SUPERVISED MACHINE LEARNING

LINEAR REGRESSION

PAMPATTIWAR SHREEVASTHAV

LINEAR REGRESSION

Step	Process
1	Define model (y = mx + b)
2	Compute loss (MSE)
3	Compute gradients of loss
4	Update parameters
5	Repeat for many epochs

LINE BY LINE CODE EXECUTION

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt
```

- pandas → Used for handling datasets (reading CSV, manipulating tables, etc.).
- numpy → Used for numerical operations (arrays, linear algebra, etc.).
- matplotlib.pyplot → Used for plotting graphs and visualizations.

READ DATA

- data = pd.read_csv('studytime_scores.csv')

- Reads the CSV file studytime_scores.csv into a pandas DataFrame called data.
- The dataset contains two main columns:StudyTime_Hours → Number of hours a student studied (< 9000 hours).
- ExamScore → Score obtained (< 300).

DATA NORMALIZATION

```
data['StudyTime_Hours_Norm'] = data['StudyTime_Hours’] /
                               data['StudyTime_Hours'].max()
```

- Creates a new column StudyTime_Hours_Norm by dividing each value by the maximum study hours.
- This normalizes the data into the range [0,1], which helps gradient descent converge faster.

LOSS FUNCTION

```
def loss_function(m, b, points):
    total_error = 0
    for i in range(len(points)):
        x = points.iloc[i].StudyTime_Hours_Norm
        y = points.iloc[i].ExamScore
        total_error += (y - (m*x + b))**2
    return total_error / float(len(points))
```

This function calculates Mean Squared Error (MSE).m → slope, b → intercept, points → dataset.

For each data point:y - (m*x + b) → difference between actual and predicted value.

Square it (y - (m*x + b))**2

Sum all squared errors, divide by number of points → average error.

GRADIENT DESCENT

```
def gradient_descent(m_now, b_now, points, L):
    m_gradient = 0
    b_gradient = 0
    n = len(points)
    for i in range(n):
        x = points.iloc[i].StudyTime_Hours_Norm
        y = points.iloc[i].ExamScore
        m_gradient += -(2/n) * x * (y - (m_now*x + b_now))
        b_gradient += -(2/n) * (y - (m_now*x + b_now))
    m = m_now - L * m_gradient
    b = b_now - L * b_gradient
    return m, b
```

Computes gradients of the loss function with respect to m and b.

-(2/n) * x * (y - (m*x + b)) → partial derivative of loss w.r.t m.

-(2/n) * (y - (m*x + b)) → partial derivative w.r.t b.

L → learning rate; controls how big the step is in each update.

Updates parameters: =

m= m_now - L * m_gradient

b = b_now - L * b_gradient

Returns the updated m and b.

INITIALIZING PARAMETERS

m = 0

b = 0

L = 0.01

epochs = 2000

Start with slope m=0 and intercept b=0.

Learning rate L=0.01 → small enough for stable convergence.

epochs=2000 → number of iterations gradient descent will run.

TRAINING THE MODEL

```
loss_history = []

