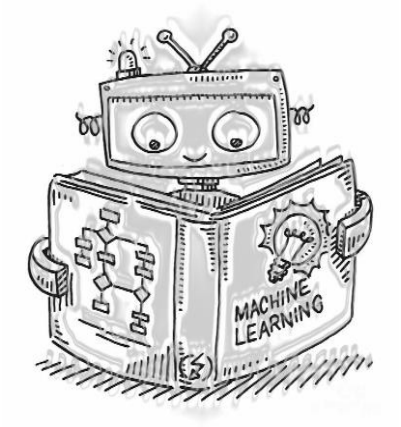
Atma Ram Sanatan Dharma College

University of Delhi



Machine Learning

Practical File

Submitted By -

Anshul Verma
Roll No. – 19/78065
B.Sc. (Hons.) Computer Science

Submitted To -

Ms. Uma Ojha
Department of Computer Science

1

Perform elementary mathematical operations in Octave/MATLAB/R/Python like addition, multiplication, division and exponentiation.

In [1]:
```
# Addition
5 + 7
```

Out[1]: 12

In [2]:
```
# Subtraction
5 - 7
```

Out[2]: -2

In [3]:
```
# Multiplication
5 * 7
```

Out[3]: 35

In [4]:
```
# Division
5 / 7
```

Out[4]: 0.7142857142857143

In [5]:
```
# Exponentiation
5 ^ 7
```

Out[5]: 2

2

Perform elementary logical operations in Octave/MATLAB/R/Python (like OR, AND, Checking for Equality, NOT, XOR).

In [6]:
```
# OR
True | False
```

Out[6]: True

In [7]:
```
# AND
True & False
```

Out[7]: False

In [8]:
```
# Check for inequality
True == False
```

Out[8]:
```
False
```

In [9]:
```
# NOT
~True
```

Out[9]:
```
-2
```

In [10]:
```
# XOR
print(True ^ True)
print(True ^ False)
```

```
False
True
```

3

Create, initialize and display simple variables and simple strings and use simple formatting for variable.

In [11]:
```
# Creating and initializing a variable
a = 5
x1 = 10
x2 = 20
name = "Anshul"
```

In [12]:
```
# Displaying variables
print(a)
print(x1)
print(x2)
print(name)
```

```
5
10
20
Anshul
```

4

Create/Define single dimension / multi-dimension arrays, and arrays with specific values like array of all ones, all zeros, array with random values within a range, or a diagonal matrix.

In [13]:
```
import numpy as np
```

In [14]:
```
# Creating single-dimension arrays
x = np.array([1, 2, 3, 4, 5])
print('x = ', x)
```

```
y = np.array([[1], [2], [3]])
print('y = \n', y)
```

```
x =  [1 2 3 4 5]
y =
 [[1]
 [2]
 [3]]
```

In [15]:
```
# Creating multi-dimension arrays
z = np.array([[1, 2, 3], [6, 7, 8]])
print('z = \n', z)

z1 = np.matrix('1 2 3; 6 7 8')
print('z1 = \n', z1)
```

```
z =
 [[1 2 3]
 [6 7 8]]
z1 =
 [[1 2 3]
 [6 7 8]]
```

In [16]:
```
# Matrix with all ones
A = np.ones((4, 4))
A
```

Out[16]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

In [17]:
```
# Matrix with all zeros
B = np.zeros((4, 4))
B
```

Out[17]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

In [18]:
```
# Matrix with random values within a range
C = np.random.randint(20, 50, (4,5))
print("C = \n", C)

# Matrix with range
C1 = np.arange(12).reshape((3, 4))
print("C1 = \n", C1)
```

```
C =
 [[30 39 40 29 32]
 [23 47 36 32 22]
 [41 27 26 20 49]
 [35 39 46 30 44]]
C1 =
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

In [19]:
```
# Diagonal matrix
D = np.diag([1, 2, 3, 4, 5])
print('D = \n', D)
```

```
D =
```

```
[[1 0 0 0 0]
[0 2 0 0 0]
[0 0 3 0 0]
[0 0 0 4 0]
[0 0 0 0 5]]
```

5

Use command to compute the size of a matrix, size/length of a particular row/column, load data from a text file, store matrix data to a text file, finding out variables and their features in the current scope.

In [20]:
```
import numpy

