Lab - 01: Introduction to Machine Learning and Python Environment

Aim: To get familiar with machine learning concepts and set up the Python environment for ML experiments.

1. Setting Up Python and Jupyter Notebook

[1]:
```
!pip install numpy pandas sklearn matplotlib seaborn

import numpy
import pandas
import sklearn
import matplotlib
import seaborn

print("All libraries imported successfully")
```

```
All libraries imported successfully
```

2. Basic Python Operations for ML

[2]:
```
# Basic data types
a = 10
b = 5.5

# List
numbers = [1, 2, 3, 4]

# Dictionary
student = {"name": "Alice", "age": 20}

# Loop
for i in numbers:
    print(i)

# Function
def square(x):
    return x * x

print(square(4))
```

```
1
2
3
4
16
```

1.3.1 NumPy Operations

[3]:
```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print("Mean:", np.mean(arr))
print("Sum:", np.sum(arr))
print("Standard Deviation:", np.std(arr))
```

```
Mean: 3.0
Sum: 15
Standard Deviation: 1.4142135623730951
```

1.4 3. Loading and Exploring Dataset (Iris)

[4]:
```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

df.head()
```

[4]:
```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
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
```

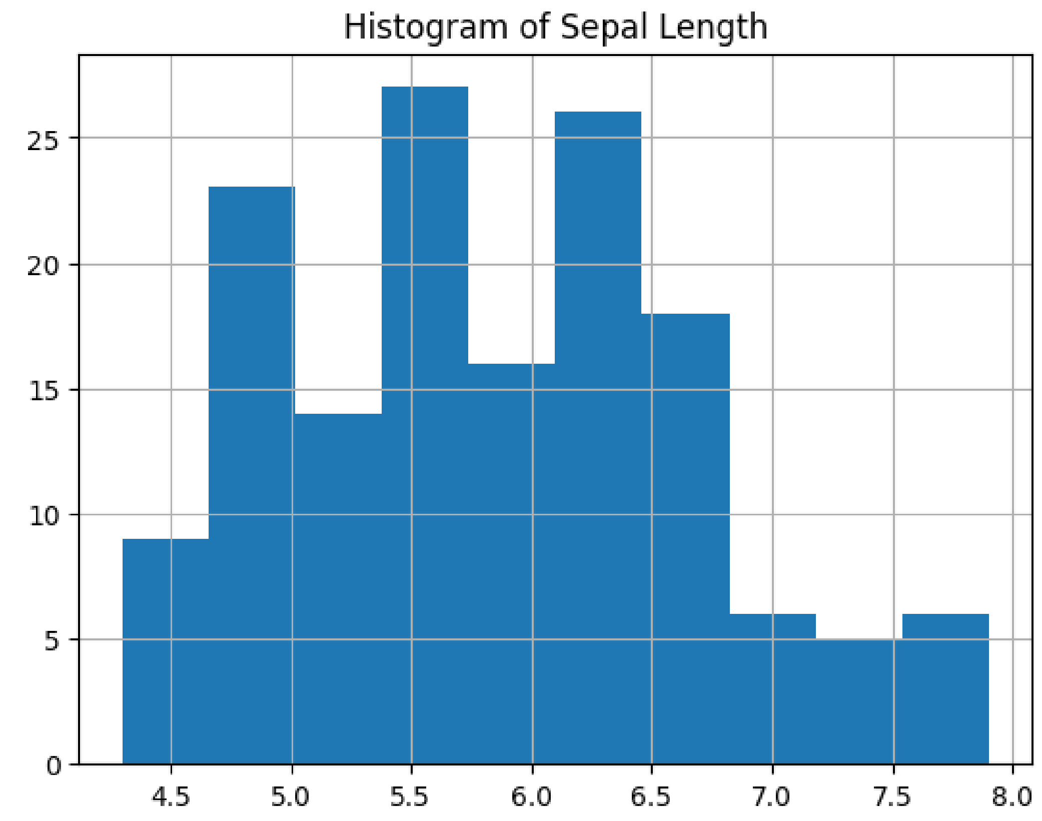
```
4       0
```

[5]:
```
df.describe()
```

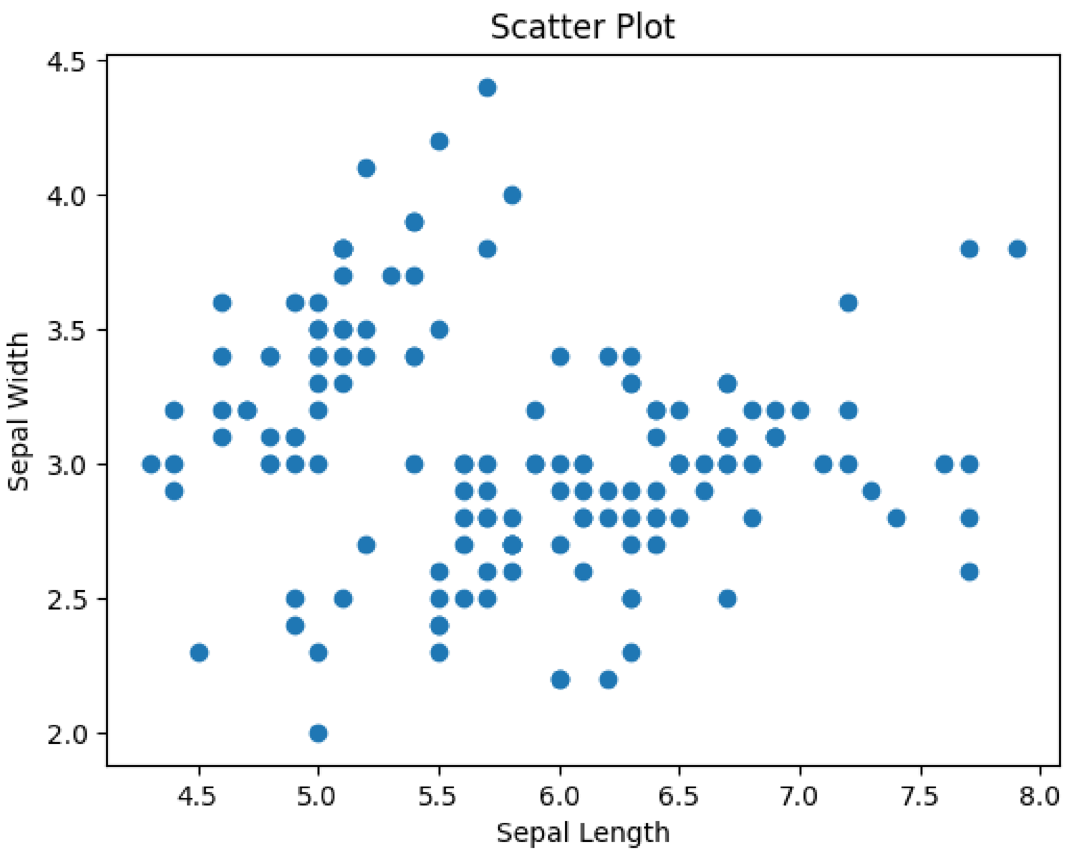
[5]:
```
       sepal length (cm)  sepal width (cm)  petal length (cm)  \
count         150.000000        150.000000         150.000000
mean            5.843333          3.057333           3.758000
std             0.828066          0.435866           1.765298
min             4.300000          2.000000           1.000000
25%             5.100000          2.800000           1.600000
50%             5.800000          3.000000           4.350000
75%             6.400000          3.300000           5.100000
max             7.900000          4.400000           6.900000

       petal width (cm)      target
count        150.000000  150.000000
mean           1.199333    1.000000
std            0.762238    0.819232
min            0.100000    0.000000
25%            0.300000    0.000000
50%            1.300000    1.000000
75%            1.800000    2.000000
max            2.500000    2.000000
```

