Assignment On

Linear Regression: Exploring Key Parameters and Model Evaluation

Course Title: Data science Lab

Submitted To:

Dr. Md. Manowarul Islam
Associate Professor
Department of CSE
Jagannath University, Dhaka

Submitted By:

Team: Zero_splash
Shabikun Nahar, Umme Mehnaz Labiba & Md. Sakibul Islam
ID: B190305013, B190305015 & B190305035

4th Year 2nd Semester

Department of Computer Science & Engineering
Jagannath University, Dhaka

Linear Regression

Task 1: Dataset Preparation

Theory

The dataset used in this assignment is an insurance dataset, which provides information about individuals' health and insurance details. This dataset was selected because it contains a straightforward relationship between variables, making it ideal for demonstrating linear regression. Specifically, we aim to predict **insurance charges** based on an individual's **BMI (Body Mass Index)**, which is a well-known indicator of health risks and healthcare costs.

Key features of the dataset:

- **age**: The age of the individual, which influences insurance charges.
- **sex**: The gender of the individual (male/female).
- **bmi**: The Body Mass Index, calculated as weight (kg) divided by height (m^2). This is a key predictor of health outcomes.
- **children**: The number of dependents covered by the insurance policy.
- **smoker**: Whether the individual is a smoker (yes/no).
- **region**: The geographical region where the individual resides (e.g., southeast, northwest).
- **charges**: The insurance charges billed to the individual, which is our target variable (dependent variable).

The relevance of the dataset lies in its real-world applicability, where healthcare providers and insurance companies often use similar data to predict costs, assess risk, and determine premiums.

Implementation

```
# Load a dataset

data = pd.read_csv('/content/sample_data/insurance.csv')
data

X = data['bmi'].values.reshape(-1, 1)
Y = data['charges'].values
print("Dataset loaded with", len(X), "data points.")
```

