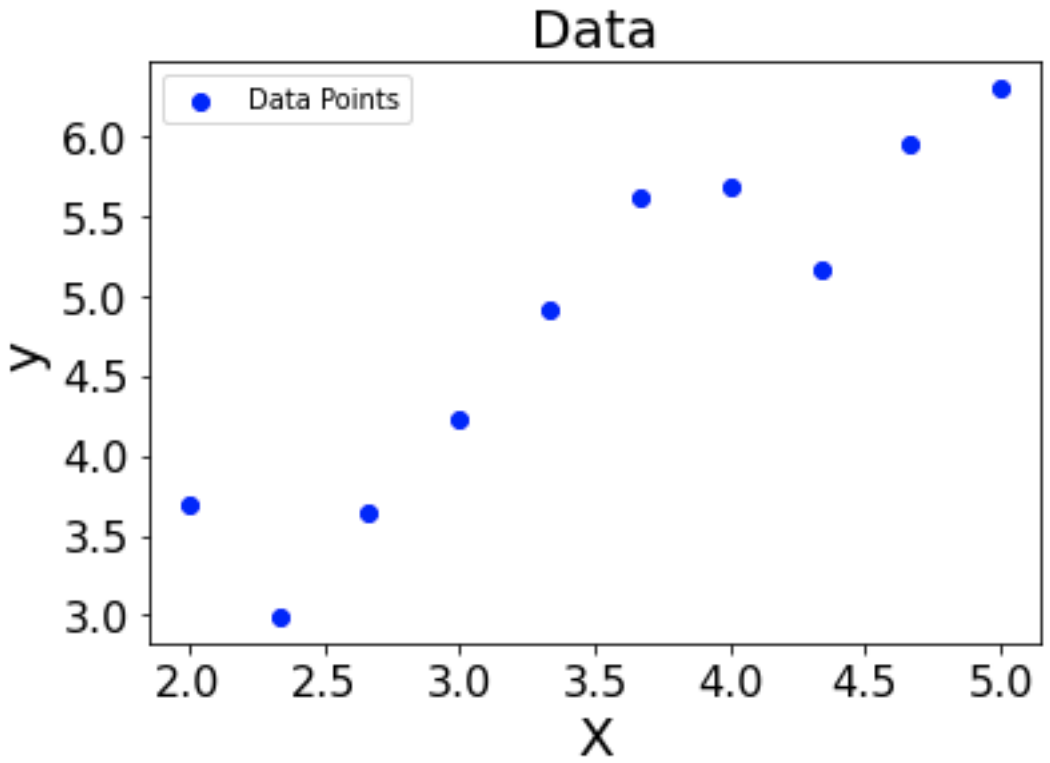
PH 3022
Machine Learning and Neural Computation

