Batch: B - 1 Roll No.: 16014022050

Experiment No.: 5

Title: To implement linear regression algorithm

Course Outcome:

CO2: Differentiate between types of machine learning and implement supervised and unsupervised learning

---

Books/ Journals/ Websites referred:

1. **"Artificial Intelligence: a Modern Approach" by Russell and Norving, Pearson education Publications**
2. **"Artificial Intelligence" By Rich and knight, Tata McGraw Hill Publications**

Introduction:

- Introduction

Linear Regression is one of the most fundamental and widely used **supervised machine learning algorithms**. It is used to model the relationship between a **dependent variable (target)** and one or more **independent variables (features)** by fitting a straight line (or hyperplane in higher dimensions). The goal is to predict continuous numeric values.

Types of Linear Regression

1. **Simple Linear Regression**
   - Involves **one independent variable** and one dependent variable.
   - The relationship is modeled using the equation:

$$y = mx + b$$

   where
   - y = dependent variable (target)
   - x = independent variable (feature)
   - m = slope (change in y for unit change in x)
   - b = intercept (value of y when x = 0)
2. **Multiple Linear Regression**
   - Involves **two or more independent variables**.
   - The model equation becomes:

$$y = b_0 + b_1x_1 + b_2x_2 + \cdots + b_nx_n$$

   where b_0 is intercept, $b_1, b_2, \ldots, b_n$ are coefficients for each feature.

Assumptions of Linear Regression

1. **Linearity** – Relationship between input and output is linear.
2. **Independence** – Observations are independent of each other.
3. **Homoscedasticity** – Constant variance of errors across all levels of independent variables.
4. **Normality of Errors** – Residuals (errors) should be normally distributed.
5. **No Multicollinearity** – Independent variables should not be highly correlated with each other.

🔹 Cost Function

The most commonly used cost function is **Mean Squared Error (MSE)**:

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

- y_i = actual values
- $\hat{y}_i$ = predicted values

The regression algorithm finds the line/plane that **minimizes this error**.

🔹 Evaluation Metrics

To check model performance:

- **Mean Absolute Error (MAE)** – average absolute difference between predicted and actual values.
- **Mean Squared Error (MSE)** – average squared difference.
- **Root Mean Squared Error (RMSE)** – square root of MSE, interpretable in original units.
- **R^2 Score (Coefficient of Determination)** – how well features explain variation in target (closer to 1 = better).

Implementation:

Dataset

Link - https://www.kaggle.com/datasets/kanzariachref/medical-insurance-cost-dataset?resource=download

Importing Libraries

✓ 11s

```
[1]  # Importing libraries
     import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns

     from sklearn.model_selection import train_test_split
     from sklearn.linear_model import LinearRegression
     from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

Checking the dataset

```
# --------------------------------
# Step 1: Load and clean dataset
# --------------------------------
df = pd.read_csv("Medical-Insurance.csv")

print("First 5 rows of dataset:")
print(df.head())
print("\nDataset info:")
print(df.info())
print("\nMissing values:")
print(df.isnull().sum())
```

```
First 5 rows of dataset:
   19  1    27.9  0 1.1  3    16884.924
0  18  2  33.770  1   0  4   1725.55230
1  28  2  33.000  3   0  4   4449.46200
2  33  2  22.705  0   0  1  21984.47061
3  32  2  28.880  0   0  1   3866.85520
4  31  1  25.740  0   ?  4   3756.62160

Dataset info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2771 entries, 0 to 2770
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   19         2771 non-null   object
 1   1          2771 non-null   int64
 2   27.9       2771 non-null   float64
 3   0          2771 non-null   int64
 4   1.1        2771 non-null   object
 5   3          2771 non-null   int64
 6   16884.924  2771 non-null   float64
dtypes: float64(2), int64(3), object(2)
memory usage: 151.7+ KB
None

Missing values:
19           0
1            0
27.9         0
0            0
1.1          0
3            0
16884.924    0
dtype: int64
```

Cleaned Data

```
# Manually rename columns
df.columns = ["age", "gender", "bmi", "children", "smoker", "region", "charges"]

# Replace '?' with NaN in smoker column
df["smoker"] = df["smoker"].replace("?", np.nan)

# Convert smoker column to numeric (0/1)
df["smoker"] = pd.to_numeric(df["smoker"])

# Drop rows with missing values (if any remain after fixing)
df.dropna(inplace=True)

print("First 5 rows after cleaning:")
print(df.head())
print("\nData types after cleaning:")
print(df.dtypes)
print("\nMissing values after cleaning:")
print(df.isnull().sum())
```

