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

data = pd.read_csv('/content/drive/MyDrive/Concept of AI -- week
5/student.csv')

# 1. Print the first 5 and last 5 rows of the dataset

print("Top 5 rows of the dataset:")
print(data.head(5))
```

```
Top 5 rows of the dataset:
   Math  Reading  Writing
0    48       68       63
1    62       81       72
2    79       80       78
3    76       83       79
4    59       64       62
```

```python
print("Bottom 5 rows of the dataset:")
print(data.tail(5))
```

```
Bottom 5 rows of the dataset:
     Math  Reading  Writing
995    72       74       70
996    73       86       90
997    89       87       94
998    83       82       78
999    66       66       72
```

```python
# 2. Print the information of the dataset
print("Information about the dataset:")
print(data.info())
```

```
Information about the dataset:
<class 'pandas.core.frame.DataFrame'>
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
None
```

```python
# 3. Gather descriptive statistics about the dataset
print("Descriptive statistics of the dataset:")
print(data.describe())
```

```
Descriptive statistics of the 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
```

```
# 4. Split the dataset into Features (X) and Labels (Y)
# Features: Math and Reading marks, Labels: Writing marks
X = data[['Math', 'Reading']]
Y = data['Writing']

print("Feature X:")
print(X.head())

print("Label Y:")
print(Y.head())
```

```
Feature X:
   Math  Reading
0    48       68
1    62       81
2    79       80
3    76       83
4    59       64
Label Y:
0    63
1    72
2    78
3    79
4    62
Name: Writing, dtype: int64
```

```
# Step 1: Convert Features (X) and Labels (Y) into matrices
X_matrix = X.to_numpy().T # Convert X to a numpy array and transpose
it to shape (d, n)
Y_matrix = Y.to_numpy()  #Convert Y to a numpy array with shape (n, )


# Step 2: Create a weights matrix (W) initialized to zeros
# Number of features = d (rows of X_matrix)
num_features = X_matrix.shape[0]
W_matrix = np.zeros((num_features,))

# Print the matrices for verification
print("Feature Matrix (X):\n", X_matrix)
```

```
print("Label Matrix (Y):\n", Y_matrix)
print("Weights Matrix (W):\n", W_matrix)
```

```
Feature Matrix (X):
 [[48 62 79 ... 89 83 66]
 [68 81 80 ... 87 82 66]]
Label Matrix (Y):
 [ 63  72  78  79  62  85  83  41  80  77  64  90  45  77  70  46  76
44
  85  72  53  66  75  49  84  83  68  66  77  78  74  83  72  65  46
66
  50  79  68  46  86  70  61  53  72  75  50  77 100  81 100  87  78
48
  50  44  48  43  67  78  58  91  92  78  42  85  73  83  61  58  60
55
  48  62  68  59  62  48  74  63  80  79  73  79  45  67  89  77  81
88
  53  68  79  77  63  73  60  67 100  79  26  51  80  57  41  78  68
49
  76  41  71  77  89  86  55  80  56  74  85  80  73  74  86  56  53
44
  41  59  71  81  74  78  67  53  56  75  82  79  99  76  59  96  75
61
  56  88  65 100  79  55  61  83  74  59  54  47  82  74  59  74  84
59
  43  65  61  78  84  73  73  92  63  72  61  59  70  87  78  65  73
62
  69  55  73  63  67  86  78  85  83  80  60  90  56  70  55  80  82
60
  78  76  94  75  68  71  85  46  58  46  84  58  57  59  77  63  68
99
  48  91  57  80  46  75  59  87  82  79  66  68  66  61  66  63  72
73
  77  84  83  42  72  76  76  39  74  43  63  74  52  31  65  45  87
63
  51  82  86  76  27  70  79  66  61  62  47  17  65  76  75  66  59
61
  93  40  66  43  71  64  55  86  65  70  65  53  49  67  76  95  76
48
  60  53  69  78  62  66  51  52  46  42  77  57 100  84  68  48  72
50
  72  55  72  77  56  94  67  82  75  80  60  73  74  62  53  69  75
60
  58  71  87  74  87  73  78  76  74  55  94  71  76  59  91  57  83
59
  93  64  58  79  96  76  64  70  80  33  95  64  92  34  72  81  57
79
  84  82  54  45  54  62  49  74  59  63  83  62  72  72  65  65  54
78
  82  85  74  83  71  83  77  66  75  52  68  84  67  70  41  91  46
```

