Assignment: Implementing Linear Regression on the Auto MPG Dataset

---

Objective: In this assignment, you will implement and evaluate a simple linear regression model using the **Auto MPG dataset**. You will also explore key metrics like Mean Squared Error (MSE) and R-squared (R^2) to evaluate your model's performance. As a challenge, you'll be asked to implement gradient descent to optimize the model's parameters.

Part 1: Understanding the Dataset

The dataset you'll be working with is the **Auto MPG Dataset**, which contains information about various characteristics of different cars, including horsepower and miles per gallon (mpg). For this assignment, we will focus on **Horsepower** (`horsepower`) as the single feature (input) to predict the **Miles Per Gallon** (`mpg`) as the target.

Dataset Details:

- **Feature (Input)**: `horsepower` – Horsepower of the car.
- **Target (Output)**: `mpg` – Miles per gallon.

This dataset is commonly used for linear regression because there's a reasonable expectation that horsepower has a relationship with miles per gallon (mpg).

Part 2: Tasks

1. **Implement the Linear Regression Model**:
    - Follow the pseudocode provided in the starter code to implement the linear regression model.
    - Compute both Mean Squared Error (MSE) and R-squared (R^2) to evaluate your model's performance.
2. **Visualize Results**:
    - Create a scatter plot of the **actual miles per gallon** against **horsepower**.
    - Plot a line for the **predicted miles per gallon** based on your linear regression model.
3. **Compare Performance**:
    - Compare your model's performance on both the training and test data by calculating MSE and R^2.

Part 3: Glossary of Key Terms

- **Linear Regression**: A statistical method that models the relationship between a dependent variable and one or more independent variables. In this case, we're predicting mpg (dependent variable) based on horsepower (independent variable).
- **Feature**: A measurable property or characteristic of a phenomenon being observed. In this case, the feature is `horsepower`.
- **Target**: The value that the model is trying to predict. Here, the target is `mpg`.
- **MSE (Mean Squared Error)**: A measure of how well the regression model's predictions match the actual values. It calculates the average of the squared differences between actual and predicted values.
- **R^2 (R-squared)**: A statistical measure that represents the proportion of the variance for the target variable that's explained by the independent variable(s). A value of 1 means perfect predictions.
- **Train-Test Split**: A technique where the dataset is divided into two parts: one for training the model and one for testing it. This helps evaluate how well the model generalizes to unseen data.

Rubric for the Auto MPG Linear Regression Assignment

---

1. Code Implementation (30 points)

- **Linear Regression Implementation (10 points)**
  The linear regression algorithm is correctly implemented based on the provided pseudocode. The model is built correctly using the input feature (`horsepower`) and the target (`mpg`).
- **MSE and R^2 Calculation (10 points)**
  Mean Squared Error (MSE) and R-squared (R^2) are correctly calculated for both the training and test datasets.
- **No Hardcoding (5 points)**
  The solution is flexible and does not contain hardcoded values (e.g., the code works with other datasets and parameter values, not just `horsepower` and `mpg`).
- **Training and Test Data (5 points)**
  The model is correctly trained on the training set and evaluated on the test set, using an appropriate train-test split.

---

2. Visualization (10 points)

- **Scatter Plot of Actual Data (5 points)**
  The student includes a correct scatter plot of actual miles per gallon (`mpg`) against horsepower (`horsepower`).
- **Prediction Line (5 points)**
  The student correctly plots the predicted regression line on top of the scatter plot to visually compare predictions with actual values.

---

3. Flexibility (10 points)

- **Model Works for Other Data (10 points)**
  The implementation is flexible enough to work with different datasets and features without major modifications. For example, the model should not be hardcoded to only work with `horsepower` and `mpg`.

---

4. Code Structure and Clarity (20 points)

- **Commenting and Documentation (10 points)**
  The code includes clear and meaningful comments that explain each step of the implementation. Variable names are descriptive, and the overall code is easy to follow.
- **Code Formatting (5 points)**
  The code is well-organized with proper indentation, making it easy to read and understand. The functions and logic are structured neatly.
- **Function Usage (5 points)**
  The code uses functions properly to break down tasks (e.g., separate functions for calculating MSE, R^2, and making predictions).