Linear Regression

Department of Physics – University of Colombo

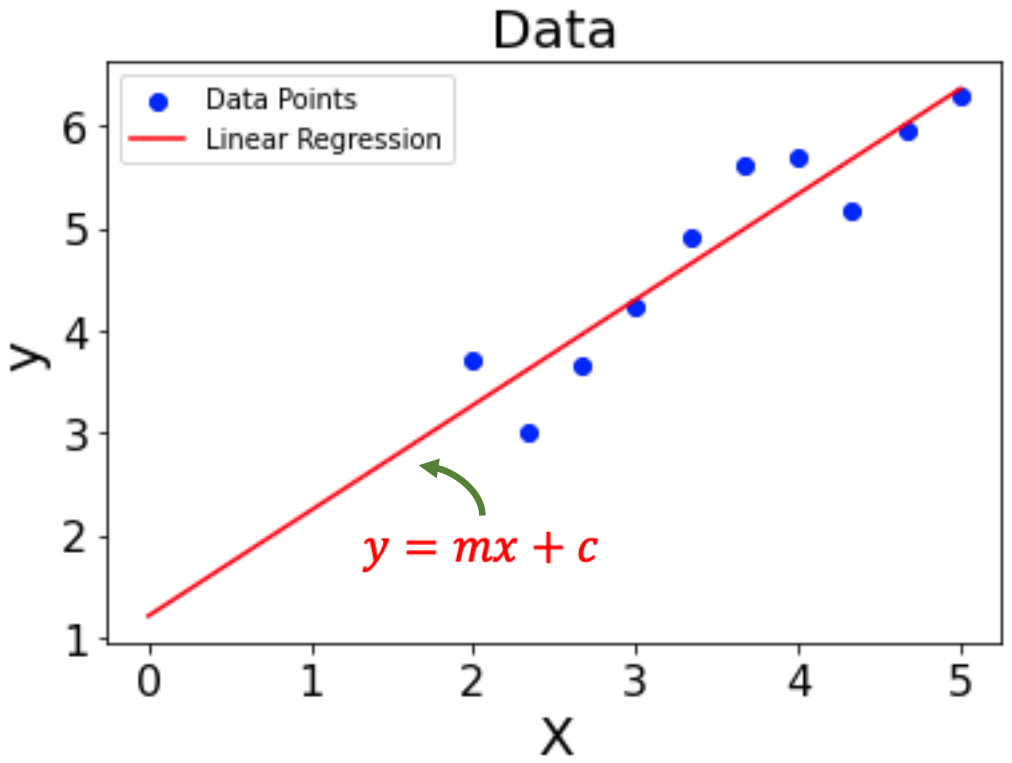
Linear Regression

- Linear Regression is a simple and fundamental **supervised machine learning algorithm** used for predicting **continuous values**.
- It assumes a linear relationship between the input features and the target variable.



Model Representation

- The relationship between input features (denoted as "X") and the target variable ("y") is represented as a straight line: $y = mX + b$.
- "m" is the slope of the line, indicating how much the target variable changes when the input feature changes.
- "b" is the intercept of the line, representing the value of the target variable when the input feature is 0.



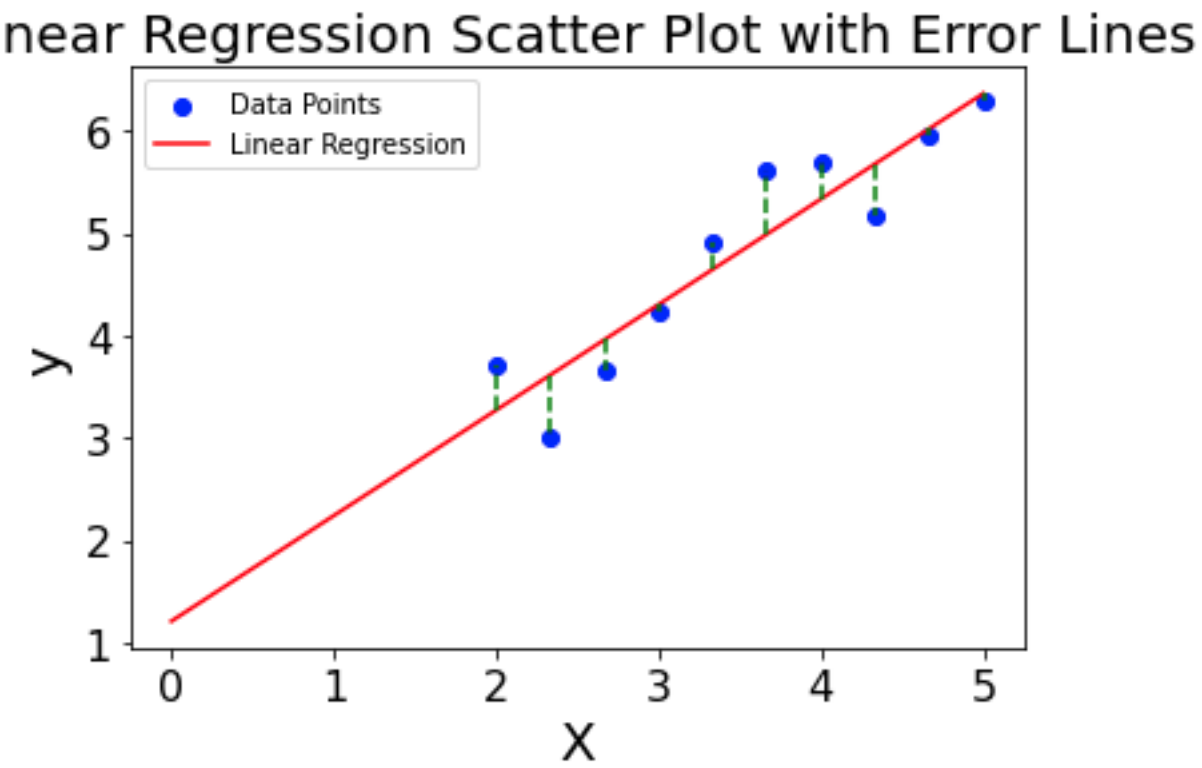
$$m = \frac{y_2 - y_1}{x_2 - x_1} = \tan(\theta)$$

θ is the angle made by the best fit with respect to x-axis.