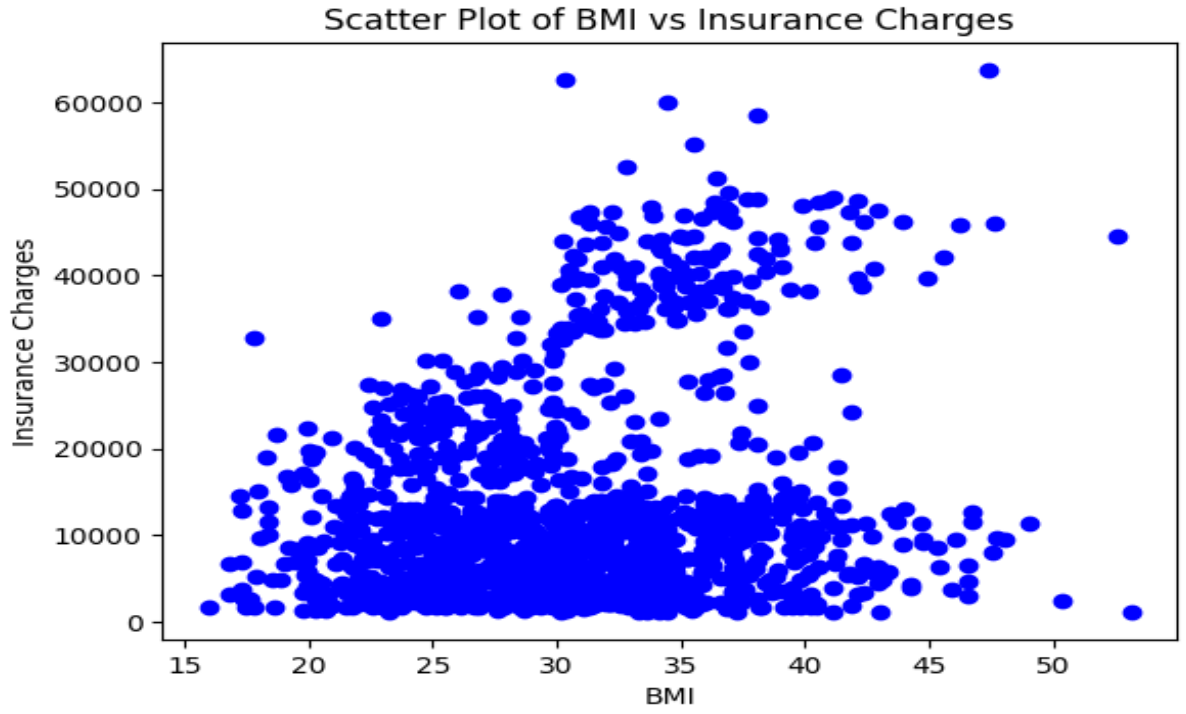
Task 2: Data Visualization

Theory

In this assignment, a scatter plot was created to represent the relationship between BMI and insurance charges. The x-axis represents **BMI**, and the y-axis represents **insurance charges**. Each data point corresponds to an individual's BMI and associated cost. The plot serves as a visual confirmation of the underlying relationship and provides a foundation for building the linear regression model.

Implementation

```
plt.scatter(X, Y, color='blue')
plt.title("Scatter Plot of BMI vs Insurance Charges")
plt.xlabel("BMI")
plt.ylabel("Insurance Charges")
plt.show()
```



Task 3: Linear Regression Implementation

Theory

Linear regression is a statistical method used to model the relationship between an independent variable (predictor, X) and a dependent variable (response, Y). In this assignment, we use linear regression to predict **insurance charges (Y)** based on an individual's **BMI (X)**.

The regression equation is represented as:

$$Y=mX+c$$

Where:

- Y: The predicted value (insurance charges).
- X: The independent variable (BMI).
- m: The slope of the line, which indicates the rate of change in Y for a unit change in X. In this dataset, it reflects how much insurance charges increase for a one-unit increase in BMI.
- c: The intercept, which is the value of Y when X=0. It represents the baseline cost irrespective of BMI.

Role of Slope and Intercept

1. **Slope (m):** Determines the strength and direction of the relationship between BMI and charges. A positive slope indicates that as BMI increases, insurance charges also increase.
2. **Intercept (c):** Represents the base cost of insurance when BMI is zero. Though BMI is rarely zero, the intercept provides a starting point for the regression line.

Implementation

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(X, Y)

slope = model.coef_[0]
intercept = model.intercept_
print(f"Slope (m): {slope}")
print(f"Intercept (c): {intercept}")
```

Task 4: Coefficient of Determination (R^2)

Theory

The coefficient of determination (R2R^2R2) is a key metric used to assess the performance of a regression model. It measures the proportion of variance in the dependent variable (YYY, insurance charges) that is explained by the independent variable (XXX, BMI) in the regression model.

The formula for R^2 is:

$$R^2 = 1 - SSres / SStot$$

Where:

- SSres: Sum of squared residuals (difference between actual and predicted YYY values).
- SStot: Total sum of squares (variance of YYY from its mean).

Implementation