A = np.arange(12).reshape((3, 4))
A
```

Out[20]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [21]:
```
# Size of matrix
# A.size
np.size(A)
```

Out[21]:
```
12
```

In [22]:
```
# Shape of matrix
A.shape
```

Out[22]:
```
(3, 4)
```

In [23]:
```
# Length of 1st row
len(A[0])
```

Out[23]:
```
4
```

In [24]:
```
# Length of first column
np.size(A, 0)
```

Out[24]:
```
3
```

In [25]:
```
# Loading data from text file
#np.loadtxt("input5.txt", delimiter=',', dtype=int)
import pandas as pd

df = pd.read_csv('input5.txt')
df
```

Out[25]:

	Sell	"List"	"Living"	"Rooms"	"Beds"	"Baths"	"Age"	"Acres"	"Taxes"
0	142	160	28	10	5	3	60	0.28	3167

	Sell	"List"	"Living"	"Rooms"	"Beds"	"Baths"	"Age"	"Acres"	"Taxes"
1	175	180	18	8	4	1	12	0.43	4033
2	129	132	13	6	3	1	41	0.33	1471
3	138	140	17	7	3	1	22	0.46	3204
4	232	240	25	8	4	3	5	2.05	3613
5	135	140	18	7	4	3	9	0.57	3028
6	150	160	20	8	4	3	18	4.00	3131
7	207	225	22	8	4	2	16	2.22	5158
8	271	285	30	10	5	2	30	0.53	5702

In [26]:
```
df.drop(df.tail(5).index,inplace=True)
df
```

Out[26]:

	Sell	"List"	"Living"	"Rooms"	"Beds"	"Baths"	"Age"	"Acres"	"Taxes"
0	142	160	28	10	5	3	60	0.28	3167
1	175	180	18	8	4	1	12	0.43	4033
2	129	132	13	6	3	1	41	0.33	1471
3	138	140	17	7	3	1	22	0.46	3204

In [27]:
```
# Saving data to txt file
df.to_csv('output5.txt', sep=' ')
```

In [28]:
```
# List of features
df.columns.values
```

Out[28]:
```
array(['Sell', ' "List"', ' "Living"', ' "Rooms"', ' "Beds"', ' "Baths"',
       ' "Age"', ' "Acres"', ' "Taxes"'], dtype=object)
```

6

Perform basic operations on matrices (like addition, subtraction, multiplication) and display specific rows or columns of the matrix.

In [29]:
```
import numpy as np
A = np.array([[3, 6, 9], [5, -10, 15], [-7, 14, 21]])
B = np.array([[9, -18, 27], [11, 22, 33], [13, -26, 39]])
print("A = \n", A, "\nB = \n", B)
```

```
A =
 [[  3   6   9]
 [  5 -10  15]
 [ -7  14  21]]
B =
 [[  9 -18  27]
 [ 11  22  33]
 [ 13 -26  39]]
```

In [30]:
```
# Matrix Addition
C = A + B
print('C = A + B =\n', C)
```

```
C = A + B =
 [[ 12 -12  36]
 [ 16  12  48]
 [  6 -12  60]]
```

In [31]:
```
# Matrix Subtraction
C = A - B
print('C = A - B =\n', C)
```

```
C = A - B =
 [[ -6  24 -18]
 [ -6 -32 -18]
 [-20  40 -18]]
```

In [32]:
```
# Matrix Multiplication
C = A.dot(B)
print('C = A * B =\n', C)
```

```
C = A * B =
 [[ 210 -156  630]
 [ 130 -700  390]
 [ 364 -112 1092]]
```

In [33]:
```
# Print 2nd row of Matrix A
print(A[1:2])
```

```
[[  5 -10  15]]
```

In [34]:
```
# Print 1st row of Matrix B
print(B[:1])
```

```
[[  9 -18  27]]
```

In [35]:
```
# Print 2nd column of Matrix A
print(A[:,1:2])
```

```
[[  6]
 [-10]
 [ 14]]
```

In [36]:
```
# Print 3rd column of Matrix B
print(B[:,2:3])
```

```
[[27]
 [33]
 [39]]
```

7

Perform other matrix operations like converting matrix data to absolute values, taking the negative of matrix values, additing/removing rows/columns from a matrix, finding the maximum or minimum values in a matrix or in a

row/column, and finding the sum of some/all elements in a matrix.

In [37]:
```
import numpy as np
A = np.array([[3, 6, 9], [5, -10, 15], [-7, 14, 21]])
B = np.array([[9, -18, 27], [11, 22, 33], [13, -26, 39]])
print("A = \n", A, "\nB = \n", B)
```

