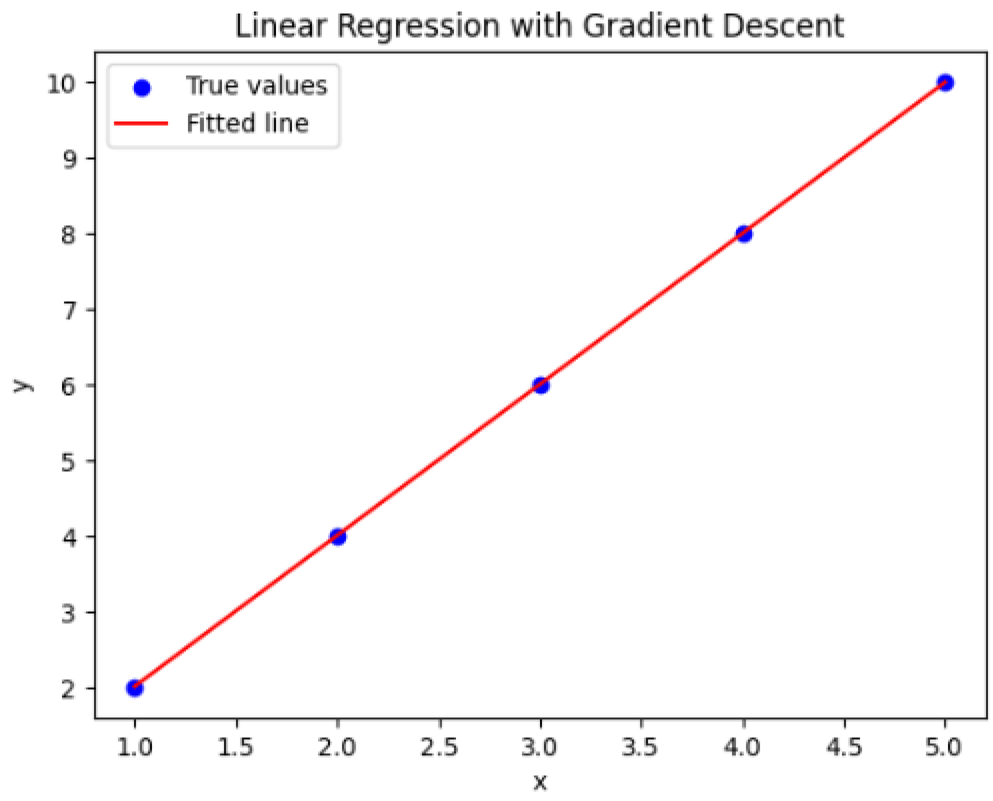
Experiment 2: Implementation of Gradient Descent for a Simple Linear Regression Problem

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

x = np.array([1, 2, 3, 4, 5])
y = np.array([2, 4, 6, 8, 10])

w = 0
b = 0

learning_rate = 0.01
num_iterations = 1000

for i in range(num_iterations):
    y_pred = w * x + b
    error = y_pred - y

    dw = (2/len(x)) * np.sum(error * x)
    db = (2/len(x)) * np.sum(error)

    w = w - learning_rate * dw
    b = b - learning_rate * db

    if i % 100 == 0:
        print(f"Iteration {i}: w = {w:.4f}, b = {b:.4f}, MSE =
{np.mean(error**2):.4f}")

plt.scatter(x, y, color='blue', label='True values')
plt.plot(x, w * x + b, color='red', label='Fitted line')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Linear Regression with Gradient Descent')
plt.legend()
plt.show()