1.5 4. Data Visualization

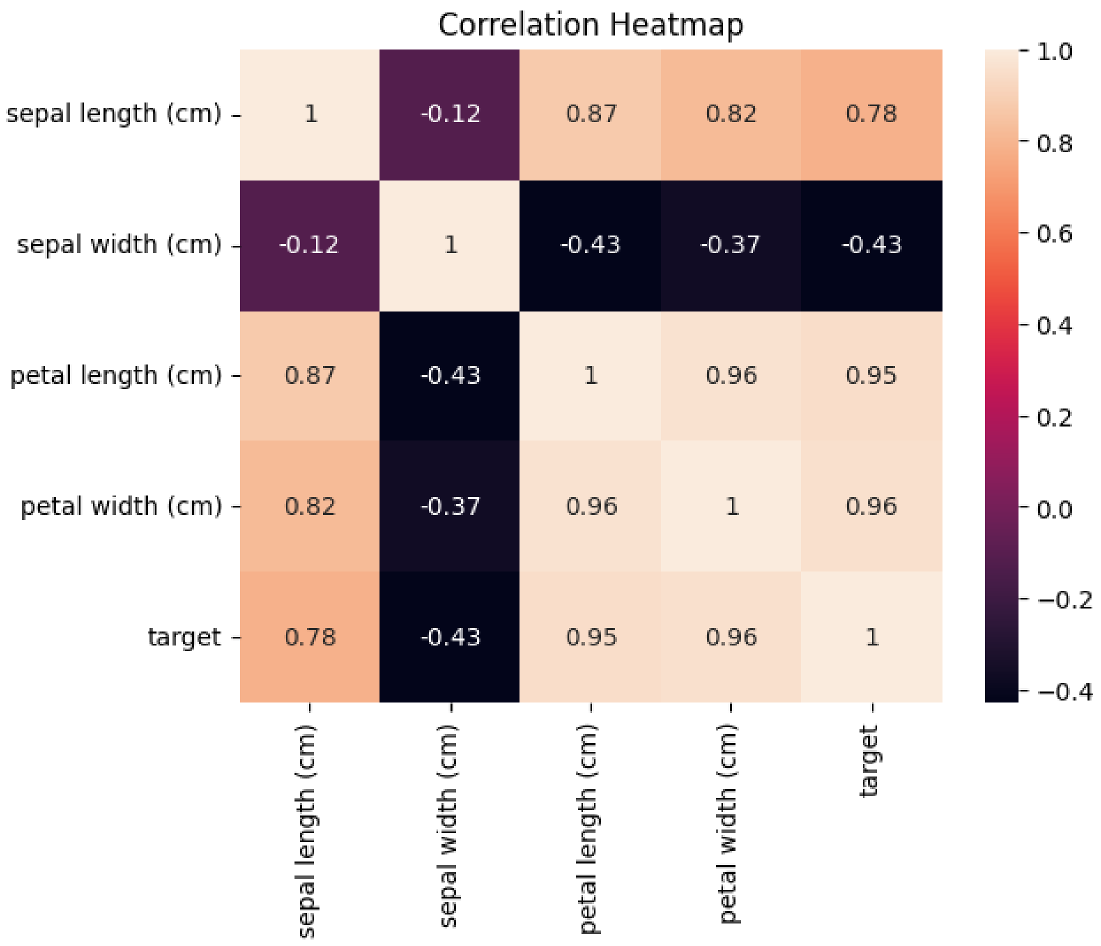
[6]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Histogram
df['sepal length (cm)'].hist()
plt.title("Histogram of Sepal Length")
plt.show()
```



[7]:
```
# Scatter plot
plt.scatter(df['sepal length (cm)'], df['sepal width (cm)'])
plt.xlabel("Sepal Length")
plt.ylabel("Sepal Width")
plt.title("Scatter Plot")
plt.show()
```



```
[8]: # Correlation heatmap
     sns.heatmap(df.corr(), annot=True)
     plt.title("Correlation Heatmap")
     plt.show()
```



1.6 5. Train-Test Split

[9]:
```
from sklearn.model_selection import train_test_split

X = df.iloc[:, :-1]
y = df['target']

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

print("Training shape:", X_train.shape)
print("Testing shape:", X_test.shape)
```

```
Training shape: (120, 4)
Testing shape: (30, 4)
```

1.7 Result

The Python environment was successfully set up and basic ML workflow was performed.

Lab – 02: Data Preprocessing in Machine Learning

Aim: To understand and implement techniques for cleaning, transforming, and preparing data for machine learning.

1. Handling Missing Values

Missing values can negatively affect machine learning models. This experiment demonstrates how to identify and handle missing data.

[1]:
```python
import pandas as pd
import numpy as np

# Create a sample dataset with missing values
data = {
    'Age': [25, 30, np.nan, 35, 40],
    'Salary': [50000, np.nan, 60000, 65000, np.nan],
    'Department': ['HR', 'IT', 'IT', np.nan, 'HR']
}

df = pd.DataFrame(data)
df
```

[1]:
```
    Age   Salary Department
0  25.0  50000.0         HR
1  30.0      NaN         IT
2   NaN  60000.0         IT
3  35.0  65000.0        NaN
4  40.0      NaN         HR
```

1.2.1 Identifying Missing Values

[2]:
```python
# Check for missing values
df.isnull()
```

[2]:
```
     Age  Salary  Department
0  False   False       False
1  False    True       False
```

```
2   True   False       False
3  False   False        True
4  False    True       False
```

[3]:
```
# Count missing values in each column
df.isnull().sum()
```

[3]:
```
Age           1
Salary        2
Department    1
dtype: int64
```

1.2.2 Handling Missing Values by Removing Rows

[4]:
```
df_drop = df.dropna()
df_drop
```

[4]:
```
    Age   Salary Department
0  25.0  50000.0         HR
```

1.2.3 Handling Missing Values by Filling with Mean / Median / Mode

[6]:
```
# Fill numerical columns

df_filled = df.copy()
df_filled['Age'].fillna(df['Age'].mean(), inplace=True)
df_filled['Salary'].fillna(df['Salary'].median(), inplace=True)

# Fill categorical column with mode

df_filled['Department'].fillna(df['Department'].mode()[0], inplace=True)
print(df_filled)
```

```
    Age   Salary Department
0  25.0  50000.0         HR
1  30.0  60000.0         IT
2  32.5  60000.0         IT
3  35.0  65000.0         HR
4  40.0  60000.0         HR
```

```
/tmp/ipython-input-2807380802.py:3: FutureWarning: A value is trying to be set
on a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.
```

```
For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  df_filled['Age'].fillna(df['Age'].mean(), inplace=True)
/tmp/ipython-input-2807380802.py:4: FutureWarning: A value is trying to be set
on a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  df_filled['Salary'].fillna(df['Salary'].median(), inplace=True)
/tmp/ipython-input-2807380802.py:7: FutureWarning: A value is trying to be set
on a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  df_filled['Department'].fillna(df['Department'].mode()[0], inplace=True)
```

1.2.4 Forward Fill and Backward Fill

[7]:
```python
# Forward fill

df_ffill = df.fillna(method='ffill')
df_ffill
```

```
/tmp/ipython-input-3129354710.py:3: FutureWarning: DataFrame.fillna with
'method' is deprecated and will raise in a future version. Use obj.ffill() or
obj.bfill() instead.
  df_ffill = df.fillna(method='ffill')
```

[7]:
```
    Age   Salary Department
0  25.0  50000.0         HR
1  30.0  50000.0         IT
```

```
2  30.0  60000.0         IT
3  35.0  65000.0         IT
4  40.0  65000.0         HR
```

[8]:
```
# Backward fill

df_bfill = df.fillna(method='bfill')
df_bfill
```

```
/tmp/ipython-input-2310688756.py:3: FutureWarning: DataFrame.fillna with
'method' is deprecated and will raise in a future version. Use obj.ffill() or
obj.bfill() instead.
  df_bfill = df.fillna(method='bfill')
```

[8]:
```
    Age   Salary Department
0  25.0  50000.0         HR
1  30.0  60000.0         IT
2  35.0  60000.0         IT
3  35.0  65000.0         HR
4  40.0      NaN         HR
```

2. Encoding Categorical Data

Machine learning models require numerical input. Categorical features must be encoded before training models.

[9]:
```
# Identify categorical columns
df_filled.dtypes
```

[9]:
```
Age            float64
Salary         float64
Department      object
dtype: object
```

Label Encoding (Ordinal Data)

[10]:
```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

df_filled['Department_LabelEncoded'] = le.fit_transform(df_filled['Department'])
df_filled
```

[10]:
```
    Age   Salary Department  Department_LabelEncoded
0  25.0  50000.0         HR                        0
1  30.0  60000.0         IT                        1
2  32.5  60000.0         IT                        1
```

