Boston Housing

In [2]:
```
# Import necessary libraries
import numpy as np
import pandas as pd
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns

# Load the Boston Housing Prices dataset
boston = load_boston()

# Convert the dataset to a DataFrame
data = pd.DataFrame(data=np.c_[boston['data'], boston['target']], colum

# Data Exploration
# Display the first few rows of the dataset
print(data.head())

# Summary statistics
print(data.describe())


```

```
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0
1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0
2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0
3  0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0
4  0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  222.0

   PTRATIO       B  LSTAT  target
0     15.3  396.90   4.98    24.0
1     17.8  396.90   9.14    21.6
2     17.8  392.83   4.03    34.7
3     18.7  394.63   2.94    33.4
4     18.7  396.90   5.33    36.2
             CRIM          ZN       INDUS        CHAS         NOX        
RM  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000
000
mean     3.613524   11.363636   11.136779    0.069170    0.554695    6.284
634
std      8.601545   23.322453    6.860353    0.253994    0.115878    0.702
617
min      0.006320    0.000000    0.460000    0.000000    0.385000    3.561
000
25%      0.082045    0.000000    5.190000    0.000000    0.449000    5.885
500
50%      0.256510    0.000000    9.690000    0.000000    0.538000    6.208
500
75%      3.677083   12.500000   18.100000    0.000000    0.624000    6.623
500
max     88.976200  100.000000   27.740000    1.000000    0.871000    8.780
000

              AGE         DIS         RAD         TAX     PTRATIO        
B  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000
000
mean    68.574901    3.795043    9.549407  408.237154   18.455534  356.674
032
std     28.148861    2.105710    8.707259  168.537116    2.164946   91.294
864
min      2.900000    1.129600    1.000000  187.000000   12.600000    0.320
000
25%     45.025000    2.100175    4.000000  279.000000   17.400000  375.377
500
50%     77.500000    3.207450    5.000000  330.000000   19.050000  391.440
000
75%     94.075000    5.188425   24.000000  666.000000   20.200000  396.225
000
max    100.000000   12.126500   24.000000  711.000000   22.000000  396.900
000

            LSTAT      target
count  506.000000  506.000000
mean    12.653063   22.532806
std      7.141062    9.197104
min      1.730000    5.000000
25%      6.950000   17.025000
50%     11.360000   21.200000
75%     16.955000   25.000000
max     37.970000   50.000000
```

```
C:\Users\Anusha V\anaconda3\lib\site-packages\sklearn\utils\deprecation.p
y:87: FutureWarning: Function load_boston is deprecated; `load_boston` is
deprecated in 1.0 and will be removed in 1.2.

    The Boston housing prices dataset has an ethical problem. You can refe
r to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use of
this
    dataset unless the purpose of the code is to study and educate about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np


        data_url = "http://lib.stat.cmu.edu/datasets/boston"
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=Non
e)
        data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
        target = raw_df.values[1::2, 2]

    Alternative datasets include the California housing dataset (i.e.
    :func:`~sklearn.datasets.fetch_california_housing`) and the Ames housi
ng
    dataset. You can load the datasets as follows::

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and::

        from sklearn.datasets import fetch_openml
        housing = fetch_openml(name="house_prices", as_frame=True)

    for the Ames housing dataset.

  warnings.warn(msg, category=FutureWarning)
```

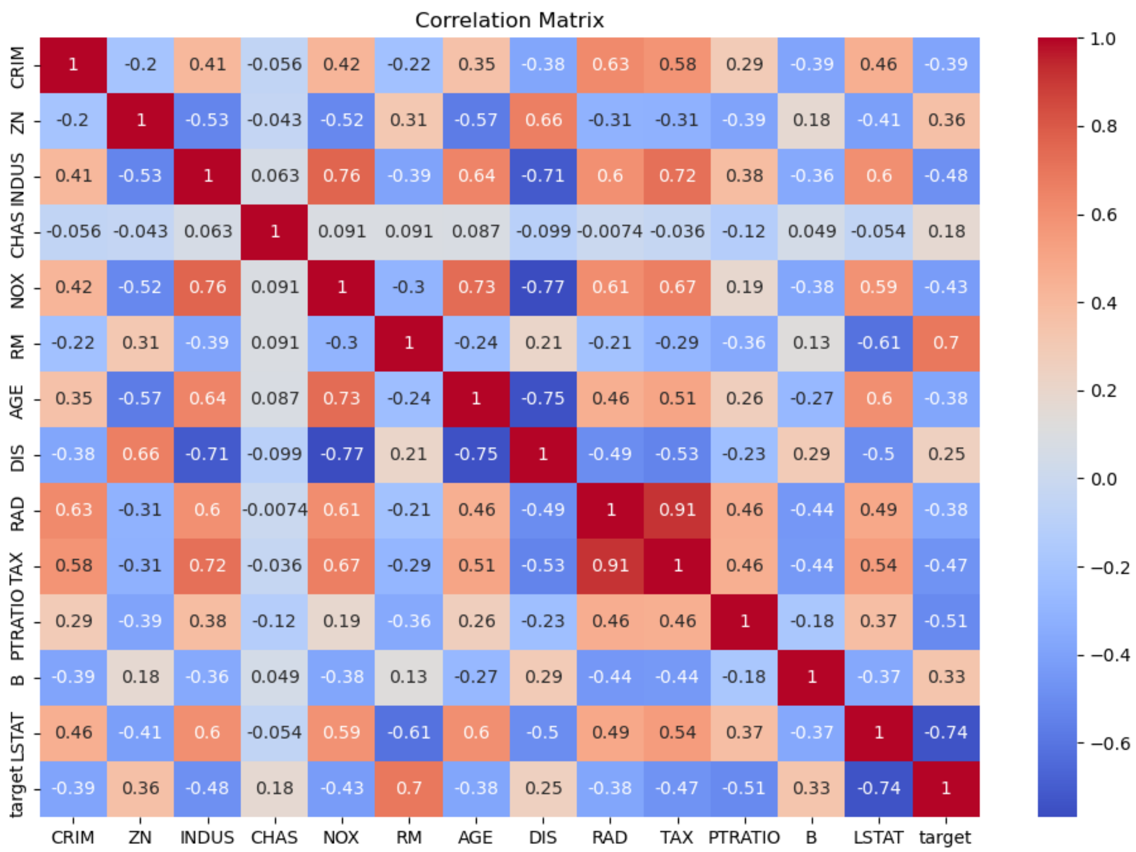
In [4]:

```
# Import necessary libraries
from sklearn.model_selection import train_test_split

# 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

# Check the shape of the split datasets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)

```

```
X_train shape: (404, 13)
X_test shape: (102, 13)
y_train shape: (404,)
y_test shape: (102,)
```

In [3]:

```python
# Correlation matrix
correlation_matrix = data.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()

# Data Preprocessing
# Split the data into features (X) and target (y)
X = data.drop('target', axis=1)
y = data['target']

# Standardize the features (mean=0, std=1)
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Data Splitting
# 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

# Check the shape of the split datasets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)

```




```
X_train shape: (404, 13)
X_test shape: (102, 13)
y_train shape: (404,)
y_test shape: (102,)
```

In [9]:
```
##Evaluation matric

##Import necessary libraries
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Create a LinearRegression model
model = LinearRegression()

# Fit the model on the training data
model.fit(X_train[:, 5:6], y_train)  # Using the 6th feature as an exam

# Make predictions on the test data
y_pred = model.predict(X_test[:, 5:6])

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print the model's coefficients and performance metrics
print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)
print("Mean Squared Error (MSE):", mse)
print("R-squared (R2) Score:", r2)

from sklearn.metrics import mean_absolute_error

# Calculate the Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, y_pred)

# Print the MAE
print("Mean Absolute Error (MAE):", mae)

```

```
Coefficients: [6.56178323]
Intercept: 22.50433758446674
Mean Squared Error (MSE): 46.144775347317264
R-squared (R2) Score: 0.3707569232254778
Mean Absolute Error (MAE): 4.478335832064148
```

In [16]:
```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression

# Create a LinearRegression model
model = LinearRegression()

# Perform cross-validation with 5 folds
scores = cross_val_score(model, X, y, cv=5, scoring='neg_mean_squared_e

# Calculate the mean of the negative mean squared error scores
mean_mse = -scores.mean()

# Print the mean squared error
print("Mean Squared Error (MSE) for Cross-Validation:", mean_mse)

```

```
Mean Squared Error (MSE) for Cross-Validation: 37.13180746769907
```

In [8]:

```
##Holdout Method (Train-Test Split):

from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# 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 and fit the model
model = LinearRegression()
model.fit(X_train[:, 5:6], y_train)

# Make predictions on the test data
y_pred = model.predict(X_test[:, 5:6])

# Evaluate the model using mean squared error
mse = mean_squar ed_error(y_test, y_pred)
print("Holdout Method - Mean Squared Error (MSE):", mse)

```

```
Holdout Method - Mean Squared Error (MSE): 46.144775347317264
```

In [11]:

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

from sklearn.model_selection import LeaveOneOut

loo = LeaveOneOut()
mse_list = []

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[:, 5:6], y_train)

    y_pred = model.predict(X_test[:, 5:6])
    mse = mean_squared_error(y_test, y_pred)
    mse_list.append(mse)

average_mse_loocv = np.mean(mse_list)
print("Leave-One-Out Cross-Validation (LOOCV) - Average MSE:", average_

```

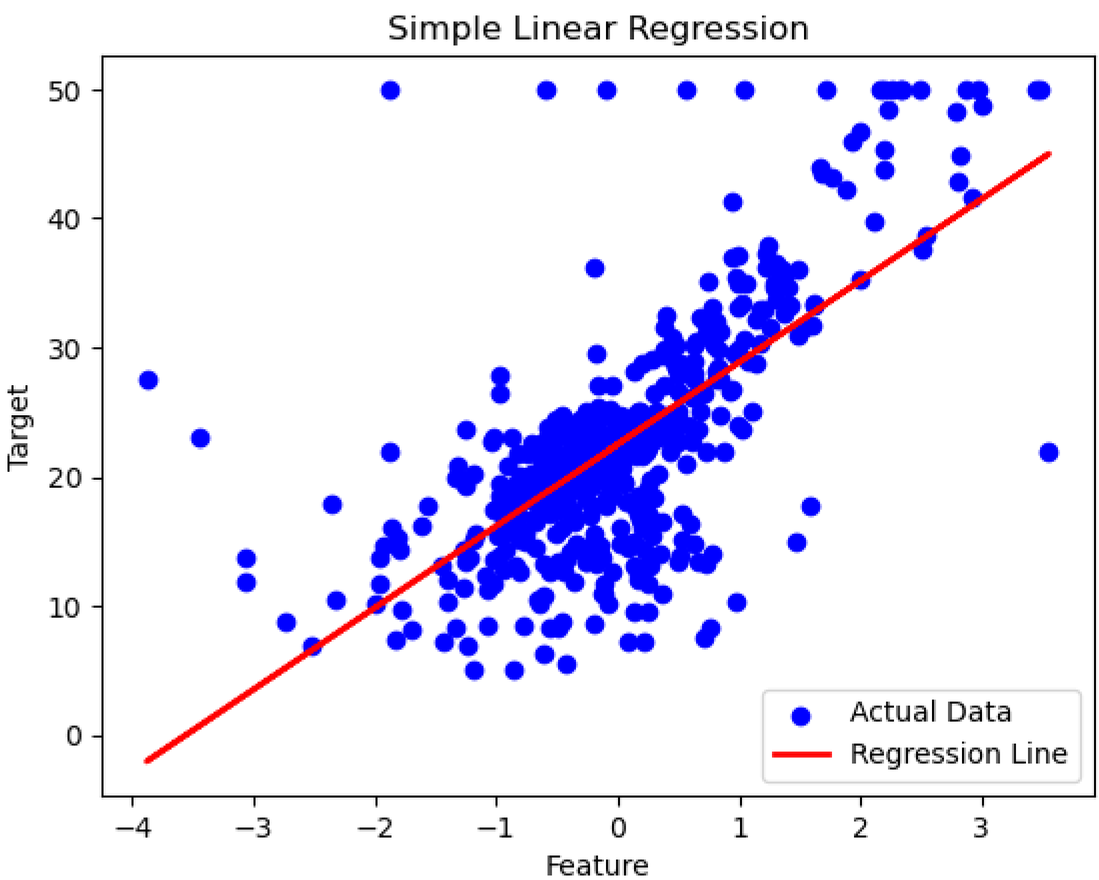
```
Leave-One-Out Cross-Validation (LOOCV) - Average MSE: 44.216664193967986
```

In [12]:
```
##K-Fold Cross-Validation (e.g., K=5):

from sklearn.model_selection import KFold

k = 5
kf = KFold(n_splits=k, shuffle=True, random_state=42)
mse_list = []

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[:, 5:6], y_train)

    y_pred = model.predict(X_test[:, 5:6])
    mse = mean_squared_error(y_test, y_pred)
    mse_list.append(mse)

average_mse_kfold = np.mean(mse_list)
print(f"{k}-Fold Cross-Validation - Average MSE:", average_mse_kfold)

```

```
5-Fold Cross-Validation - Average MSE: 43.84204126558527
```

In [14]:

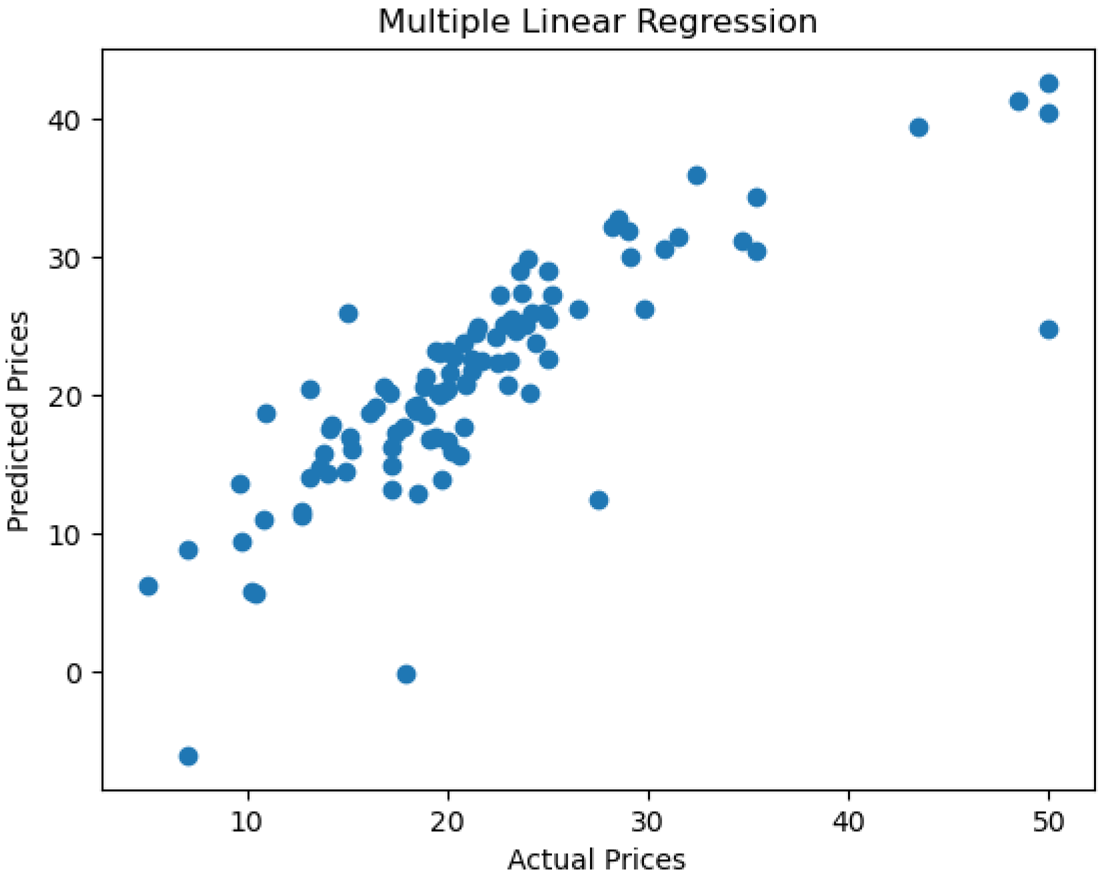
```
##Simple liner Regression


import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Create a LinearRegression model
model = LinearRegression()

