UNIVERSITY OF ENGINEERING AND TECHNOLOGY, TAXILA

FACULTY OF TELECOMMUNICATION AND INFORMATION ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

Lab 4: Linear Regression with Multiple Variables

Objective:

To understand and implement **Multiple Linear Regression**, where multiple independent variables influence the dependent variable. This lab will focus on core concepts, assumptions, and practical implementation using Python. Students will learn to visualize relationships, interpret results, and evaluate model performance.

Prerequisites:

- Knowledge of basic Python programming.
- Familiarity with libraries: Pandas, NumPy, Matplotlib, and Scikit-learn.
- Understanding of **Simple Linear Regression** and its assumptions.

1. Introduction to Multiple Linear Regression

What is Regression?

Regression is a statistical method for modeling relationships between a dependent variable (**target**) and multiple independent variables (**predictors**).

In **Multiple Linear Regression**, the relationship is modeled as:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n + \epsilon$$

2. Assumptions of Multiple Linear Regression

1. **Linearity**: The relationship between **x** and **y** is linear.
2. **Independence**: Observations are independent of each other.
3. **Homoscedasticity**: Residuals (errors) have **constant variance**.
4. **Normality of Residuals**: Residuals follow a normal distribution.

Theory: Evaluating Multiple Linear Regression Models

Key Metrics:

1. **Mean Squared Error (MSE):** Measures the average squared error between predicted and actual values.
2. **R-squared (R^2):** Explains the proportion of variance in **y** explained by **x** variables.

$$R^2 = 1 - \frac{Sum\ of\ Squared\ Errors}{Total\ Sum\ of\ Squares}$$

Visualization:

- **Regression Line (in multi-dimensions):** Helps assess model performance.
- **Residual Plot:** Checks for assumption violations (e.g., non-linearity, heteroscedasticity).

4. Implementation of Multiple Linear Regression

Step 1: Import Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
```

 Pandas: For data handling
 NumPy: For numerical operations
 Matplotlib & Seaborn: For visualization
 Scikit-learn: For model training and evaluation

Step 2: Load Dataset

```
data = pd.read_csv("housing.csv")
print("First 5 rows of the dataset:")
print(data.head())
```

 pd.read_csv: Reads a CSV file into a DataFrame
 data.head(): Displays the first five rows to verify dataset structure

Step 3: Data Preprocessing

1. **Select Independent (X) and Dependent (y) Variables**
2. **Handle Missing Values (if any)**

```
# Selecting independent variables (predictors)
X = data[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
         'Avg. Area Number of Bedrooms', 'Area Population']]

# Selecting dependent variable (target)
y = data['Price']

# Check for missing values
print("\nMissing values in the dataset:")
print(data.isnull().sum())
```

- X: Extracts multiple independent variables
- y: Extracts the dependent variable (**Price**)
- isnull().sum(): Identifies missing values

Step 4: Train-Test Split

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- train_test_split: Splits data into **training (80%)** and **testing (20%)** sets
- random_state=42: Ensures **consistent** results across runs

Step 5: Train the Model

```
# Initialize and train the multiple linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Print model parameters
print("Intercept (β0):", model.intercept_)
print("Coefficients (β1, β2, ..., βn):", model.coef_)
```

- LinearRegression(): Initializes the model
- model.fit: Trains the model on training data
- model.intercept_: Displays the **intercept**
- model.coef_: Displays the **coefficients** for each variable

Step 6: Make Predictions

```
y_pred = model.predict(X_test)

# Compare actual and predicted values
comparison = pd.DataFrame({'Actual': y_test.values, 'Predicted': y_pred})
print(comparison.head())
```

- model.predict: Generates predictions for the **test set**
- DataFrame: Combines **actual** and **predicted** values

Step 7: Evaluate the Model

```
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("\nModel Evaluation Metrics:")
print("Mean Squared Error (MSE):", mse)
print("R-squared (R²):", r2)
```

- mean_squared_error: Computes **MSE**
- r2_score: Computes **R**

Step 8: Visualize the Results

```
# Plot Actual vs Predicted values
plt.figure(figsize=(8,6))
sns.scatterplot(x=y_test, y=y_pred, color="blue")
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color="red", linestyle="--")
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.title("Actual vs Predicted Prices")
plt.show()
```

- **Scatter plot**: Visualizes actual vs predicted prices
- **Red line**: Represents perfect predictions (ideal case)

5. Summary and Insights

- **Intercept (β0):** Predicted price when all independent variables are **zero**.
- **Coefficients (β1, β2, ..., βn):** Measure the impact of each variable on price.
- **R-squared (R^2):** Evaluates how well the model explains variations in **price**.
- **Visualization:** Helps verify model performance.

6. Deliverables

1. A **Jupyter Notebook** with full implementation.
2. Visualizations of regression results.
3. Answers to lab questions.

Lab Questions

1. What does each **coefficient (β1, β2, ...)** indicate for this dataset?
2. How well does the model predict **Price** based on R^2?
3. Are there any **patterns** in the residuals that violate regression assumptions?