```
3  35.0  65000.0        HR                          0
4  40.0  60000.0        HR                          0
```

1.3.2 One-Hot Encoding (Nominal Data)

[11]:
```
# One-hot encoding using pandas
df_onehot = pd.get_dummies(df_filled, columns=['Department'])
df_onehot
```

[11]:
```
    Age   Salary  Department_LabelEncoded  Department_HR  Department_IT
0  25.0  50000.0                        0           True          False
1  30.0  60000.0                        1          False           True
2  32.5  60000.0                        1          False           True
3  35.0  65000.0                        0           True          False
4  40.0  60000.0                        0           True          False
```

1.4 Result

Data preprocessing techniques including handling missing values and encoding categorical data were successfully implemented. The dataset is now suitable for machine learning models.

Lab – 03: Exploratory Data Analysis (EDA) in Machine Learning

1.1 Aim: To explore and analyze datasets to uncover patterns, detect anomalies, and summarize key statistics.

1.2 Dataset Used

Iris Dataset from sklearn is used for Exploratory Data Analysis.

[1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris

iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['species'] = iris.target
df
```

[1]:
```
     sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
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
..                 …                 …                  …                 …
145                6.7               3.0                5.2               2.3
146                6.3               2.5                5.0               1.9
147                6.5               3.0                5.2               2.0
148                6.2               3.4                5.4               2.3
149                5.9               3.0                5.1               1.8

     species
0          0
1          0
2          0
```

```
3         0
4         0
..      …
145       2
146       2
147       2
148       2
149       2

[150 rows x 5 columns]
```

1.3 1. Loading and Inspecting Data

[2]:
```
df.head()
```

[2]:
```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
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

   species
0        0
1        0
2        0
3        0
4        0
```

[3]:
```
df.tail()
```

[3]:
```
     sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
145                6.7               3.0                5.2               2.3
146                6.3               2.5                5.0               1.9
147                6.5               3.0                5.2               2.0
148                6.2               3.4                5.4               2.3
149                5.9               3.0                5.1               1.8

     species
145        2
146        2
147        2
148        2
149        2
```

[4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   sepal length (cm)  150 non-null    float64
 1   sepal width (cm)   150 non-null    float64
 2   petal length (cm)  150 non-null    float64
 3   petal width (cm)   150 non-null    float64
 4   species            150 non-null    int64
dtypes: float64(4), int64(1)
memory usage: 6.0 KB
```

[5]:
```
df.describe()
```

[5]:
```
       sepal length (cm)  sepal width (cm)  petal length (cm)  \
count         150.000000        150.000000         150.000000
mean            5.843333          3.057333           3.758000
std             0.828066          0.435866           1.765298
min             4.300000          2.000000           1.000000
25%             5.100000          2.800000           1.600000
50%             5.800000          3.000000           4.350000
75%             6.400000          3.300000           5.100000
max             7.900000          4.400000           6.900000

       petal width (cm)     species
count        150.000000  150.000000
mean           1.199333    1.000000
std            0.762238    0.819232
min            0.100000    0.000000
25%            0.300000    0.000000
50%            1.300000    1.000000
75%            1.800000    2.000000
max            2.500000    2.000000
```

1.4 2. Statistical Summary of Data

[6]:
```
# Mean
df.mean(numeric_only=True)
```

[6]:
```
sepal length (cm)    5.843333
sepal width (cm)     3.057333
petal length (cm)    3.758000
petal width (cm)     1.199333
species              1.000000
dtype: float64
```

```
[7]: # Median
     df.median(numeric_only=True)
```

```
[7]: sepal length (cm)    5.80
     sepal width (cm)     3.00
     petal length (cm)    4.35
     petal width (cm)     1.30
     species              1.00
     dtype: float64
```

```
[8]: # Mode
     df.mode().iloc[0]
```

```
[8]: sepal length (cm)    5.0
     sepal width (cm)     3.0
     petal length (cm)    1.4
     petal width (cm)     0.2
     species              0.0
     Name: 0, dtype: float64
```

```
[9]: # Standard Deviation and Variance
     df.std(numeric_only=True), df.var(numeric_only=True)
```

```
[9]: (sepal length (cm)    0.828066
      sepal width (cm)     0.435866
      petal length (cm)    1.765298
      petal width (cm)     0.762238
      species              0.819232
      dtype: float64,
      sepal length (cm)    0.685694
      sepal width (cm)     0.189979
      petal length (cm)    3.116278
      petal width (cm)     0.581006
      species              0.671141
      dtype: float64)
```

```
[10]: # Correlation matrix
      df.corr(numeric_only=True)
```

```
[10]:                    sepal length (cm)  sepal width (cm)  petal length (cm)  \
      sepal length (cm)           1.000000         -0.117570           0.871754
      sepal width (cm)           -0.117570          1.000000          -0.428440
      petal length (cm)           0.871754         -0.428440           1.000000
      petal width (cm)            0.817941         -0.366126           0.962865
      species                     0.782561         -0.426658           0.949035

                         petal width (cm)   species
```

```
sepal length (cm)          0.817941  0.782561
sepal width (cm)          -0.366126 -0.426658
petal length (cm)          0.962865  0.949035
petal width (cm)           1.000000  0.956547
species                    0.956547  1.000000
```

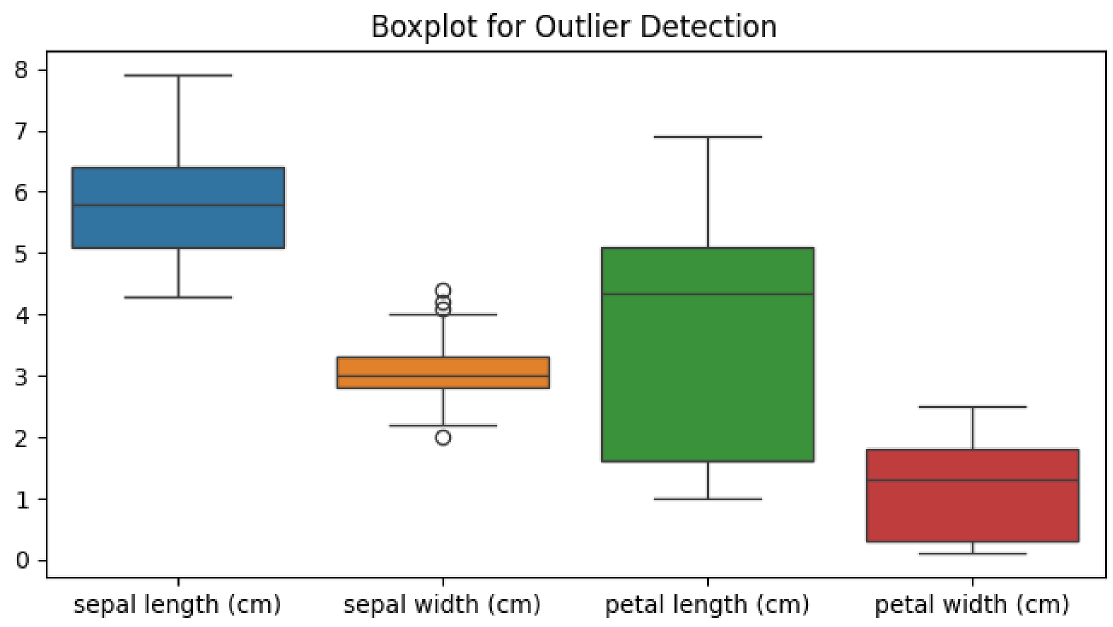
1.5 3. Handling Missing Values and Outliers

[11]:
```python
# Check missing values
df.isnull().sum()
```

[11]:
```
sepal length (cm)    0
sepal width (cm)     0
petal length (cm)    0
petal width (cm)     0
species              0
dtype: int64
```

1.5.1 Outlier Detection using Boxplot

[12]:
```python
plt.figure(figsize=(8,4))
sns.boxplot(data=df.iloc[:, :-1])
plt.title("Boxplot for Outlier Detection")
plt.show()
```



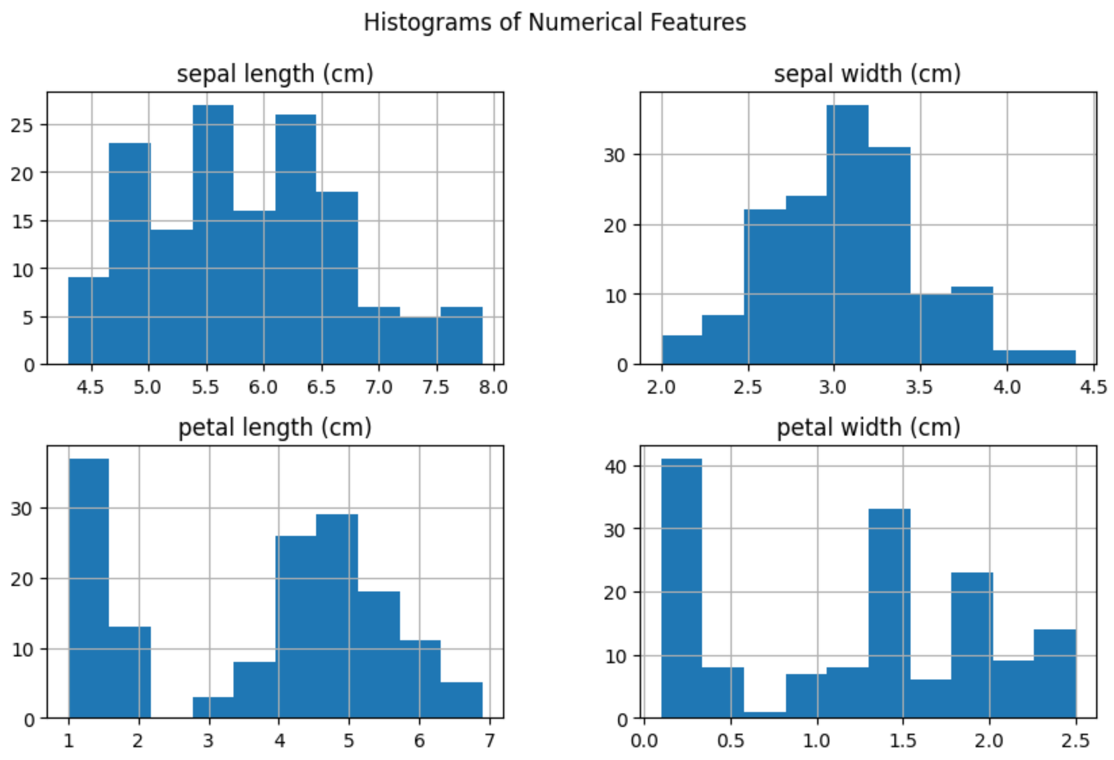
1.5.2 Handling Outliers (IQR Method)

[13]:
```
Q1 = df.iloc[:, :-1].quantile(0.25)
Q3 = df.iloc[:, :-1].quantile(0.75)
IQR = Q3 - Q1