# Fit the model on the training data
model.fit(X_train[:, 5:6], y_train)  # Using the 6th feature as an exam

# Make predictions on the entire dataset
y_pred = model.predict(X[:, 5:6])

# Create a scatter plot of the actual data points
plt.scatter(X[:, 5], y, color='blue', label='Actual Data')

# Overlay the regression line
plt.plot(X[:, 5], y_pred, color='red', linewidth=2, label='Regression L

# Set labels and title
plt.xlabel('Feature')
plt.ylabel('Target')
plt.title('Simple Linear Regression')

# Add a legend
plt.legend()

# Show the plot
plt.show()

```



In [16]:

```
##multiple linear regression

import numpy as np
import pandas as pd
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

# Load the Boston Housing Prices dataset
boston = load_boston()
data = pd.DataFrame(data=np.c_[boston['data'], boston['target']], colum

# Split the data into features (X) and target (y)
X = data.drop('target', axis=1)
y = data['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 Linear Regression model
model = LinearRegression()

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

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

# Visualize the results
plt.scatter(y_test, y_pred)
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Multiple Linear Regression")
plt.show()

```

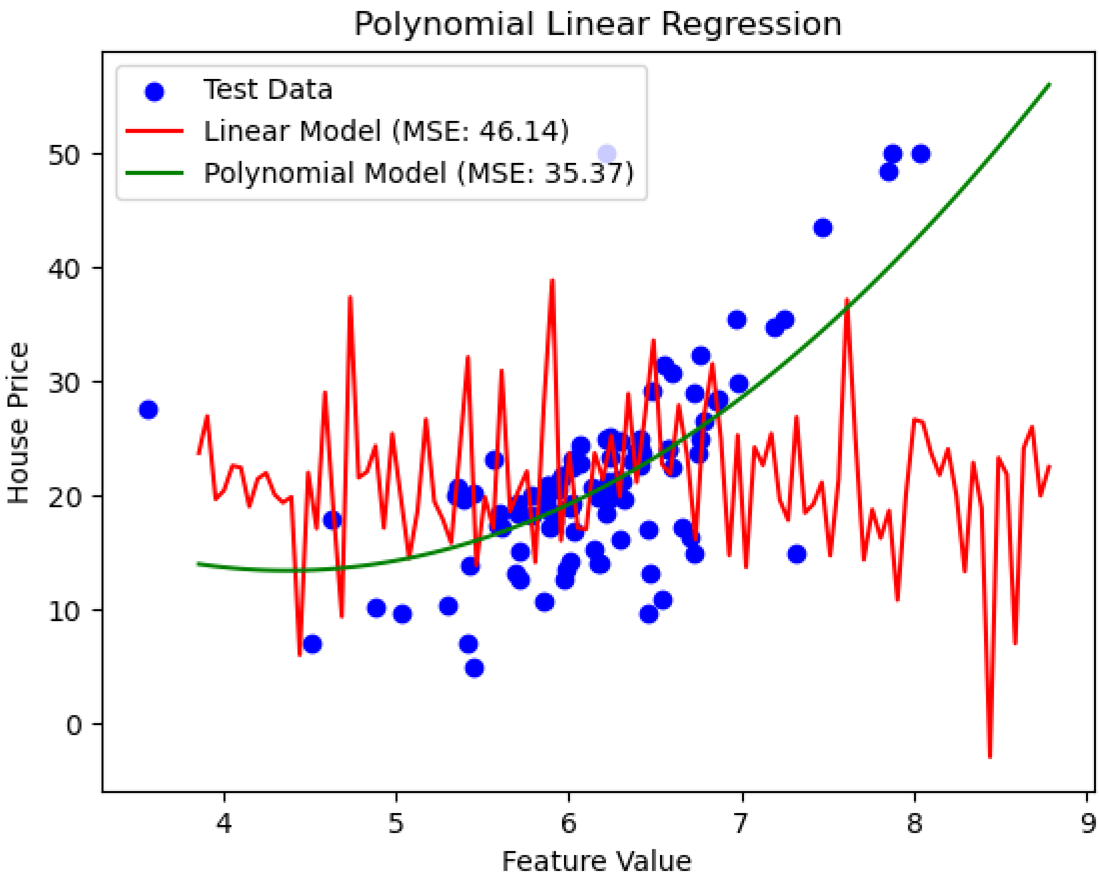
```
C:\Users\Anusha V\anaconda3\lib\site-packages\sklearn\utils\deprecation.p
y:87: FutureWarning: Function load_boston is deprecated; `load_boston` is
deprecated in 1.0 and will be removed in 1.2.

    The Boston housing prices dataset has an ethical problem. You can refe
r to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use of
this
    dataset unless the purpose of the code is to study and educate about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np


        data_url = "http://lib.stat.cmu.edu/datasets/boston"
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=Non
e)
        data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
        target = raw_df.values[1::2, 2]

    Alternative datasets include the California housing dataset (i.e.
    :func:`~sklearn.datasets.fetch_california_housing`) and the Ames housi
ng
    dataset. You can load the datasets as follows::

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and::

        from sklearn.datasets import fetch_openml
        housing = fetch_openml(name="house_prices", as_frame=True)

    for the Ames housing dataset.

  warnings.warn(msg, category=FutureWarning)
```


Multiple Linear Regression
Predicted Prices
Actual Prices
40
30
20
10
0
10
20
30
40
50

In [24]:

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

# 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 LinearRegression model
linear_model = LinearRegression()

# Select a feature for polynomial regression (e.g., the 6th feature)
feature_idx = 5

# Extract the chosen feature as a Pandas Series
X_train_feature = X_train.iloc[:, feature_idx].values.reshape(-1, 1)
X_test_feature = X_test.iloc[:, feature_idx].values.reshape(-1, 1)

# Ensure that X_train_feature and y_train have the same number of data
X_train_feature = X_train_feature[:len(y_train)]

# Fit the linear model
linear_model.fit(X_train_feature, y_train)

# Generate predictions for the linear model
y_pred_linear = linear_model.predict(X_test_feature)

# Create a range of values for the x-axis (for plotting)
x_range = np.linspace(X_train_feature.min(), X_train_feature.max(), num

# Create a PolynomialFeatures transformer (e.g., quadratic degree=2)
poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X_train_feature)

# Fit a polynomial regression model
poly_model = LinearRegression()
poly_model.fit(X_poly, y_train)

# Generate predictions for the polynomial model
X_poly_range = poly.transform(x_range)
y_pred_poly = poly_model.predict(X_poly_range)

# Calculate the Mean Squared Error (MSE) for both models
mse_linear = mean_squared_error(y_test, y_pred_linear)
mse_poly = mean_squared_error(y_test, poly_model.predict(poly.transform

# Plot the results
plt.scatter(X_test_feature, y_test, label='Test Data', color='b')
plt.plot(x_range, y_pred_linear, label=f'Linear Model (MSE: {mse_linear
plt.plot(x_range, y_pred_poly, label=f'Polynomial Model (MSE: {mse_poly
plt.xlabel("Feature Value")
plt.ylabel("House Price")
plt.legend()
plt.title("Polynomial Linear Regression")
plt.show()
```

```
58
```



Build a SVM Model in python for Fish

dataset from Kaggl

In [2]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

data = pd.read_csv(r"C:\Users\Anusha V\Downloads\fish.csv")
X = data[['nofish', 'livebait', 'camper']]
y = data['persons']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
svm_model = SVC()
svm_model.fit(X_train, y_train)
y_pred = svm_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of the SVM model: {accuracy}")

```

```
C:\Users\Anusha V\anaconda3\lib\site-packages\scipy\__init__.py:155: UserW
arning: A NumPy version >=1.18.5 and <1.25.0 is required for this version
of SciPy (detected version 1.26.1
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"

Accuracy of the SVM model: 0.28
```

gini and entropy

The Gini impurity is a measure of how often a randomly chosen element from a set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the set.

In [3]:

```
import pandas as pd

y = data['camper']
def gini_impurity(labels):
 total_samples = len(labels)
 label_counts = labels.value_counts()
 impurity = 1
 for count in label_counts:
    label_probability = count / total_samples
    impurity -= label_probability ** 2
 return impurity
gini = gini_impurity(y)
print(f"Gini Impurity for '{X}': {gini}")

```

```
Gini Impurity for '     nofish  livebait  camper
0         1         0       0
1         0         1       1
2         0         1       0
3         0         1       1
4         0         1       0
..      ...       ...     ...
245       1         1       1
246       0         1       1
247       0         1       1
248       1         1       1
249       1         1       1

[250 rows x 3 columns]': 0.48451200000000005
```

In [4]:

```
import pandas as pd
import math

y = data['camper']
def entropy(labels):
 total_samples = len(y)
 label_counts = y.value_counts()
 entropy_val = 0
 for count in label_counts:
        label_probability = count / total_samples
        entropy_val -= label_probability * math.log(label_probability,
 return entropy_val
entropy_result = entropy(y)
print(f"Entropy for '{y}': {entropy_result}")

```

```
Entropy for '0      0
1      1
2      0
3      1
4      0
      ..
245    1
246    1
247    1
248    1
249    1
Name: camper, Length: 250, dtype: int64': 0.9775387286988189
```

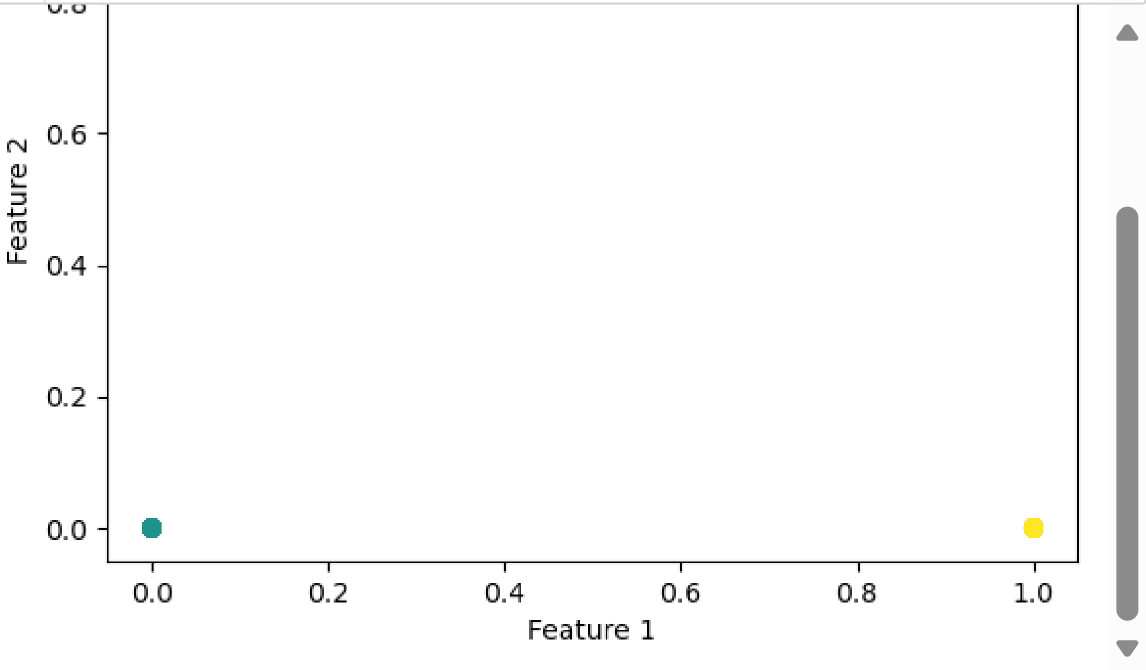
Random Forest Classifier:

In [10]:
```
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
data = load_iris()
X = data.data
y = data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)
predictions = clf.predict(X_test)
accuracy = accuracy_score(y_test, predictions)
print(f"Accuracy of Random Forest Classifier: {accuracy}")
```

```
Accuracy of Random Forest Classifier: 1.0
```

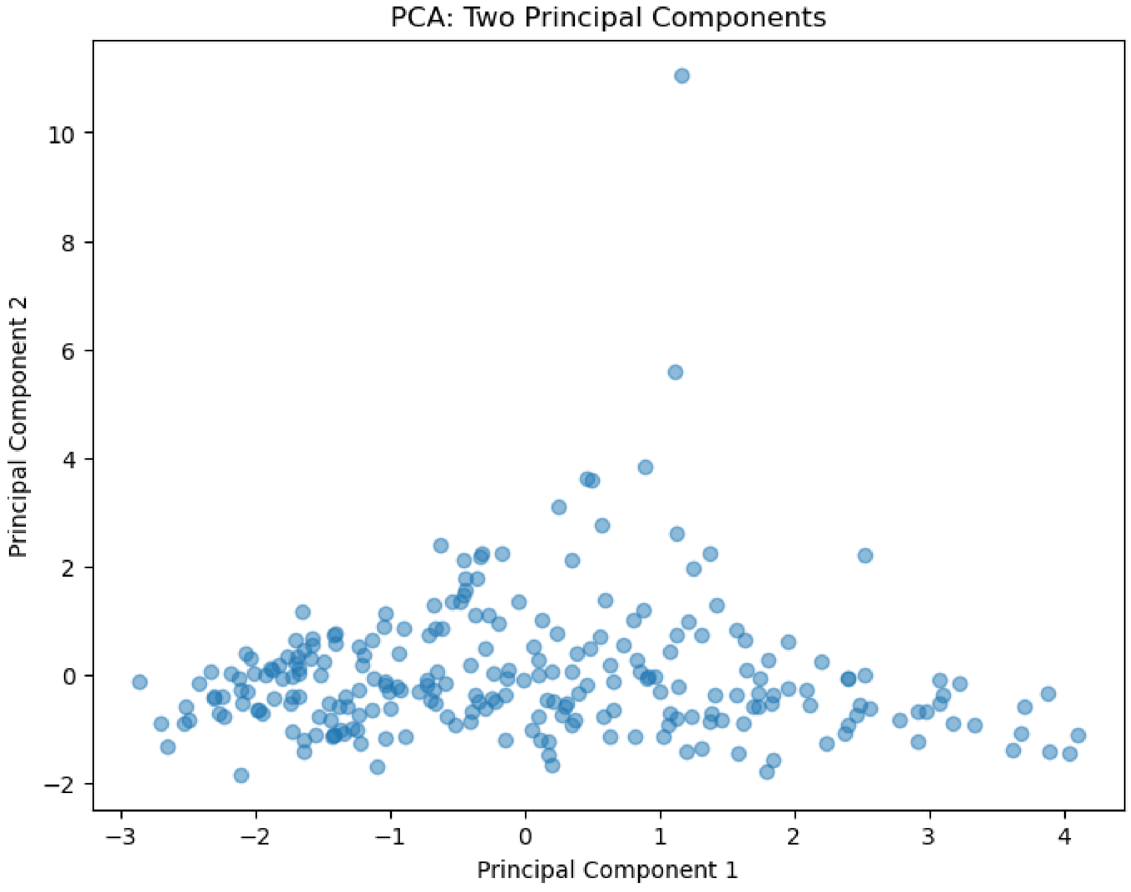
K-means Clustering

In [9]:
```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt


features = data[['nofish', 'camper']]
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)
num_clusters = 3 # You can change this value as needed
kmeans = KMeans(n_clusters=num_clusters)
kmeans.fit(scaled_features)
data['cluster'] = kmeans.labels_
plt.scatter(data['nofish'], data['camper'], c=data['cluster'], cmap='vi
plt.title('K-means Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()

```



PCA

In [19]:
```python
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

features = data.drop(columns=['nofish', 'livebait'])
# Standardize the features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)
# Apply PCA
pca = PCA(n_components=2) # Choose the number of components you want to
principal_components = pca.fit_transform(scaled_features)
# Create a DataFrame for the principal components
principal_df = pd.DataFrame(data=principal_components, columns=['PC1',
# Visualize the PCA results
plt.figure(figsize=(8, 6))
plt.scatter(principal_df['PC1'], principal_df['PC2'], alpha=0.5)
plt.title('PCA: Two Principal Components')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()

```



MLOps

In [53]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import joblib

# Load your dataset
data = pd.read_csv(r"C:\Users\Anusha V\Downloads\fish.csv")

# Assuming you want to drop the 'nofish' column
X = data.drop(columns=['nofish'])
y = data['livebait']

# 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 Random Forest model
model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
model.fit(X_train, y_train)

# Define the filename to save the model
model_filename = "your_model_filename.joblib"

# Save the trained model
joblib.dump(model, model_filename)

# Load the saved model
loaded_model = joblib.load(model_filename)

# Make predictions on the test set
predictions = loaded_model.predict(X_test)

# Print predictions
print("Predictions:", predictions)

