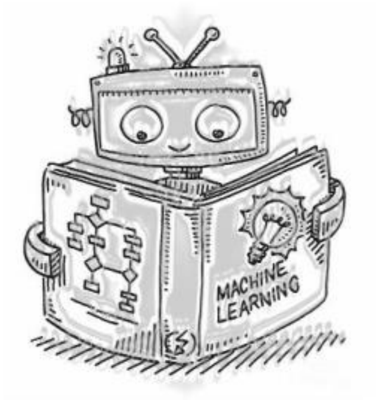
RAM LAL ANAND COLLEGE

UNIVERSITY OF DELHI

January to May 2023



Machine Learning

Practical File

Paper Code :- 32347607

Submitted By –

Harsh Jaiswal

Roll No. – 20058570011

B.Sc. (Hons.) Computer Science 6th semester

Submitted To –

Mr. Arun Gautam Sir

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 [6]:
```
# Exponentiation
5 ** 7
```

Out[6]: 78125

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

In [7]:
```
# OR
True | False
```

Out[7]: True

In [8]:
```
# AND
True & False
```

Out[8]: False

In [9]:
```
# Check for inequality
```

```
True == False
```

Out[9]: False

In [10]:
```
# NOT
~True
```

Out[10]: -2

In [11]:
```
# 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 [12]:
```
# Creating and initializing a variable
a = 5
x1 = 10
x2 = 20
name = "Harsh"
```

In [13]:
```
# Displaying variables
print(a)
print(x1)
print(x2)
print(name)
```

```
5
10
20
Harsh
```

4 Create/Define single dimension / multidimension 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 [14]:
```
import numpy as np
```

In [17]:
```
# Creating single-dimension arrays
x = np.array([1, 2, 3, 4, 5])
```

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

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

In [18]:
```python
# 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 [19]:
```python
# Matrix with all ones
A = np.ones((4, 4))
A
```

Out[19]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

In [20]:
```python
# Matrix with all zeros
B = np.zeros((4, 4))
B
```

Out[20]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

In [21]:
```python
# 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 = 
 [[39 32 38 36 24]
 [47 28 43 49 29]
 [37 23 31 43 22]
 [31 24 27 24 49]]
C1 = 
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

In [22]:
```
# 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 [23]:
```
import numpy
A = np.arange(12).reshape((3, 4))
A
```

Out[23]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [24]:
```
# Size of matrix
np.size(A)
```

Out[24]: 12

In [25]:
```
# Shape of matrix
A.shape
```

Out[25]: (3, 4)

In [26]:
```
# Length of 1st row
len(A[0])
```

Out[26]: 4

In [27]:
```
# Length of first column
np.size(A, 0)
```

Out[27]: 3

In [28]:
```
# Loading data from text file
import pandas as pd
df = pd.read_csv('/content/input.txt')
df
```

Out[28]:

	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
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 [29]:

```
df.drop(df.tail(5).index,inplace=True)
df
```

Out[29]:

	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 [30]:

```
# Saving data to txt file
df.to_csv('output.txt', sep=' ')
```

In [31]:

```
# List of features
df.columns.values
```

Out[31]:

```
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 [32]:

```
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 [33]:
```
# Matrix Addition
C = A + B
print('C = A + B =\n', C)
```

