

Name: Ritesh Chaudhary

Id: 2438464

Workshop Week-5

```
[ ] #Name: Ritesh_Chaudhary  
    #ID:2438464
```

```
import pandas as pd  
import numpy as np  
import matplotlib as plt  
from sklearn.model_selection import train_test_split  
from sklearn.metrics import mean_squared_error, r2_score  
  
dataSet=pd.read_csv("/content/drive/MyDrive/Concept of Ai Technology/student.csv")  
headData=dataSet.head()  
print("This is the most top data : \n" ,headData,"\n")
```

```
⇒ This is the most top data :  
   Math  Reading  Writing  
0    48      68     63  
1    62      81     72  
2    79      80     78  
3    76      83     79  
4    59      64     62
```

+ Code + Text

✓
0s

```
[4] tailData=dataset.tail()
    print("This is the most button data :\n" ,tailData,"\n" )
```

⇨ This is the most button data :

	Math	Reading	Writing
995	72	74	70
996	73	86	90
997	89	87	94
998	83	82	78
999	66	66	72

✓
0s

```
dataInfo=dataset.info
print("Basic info of the given dataset:" ,dataInfo,"\n")
```

⇨ Basic info of the given dataset: <bound method DataFrame.info of

	Math	Reading	Writing
--	------	---------	---------

0	48	68	63
1	62	81	72
2	79	80	78
3	76	83	79
4	59	64	62
...
995	72	74	70
996	73	86	90
997	89	87	94
998	83	82	78
999	66	66	72

[1000 rows x 3 columns]>

✓
0s

```
[6] dataDesc=dataset.describe()
    print("Description of the given dataset : \n" ,dataDesc,"\n")
```

⇨ Description of the given dataset :

	Math	Reading	Writing
count	1000.000000	1000.000000	1000.000000
mean	67.290000	69.872000	68.616000
std	15.085008	14.657027	15.241287
min	13.000000	19.000000	14.000000
25%	58.000000	60.750000	58.000000
50%	68.000000	70.000000	69.500000
75%	78.000000	81.000000	79.000000
max	100.000000	100.000000	100.000000

✓
0s



```
X = dataSet[['Math', 'Reading']]
Y = dataSet['Writing']
```

```
print("Features (X):")
print(X)
```

```
print("\nTarget (Y):")
print(Y)
```



Features (X):

	Math	Reading
0	48	68
1	62	81
2	79	80
3	76	83
4	59	64
..
995	72	74
996	73	86
997	89	87
998	83	82
999	66	66

[1000 rows x 2 columns]

Target (Y):

0	63
1	72
2	78
3	79
4	63

```

X = dataSet[['Math', 'Reading']].to_numpy()
Y_actual = dataSet['Writing'].to_numpy()
X = X.T
d = X.shape[0]
W = np.random.rand(d, 1)
Y_pred = np.dot(W.T, X).T
print("Weight Vector (W):")
print(W)

print("\nFeature Matrix (X):")
print(X)

print("\nPredicted Target Vector (Y):")
print(Y_pred)
print("\nShapes:")
print("W:", W.shape)
print("X:", X.shape)
print("Y (predicted):", Y_pred.shape)

```

```

⇒ Weight Vector (W):
[[0.6862043 ]
 [0.74594273]]

Feature Matrix (X):
[[48 62 79 ... 89 83 66]
 [68 81 80 ... 87 82 66]]

Predicted Target Vector (Y):
[[ 83.66191247]]

```

[71.24892128]
[113.76608181]
[94.07445388]
[66.44549115]
[94.13419231]
[66.68444488]
[115.85535159]
[107.41102779]
[88.55416258]
[88.34586573]
[99.68356517]
[82.01989327]
[110.81296775]
[113.79673869]
[91.77688725]
[94.37314604]
[114.45228612]
[105.0843796]
[106.15809605]
[114.48294299]
[99.11683773]
[103.62157569]
[72.44211459]
[77.75410903]
[73.84518006]
[108.90291326]
[102.54785924]
[61.64206103]
[114.989932]
[110.96152617]
[90.85172922]
[71.54761343]

```
from sklearn.model_selection import train_test_split
X = dataSet[['Math', 'Reading']].to_numpy()
Y = dataSet['Writing'].to_numpy()
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
print("Training set shapes:")
print("X_train:", X_train.shape, "Y_train:", Y_train.shape)

print("\nTest set shapes:")
print("X_test:", X_test.shape, "Y_test:", Y_test.shape)
```

➡ Training set shapes:
X_train: (800, 2) Y_train: (800,)

