Simple Linear Regression

In [14]:

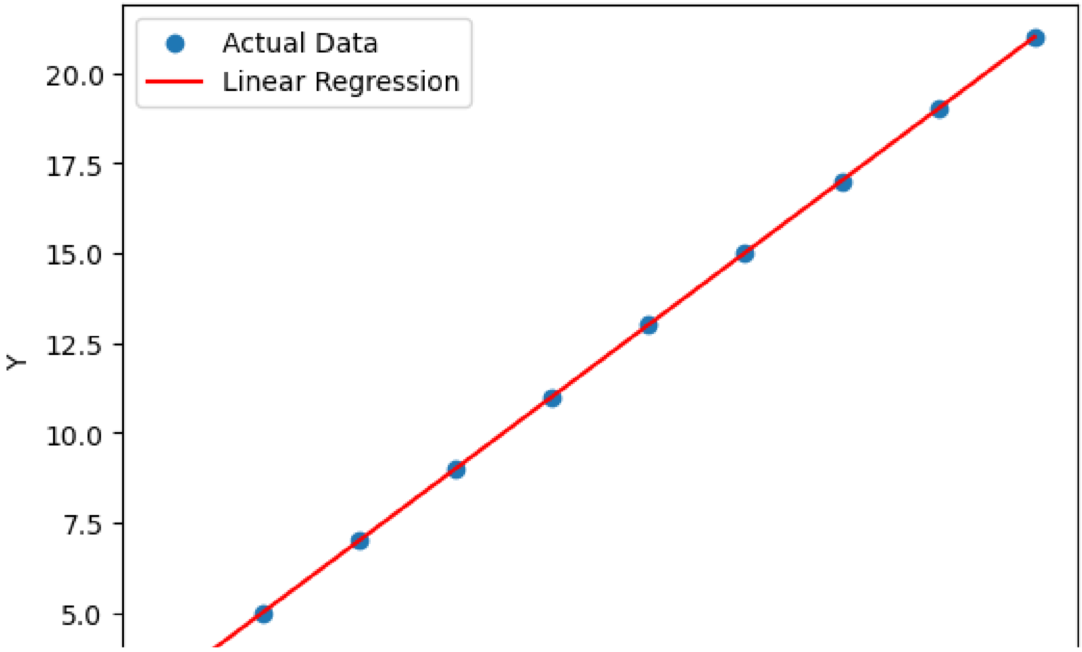
```
import numpy as np
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

# Generate synthetic data
X = np.arange(1, 11).reshape(-1, 1)
y = 2 * X + 1

# Create and train a simple linear regression model
model = LinearRegression()
model.fit(X, y)

# Make predictions
y_pred = model.predict(X)

# Plot the data and regression line
plt.scatter(X, y, label="Actual Data")
plt.plot(X, y_pred, color='red', label="Linear Regression")
plt.xlabel("X")
plt.ylabel("Y")
plt.legend()
plt.show()
```



Multiple Linear Regression

In [15]:
```
# Generate synthetic data with two features
X = np.array([[1, 2], [2, 3], [3, 4], [4, 5], [5, 6]])
y = 2 * X[:, 0] + 3 * X[:, 1] + 1

# Create and train a multiple linear regression model
model = LinearRegression()
model.fit(X, y)

# Make predictions
y_pred = model.predict(X)
y_pred
```

Out[15]:
```
array([ 9., 14., 19., 24., 29.])
```

Polynomial Regression:

In [16]:

