Data Science Lab

CSEL-42--

Assignment on Linear Regression

Submitted by

NISHAT MAHMUD
ID: B190305003

FAHIM HASAN
ID: B190305029

MD. WALIUL ISLAM RAYHAN
ID: B190305034

Submitted To

MD. MANOWARUL ISLAM, PHD
Associate Professor
Dept. of CSE
Jagannath University, Dhaka - 1100

Task 1

Exploring Key Parameters and Model Evaluation

The task 1 of the assignment performs linear regression on a synthetic dataset to model the relationship between an independent variable X and a dependent variable Y. The process includes data visualization, implementation of a linear regression model, evaluation using metrics, and analysis of residuals. The steps are described below:

1. Dataset Generation

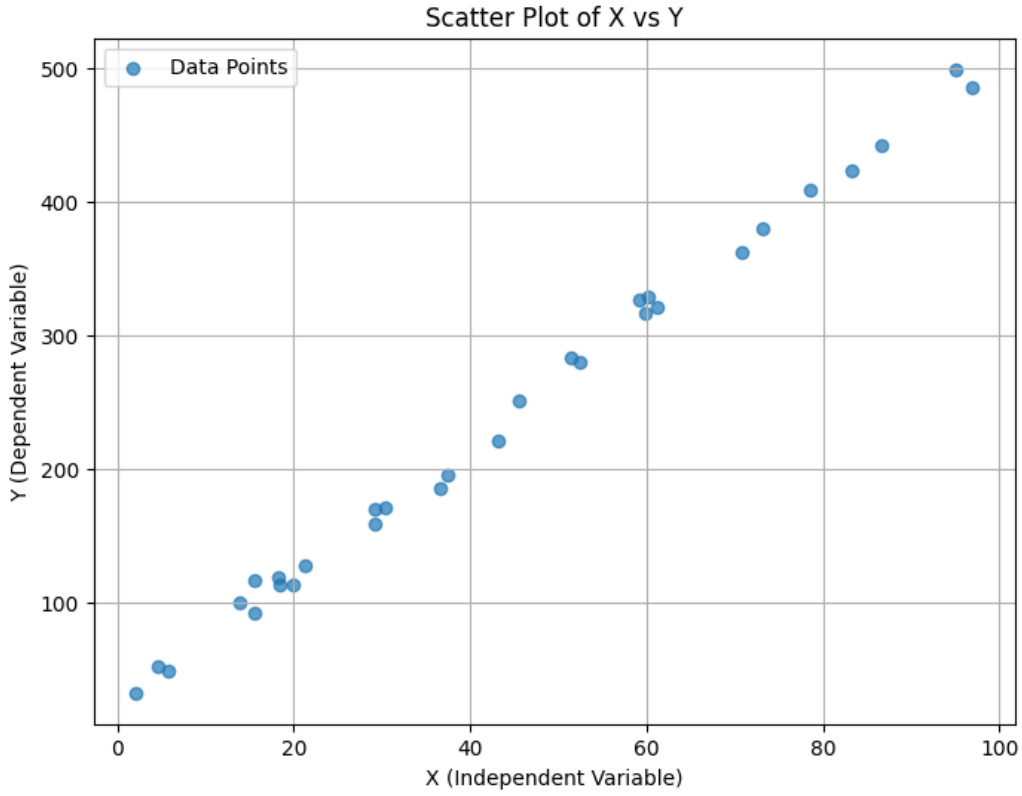
The dataset is synthetically generated to simulate a linear relationship between X (independent variable) and Y (dependent variable). Noise is added to Y to mimic real-world variability.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error

# Generate synthetic dataset
np.random.seed(42)  # For reproducibility
X = np.random.uniform(0, 100, 30)  # Independent variable
noise = np.random.normal(0, 10, X.shape)  # Add noise
Y = 5 * X + 20 + noise  # Dependent variable (linear relationship with noise)
dataset = pd.DataFrame({'X': X, 'Y': Y})
```

2. Data Visualization

The scatter plot provides a visual representation of the relationship between X and Y. This helps in understanding the spread and potential correlation between the two variables.

```
plt.figure(figsize=(8, 6))
plt.scatter(dataset['X'], dataset['Y'], alpha=0.7, label='Data Points')
plt.title('Scatter Plot of X vs Y')
plt.xlabel('X (Independent Variable)')
plt.ylabel('Y (Dependent Variable)')
plt.legend()
plt.grid()
plt.show()
```



3. Linear Regression Implementation

The linear regression model is implemented using the `LinearRegression` class from the `scikit-learn` library. The model is trained on the dataset, and the slope (m) and intercept (c) of the regression line are extracted.

