fast for forth from go high i.e. in into just later
    low more off on open out over per pie raising start teeth that through
    under unto up up-pp upon whole with you
SYM: symbol
    % & ' '' ''. ) ). * + ,. < = > @ A[fj] U.S U.S.S.R * ** ***
TO: "to" as preposition or infinitive marker
    to
UH: interjection
    Goodbye Goody Gosh Wow Jeepers Jee-sus Hubba Hey Kee-reist Oops amen
    huh howdy uh dammit whammo shucks heck anyways whodunnit honey golly
    man baby diddle hush sonuvabitch ...
VB: verb, base form
    ask assemble assess assign assume atone attention avoid bake balkanize
    bank begin behold believe bend benefit bevel beware bless boil bomb
    boost brace break bring broil brush build ...
VBD: verb, past tense
    dipped pleaded swiped regummed soaked tidied convened halted registered
    cushioned exacted snubbed strode aimed adopted belied figgered
    speculated wore appreciated contemplated ...
VBG: verb, present participle or gerund
    telegraphing stirring focusing angering judging stalling lactating
    hankerin' alleging veering capping approaching traveling besieging
    encrypting interrupting erasing wincing ...
VBN: verb, past participle
    multihulled dilapidated aerosolized chaired languished panelized used
    experimented flourished imitated reunifed factored condensed sheared
    unsettled primed dubbed desired ...
VBP: verb, present tense, not 3rd person singular
    predominate wrap resort sue twist spill cure lengthen brush terminate
    appear tend stray glisten obtain comprise detest tease attract
    emphasize mold postpone sever return wag ...
VBZ: verb, present tense, 3rd person singular
    bases reconstructs marks mixes displeases seals carps weaves snatches
    slumps stretches authorizes smolders pictures emerges stockpiles
    seduces fizzes uses bolsters slaps speaks pleads ...
WDT: WH-determiner
    that what whatever which whichever
WP: WH-pronoun
    that what whatever whatsoever which who whom whosoever
WP$: WH-pronoun, possessive
    whose
WRB: Wh-adverb
    how however whence whenever where whereby whereever wherein whereof why
``: opening quotation mark
    ` ``
```

In [ ]:
```
sentence = word_tokenize("Third wave of corona virus is here")
nltk.pos_tag(sentence)
```

Out[ ]:
```
[('Third', 'JJ'),
 ('wave', 'NN'),
 ('of', 'IN'),
 ('corona', 'NN'),
 ('virus', 'NN'),
 ('is', 'VBZ'),
 ('here', 'RB')]
```

In [ ]:
```
<s> I live in kollam </s>
# n GRAMS IN SENTIMENT ANALYSIS
import numpy as np
import pandas as pd
from sklearn import model_selection, naive_bayes, svm
```

In [ ]:
```
df=pd.read_csv("/content/sample_data/Reviews.csv",engine='python',error_bad_lines=False)
```

```
#Lines with too many fields (e.g. a csv line with too many commas) will by default cause
#If False, then these "bad lines" will dropped from the DataFrame that is returned.
```

```
/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py:2882: FutureWarn
ing: The error_bad_lines argument has been deprecated and will be removed in a future ve
rsion.


  exec(code_obj, self.user_global_ns, self.user_ns)
Skipping line 10366: unexpected end of data
```

In [ ]:
```
df.head(2)
```

Out[ ]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	130
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1	134

In [ ]:
```
df.shape
```

Out[ ]:
```
(10364, 10)
```

In [ ]:
```
df.isnull().sum()
```

Out[ ]:
```
Id                      0
ProductId               0
UserId                  0
ProfileName             0
HelpfulnessNumerator    0
HelpfulnessDenominator  0
Score                   0
Time                    0
Summary                 0
Text                    0
dtype: int64
```

In [ ]:
```
df.dropna(inplace=True)
```

In [ ]:
```
df.isnull().sum()
```

Out[ ]:
```
Id                      0
ProductId               0
UserId                  0
ProfileName             0
HelpfulnessNumerator    0
HelpfulnessDenominator  0
Score                   0
Time                    0
Summary                 0
Text                    0
dtype: int64
```

In [ ]:
```
df['Score'].value_counts()
```

Out[ ]:
```
5    6404
4    1473
1     985
3     889
2     613
Name: Score, dtype: int64
```

In [ ]:
```
df['positively rated']=np.where(df['Score']>=3,1,0)
```

In [ ]:
```
df.head(2)
```

Out[ ]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	130
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1	134

In [ ]:
```
df['positively rated'].value_counts()
```

Out[ ]:
```
1    8766
0    1598
Name: positively rated, dtype: int64
```

In [ ]:
```
#Highly imbalanced
from sklearn.model_selection import train_test_split
```

In [ ]:
```
X_train,X_test,y_train,y_test=train_test_split(df["Summary"],df["positively rated"],rand
```

In [ ]:
```
print(X_train)
```