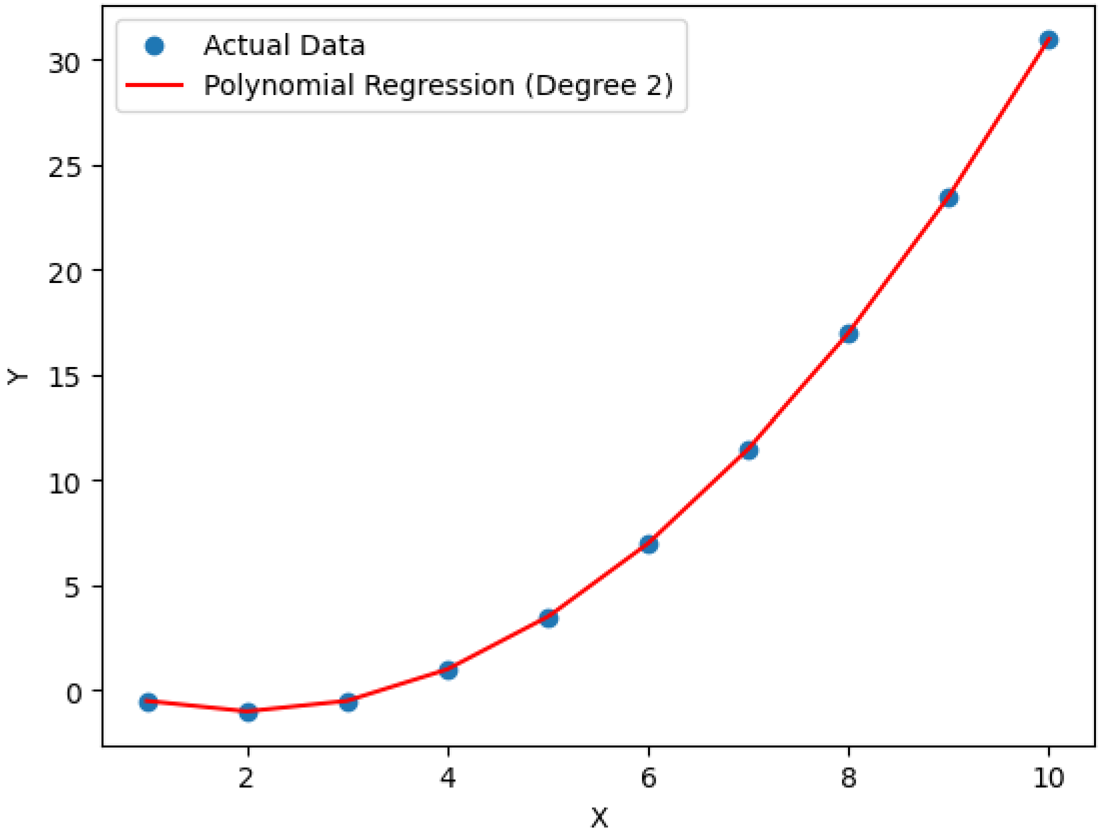
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline

# Generate synthetic data
X = np.arange(1, 11).reshape(-1, 1)
y = 0.5 * X**2 - 2 * X + 1

# Create a polynomial regression model
degree = 2  # Degree of the polynomial
polyreg = make_pipeline(PolynomialFeatures(degree), LinearRegression())
polyreg.fit(X, y)

# Make predictions
y_pred = polyreg.predict(X)

# Plot the data and polynomial regression curve
plt.scatter(X, y, label="Actual Data")
plt.plot(X, y_pred, color='red', label=f"Polynomial Regression (Degree
plt.xlabel("X")
plt.ylabel("Y")
plt.legend()
plt.show()
```



Evalution Matrix

R Squared[R2]

In [17]:
```
from sklearn.metrics import mean_absolute_error
# Actual values (ground truth)
actual = [10, 20, 30, 40, 50]

# Predicted values from your model
predicted = [12, 18, 28, 37, 45]

# Calculate the Mean Absolute Error
mae = mean_absolute_error(actual, predicted)

print(f"Mean Absolute Error: {mae}")
```

```
Mean Absolute Error: 2.8
```

mean_absolute_error

In [1]:
```
import numpy as np
from sklearn.metrics import mean_squared_error
from math import sqrt

# Actual values
actual = [10, 12, 15, 18, 20]

# Predicted values
predicted = [9, 11, 14, 17, 19]

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(actual, predicted)

# Calculate RMSE
rmse = sqrt(mse)

print("RMSE:", rmse)
```

```
RMSE: 1.0
```

R-squared (R2) score

In [19]:
```
from sklearn.metrics import r2_score

# Assuming you have observed actual and predicted values
actual_values = [10, 20, 30, 40, 50]
predicted_values = [12, 18, 28, 38, 52]

# Calculate the R-squared (R2) score
r2 = r2_score(actual_values, predicted_values)

print(f"R-squared (R2) Score: {r2}")
```

```
R-squared (R2) Score: 0.98
```

Cross Validation

Hold-Out Validation:

In [20]:
```
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris  # You can replace this with you

# Load your dataset (replace with your data loading code)
data = load_iris()

# Split the dataset into features (X) and target labels (y)
X = data.data
y = data.target

# Split the data into a training set and a test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2
X
```

Out[20]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
```