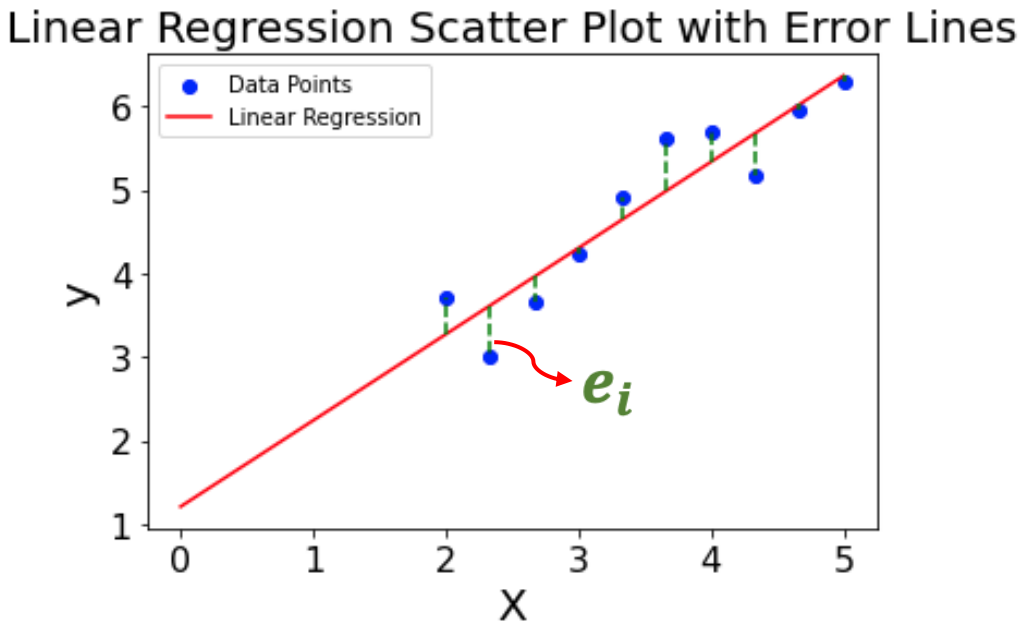
Training Process:

- The objective is to minimize the sum of squared differences between the actual target values and the predicted values.

- This optimization process is typically done using the Least Squares method.