```
A =
 [[  3   6   9]
 [  5 -10  15]
 [ -7  14  21]]
B =
 [[  9 -18  27]
 [ 11  22  33]
 [ 13 -26  39]]
```

In [38]:
```
# Converting matrix A data to its absolute values
np.absolute(A)
```

Out[38]:
```
array([[ 3,  6,  9],
       [ 5, 10, 15],
       [ 7, 14, 21]])
```

In [39]:
```
# Converting matrix B data to its negative values
np.negative(B)
```

Out[39]:
```
array([[ -9,  18, -27],
       [-11, -22, -33],
       [-13,  26, -39]])
```

In [40]:
```
# Deleting a row from Matrix A
np.delete(A, 1, 0)
```

Out[40]:
```
array([[ 3,  6,  9],
       [-7, 14, 21]])
```

In [41]:
```
# Deleting a column from Matrix B
np.delete(B, 0, 1)
```

Out[41]:
```
array([[-18,  27],
       [ 22,  33],
       [-26,  39]])
```

In [42]:
```
# Adding a row to Matrix A
np.append(A, np.array([[23, -45, 56]]), axis=0)
```

Out[42]:
```
array([[  3,   6,   9],
       [  5, -10,  15],
       [ -7,  14,  21],
       [ 23, -45,  56]])
```

In [43]:
```
# Adding a column to Matrix B
np.append(B, [[23], [-45], [56]], axis=1)
```

Out[43]:
```
array([[  9, -18,  27,  23],
       [ 11,  22,  33, -45],
       [ 13, -26,  39,  56]])
```

In [44]:
```
# Maximum of 2nd row of Matrix A
np.max(A, 0)[1]
```

Out[44]: 14

In [45]:
```
# Minimum of 2nd row of Matrix A
np.min(A, 0)[1]
```

Out[45]: -10

In [46]:
```
# Maximum of 3rd column of Matrix B
np.max(B, 1)[2]
```

Out[46]: 39

In [47]:
```
# Minimum of 3rd column of Matrix B
np.min(B, 1)[2]
```

Out[47]: -26

In [48]:
```
# Sum of some elements of array
np.sum(A[1:, 1:])
```

Out[48]: 40

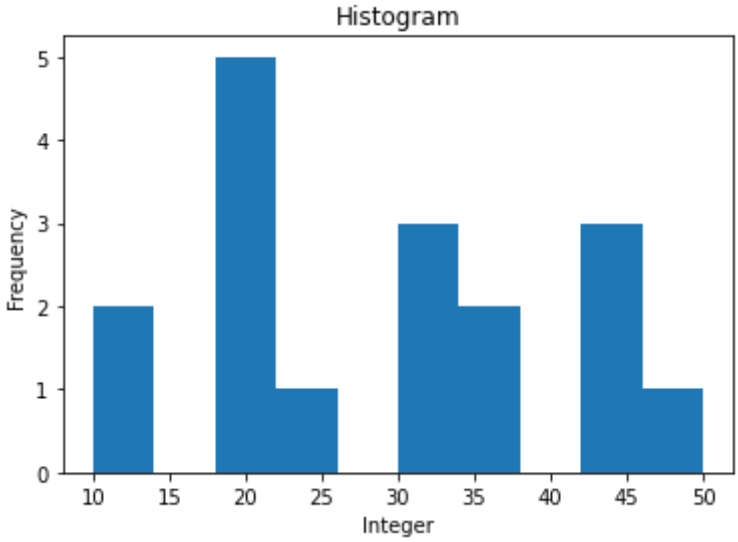
In [49]:
```
# Sum of all elements of array
sumA = np.sum(A)
sumB = np.sum(B)
print('sumA = ', sumA, ', sumB = ', sumB)
```

```
sumA =  56 , sumB =  110
```

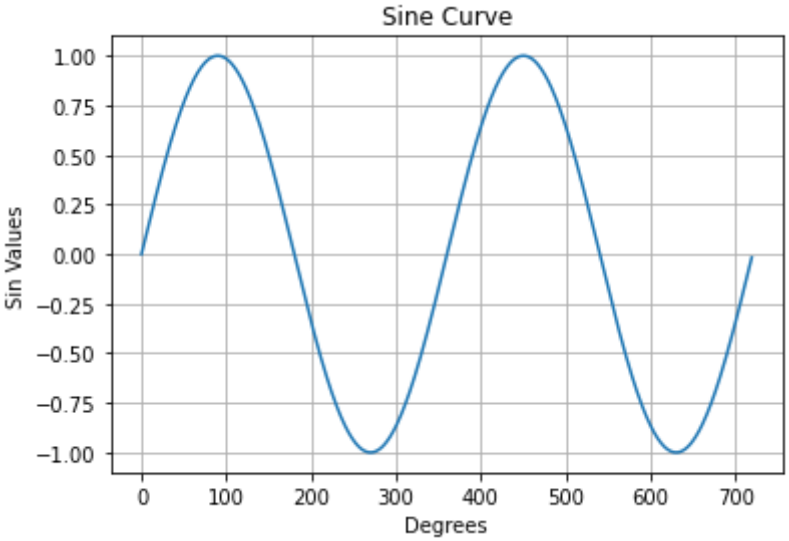
8

Create various type of plots/charts like histograms, plot based on sine/cosine function based on data from a matrix. Further label different axes in a plot and data in a plot.

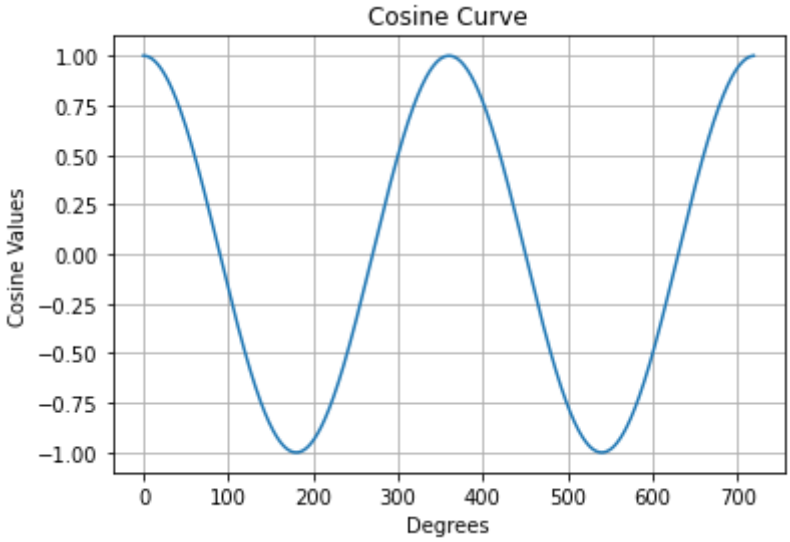
In [50]:
```
import numpy as np
import matplotlib.pyplot as plt