Leave-One-Out Cross-Validation (LOOCV):

In [21]:
```
from sklearn.model_selection import LeaveOneOut
from sklea rn.linear_model import LinearRegression
loo = LeaveOneOut()
total_mse = 0

for train_index, test_index in loo.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    model = LinearRegression()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    mse = mean_squared_error(y_test, y_pred)
    total_mse += mse

average_mse = total_mse / len(X)
print("Average Mean Squared Error (LOOCV):", average_mse)
```

```
Average Mean Squared Error (LOOCV): 0.04954185325184582
```

K-Fold Cross-Validation:

In [22]:
```
from sklearn.model_selection import KFold

k = 5  # You can choose the number of folds
kf = KFold(n_splits=k)
total_mse = 0

for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    model = LinearRegression()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    mse = mean_squared_error(y_test, y_pred)
    total_mse += mse

average_mse = total_mse / k
print(f"Average Mean Squared Error ({k}-fold CV):", average_mse)
```

```
Average Mean Squared Error (5-fold CV): 0.06897012554624687
```

In [4]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import numpy as np

# Generate some example data
X = np.array([[1, 2], [2, 3], [3, 4], [4, 5], [5, 6], [6, 7]])
y = np.array([0, 0, 1, 1, 2, 2])

# Specify the number of folds
k = 3

# Initialize the KFold class
kf = KFold(n_splits=k, shuffle=True, random_state=42)

# Initialize a variable to keep track of the average accuracy
avg_accuracy = 0

# Loop through each fold
for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Train your machine learning model on the training data
    # For simplicity, let's use a basic classifier (e.g., Decision Tree
    from sklearn.tree import DecisionTreeClassifier
    clf = DecisionTreeClassifier()
    clf.fit(X_train, y_train)

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

    # Calculate the accuracy for this fold
    accuracy = accuracy_score(y_test, y_pred)
    avg_accuracy += accuracy

# Calculate the average accuracy across all folds
avg_accuracy /= k

print("Average Accuracy:", avg_accuracy)

```

```
Average Accuracy: 0.5
```

In [8]:
```
from sklearn.model_selection import KFold
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import numpy as np

X = np.array([[1, 2], [2, 3], [3, 4], [4, 5], [5, 6], [6, 7]])
y = np.array([0, 0, 1, 1, 2, 2])
k = 3
kf = KFold(n_splits=k, shuffle=True, random_state=42)

avg_accuracy = sum(accuracy_score(y[test], DecisionTreeClassifier().fit

print("Average Accuracy:", avg_accuracy)

```

```
Average Accuracy: 0.5
```

Regression Algorithem

Random Forest Regressor¶

In [29]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import make_regression
X, y = make_regression(n_features=4, n_informative=2, random_state=0, s
rfr = RandomForestRegressor(max_depth=3)
rfr.fit(X, y)
print(rfr.predict([[0, 1, 0, 1]]))
```

```
[33.07556472]
```

Lasso Regression

In [30]:
```
from sklearn import linear_model
import numpy as np
rng = np.random.RandomState(1)
X = np.sort(5 * rng.rand(80, 1), axis=0)
y = np.sin(X).ravel()
y[::5] += 3 * (0.5 - rng.rand(16))
# Fit regression model
lassoReg = linear_model.Lasso(alpha=0.1)
lassoReg.fit(X,y)
# Predict
X_test = np.arange(0.0, 5.0, 1)[:, np.newaxis]
lassoReg.predict(X_test)
```

Out[30]:
```
array([ 0.78305084,  0.49957596,  0.21610108, -0.0673738 , -0.35084868])
```

Support Vector Regression

In [31]:
```
from sklearn.svm import SVR
import numpy as np
rng = np.random.RandomState(1)
X = np.sort(5 * rng.rand(80, 1), axis=0)
y = np.sin(X).ravel()
y[::5] += 3 * (0.5 - rng.rand(16))
# Fit regression model
svr = SVR().fit(X, y)
# Predict
X_test = np.arange(0.0, 5.0, 1)[:, np.newaxis]
svr.predict(X_test)
```

Out[31]:
```
array([-0.07840308,  0.78077042,  0.81326895,  0.08638149, -0.6928019 ])
```