```
58
  67  70  83  64 100  49  77  57  67  80  74  41  67  59  86  88  57
80
  58  52  31  84  97  71  62  58  71  41  66 100  51  35  81  94  72
38
  82  79  55  75  90  95  65  39  85  86  54  93  69  84  78  58  73
60
  44  67  69  55  59  88  42  78  84  68  66  51  43  38  69  90  73
67
  57  81  63  80  78  65  74  80  60  60  63  64  72  51  71  63  82
76
  39  79  48  70  90  73  58 100  80  75  72  79  52  56  65  45  59
61
  47  62  83  90  76  72  69  57  56  40  79  48  57  47  78  45  74
69
  59  85  45  54  72  74  75  55  49  53  83  22 100  67  83  46  43
74
  64  35  67  87  77  91  74  96  82  78  73  52  91  66  67  71  74
71
  61  47  76  85  93  41  81  86  53  91  68  96  48  71  75  72  71
62
  67  53  74  63  82  57  69  52  91  73  73  75  36  71  62 100  50
74
  60  75  83  83 100  67  71  77  67  95  52  71  74  60  67  79  75
95
  69  80  48  61  82  39  70  70  69  32  79  53  59  83 100  80  80
82
  56  83  85  88  81  95  63  70  89  59  56  62  95  63  82  69  58
74
  66  82  94  70  78  63  91  70  62  79  65  74  56  65 100  70  66
54
  72  90  56  65  50  95  38  76  84  76  55  85  70  73  80  83  53
67
 100  67  44  96  48  77 100  40  91  55  41  25  63  59  63  77  46
49
  46  93  39  58  87  57  77 100  65  34  87  81  63  69  74  70  93
63
  81  81  63  87  76  54  89  63  76  79  75  50  36  82  83  85  82
41
  82  45  57  88  81  98  61  95  84  71  52  71  90  75  62  63  86
70
  77  68  80  67  67  89  60  79  80  78  70  72  43  14  54  92  71
65
  58  56  67  64  81  55  45  86  52  75  81  62  42  21  72  55  66
69
  86  67  78  85  66  47 100  63  62  61  69  57  76  52  47  51  61
45
  59  81  65  53  61  90  74  62  67  50  84  70  52  92  65  65  67
72
```

```
  66  62  99  62  53  57  78  56  87  79  63  87  86  75  70  60  49
41
  78  58  75  89  34  60  80  85  73  58  69  74  52  58  79  86  61
68
  67  48  65  73  57  73  57  80  85  81  61  69 100  99  92  72  57
44
  59  62  93  64  57  72  40  85  60  83  63  74  44  61  74  68  78
50
  70  68  82  46  96 100  44  41  95  79  67  52  87  75  61  42  60
57
  64  52  68  58  93  75  77  66  63  90  43  65  95  86  31  95  52
63
  87  70  59  84  79  77  75  66  69  85  63  50  58  80  47  55  61
87
  77  54  66  68  54  69  74  81  72  61  76  63  64  73  62  92  69
70
  65  53  74  61  80  85  62  80  83  56  76  52  51  74  57  63  61
87
  60  54  89  67  56  70  90  94  78  72]
Weights Matrix (W):
 [0. 0.]

# Step 1: Split the dataset
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size =
0.2, random_state = 42)

# Step 2: Verify the split
print("\nTraining Features (X_train):")
print(X_train.head())

print("\nTraining Labels (Y_train):")
print(Y_train.head())

print("\nTesting Features (X_test):")
print(X_test.head())

print("\nTesting Labels (Y_test):")
print(Y_test.head())


Training Features (X_train):
     Math  Reading
29     64       82
535    62       70
695    36       21
557    81       70
836    82       86

Training Labels (Y_train):
29     78
```

```
535    67
695    25
557    71
836    87
Name: Writing, dtype: int64

Testing Features (X_test):
     Math  Reading
521    63       69
737    42       37
740    69       62
660    69       59
411   100       92

Testing Labels (Y_test):
521    69
737    41
740    57
660    56
411    88
Name: Writing, dtype: int64
```

```python
#Define the cost function
def cost_function(X, Y, W):
  """ Parameters:
  This function finds the Mean Square Error.
  Input parameters:
  X: Feature Matrix
  Y: Target Matrix
  W: Weight Matrix
  Output Parameters:
  cost: accumulated mean square error.
  """
  n = len(Y)
  Y_pred = np.dot(X, W)
  cost = (1/n) * np.sum((Y_pred - Y) ** 2)
  return cost

def gradient_descent(X, Y, W, alpha, iterations):
  """
  Perform gradient descent to optimize weights.
  Parameters:
  X: Feature matrix (n_samples x n_features)
  Y: Target vector (n_samples)
  W: Weight vector (n_features)
  alpha: Learning rate
  iterations: Number of iterations

  Returns:
  W_update: Optimized weights
```

```
  cost_history: History of cost function values
  """
  cost_history = []
  m = len(Y)

  for i in range(iterations):
    # Step 1: Compute predictions
    Y_pred = np.dot(X, W)

    # Step 2: Compute gradient
    gradient = (1/m) * np.dot(X.T, (Y_pred - Y))

    # Step 3: Update weights
    W -= alpha * gradient

    # Step 4: Compute and store cost
    cost = cost_function(X, Y, W)
    cost_history.append(cost)

  return W, cost_history

# RMSE Calculation
def rmse(Y, Y_pred):
  """
  Calculate Root Mean Squared Error.
  """
  return np.sqrt(np.mean((Y - Y_pred) ** 2))