```
First 5 rows after cleaning:
  age  gender     bmi  children  smoker  region      charges
0  18       2  33.770         1     0.0       4   1725.55230
1  28       2  33.000         3     0.0       4   4449.46200
2  33       2  22.705         0     0.0       1  21984.47061
3  32       2  28.880         0     0.0       1   3866.85520
5  46       1  33.440         1     0.0       4   8240.58960

Data types after cleaning:
age          object
gender        int64
bmi         float64
children      int64
smoker      float64
region        int64
charges     float64
dtype: object

Missing values after cleaning:
age         0
gender      0
bmi         0
children    0
smoker      0
region      0
charges     0
dtype: int64
```

Defining features and target

```
# --------------------------------
# Step 2: Define features and target
# Target = charges (last column, assumed in dataset)
# --------------------------------
X = df.drop("charges", axis=1)   # independent variables
y = df["charges"]                # dependent variable (target)
```

Train-split the dataset

```
# --------------------------------
# Step 3: Train-Test Split
# --------------------------------
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Training linear regression model

```
[13] # --------------------------------
     # Step 4: Linear Regression
     # --------------------------------
     model = LinearRegression()
     model.fit(X_train, y_train)
```

```
LinearRegression
LinearRegression()
```

```
# Predictions
y_pred = model.predict(X_test)
```

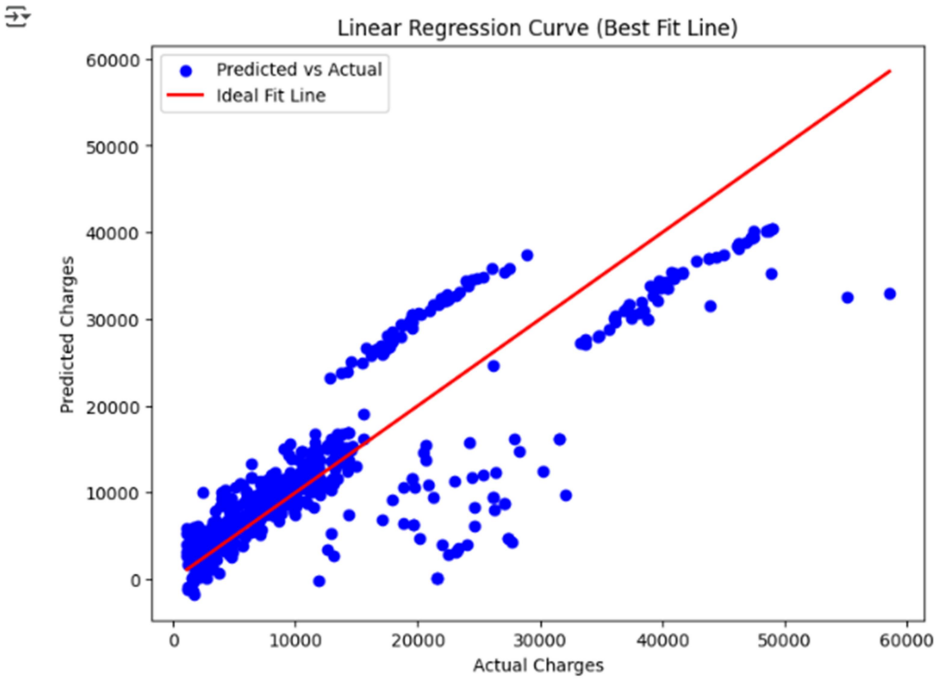
Evaluation Metrics

```
# --------------------------------
# Step 5: Evaluation
# --------------------------------
print("\nEvaluation Metrics:")
print("Mean Absolute Error (MAE):", mean_absolute_error(y_test, y_pred))
print("Mean Squared Error (MSE):", mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error (RMSE):", np.sqrt(mean_squared_error(y_test, y_pred)))
print("R2 Score:", r2_score(y_test, y_pred))
```

```
Evaluation Metrics:
Mean Absolute Error (MAE): 4460.537160958727
Mean Squared Error (MSE): 42791890.93506383
Root Mean Squared Error (RMSE): 6541.551110788926
R2 Score: 0.7228106584272765
```