```

```
Predictions: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 0 0 1
 1 0 1 1 1 1 1 1 1 1 1 1 1]
```

machine learning pipline

In [26]:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score
data = load_iris()
X = data.data
y = data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2
steps = [
 ('scaler', StandardScaler()), # Step 1: Standard Scaling
 ('pca', PCA(n_components=2)), # Step 2: PCA for dimensionality reducti
 ('classifier', SVC(kernel='rbf', C=1.0)) # Step 3: Support Vector Clas
]
pipeline = Pipeline(steps)
pipeline.fit(X_train, y_train)
predictions = pipeline.predict(X_test)
accuracy = accuracy_score(y_test, predictions)
print(f"Accuracy of the Pipeline: {accuracy}")

```

```
Accuracy of the Pipeline: 0.9333333333333333
```

Basic Ensemble Techniques

Max Voting:

Max Voting is one of the simplest ensemble techniques where you combine the predictio

In [35]:

```
from sklearn.datasets import load_iris  # Import the dataset you need

# Load the dataset (for example, the Iris dataset)
data = load_iris()
X = data.data  # Features
y = data.target  # Labels

```

In [36]:

```
from sklearn.ensemble import VotingClassifier
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC

# Create individual models
model1 = LogisticRegression()
model2 = DecisionTreeClassifier()
model3 = SVC()

# Create a Voting Classifier
ensemble_model = VotingClassifier(estimators=[('lr', model1), ('dt', mo

# Load your dataset and split it into train and test sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3

# Fit the ensemble model on the training data
ensemble_model.fit(X_train, y_train)

# Make predictions using the ensemble model
y_pred = ensemble_model.predict(X_test)
y_pred
```

Out[36]:

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

Averaging:

Averaging is an ensemble technique where you average the predictions of multiple models to get the final prediction.

In [37]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC

# Create individual models
model1 = LogisticRegression()
model2 = DecisionTreeClassifier()
model3 = SVC()

# Load your dataset and split it into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3

# Fit each model on the training data
model1.fit(X_train, y_train)
model2.fit(X_train, y_train)
model3.fit(X_train, y_train)

# Make predictions using each model
pred1 = model1.predict(X_test)
pred2 = model2.predict(X_test)
pred3 = model3.predict(X_test)

# Average the predictions
final_pred = (pred1 + pred2 + pred3) / 3
final_pred
```

Out[37]:
```
array([1.        , 1.        , 0.        , 0.        , 0.        ,
       1.        , 2.        , 2.        , 0.        , 0.        ,
       0.        , 0.        , 1.        , 2.        , 1.        ,
       0.        , 1.        , 1.33333333, 1.        , 1.        ,
       1.        , 0.        , 0.        , 1.33333333, 1.66666667,
       0.        , 1.        , 1.        , 0.        , 0.        ,
       0.        , 1.        , 1.        , 2.        , 2.        ,
       2.        , 0.        , 0.        , 1.        , 0.        ,
       0.        , 1.66666667, 2.        , 2.        , 1.        ])
```

Weighted Average:

Weighted Averaging is similar to Averaging, but you assign different weights to the models

In [38]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC

# Create individual models
model1 = LogisticRegression()
model2 = DecisionTreeClassifier()
model3 = SVC()

# Load your dataset and split it into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3

# Fit each model on the training data
model1.fit(X_train, y_train)
model2.fit(X_train, y_train)
model3.fit(X_train, y_train)

# Make predictions using each model
pred1 = model1.predict(X_test)
pred2 = model2.predict(X_test)
pred3 = model3.predict(X_test)

# Assign weights to each model
weight1 = 0.3
weight2 = 0.4
weight3 = 0.3

# Calculate the weighted average
final_pred = (weight1 * pred1 + weight2 * pred2 + weight3 * pred3)
final_pred
```

Out[38]:
```
array([2., 2., 0., 0., 1., 2., 2., 1., 2., 1., 2., 2., 2., 0., 0., 0., 2.,
       2., 2., 2., 2., 2., 2., 1., 0., 0., 1., 1., 1., 2., 0., 1., 0., 2.,
       0., 1., 1., 0., 1., 1., 2., 0., 1., 1., 0.])
```

Advanced Ensemble Techniques

Stacking:

Stacking is an ensemble technique that combines the predictions of multiple models using another model, called a meta-learner. Here's a simple example of stacking using scikit-learn:

In [39]:

```
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Load a sample dataset
data = load_iris()
X, y = data.data, 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=0.3

# Base models
base_models = [
    ('rf', RandomForestClassifier(n_estimators=100, random_state=42)),
    ('svc', SVC(kernel='linear', C=1, probability=True))
]

# Meta-learner
meta_learner = LogisticRegression()

# Create the stacking classifier
stacking_model = StackingClassifier(estimators=base_models, final_estim

# Fit the stacking model on the training data
stacking_model.fit(X_train, y_train)

# Make predictions using the stacked model
y_pred = stacking_model.predict(X_test)

# Evaluate the performance of the stacking model
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

```

```
Accuracy: 1.0
```

Blending:

Blending is similar to stacking but uses a separate validation set to create meta-features for the final prediction. Here's a simple example of blending:

In [40]:

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

# Load a sample dataset
data = load_iris()
X, y = data.data, 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=0.3

# Train base models
rf = RandomForestClassifier(n_estimators=100, random_state=42)
svc = SVC(kernel='linear', C=1, probability=True)

rf.fit(X_train, y_train)
svc.fit(X_train, y_train)

# Create meta-features using validation set
rf_preds = rf.predict(X_test)
svc_preds = svc.predict(X_test)

# Combine predictions
blend_preds = (rf_preds + svc_preds) / 2

# Evaluate the performance of the blending model
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, blend_preds)
print("Accuracy:", accuracy)

```

```
Accuracy: 1.0
```

Bagging:

Bagging is an ensemble technique that uses bootstrap resampling to train multiple base models. Here's a simple example using the RandomForestClassifier in scikit-learn:

In [41]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Load a sample dataset
data = load_iris()
X, y = data.data, 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=0.3

# Create a bagging classifier
bagging_model = RandomForestClassifier(n_estimators=100, random_state=4

# Fit the bagging model on the training data
bagging_model.fit(X_train, y_train)

# Make predictions using the bagging model
y_pred = bagging_model.predict(X_test)

# Evaluate the performance of the bagging model
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

```

```
Accuracy: 1.0
```

Random forest regression¶

In [42]:

```
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]]))

```

```
[32.55050435]
```

LogisticRegression

In [59]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

X = data.drop('nofish', axis=1)
y = data['livebait']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")

```

```
Accuracy: 1.0
```

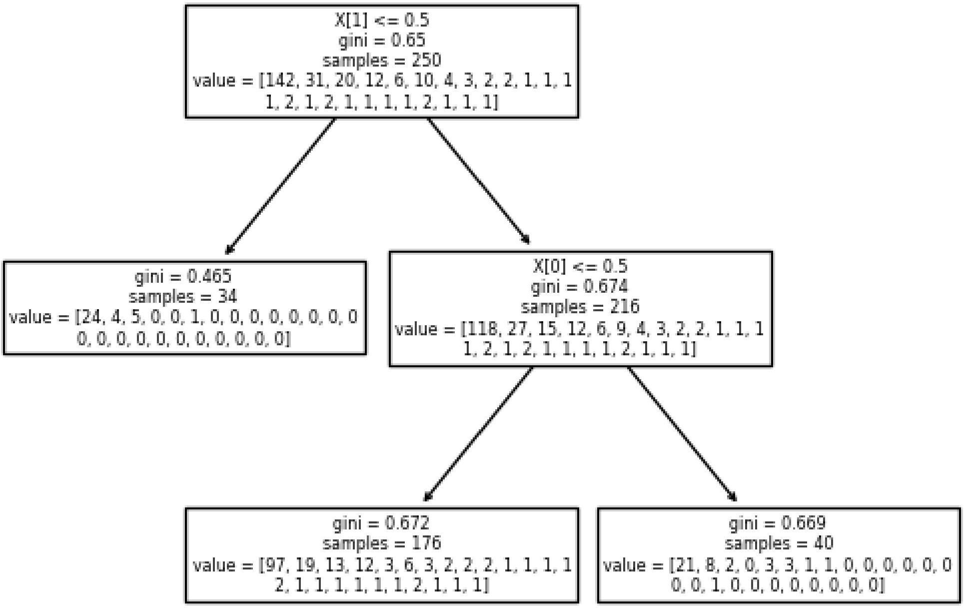
DecisionTreeClassifier

In [60]:

```python
import pandas as pd

X = data.iloc[:, :-1]
from sklearn.tree import DecisionTreeClassifier
clf1 = DecisionTreeClassifier()
clf1.fit(data.iloc[:,0:2],data.iloc[:,-1])
from sklearn.tree import plot_tree
plot_tree(clf1)
```

Out[60]:

```
[Text(0.4, 0.8333333333333334, 'X[1] <= 0.5\ngini = 0.65\nsamples = 250\nv
alue = [142, 31, 20, 12, 6, 10, 4, 3, 2, 2, 1, 1, 1\n1, 2, 1, 2, 1, 1, 1,
1, 2, 1, 1, 1]'),
 Text(0.2, 0.5, 'gini = 0.465\nsamples = 34\nvalue = [24, 4, 5, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]'),
 Text(0.6, 0.5, 'X[0] <= 0.5\ngini = 0.674\nsamples = 216\nvalue = [118, 2
7, 15, 12, 6, 9, 4, 3, 2, 2, 1, 1, 1\n1, 2, 1, 2, 1, 1, 1, 1, 2, 1, 1,
1]'),
 Text(0.4, 0.16666666666666666, 'gini = 0.672\nsamples = 176\nvalue = [97,
19, 13, 12, 3, 6, 3, 2, 2, 2, 1, 1, 1, 1\n2, 1, 1, 1, 1, 1, 1, 2, 1, 1,
1]'),
 Text(0.8, 0.16666666666666666, 'gini = 0.669\nsamples = 40\nvalue = [21,
8, 2, 0, 3, 3, 1, 1, 0, 0, 0, 0, 0, 0\n0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]')]
```



CRICKET MATCH RESULT-PAST DATA

In [1]:
```
import pandas as pd
data = pd.read_csv(r"C:\Users\Anusha V\Desktop\cricket.csv")
data
```

Out[1]:

	id	season	city	date	team1	team2	toss_winner	toss_decision	re
0	1.0	2017.0	Hyderabad	05-04-2017	Sunrisers Hyderabad	Royal Challengers Bangalore	Royal Challengers Bangalore	field	nor
1	2.0	2017.0	Pune	06-04-2017	Mumbai Indians	Rising Pune Supergiant	Rising Pune Supergiant	field	nor
2	3.0	2017.0	Rajkot	07-04-2017	Gujarat Lions	Kolkata Knight Riders	Kolkata Knight Riders	field	nor
3	4.0	2017.0	Indore	08-04-2017	Rising Pune Supergiant	Kings XI Punjab	Kings XI Punjab	field	nor
4	5.0	2017.0	Bangalore	08-04-2017	Royal Challengers Bangalore	Delhi Daredevils	Royal Challengers Bangalore	bat	nor
...	...	...	...	...	...	...	...	...	
631	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
632	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
633	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
634	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
635	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N

636 rows × 18 columns

In [20]:
```
run_a_wins = len(data[data['win_by_runs'] == 'run A'])
hs_b_wins = len(data[data['win_by_wickets'] == 'highest_scores B'])
print("Number of matches won by Team A:", run_a_wins)
print("Number of matches won by Team B:", hs_b_wins)

```

```
Number of matches won by Team A: 0
Number of matches won by Team B: 0
```

In [45]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

```

In [46]:

```
##Data Exploration:

# Display the first few rows of the dataset
print(data.head())

# Summary statistics
print(data.describe())

# Check for missing values
print(data.isnull().sum())

# Visualize data as needed (e.g., using matplotlib or seaborn)

```

```
Empty DataFrame
Columns: [id, season, city, date, team1, team2, toss_winner, toss_decisio
n, result, dl_applied, winner, win_by_runs, win_by_wickets, player_of_matc
h, venue, umpire1, umpire2, umpire3]
Index: []
        id  season  dl_applied  win_by_runs  win_by_wickets  umpire3
count  0.0     0.0         0.0          0.0             0.0      0.0
mean   NaN     NaN         NaN          NaN             NaN      NaN
std    NaN     NaN         NaN          NaN             NaN      NaN
min    NaN     NaN         NaN          NaN             NaN      NaN
25%    NaN     NaN         NaN          NaN             NaN      NaN
50%    NaN     NaN         NaN          NaN             NaN      NaN
75%    NaN     NaN         NaN          NaN             NaN      NaN
max    NaN     NaN         NaN          NaN             NaN      NaN
id                 0.0
season             0.0
city               0.0
date               0.0
team1              0.0
team2              0.0
toss_winner        0.0
toss_decision      0.0
result             0.0
dl_applied         0.0
winner             0.0
win_by_runs        0.0
win_by_wickets     0.0
player_of_match    0.0
venue              0.0
umpire1            0.0
umpire2            0.0
umpire3            0.0
dtype: float64
```

In [53]:
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Data Exploration
# (Insert code to explore the dataset)

# Handle missing values (if any)
data.dropna(inplace=True)

# Encode categorical variables (if needed) using techniques like one-ho
# (Insert code to encode categorical variables)

# Data Preprocessing

# Extract the target variable (predicting the winning team)
y = data['winner']

# Extract relevant features (predictor variables)
# Here, you can select the features that you believe are relevant
X = data[['team1', 'team2', 'toss_winner', 'toss_decision', 'venue']]

# Check if there are enough samples for splitting
if len(data) > 0:
    # Data Splitting
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size

    # Check the shape of the split datasets
    print("X_train shape:", X_train.shape)
    print("X_test shape:", X_test.shape)
    print("y_train shape:", y_train.shape)
    print("y_test shape:", y_test.shape)

    # Standardize/normalize the features
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Continue with the rest of your analysis or modeling.