# Histogram
list = np.array([20, 45, 45, 35, 30, 10, 30, 20, 20, 50, 30, 20, 20, 10, 45
plt.hist(list)
plt.xlabel('Integer')
plt.ylabel('Frequency')
plt.title('Histogram')
plt.show()
```



In [51]:
```
import math

# Sine curve
degrees = range(0 , 720)
sinValues = [math.sin(math.radians(i)) for i in degrees]
plt.plot(sinValues)
plt.xlabel('Degrees')
plt.ylabel('Sin Values')
plt.title('Sine Curve')
plt.grid()
plt.show()
```



In [52]:
```
# Cosine curve
degrees = range(0 , 720)
sinValues = [math.cos(math.radians(i)) for i in degrees]
plt.plot(sinValues)
plt.xlabel('Degrees')
plt.ylabel('Cosine Values')
plt.title('Cosine Curve')
plt.grid()
plt.show()
```

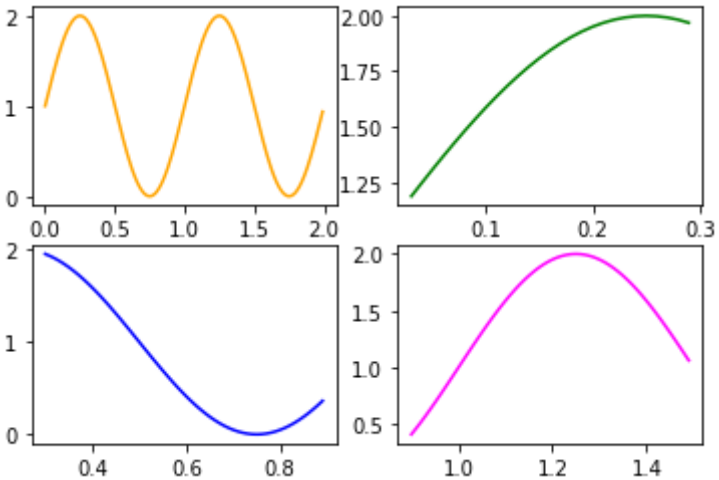


9

Generate different subplots from a given plot and color plot data.

In [53]:
```
import matplotlib.pyplot as plt
import numpy as np

# Data for plotting
x = np.arange(0.0, 2.0, 0.01)
y = 1 + np.sin(2 * np.pi * x)

# Creating 6 subplots and unpacking the output array immediately
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2)
ax1.plot(x, y, color="orange")
ax2.plot(x[3:30], y[3:30], color="green")
ax3.plot(x[30:90], y[30:90], color="blue")
ax4.plot(x[90:150], y[90:150], color="magenta")
```

Out[53]:
```
[<matplotlib.lines.Line2D at 0x1d7944a4ca0>]
```



10

Use conditional statements and different type of loops based on simple example/s.

In [54]:
```
#if - elif - else
grade = None
marks = 90
if marks >= 95:
    grade = 'A+'
elif marks >= 90:
    grade = 'A'
elif marks >= 80:
    grade = 'B'
elif marks >= 75:
    grade = 'C'
elif marks >= 65:
    grade = 'D'
else:
    grade = 'F'
grade
```

Out[54]:
```
'A'
```

In [55]:
```
# while loop
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

```
1
2
3
```

In [56]:
```
# for loop
fruits = ["apple", "cherry", "banana"]
for x in fruits:
  print(x)
  if x == "banana":
    break
```

```
apple
cherry
banana
```

11

Perform vectorized implementation of simple matrix operation like finding the transpose of a matrix, adding, subtracting or multiplying two matrices.