Slope and Intercept



The error between actual and predicted data is given by

$$e_i = y_i^{pred} - y_i$$

$$e_i = mx_i + c - y_i$$

η can be defined as

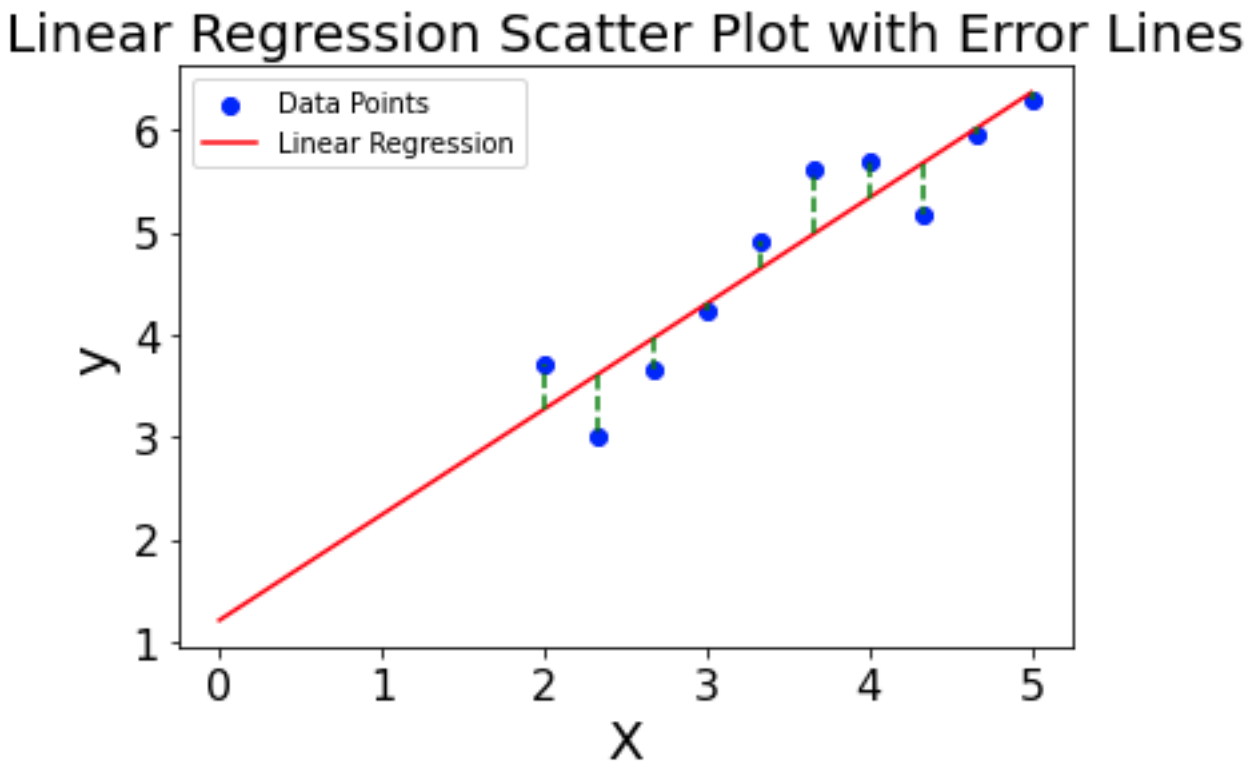
$$\eta = \sum_i e_i^2$$

Let's differentiate this expression with respect to "m" and set the derivative equal to zero to find the optimal value for the slope "m":

$$\frac{d\eta}{dm} = \sum_i (mx_i + c - y_i)^2 = 0$$

$$m = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sum_i (x - \bar{x})}$$

Slope and Intercept



The slope "m" is calculated using the formula:

$$m = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sum_i (x - \bar{x})}$$

The intercept "c" is calculated using the formula:

$$c = \bar{y} - m\,\bar{x}$$

x_i is an individual data point of the input feature.

$\bar{x}$ is the mean of the input feature.

y_i is the corresponding target value.

$\bar{y}$ is the mean of the target values.

Scikit-Learn Python Package

sklearn.linear_model.LinearRegression

class sklearn.linear_model.**LinearRegression**(*, *fit_intercept=True*, *copy_X=True*, *n_jobs=None*, *positive=False*)

[source]

Ordinary least squares Linear Regression.

LinearRegression fits a linear model with coefficients w = (w1, ..., wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

In scikit-learn, linear regression is implemented through the LinearRegression class, which represents a linear regression model.

Refrence: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

Example

Dataset: Student Study Hours and Exam Scores

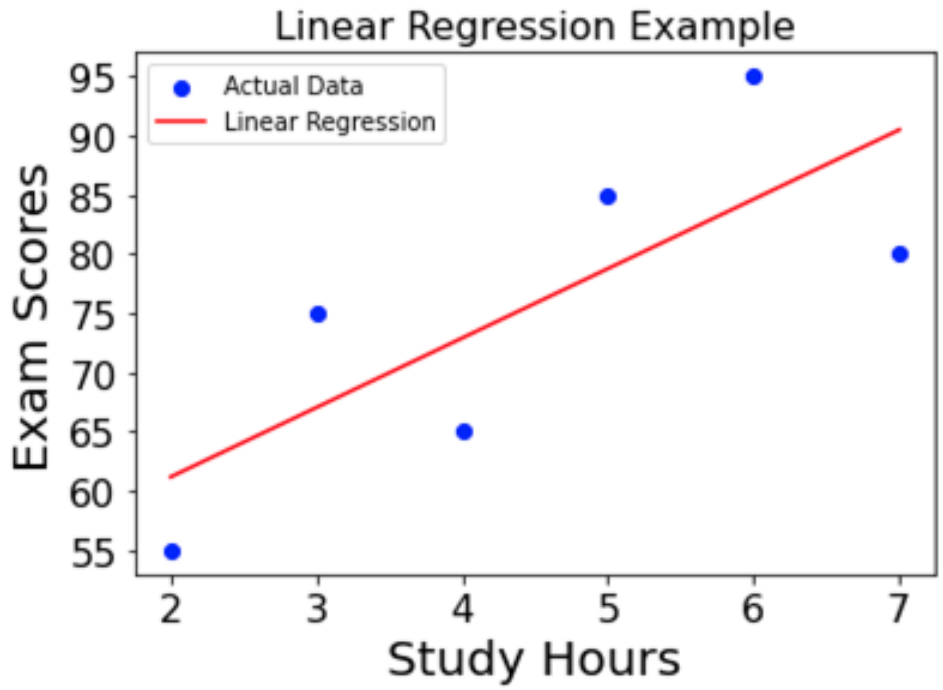
This dataset contains information about the number of hours students studied and their corresponding exam scores. The goal is to predict the exam score based on the number of study hours.