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

In [34]:
```
# Matrix Subtraction
C = A - B
print('C = A - B =\n', C)
```

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

In [35]:
```
# 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 [36]:
```
# Print 2nd row of Matrix A
print(A[1:2])
```

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

In [37]:
```
# Print 1st row of Matrix B
print(B[:1])
```

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

In [38]:
```
# Print 2nd column of Matrix A
print(A[:,1:2])
```

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

In [39]:
```
# 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 [40]:
```
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 [41]:
```
# Converting matrix A data to its absolute values
np.absolute(A)
```

Out[41]:
```
array([[ 3,  6,  9],
       [ 5, 10, 15],
       [ 7, 14, 21]])
```

In [42]:
```
# Converting matrix B data to its negative values
np.negative(B)
```

Out[42]:
```
array([[ -9,  18, -27],
       [-11, -22, -33],
       [-13,  26, -39]])
```

In [43]:
```
# Deleting a row from Matrix A
np.delete(A, 1, 0)
```

Out[43]:
```
array([[ 3,  6,  9],
       [-7, 14, 21]])
```

In [44]:
```
# Deleting a column from Matrix B
np.delete(B, 0, 1)
```

Out[44]:
```
array([[-18,  27],
       [ 22,  33],
       [-26,  39]])
```

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

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

In [46]:
```python
# Adding a column to Matrix B
np.append(B, [[23], [-45], [56]], axis=1)
```

Out[46]:
```
array([[  9, -18,  27,  23],
       [ 11,  22,  33, -45],
       [ 13, -26,  39,  56]])
```

In [47]:
```python
# Maximum of 2nd row of Matrix A
np.max(A, 0)[1]
```

Out[47]:
```
14
```

In [48]:
```python
# Minimum of 2nd row of Matrix A
np.min(A, 0)[1]
```

Out[48]:
```
-10
```

In [49]:
```python
# Maximum of 3rd column of Matrix B
np.max(B, 1)[2]
```

Out[49]:
```
39
```

In [50]:
```python
# Minimum of 3rd column of Matrix B
np.min(B, 1)[2]
```

Out[50]:
```
-26
```

In [51]:
```python
# Sum of some elements of array
np.sum(A[1:, 1:])
```

Out[51]:
```
40
```

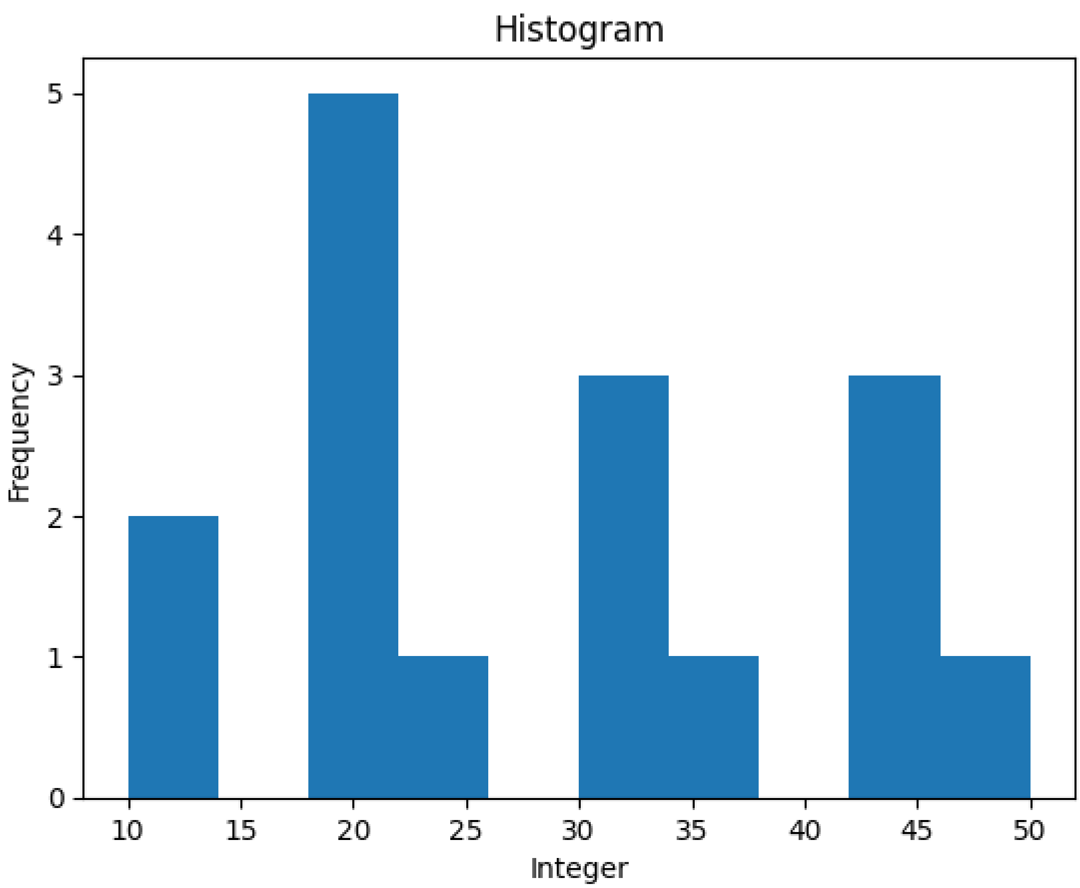
In [52]:
```python
# Sum of all elements of array
sumA = np.sum(A)
sumB = np.sum(B)
print('sumA = ', sumA, ', sumB = ', sumB)
```

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

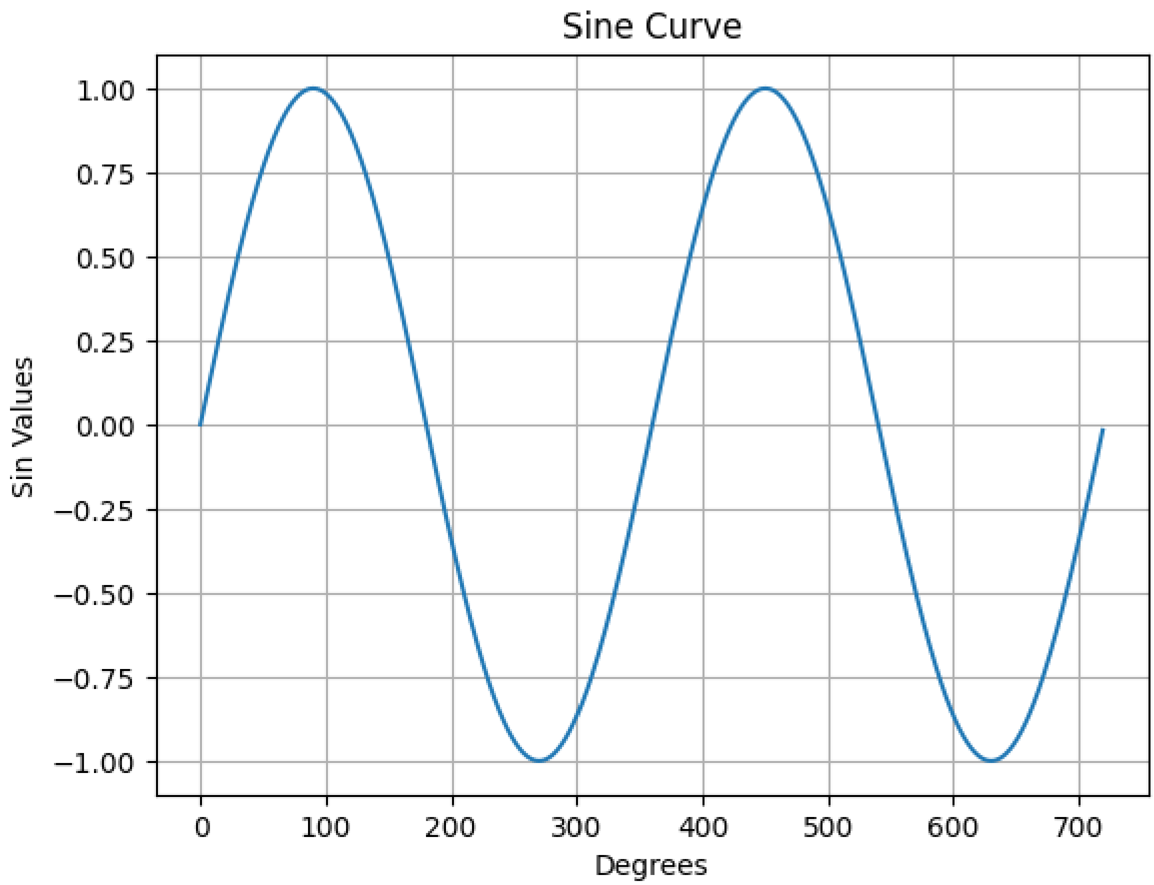
8 Create various type of plots/charts like histograms, plot based on sine/cosine function based on data from a matrix.

Further label different axes in a plot and data in a plot.

In [55]:
```
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, 45,25])
plt.hist(list)
plt.xlabel('Integer')
plt.ylabel('Frequency')
plt.title('Histogram')
plt.show()
```

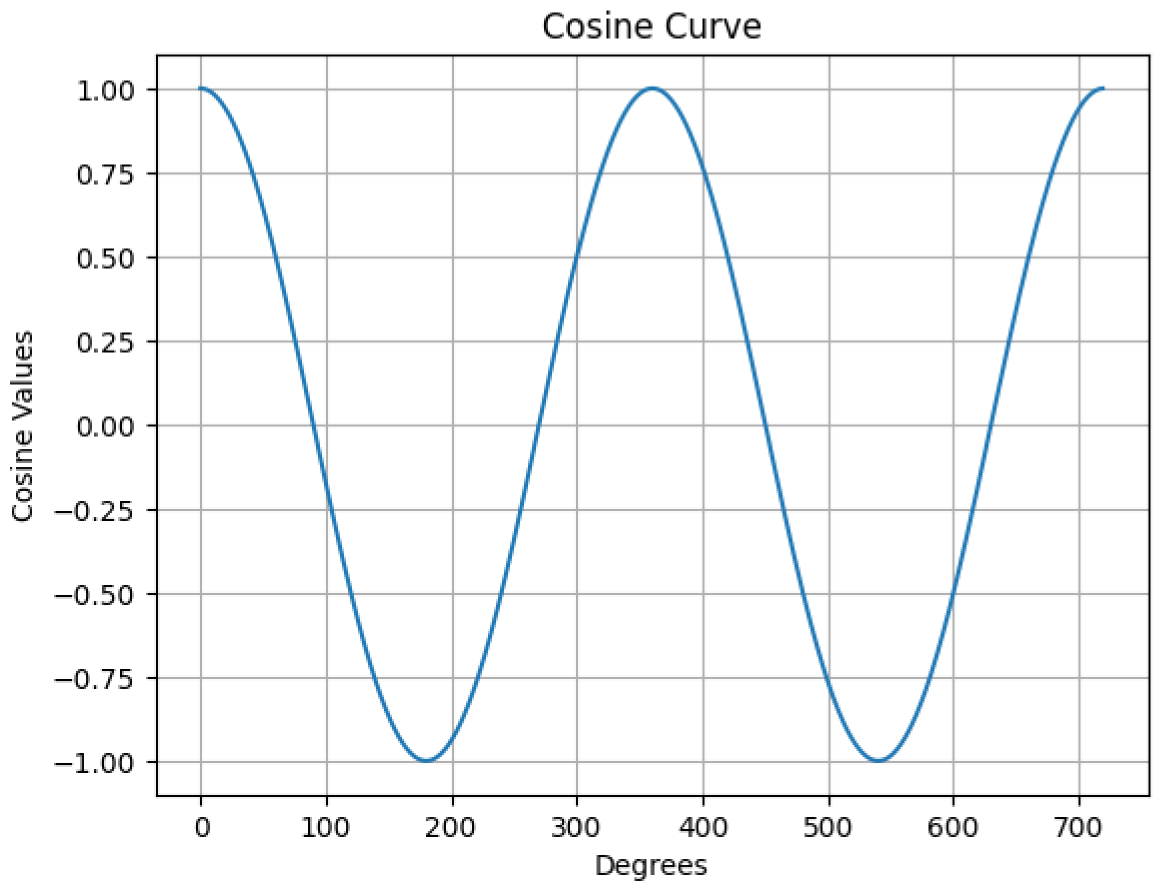


In [56]:
```
import math
# Sine curve
degrees = range(0 , 720)
sinValues = [math.sin(math.radians(i)) for i in degrees]
plt.plot(sinValues)
plt.xlabel('Degrees')
plt.ylabel('Sin Values')
plt.title('Sine Curve')
plt.grid()
plt.show()
```



In [57]:

```
# 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()
```



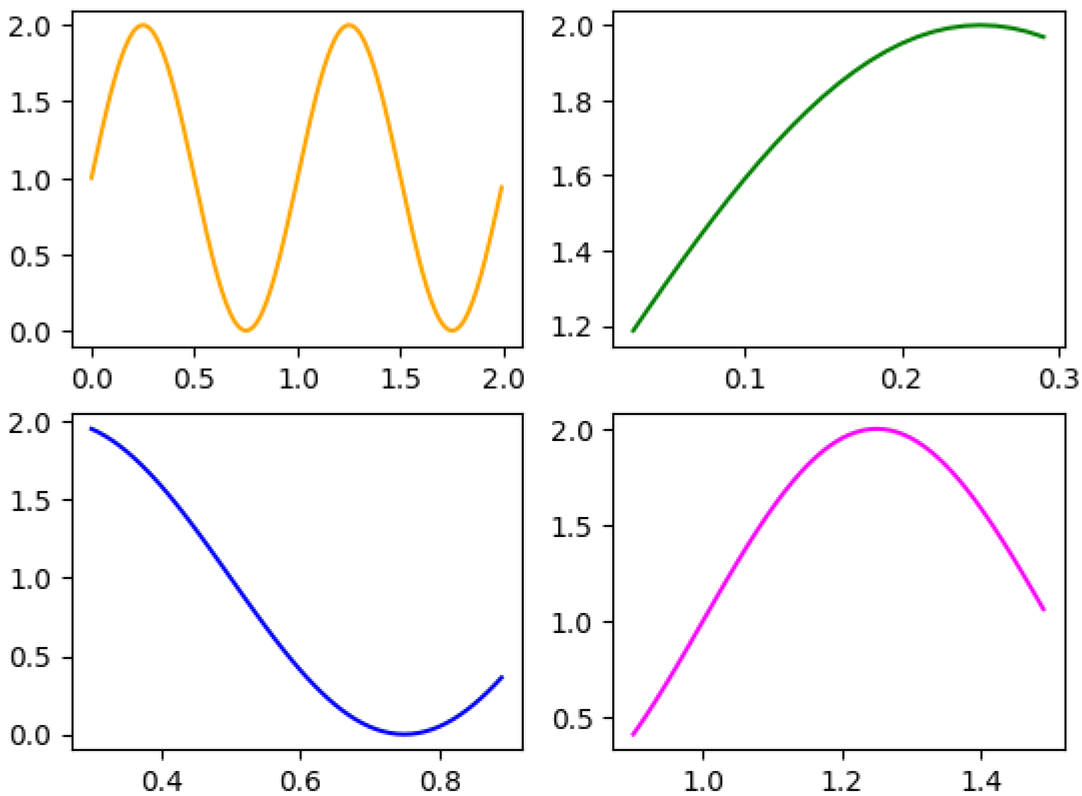
9 Generate different subplots from a given plot and color plot data.

In [58]:

```
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[58]:

```
[<matplotlib.lines.Line2D at 0x7f22241287f0>]
```



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

In [59]:
```
#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[59]: 'A'

In [61]:
```
# while loop
i = 1
while i < 6:
 print(i)
 if i == 3:
```

```
        break
    i += 1
```

```
1
2
3
```

In [62]:
```
# 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 [63]:
```
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 [64]:
```
# Addition
A + B
```

Out[64]:
```
array([[ 12, -12,  36],
       [ 16,  12,  48],
       [  6, -12,  60]])
```

In [65]:
```
# Subtraction
A - B
```

Out[65]:
```
array([[ -6,  24, -18],
       [ -6, -32, -18],
       [-20,  40, -18]])
```

In [66]:
```
# Multiplication
A @ B
```

Out[66]:
```
array([[ 210, -156,  630],
       [ 130, -700,  390],
       [ 364, -112, 1092]])
```

In [67]:
```
# 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 [124…
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [125…
```
dataset = pd.read_csv('/content/houseprices.csv')
dataset.head()
```

Out[125]:

	Area	Price
0	372.504664	17648.708613
1	161.218544	7606.327793
2	844.815263	42227.733081
3	550.770094	27571.592292
4	499.007442	24372.488520

In [126…
```
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

In [127…
```
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_s
```

In [128…
```
print(len(X_train),len(y_train),  len(X_test), len(y_test))
```

```
33 33 17 17
```

In [129…
```
class LinearRegression:
    def __init__(self):
        self.m = None
        self.b = None

    def fit(self,X_train,y_train):

        num = 0
        den = 0

        for i in range(X_train.shape[0]):

            num = num + ((X_train[i] - X_train.mean())*(y_train[i] - y_train.mean()
            den = den + ((X_train[i] - X_train.mean())*(X_train[i] - X_train.mean()
```

