24/March/25

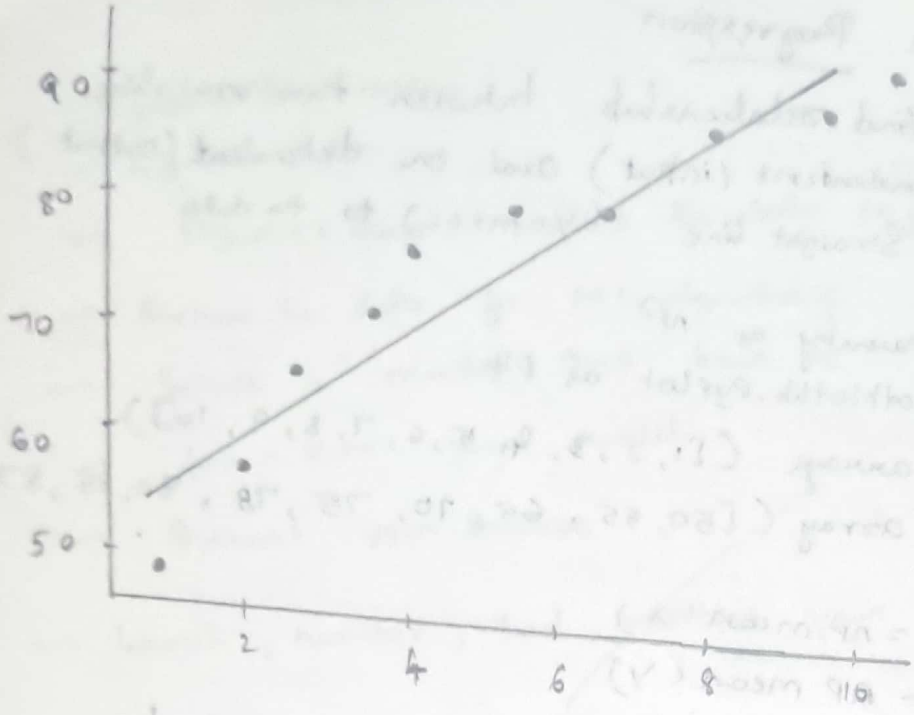
Lab - 3

1) Linear Regression

Used to find relationship between two variables one independent (input) and one dependent (output)

if fit a Straight line ($y = mx + c$) to the data

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
y = np.array([50, 55, 65, 70, 75, 78, 80, 85, 87, 90])

x_mean = np.mean(x)
y_mean = np.mean(y)

m = sum((x - x_mean) * (y - y_mean)) /
        sum((x - x_mean) ** 2)

c = y_mean - m * x_mean

def Predict(x):
    return m * x + c
plt.scatter(x, y, color = 'blue', label = "Actual Data")

y_pred = Predict(x)
plt.plot(x, y_pred, color = 'red', label = "Regression line")