In [57]:
```
import numpy as np
A = np.array([[3, 6, 9], [5, -10, 15], [-7, 14, 21]])
B = np.array([[9, -18, 27], [11, 22, 33], [13, -26, 39]])
print("A = \n", A, "\nB = \n", B)
```

```
A =
 [[  3   6   9]
 [  5 -10  15]
 [ -7  14  21]]
B =
 [[  9 -18  27]
 [ 11  22  33]
 [ 13 -26  39]]
```

In [58]:
```
# Addition
A + B
```

Out[58]:
```
array([[ 12, -12,  36],
       [ 16,  12,  48],
       [  6, -12,  60]])
```

In [59]:
```
# Subtraction
A - B
```

Out[59]:
```
array([[ -6,  24, -18],
       [ -6, -32, -18],
       [-20,  40, -18]])
```

In [60]:
```
# Multiplication
A @ B
```

Out[60]:
```
array([[ 210, -156,  630],
       [ 130, -700,  390],
       [ 364, -112, 1092]])
```

In [61]:
```
# Transpose
print("A' = \n", np.transpose(A),
      "\nB' = \n", np.transpose(B))
```

```
A' =
 [[  3   5  -7]
 [  6 -10  14]
 [  9  15  21]]
B' =
 [[  9  11  13]
 [-18  22 -26]
 [ 27  33  39]]
```

12.

Implement Linear Regression problem. For example, based on a dataset comprising of existing set of prices and area/size of the houses, predict the estimated price of a given house.

In [1]:
```
#Required imports
import pandas as pd
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt
```

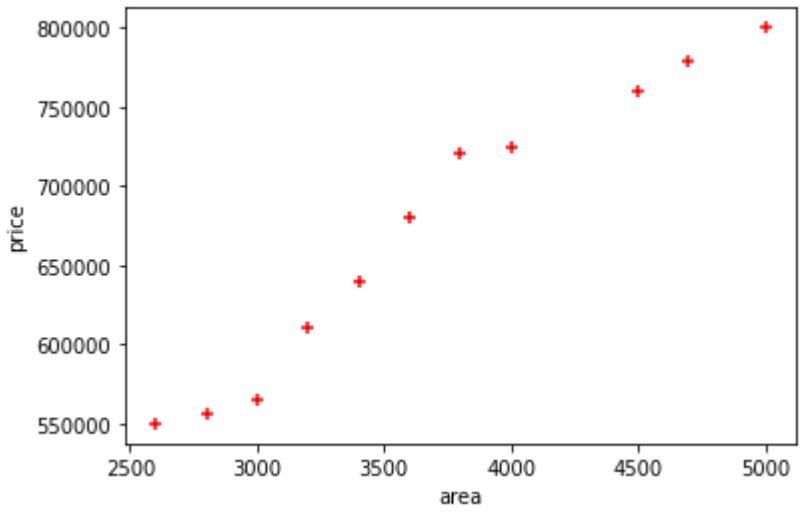
In [2]:
```
# Reading csv file to dataframe
df = pd.read_csv('houseprices.csv')
df.head()
```

Out[2]:

	area	price
0	2600	550000
1	2800	556000
2	3000	565000
3	3200	610000
4	3400	640000

In [3]:
```
# Scatter plot for the dataset
%matplotlib inline
plt.xlabel('area')
plt.ylabel('price')
plt.scatter(df.area,df.price,color='red',marker='+')
```

Out[3]:
```
<matplotlib.collections.PathCollection at 0x1c0d22ae3d0>
```



Preparing data for training

In [4]:
```
x_df = df.drop('price',axis='columns')
x_df.head()
```

Out[4]:

	area
0	2600
1	2800
2	3000
3	3200
4	3400

In [5]:
```
price = df.price
price
```

Out[5]:
```
0     550000
1     556000
2     565000
3     610000
4     640000
5     680000
6     720000
7     725000
8     760000
9     779000
10    800000
Name: price, dtype: int64
```

Applying Linear Regression

In [6]:
```
# Create linear regression object
reg = linear_model.LinearRegression()
reg.fit(x_df,price)
```

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

In [7]:
```
m = reg.coef_
c = reg.intercept_
print('Coefficient, m = ', m)
print('Intercept, c = ', c)
```

```
Coefficient, m =  [114.12402428]
Intercept, c =  250142.23764093674
```

Predictions

In [8]:
```
ans1 = reg.predict([[3300]])
print('(1) Price of a house with area = 3300 sqr ft: ', ans1)
```

```
(1) Price of a house with area = 3300 sqr ft:  [626751.51777971]
```

In [9]:
```
y = m*3300 + c
print('y = m*x + c =', y)
```

```
y = m*x + c = [626751.51777971]
```

Here, we can see that `y = ans1 = 626751.51777971`

Another prediction

In [10]:
```
ans2 = reg.predict([[6000]])
print('(2) Price of a house with area = 6000 sqr ft: ', ans2)
```

