LAB - 03

(a) Linear Regression

(1) solving following linear Regression problem using matrix approach.

x: (week)	1	2	3	4
y: (Sales in thousand)	2	4	5	9

Implementation

```
import numpy as np

X = np.array([[1, 1],
              [1, 2],
              [1, 3],
              [1, 4]])

Y = np.array([2, 4, 5, 9])

XT = X.T
XTX = np.dot(XT, X)
XTX_inv = np.linalg.inv(XTX)
XTY = np.dot(XT, Y)
beta = np.dot(XTX_inv, XTY)

beta_0, beta_1 = beta
print(f"Intercept (B0): {beta_0:.2f}")
Print(f"Shape (B1): {beta_1:.2f}")

Y_pred = np.dot(X, beta)
print("Predicted sales:", Y_pred)
```

Output :
Intercept (β0) : -0.50
slope (β1) : 2.20
predicted sales : [1.7, 3.9, 6.1, 8.3]

(1) Multiple Linear Regression

Q: predict Glucose level of diabetes_data.csv using MLR.

diabetes_data.csv :

Age	BMI	BP	Insulin	D.PedigreeFun	glucose
50	25.3	80	150	0.5	140
:	:	:	:	:	:
:	:	:	:	:	:
33	29.1	81	195	0.67	160

Implementation

```
import numpy as np
import pandas as pd
df = pd.read_csv("diabetes_data.csv");

x = df[['Age', 'BMI', 'BP', 'Insulin', 'D.PedigreeFun']].values

y = df['Glucose'].values

x = np.c_[np.ones(x.shape[0]), x]
```

PAGE NO :
DATE :

```
train_size = int(0.8 * len(x))
x_train, x_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]

theta = np.linalg.inv(x_train.T @ x_train) @
        x_train.T @ y_train.

y_pred = x_test @ theta

mse = np.mean((y_test - y_pred) ** 2)
r2 = 1 - (np.sum((y_test - y_pred) ** 2) / np.sum
        ((y_test - np.mean(y_test)) ** 2))

print("Multiple Linear Regn model (Age, BMI, I, DP -> 619)
print(f"Mean Squared Error : {mse}")
print(f"R^2 Score : {r2}")
```

Output :-

MLR :-

Mean Squared Error : 41.23512336576399
R^2 Score : 0.93127479439039394

(c) Logistic Regression.

```
X = df[['Age', 'BMI', 'BP', 'Insulin', 'DPF']].values
y = df['Diabetes'].values

X = np.c_[np.ones(X.shape[0]), X]

theta = np.zeros(X.shape[1])

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

def compute_cost(X, y, theta):
    m = len(y)
    h = sigmoid(X @ theta)
    return (-1/m) * np.sum(y * np.log(h) + (1-y)
            * np.log(1-h))

def gradient_descent(X, y, theta, alpha, iters):
    m = len(y)
    h = sigmoid(X @ theta)
    gradient = (1/m) * (X.T @ (h-y))
    theta -= alpha * gradient
    cost_history.append(compute_cost(X, y, theta))

    return theta, cost_history

alpha = 0.01
iterations = 1000
theta, cost_history = gradient_descent(X, y,
        theta, alpha, iterations).
```

PAGE NO :
DATE :

```
print(" Final theta:", theta)

def predict(x, theta):
    return (sigmoid(x @ theta) >= 0.5).astype(int)

predictions = predict(x, theta)
accuracy = np.mean(predictions == y) * 100

print(f"LRA : {accuracy:.2f} %")
```

Output :-

Final theta values

[0.0046, 3.8204, -0.0034, 0.0843, -1.8403, -0.0101]

Accuracy = 100.00 %