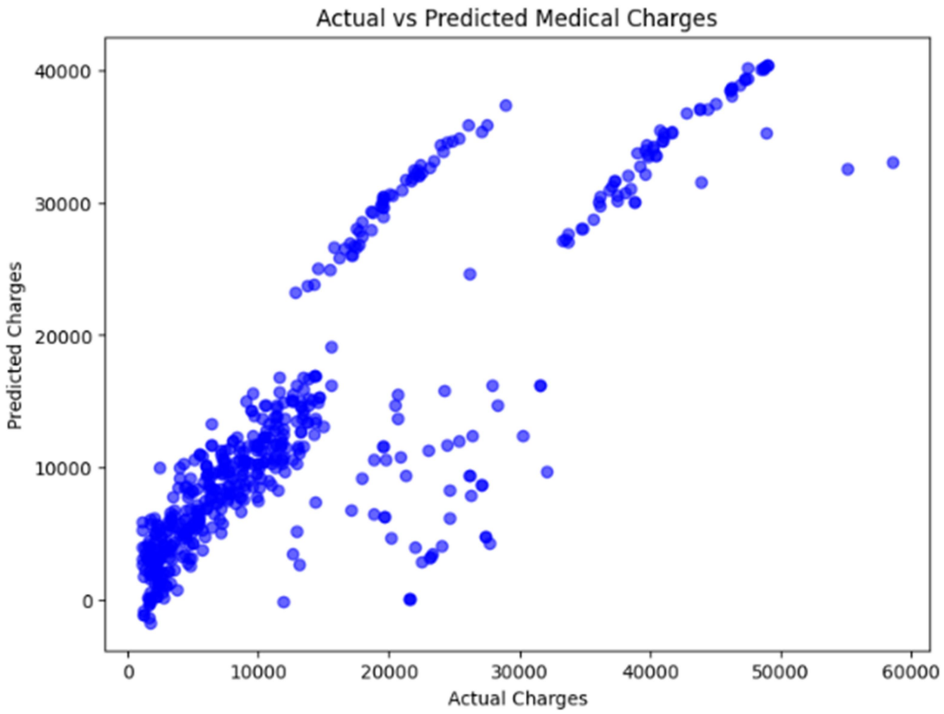
Regression Line

```
# Scatter plot of actual vs. predicted with regression line
plt.figure(figsize=(8,6))

# Plot the actual data points
plt.scatter(y_test, y_pred, color="blue", label="Predicted vs Actual")

# Plot regression line (perfect fit reference line)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
         color="red", linewidth=2, label="Ideal Fit Line")

plt.xlabel("Actual Charges")
plt.ylabel("Predicted Charges")
plt.title("Linear Regression Curve (Best Fit Line)")
plt.legend()
plt.show()
```



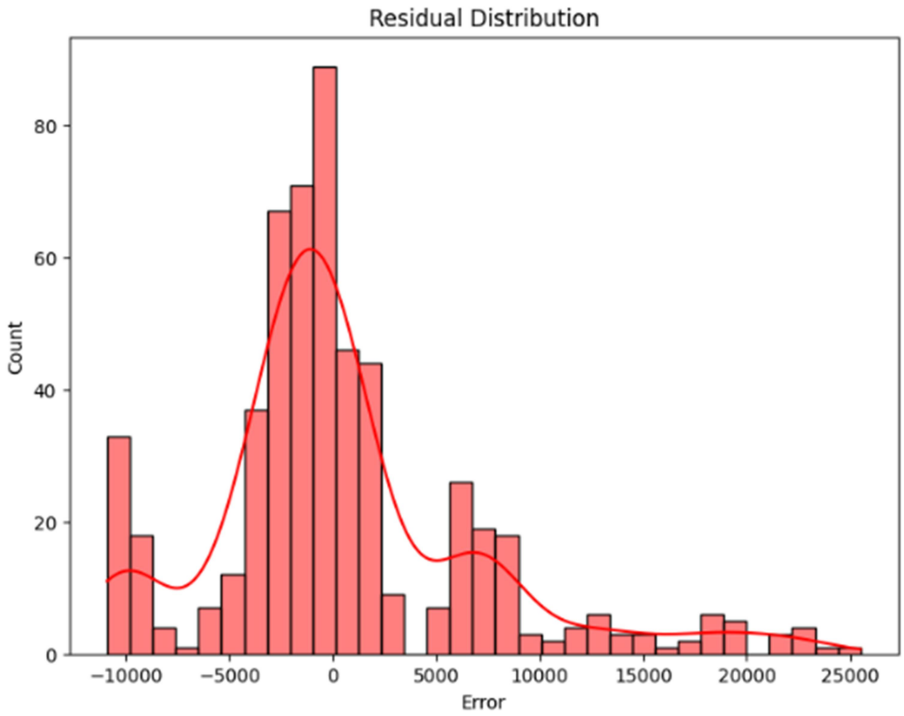
Graphs

```
# ------------------------------
# Step 6: Graphical Analysis
# ------------------------------

# Actual vs Predicted
plt.figure(figsize=(8,6))
plt.scatter(y_test, y_pred, alpha=0.6, color="blue")
plt.xlabel("Actual Charges")
plt.ylabel("Predicted Charges")
plt.title("Actual vs Predicted Medical Charges")
plt.show()
```