for i in range(epochs+1):
    m, b = gradient_descent(m, b, data, L)
    current_loss = loss_function(m, b, data)
    loss_history.append(current_loss)
    if i % 200 == 0:
        print(f"Epoch {i}: m={m:.6f}, b={b:.3f}, loss={loss_function(m,b,data):.3f}")
```

Loop runs gradient descent 2000 times.

Each iteration:Update m and b using gradients.

Compute current loss and store in loss_history.

Every 200 epochs, print current slope, intercept, and loss.

FINAL MODEL

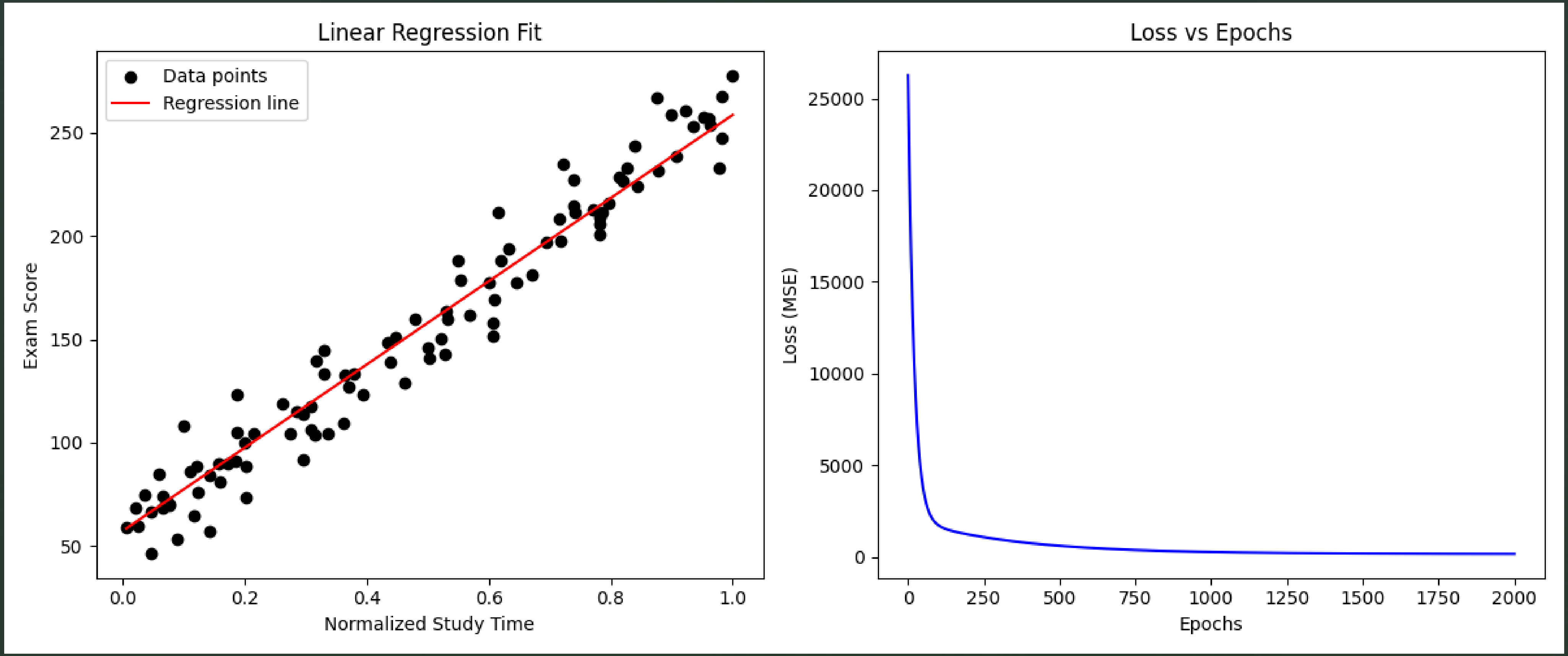
```
print(f"\nFinal model: y = {m:.6f}x + {b:.3f}")
```

- Prints the **final linear regression equation** after training.

PLOTTING RESULTS

```
plt.figure(figsize=(12,5))

plt.subplot(1, 2, 1)

plt.scatter(data.StudyTime_Hours_Norm, data.ExamScore,
color="black", label="Data points")

x_range = np.linspace(data.StudyTime_Hours_Norm.min(),
data.StudyTime_Hours_Norm.max(), 100)

plt.plot(x_range, m*x_range + b, color="red",
label="Regression line")

plt.xlabel("Normalized Study Time")

plt.ylabel("Exam Score")

plt.title("Linear Regression Fit")

plt.legend()

plt.subplot(1, 2, 2)

plt.plot(range(epochs+1), loss_history,
color='blue')

plt.xlabel("Epochs")

plt.ylabel("Loss (MSE)")

plt.title("Loss vs Epochs")

plt.tight_layout()

plt.show()
```

OUTPUT



Linear Regression Fit
Data points
Regression line
Exam Score
250
200
150
100
50
0.0
0.2
0.4
0.6
0.8
1.0
Normalized Study Time
Loss vs Epochs
Loss (MSE)
25000
20000
15000
10000
5000
0
0
250
500
750
1000
1250
1500
1750
2000
Epochs

HOW MODEL GOT TRAINED ?

Epoch 0: m=1.834035, b=3.059, loss=26261.503

Epoch 200: m=101.235673, b=107.721, loss=1241.455

Epoch 400: m=128.694817, b=94.626, loss=775.218

Epoch 600: m=148.952100, b=84.229, loss=514.378

Epoch 800: m=164.118378, b=76.441, loss=368.133

Epoch 1000: m=175.474566, b=70.609, loss=286.138

Epoch 1200: m=183.977848, b=66.242, loss=240.165

Epoch 1400: m=190.344932, b=62.972, loss=214.390

Epoch 1600: m=195.112475, b=60.524, loss=199.938

Epoch 1800: m=198.682314, b=58.691, loss=191.836

Epoch 2000: m=201.355337, b=57.318, loss=187.293

Final model: y = 201.355337x + 57.318