```
r_squared = model.score(X, Y)
print(f"R²: {r_squared}")
```

Task 5: Predictions and Visualization

Theory

1. The regression model predicts the insurance charges for each individual's BMI.

2.The scatter plot of actual data points (BMI vs. charges) is overlaid with the regression line representing the predicted values.

3.If the line closely follows the data points, it indicates that the model is performing well.

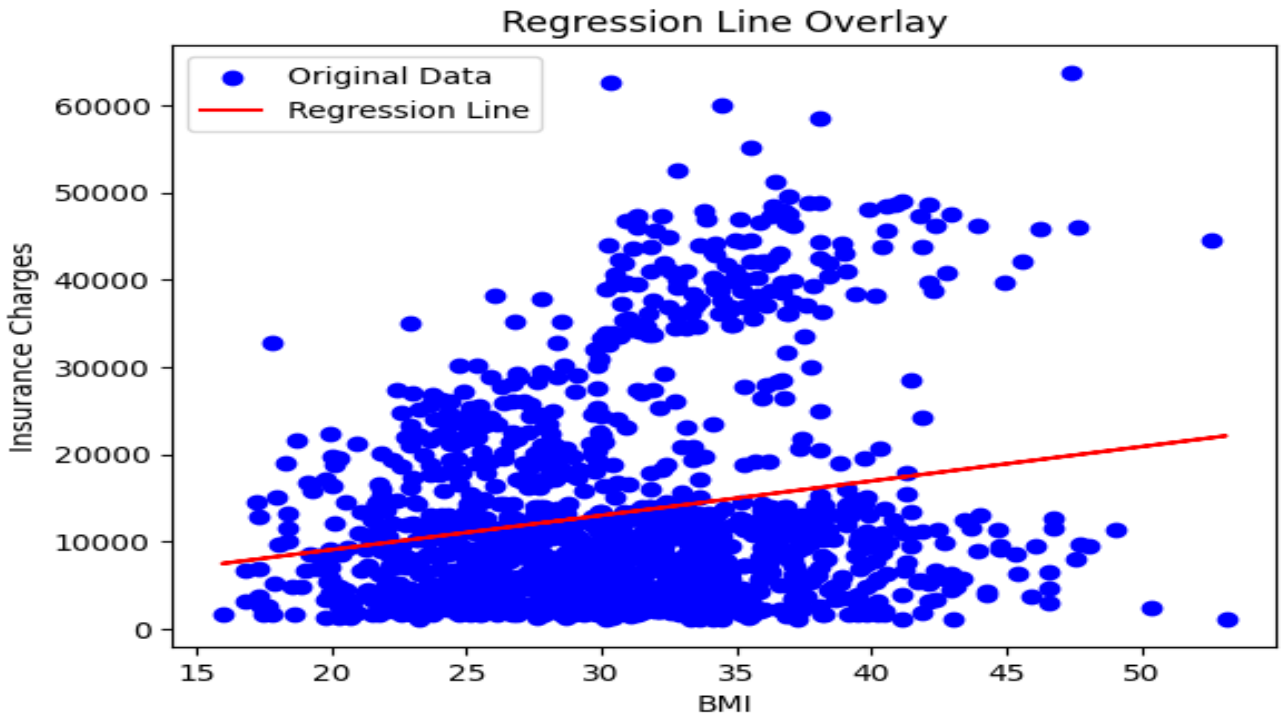
4.Significant deviations might suggest the presence of other influential factors (e.g., smoking, age) that the current model does not account for.