```
        self.m = num/den
        self.b = y_train.mean() - (self.m * X_train.mean())
        print('m is ',self.m)
        print('b is ',self.b)

    def predict(self,X_test):

        return self.m * X_test + self.b
```

In [130…

```
regressor = LinearRegression()
```

In [131…

```
regressor.fit(X_train,y_train)
```

```
m is  [49.97975739]
b is  [-12.9740026]
```

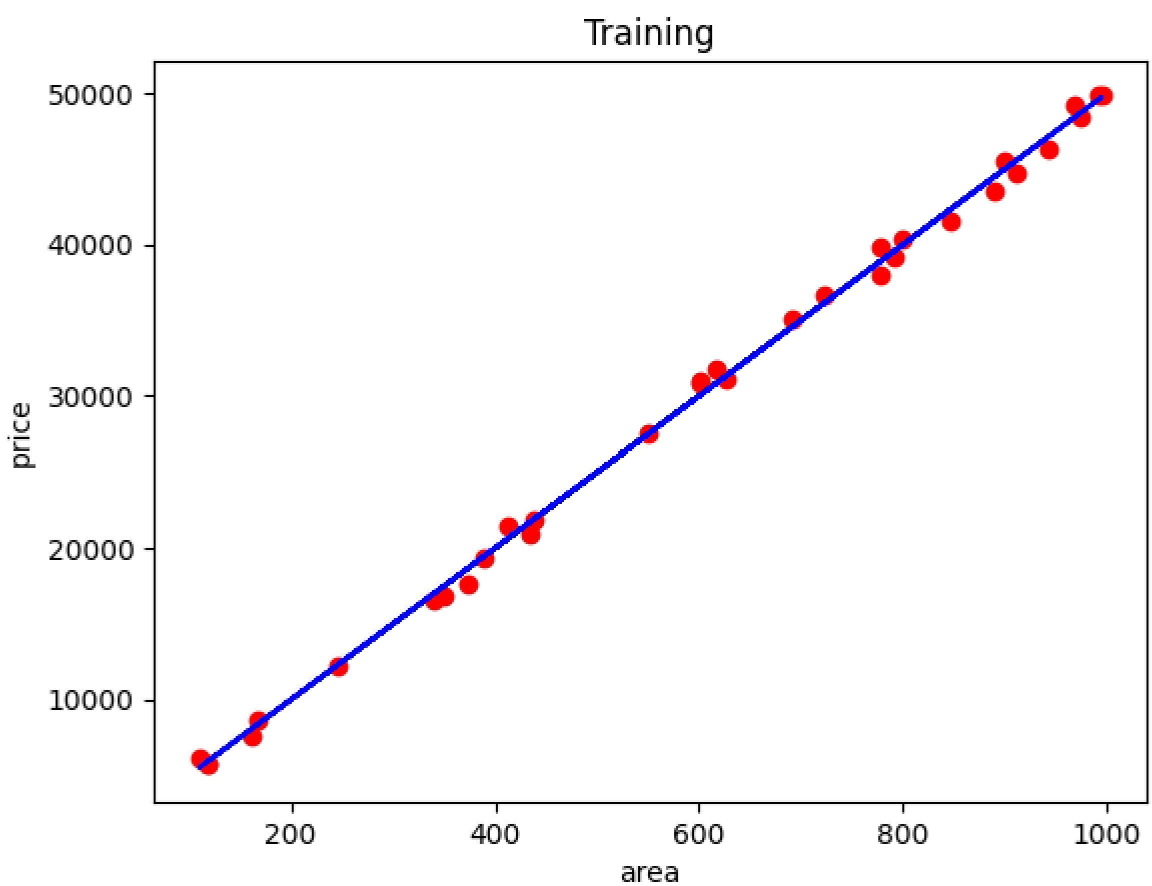
In [132…

```
y_pred = regressor.predict(X_test)
```

In [133…

```
plt.scatter(X_train, y_train, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color = 'blue')
plt.title('Training')
plt.xlabel('area')
plt.ylabel('price')
```

Out[133]: Text(0, 0.5, 'price')

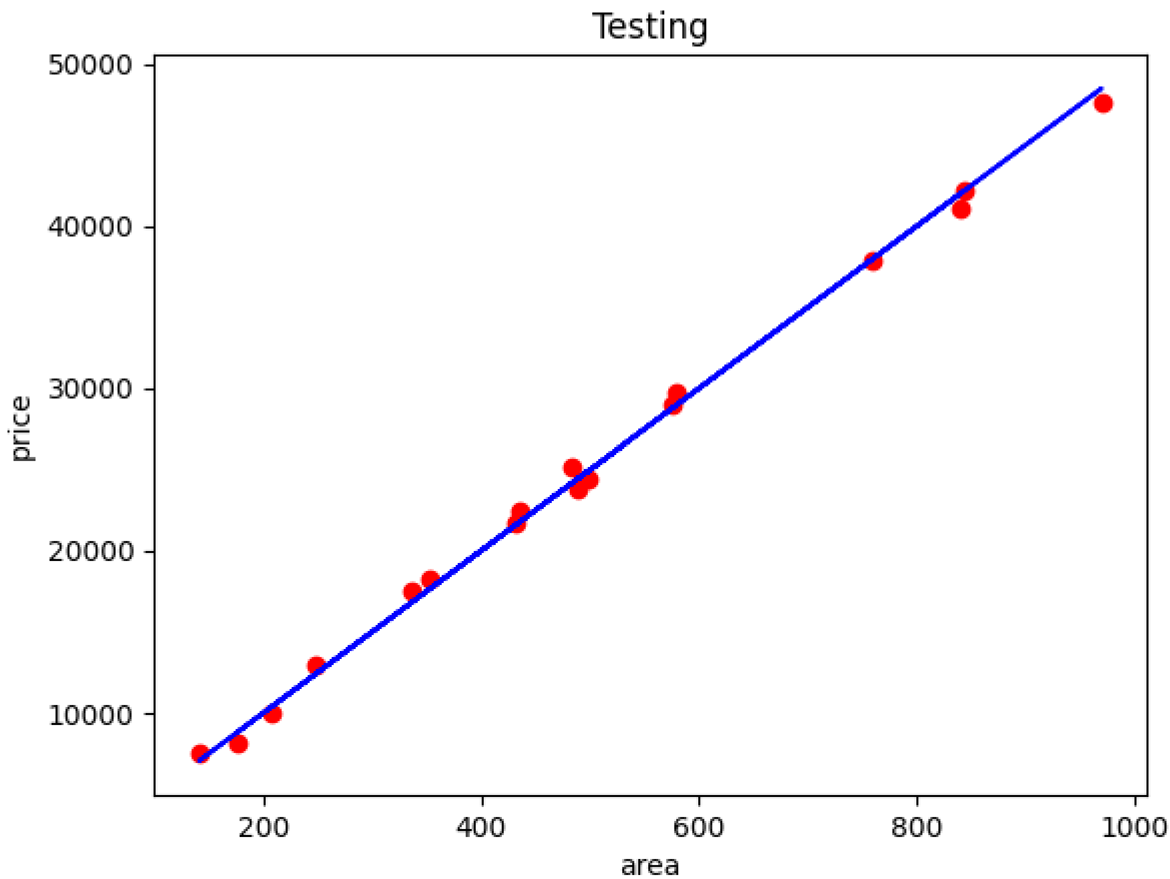


In [134…