```

In [55]:

```
X_train
```

Out[55]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	
477	15.02340	0.0	18.10	0.0	0.6140	5.304	97.3	2.1007	24.0	666.0	20.2	349.4
15	0.62739	0.0	8.14	0.0	0.5380	5.834	56.5	4.4986	4.0	307.0	21.0	395.6
332	0.03466	35.0	6.06	0.0	0.4379	6.031	23.3	6.6407	1.0	304.0	16.9	362.2
423	7.05042	0.0	18.10	0.0	0.6140	6.103	85.1	2.0218	24.0	666.0	20.2	2.5
19	0.72580	0.0	8.14	0.0	0.5380	5.727	69.5	3.7965	4.0	307.0	21.0	390.9
...	...	...	...	...	...	...	...	...	...	...	...	
106	0.17120	0.0	8.56	0.0	0.5200	5.836	91.9	2.2110	5.0	384.0	20.9	395.6
270	0.29916	20.0	6.96	0.0	0.4640	5.856	42.1	4.4290	3.0	223.0	18.6	388.6
348	0.01501	80.0	2.01	0.0	0.4350	6.635	29.7	8.3440	4.0	280.0	17.0	390.9
435	11.16040	0.0	18.10	0.0	0.7400	6.629	94.6	2.1247	24.0	666.0	20.2	109.8
102	0.22876	0.0	8.56	0.0	0.5200	6.405	85.4	2.7147	5.0	384.0	20.9	70.8

404 rows × 13 columns

In [56]:

```
X_test
```

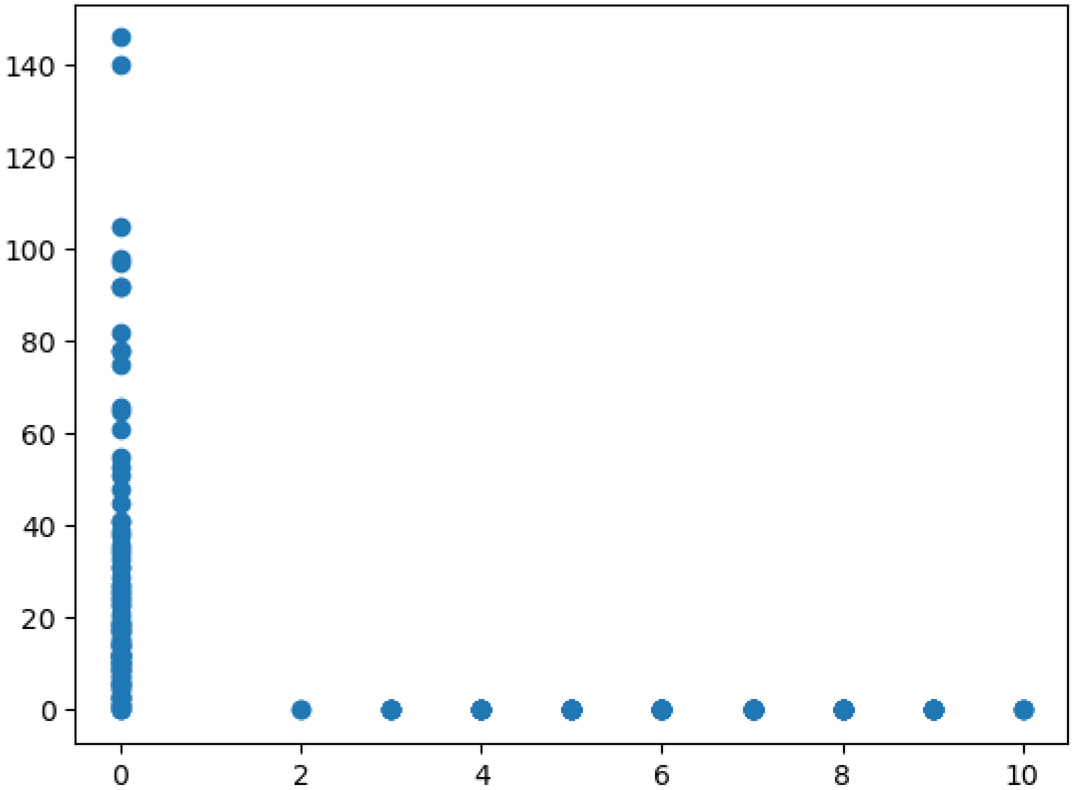
Out[56]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	I
173	0.09178	0.0	4.05	0.0	0.510	6.416	84.1	2.6463	5.0	296.0	16.6	395.5
274	0.05644	40.0	6.41	1.0	0.447	6.758	32.9	4.0776	4.0	254.0	17.6	396.9
491	0.10574	0.0	27.74	0.0	0.609	5.983	98.8	1.8681	4.0	711.0	20.1	390.1
72	0.09164	0.0	10.81	0.0	0.413	6.065	7.8	5.2873	4.0	305.0	19.2	390.9
452	5.09017	0.0	18.10	0.0	0.713	6.297	91.8	2.3682	24.0	666.0	20.2	385.0
...	...	...	...	...	...	...	...	...	...	...	...	.
412	18.81100	0.0	18.10	0.0	0.597	4.628	100.0	1.5539	24.0	666.0	20.2	28.7
436	14.42080	0.0	18.10	0.0	0.740	6.461	93.3	2.0026	24.0	666.0	20.2	27.4
411	14.05070	0.0	18.10	0.0	0.597	6.657	100.0	1.5275	24.0	666.0	20.2	35.0
86	0.05188	0.0	4.49	0.0	0.449	6.015	45.1	4.4272	3.0	247.0	18.5	395.9
75	0.09512	0.0	12.83	0.0	0.437	6.286	45.0	4.5026	5.0	398.0	18.7	383.2

102 rows × 13 columns

In [4]:
```
import matplotlib.pyplot as plt
plt.scatter(data['win_by_wickets'],data['win_by_runs'])
```

Out[4]: <matplotlib.collections.PathCollection at 0x13fd2f1b490>



In [ ]:
```

```

In [ ]:
```

```

CROP YIELD - PAST DATA

In [10]:

```
import pandas as pd

# Replace the file path with the correct one
file_path = r"C:\Users\Anusha V\Desktop\CropYield.xlsx"

try:
    data1 = pd.read_excel(file_path)
except Exception as e:
    print(f"An error occurred: {str(e)}")

# Now you can work with the 'data1' DataFrame
print(data1)

```

```
                          State        District       Crop  Crop_Year  \
0    Andaman and Nicobar Island        NICOBARS   Arecanut       2007   
1    Andaman and Nicobar Island        NICOBARS   Arecanut       2007   
2    Andaman and Nicobar Island        NICOBARS   Arecanut       2008   
3    Andaman and Nicobar Island        NICOBARS   Arecanut       2008   
4    Andaman and Nicobar Island        NICOBARS   Arecanut       2009   
..                          ...             ...        ...        ...   
197  Andaman and Nicobar Island  SOUTH ANDAMANS  Cashewnut       2010   
198  Andaman and Nicobar Island  SOUTH ANDAMANS  Cashewnut       2011   
199  Andaman and Nicobar Island  SOUTH ANDAMANS  Cashewnut       2012   
200  Andaman and Nicobar Island  SOUTH ANDAMANS  Cashewnut       2013   
201  Andaman and Nicobar Island  SOUTH ANDAMANS  Cashewnut       2014   

          Season    Area  Production  Yield  
0    Kharif       2439.6      3415.0   1.40  
1    Rabi         1626.4      2277.0   1.40  
2    Autumn       4147.0      3060.0   0.74  
3    Summer       4147.0      2660.0   0.64  
4    Autumn       4153.0      3120.0   0.75  
..           ...     ...         ...    ...  
197  Rabi           15.0        11.0   0.70  
198  Rabi           20.0        15.0   0.77  
199  Rabi           35.4        21.0   0.60  
200  Rabi           29.5        22.0   0.74  
201  Rabi           14.9        23.0   1.53  

[202 rows x 8 columns]
```

In [11]:

```
import pandas as pd

# Display the first few rows of the dataset
print(data1.head())

# Check for missing values
print(data1.isnull().sum())

# Summary statistics
print(data1.describe())

```

```
                       State District       Crop  Crop_Year       Season  \
0  Andaman and Nicobar Island  NICOBARS  Arecanut       2007  Kharif
1  Andaman and Nicobar Island  NICOBARS  Arecanut       2007  Rabi
2  Andaman and Nicobar Island  NICOBARS  Arecanut       2008  Autumn
3  Andaman and Nicobar Island  NICOBARS  Arecanut       2008  Summer
4  Andaman and Nicobar Island  NICOBARS  Arecanut       2009  Autumn

     Area  Production  Yield
0  2439.6      3415.0   1.40
1  1626.4      2277.0   1.40
2  4147.0      3060.0   0.74
3  4147.0      2660.0   0.64
4  4153.0      3120.0   0.75
State         0
District      0
Crop          0
Crop_Year     0
Season        0
Area          0
Production    1
Yield         0
dtype: int64
         Crop_Year         Area    Production       Yield
count   202.000000   202.000000    201.000000  202.000000
mean   2010.747525   746.185485   2220.756219    2.943020
std       5.393991   880.050543   3273.402037    4.637631
min    2000.000000     0.500000      0.000000    0.000000
25%    2006.000000    81.000000     21.000000    0.220000
50%    2011.000000   450.000000    208.000000    0.700000
75%    2015.000000  1034.300000   3602.000000    3.350000
max    2019.000000  4153.000000  17374.000000   26.450000
```

In [12]:

```
from sklearn.model_selection import train_test_split
import pandas as pd
data1 = pd.read_csv(r"C:\Users\Anusha V\Downloads\archive (8)\APY.csv")
X = data1.drop(columns=["Crop"])
y = data1["Crop"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

```

In [13]:
```
1 X_train
```

Out[13]:

	State	District	Crop_Year	Season	Area	Production	Yield
114808	Jharkhand	RAMGARH	2013	Winter	1361.0	7544.0	5.54
109257	Jammu and Kashmir	REASI	2015	Kharif	13.0	9.0	0.70
101966	Haryana	GURGAON	2009	Whole Year	39.0	800.0	20.51
313553	Uttar Pradesh	JALAUN	2010	Kharif	9.0	3.0	0.33
270181	Tamil Nadu	MADURAI	2006	Whole Year	6006.0	656204.0	109.26
...	...	...	...	...	...	...	...
119879	Karnataka	GADAG	2018	Kharif	13.0	7.0	0.54
259178	Tamil Nadu	TIRUCHIRAPPALLI	2002	Whole Year	5991.0	84200000.0	14054.41
131932	Karnataka	CHIKKABALLAPURA	2017	Kharif	63.0	14.0	0.22
146867	Kerala	IDUKKI	2014	Whole Year	4.0	NaN	0.00
121958	Karnataka	BELAGAVI	2017	Rabi	279.0	113.0	0.41

276268 rows × 7 columns

In [14]:
```
1 X_test
```

Out[14]:

	State	District	Crop_Year	Season	Area	Production	Yield
212198	Nagaland	PHEK	2017	Kharif	180.0	3650.0	20.28
2202	Andhra Pradesh	RANGAREDDI	2005	Whole Year	40.0	836.0	20.90
36666	Assam	JORHAT	2011	Whole Year	74.0	244.0	3.30
110896	Jammu and Kashmir	UDHAMPUR	2006	Rabi	2833.0	2079.0	0.73
212200	Nagaland	PHEK	2019	Kharif	200.0	4052.0	20.26
...	...	...	...	...	...	...	...
162204	Madhya Pradesh	MANDSAUR	2010	Kharif	37232.0	50743.0	1.36
35448	Assam	GOLAGHAT	2002	Kharif	40.0	18.0	0.45
185511	Maharashtra	BULDHANA	1997	Kharif	20000.0	16400.0	0.82
43783	Bihar	NALANDA	2017	Rabi	5369.0	6513.0	1.21
285273	Uttar Pradesh	ETAWAH	2019	Rabi	1566.0	5382.0	3.44

69068 rows × 7 columns

In [15]:
```
y_train
```

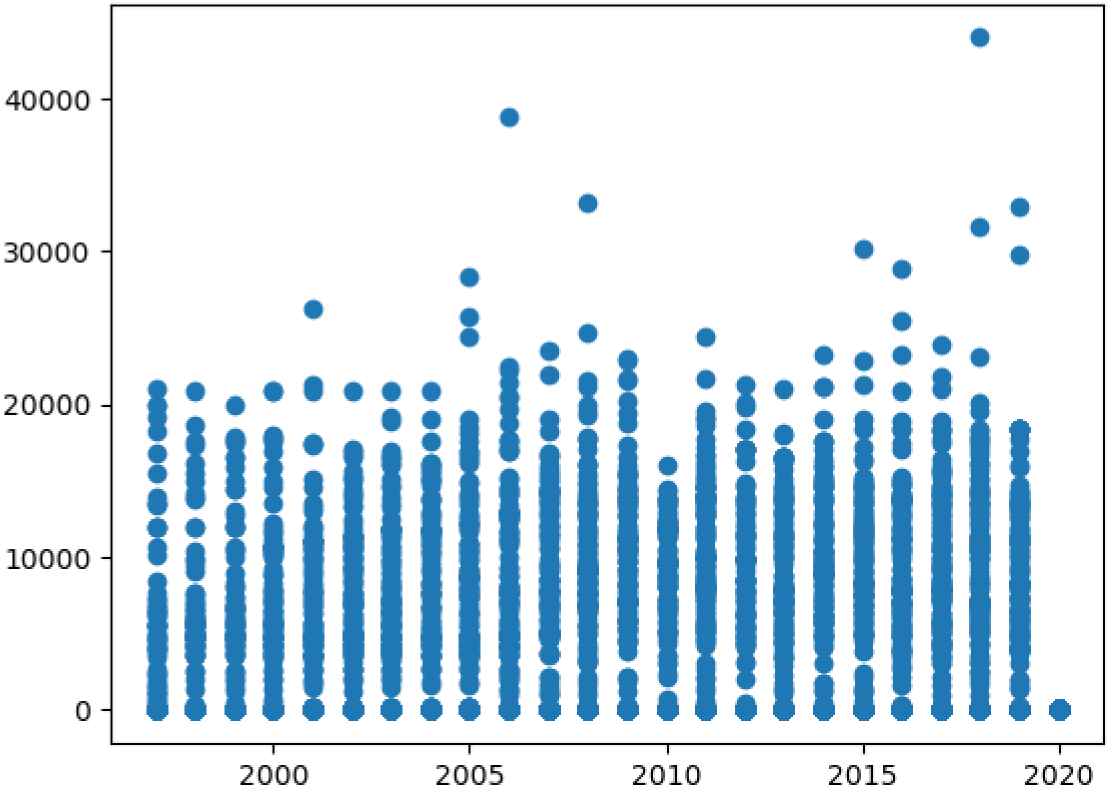
Out[15]:
```
114808                 Potato
109257     Moong(Green Gram)
101966          Sweet potato
313553              Sannhamp
270181             Sugarcane
                    ...
119879           Castor seed
259178              Coconut
131932            Niger seed
146867                 Onion
121958         Cowpea(Lobia)
Name: Crop, Length: 276268, dtype: object
```