Implementation

```
Y_pred = model.predict(X)

plt.scatter(X, Y, color='blue', label='Original Data')
plt.plot(X, Y_pred, color='red', label='Regression Line')
plt.title("Regression Line Overlay")
plt.xlabel("BMI")
```

```
plt.ylabel("Insurance Charges")
plt.legend()
plt.show()
```



Task 6: Residual Analysis

Theory

Residual analysis is a critical step in verifying the assumptions of a linear regression model. Residuals are the differences between the actual values (Y) and the predicted values (Y^):

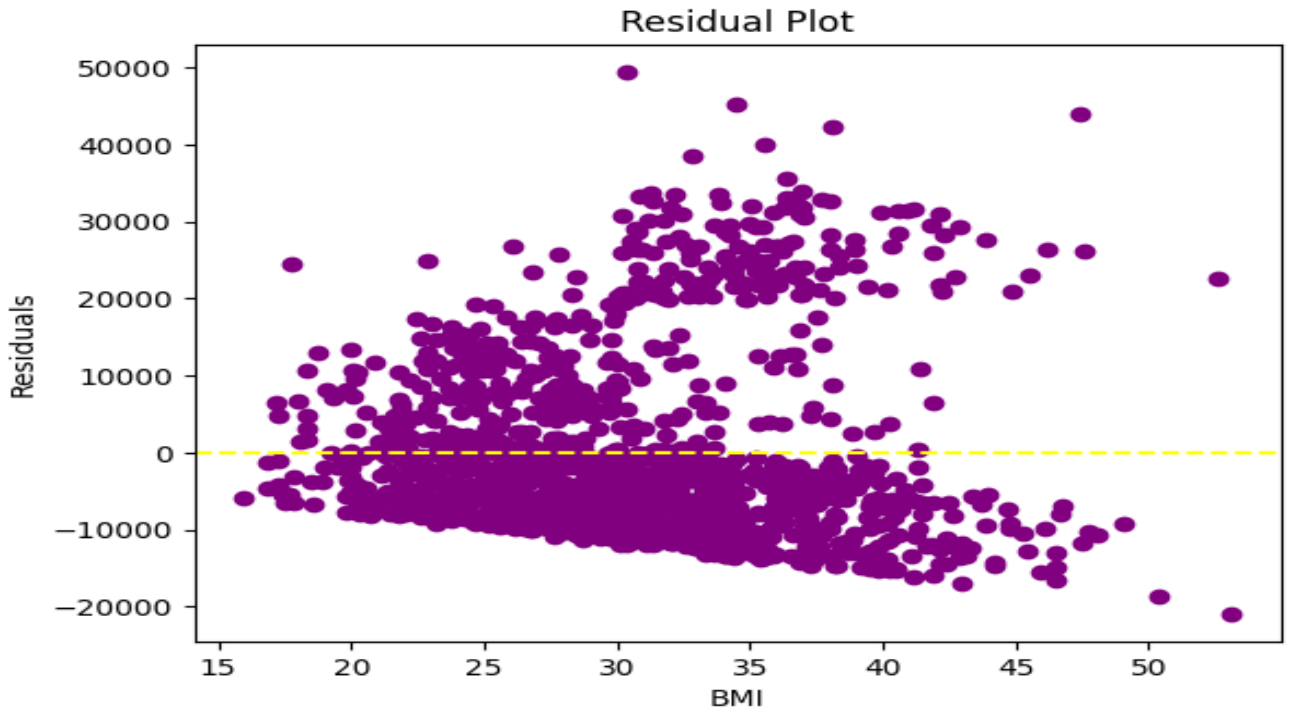
$$\text{Residual} = Y - \hat{Y}$$

Implementation