```
# Reshape data for model fitting
X = dataset[['X']].values
Y = dataset['Y'].values

# Initialize and fit the model
model = LinearRegression()
model.fit(X, Y)

# Extract slope and intercept
slope = model.coef_[0]
intercept = model.intercept_
```

4. Coefficient of Determination (R^2)

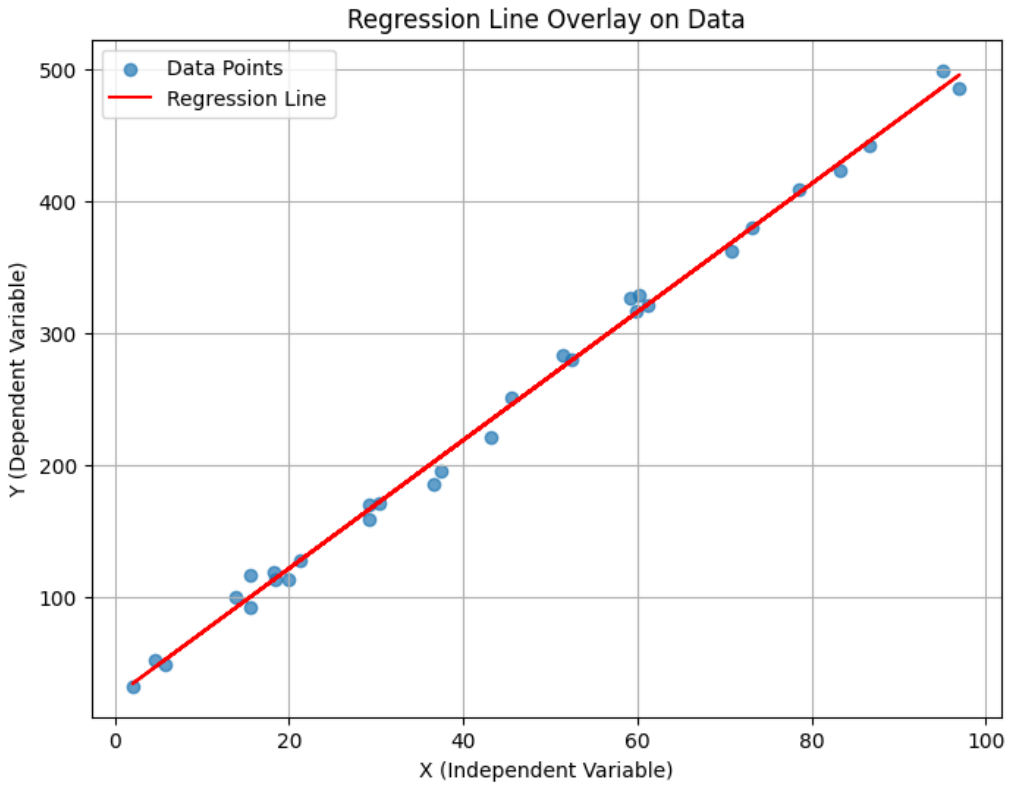
The coefficient of determination (R^2) is calculated to evaluate the goodness-of-fit of the regression model. It indicates the proportion of variance in Y that can be explained by X.

```
# Predict Y values using the fitted model
predicted_Y = model.predict(X)