```
plt.scatter(X_test, y_test, color = 'red')
```

```
plt.plot(X_test, y_pred, color = 'blue')
plt.title('Testing')
plt.xlabel('area')
plt.ylabel('price')
```

Out[134]: Text(0, 0.5, 'price')



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 [249…
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [250…
```
df = pd.read_csv('/content/homeprice.csv')
X = df.iloc[:, :-1].values
```

```
y = df.iloc[:, -1].values
df.head()
```

Out[250]:

	Area	Bedrooms	Age	Price
0	428.635645	5	15	21625.615922
1	755.488839	2	1	37095.019710
2	662.634921	3	3	33020.292816
3	199.079204	2	20	8888.712606
4	838.612265	5	2	42581.546034

In [251…

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [252…

```
print(len(X_train),len(y_train),  len(X_test), len(y_test))
```

```
40 40 10 10
```

In [253…

```
class MultipleLayerFeatureLinearRegression:
    def __init__(self):
        self.coef_ = None
        self.intercept_ = None

    def fit(self,X_train,y_train):
        X_train = np.insert(X_train,0,1,axis=1)

        # calcuate the coeffs
        betas = np.linalg.inv(np.dot(X_train.T,X_train)).dot(X_train.T).dot(y_trai
        self.intercept_ = betas[0]
        self.coef_ = betas[1:]
        print('Intercept is ',self.intercept_,' Cofficient are ',self.coef_)

    def predict(self,X_test):
        y_pred = np.dot(X_test,self.coef_) + self.intercept_
        return y_pred
```

In [254…

```
model = MultipleLayerFeatureLinearRegression()
model.fit(X_train, y_train)
```

```
Intercept is  48.33983123107282  Cofficient are  [ 50.03920782 139.40047365 -72.8514
4255]
```

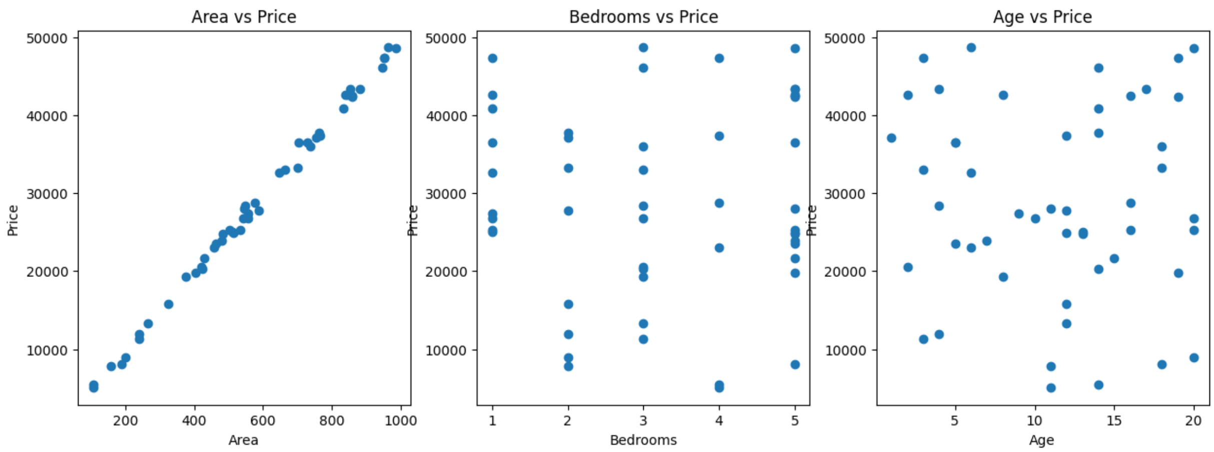
In [255…

```
y_pred = model.predict(X_test)
```

In [256…

```
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15, 5))

ax1.scatter(df['Area'], df['Price'])
ax1.set_xlabel('Area')
ax1.set_ylabel('Price')
ax1.set_title('Area vs Price')

ax2.scatter(df['Bedrooms'], df['Price'])
```

```
ax2.set_xlabel('Bedrooms')
ax2.set_ylabel('Price')
ax2.set_title('Bedrooms vs Price')

ax3.scatter(df['Age'], df['Price'])
ax3.set_xlabel('Age')
ax3.set_ylabel('Price')
ax3.set_title('Age vs Price')

plt.show()
```



In [257…

```
plt.scatter(y_test, y_pred)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=3)
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.title('Predicted vs Actual Price on Test Set')

# Display the plot
plt.show()
```

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 [159…

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [285…

```
data = pd.read_csv('/content/student.csv')
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values
X = np.c_[np.ones((X.shape[0], 1)), X]
y = y[:, np.newaxis]
data.head()
```

Out[285]:

	Hours_studied	Hours_slept	Result
0	7.329712	7.848625	0
1	14.649273	10.545618	1
2	3.431501	6.127123	0
3	5.888299	6.204252	0
4	3.680169	6.072624	0

In [286…

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
print(len(X_train),len(y_train),  len(X_test), len(y_test))
```

```
40 40 10 10
```

In [267…

```
def sigmoid(x):
    return 1/(1+np.exp(-x))
```

In [279…

```
def cost_function(X, y, theta):

    m = len(y)

    # hypothesis
    h = sigmoid(X.dot(theta))

    # cost
    J = (1 / m) * np.sum((-y * np.log(h)) - ((1 - y) * np.log(1 - h)))

    return J
```

In [281…

```
def gradient_descent(X, y, theta, alpha, iterations):

    m = len(y)

    # cost history
    J_history = np.zeros((iterations, 1))

    for i in range(iterations):

        # hypothesis
        h = sigmoid(X.dot(theta))

        # gradient
        theta = theta - (alpha / m) * (X.T.dot(h - y))

        # cost
        J_history[i] = cost_function(X, y, theta)

    return (J_history, theta)
```

In [280…

```
def predict(X, theta):
```

```
        # hypothesis
        h = sigmoid(X.dot(theta))

        # convert probabilities to 0 or 1
        h[h >= 0.5] = 1
        h[h < 0.5] = 0

        return h
```

In [287…

```
alpha = 0.01
theta = np.zeros((X_train.shape[1], 1))
iterations = 10000
print('Initial cost is: ', cost_function(X_train, y_train, theta))
```

```
Initial cost is:  0.6931471805599454
```