Test set shapes:
X_test: (200, 2) Y_test: (200,)

```
0s ✓ def cost_function(X, Y, W):
    """
    Parameters:
    - X: Feature Matrix
    - Y: Target Matrix
    - W: Weight Matrix

    Output:
    - cost: Mean Squared Error
    """
    y_pred = np.dot(W.T, X).flatten()
    error = y_pred - Y
    cost = np.mean(error ** 2) / 2

    return cost
```

```
✓ [11] def cost_function(X, Y, W):
```

```

def cost_function(X, Y, W):
    """
    Parameters:
    - X: Feature Matrix
    - Y: Target Matrix
    - W: Weight Matrix

    Output:
    - cost: Mean Squared Error
    """
    y_pred = np.dot(X, W)
    error = y_pred.flatten() - Y
    cost = np.mean(error ** 2) / 2
    return cost

X_test = np.array([[1, 2], [3, 4], [5, 6]])
Y_test = np.array([3, 7, 11])
W_test = np.array([1, 1]).reshape(2, 1)
cost = cost_function(X_test, Y_test, W_test)
print("Cost function output:", cost)
if cost == 0:
    print("Proceed further.")
else:
    print("Something went wrong: Reimplement the cost function.")

```

```

→ Cost function output: 0.0
Proceed further.

```

```

def gradient_descent(X, Y, W, alpha, iterations):
    """
    Perform gradient descent to optimize the parameters of a linear regression model.

    Parameters:
    - X (numpy.ndarray): Feature matrix (m x n).
    - Y (numpy.ndarray): Target vector (m x 1).
    - W (numpy.ndarray): Initial guess for parameters (n x 1).
    - alpha (float): Learning rate.
    - iterations (int): Number of iterations for gradient descent.

    Returns:
    - W_update (numpy.ndarray): Updated parameters (n x 1).
    - cost_history (list): History of cost values over iterations.
    """
    m = len(Y)
    cost_history = [0] * iterations

    for iteration in range(iterations):
        Y_pred = np.dot(X, W)
        loss = Y_pred - Y
        dw = (1 / m) * np.dot(X.T, loss)
        W = W - alpha * dw
        cost = cost_function(X, Y, W)
        cost_history[iteration] = cost

    return W, cost_history

```

```

[13] np.random.seed(0)
X = np.random.rand(100, 3)
Y = np.random.rand(100)
W = np.random.rand(3)
alpha = 0.01
iterations = 1000
final_params, cost_history = gradient_descent(X, Y, W, alpha, iterations)
print("Final Parameters:", final_params)
print("Cost History:", cost_history)