# Calculate R²
r2 = r2_score(Y, predicted_Y)
```

5. Predictions and Visualization

The regression line is overlaid on the scatter plot of the original data to visualize the linear fit of the model.

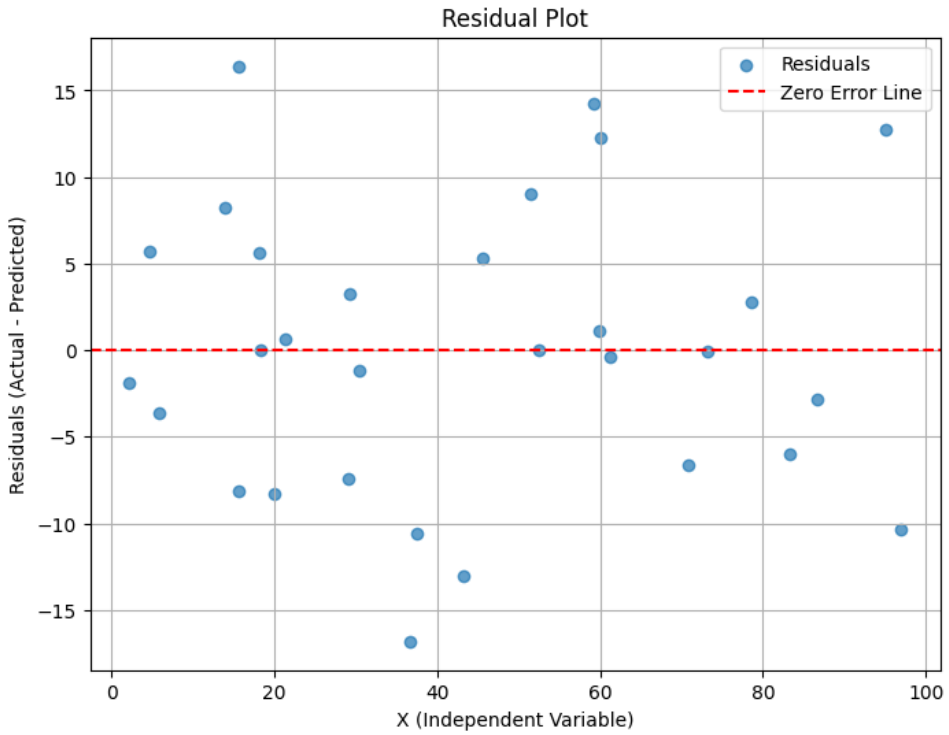
```
plt.figure(figsize=(8, 6))
plt.scatter(dataset['X'], dataset['Y'], alpha=0.7, label='Data Points')
plt.plot(dataset['X'], predicted_Y, color='red', label='Regression Line')
plt.title('Regression Line Overlay on Data')
plt.xlabel('X (Independent Variable)')
plt.ylabel('Y (Dependent Variable)')
plt.legend()
plt.grid()
plt.show()
```



6. Residual Analysis

Residuals (the differences between actual and predicted Y values) are plotted to check for patterns. A random distribution of residuals around zero suggests that the model assumptions are valid.

```
residuals = Y - predicted_Y
plt.figure(figsize=(8, 6))
plt.scatter(dataset['X'], residuals, alpha=0.7, label='Residuals')
plt.axhline(0, color='red', linestyle='--', label='Zero Error Line')
plt.title('Residual Plot')
plt.xlabel('X (Independent Variable)')
plt.ylabel('Residuals (Actual - Predicted)')
plt.legend()
plt.grid()
plt.show()
```



7. Model Evaluation Metrics

Three key metrics are calculated to evaluate the performance of the model:

- **Mean Absolute Error (MAE):** The average of absolute errors between actual and predicted values.
- **Mean Squared Error (MSE):** The average of squared errors.
- **Root Mean Squared Error (RMSE):** The square root of the MSE, providing a measure of error in the same units as the dependent variable.

```python
mae = mean_absolute_error(Y, predicted_Y)
mse = mean_squared_error(Y, predicted_Y)
rmse = np.sqrt(mse)

# Print results
results = {
    "Slope (m)": slope,
    "Intercept (c)": intercept,
    "R²": r2,
    "Mean Absolute Error (MAE)": mae,
    "Mean Squared Error (MSE)": mse,
    "Root Mean Squared Error (RMSE)": rmse
}