Study Hours (x)	Exam Scores (y)
2	55
3	75
4	65
5	85
6	95
7	80

Example

```python
import numpy as np
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

# Dataset
study_hours = np.array([2, 3, 4, 5, 6, 7]).reshape(-1, 1)
exam_scores = np.array([55, 75, 65, 85, 95, 80])

# Initialize the Linear Regression model
model = LinearRegression()

# Train the model
model.fit(study_hours, exam_scores)

# Make predictions
predicted_scores = model.predict(study_hours)

# Plot the data and the linear regression line
plt.scatter(study_hours, exam_scores, color='blue', label='Actual Data')
plt.plot(study_hours, predicted_scores, color='red', label='Linear Regression')
plt.xlabel('Study Hours',fontsize=20)
plt.ylabel('Exam Scores',fontsize=20)
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
plt.title('Linear Regression Example',fontsize=16)
plt.legend()
plt.show()

# Predict for new data
new_study_hours = np.array([[8]])
predicted_score_new = model.predict(new_study_hours)
print(f"Predicted Exam Score for 8 Study Hours: {predicted_score_new[0]:.2f}")
```

```
array([[2],
       [3],
       [4],
       [5],
       [6]])
```




```
Predicted Exam Score for 8 Study Hours: 96.33
```

Multiple Linear Regression

Multiple regression equation, is a mathematical model used in statistics and data analysis to describe the relationship between a dependent variable and two or more independent variables.

The general form of a multiple linear equation is as follows:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \cdots + \beta_k X_k \ldots + \beta_N X_N$$

Y represents the dependent variable you want to predict.

β_0 is the intercept or constant term, which represents the value of Y when all independent variables are zero.

$\beta_1, \beta_2, \beta_3, \ldots, \beta_k \ldots \beta_N$ are the coefficients of the independent variables $X_1, X_2, X_3, \ldots, X_k \ldots, X_N$ respectively.

The goal in multiple linear regression is to estimate the values of the coefficients ($\beta_0, \beta_1, \beta_2, \ldots, \beta_k, \ldots \beta_N,$) that best fit the data.

Once you have these coefficient estimates, you can use the equation to make predictions for Y based on specific values of the independent variables.

Reference: https://www.sciencedirect.com/topics/medicine-and-dentistry/multiple-linear-regression-analysis

Multiple Linear Regression

Let's assume that the multiple linear regression model predicts $\widehat{Y}_i$ for X_i value where actual value is Y_i.

The error term for the data point i is given by

$$e_i = Y_i - \widehat{Y}_i$$

In multiple linear regression model, the sum of squared residuals η is minimized.

$$\eta = \sum_{\{i=1\}}^{N} e_i^2 = \sum_{\{i=1\}}^{N} (Y_i - \widehat{Y}_i)^2$$

The closed-form of the solution of minimizing η is given by

$$\hat{\beta} = (X^T X)^{-1} X^T Y$$

$\hat{\beta}$ is a vector of estimated beta values and X^T is transpose of X matrix.