TypesOFClassifications

Binary Classification

In [40]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

X = [[1.2, 2.3], [2.4, 3.5], [3.6, 4.7], [4.8, 5.9], [5.0, 6.1]]
y = [0, 1, 0, 1, 1]
# Load and preprocess your data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Create and train a binary classification model (e.g., logistic regres
model = LogisticRegression()
model.fit(X_train, y_train)


# Make predictions
predictions = model.predict(X_test)
predictions
```

Out[40]:
```
array([1])
```

multi-class classification

In [62]:

```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

# Load and preprocess your data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Create and train a multi-class classification model (e.g., support ve
model = SVC()
model.fit(X_train, y_train)

# Make predictions
predictions = model.predict(X_test)
predictions
```

Out[62]:

```
array([0, 0, 2, 2, 0, 2, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 2, 1, 2, 0, 0, 0,
       2, 0, 0, 1, 1, 2, 2, 1])
```

In [48]:

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report

# Load the Iris dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3

# Evaluate the classifier's performance
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred, target_names=iris.target
# Print the results
print("Accuracy:", accuracy)
print("Classification Report:\n", report)

```

```
Accuracy: 1.0
Classification Report:
               precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

multi-label classification

In [60]:
```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load a sample dataset (Iris dataset as an example)
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Create a Support Vector Machine (SVM) classifier
clf = SVC(kernel='linear', C=1)

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

# Make predictions on the test data
y_pred = clf.predict
y_pred
```

Out[60]:
```
<bound method BaseSVC.predict of SVC(C=1, kernel='linear')>
```

Confusion Matix

In [34]:
```
# Import the necessary libraries
from sklearn.metrics import confusion_matrix
import numpy as np

# Actual and predicted labels (replace these with your actual data)
actual = np.array([1, 0, 1, 1, 0, 1, 0, 1, 0, 0])
predicted = np.array([1, 0, 1, 0, 1, 1, 0, 1, 0, 1])

# Create the confusion matrix
cm = confusion_matrix(actual, predicted)

# Print the confusion matrix
print("Confusion Matrix:")
print(cm)

```

```
Confusion Matrix:
[[3 2]
 [1 4]]
```

In [33]:

```
import numpy as np

# Given confusion matrix
confusion_matrix = np.array([
    [13, 45],
    [0, 15]
])

# (i) Accuracy
total_samples = np.sum(confusion_matrix)
correct_predictions = np.trace(confusion_matrix)
accuracy = correct_predictions / total_samples
print(f"(i) Accuracy: {accuracy:.2f}")

# (ii) Precision for Class 1
precision_class_1 = confusion_matrix[1, 1] / (confusion_matrix[0, 1] +
print(f"(ii) Precision for Class 1: {precision_class_1:.2f}")

# (iii) Recall for Class 1
recall_class_1 = confusion_matrix[1, 1] / (confusion_matrix[1, 0] + con
print(f"(iii) Recall for Class 1: {recall_class_1:.2f}")

# (iv) Specificity for Class 0
specificity_class_0 = confusion_matrix[0, 0] / (confusion_matrix[0, 0]
print(f"(iv) Specificity for Class 0: {specificity_class_0:.2f}")

# (v) F1-Score for Class 1
f1_score_class_1 = 2 * (precision_class_1 * recall_class_1) / (precisio
print(f"(v) F1-Score for Class 1: {f1_score_class_1:.2f}")

```

```
(i) Accuracy: 0.38
(ii) Precision for Class 1: 0.25
(iii) Recall for Class 1: 1.00
(iv) Specificity for Class 0: 0.22
(v) F1-Score for Class 1: 0.40
```

decision tree

In [58]:

```
# Import the necessary libraries
from sklearn import tree

# Create a dataset for training the decision tree
# In this example, we have two features (X) and a target variable (y)
X = [[0, 0], [1, 1]]
y = [0, 1]

# Create a decision tree classifier
clf = tree.DecisionTreeClassifier()

# Train the classifier on the dataset
clf = clf.fit(X, y)

# Make predictions using the trained decision tree
predictions = clf.predict([[2, 2], [0.5, 0.5]])

# Print the predictions
print(predictions)

```

```
[1 0]
```

decision tree classifier

In [59]:

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Load a sample dataset (Iris dataset as an example)
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Create a Decision Tree Classifier
clf = DecisionTreeClassifier()

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

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

# Calculate the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

```

```
Accuracy: 100.00%
```

In [ ]:
```
1
```