for key, value in results.items():
    print(f"{key}: {value:.3f}")
```

```
Slope (m): 4.861
Intercept (c): 24.290
R²: 0.996
Mean Absolute Error (MAE): 6.487
Mean Squared Error (MSE): 67.166
Root Mean Squared Error (RMSE): 8.196
```

The linear regression model successfully fits the synthetic data, as evidenced by the high R^2 value and low error metrics. The residual analysis confirms that the model assumptions hold true, with residuals distributed randomly around zero.

Task 2

Gradient Descent

The task 2 of the assignment implements a linear regression model using gradient descent to optimize the parameters (θ_0, θ_1). The process includes dataset preparation, gradient descent implementation, cost function computation, visualization of results, and model evaluation. Each step is explained in detail below:

1. Dataset Preparation

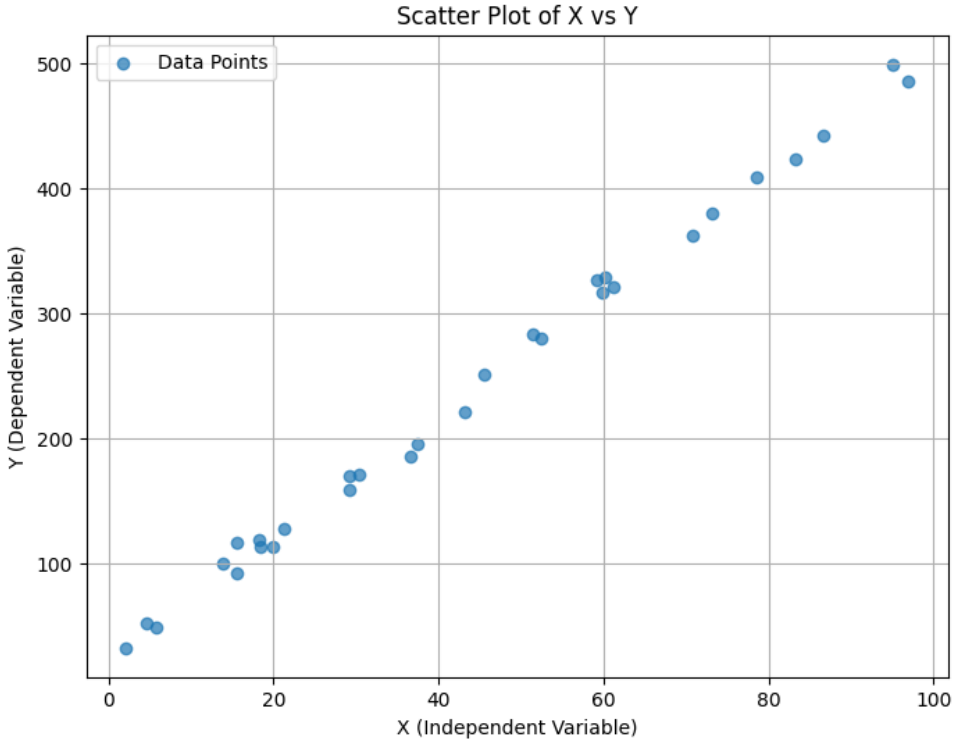
A synthetic dataset is generated to simulate a linear relationship between X (independent variable) and Y (dependent variable). Noise is added to Y to mimic real-world scenarios.

```python
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt

# Generate synthetic dataset
np.random.seed(42)
X = np.random.uniform(0, 100, 30)  # Independent variable
noise = np.random.normal(0, 10, X.shape)  # Add noise
Y = 5 * X + 20 + noise  # Dependent variable (linear relationship with noise)
```

2. Data Visualization

A scatter plot is created to visualize the relationship between X and Y, providing insights into the data distribution.

```python
# Visualize the dataset using a scatter plot
plt.figure(figsize=(8, 6))
plt.scatter(X, Y, alpha=0.7, label="Data Points")
plt.title("Scatter Plot of X vs Y")
plt.xlabel("X (Independent Variable)")
plt.ylabel("Y (Dependent Variable)")
plt.legend()
plt.grid()
plt.show()
```



3. Cost Function Definition

The Mean Squared Error (MSE) is defined as the cost function to measure the performance of the regression model. It quantifies the difference between actual and predicted values.

```python
# Define the Mean Squared Error (MSE) as the cost function
def compute_cost(X, Y, theta0, theta1):
    m = len(Y)  # Number of data points
    predictions = theta0 + theta1 * X  # Predicted values
    errors = predictions - Y  # Residuals (differences)
    cost = (1 / (2 * m)) * np.sum(errors**2)  # Mean squared error
    return cost
```

4. Gradient Descent Implementation

The gradient descent algorithm is implemented to iteratively update the parameters θ_0 and θ_1 by minimizing the cost function.

```python
# Implement Gradient Descent Algorithm
def gradient_descent(X, Y, theta0, theta1, alpha, iterations):
    m = len(Y)  # Number of data points
    cost_history = []  # Store cost at each iteration

    for _ in range(iterations):
        predictions = theta0 + theta1 * X  # Predicted values
        errors = predictions - Y  # Residuals
        # Compute gradients
        grad_theta0 = (1 / m) * np.sum(errors)
        grad_theta1 = (1 / m) * np.sum(errors * X)
        # Update parameters
        theta0 -= alpha * grad_theta0
        theta1 -= alpha * grad_theta1
        # Store the cost
        cost_history.append(compute_cost(X, Y, theta0, theta1))