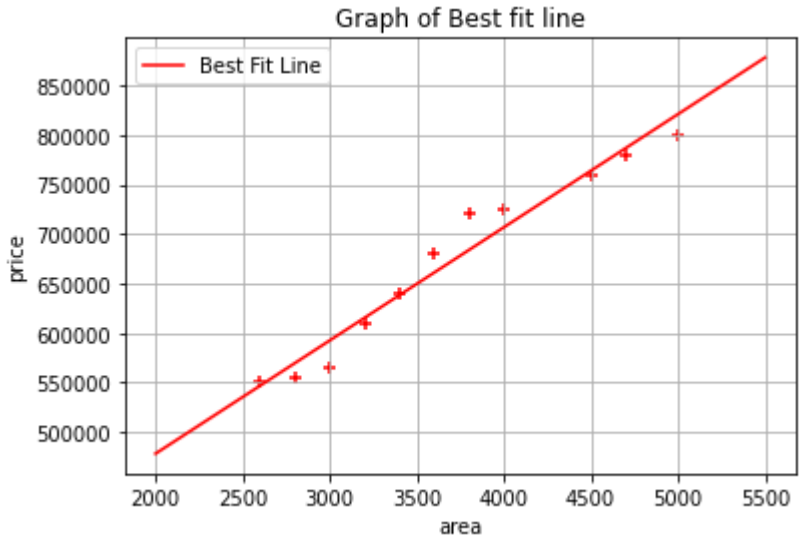
```
(2) Price of a house with area = 6000 sqr ft:  [934886.38334779]
```

In [11]:
```
y = m*6000 + c
print('y = m*x + c =', y)
```

```
y = m*x + c = [934886.38334779]
```

Visualising Best Fit Line

In [12]:
```
x = np.linspace(2000,5500)
y = m*x+c
plt.plot(x, y, '-r', label='Best Fit Line')
plt.legend(loc='upper left')
plt.title('Graph of Best fit line')
plt.xlabel('area')
plt.ylabel('price')
plt.scatter(df.area,df.price,color='red',marker='+')
plt.grid()
plt.show()
```



13.

Based on multiple features/variables perform Linear Regression. For example, based on a number of additional features like number of bedrooms, servant room, number of balconies,

number of houses of years a house has been built – predict the price of a house.

In [13]:
```
#Required imports
import pandas as pd
import numpy as np
from sklearn import linear_model
```

In [14]:
```
# Reading csv file to dataframe
df = pd.read_csv('houseprices2.csv')
df
```

Out[14]:

	area	bedrooms	age	price
0	2600	3.0	20	550000
1	3000	4.0	15	565000
2	3200	NaN	18	610000
3	3600	3.0	30	595000
4	4000	5.0	8	760000
5	4100	6.0	8	810000

Data Preprocessing: Fill NA values with median value of a column

In [15]:
```
df.bedrooms.median()
```

Out[15]:
```
4.0
```

In [16]:
```
df.bedrooms = df.bedrooms.fillna(df.bedrooms.median())
df
```

Out[16]:

	area	bedrooms	age	price
0	2600	3.0	20	550000
1	3000	4.0	15	565000
2	3200	4.0	18	610000
3	3600	3.0	30	595000
4	4000	5.0	8	760000
5	4100	6.0	8	810000

Applying Linear Regression

In [17]:
```
reg = linear_model.LinearRegression()
reg.fit(df.drop('price',axis='columns'), df.price)
```

Out[17]:
```
LinearRegression()
```

In [18]:
```
m1, m2, m3 = reg.coef_
c = reg.intercept_
print('Coefficients, \
\n\tm1 = {}, \
\n\tm2 = {}, \
\n\tm3 = {}'.format(m1, m2, m3))
print('Intercept, c = ', c)
```

```
Coefficients,
        m1 = 112.06244194213456,
        m2 = 23388.880077939153,
        m3 = -3231.717908632967
Intercept, c =  221323.00186540443
```

Predictions

In [19]:
```
ans1 = reg.predict([[3000, 3, 40]])
print('(1) Price of home with 3000 sqr ft area, 3 bedrooms, 40 year old: ', ar
```

```
(1) Price of home with 3000 sqr ft area, 3 bedrooms, 40 year old:  [498408.251
58031]
```

In [20]:
```
 y1 = m1*3000 + m2*3 + m3*40 + c
print('\ty1 = m1*x1 + m2*x2 + m3*x3 + c =\n\t', y1)
```

```
        y1 = m1*x1 + m2*x2 + m3*x3 + c =
         498408.2515803069
```

In [21]:
```
reg.predict([[2500, 5, 10]])
print('(2) Price of home with 2500 sqr ft area, 5 bedrooms, 10 year old: ', ar
```

```
(2) Price of home with 2500 sqr ft area, 5 bedrooms, 10 year old:  [934886.383
34779]
```

In [22]:
```
 y1 = m1*2500 + m2*5 + m3*10 + c
print('\ty1 = m1*x1 + m2*x2 + m3*x3 + c =\n\t', y1)
```

```
        y1 = m1*x1 + m2*x2 + m3*x3 + c =
         586106.3280241069
```

14.