print(f"Final weight (w) = {w:.4f}, Final bias (b) = {b:.4f}")
```

```
Iteration 0: w = 0.4400, b = 0.1200, MSE = 44.0000
Iteration 100: w = 1.8988, b = 0.3655, MSE = 0.0245
Iteration 200: w = 1.9279, b = 0.2605, MSE = 0.0124
Iteration 300: w = 1.9486, b = 0.1856, MSE = 0.0063
Iteration 400: w = 1.9634, b = 0.1323, MSE = 0.0032
Iteration 500: w = 1.9739, b = 0.0943, MSE = 0.0016
Iteration 600: w = 1.9814, b = 0.0672, MSE = 0.0008
Iteration 700: w = 1.9867, b = 0.0479, MSE = 0.0004
Iteration 800: w = 1.9905, b = 0.0341, MSE = 0.0002
Iteration 900: w = 1.9933, b = 0.0243, MSE = 0.0001
```



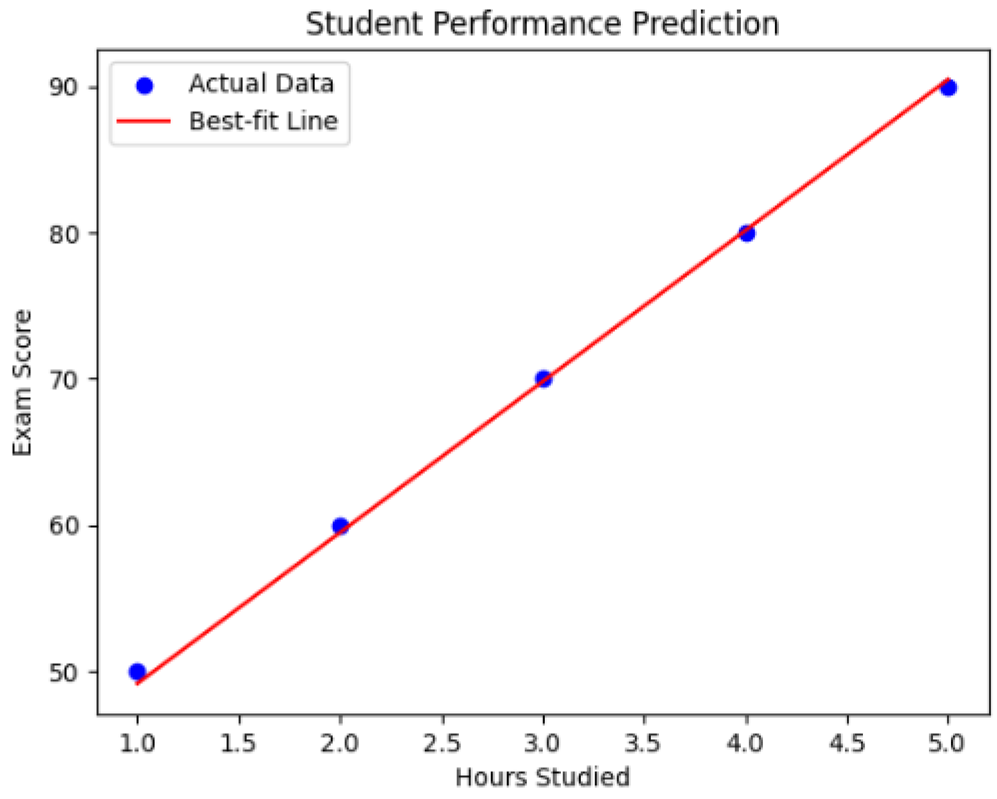
```
Final weight (w) = 1.9952, Final bias (b) = 0.0174
```

Q1: Student Performance Prediction:-

```python
#Q1: Student Performance Prediction:-
import numpy as np
import matplotlib.pyplot as plt

x = np.array([1, 2, 3, 4, 5])  # Study hours
y = np.array([50, 60, 70, 80, 90])  # Exam scores

# Initialize parameters
m = 0
b = 0
```

```
learning_rate = 0.01
iterations = 1000

# Number of data points
n = len(x)

# Gradient Descent implementation
for _ in range(iterations):
    # Predictions
    y_pred = m * x + b

    # Calculate gradients
    dm = -(2 / n) * np.sum(x * (y - y_pred))
    db = -(2 / n) * np.sum(y - y_pred)

    # Update parameters
    m -= learning_rate * dm
    b -= learning_rate * db

# Final parameters
print(f"Final slope (m): {m:.2f}")
print(f"Final intercept (b): {b:.2f}")

# Predict exam scores based on study hours
def predict(hours):
    return m * hours + b

# Visualize the dataset and the best-fit line
plt.scatter(x, y, color='blue', label='Actual Data')
plt.plot(x, predict(x), color='red', label='Best-fit Line')
plt.xlabel('Hours Studied')
plt.ylabel('Exam Score')
plt.title('Student Performance Prediction')
plt.legend()
plt.show()