df_no_outliers = df[~((df.iloc[:, :-1] < (Q1 - 1.5 * IQR)) |
                      (df.iloc[:, :-1] > (Q3 + 1.5 * IQR))).any(axis=1)]
df_no_outliers
```

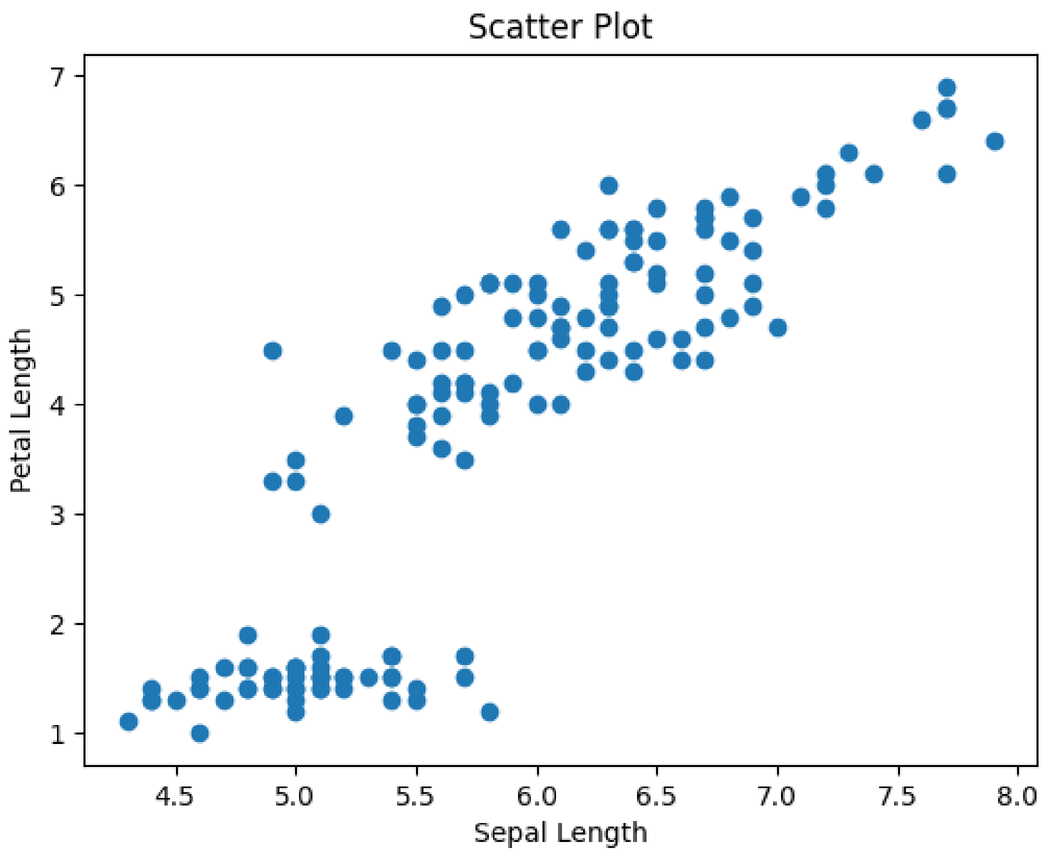
[13]:
```
     sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                  5.1               3.5                1.4               0.2
1                  4.9               3.0                1.4               0.2
2                  4.7               3.2                1.3               0.2
3                  4.6               3.1                1.5               0.2
4                  5.0               3.6                1.4               0.2
..                 …                …                 …                …
145                6.7               3.0                5.2               2.3
146                6.3               2.5                5.0               1.9
147                6.5               3.0                5.2               2.0
148                6.2               3.4                5.4               2.3
149                5.9               3.0                5.1               1.8

     species
0          0
1          0
2          0
3          0
4          0
..       …
145        2
146        2
147        2
148        2
149        2

