Result of Worksheet5

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

```python
#TO-DO 1
# Load dataset
data = pd.read_csv("student.csv")

# 1. View top 5 rows
print("Top 5 rows:")
display(data.head())

# 2. View bottom 5 rows
print("Bottom 5 rows:")
display(data.tail())

# 3. Dataset information
print("Dataset Info:")
data.info()

# 4. Descriptive statistics
print("Dataset Description:")
display(data.describe())

# 5. Split Features (X) and Label (Y)
X = data[['Math', 'Reading']].values
Y = data['Writing'].values
```

```
Top 5 rows:
```

	Math	Reading	Writing
0	48	68	63
1	62	81	72
2	79	80	78
3	76	83	79
4	59	64	62

```
Bottom 5 rows:
```

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

```
Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
```

```
RangeIndex: 1000 entries, 0 to 999
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Math     1000 non-null   int64
 1   Reading  1000 non-null   int64
 2   Writing  1000 non-null   int64
dtypes: int64(3)
memory usage: 23.6 KB
Dataset Description:
```

	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

```python
#TO-DO 2
# Feature matrix X (d x n → here n x d, handled via dot product)
# Weight vector W will be (d,)
# Y will be (n,)
print("Shape of X:", X.shape)
print("Shape of Y:", Y.shape)
```

```
Shape of X: (1000, 2)
Shape of Y: (1000,)
```

```python
#TO-DO 3
X_train, X_test, Y_train, Y_test = train_test_split(
    X, Y, test_size=0.2, random_state=42
)

print("Training samples:", X_train.shape)
print("Testing samples:", X_test.shape)
```

```
Training samples: (800, 2)
Testing samples: (200, 2)
```

[6]
✓ 0s

```python
#TO-DO 4
def cost_function(X, Y, W):
    """
    Mean Squared Error Cost Function
    """
    n = len(Y)
    Y_pred = np.dot(X, W)
    error = Y_pred - Y
    cost = (1 / (2 * n)) * np.sum(error ** 2)
    return cost
```

```python
#TO-DO 5
# Test case
X_test_case = np.array([[1, 2], [3, 4], [5, 6]])
Y_test_case = np.array([3, 7, 11])
W_test_case = np.array([1, 1])

cost = cost_function(X_test_case, Y_test_case, W_test_case)

if cost == 0:
    print("Proceed Further")
else:
    print("Something went wrong")

print("Cost function output:", cost)
```

```
Proceed Further
Cost function output: 0.0
```

```python
#TO-DO 6
def gradient_descent(X, Y, W, alpha, iterations):
    """
    Gradient Descent for Linear Regression
    """
    cost_history = []
    m = len(Y)

    for _ in range(iterations):
        # Step 1: Prediction
        Y_pred = np.dot(X, W)

        # Step 2: Loss
        loss = Y_pred - Y

        # Step 3: Gradient
        dw = (1 / m) * np.dot(X.T, loss)

        # Step 4: Update weights
        W = W - alpha * dw

        # Step 5: Store cost
        cost = cost_function(X, Y, W)
        cost_history.append(cost)

    return W, cost_history
```

10]
0s

```python
#TO-DO 7
np.random.seed(0)

X_rand = np.random.rand(100, 3)
Y_rand = np.random.rand(100)
W_rand = np.random.rand(3)

alpha = 0.01
iterations = 1000

final_params, cost_history = gradient_descent(
    X_rand, Y_rand, W_rand, alpha, iterations
)

print("Final Parameters:", final_params)
print("First 10 Cost Values:", cost_history[:10])
```

```
Final Parameters: [0.20551667 0.54295081 0.10388027]
First 10 Cost Values: [np.float64(0.10711197094660153), np.float64(0.10634880599939901), np.float64(0.10559826315680618), np.float64
```

```python
#TO-DO 8
def rmse(Y, Y_pred):
    """
    Root Mean Squared Error
    """
    rmse_value = np.sqrt(np.mean((Y - Y_pred) ** 2))
    return rmse_value
```

```python
#TO-DO 9
def r2(Y, Y_pred):
    """
    R-Squared Metric
    """
    mean_y = np.mean(Y)
    ss_tot = np.sum((Y - mean_y) ** 2)
    ss_res = np.sum((Y - Y_pred) ** 2)
    r2_value = 1 - (ss_res / ss_tot)
    return r2_value
```

```python
#To-DO 10
def main():
    # Load dataset
    data = pd.read_csv("student.csv")

    # Features and Target
    X = data[['Math', 'Reading']].values
    Y = data['Writing'].values

    # Train-test split
    X_train, X_test, Y_train, Y_test = train_test_split(
        X, Y, test_size=0.2, random_state=42
    )

    # Initialize parameters
    W = np.zeros(X_train.shape[1])
    alpha = 0.00001
    iterations = 1000

    # Train model
    W_optimal, cost_history = gradient_descent(
        X_train, Y_train, W, alpha, iterations
    )

    # Predictions
    Y_pred = np.dot(X_test, W_optimal)
```

```python
    # Predictions
    Y_pred = np.dot(X_test, W_optimal)

    # Evaluation
    model_rmse = rmse(Y_test, Y_pred)
    model_r2 = r2(Y_test, Y_pred)

    # Results
    print("Final Weights:", W_optimal)
    print("Cost History (First 10):", cost_history[:10])
    print("RMSE on Test Set:", model_rmse)
    print("R-Squared on Test Set:", model_r2)

# Run
main()
```

```
Final Weights: [0.34811659 0.64614558]
Cost History (First 10): [np.float64(2013.165570783755), np.float64(1640.286832599692), np.float64(1337.0619994901588), np.float64(1
RMSE on Test Set: 5.2798239764188635
R-Squared on Test Set: 0.8886354462786421
```

TO-DO 11

1

The model does not overfit beacause trainging is done using gradient descent with limited iterations Performance is acceptable if: RMSE is reasonably low R square is close to 1

2

learning Rate Experiment: Low α (0.000001) ⟶ Very slow convergence

Optimal α (0.00001) ⟶ Stable learning

High α (0.001) ⟶ Cost may diverge