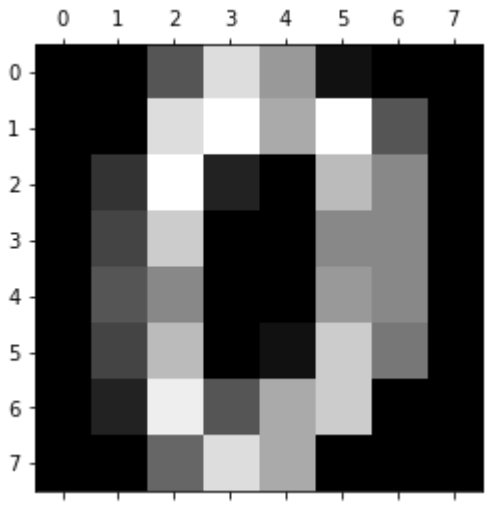
Implement a classification/ logistic regression problem. For example based on different features of students data, classify, whether a student is suitable for a particular activity. Based on the available dataset, a student can also implement another classification problem like checking whether an email is spam or not.

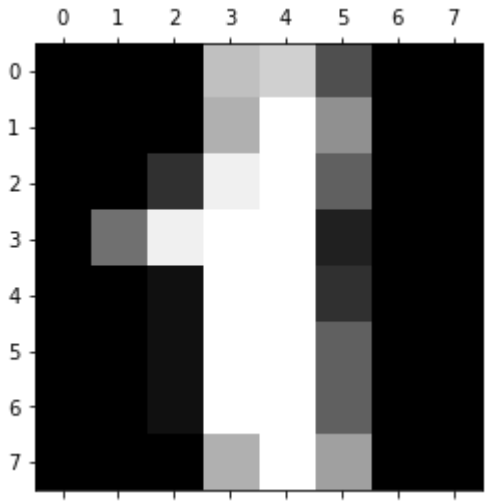
In [23]:
```
# Import and load digits dataset
from sklearn.datasets import load_digits
digits = load_digits()

# Import matplotlib
```

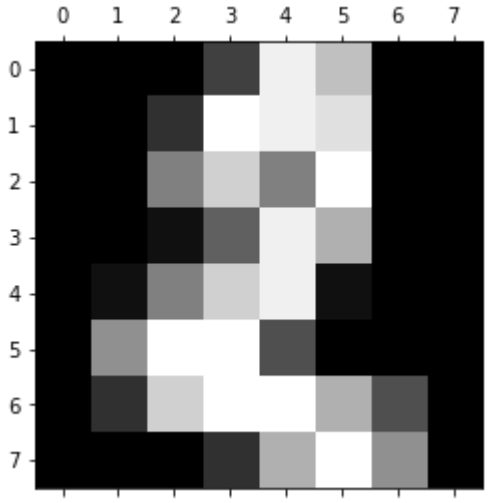
```
%matplotlib inline
import matplotlib.pyplot as plt
```

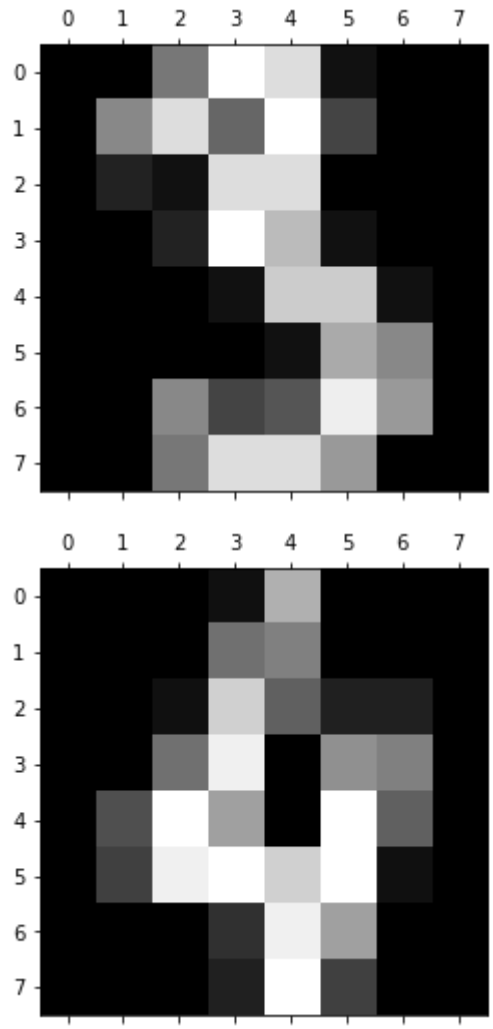
In [24]:
```
# Plot 2D matrix data of digits
plt.gray()
for i in range(5):
    plt.matshow(digits.images[i])
```

```
<Figure size 432x288 with 0 Axes>
```