[146 rows x 5 columns]
```

1.6 4. Data Visualization

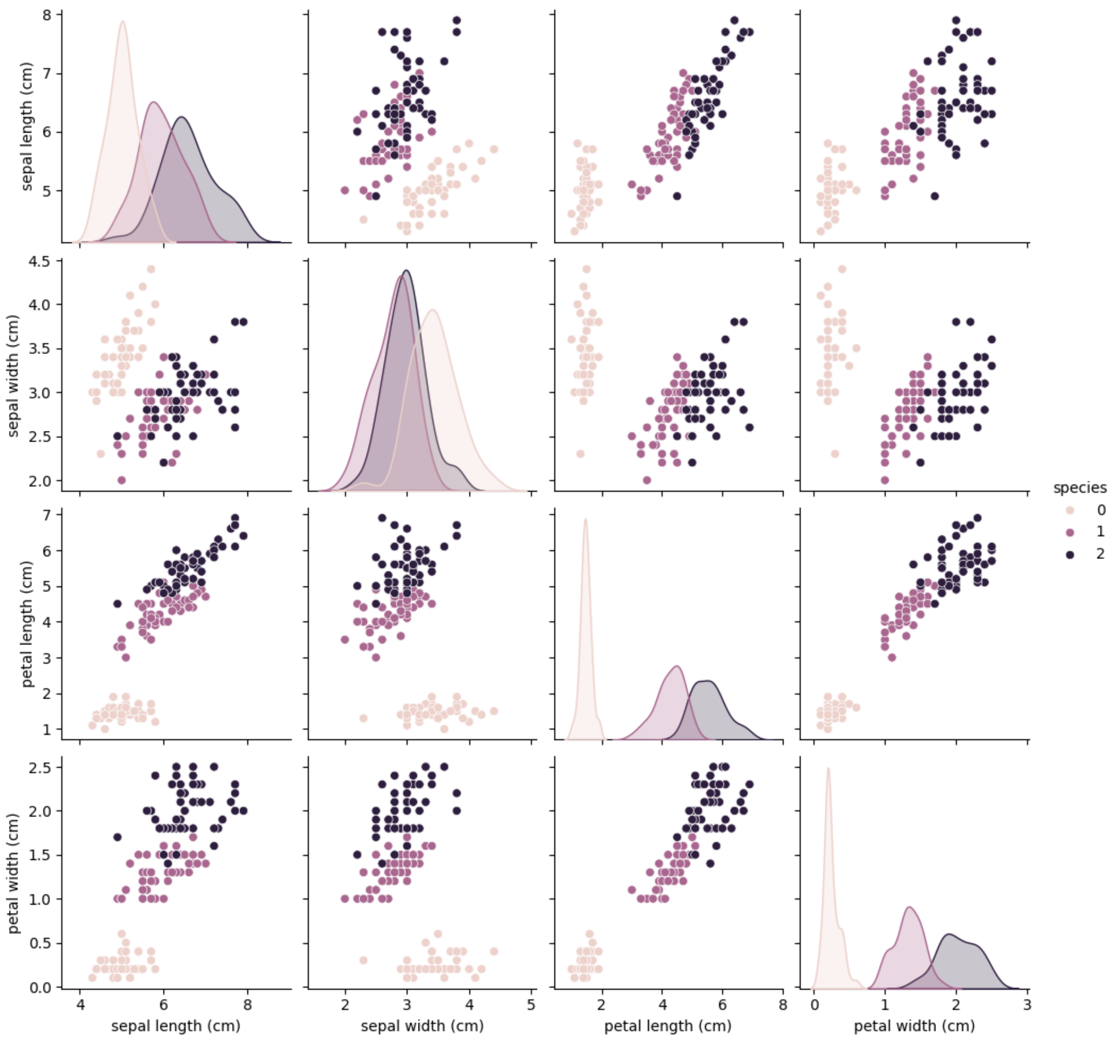
[14]:
```
# Histograms
df.iloc[:, :-1].hist(figsize=(10,6))
plt.suptitle("Histograms of Numerical Features")
plt.show()
```



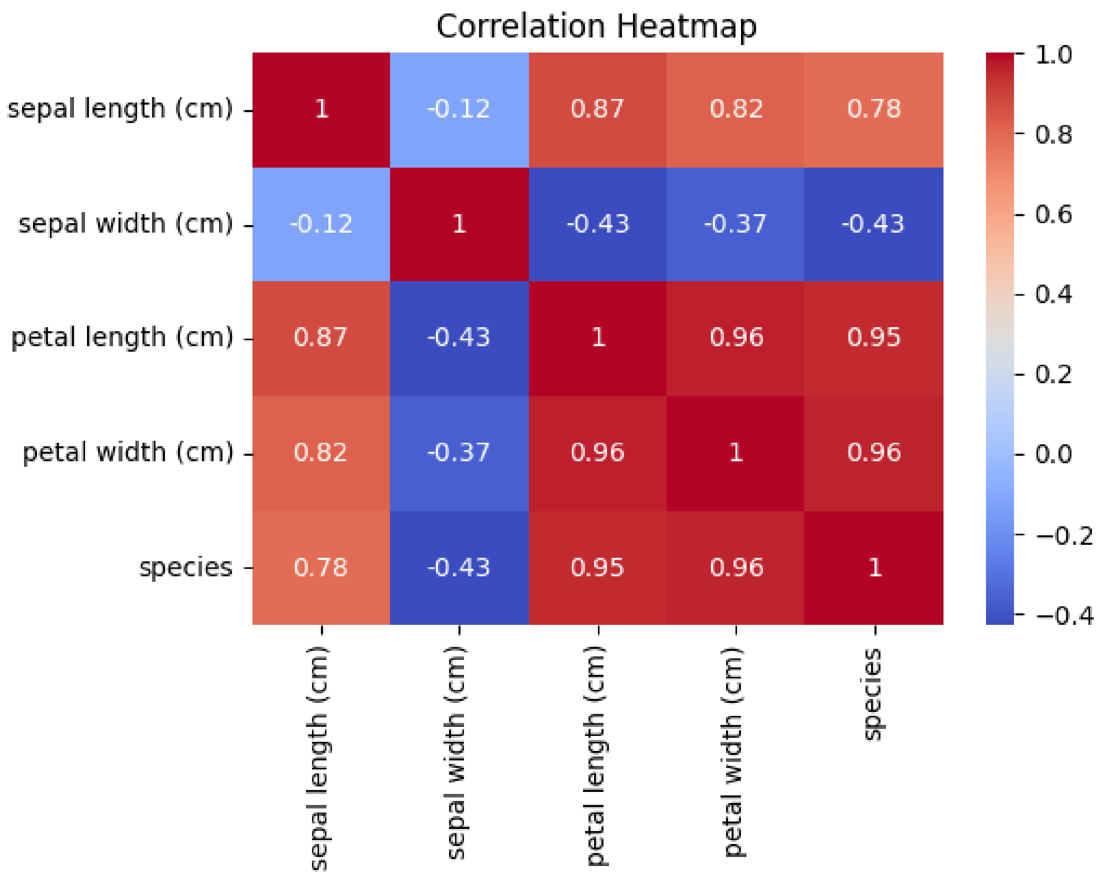
```
[15]: # Scatter plot
plt.scatter(df['sepal length (cm)'], df['petal length (cm)'])
plt.xlabel("Sepal Length")
plt.ylabel("Petal Length")
plt.title("Scatter Plot")
plt.show()
```