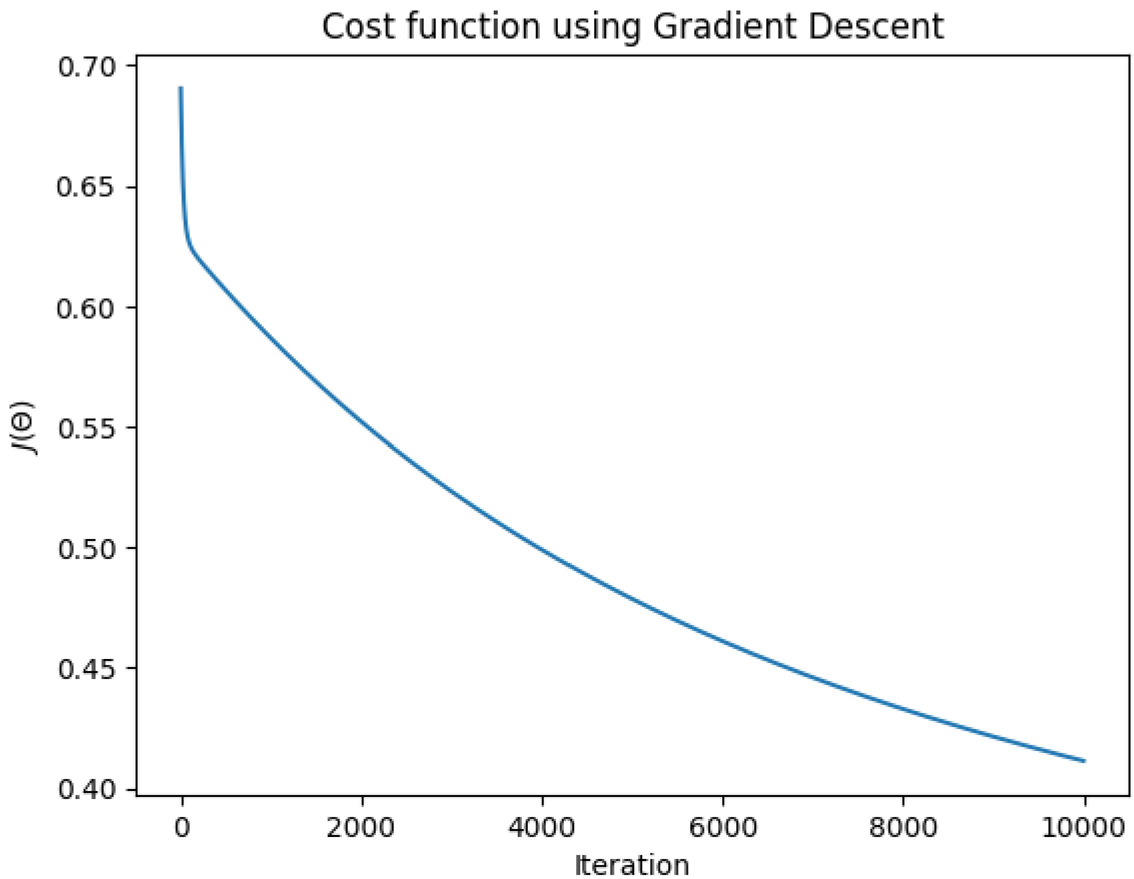
In [288…

```
J_history, theta_optimized = gradient_descent(X_train, y_train, theta, alpha, itera
print('Optimized cost is: ', J_history[-1])
print('Optimized parameters are: ', theta_optimized)
```

```
Optimized cost is:  [0.41143824]
Optimized parameters are:  [[-4.53417377]
 [ 0.31383089]
 [ 0.1815459 ]]
```

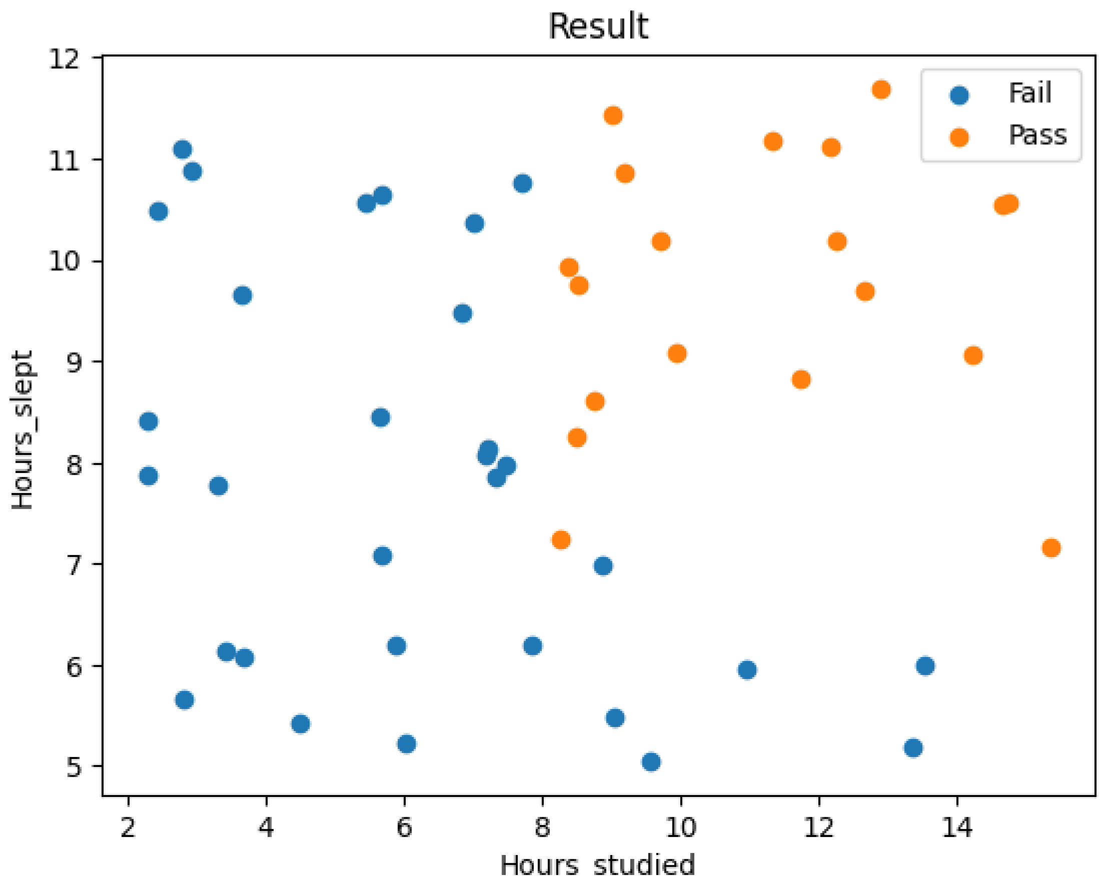
In [289…

```
plt.plot(J_history)
plt.xlabel('Iteration')
plt.ylabel('$J(\Theta)$')
plt.title('Cost function using Gradient Descent')
plt.show()
```



In [290…

```
plt.scatter(X[:, 1][y[:, 0] == 0], X[:, 2][y[:, 0] == 0], label='Fail')
plt.scatter(X[:, 1][y[:, 0] == 1], X[:, 2][y[:, 0] == 1], label='Pass')
plt.xlabel('Hours_studied')
plt.ylabel('Hours_slept')
plt.legend()
plt.title('Result')
plt.show()
```



In [293…

```
y_pred = predict(X_test, theta_optimized)
print('Accuracy: {} %'.format(100 * np.sum(y_pred == y_test) / len(y_test)))
```

```
Accuracy: 100.0 %
```