In [16]:
```
y_test
```

Out[16]:
```
212198                Tapioca
2202                   Banana
36666           Sweet potato
110896     Rapeseed &Mustard
212200                Tapioca
                    ...
162204                  Maize
35448          Small millets
185511                  Maize
43783                    Gram
285273                 Barley
Name: Crop, Length: 69068, dtype: object
```

In [17]:
```
import matplotlib.pyplot as plt
plt.scatter(data1['Crop_Year'],data1['Yield'])
```

Out[17]:
```
<matplotlib.collections.PathCollection at 0x2ba2e4a0f70>
```



k mean

In [43]:
```
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

```

In [44]:
```
k = 3 # Set the value of k
knn_classifier = KNeighborsClassifier(n_neighbors=k)
knn_classifier.fit(X_train, y_train)
```

Out[44]:
```
KNeighborsClassifier(n_neighbors=3)
```

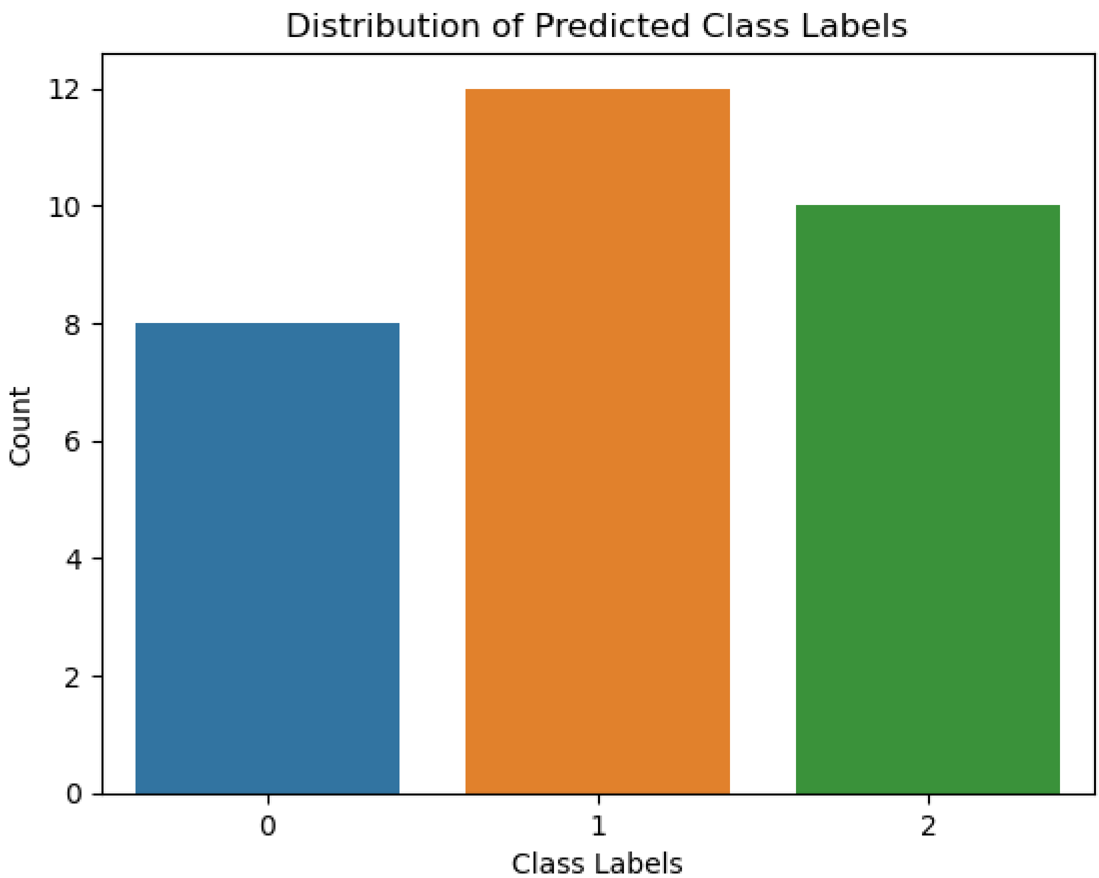
In [48]:
```
y_pred = knn_classifier.predict(X_test)
y_pred

```

```
C:\Users\Anusha V\anaconda3\lib\site-packages\sklearn\neighbors\_classific
ation.py:228: FutureWarning: Unlike other reduction functions (e.g. `skew
`, `kurtosis`), the default behavior of `mode` typically preserves the axi
s it acts along. In SciPy 1.11.0, this behavior will change: the default v
alue of `keepdims` will become False, the `axis` over which the statistic
is taken will be eliminated, and the value None will no longer be accepte
d. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

Out[48]:
```
array([0, 1, 2, 2, 1, 2, 1, 1, 1, 0, 1, 0, 0, 2, 1, 2, 2, 2, 1, 1, 2, 2,
       1, 0, 1, 0, 0, 2, 0, 1])
```

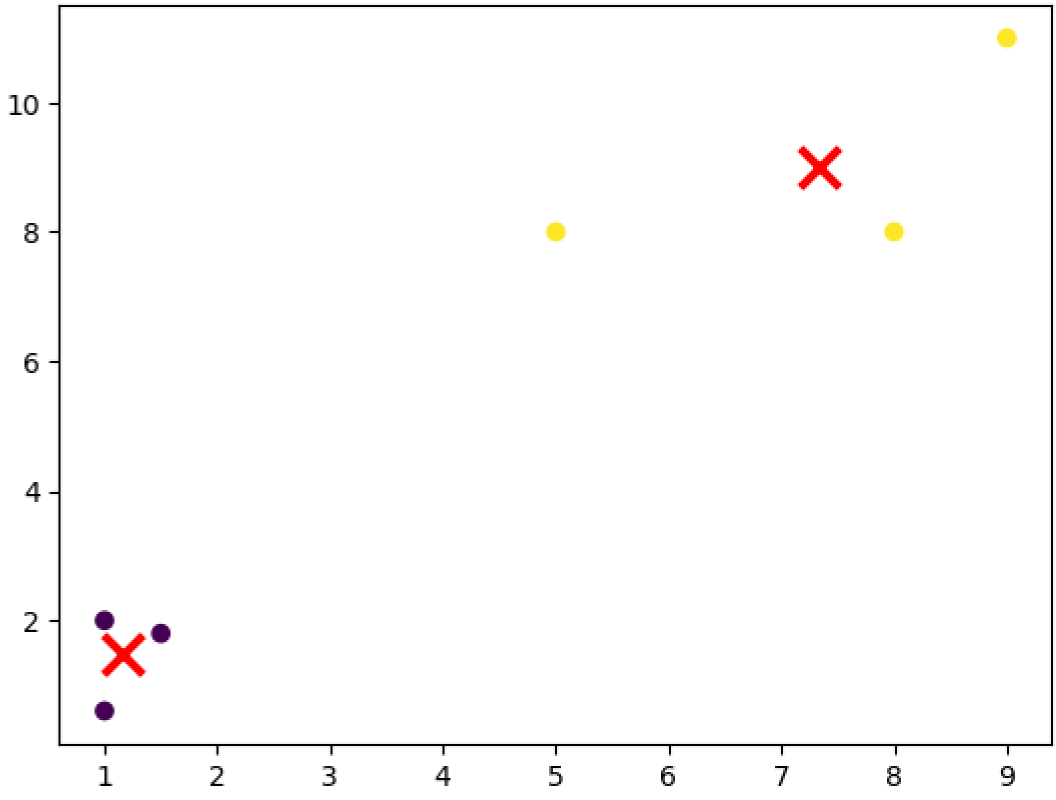
In [49]:
```
import seaborn as sns
import matplotlib.pyplot as plt
sns.countplot(y_pred)
plt.title("Distribution of Predicted Class Labels")
plt.xlabel("Class Labels")
plt.ylabel("Count")
plt.show()
```

```
C:\Users\Anusha V\anaconda3\lib\site-packages\seaborn\_decorators.py:36: F
utureWarning: Pass the following variable as a keyword arg: x. From versio
n 0.12, the only valid positional argument will be `data`, and passing oth
er arguments without an explicit keyword will result in an error or misint
erpretation.
  warnings.warn(
```



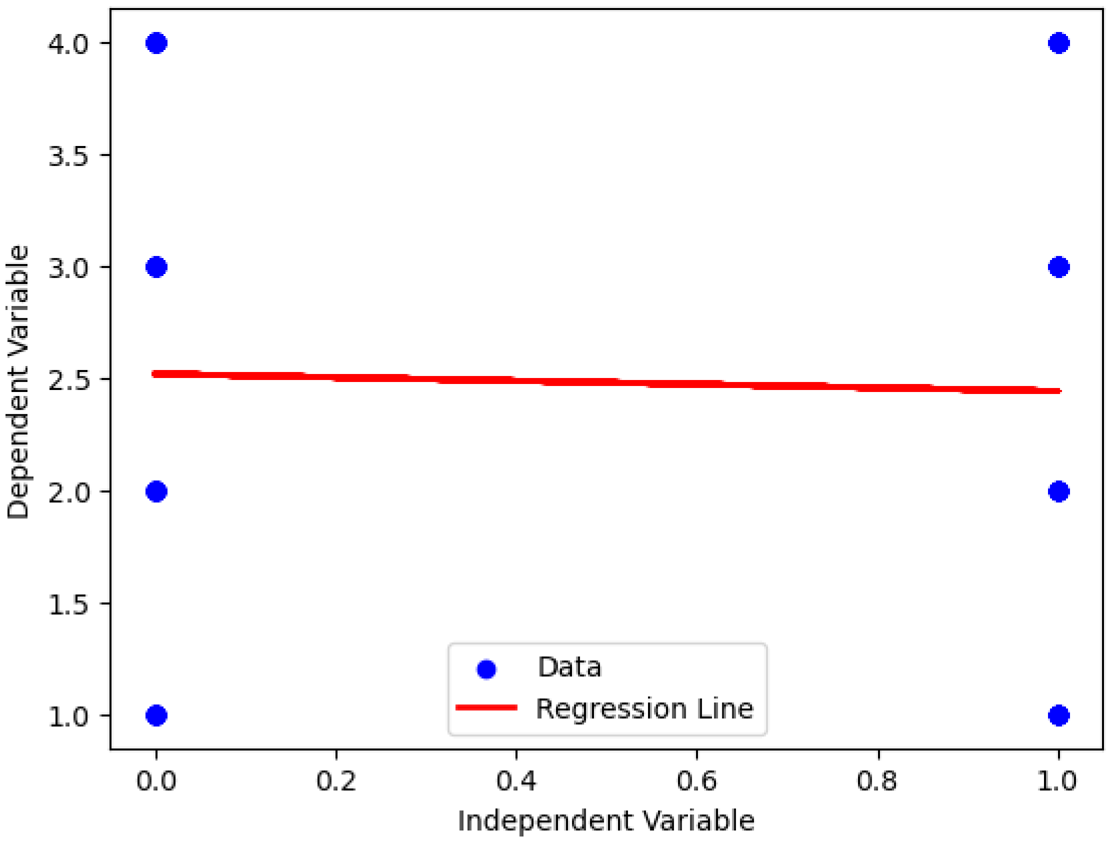
In [50]:

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Generate some example data (replace this with your own data)
import numpy as np
X = np.array([[1, 2], [5, 8], [1.5, 1.8], [8, 8], [1, 0.6], [9, 11]])

# Create a K-Means model with the desired number of clusters (e.g., 2)
kmeans = KMeans(n_clusters=2)

# Fit the model to your data
kmeans.fit(X)

# Get the cluster labels for each data point
labels = kmeans.labels_

# Get the coordinates of the cluster centers
centers = kmeans.cluster_centers_

# Plot the data points and cluster centers
plt.scatter(X[:, 0], X[:, 1], c=labels)
plt.scatter(centers[:, 0], centers[:, 1], marker='x', s=200, linewidths
plt.show()
```



Simple linear regression

In [7]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
data1= pd.read_csv(r"C:\Users\Anusha V\Downloads\fish.csv")

X = data1[['nofish']]
y = data1['persons']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
plt.scatter(X, y, color='blue', label='Data')
plt.plot(X_test, y_pred, color='red', linewidth=2, label='Regression Li
plt.xlabel('Independent Variable')
plt.ylabel('Dependent Variable')
plt.legend()
plt.show()
slope = model.coef_[0]
intercept = model.intercept_
print(f"Slope (m): {slope}")
print(f"Intercept (b): {intercept}")
```




```
Slope (m): -0.07705253035220574
Intercept (b): 2.517730496453901
```

Breast Cancer Wisconsin (diagnostic)¶

In [31]:
```
import numpy as np
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
```

In [32]:
```
data = load_breast_cancer()
X = data.data
y = data.target
```

In [33]:
```
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=4
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2
```

In [34]:
```
rf_classifier. fit(X_train, y_train)

```

Out[34]:
```
RandomForestClassifier(random_state=42)
```

In [35]:
```
y_pred = rf_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
accuracy
```

Out[35]:
```
0.9649122807017544
```

In [36]:
```
report = classification_report(y_test, y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.98      0.93      0.95        43
           1       0.96      0.99      0.97        71

    accuracy                           0.96       114
   macro avg       0.97      0.96      0.96       114
weighted avg       0.97      0.96      0.96       114
```

Build decision tree-based model in python

like Breast Cancer Wisconsin (diagnostic) dataset from sci-kit learn Or any classification dataset from UCI , Kaggle

In [40]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
breast_cancer = load_breast_cancer()
X = breast_cancer.data # Features
y = breast_cancer.target # Target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print("Classification Report:")
print(classification_report(y_test, y_pred))


```

```
Accuracy: 0.95
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.93      0.93        43
           1       0.96      0.96      0.96        71

    accuracy                           0.95       114
   macro avg       0.94      0.94      0.94       114
weighted avg       0.95      0.95      0.95       114
```

Build Random Forest-based model in python for

Breast Cancer Wisconsin (diagnostic) dataset from sci-kit learn Or dataset from UCI , Kaggle

In [41]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
breast_cancer = load_breast_cancer()
X = breast_cancer.data # Features
y = breast_cancer.target # Target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=4
rf_classifier.fit(X_train, y_train)
y_pred = rf_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print("Classification Report:")
print(classification_report(y_test, y_pred))

```

```
Accuracy: 0.96
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.93      0.95        43
           1       0.96      0.99      0.97        71

    accuracy                           0.96       114
   macro avg       0.97      0.96      0.96       114
weighted avg       0.97      0.96      0.96       114
```

Iris dataset from sci-kit learnPerform data exploration,

preprocessing and splitting

In [29]:
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
import pandas as pd
iris = load_iris()
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
iris_df['target'] = iris.target
print(iris_df.head())
X = iris.data # Features
y = iris.target # Target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (c
m)  \
0                5.1               3.5                1.4                0.
2
1                4.9               3.0                1.4                0.
2
2                4.7               3.2                1.3                0.
2
3                4.6               3.1                1.5                0.
2
4                5.0               3.6                1.4                0.
2

   target
0       0
1       0
2       0
3       0
4       0
X_train shape: (120, 4)
X_test shape: (30, 4)
y_train shape: (120,)
y_test shape: (30,)
```

Regression Model with Deep Learning:

In [22]:
```
import numpy as np
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras import models, layers

# Load the Boston Housing Prices dataset
boston = load_boston()
data = boston.data
targets = boston.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data, targets, test

# Standardize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build the regression model
model_reg = models.Sequential()
model_reg.add(layers.Dense(64, activation='relu', input_shape=(X_train.
model_reg.add(layers.Dense(1))  # Output layer for regression

# Compile the model
model_reg.compile(optimizer='adam', loss='mean_squared_error', metrics=

# Train the model
model_reg.fit(X_train, y_train, epochs=50, batch_size=16, validation_da