```
[16]: # Pairplot
      sns.pairplot(df, hue='species')
      plt.show()
```



```
[17]: # Correlation heatmap
      plt.figure(figsize=(6,4))
      sns.heatmap(df.corr(numeric_only=True), annot=True, cmap='coolwarm')
      plt.title("Correlation Heatmap")
      plt.show()
```

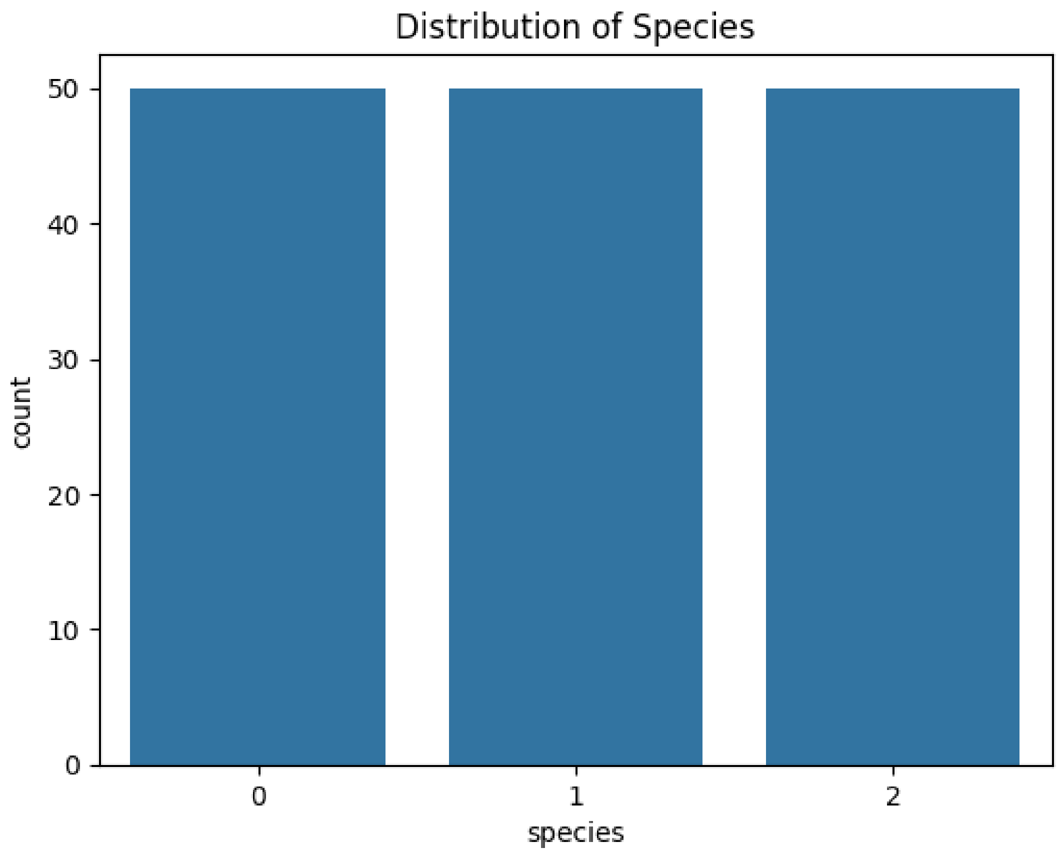


1.7 5. Feature Analysis

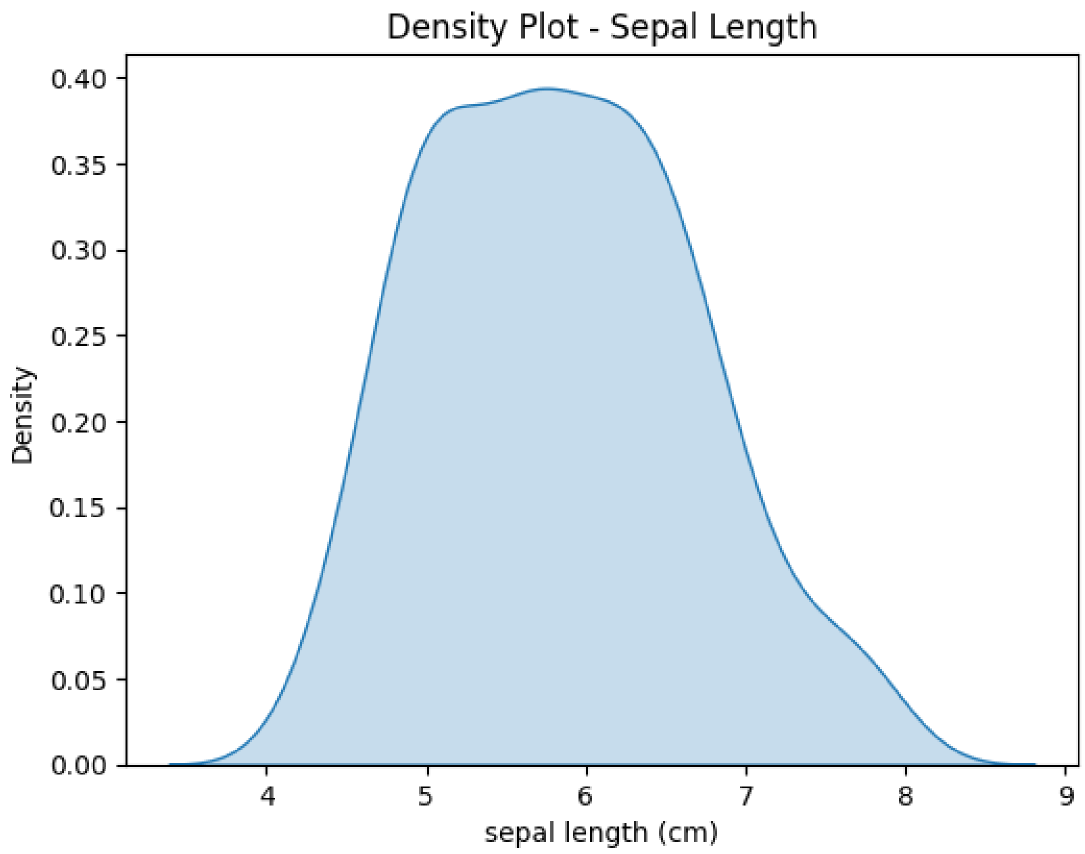
[18]:
```
# Categorical feature analysis
df['species'].value_counts()
```

[18]:
```
species
0    50
1    50
2    50
Name: count, dtype: int64
```

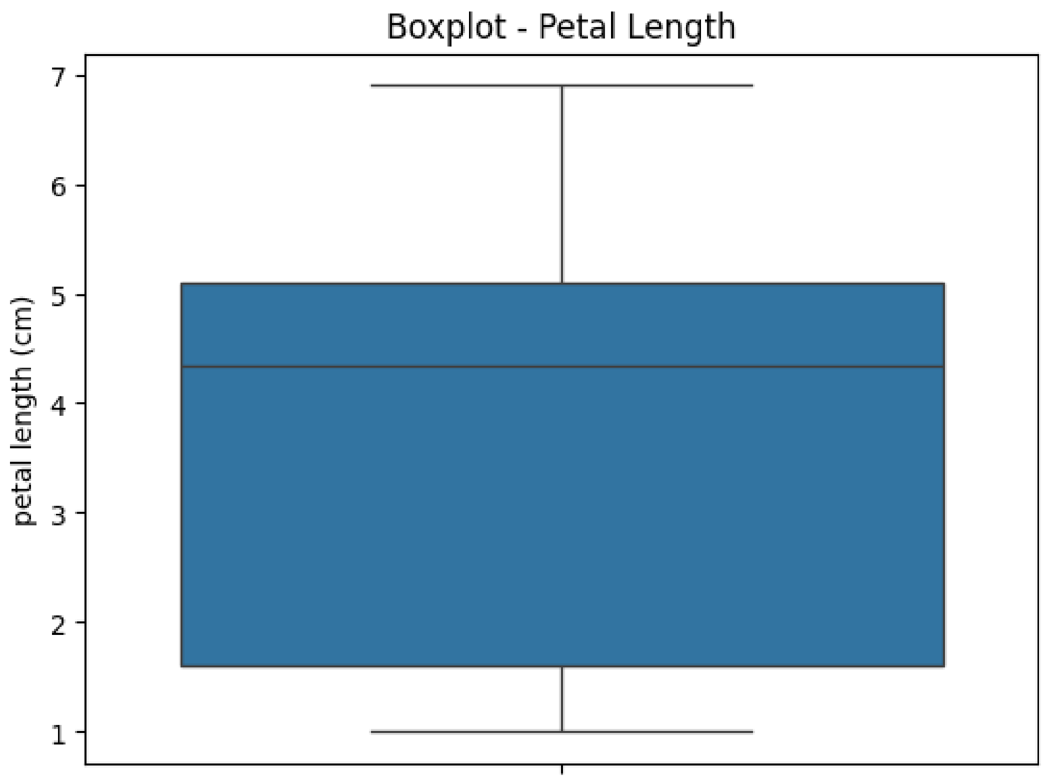
[19]:
```
# Bar plot for categorical feature
sns.countplot(x='species', data=df)
plt.title("Distribution of Species")
plt.show()
```



[20]:
```
# Density plot
sns.kdeplot(df['sepal length (cm)'], fill=True)
plt.title("Density Plot - Sepal Length")
plt.show()
```



[21]:
```
# Boxplot for numerical feature
sns.boxplot(y=df['petal length (cm)'])
plt.title("Boxplot - Petal Length")
plt.show()
```



1.8 Result

Exploratory Data Analysis was successfully performed. Statistical summaries, visualizations, and feature analysis helped in understanding data distribution, relationships, and potential anomalies.

Lab – 04: Linear Regression in Machine Learning

Aim: To implement and evaluate simple and multiple linear regression models using Python.

1.2 1. Simple Linear Regression

Simple Linear Regression models the relationship between one independent variable and one dependent variable. In this experiment, we use a Salary vs Experience dataset.

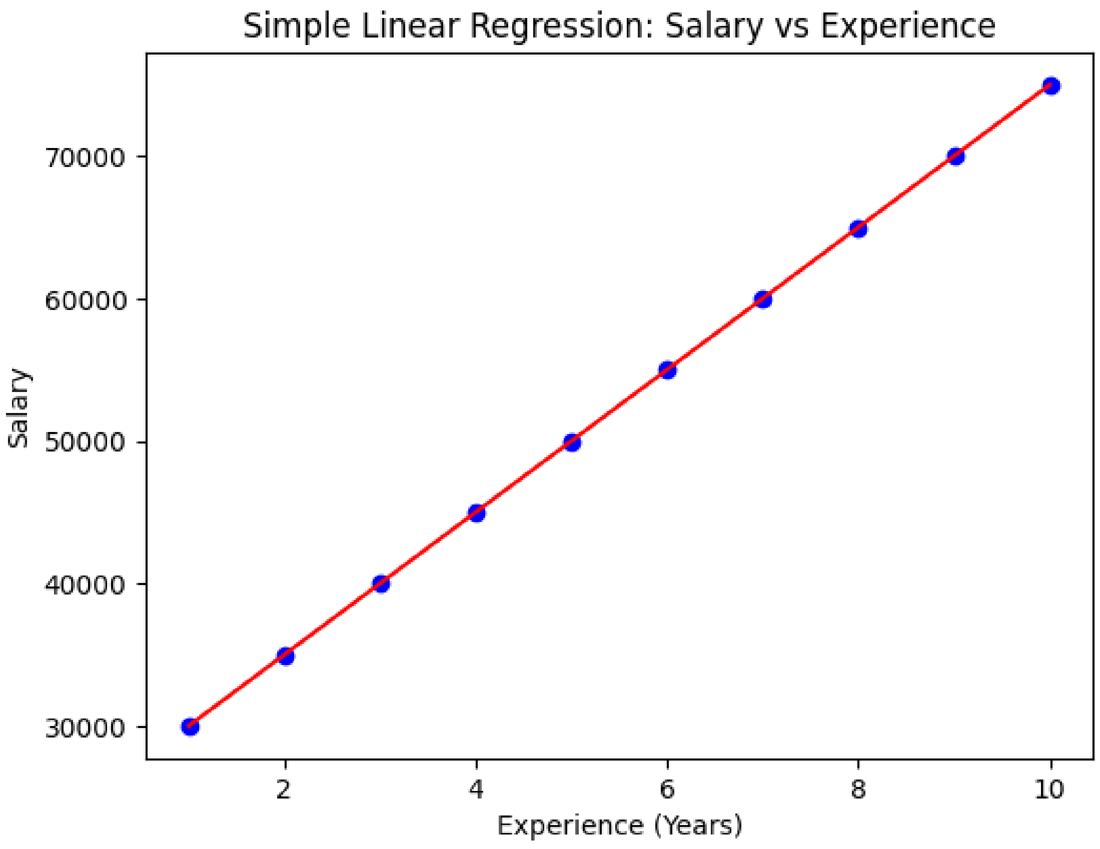
[2]:
```python
#import libraries