15 Use some function for regularization of dataset based on problem 14

In [294…

```
# logistic regression with L1 regularization
def logistic_regression_L1(X, y, theta, alpha, iterations, l1):

        m = len(y)
        # cost history
        J_history = np.zeros((iterations, 1))

        for i in range(iterations):

            # hypothesis
            h = sigmoid(X.dot(theta))

            # gradient
            theta = theta - (alpha / m) * (X.T.dot(h - y)) + (l1 / m) * np.sign(the

            # cost
            J_history[i] = cost_function(X, y, theta)
```

```python
    return (J_history, theta)
```

In [295...

```python
alpha = 0.01
iterations = 10000
# regularization parameter
l1 = 0.1
print('Initial cost is: ', cost_function(X_train, y_train, theta))
```

```
Initial cost is:  0.6931471805599454
```

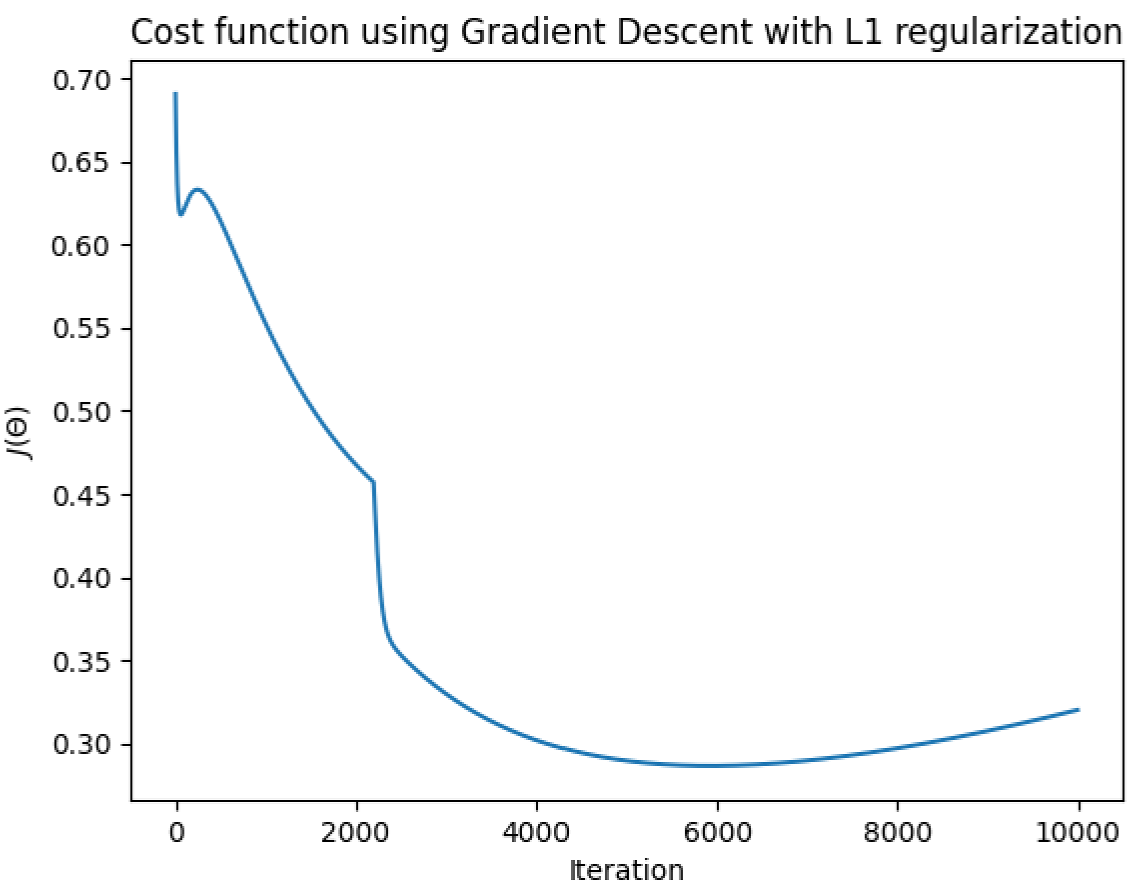
In [296...

```python
J_history, theta_optimized = logistic_regression_L1(X_train, y_train, theta, alpha,
print('Optimized cost is: ', J_history[-1])
print('Optimized parameters are: ', theta_optimized)
```

```
Optimized cost is:  [0.32018745]
Optimized parameters are:  [[-27.89570423]
 [  1.31819399]
 [  1.87622135]]
```

In [297...

```python
plt.plot(J_history)
plt.xlabel('Iteration')
plt.ylabel('$J(\Theta)$')
plt.title('Cost function using Gradient Descent with L1 regularization')
plt.show()
```



In [298...

```python
y_pred = predict(X_test, theta_optimized)
print('Accuracy: {} %'.format(100 * np.sum(y_pred == y_test) / len(y_test)))
```

```
Accuracy: 100.0 %
```

16 Use some function for neural networks, like Stochastic gradient Descent or backpropagation algorithm to predict the value of a variable based on the dataset of problem 14

In [199…

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [302…

```
data = pd.read_csv('/content/student.csv')
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values.reshape(-1, 1)
data.head()
```

Out[302]:

	Hours_studied	Hours_slept	Result
0	7.329712	7.848625	0
1	14.649273	10.545618	1
2	3.431501	6.127123	0
3	5.888299	6.204252	0
4	3.680169	6.072624	0

In [247…

```
def initialize_parameters(layer_dims):

  np.random.seed(3)
  parameters = {}
  L = len(layer_dims)

  for l in range(1, L):

    parameters['W' + str(l)] = np.ones((layer_dims[l-1], layer_dims[l]))*0.1
    parameters['b' + str(l)] = np.zeros((layer_dims[l], 1))


  return parameters
```

In [200…

```
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
```

In [236…

```
def linear_forward(A_prev, W, b):

  Z = np.dot(W.T, A_prev) + b
```

```
    A = sigmoid(Z)

    return A
```

In [246…

```
# L-layer feed forward

def L_layer_forward(X, parameters):

  A = X
  L = len(parameters) // 2                  # number of layers in the neural networ

  for l in range(1, L+1):
    A_prev = A
    Wl = parameters['W' + str(l)]
    bl = parameters['b' + str(l)]

    A = linear_forward(A_prev, Wl, bl)


  return A,A_prev
```

In [245…

```
def update_parameters(parameters,y,y_hat,A1,X):
  parameters['W2'][0][0] = parameters['W2'][0][0] + (0.0001 * (y - y_hat)*A1[0][0])
  parameters['W2'][1][0] = parameters['W2'][1][0] + (0.0001 * (y - y_hat)*A1[1][0])
  parameters['b2'][0][0] = parameters['W2'][1][0] + (0.0001 * (y - y_hat))

  parameters['W1'][0][0] = parameters['W1'][0][0] + (0.0001 * (y - y_hat)*parameter
  parameters['W1'][0][1] = parameters['W1'][0][1] + (0.0001 * (y - y_hat)*parameter
  parameters['b1'][0][0] = parameters['b1'][0][0] + (0.0001 * (y - y_hat)*parameter

  parameters['W1'][1][0] = parameters['W1'][1][0] + (0.0001 * (y - y_hat)*parameter
  parameters['W1'][1][1] = parameters['W1'][1][1] + (0.0001 * (y - y_hat)*parameter
  parameters['b1'][1][0] = parameters['b1'][1][0] + (0.0001 * (y - y_hat)*parameter
```