# R-Squared Calculation
def r2(Y, Y_pred):
  """
  Calculate R-squared value.
  """

  mean_y = np.mean(Y)
  ss_tot = np.sum((Y - mean_y) ** 2)
  ss_red = np.sum((Y - Y_pred) ** 2)
  return 1 - (ss_red / ss_tot)

def main():
  data = pd.read_csv('/content/drive/MyDrive/Concept of AI -- week
5/student.csv')

  # Step 2: Split the data into features (X) and target (Y)
  X = data[['Math', 'Reading']].values
  Y = data['Writing'].values

  #Step 3: Split the data into training and test sets
  X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=0.2, random_state = 42)
```

```
  # Step 4: Initialize weights (W), learning rate, and iterations
  W = np.zeros(X.shape[1])
  alpha = 0.00001
  iterations = 1000

  # Step 5: Perform Gradient Descent
  W_optimal, cost_history = gradient_descent(X_train, Y_train, W,
alpha, iterations)

  # Step 6: Make predictions on the test set
  Y_pred = np.dot(X_test, W_optimal)

  # Step 7: Evaluate the model using RMSE and R-Squared
  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 the Set:", model_rmse )
  print("R-squared on the Set:",model_r2)

# Execute the main function
if __name__ == "__main__":
    main()
```

```
Final Weights: [0.34811659 0.64614558]
Cost History (First 10 iterations): [4026.33114156751,
3280.573665199384, 2674.1239989803175, 2180.9589785701155,
1779.9166540166468, 1453.788198601909, 1188.5794521617188,
972.910410590327, 797.5268927198968, 654.9034294649376]
RMSE on the Set: 5.2798239764188635
R-squared on the Set: 0.8886354462786421
```

1. Model Performance: Overfitting, Underfitting, or Acceptable Performance? Key Metrics to Evaluate Performance:

Root Mean Square Error (RMSE):

-- Low RMSE indicates the model predicts the target values well.

R-squared (R^2):

-- A value close to 1 implies the model explains most of the variance in the target variable.

Scenario Evaluation:

- Overfitting:

-- If R^2 on the training set is significantly higher than R^2 on the test set, the model memorizes the training data and fails to generalize.

- Underfitting:

-- Both training and test R^2 are low, indicating the model is too simple to capture the relationship.

- Acceptable Performance:

-- The model achieves a similar R^2 score for both training and test sets, with a low RMSE.

Observation from the Results:

-- If the RMSE on the test set is small (e.g., ~4.57) and the R^2 is high (~0.85), it suggests the model performs well and generalizes effectively. Comparing training and test performance metrics (which can be included in the main function) will confirm whether the model overfits, underfits, or is acceptable.

1. Experimenting with Learning Rate: Impact of Learning Rate (α):

Higher Learning Rate: Convergence may happen faster, but too high a learning rate can cause the model to diverge or oscillate. Lower Learning Rate: The model converges slowly and may require more iterations, but it ensures stability. Experimentation:

Learning Rate (α) = 0.1: Likely to converge faster but might overshoot the minimum or diverge. Learning Rate (α) = 0.0001: Likely stable but may take many iterations to converge. Default Learning Rate (α = 0.00001): A balanced learning rate providing stable and accurate convergence.

```
# Set hyperparameters
iterations = 1000  # Define the number of iterations for gradient
descent

# Experiment with Learning Rates
for alpha in [0.1, 0.01, 0.0001, 0.00001]:
    W_optimal, cost_history = gradient_descent(X_train, Y_train,
np.zeros(X_train.shape[1]), 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(f"Learning Rate: {alpha}")
    print(f"RMSE on Test Set: {model_rmse}")
    print(f"R-Squared on Test Set: {model_r2}")
    print("-" * 40)
```

```
<ipython-input-15-e64a3002b6cd>:26: RuntimeWarning: invalid value
encountered in subtract
  W -= alpha * gradient

Learning Rate: 0.1
RMSE on Test Set: nan
R-Squared on Test Set: 1.0
----------------------------------------
```

```
/usr/local/lib/python3.10/dist-packages/numpy/core/_methods.py:49:
RuntimeWarning: overflow encountered in reduce
  return umr_sum(a, axis, dtype, out, keepdims, initial, where)
<ipython-input-15-e64a3002b6cd>:26: RuntimeWarning: invalid value
encountered in subtract
  W -= alpha * gradient

Learning Rate: 0.01
RMSE on Test Set: nan
R-Squared on Test Set: 1.0
----------------------------------------
Learning Rate: 0.0001
RMSE on Test Set: 4.792607360540954
R-Squared on Test Set: 0.908240340333986
----------------------------------------
Learning Rate: 1e-05
RMSE on Test Set: 5.2798239764188635
R-Squared on Test Set: 0.8886354462786421
----------------------------------------
```