    return theta0, theta1, cost_history
```

5. Perform Gradient Descent

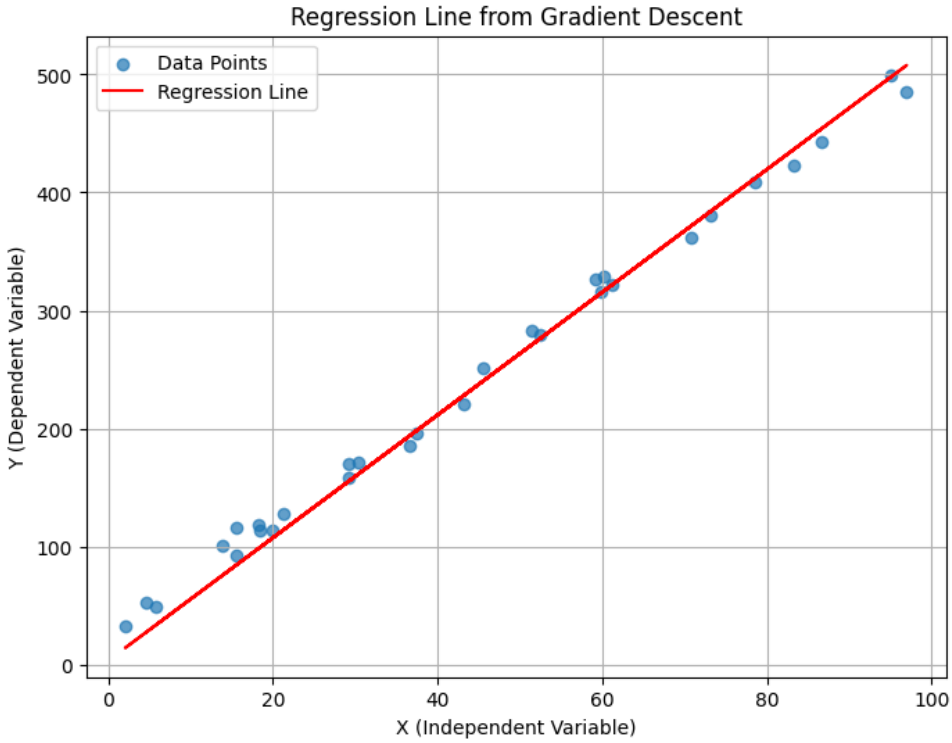
The gradient descent function is called with the initial parameters, learning rate, and number of iterations. This optimizes the parameters θ_0 and θ_1.

```python
# Initialize parameters and perform gradient descent
theta0_initial = 0  # Initial intercept
theta1_initial = 0  # Initial slope
alpha = 0.0005  # Learning rate
iterations = 1000  # Number of iterations

# Optimize parameters
theta0, theta1, cost_history = gradient_descent(X, Y, theta0_initial, theta1_initial, alpha, iterations)
```

6. Predictions and Regression Line Visualization

The optimized parameters are used to predict Y values, and the regression line is plotted over the original scatter plot to visualize the fit.

```python
# Predict values using optimized parameters
predicted_Y = theta0 + theta1 * X

# Visualize regression line overlay on scatter plot
plt.figure(figsize=(8, 6))
plt.scatter(X, Y, alpha=0.7, label="Data Points")
plt.plot(X, predicted_Y, color="red", label="Regression Line")
plt.title("Regression Line from Gradient Descent")
plt.xlabel("X (Independent Variable)")
plt.ylabel("Y (Dependent Variable)")
plt.legend()
plt.grid()
plt.show()
```



7. Model Evaluation

The model is evaluated using performance metrics: Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Coefficient of Determination (R^2).

```python
# Compute model evaluation metrics
mae = np.mean(np.abs(Y - predicted_Y))  # Mean Absolute Error
mse = np.mean((Y - predicted_Y)**2)  # Mean Squared Error
rmse = np.sqrt(mse)  # Root Mean Squared Error
r2 = 1 - (np.sum((Y - predicted_Y)**2) / np.sum((Y - np.mean(Y))**2))  # R²

# Print Results
print("Theta0 (Intercept):", theta0)
print("Theta1 (Slope):", theta1)
print("Final Cost (MSE):", cost_history[-1])
print("Mean Absolute Error (MAE):", mae)
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("R²:", r2)
```

```
Theta0 (Intercept): 3.311198001564402
Theta1 (Slope): 5.20218954784934
Final Cost (MSE): 96.54097401518683
Mean Absolute Error (MAE): 11.480326346954957
Mean Squared Error (MSE): 193.08194803037367
Root Mean Squared Error (RMSE): 13.895393050589597
R²: 0.989438691787125
```

The gradient descent algorithm successfully optimized the parameters θ_0 and θ_1, resulting in a well-fitted regression model. The low error metrics and high R^2 value indicate a strong linear relationship between X and Y.