```

```
26/26 [==============================] - 0s 19ms/step - loss: 27.8384 -
mae: 3.8863 - val_loss: 29.0147 - val_mae: 3.6542
Epoch 27/50
26/26 [==============================] - 0s 14ms/step - loss: 27.1955 -
mae: 3.8508 - val_loss: 28.2891 - val_mae: 3.6082
Epoch 28/50
26/26 [==============================] - 0s 13ms/step - loss: 26.5516 -
mae: 3.8142 - val_loss: 27.7366 - val_mae: 3.5744
Epoch 29/50
26/26 [==============================] - 0s 10ms/step - loss: 26.0652 -
mae: 3.7790 - val_loss: 27.1639 - val_mae: 3.5189
Epoch 30/50
26/26 [==============================] - 0s 11ms/step - loss: 25.4592 -
mae: 3.7307 - val_loss: 26.7644 - val_mae: 3.4852
Epoch 31/50
26/26 [==============================] - 0s 11ms/step - loss: 24.9334 -
mae: 3.7008 - val_loss: 26.1122 - val_mae: 3.4524
Epoch 32/50
26/26 [==============================] - 0s 9ms/step - loss: 24.4530 - m
ae: 3.6647 - val_loss: 25.7499 - val_mae: 3.4131
```

Classification Model with Deep Learning:

In [23]:
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from tensorflow.keras import models, layers, utils

# Load the Iris dataset
iris = load_iris()
data = iris.data
targets = iris.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data, targets, test

# Standardize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# One-hot encode the labels
y_train = utils.to_categorical(y_train)
y_test = utils.to_categorical(y_test)

# Build the classification model
model_cls = models.Sequential()
model_cls.add(layers.Dense(64, activation='relu', input_shape=(X_train.
model_cls.add(layers.Dense(3, activation='softmax'))  # Output layer fo

# Compile the model
model_cls.compile(optimizer='adam', loss='categorical_crossentropy', me

# Train the model
model_cls.fit(X_train, y_train, epochs=50, batch_size=16, validation_da

```

```
Epoch 1/50
8/8 [==============================] - 2s 77ms/step - loss: 1.0962 - accur
acy: 0.5750 - val_loss: 1.0760 - val_accuracy: 0.6000
Epoch 2/50
8/8 [==============================] - 0s 18ms/step - loss: 0.9886 - accur
acy: 0.6250 - val_loss: 0.9611 - val_accuracy: 0.6000
Epoch 3/50
8/8 [==============================] - 0s 27ms/step - loss: 0.8924 - accur
acy: 0.6333 - val_loss: 0.8628 - val_accuracy: 0.6000
Epoch 4/50
8/8 [==============================] - 0s 28ms/step - loss: 0.8104 - accur
acy: 0.6583 - val_loss: 0.7776 - val_accuracy: 0.8000
Epoch 5/50
8/8 [==============================] - 0s 19ms/step - loss: 0.7405 - accur
acy: 0.7583 - val_loss: 0.7028 - val_accuracy: 0.8333
Epoch 6/50
8/8 [==============================] - 0s 18ms/step - loss: 0.6797 - accur
acy: 0.8000 - val_loss: 0.6372 - val_accuracy: 0.8667
Epoch 7/50
8/8 [==============================] - 0s 21ms/step - loss: 0.6272 - accur
acy: 0.8167 - val_loss: 0.5803 - val_accuracy: 0.9000
Epoch 8/50
8/8 [==============================] - 0s 21ms/step - loss: 0.5848 - accur
acy: 0.8083 - val_loss: 0.5305 - val_accuracy: 0.9000
Epoch 9/50
8/8 [==============================] - 0s 26ms/step - loss: 0.5440 - accur
acy: 0.8083 - val_loss: 0.4928 - val_accuracy: 0.9000
Epoch 10/50
8/8 [==============================] - 0s 25ms/step - loss: 0.5150 - accur
acy: 0.8167 - val_loss: 0.4574 - val_accuracy: 0.9000
Epoch 11/50
8/8 [==============================] - 0s 25ms/step - loss: 0.4879 - accur
acy: 0.8167 - val_loss: 0.4292 - val_accuracy: 0.9000
Epoch 12/50
8/8 [==============================] - 0s 25ms/step - loss: 0.4653 - accur
acy: 0.8167 - val_loss: 0.4060 - val_accuracy: 0.9000
Epoch 13/50
8/8 [==============================] - 0s 26ms/step - loss: 0.4465 - accur
acy: 0.8167 - val_loss: 0.3853 - val_accuracy: 0.9000
Epoch 14/50
8/8 [==============================] - 0s 12ms/step - loss: 0.4301 - accur
acy: 0.8167 - val_loss: 0.3672 - val_accuracy: 0.9000
Epoch 15/50
8/8 [==============================] - 0s 12ms/step - loss: 0.4154 - accur
acy: 0.8167 - val_loss: 0.3516 - val_accuracy: 0.9000
Epoch 16/50
8/8 [==============================] - 0s 11ms/step - loss: 0.4025 - accur
acy: 0.8250 - val_loss: 0.3384 - val_accuracy: 0.9000
Epoch 17/50
8/8 [==============================] - 0s 11ms/step - loss: 0.3915 - accur
acy: 0.8250 - val_loss: 0.3253 - val_accuracy: 0.9000
Epoch 18/50
8/8 [==============================] - 0s 22ms/step - loss: 0.3807 - accur
acy: 0.8250 - val_loss: 0.3140 - val_accuracy: 0.9000
Epoch 19/50
8/8 [==============================] - 0s 13ms/step - loss: 0.3711 - accur
acy: 0.8333 - val_loss: 0.3046 - val_accuracy: 0.9000
Epoch 20/50
8/8 [==============================] - 0s 16ms/step - loss: 0.3625 - accur
acy: 0.8333 - val_loss: 0.2951 - val_accuracy: 0.9000
Epoch 21/50
```

```
8/8 [==============================] - 0s 31ms/step - loss: 0.3544 - accur
acy: 0.8333 - val_loss: 0.2864 - val_accuracy: 0.9000
Epoch 22/50
8/8 [==============================] - 0s 16ms/step - loss: 0.3466 - accur
acy: 0.8417 - val_loss: 0.2783 - val_accuracy: 0.9000
Epoch 23/50
8/8 [==============================] - 0s 13ms/step - loss: 0.3391 - accur
acy: 0.8417 - val_loss: 0.2716 - val_accuracy: 0.9000
Epoch 24/50
8/8 [==============================] - 0s 15ms/step - loss: 0.3318 - accur
acy: 0.8417 - val_loss: 0.2645 - val_accuracy: 0.9000
Epoch 25/50
8/8 [==============================] - 0s 14ms/step - loss: 0.3252 - accur
acy: 0.8417 - val_loss: 0.2577 - val_accuracy: 0.9000
Epoch 26/50
8/8 [==============================] - 0s 13ms/step - loss: 0.3189 - accur
acy: 0.8583 - val_loss: 0.2509 - val_accuracy: 0.9000
Epoch 27/50
8/8 [==============================] - 0s 13ms/step - loss: 0.3122 - accur
acy: 0.8583 - val_loss: 0.2441 - val_accuracy: 0.9000
Epoch 28/50
8/8 [==============================] - 0s 12ms/step - loss: 0.3060 - accur
acy: 0.8667 - val_loss: 0.2387 - val_accuracy: 0.9000
Epoch 29/50
8/8 [==============================] - 0s 11ms/step - loss: 0.3000 - accur
acy: 0.8667 - val_loss: 0.2339 - val_accuracy: 0.9000
Epoch 30/50
8/8 [==============================] - 0s 19ms/step - loss: 0.2946 - accur
acy: 0.8667 - val_loss: 0.2284 - val_accuracy: 0.9000
Epoch 31/50
8/8 [==============================] - 0s 13ms/step - loss: 0.2890 - accur
acy: 0.8667 - val_loss: 0.2237 - val_accuracy: 0.9000
Epoch 32/50
8/8 [==============================] - 0s 26ms/step - loss: 0.2833 - accur
acy: 0.8667 - val_loss: 0.2187 - val_accuracy: 0.9000
Epoch 33/50
8/8 [==============================] - 0s 17ms/step - loss: 0.2778 - accur
acy: 0.8833 - val_loss: 0.2142 - val_accuracy: 0.9000
Epoch 34/50
8/8 [==============================] - 0s 17ms/step - loss: 0.2731 - accur
acy: 0.8917 - val_loss: 0.2102 - val_accuracy: 0.9667
Epoch 35/50
8/8 [==============================] - 0s 24ms/step - loss: 0.2670 - accur
acy: 0.9000 - val_loss: 0.2049 - val_accuracy: 0.9667
Epoch 36/50
8/8 [==============================] - 0s 18ms/step - loss: 0.2614 - accur
acy: 0.9000 - val_loss: 0.2012 - val_accuracy: 0.9667
Epoch 37/50
8/8 [==============================] - 0s 13ms/step - loss: 0.2564 - accur
acy: 0.9000 - val_loss: 0.1968 - val_accuracy: 0.9667
Epoch 38/50
8/8 [==============================] - 0s 16ms/step - loss: 0.2518 - accur
acy: 0.9083 - val_loss: 0.1921 - val_accuracy: 0.9667
Epoch 39/50
8/8 [==============================] - 0s 13ms/step - loss: 0.2465 - accur
acy: 0.9083 - val_loss: 0.1880 - val_accuracy: 0.9667
Epoch 40/50
8/8 [==============================] - 0s 16ms/step - loss: 0.2415 - accur
acy: 0.9083 - val_loss: 0.1835 - val_accuracy: 0.9667
Epoch 41/50
8/8 [==============================] - 0s 15ms/step - loss: 0.2376 - accur
```

```
acy: 0.9167 - val_loss: 0.1780 - val_accuracy: 0.9667
Epoch 42/50
8/8 [==============================] - 0s 18ms/step - loss: 0.2329 - accur
acy: 0.9167 - val_loss: 0.1739 - val_accuracy: 0.9667
Epoch 43/50
8/8 [==============================] - 0s 16ms/step - loss: 0.2272 - accur
acy: 0.9250 - val_loss: 0.1711 - val_accuracy: 0.9667
Epoch 44/50
8/8 [==============================] - 0s 13ms/step - loss: 0.2237 - accur
acy: 0.9250 - val_loss: 0.1689 - val_accuracy: 0.9667
Epoch 45/50
8/8 [==============================] - 0s 13ms/step - loss: 0.2183 - accur
acy: 0.9250 - val_loss: 0.1641 - val_accuracy: 0.9667
Epoch 46/50
8/8 [==============================] - 0s 13ms/step - loss: 0.2138 - accur
acy: 0.9333 - val_loss: 0.1597 - val_accuracy: 0.9667
Epoch 47/50
8/8 [==============================] - 0s 20ms/step - loss: 0.2097 - accur
acy: 0.9417 - val_loss: 0.1568 - val_accuracy: 0.9667
Epoch 48/50
8/8 [==============================] - 0s 33ms/step - loss: 0.2052 - accur
acy: 0.9417 - val_loss: 0.1526 - val_accuracy: 0.9667
Epoch 49/50
8/8 [==============================] - 0s 20ms/step - loss: 0.2008 - accur
acy: 0.9417 - val_loss: 0.1496 - val_accuracy: 0.9667
Epoch 50/50
8/8 [==============================] - 0s 26ms/step - loss: 0.1971 - accur
acy: 0.9500 - val_loss: 0.1473 - val_accuracy: 0.9667
```

Out[23]:

```
<keras.src.callbacks.History at 0x1d313fb0a60>
```

Analyzing Performance:

In [26]:

```
# Build the regression model
model_reg = models.Sequential()
model_reg.add(layers.Dense(64, activation='relu', input_shape=(X_train.
model_reg.add(layers.Dense(1))  # Output layer for regression

# Compile the model
model_reg.compile(optimizer='adam', loss='mean_squared_error', metrics=
model_reg.compile
```

Out[26]:

```
<bound method Model.compile of <keras.src.engine.sequential.Sequential obj
ect at 0x000001D320980490>>
```

Create a model to analyses the relation between CIE and SEE result

In [27]:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras import models, layers

# Assuming you have a dataset with columns 'CIE' and 'SEE'
# Load your dataset
# For example, assuming you have a CSV file named 'your_dataset.csv'
# df = pd.read_csv('your_dataset.csv')

# For the purpose of this example, let's generate some random data
np.random.seed(42)
num_samples = 1000
cie_data = np.random.rand(num_samples) * 10  # Random CIE values betwee
see_data = 2 * cie_data + np.random.randn(num_samples) * 2  # Linear re

# Create a DataFrame
df = pd.DataFrame({'CIE': cie_data, 'SEE': see_data})

# Split the data into training and testing sets
train_data, test_data = train_test_split(df, test_size=0.2, random_stat

# Standardize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(train_data[['CIE']])
y_train = scaler.fit_transform(train_data[['SEE']])
X_test = scaler.transform(test_data[['CIE']])
y_test = scaler.transform(test_data[['SEE']])

# Build the regression model
model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(1,)))
model.add(layers.Dense(1))  # Output layer for regression

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['ma

# Train the model
model.fit(X_train, y_train, epochs=50, batch_size=16, validation_data=(

```

```
50/50 [==============================] - 0s 7ms/step - loss: 0.1081 - ma
```

Create a model to analyze the relation between crop yield and rain fall rate

In [28]:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras import models, layers

# Assuming you have a dataset with columns 'Crop_Yield' and 'Rainfall_R
# Load your dataset
# For example, assuming you have a CSV file named 'your_dataset.csv'
# df = pd.read_csv('your_dataset.csv')

# For the purpose of this example, let's generate some random data
np.random.seed(42)
num_samples = 1000
rainfall_data = np.random.rand(num_samples) * 100  # Random rainfall va
crop_yield_data = 2 * rainfall_data + np.random.randn(num_samples) * 10

# Create a DataFrame
df = pd.DataFrame({'Rainfall_Rate': rainfall_data, 'Crop_Yield': crop_y

# Split the data into training and testing sets
train_data, test_data = train_test_split(df, test_size=0.2, random_stat

# Standardize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(train_data[['Rainfall_Rate']])
y_train = scaler.fit_transform(train_data[['Crop_Yield']])
X_test = scaler.transform(test_data[['Rainfall_Rate']])
y_test = scaler.transform(test_data[['Crop_Yield']])

# Build the regression model
model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(1,)))
model.add(layers.Dense(1))  # Output layer for regression

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['ma

# Train the model
model.fit(X_train, y_train, epochs=50, batch_size=16, validation_data=(

```

```
e: 0.1466 - val loss: 0.9855 - val mae: 0.8362
```

Build linear regression model using

- Stats model
- Scikit learn

Using Statsmodels

In [29]:

```
import statsmodels.api as sm
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Assuming you have a dataset with columns 'Rainfall_Rate' and 'Crop_Yi
# Load your dataset
# For example, assuming you have a CSV file named 'your_dataset.csv'
# df = pd.read_csv('your_dataset.csv')

# For the purpose of this example, let's generate some random data
np.random.seed(42)
num_samples = 1000
rainfall_data = np.random.rand(num_samples) * 100  # Random rainfall va
crop_yield_data = 2 * rainfall_data + np.random.randn(num_samples) * 10