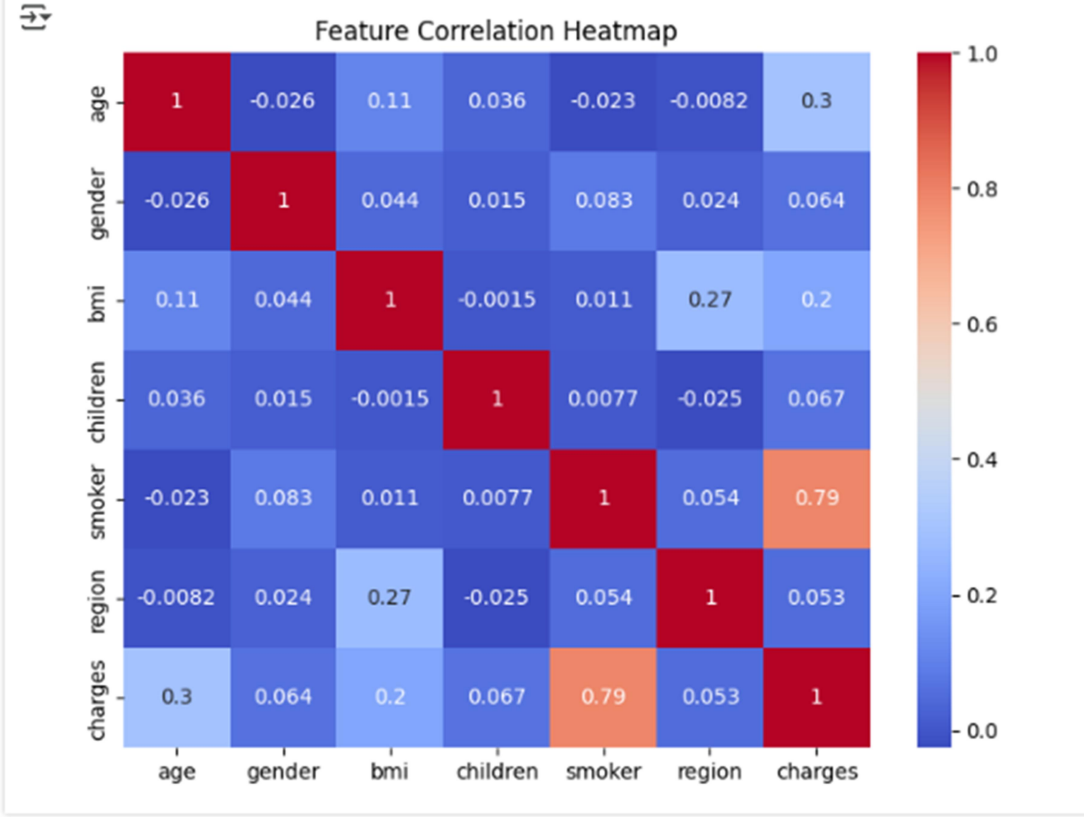
- The scatterplot shows most points are close to a diagonal line → predictions follow actual values fairly well.
- However, we can see clusters:
  - Low charges (<10,000)
  - Mid charges (10,000–25,000)
  - High charges (>30,000, mostly smokers)
- Model handles general trends but some spread exists, meaning it doesn't perfectly capture all variations (possible nonlinear effects).

```
# Residual Plot
residuals = y_test - y_pred
plt.figure(figsize=(8,6))
sns.histplot(residuals, kde=True, color="red")
plt.title("Residual Distribution")
plt.xlabel("Error")
plt.show()
```



- Residuals are mostly centered around 0, which is good (model has no large consistent bias).
- Some positive tail (errors up to +20,000) indicates the model sometimes underestimates very high charges.
- Distribution is not perfectly normal → suggesting linear regression might not fully capture complex relationships (nonlinear models like Decision Trees / Random Forest could improve results).

```
# Correlation Heatmap
plt.figure(figsize=(8,6))
sns.heatmap(df.corr(), annot=True, cmap="coolwarm")
plt.title("Feature Correlation Heatmap")
plt.show()
```



- Strongest positive correlation with charges is smoker (0.79) → smoking has the biggest impact on medical charges.
- Moderate correlation with age (0.30) and BMI (0.20) → older age and higher BMI also tend to increase charges.
- Gender, children, and region show very weak correlation → they don't significantly affect charges.
- This means features like smoker, age, and BMI are the most important for predicting costs.