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

[3]:
```python
# Salary vs Experience dataset
experience = np.array([1,2,3,4,5,6,7,8,9,10]).reshape(-1,1)
salary = np.array([30000,35000,40000,45000,50000,55000,60000,65000,70000,75000])

print(experience)
print(salary)
```

```
[[ 1]
 [ 2]
 [ 3]
 [ 4]
 [ 5]
 [ 6]
 [ 7]
 [ 8]
 [ 9]
 [10]]
[30000 35000 40000 45000 50000 55000 60000 65000 70000 75000]
```

[4]:
```python
# Create and train model
slr = LinearRegression()
slr.fit(experience, salary)
```

```
# Predictions
salary_pred = slr.predict(experience)

# Visualization
plt.scatter(experience, salary, color='blue')
plt.plot(experience, salary_pred, color='red')
plt.xlabel("Experience (Years)")
plt.ylabel("Salary")
plt.title("Simple Linear Regression: Salary vs Experience")
plt.show()
```



1.3 2. Multiple Linear Regression

Multiple Linear Regression uses more than one independent variable to predict a target. We use the California Housing dataset which contains multiple features.

[10]:
```
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split

# Load dataset
```

```
housing = fetch_california_housing()
X = housing.data
y = housing.target

print(X)
print(y)
```

```
[[   8.3252       41.              6.98412698 …    2.55555556
    37.88       -122.23      ]
 [   8.3014       21.              6.23813708 …    2.10984183
    37.86       -122.22      ]
 [   7.2574       52.              8.28813559 …    2.80225989
    37.85       -122.24      ]
 …
 [   1.7          17.              5.20554273 …    2.3256351
    39.43       -121.22      ]
 [   1.8672       18.              5.32951289 …    2.12320917
    39.43       -121.32      ]
 [   2.3886       16.              5.25471698 …    2.61698113
    39.37       -121.24      ]]
[4.526 3.585 3.521 … 0.923 0.847 0.894]
```

[11]:
```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)

# Train model
mlr = LinearRegression()
mlr.fit(X_train, y_train)

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

1.4 3. Evaluating Model Performance

Regression performance is evaluated using error-based and goodness-of-fit metrics.

[6]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

print("Mean Absolute Error (MAE):", mean_absolute_error(y_test, y_pred))
print("Mean Squared Error (MSE):", mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error (RMSE):", np.sqrt(mean_squared_error(y_test,␣
  ↪y_pred)))
print("R2 Score:", r2_score(y_test, y_pred))
```

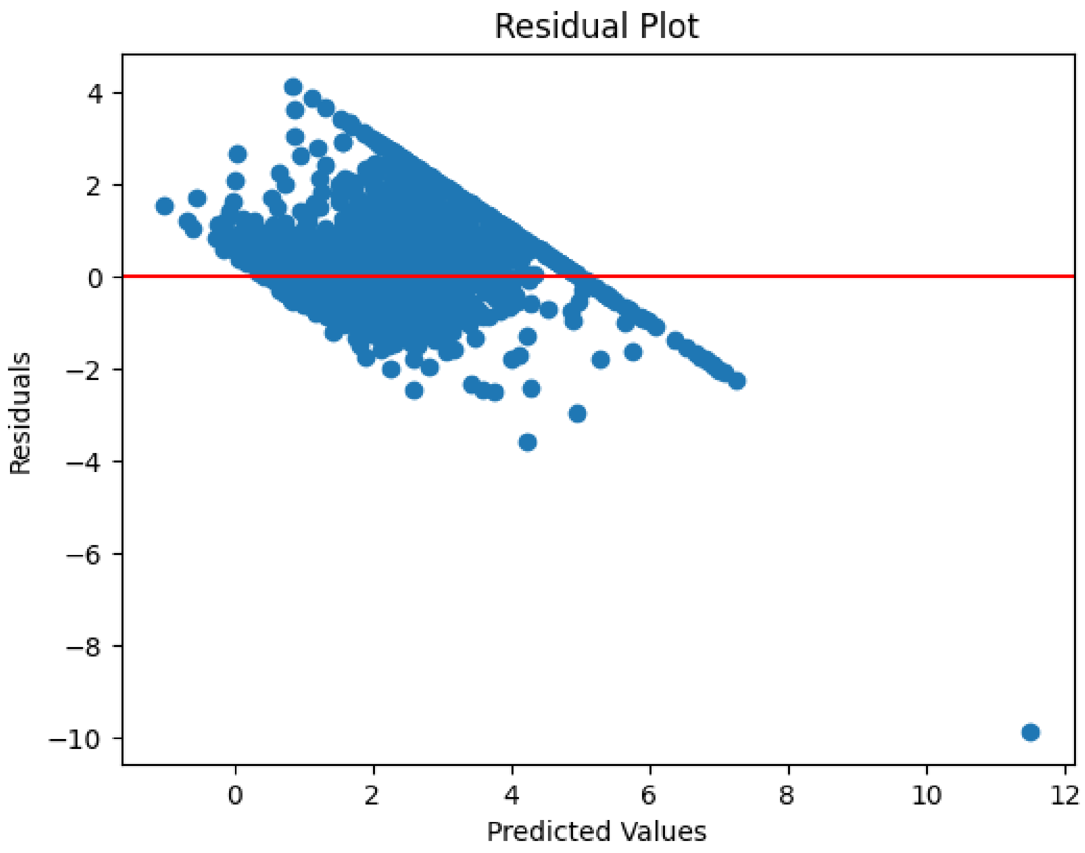
```
Mean Absolute Error (MAE): 0.5332001304956698
Mean Squared Error (MSE): 0.5558915986952422
```

```
Root Mean Squared Error (RMSE): 0.7455813830127749
R2 Score: 0.5757877060324524
```

1.5 4. Visualizing Residuals

Residuals are the difference between actual and predicted values. A random pattern around zero indicates a good fit.