In [25]:
```
# Get the attributes/columns of digits dataset
dir(digits)
```

Out[25]:
```
['DESCR', 'data', 'feature_names', 'frame', 'images', 'target', 'target_name
s']
```

Creating and training the logistic regression model

In [26]:
```
# Import the model
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(max_iter=3000)
```

In [27]:
```
# Import train_test_split
from sklearn.model_selection import train_test_split

# Split the dataset into training and testing datasets
X_train, X_test, y_train, y_test = train_test_split(digits.data,digits.target,
```

In [28]:
```
print(len(X_train), len(X_test), len(y_train), len(y_test))
```

```
1437 360 1437 360
```

In [29]:
```
# Training the model
model.fit(X_train, y_train)
```

Out[29]:
```
LogisticRegression(max_iter=3000)
```

Measuring accuracy of our model

In [30]:
```
model.score(X_test, y_test)
```

Out[30]:
```
0.9611111111111111
```

Predictions

In [31]:
```
model.predict(digits.data[0:5])
```

Out[31]:
```
array([0, 1, 2, 3, 4])
```

In [32]:
```
y_predicted = model.predict(X_test)
y_predicted
```

Out[32]:
```
array([0, 1, 1, 4, 3, 4, 6, 8, 6, 2, 1, 6, 8, 6, 7, 2, 7, 2, 1, 3, 5, 4,
       3, 9, 4, 9, 6, 7, 7, 7, 1, 6, 8, 5, 6, 4, 4, 1, 4, 4, 4, 2, 9, 3,
       6, 9, 1, 7, 8, 3, 6, 9, 8, 0, 9, 1, 2, 7, 8, 9, 8, 8, 9, 1, 3, 8,
       8, 5, 7, 6, 3, 6, 7, 4, 6, 7, 6, 6, 8, 8, 0, 5, 4, 9, 7, 9, 0, 1,
       8, 7, 2, 1, 6, 3, 8, 0, 2, 1, 1, 5, 0, 0, 7, 3, 5, 1, 5, 6, 0, 5,
       5, 5, 2, 0, 0, 7, 3, 0, 7, 4, 5, 9, 0, 6, 5, 9, 1, 7, 8, 9, 3, 8,
       4, 3, 9, 0, 0, 0, 9, 0, 4, 0, 7, 5, 3, 0, 7, 1, 1, 9, 3, 0, 5, 5,
       3, 7, 6, 8, 7, 9, 8, 7, 6, 5, 9, 4, 8, 2, 6, 2, 9, 3, 0, 4, 6, 9,
       9, 1, 2, 0, 2, 0, 1, 4, 0, 4, 1, 6, 1, 3, 5, 1, 9, 0, 3, 3, 9, 2,
       0, 1, 2, 5, 1, 4, 9, 9, 2, 7, 2, 6, 0, 9, 0, 4, 4, 3, 7, 4, 5, 0,
       4, 4, 7, 0, 9, 3, 1, 4, 3, 6, 5, 7, 2, 3, 5, 7, 2, 9, 7, 4, 2, 1,
       1, 6, 4, 3, 8, 6, 1, 8, 2, 5, 8, 7, 5, 5, 0, 5, 8, 9, 7, 3, 6, 0,
       8, 1, 3, 5, 7, 0, 8, 7, 9, 1, 3, 9, 9, 5, 3, 9, 2, 1, 7, 9, 5, 6,
       3, 3, 0, 7, 9, 5, 8, 4, 5, 8, 1, 8, 1, 9, 4, 9, 3, 7, 7, 3, 7, 9,
       4, 2, 2, 7, 0, 9, 2, 3, 9, 0, 0, 1, 9, 5, 9, 3, 3, 5, 4, 2, 1, 2,
       2, 4, 1, 3, 8, 4, 7, 8, 7, 9, 3, 0, 4, 4, 4, 5, 9, 1, 3, 2, 6, 6,
       9, 0, 0, 6, 2, 7, 3, 5])
```

Confusion matrix

In [33]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_predicted)
cm
```

Out[33]:
```
array([[39,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0, 33,  0,  0,  0,  0,  1,  0,  1,  1],
       [ 0,  1, 29,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0, 38,  0,  1,  0,  0,  0,  0],
       [ 0,  1,  0,  0, 36,  0,  0,  1,  0,  0],
       [ 0,  0,  0,  0,  0, 32,  0,  0,  0,  0],
       [ 0,  1,  0,  0,  0,  0, 30,  0,  0,  0],
       [ 0,  0,  0,  1,  0,  0,  0, 38,  0,  2],
       [ 0,  1,  0,  0,  0,  0,  0,  0, 29,  0],
       [ 0,  0,  0,  0,  0,  1,  0,  1,  0, 42]], dtype=int64)
```

In [34]:
```
import seaborn as sn
plt.figure(figsize = (10,7))
sn.heatmap(cm, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

Out[34]: Text(69.0, 0.5, 'Truth')