Running model with own data

```
# Example test case: [age, gender, bmi, children, smoker, region]
# gender: 1 = male, 2 = female
# smoker: 1 = smoker, 0 = non-smoker
# region: 1 = NW, 2 = NE, 3 = SW, 4 = SE

test_data = pd.DataFrame({
    "age": [45],
    "gender": [1],
    "bmi": [28.5],
    "children": [2],
    "smoker": [1],
    "region": [3]
})

predicted_charge = model.predict(test_data)
print("Predicted Medical Charge:", predicted_charge[0])
```

```
Predicted Medical Charge: 33623.44101594797
```

Complete Implementation Code

```
# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# ------------------------------
# Step 1: Load and clean dataset
# ------------------------------
df = pd.read_csv("Medical-Insurance.csv")

print("First 5 rows of dataset:")
print(df.head())
print("\nDataset info:")
print(df.info())
print("\nMissing values:")
print(df.isnull().sum())

# ------------------------------
# Step 2: Define features and target
# Target = charges (last column, assumed in dataset)
# ------------------------------
```

```
X = df.drop("charges", axis=1)   # independent variables
y = df["charges"]                # dependent variable (target)

# ------------------------------
# Step 3: Train-Test Split
# ------------------------------
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# ------------------------------
# Step 4: Linear Regression
# ------------------------------
model = LinearRegression()
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# ------------------------------
# Step 5: Evaluation
# ------------------------------
print("\nEvaluation Metrics:")
print("Mean Absolute Error (MAE):", mean_absolute_error(y_test, y_pred))
print("Mean Squared Error (MSE):", mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error (RMSE):", np.sqrt(mean_squared_error(y_test,
y_pred)))
print("R2 Score:", r2_score(y_test, y_pred))

# ------------------------------
# Step 6: Graphical Analysis
# ------------------------------

# Actual vs Predicted
plt.figure(figsize=(8,6))
plt.scatter(y_test, y_pred, alpha=0.6, color="blue")
plt.xlabel("Actual Charges")
plt.ylabel("Predicted Charges")
plt.title("Actual vs Predicted Medical Charges")
plt.show()

# Residual Plot
residuals = y_test - y_pred
plt.figure(figsize=(8,6))
sns.histplot(residuals, kde=True, color="red")
plt.title("Residual Distribution")
plt.xlabel("Error")
```

```
plt.show()

# Correlation Heatmap
plt.figure(figsize=(8,6))
sns.heatmap(df.corr(), annot=True, cmap="coolwarm")
plt.title("Feature Correlation Heatmap")
plt.show()
```

Post Lab Descriptive Questions:

1. **What is the purpose of a linear regression model? How does it predict the output variable from the input variable(s)?**
   The purpose of a linear regression model is to find a mathematical relationship between independent variables (features) and a dependent variable (target). It predicts the output by fitting a straight line (or hyperplane) through the data, such that for a given set of input values, the model calculates the target value using the learned coefficients.
2. **In simple linear regression, what do the slope and intercept represent in the equation y=mx+b?**
   - The slope (m) represents how much the dependent variable (y) changes for every one-unit increase in the independent variable (x).
   - The intercept (b) is the value of y when x = 0, i.e., it shows where the line crosses the y-axis.

3. **What is the role of the cost function (e.g., mean squared error) in training a linear regression model?**
   The cost function measures the difference between the predicted values and the actual values. In linear regression, Mean Squared Error (MSE) is commonly used. The training process tries to minimize this cost function, so that the predicted values are as close as possible to the actual outputs.

4. **How can you identify if your linear regression model is overfitting or underfitting the data?**
   - Overfitting: The model performs very well on training data but poorly on test data (too complex, memorizes noise).
   - Underfitting: The model performs poorly on both training and test data (too simple, cannot capture the trend).
   - We can detect this by comparing training vs. testing performance scores (e.g., R^2).

5. **What metrics can be used to evaluate the accuracy and goodness-of-fit of a linear regression model?**
    - Mean Absolute Error (MAE) – average of absolute errors.
    - Mean Squared Error (MSE) – average of squared errors.
    - Root Mean Squared Error (RMSE) – error in original units.
    - R^2 Score (Coefficient of Determination) – proportion of variance in target explained by the model.