```
residuals = Y - Y_pred

plt.scatter(X, residuals, color='purple')
plt.axhline(y=0, color='yellow', linestyle='--')
plt.title("Residual Plot")
plt.xlabel("BMI")
```

```
plt.ylabel("Residuals")
plt.show()
```



Task 7: Model Evaluation

Theory

Model evaluation metrics are essential for assessing the accuracy and reliability of a regression model. These metrics quantify the differences between the actual values (Y) and the predicted values (Y^), providing insights into the model's performance.

Mean Absolute Error (MAE):

$$MAE=\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}|$$

Mean Squared Error (MSE):

$$MSE=\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}|^2$$

Root Mean Squared Error (RMSE):

$$\text{RMSE}=\sqrt{MSE}$$

Implementation

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

print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
```

Output:

```
Mean Absolute Error (MAE): 9172.351145507564
Mean Squared Error (MSE): 140777900.09850758
Root Mean Squared Error (RMSE): 11864.986308399499
```

Gradient Descent

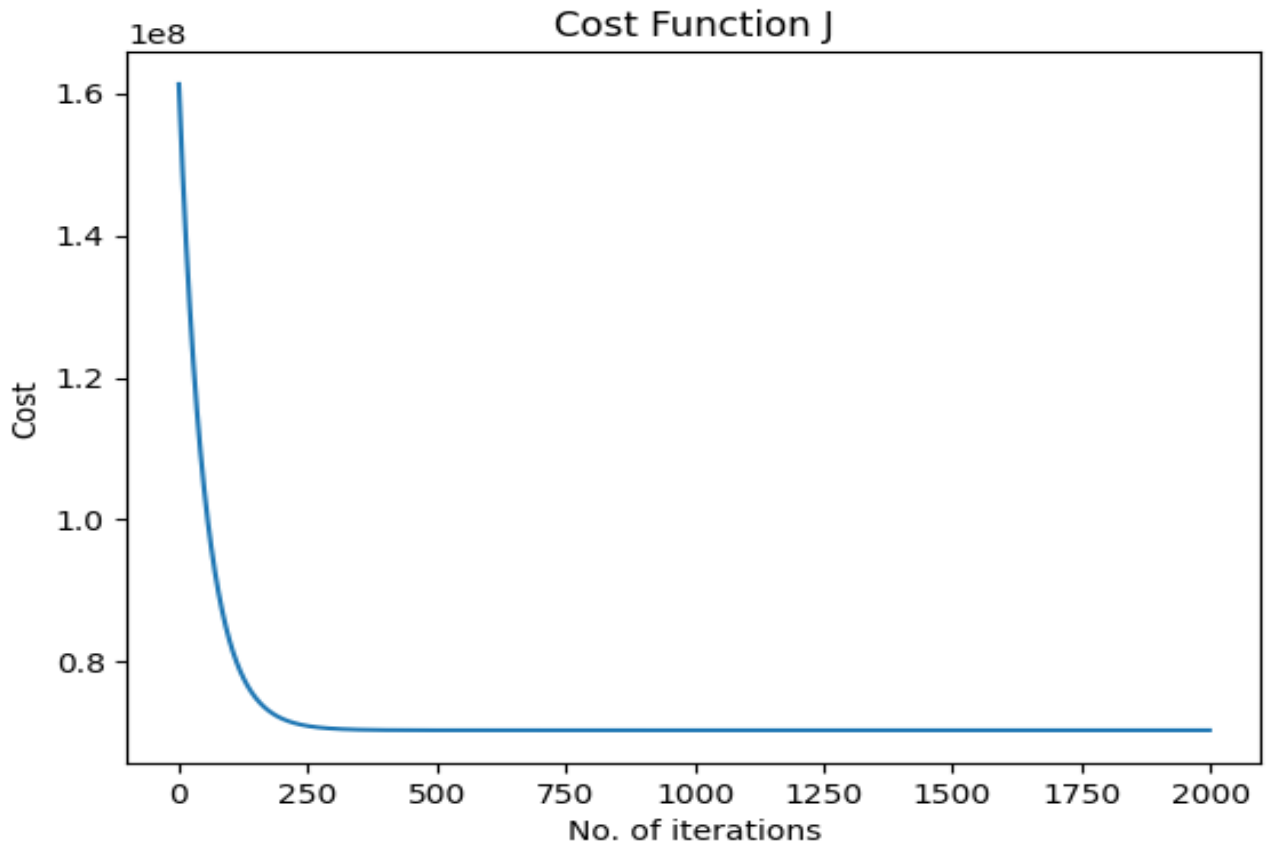
Theory

Cost Function: In this dataset, the cost function is the MSE, which measures the error between the actual insurance charges (Y) and the predicted values (Y^):

$$MSE=\frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}|^2$$

Implementation

```
plt.title('Cost Function J')
plt.xlabel('No. of iterations')
plt.ylabel('Cost')
plt.plot(past_costs)
plt.show()
```



Gradient: The gradient represents the rate of change of the cost function with respect to the model parameters (m and c). For each parameter:

Partial derivative with respect to m:

$$\frac{\partial MSE}{\partial m} = -\frac{2}{n}\sum_{i=1}^{n} |y_i - \hat{y}| \, x_i$$

Partial derivative with respect to c:

$$\frac{\partial MSE}{\partial c} = -\frac{2}{n}\sum_{i=1}^{n} |y_i - \hat{y}|$$

Parameter Update Rule:

The parameters are updated in the direction opposite to the gradient:

$$m_{new} = m_{old} - \alpha . \frac{\partial MSE}{\partial m}$$

$$c_{new} = c_{old} - \alpha . \frac{\partial MSE}{\partial c}$$

Implementation

```
alpha = 0.01 #Step size
iterations = 2000 #No. of iterations
m = y.size #No. of data points
np.random.seed(123) #Set the seed
theta = np.random.rand(2) #Pick some random values to start with


#GRADIENT DESCENT
def gradient_descent(x, y, theta, iterations, alpha):
    past_costs = []
    past_thetas = [theta]
    for i in range(iterations):
        prediction = np.dot(x, theta)
        error = prediction - y
        cost = 1/(2*m) * np.dot(error.T, error)
        past_costs.append(cost)
        theta = theta - (alpha * (1/m) * np.dot(x.T, error))
        past_thetas.append(theta)
```

```
    return past_thetas, past_costs

#Pass the relevant variables to the function and get the new values
back...
past_thetas, past_costs = gradient_descent(x, y, theta, iterations, alpha)
theta = past_thetas[-1]

#Print the results...
print("Gradient Descent: {:.2f}, {:.2f}".format(theta[0], theta[1]))
```

Output:

```
Gradient Descent: 13270.42, 2401.91
```

```
#Animation output for gradient decent scatterplot
import io
import base64
from IPython.display import HTML

filename = 'animation.gif'

video = io.open(filename, 'r+b').read()
encoded = base64.b64encode(video)
HTML(data='''<img src="{0}" type="gif"
/>'''.format(encoded.decode('asc ii')))
```