# Example prediction
study_hours = 7
predicted_score = predict(study_hours)
print(f"Predicted exam score for studying {study_hours} hours daily:
{predicted_score:.2f}")
```

```
Final slope (m): 10.32
Final intercept (b): 38.83
```

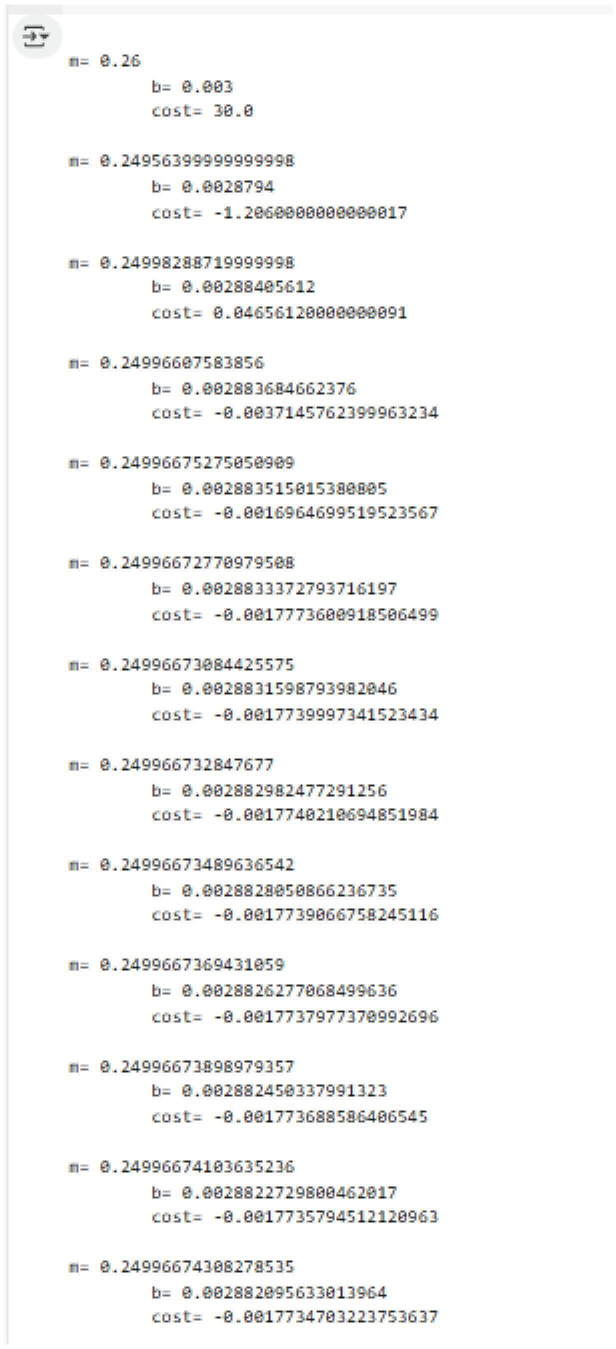


```
Predicted exam score for studying 7 hours daily: 111.10
```

Q2: Temperature Prediction

```
#Q2: From Altitude Tempreture prediction:-
import numpy as np
import matplotlib.pyplot as plt

x = np.array([0,20,40,60,80,100,120])  # Altitude
y = np.array([0,5,10,15,20,25,30])  # Tempreture

# Initialize parameters
m = 0
b = 0
learning_rate = 0.0001
iterations = 100

# Number of data points
n = len(x)

# Gradient Descent implementation
```

```
for iterations in range(iterations):
    # Predictions
  y_pred = m * x + b

    # Calculate gradients
  cost=(1/n) * sum((y-y_pred) * 2 )
  dm = -(2 / n) * sum(x * (y - y_pred))
  db = -(2 / n) * sum(y - y_pred)

    # Update parameters
  m -= learning_rate * dm
  b -= learning_rate * db
  print("\nm=",m)
  print("\tb=",b)
  print("\tcost=",cost)
# Predict exam scores based on study hours
def predict(temp):
    return m * temp + b


plt.scatter(x, y, color='blue', label='Actual Data')
plt.plot(x, predict(x), color='red', label='Best-fit Line')
plt.xlabel('Altitude')
plt.ylabel('Tempreture')
plt.title('Altitude Tempreture predictions')
plt.legend()
plt.show()