In [248…

```
# epochs implementation

parameters = initialize_parameters([2,2,1])
epochs = 100

for i in range(epochs):

  Loss = []

  for j in range(df.shape[0]):

    X = data[['Hours_studied', 'Hours_slept']].values[j].reshape(2,1) # Shape(no of
    y = data[['Result']].values[j][0]

    # Parameter initialization


    y_hat,A1 = L_layer_forward(X,parameters)
    y_hat = y_hat[0][0]

    update_parameters(parameters,y,y_hat,A1,X)
```

```
    Loss.append(-y*np.log(y_hat) - (1-y)*np.log(1-y_hat))

  print('Epoch - ',i+1,'Loss - ',np.array(Loss).mean())

parameters
```

```
Epoch -  1 Loss -  0.7285424020448542
Epoch -  2 Loss -  0.7293378301579925
Epoch -  3 Loss -  0.7290280582827963
Epoch -  4 Loss -  0.7287202358232555
Epoch -  5 Loss -  0.7284143478203307
Epoch -  6 Loss -  0.7281103794440932
Epoch -  7 Loss -  0.7278083159927268
Epoch -  8 Loss -  0.727508142891527
Epoch -  9 Loss -  0.7272098456919093
Epoch -  10 Loss -  0.7269134100704149
Epoch -  11 Loss -  0.7266188218277208
Epoch -  12 Loss -  0.7263260668876553
Epoch -  13 Loss -  0.7260351312962137
Epoch -  14 Loss -  0.7257460012205778
Epoch -  15 Loss -  0.7254586629481398
Epoch -  16 Loss -  0.7251731028855295
Epoch -  17 Loss -  0.7248893075576442
Epoch -  18 Loss -  0.7246072636066834
Epoch -  19 Loss -  0.7243269577911845
Epoch -  20 Loss -  0.7240483769850684
Epoch -  21 Loss -  0.7237715081766819
Epoch -  22 Loss -  0.7234963384678492
Epoch -  23 Loss -  0.7232228550729254
Epoch -  24 Loss -  0.7229510453178549
Epoch -  25 Loss -  0.7226808966392341
Epoch -  26 Loss -  0.7224123965833782
Epoch -  27 Loss -  0.7221455328053925
Epoch -  28 Loss -  0.7218802930682492
Epoch -  29 Loss -  0.7216166652418669
Epoch -  30 Loss -  0.7213546373021977
Epoch -  31 Loss -  0.7210941973303148
Epoch -  32 Loss -  0.7208353335115103
Epoch -  33 Loss -  0.7205780341343937
Epoch -  34 Loss -  0.7203222875899955
Epoch -  35 Loss -  0.7200680823708788
Epoch -  36 Loss -  0.7198154070702537
Epoch -  37 Loss -  0.7195642503810972
Epoch -  38 Loss -  0.7193146010952772
Epoch -  39 Loss -  0.719066448102684
Epoch -  40 Loss -  0.7188197803903659
Epoch -  41 Loss -  0.7185745870416684
Epoch -  42 Loss -  0.7183308572353821
Epoch -  43 Loss -  0.7180885802448931
Epoch -  44 Loss -  0.7178477454373393
Epoch -  45 Loss -  0.7176083422727737
Epoch -  46 Loss -  0.7173703603033311
Epoch -  47 Loss -  0.7171337891724014
Epoch -  48 Loss -  0.7168986186138089
Epoch -  49 Loss -  0.7166648384509944
Epoch -  50 Loss -  0.7164324385962065
Epoch -  51 Loss -  0.7162014090496968
Epoch -  52 Loss -  0.7159717398989187
Epoch -  53 Loss -  0.7157434213177359
Epoch -  54 Loss -  0.7155164435656325
Epoch -  55 Loss -  0.7152907969869317
Epoch -  56 Loss -  0.7150664720100174
```

```
Epoch -  57 Loss -  0.7148434591465649
Epoch -  58 Loss -  0.7146217489907737
Epoch -  59 Loss -  0.7144013322186068
Epoch -  60 Loss -  0.7141821995870384
Epoch -  61 Loss -  0.7139643419333042
Epoch -  62 Loss -  0.7137477501741575
Epoch -  63 Loss -  0.7135324153051323
Epoch -  64 Loss -  0.7133183283998119
Epoch -  65 Loss -  0.7131054806091021
Epoch -  66 Loss -  0.7128938631605106
Epoch -  67 Loss -  0.7126834673574335
Epoch -  68 Loss -  0.7124742845784436
Epoch -  69 Loss -  0.7122663062765885
Epoch -  70 Loss -  0.712059523978692
Epoch -  71 Loss -  0.711853929284662
Epoch -  72 Loss -  0.7116495138668034
Epoch -  73 Loss -  0.7114462694691361
Epoch -  74 Loss -  0.7112441879067213
Epoch -  75 Loss -  0.7110432610649885
Epoch -  76 Loss -  0.7108434808990739
Epoch -  77 Loss -  0.7106448394331593
Epoch -  78 Loss -  0.7104473287598185
Epoch -  79 Loss -  0.710250941039371
Epoch -  80 Loss -  0.710055668499238
Epoch -  81 Loss -  0.7098615034333049
Epoch -  82 Loss -  0.7096684382012907
Epoch -  83 Loss -  0.7094764652281217
Epoch -  84 Loss -  0.7092855770033094
Epoch -  85 Loss -  0.7090957660803359
Epoch -  86 Loss -  0.7089070250760446
Epoch -  87 Loss -  0.7087193466700331
Epoch -  88 Loss -  0.7085327236040562
Epoch -  89 Loss -  0.7083471486814293
Epoch -  90 Loss -  0.7081626147664416
Epoch -  91 Loss -  0.7079791147837713
Epoch -  92 Loss -  0.707796641717907
Epoch -  93 Loss -  0.7076151886125741
Epoch -  94 Loss -  0.7074347485701682
Epoch -  95 Loss -  0.7072553147511901
Epoch -  96 Loss -  0.707076880373689
Epoch -  97 Loss -  0.7068994387127091
Epoch -  98 Loss -  0.7067229830997424
Epoch -  99 Loss -  0.7065475069221854
Epoch -  100 Loss -  0.7063730036228019
```

```
Out[248]: {'W1': array([[0.09540141, 0.09117589],
                 [0.09541098, 0.09110489]]),
           'b1': array([[-0.00129987],
                 [-0.00131423]]),
           'W2': array([[0.04190988],
                 [0.042359  ]]),
           'b2': array([[0.04240612]])}
```