Example

```python
import numpy as np
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

# Dataset
# Generate some sample data
np.random.seed(0)
X1 = np.random.rand(100, 1)  # Independent variable 1
X2 = np.random.rand(100, 1)  # Independent variable 2
X= np.hstack((X1, X2)) #Independent variables in a single list
y= 2*X1 + 3*X2 + np.random.rand(100, 1)  # Dependent variable with noise

# Initialize the Linear Regression model
model = LinearRegression()

# Train the model
model.fit(X, y)


# Predict for new data
X_new = np.array([[8.5,2.5]])
y_pred = model.predict(X_pred)
print(f"Predicted y when X1=8.5 and X2 = 2.5: {y_pred[0][0]:.2f}")
```

```
Predicted y when X1=8.5 and X2 = 2.5: 21.95
```

Error Analysis in Regression Models

- Error analysis in regression models is a crucial step in assessing the performance and reliability of your predictive models.
- It involves analyzing the discrepancies between the predicted values generated by your regression model and the actual observed values in your dataset.
- We will discuss three parameters in this lecture for error analysis.

    1. Mean Absolute Error (MAE)
    2. Mean Squared Error (MSE)
    3. Root Mean Squared Error (RMSE)
    4. Coefficient of Determination (R^2)

Mean Average Error (MAE)

MAE measures the average absolute difference between the predicted values generated by the model and the actual observed values in the dataset.

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N} |y_{\text{actual}} - y_{\text{predicted}}|$$

Where:

- N is the number of data points in your dataset.
- Σ represents the sum over all data points.
- $| y_{\text{actual}} - y_{\text{predicted}} |$ calculates the absolute difference between the actual (observed) value and the predicted value for each data point.

Mean Squared Error (MSE)

MSE quantifies the average of the squared differences between the predicted values generated by the model and the actual observed values in the dataset.

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - f_i)^2$$

y_i: actual value f_i: predicted value

Where:

- N is the number of data points in your dataset.
- Σ represents the sum over all data points.
- $(y_{\text{actual}} - y_{\text{predicted}})^2$ calculates the square of difference between the actual (observed) value and the predicted value for each data point.

Root Mean Squared Error (RMSE)

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - f_i)^2} = \sqrt{\text{MSE}}$$

y_i: actual value f_i: predicted value

Where:

- N is the number of data points in your dataset.
- Σ represents the sum over all data points.
- $(y_{\text{actual}} - y_{\text{predicted}})^2$ calculates the square of difference between the actual (observed) value and the predicted value for each data point.

Coefficient of Determination (R^2)

$$R^2 = 1 - \frac{SSE}{SST}$$

R^2: Coefficient of Determination (R-squared)

SSE: Sum of Squared Errors (also known as the residual sum of squares), which represents the sum of the squared differences between the observed values and the predicted values.

SST: Total Sum of Squares, which represents the sum of squared differences between the observed values and the mean of the observed values.

$$SSE = \sum_i (y_i - f_i)^2$$

$$SST = \sum_i (y_i - \bar{y})^2$$

$$\bar{y} = \frac{1}{N}\sum_i (y_i - \bar{y})^2$$

y_i: actual value f_i: predicted value $\bar{y}$: average

Example

The primary purpose of the test set is to provide an unbiased evaluation of a final trained model.

By using data that the model has not seen during training, you can assess how well the model generalizes to new, unseen examples.

```python
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Sample data
#Training set
X_train = np.array([ 4.21052632,  2.63157895,  5.78947368,  1.57894737,  9.47368421,
        8.42105263,  6.84210526,  1.05263158,  4.73684211, 10.0],).reshape(-1, 1)
y_train = np.array([ 9.21461493,  4.30860213, 15.48749438,  8.63968114, 20.57350382,
       20.83026341, 14.92756056,  5.06273913, 11.29488121, 19.29180852])

#Test set
X_test = np.array([0.0, 8.94736842, 7.89473684, 0.52631579]).reshape(-1, 1)
y_test = np.array([ 4.52810469, 18.48442031, 17.45682234,  2.852946  ])

# Create a linear regression model
model = LinearRegression()

# Fit the model to the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Calculate Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, y_pred)
print(f"Mean Absolute Error (MAE): {mae:.2f}")

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error (MSE): {mse:.2f}")

# Calculate R-squared (R²)
r2 = r2_score(y_test, y_pred)
print(f"R-squared (R²): {r2:.2f}")
```

```
Mean Absolute Error (MAE): 0.88
Mean Squared Error (MSE): 1.07
R-squared (R²): 0.98
```