```

Final Parameters: [0.20551667 0.54295081 0.10388027]
Cost History: [0.10711197094660153, 0.10634880599939901, 0.10559826315680616, 0.10486012948320558, 0.1041341956428534, 0.10342025583900626, 0.1027181077540776, 0.10201611111111111, 0.1013141956428534, 0.1006122948320558, 0.099910388027, 0.09920847991791881, 0.09850657181077541, 0.09780466370966371, 0.09710275560855609, 0.09640084750744751, 0.09569893940633941, 0.09500000000000001, 0.09429606060606061, 0.09359212121212122, 0.09288818181818182, 0.09218424242424243, 0.09148030303030303, 0.09077636363636364, 0.09007242424242425, 0.08936848484848485, 0.08866454545454546, 0.08796060606060606, 0.08725666666666667, 0.08655272727272728, 0.08584878787878788, 0.08514484848484849, 0.0844409090909091, 0.0837369696969697, 0.08303303030303031, 0.08232909090909091, 0.08162515151515152, 0.08092121212121213, 0.08021727272727273, 0.07951333333333333, 0.07880939393939394, 0.07810545454545455, 0.07740151515151516, 0.07669757575757577, 0.07599363636363637, 0.07528969696969698, 0.07458575757575759, 0.07388181818181819, 0.0731778787878788, 0.0724739393939394, 0.07176999999999999, 0.07106606060606061, 0.07036212121212122, 0.06965818181818182, 0.06895424242424243, 0.06825030303030304, 0.06754636363636364, 0.06684242424242425, 0.06613848484848486, 0.06543454545454546, 0.06473060606060607, 0.06402666666666667, 0.06332272727272728, 0.06261878787878789, 0.0619148484848485, 0.0612109090909091, 0.0605069696969697, 0.05980303030303031, 0.05909909090909091, 0.05839515151515152, 0.05769121212121213, 0.05698727272727273, 0.05628333333333333, 0.05557939393939394, 0.05487545454545455, 0.05417151515151516, 0.05346757575757577, 0.05276363636363637, 0.05205969696969698, 0.05135575757575759, 0.05065181818181819, 0.0499478787878788, 0.0492439393939394, 0.04853999999999999, 0.04783606060606061, 0.04713212121212122, 0.04642818181818182, 0.04572424242424243, 0.04502030303030304, 0.04431636363636364, 0.04361242424242425, 0.04290848484848486, 0.04220454545454546, 0.04150060606060607, 0.04079666666666667, 0.04009272727272728, 0.03938878787878789, 0.0386848484848485, 0.0379809090909091, 0.0372769696969697, 0.03657303030303031, 0.03586909090909091, 0.03516515151515152, 0.03446121212121213, 0.03375727272727273, 0.03305333333333333, 0.03234939393939394, 0.03164545454545455, 0.03094151515151516, 0.03023757575757577, 0.02953363636363637, 0.02882969696969698, 0.02812575757575759, 0.02742181818181819, 0.0267178787878788, 0.0260139393939394, 0.02530999999999999, 0.02460606060606061, 0.02390212121212122, 0.02319818181818182, 0.02249424242424243, 0.02179030303030304, 0.02108636363636364, 0.02038242424242425, 0.01967848484848486, 0.01897454545454546, 0.01827060606060607, 0.01756666666666667, 0.01686272727272728, 0.01615878787878789, 0.0154548484848485, 0.0147509090909091, 0.0140469696969697, 0.01334303030303031, 0.01263909090909091, 0.01193515151515152, 0.01123121212121213, 0.01052727272727273, 0.00982333333333333, 0.00911939393939394, 0.00841545454545455, 0.00771151515151516, 0.00700757575757577, 0.00630363636363637, 0.00559969696969698, 0.00489575757575759, 0.00419181818181819, 0.0034878787878788, 0.0027839393939394, 0.0020800000000000001, 0.0013760606060606061, 0.0006721212121212122, 0.00000000000000001]

✓
0s

```
def rmse(Y, Y_pred):  
    """  
    Calculate the Root Mean Squared Error (RMSE).  
  
    Parameters:  
    - Y: Actual target values (1D array).  
    - Y_pred: Predicted target values (1D array).  
  
    Returns:  
    - rmse: Root Mean Squared Error.  
    """  
    n = len(Y)  
    rmse = np.sqrt(np.sum((Y - Y_pred) ** 2) / n)  
    return rmse
```

✓
0s

```
[15] def r2(Y, Y_pred):
```

✓
0s

```
def r2(Y, Y_pred):  
    """  
    Calculate the R Squared (coefficient of determination) value.  
  
    Parameters:  
    - Y: Actual target values (1D array).  
    - Y_pred: Predicted target values (1D array).  
  
    Returns:  
    - r2: R Squared value.  
    """  
    n = len(Y)  
    mean_y = np.mean(Y)  
    ss_total = np.sum((Y - mean_y) ** 2)  
    ss_residual = np.sum((Y - Y_pred) ** 2)  
    r2 = 1 - (ss_residual / ss_total)  
    return r2
```

```

# Main Function
def main():
    X = dataSet[['Math', 'Reading']].values
    Y = dataSet['Writing'].values
    X_mean = X.mean(axis=0)
    X_std = X.std(axis=0)
    X_normalized = (X - X_mean) / X_std
    X_train, X_test, Y_train, Y_test = train_test_split(X_normalized, Y, test_size=0.2, random_state=42)
    W = np.zeros(X_train.shape[1])
    alpha = 0.0001
    iterations = 1000
    W_optimal, cost_history = gradient_descent(X_train, Y_train, W, alpha, iterations)
    Y_pred = np.dot(X_test, W_optimal)
    model_rmse = rmse(Y_test, Y_pred)
    model_r2 = r2(Y_test, Y_pred)
    print("Final Weights:", W_optimal)
    print("Cost History (First 10 iterations):", cost_history[:10])
    print("RMSE on Test Set:", model_rmse)
    print("R-Squared on Test Set:", model_r2)
if __name__ == "__main__":
    main()

```

Final Weights: [1.06484959 1.40144515]
 Cost History (First 10 iterations): [2471.6620474310935, 2471.6253575184387, 2471.5886802576224, 2471.552015644236, 2471.5153636738696, 2471.4787243421174, 2471.44208
 RMSE on Test Set: 69.78693505233176
 R-Squared on Test Set: -18.456124807688028