```
8885                                 Smooth to the last drop
3675                                          Great muffins!
4814                                        this is the one
3775                                                    YUM
5682                               Davidson's Darjeeling Tea
                                 ...
8324                                     Tasteless and watery
10206                                                Yummy!
6253                                               Nice tea
10123    The Microwave Pork Rinds Are Surpisingly Fanta...
5600                                          Yuban coffee
Name: Summary, Length: 7773, dtype: object
```

In [ ]:
```
df.head(2)
```

Out[ ]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	130
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1	134

In [ ]:
```
print(y_train)
```

```
8885     1
3675     1
4814     1
3775     1
5682     0
        ..
8324     0
10206    1
6253     1
10123    1
5600     1
Name: positively rated, Length: 7773, dtype: int64
```

In [ ]:
```
from sklearn.feature_extraction.text import CountVectorizer
```

In [ ]:
```
vect=CountVectorizer(min_df=5,ngram_range=(1,2),).fit(X_train)#minimum word count should
```

In [ ]:
```
len(vect.get_feature_names())#This will print feature names selected (terms selected) fr
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: F
unction get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will
be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

Out[ ]:
```
1395
```

In [ ]:
```
X_train_vectorized=vect.transform(X_train)
```

In [ ]:
```
from sklearn.naive_bayes import MultinomialNB
model=MultinomialNB()
model.fit(X_train_vectorized,y_train)
```

Out[ ]:
```
MultinomialNB()
```

In [ ]:
```
predictions=model.predict(vect.transform(X_test))
```

In [ ]:
```
print(predictions)
```

```
[0 1 1 ... 1 1 0]
```

In [ ]:
```
from sklearn.metrics import classification_report,confusion_matrix
```

In [ ]:
```
print(confusion_matrix(y_test,predictions))
print(classification_report(y_test,predictions))
```

```
[[ 201  192]
 [ 116 2082]]
              precision    recall  f1-score   support

           0       0.63      0.51      0.57       393
           1       0.92      0.95      0.93      2198

    accuracy                           0.88      2591
   macro avg       0.77      0.73      0.75      2591
weighted avg       0.87      0.88      0.88      2591
```

In [ ]:
```
from sklearn.metrics import roc_auc_score
print("AUC score is",roc_auc_score(y_test,predictions))
```

```
AUC score is 0.7293375657259549
```

In [ ]:
```
import nltk
nltk.download('averaged_perceptron_tagger')
sample_text="""
The function of education is to teach one to think intensively and to think critically.
"""
tokenized=nltk.sent_tokenize(sample_text)
for i in tokenized:
  words=nltk.word_tokenize(i)
  # print(words)
  tagged_words=nltk.pos_tag(words)
  # print(tagged_words)
  chunkGram=r"""chunk: {<RB.?>*<VB.?>*<NNP><NN>?}"""
  chunkParser=nltk.RegexpParser(chunkGram)
  chunked=chunkParser.parse(tagged_words)
  print(chunked)
 # chunked.draw()
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
(S
  The/DT
  function/NN
  of/IN
  education/NN
  is/VBZ
  to/TO
  teach/VB
  one/CD
  to/TO
  think/VB
  intensively/RB
  and/CC
  to/TO
  think/VB
  critically/RB
  ./.)
(S
  Intelligence/NN
  plus/CC
  character/NN
  -/:
  that/WDT
  is/VBZ
  the/DT
```

```
goal/NN
of/IN
true/JJ
education/NN
./.)
```

RESULT: The program is executed successfully and output is obtained