# Create a DataFrame
df = pd.DataFrame({'Rainfall_Rate': rainfall_data, 'Crop_Yield': crop_y

# Split the data into training and testing sets
train_data, test_data = train_test_split(df, test_size=0.2, random_stat

# Standardize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(train_data[['Rainfall_Rate']])
y_train = train_data['Crop_Yield']

# Add a constant term for the intercept
X_train = sm.add_constant(X_train)

# Create and fit the model
model = sm.OLS(y_train, X_train)
result = model.fit()

# Display the summary of the regression
print(result.summary())

```

```
                            OLS Regression Results
==========================================================================
====
Dep. Variable:             Crop_Yield   R-squared:
0.971
Model:                            OLS   Adj. R-squared:
0.971
Method:                 Least Squares   F-statistic:                 2.673
e+04
Date:                Sun, 26 Nov 2023   Prob (F-statistic):
0.00
Time:                        13:54:17   Log-Likelihood:                -29
79.0
No. Observations:                 800   AIC:                             5
962.
Df Residuals:                     798   BIC:                             5
971.
Df Model:                           1
Covariance Type:            nonrobust
==========================================================================
====
                 coef    std err          t      P>|t|      [0.025      0.
975]
--------------------------------------------------------------------------
----
const        100.0905      0.355    282.119      0.000      99.394     10
0.787
x1            58.0053      0.355    163.496      0.000      57.309      5
8.702
==========================================================================
====
Omnibus:                        1.240   Durbin-Watson:
1.924
Prob(Omnibus):                  0.538   Jarque-Bera (JB):
1.235
Skew:                           0.014   Prob(JB):
0.539
Kurtosis:                       2.810   Cond. No.
1.00
==========================================================================
====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is cor
rectly specified.
```

Using Scikit-learn:

In [30]:

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy as np

# Assuming you have a dataset with columns 'Rainfall_Rate' and 'Crop_Yi
# Load your dataset
# For example, assuming you have a CSV file named 'your_dataset.csv'
# df = pd.read_csv('your_dataset.csv')

# For the purpose of this example, let's generate some random data
np.random.seed(42)
num_samples = 1000
rainfall_data = np.random.rand(num_samples) * 100  # Random rainfall va
crop_yield_data = 2 * rainfall_data + np.random.randn(num_samples) * 10

# Create a DataFrame
df = pd.DataFrame({'Rainfall_Rate': rainfall_data, 'Crop_Yield': crop_y

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df[['Rainfall_Rate'

# Standardize the data (not strictly necessary for linear regression, b
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create and fit the model
model = LinearRegression()
model.fit(X_train_scaled, y_train)

# Display the coefficients and intercept
print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)

```

```
Coefficients: [58.00531307]
Intercept: 100.09054272144182
```

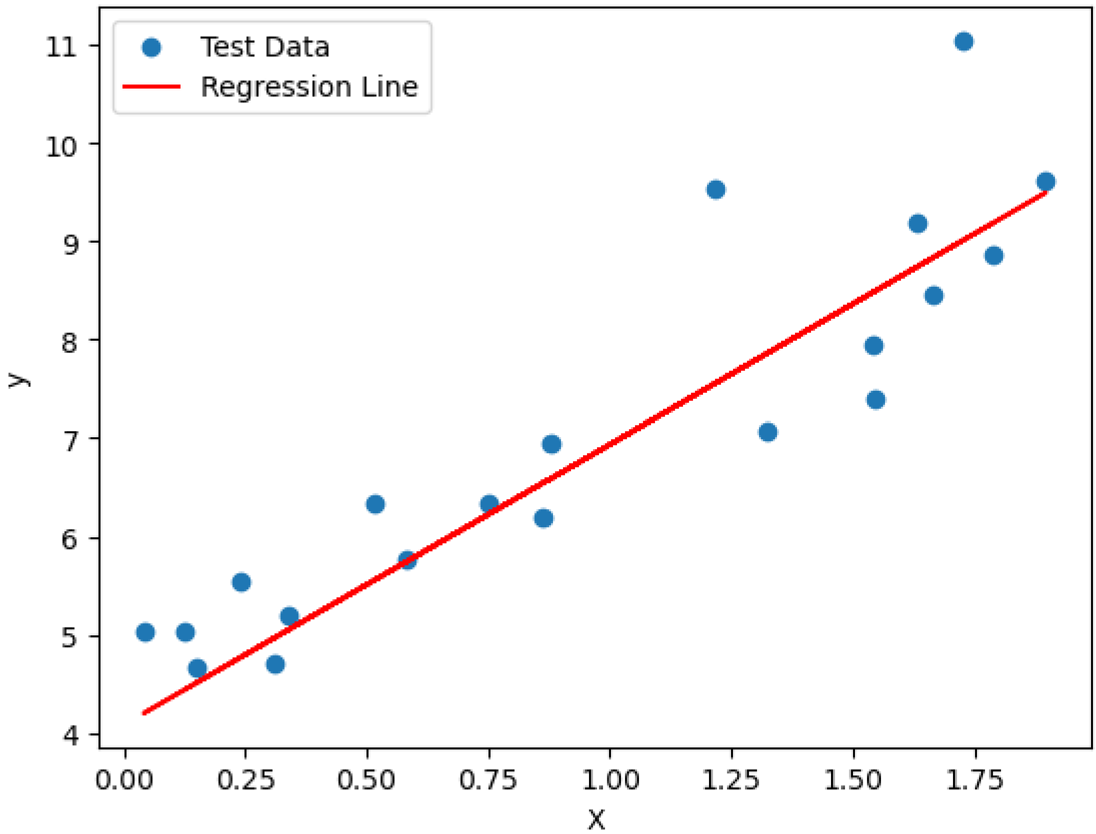
Implementation in python

- Build regression model
- Evaluate the model
- To minimize the cost function

In [31]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Generate some random data for demonstration
np.random.seed(42)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)

# 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

# Add a bias term to the features
X_train_bias = np.c_[np.ones((len(X_train), 1)), X_train]
X_test_bias = np.c_[np.ones((len(X_test), 1)), X_test]

# Initialize random coefficients
theta = np.random.randn(2, 1)

# Define the learning rate and number of iterations
learning_rate = 0.01
n_iterations = 1000

# Implement gradient descent to minimize the cost function
for iteration in range(n_iterations):
    gradients = 2/len(X_train) * X_train_bias.T.dot(X_train_bias.dot(th
    theta = theta - learning_rate * gradients

# Evaluate the model on the test set
y_pred = X_test_bias.dot(theta)

# Calculate the mean squared error
mse = mean_squared_error(y_test, y_pred)

# Print the learned coefficients and MSE
print("Learned Coefficients:", theta.flatten())
print("Mean Squared Error on Test Set:", mse)

# Visualize the data and the learned regression line
plt.scatter(X_test, y_test, label='Test Data')
plt.plot(X_test, y_pred, color='red', label='Regression Line')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.show()

```

```
Learned Coefficients: [4.08358595 2.85220796]
Mean Squared Error on Test Set: 0.6645849137924722
```



build Logistic regression model in pythonEvaluation and optimization of the model

In [32]:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classific

# Assuming you have a dataset with features and labels
# For example, assuming you have a CSV file named 'your_dataset.csv'
# df = pd.read_csv('your_dataset.csv')

# For the purpose of this example, let's generate some random data
np.random.seed(42)
num_samples = 1000
X = np.random.rand(num_samples, 2) * 10  # Random features
y = (X[:, 0] + X[:, 1] > 10).astype(int)  # Binary classification task

# 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

# Standardize the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Build the logistic regression model
model = LogisticRegression()

# Train the model
model.fit(X_train_scaled, y_train)

# Evaluate the model on the test set
y_pred = model.predict(X_test_scaled)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)

# Print confusion matrix and classification report
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)

# Hyperparameter tuning using GridSearchCV
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}  # Regularization pa
grid_search = GridSearchCV(LogisticRegression(), param_grid, cv=5)
grid_search.fit(X_train_scaled, y_train)

# Print the best hyperparameters
print("Best Hyperparameters:", grid_search.best_params_)

# Evaluate the model with the best hyperparameters
best_model = grid_search.best_estimator_
y_pred_best = best_model.predict(X_test_scaled)

# Calculate accuracy for the optimized model
accuracy_best = accuracy_score(y_test, y_pred_best)

# Print confusion matrix and classification report for the optimized mo
```

```
conf_matrix_best = confusion_matrix(y_test, y_pred_best)
class_report_best = classification_report(y_test, y_pred_best)

print("Optimized Model Accuracy:", accuracy_best)
print("Optimized Model Confusion Matrix:\n", conf_matrix_best)
print("Optimized Model Classification Report:\n", class_report_best)

```

```
Accuracy: 1.0
Confusion Matrix:
 [[106   0]
 [  0  94]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00       106
           1       1.00      1.00      1.00        94

    accuracy                           1.00       200
   macro avg       1.00      1.00      1.00       200
weighted avg       1.00      1.00      1.00       200

Best Hyperparameters: {'C': 1}
Optimized Model Accuracy: 1.0
Optimized Model Confusion Matrix:
 [[106   0]
 [  0  94]]
Optimized Model Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00       106
           1       1.00      1.00      1.00        94

    accuracy                           1.00       200
   macro avg       1.00      1.00      1.00       200
weighted avg       1.00      1.00      1.00       200
```

Feature scaling with StandardScalar() or other method Dropping unnecessary features

Data splitting Dealing with imbalanced dataset

In [34]:
```
pip install pandas scikit-learn imbalanced-learn

```

```
0:20:14

WARNING: Retrying (Retry(total=4, connect=None, read=None, redirect=Non
e, status=None)) after connection broken by 'ReadTimeoutError("HTTPSConn
ectionPool(host='pypi.org', port=443): Read timed out. (read timeout=1
5)")': /simple/numpy/
WARNING: Retrying (Retry(total=3, connect=None, read=None, redirect=Non
e, status=None)) after connection broken by 'ReadTimeoutError("HTTPSConn
ectionPool(host='pypi.org', port=443): Read timed out. (read timeout=1
5)")': /simple/numpy/
WARNING: Retrying (Retry(total=2, connect=None, read=None, redirect=Non
e, status=None)) after connection broken by 'ReadTimeoutError("HTTPSConn
ectionPool(host='pypi.org', port=443): Read timed out. (read timeout=1
5)")': /simple/numpy/
WARNING: Retrying (Retry(total=1, connect=None, read=None, redirect=Non
e, status=None)) after connection broken by 'ReadTimeoutError("HTTPSConn
ectionPool(host='pypi.org', port=443): Read timed out. (read timeout=1
5)")': /simple/numpy/
WARNING: Retrying (Retry(total=0, connect=None, read=None, redirect=Non
e, status=None)) after connection broken by 'ReadTimeoutError("HTTPSConn
```

In [36]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
from sklearn.datasets import make_classification

# Generate synthetic imbalanced data for demonstration
X, y = make_classification(n_samples=1000, n_features=10, n_classes=2,

# Create a DataFrame (replace this with your actual dataset loading log
df = pd.DataFrame(X, columns=[f'feature_{i}' for i in range(X.shape[1])
df['target'] = y

# Check existing column names
print("Existing Columns:", df.columns)

# Drop unnecessary features (replace 'feature_to_drop' with actual feat
features_to_drop = ['feature_to_drop']

# Check if the columns to drop exist in the DataFrame
columns_to_drop = [col for col in features_to_drop if col in df.columns
df = df.drop(columns=columns_to_drop)

# Separate features (X) and target variable (y)
X = df.drop(columns=['target'])
y = df['target']

# Feature scaling using StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Data splitting into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_s

# Dealing with an imbalanced dataset using SMOTE
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_tr

# Now you can use X_train_resampled and y_train_resampled for training
# Remember to use X_test for evaluating the model, not X_train

```

```
Existing Columns: Index(['feature_0', 'feature_1', 'feature_2', 'feature_
3', 'feature_4',
       'feature_5', 'feature_6', 'feature_7', 'feature_8', 'feature_9',
       'target'],
      dtype='object')
```

Building Shallow Neural Network with Keras Dense Layer

In [37]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from keras.models import Sequential
from keras.layers import Dense

# Assuming you have a dataset with features and labels
# For example, assuming you have a CSV file named 'your_dataset.csv'
# df = pd.read_csv('your_dataset.csv')

# For the purpose of this example, let's generate some random data
np.random.seed(42)
num_samples = 1000
X = np.random.rand(num_samples, 10)  # Random features
y = np.random.randint(2, size=num_samples)  # Binary classification lab

# Create a DataFrame
df = pd.DataFrame(X, columns=[f'feature_{i}' for i in range(X.shape[1])
df['target'] = y

# Separate features (X) and target variable (y)
X = df.drop(columns=['target'])
y = df['target']

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_s

# Build the shallow neural network model
model = Sequential()
model.add(Dense(units=16, input_dim=X_train.shape[1], activation='relu'
model.add(Dense(units=1, activation='sigmoid'))

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['a

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=

# Evaluate the model on the test set
y_pred_prob = model.predict(X_test)
y_pred = (y_pred_prob > 0.5).astype(int)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Test Accuracy:", accuracy)

```

```
Epoch 1/10
20/20 [==============================] - 1s 17ms/step - loss: 0.7839 - acc
uracy: 0.4828 - val_loss: 0.7774 - val_accuracy: 0.4875
Epoch 2/10
20/20 [==============================] - 0s 7ms/step - loss: 0.7614 - accu
racy: 0.4844 - val_loss: 0.7619 - val_accuracy: 0.4688
Epoch 3/10
20/20 [==============================] - 0s 7ms/step - loss: 0.7452 - accu
racy: 0.4891 - val_loss: 0.7473 - val_accuracy: 0.4688
Epoch 4/10
20/20 [==============================] - 0s 6ms/step - loss: 0.7322 - accu
racy: 0.4891 - val_loss: 0.7357 - val_accuracy: 0.4688
Epoch 5/10
20/20 [==============================] - 0s 7ms/step - loss: 0.7219 - accu
racy: 0.4953 - val_loss: 0.7263 - val_accuracy: 0.4750
Epoch 6/10
20/20 [==============================] - 0s 6ms/step - loss: 0.7133 - accu
racy: 0.4953 - val_loss: 0.7188 - val_accuracy: 0.4688
Epoch 7/10
20/20 [==============================] - 0s 7ms/step - loss: 0.7061 - accu
racy: 0.4984 - val_loss: 0.7139 - val_accuracy: 0.4625
Epoch 8/10
20/20 [==============================] - 0s 5ms/step - loss: 0.7004 - accu
racy: 0.5063 - val_loss: 0.7085 - val_accuracy: 0.4938
Epoch 9/10
20/20 [==============================] - 0s 7ms/step - loss: 0.6954 - accu
racy: 0.5312 - val_loss: 0.7047 - val_accuracy: 0.4938
Epoch 10/10
20/20 [==============================] - 0s 6ms/step - loss: 0.6921 - accu
racy: 0.5375 - val_loss: 0.7025 - val_accuracy: 0.5000
7/7 [==============================] - 0s 3ms/step
Test Accuracy: 0.505
```

Building Deep Neural Network with Keras Dense Layers

In [38]:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from keras.models import Sequential
from keras.layers import Dense

# Assuming you have a dataset with features and labels
# For example, assuming you have a CSV file named 'your_dataset.csv'
# df = pd.read_csv('your_dataset.csv')

# For the purpose of this example, let's generate some random data
np.random.seed(42)
num_samples = 1000
X = np.random.rand(num_samples, 10)  # Random features
y = np.random.randint(2, size=num_samples)  # Binary classification lab