---

5. Reproducibility (10 points)

- **Reproducibility of Results (10 points)**
  The code runs successfully with the provided Auto MPG dataset without errors. The results can be reproduced when using the same random state in the train-test split.

---

6. Evaluation and Results (20 points)

- **Model Performance on Training Data (10 points)**
  The model's performance on the training data is evaluated correctly, with MSE and R^2 computed and interpreted. The student should demonstrate an understanding of what these metrics mean for the model's fit.
- **Model Performance on Test Data (10 points)**
  The model's performance on the test data is evaluated correctly, providing a comparison between training and test results. The student should explain any potential overfitting or underfitting.

Total Points Available: 100

Key Requirements for the Assignment:

1. **Correctness**:
   The linear regression model must be implemented correctly, including computing MSE and R^2 metrics for both the training and test datasets.

2. **No Hardcoding**:
   The solution should work dynamically with any input feature and target, not hardcoded for specific values like `horsepower` and `mpg`.
3. **Visualization**:
   The student must provide a clear and accurate scatter plot of the actual data points (e.g., `mpg` vs `horsepower`), as well as a line plot representing the predicted values.
4. **Code Structure**:
   The code must be well-organized, with proper function usage and clear comments explaining the purpose of each section. Descriptive variable names and readable formatting are important.
5. **Train-Test Split**:
   The student should correctly split the dataset into training and test sets, ensuring the model is trained on one set and evaluated on the other.
6. **Evaluation of Results**:
   The student should compute and interpret MSE and R^2 values for both the training and test datasets, explaining what the values mean in terms of model performance.
7. **Reproducibility**:
   The code should run without errors, and the results should be reproducible with the same dataset and random state.

Part 4: Challenge – Implementing Gradient Descent

What is Gradient Descent?

Gradient descent is an optimization algorithm used to minimize the error of a model by iteratively adjusting the model's parameters (like intercept and slope in linear regression). It finds the optimal set of parameters that best fit the data by moving in the direction that reduces the error (MSE) the most.

Gradient Descent Math (Simplified):

For a linear regression model with parameters θ and input feature *x*, the prediction is:

$$y_{\text{pred}} = \theta_0 + \theta_1 \times x$$

Where:

- θ_0 is the intercept.
- θ_1 is the slope of the line.

To minimize the error (MSE), we adjust θ_0 and θ_1 using gradient descent. The update rule for each parameter is:

$$\theta_j := \theta_j - \alpha \times \frac{1}{m} \sum_{i=1}^{m} (y_{\text{pred},i} - y_i) \times x_i$$

Where:

- α is the learning rate (step size).
- m is the number of training examples.
- θ_j represents each of the parameters θ_0 and θ_1.

Example:

Suppose you have one data point:

- $x = 100$ (horsepower)
- $y = 25$ (miles per gallon)

With initial guesses for θ_0 (intercept) as 0 and θ_1 (slope) as 1:

1. Predicted value:

$$y_{\text{pred}} = 0 + 1 \times 100 = 100$$

   The actual value y is 25, so the error is $y - y_{\text{pred}} = 25 - 100 = -75$.

2. Gradient Descent Step (for θ_0 and θ_1):

$$\theta_0 := \theta_0 - \alpha \times (y_{\text{pred}} - y)$$

$$\theta_1 := \theta_1 - \alpha \times (y_{\text{pred}} - y) \times x$$

   If $\alpha = 0.001$, the update would be:

$$\theta_0 := 0 - 0.001 \times (-75) = 0.075$$

$$\theta_1 := 1 - 0.001 \times (-75) \times 100 = 8.5$$

Repeat this process for each data point over multiple iterations to minimize the error.