plt.xlabel("Hours Studied")
plt.ylabel("Test Score")
plt.title("Linear Regression : Hours vs Score")
plt.legend()
plt.show()
Print("Prediction for 11 hrs of study:", Predict(11))
```



Linear regression: Hours vs Score • Actual data

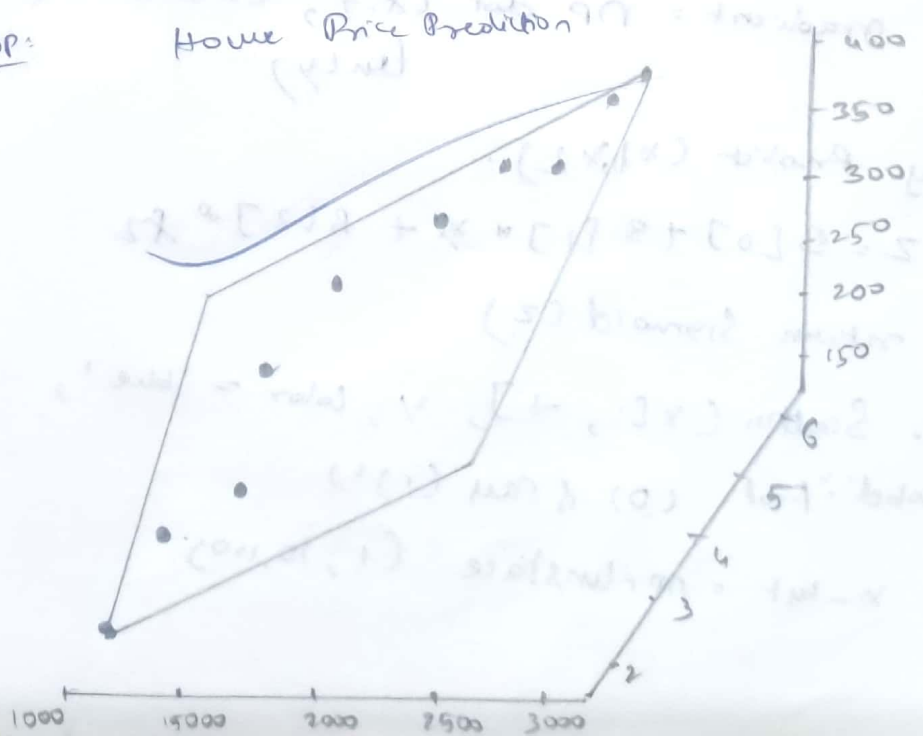
Prediction for 11 hrs of Study: 97.4 — Regression line

2) Multiple Regression

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

X = np.array([
    [800, 2], [1000, 3], [1200, 3],
    [1500, 4], [1800, 4], [2000, 5], [2200, 6],
    [2800, 6], [3000, 6]])

Y = np.array([150, 180, 200, 240, 280, 310, 340,
              370, 400, 420]) # Prices in $1000

X = np.column_stack([np.ones(len(X)), X])
B = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(Y)

def Predict(sqft, beds):
fig = plt.figure(sqft, beds):
```

```
return B[0] + B[1] * Sqft + B[2] * beds

fig = PH.figure()
ax = fig.add_subPlot(111, Projection = '3d')

Sqft_vals = X[:, 1]
bed_vals = X[:, 2]
Price_vals = y

ax.scatter(Sqft_vals, bed_vals, Price_vals, color = 'blue',
label = "Actual Data")

Sqft_grid, bed_grid = np.meshgrid(np.linspace(800,
    3000, 10), np.linspace(2, 6, 5))
Price_grid = Predict(Sqft_grid, bed_grid)
ax.plot_surface(Sqft_grid, bed_grid, Price_grid,
    color = 'red', alpha = 0.5)
ax.set_xlabel("Square Footage")
ax.set_ylabel("Bedrooms")
ax.set_zlabel("Price ($1000)")
ax.set_title("Multiple linear regression:
    House Price Prediction")

Plt.show()

Print("Prediction for 2600 Sqft, 5 bedrooms:",
    Predict(2600, 5))
```

OP:



8) logistic regression

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

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

x = np.array([
[1, 50], [2,55], [3, 60], [4, 62],
[5, 65], [6, 70], [7, 75], [8, 78],
[9, 85], [10, 90]])

y = np.array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1])
#0 = fail,  1 = Pass

x = np.column_stack((np.ones(len(x)), x))
B = np.zeros(x.shape[1])

lr = 0.01
epochs = 2000

for _ in range(epochs):
    Prediction = sigmoid(np.dot(x, B))
    gradient = np.dot(x.T, (Prediction - y))/
               len(y)

def Predict(x1, x2):
    z = B[0] + B[1] * x1 + B[2] * x2
    return Sigmoid(z)

plt.scatter(x[:, 1], y, color = 'blue',
label = "Fail (0) & Pass (1)")
x_test = np.linspace(1, 10, 100)
```

```
y_test = [Predict(x,
plt.plot(x_test, y_test
"Logistic Regression
plt.xlabel("Hours St
plt.ylabel("Probabilit
plt.legend()
plt.show()
print("Probability of
Studied 5 75
Predict(

Logistic Regression
This Studied
```

```
Y_test = [predict(x, 65) for x in X_test]

plt.plot(X_test, Y_test, color='red', label=
    "Logistic Regression Curve")
plt.xlabel("Hours Studied")
plt.ylabel("Probability of Passing")
plt.legend()
plt.show()
print("Probability of Passing for 7 hrs
    Studied & 75 Prev Score:",
        predict(7, 75))
```

Logistic Regression : Pass/fail Prediction for 7 hrs Studied & 75