# Create a DataFrame
df = pd.DataFrame(X, columns=[f'feature_{i}' for i in range(X.shape[1])
df['target'] = y

# Separate features (X) and target variable (y)
X = df.drop(columns=['target'])
y = df['target']

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_s

# Build the deep neural network model
model = Sequential()
model.add(Dense(units=64, input_dim=X_train.shape[1], activation='relu'
model.add(Dense(units=32, activation='relu'))
model.add(Dense(units=16, activation='relu'))
model.add(Dense(units=1, activation='sigmoid'))

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['a

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=

# Evaluate the model on the test set
y_pred_prob = model.predict(X_test)
y_pred = (y_pred_prob > 0.5).astype(int)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Test Accuracy:", accuracy)

```

```
Epoch 1/10
20/20 [==============================] - 2s 22ms/step - loss: 0.7064 - acc
uracy: 0.4922 - val_loss: 0.6907 - val_accuracy: 0.5500
Epoch 2/10
20/20 [==============================] - 0s 8ms/step - loss: 0.6852 - accu
racy: 0.5359 - val_loss: 0.6877 - val_accuracy: 0.5437
Epoch 3/10
20/20 [==============================] - 0s 7ms/step - loss: 0.6764 - accu
racy: 0.5922 - val_loss: 0.6870 - val_accuracy: 0.5312
Epoch 4/10
20/20 [==============================] - 0s 7ms/step - loss: 0.6711 - accu
racy: 0.5953 - val_loss: 0.6866 - val_accuracy: 0.5250
Epoch 5/10
20/20 [==============================] - 0s 6ms/step - loss: 0.6629 - accu
racy: 0.6109 - val_loss: 0.6858 - val_accuracy: 0.5312
Epoch 6/10
20/20 [==============================] - 0s 6ms/step - loss: 0.6567 - accu
racy: 0.6328 - val_loss: 0.6860 - val_accuracy: 0.5500
Epoch 7/10
20/20 [==============================] - 0s 6ms/step - loss: 0.6502 - accu
racy: 0.6375 - val_loss: 0.6842 - val_accuracy: 0.5500
Epoch 8/10
20/20 [==============================] - 0s 6ms/step - loss: 0.6446 - accu
racy: 0.6406 - val_loss: 0.6889 - val_accuracy: 0.5125
Epoch 9/10
20/20 [==============================] - 0s 7ms/step - loss: 0.6412 - accu
racy: 0.6438 - val_loss: 0.6918 - val_accuracy: 0.4875
Epoch 10/10
20/20 [==============================] - 0s 6ms/step - loss: 0.6322 - accu
racy: 0.6750 - val_loss: 0.6894 - val_accuracy: 0.5188
7/7 [==============================] - 0s 5ms/step
Test Accuracy: 0.55
```

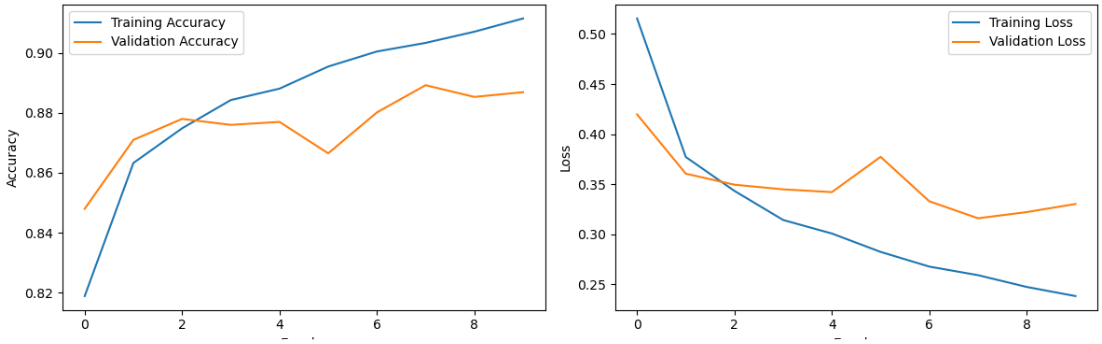
Create a complete end to end neural network model using Keras Sequential Model and Keras Layer API

In [39]:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt

# Load and preprocess the Fashion-MNIST dataset
(train_images, train_labels), (test_images, test_labels) = fashion_mnis

# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0

# One-hot encode the labels
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

# Build the neural network using Keras Sequential Model and Keras Layer
model = models.Sequential()

# Flatten layer to flatten the input
model.add(layers.Flatten(input_shape=(28, 28)))

# Dense layers with ReLU activation
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(64, activation='relu'))

# Output layer with softmax activation for multi-class classification
model.add(layers.Dense(10, activation='softmax'))

# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Display the summary of the model
model.summary()

# Train the model
history = model.fit(train_images, train_labels, epochs=10, validation_s

# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(test_images, test_labels)
print(f'Test Accuracy: {test_accuracy}')

# Plot training history (optional)
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
```

```
plt.tight_layout()
plt.show()
```

```
Epoch 9/10
1500/1500 [==============================] - 9s 6ms/step - loss: 0.2475
- accuracy: 0.9071 - val_loss: 0.3222 - val_accuracy: 0.8853
Epoch 10/10
1500/1500 [==============================] - 8s 5ms/step - loss: 0.2384
- accuracy: 0.9115 - val_loss: 0.3303 - val_accuracy: 0.8869
313/313 [==============================] - 1s 3ms/step - loss: 0.3461 -
accuracy: 0.8807
Test Accuracy: 0.8806999921798706
```



MNIST dataset (classify handwritten numerals)

or fashion-MNIST dataset or dataset from other source

This code defines a simple convolutional neural network (CNN) using the Keras API with TensorFlow backend. It loads the MNIST dataset, preprocesses the data, builds the model, compiles it, trains it, and evaluates its performance on the test set.

In [21]:

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

# Load and preprocess the MNIST dataset
(train_images, train_labels), (test_images, test_labels) = mnist.load_d
train_images = train_images.reshape((60000, 28, 28, 1)).astype('float32
test_images = test_images.reshape((10000, 28, 28, 1)).astype('float32')
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

# Build a simple convolutional neural network (CNN)
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28,
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
model.fit(train_images, train_labels, epochs=5, batch_size=64, validati

# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Test accuracy: {test_acc}')


```

```
Epoch 1/5
750/750 [==============================] - 60s 77ms/step - loss: 0.2014 -
accuracy: 0.9379 - val_loss: 0.0767 - val_accuracy: 0.9764
Epoch 2/5
750/750 [==============================] - 59s 78ms/step - loss: 0.0564 -
accuracy: 0.9827 - val_loss: 0.0551 - val_accuracy: 0.9818
Epoch 3/5
750/750 [==============================] - 50s 67ms/step - loss: 0.0391 -
accuracy: 0.9873 - val_loss: 0.0470 - val_accuracy: 0.9863
Epoch 4/5
750/750 [==============================] - 55s 73ms/step - loss: 0.0297 -
accuracy: 0.9903 - val_loss: 0.0448 - val_accuracy: 0.9872
Epoch 5/5
750/750 [==============================] - 61s 82ms/step - loss: 0.0235 -
accuracy: 0.9920 - val_loss: 0.0416 - val_accuracy: 0.9895
313/313 [==============================] - 4s 12ms/step - loss: 0.0318 - a
ccuracy: 0.9893
Test accuracy: 0.989300012588501
```

In [18]:
```
from sklearn import datasets

```

In [2]:
```
digits = datasets.load_digits()
dir(digits)
```

Out[2]:
```
['DESCR', 'data', 'feature_names', 'frame', 'images', 'target', 'target_na
mes']
```

In [3]:
```
print(digits.images[0])
```

```
[[ 0.  0.  5. 13.  9.  1.  0.  0.]
 [ 0.  0. 13. 15. 10. 15.  5.  0.]
 [ 0.  3. 15.  2.  0. 11.  8.  0.]
 [ 0.  4. 12.  0.  0.  8.  8.  0.]
 [ 0.  5.  8.  0.  0.  9.  8.  0.]
 [ 0.  4. 11.  0.  1. 12.  7.  0.]
 [ 0.  2. 14.  5. 10. 12.  0.  0.]
 [ 0.  0.  6. 13. 10.  0.  0.  0.]]
```

In [5]:
```
import matplotlib.pyplot as plt
def plot_multi(i):
 nplots = 16
 fig = plt.figure(figsize=(15, 15))
 for j in range(nplots):
    plt.subplot(4, 4, j+1)
    plt.imshow(digits.images[i+j], cmap='binary')
    plt.title(digits.target[i+j])
    plt.axis('off')
    plt.show()
    plot_multi(0)
y = digits.target
x = digits.images.reshape((len(digits.images), -1))
# gives the shape of the data
x.shape

```

Out[5]:
```
(1797, 64)
```

In [6]:
```
x[0]
```

Out[6]:
```
array([ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.,  0.,  0., 13., 15., 10.,
       15.,  5.,  0.,  0.,  3., 15.,  2.,  0., 11.,  8.,  0.,  0.,  4.,
       12.,  0.,  0.,  8.,  8.,  0.,  0.,  5.,  8.,  0.,  0.,  9.,  8.,
        0.,  0.,  4., 11.,  0.,  1., 12.,  7.,  0.,  0.,  2., 14.,  5.,
       10., 12.,  0.,  0.,  0.,  0.,  6., 13., 10.,  0.,  0.,  0.])
```

In [7]:
```
x_train = x[:1000]
y_train = y[:1000]
x_test = x[1000:]
y_test = y[1000:]

```

In [8]:
```
x_train
```

Out[8]:
```
array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ..., 10.,  0.,  0.],
       [ 0.,  0.,  0., ..., 16.,  9.,  0.],
       ...,
       [ 0.,  0.,  3., ...,  9.,  0.,  0.],
       [ 0.,  0.,  0., ...,  6.,  0.,  0.],
       [ 0.,  0.,  9., ..., 10.,  0.,  0.]])
```

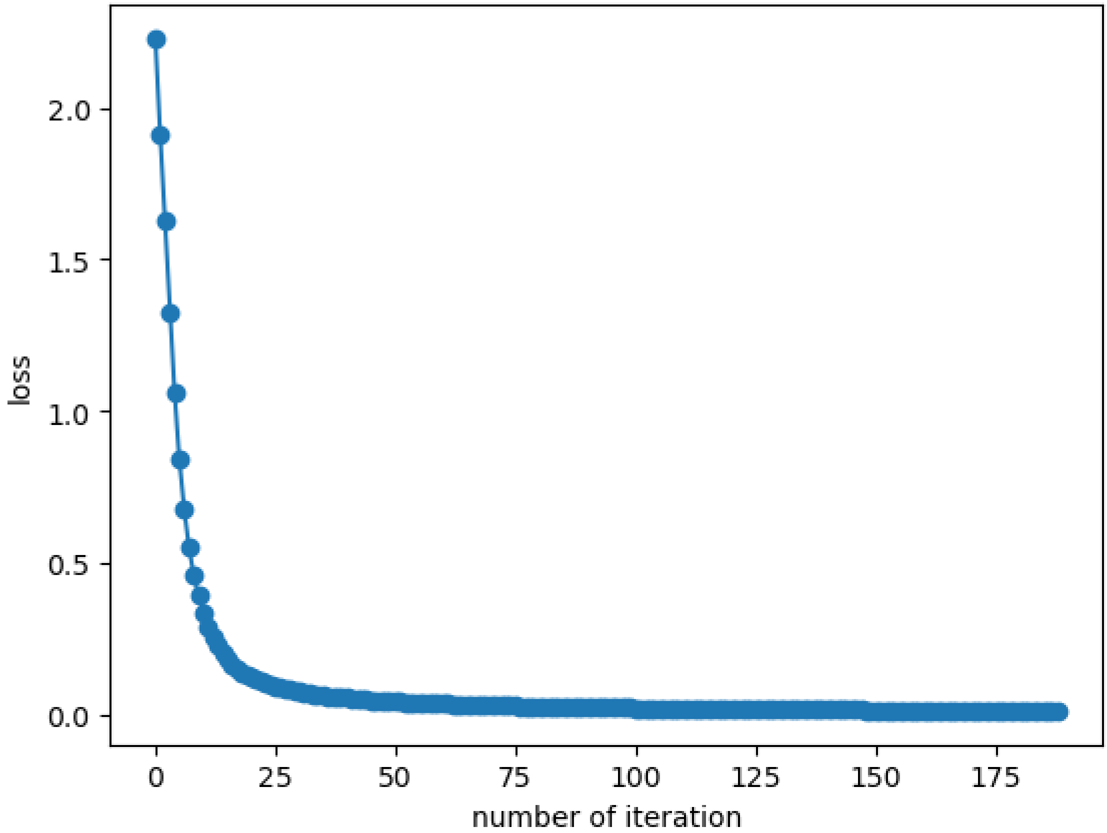
In [11]:
```
x_test
```

Out[11]:
```
array([[ 0.,  0.,  1., ..., 15., 16., 15.],
       [ 0.,  0.,  0., ...,  5.,  0.,  0.],
       [ 0.,  0.,  6., ...,  3.,  0.,  0.],
       ...,
       [ 0.,  0.,  1., ...,  6.,  0.,  0.],
       [ 0.,  0.,  2., ..., 12.,  0.,  0.],
       [ 0.,  0., 10., ..., 12.,  1.,  0.]])
```

In [12]:
```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(15,),
 activation='logistic',
alpha=1e-4, solver='sgd',
 tol=1e-4, random_state=1,
 learning_rate_init=.1,
verbose=True)
mlp.fit(x_train, y_train)

```

```
Iteration 1, loss = 2.22958289
Iteration 2, loss = 1.91207743
Iteration 3, loss = 1.62507727
Iteration 4, loss = 1.32649842
Iteration 5, loss = 1.06100535
Iteration 6, loss = 0.83995513
Iteration 7, loss = 0.67806075
Iteration 8, loss = 0.55175832
Iteration 9, loss = 0.45840445
Iteration 10, loss = 0.39149735
Iteration 11, loss = 0.33676351
Iteration 12, loss = 0.29059880
Iteration 13, loss = 0.25437208
Iteration 14, loss = 0.22838372
Iteration 15, loss = 0.20200554
Iteration 16, loss = 0.18186565
Iteration 17, loss = 0.16461183
Iteration 18, loss = 0.14990228
Iteration 19, loss = 0.13892154
Iteration 20, loss = 0.12833784
```

In [13]:
```
fig, axes = plt.subplots(1,1)
axes.plot(mlp.loss_curve_, 'o-')
axes.set_xlabel("number of iteration")
axes.set_ylabel("loss")
plt.show()

```



In [15]:
```
predictions = mlp.predict(x_test)
predictions[:50]

```

Out[15]:
```
array([1, 4, 0, 5, 3, 6, 9, 6, 1, 7, 5, 4, 4, 7, 2, 8, 2, 2, 5, 7, 9, 5,
       4, 4, 9, 0, 8, 9, 8, 0, 1, 2, 3, 4, 5, 6, 7, 8, 3, 0, 1, 2, 3, 4,
       5, 6, 7, 8, 5, 0])
```

In [16]:
```
y_test[:50]

```

Out[16]:
```
array([1, 4, 0, 5, 3, 6, 9, 6, 1, 7, 5, 4, 4, 7, 2, 8, 2, 2, 5, 7, 9, 5,
       4, 4, 9, 0, 8, 9, 8, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4,
       5, 6, 7, 8, 9, 0])
```

In [17]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, predictions)
```

Out[17]:
```
0.9146800501882058
```

In [ ]:
```

```