Pseudocode for Gradient Descent:

```
def gradient_descent(X, y, learning_rate, iterations):
    # Initialize theta (intercept and slope) to small random values
    # Number of training samples

    for each iteration:
        # Predict y based on current theta values

        # Compute gradients for theta_0 and theta_1
        # Gradient for theta_0 is the average difference between
predictions and actual values
        # Gradient for theta_1 is the average difference between
predictions and actual values, multiplied by X

        # Update theta parameters using the gradients and learning rate
        # theta_0 := theta_0 - (learning_rate * gradient for theta_0)
        # theta_1 := theta_1 - (learning_rate * gradient for theta_1)

        # Optional: Compute and print the cost (MSE) every 100 iterations

    # Return the optimized theta values
```

Rubric for the Challenge Question: Implementing Gradient Descent

---

1. Gradient Descent Implementation (30 points)

- **Correct Algorithm Implementation (10 points)**
  The gradient descent algorithm is correctly implemented based on the provided pseudocode.
- **Parameter Updates (10 points)**
  Parameters θ0\theta_0θ0 (intercept) and θ1\theta_1θ1 (slope) are updated correctly during each iteration.
- **Cost Reduction Over Time (5 points)**
  The cost (Mean Squared Error) decreases over time, indicating that gradient descent is working properly.
- **No Hardcoding (5 points)**
  The solution is flexible and not hardcoded for specific values. It should work with any dataset.

---

2. Learning Rate and Iterations (20 points)

- **Appropriate Learning Rate (10 points)**
  The learning rate chosen is reasonable (not too large to cause divergence, nor too small to make the algorithm inefficient).
- **Sufficient Number of Iterations (10 points)**
  The number of iterations is enough for the gradient descent to converge, meaning the cost function reaches a minimum.

---

3. Explanation and Comments (20 points)

- **Clear Explanation of Algorithm (10 points)**
  The student clearly explains how gradient descent works, including the update rule for the parameters θ0\theta_0θ0 and θ1\theta_1θ1, and the goal of minimizing the cost function.
- **Commenting of Code (10 points)**
  The code is well-commented, with clear explanations of each section, making the logic and purpose of each step easy to follow.

---

4. Code Quality and Structure (10 points)

- **Modular and Clean Code (5 points)**
  The code is well-structured, with proper function usage (e.g., separate functions for gradient descent, MSE, and R-squared). Code is also properly indented and readable.
- **Reproducibility (5 points)**
  The code runs successfully with the provided dataset, and the results can be reproduced.

---

5. Visualization and Performance (10 points)

- **Graphical Visualization of Results (5 points)**
  A plot is included showing the actual data points (e.g., a scatter plot of `mpg` vs `horsepower`) along with the predicted regression line from gradient descent.
- **Reasonable Performance (5 points)**
  The learned model produces reasonable predictions after gradient descent converges, reflected in the plot and final metrics.

---

6. Flexibility (10 points)

- **Solution Works with Other Data (10 points)**
  The code is flexible enough to handle other input features and target variables without hardcoding for specific values (e.g., not tied specifically to `horsepower` and `mpg`).

Total Points Available: 100

Key Requirements for the Challenge Question:

1. **Correctness**:
   The gradient descent algorithm must be implemented correctly. The model parameters θ0 and θ1 should be updated in each iteration to minimize the cost function.
2. **No Hardcoding**:
   The solution should work dynamically for various datasets and input features, without specific hardcoded values.
3. **Appropriate Learning Rate and Iterations**:
   The student must select a reasonable learning rate that enables the model to converge. The number of iterations should be sufficient to allow the gradient descent algorithm to reach a minimum cost.
4. **Explanation**:
   The student should explain the gradient descent algorithm clearly, including the purpose of parameter updates and how they minimize the cost function. The code should also be well-commented, with explanations of each step.
5. **Visualization**:
   The student must provide a visual representation of the actual data points and the predicted regression line after gradient descent optimization.
6. **Flexibility**:
   The code should be adaptable to work with other datasets or input features and target variables, showcasing a general understanding of gradient descent.