# Example
altitude = 110
predicted_temp = predict(altitude)
print("\nfinal output : ",predicted_temp)
```



```
m= 0.26
        b= 0.003
        cost= 30.0

m= 0.24956399999999998
        b= 0.0028794
        cost= -1.2060000000000017

m= 0.24998288719999998
        b= 0.00288405612
        cost= 0.04656120000000091

m= 0.24996607583856
        b= 0.002883684662376
        cost= -0.0037145762399963234

m= 0.24996675275050909
        b= 0.0028835150153808005
        cost= -0.0016964699519523567

m= 0.24996672770979508
        b= 0.0028833372793716197
        cost= -0.0017773600918506499

m= 0.24996673084425575
        b= 0.0028831598793982046
        cost= -0.0017739997341523434

m= 0.249966732847677
        b= 0.002882982477291256
        cost= -0.0017740210694851984

m= 0.24996673489636542
        b= 0.0028828050866236735
        cost= -0.0017739066758245116

m= 0.2499667369431059
        b= 0.0028826277068499636
        cost= -0.0017737977370992696

m= 0.24996673898979357
        b= 0.002882450337991323
        cost= -0.001773688586406545

m= 0.24996674103635236
        b= 0.0028822729800462017
        cost= -0.0017735794512120963

m= 0.24996674308278535
        b= 0.002882095633013964
        cost= -0.0017734703223753637
```

Q3: House Price Prediction

```
#Q3: From Square feet predict House values:-
import numpy as np
import matplotlib.pyplot as plt


x = np.array([800,850,900,950,1000])  # Square feet values
y = np.array([16000,17000,18000,19000,20000])  # House values
```

```
# Initialize parameters
m = 0
b = 0
learning_rate = 0.000001
iterations = 110

# Number of data points
n = len(x)

# Gradient Descent implementation
for iterations in range(iterations):
    # Predictions
  y_pred = m * x + b

    # Calculate gradients
  cost=(1/n) * sum((y-y_pred) * 2 )
  dm = -(2 / n) * sum(x * (y - y_pred))
  db = -(2 / n) * sum(y - y_pred)

    # Update parameters
  m -= learning_rate * dm
  b -= learning_rate * db
  print("\nm=",m)
  print("\tb=",b)
  print("\tcost=",cost)
# Predict exam scores based on study hours
def predict(house):
    return m * house + b

# Visualize the dataset and the best-fit line
plt.scatter(x, y, color='blue', label='Actual Data')
plt.plot(x, predict(x), color='red', label='Best-fit Line')
plt.xlabel('Square Feet Values')
plt.ylabel('House values')
plt.title('House Value predictions')
plt.legend()
plt.show()

# Example prediction
Square = 920
predicted_house = predict(Square)
print("\nfinal output : ",predicted_house)
```

```
m= 32.6
        b= 0.036
        cost= 36000.0

m= 12.0619352
        b= 0.013319927999999998
        cost= -22680.072

m= 25.000956848129597
        b= 0.02760841800014399
        cost= 14288.490000143996

m= 16.84934749052595
        b= 0.018606640456674713
        cost= -9001.777543469278

m= 21.98487758901583
        b= 0.024277777760447088
        cost= 5671.137303772376

m= 18.749483418920057
        b= 0.020704949544663072
        cost= -3572.8282157840167

m= 20.787788177171183
        b= 0.022955837980070788
        cost= 2250.8884360448073

m= 19.503652127873792
        b= 0.021537773350l2379
        cost= -1418.06463058409

m= 20.31266039144748
        b= 0.022431156444440426
        cost= 893.3830942804715

m= 19.802983577306485
        b= 0.021868322877485906
        cost= -562.8335669183551

m= 20.124080983315736
        b= 0.022222908701688478
        cost= 354.58582420257227

m= 19.921788979275423
        b= 0.021999518485902 75
        cost= -223.3902157857272

m= 20.04923334392321
        b= 0.02214025432417002
        cost= 140.735838267269
```