[7]:
```
residuals = y_test - y_pred

plt.scatter(y_pred, residuals)
plt.axhline(y=0, color='red')
plt.xlabel("Predicted Values")
plt.ylabel("Residuals")
plt.title("Residual Plot")
plt.show()
```



1.6 5. Predicting New Data

The trained model is used to predict outcomes for new input data and compare predicted and actual values.

[8]:
```python
# Predict for new samples
new_samples = X_test[:5]

print("Predicted Values:", mlr.predict(new_samples))
print("Actual Values:", y_test[:5])
```

```
Predicted Values: [0.71912284 1.76401657 2.70965883 2.83892593 2.60465725]
Actual Values: [0.477   0.458   5.00001 2.186   2.78   ]
```

1.7 Result

Simple and multiple linear regression models were successfully implemented. Model performance was evaluated using standard regression metrics, and residual analysis confirmed the validity of model assumptions.

Lab – 05: Logistic Regression in Machine Learning

Aim: To implement and evaluate logistic regression models for classification tasks using Python.

1.2 1. Binary Classification using Logistic Regression

In this experiment, logistic regression is applied to a binary classification problem. The Iris dataset is converted into a binary dataset by selecting two classes.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

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

print(X)
print(y)
```

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
```

```
[5.8 4.  1.2 0.2]
[5.7 4.4 1.5 0.4]
[5.4 3.9 1.3 0.4]
[5.1 3.5 1.4 0.3]
[5.7 3.8 1.7 0.3]
[5.1 3.8 1.5 0.3]
[5.4 3.4 1.7 0.2]
[5.1 3.7 1.5 0.4]
[4.6 3.6 1.  0.2]
[5.1 3.3 1.7 0.5]
[4.8 3.4 1.9 0.2]
[5.  3.  1.6 0.2]
[5.  3.4 1.6 0.4]
[5.2 3.5 1.5 0.2]
[5.2 3.4 1.4 0.2]
[4.7 3.2 1.6 0.2]
[4.8 3.1 1.6 0.2]
[5.4 3.4 1.5 0.4]
[5.2 4.1 1.5 0.1]
[5.5 4.2 1.4 0.2]
[4.9 3.1 1.5 0.2]
[5.  3.2 1.2 0.2]
[5.5 3.5 1.3 0.2]
[4.9 3.6 1.4 0.1]
[4.4 3.  1.3 0.2]
[5.1 3.4 1.5 0.2]
[5.  3.5 1.3 0.3]
[4.5 2.3 1.3 0.3]
[4.4 3.2 1.3 0.2]
[5.  3.5 1.6 0.6]
[5.1 3.8 1.9 0.4]
[4.8 3.  1.4 0.3]
[5.1 3.8 1.6 0.2]
[4.6 3.2 1.4 0.2]
[5.3 3.7 1.5 0.2]
[5.  3.3 1.4 0.2]
[7.  3.2 4.7 1.4]
[6.4 3.2 4.5 1.5]
[6.9 3.1 4.9 1.5]
[5.5 2.3 4.  1.3]
[6.5 2.8 4.6 1.5]
[5.7 2.8 4.5 1.3]
[6.3 3.3 4.7 1.6]
[4.9 2.4 3.3 1. ]
[6.6 2.9 4.6 1.3]
[5.2 2.7 3.9 1.4]
[5.  2.  3.5 1. ]
[5.9 3.  4.2 1.5]
```

```
[6.  2.2 4.  1. ]
[6.1 2.9 4.7 1.4]
[5.6 2.9 3.6 1.3]
[6.7 3.1 4.4 1.4]
[5.6 3.  4.5 1.5]
[5.8 2.7 4.1 1. ]
[6.2 2.2 4.5 1.5]
[5.6 2.5 3.9 1.1]
[5.9 3.2 4.8 1.8]
[6.1 2.8 4.  1.3]
[6.3 2.5 4.9 1.5]
[6.1 2.8 4.7 1.2]
[6.4 2.9 4.3 1.3]
[6.6 3.  4.4 1.4]
[6.8 2.8 4.8 1.4]
[6.7 3.  5.  1.7]
[6.  2.9 4.5 1.5]
[5.7 2.6 3.5 1. ]
[5.5 2.4 3.8 1.1]
[5.5 2.4 3.7 1. ]
[5.8 2.7 3.9 1.2]
[6.  2.7 5.1 1.6]
[5.4 3.  4.5 1.5]
[6.  3.4 4.5 1.6]
[6.7 3.1 4.7 1.5]
[6.3 2.3 4.4 1.3]
[5.6 3.  4.1 1.3]
[5.5 2.5 4.  1.3]
[5.5 2.6 4.4 1.2]
[6.1 3.  4.6 1.4]
[5.8 2.6 4.  1.2]
[5.  2.3 3.3 1. ]
[5.6 2.7 4.2 1.3]
[5.7 3.  4.2 1.2]
[5.7 2.9 4.2 1.3]
[6.2 2.9 4.3 1.3]
[5.1 2.5 3.  1.1]
[5.7 2.8 4.1 1.3]
[6.3 3.3 6.  2.5]
[5.8 2.7 5.1 1.9]
[7.1 3.  5.9 2.1]
[6.3 2.9 5.6 1.8]
[6.5 3.  5.8 2.2]
[7.6 3.  6.6 2.1]
[4.9 2.5 4.5 1.7]
[7.3 2.9 6.3 1.8]
[6.7 2.5 5.8 1.8]
[7.2 3.6 6.1 2.5]
```

```
 [6.5 3.2 5.1 2. ]
 [6.4 2.7 5.3 1.9]
 [6.8 3.  5.5 2.1]
 [5.7 2.5 5.  2. ]
 [5.8 2.8 5.1 2.4]
 [6.4 3.2 5.3 2.3]
 [6.5 3.  5.5 1.8]
 [7.7 3.8 6.7 2.2]
 [7.7 2.6 6.9 2.3]
 [6.  2.2 5.  1.5]
 [6.9 3.2 5.7 2.3]
 [5.6 2.8 4.9 2. ]
 [7.7 2.8 6.7 2. ]
 [6.3 2.7 4.9 1.8]
 [6.7 3.3 5.7 2.1]
 [7.2 3.2 6.  1.8]
 [6.2 2.8 4.8 1.8]
 [6.1 3.  4.9 1.8]
 [6.4 2.8 5.6 2.1]
 [7.2 3.  5.8 1.6]
 [7.4 2.8 6.1 1.9]
 [7.9 3.8 6.4 2. ]
 [6.4 2.8 5.6 2.2]
 [6.3 2.8 5.1 1.5]
 [6.1 2.6 5.6 1.4]
 [7.7 3.  6.1 2.3]
 [6.3 3.4 5.6 2.4]
 [6.4 3.1 5.5 1.8]
 [6.  3.  4.8 1.8]
 [6.9 3.1 5.4 2.1]
 [6.7 3.1 5.6 2.4]
 [6.9 3.1 5.1 2.3]
 [5.8 2.7 5.1 1.9]
 [6.8 3.2 5.9 2.3]
 [6.7 3.3 5.7 2.5]
 [6.7 3.  5.2 2.3]
 [6.3 2.5 5.  1.9]
 [6.5 3.  5.2 2. ]
 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

[15]:
```
# Convert to binary classification (classes 0 and 1)
X_binary = X[y != 2]
y_binary = y[y != 2]
```

[7]:
```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X_binary, y_binary, test_size=0.2, random_state=42)
```

[8]:
```
# Train logistic regression model
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)
```

[8]:
```
LogisticRegression()
```

[9]:
```
# Predictions
y_pred = log_reg.predict(X_test)

# Accuracy
print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
Accuracy: 1.0
```

1.3 2. Confusion Matrix and Classification Report

The confusion matrix shows correct and incorrect predictions, while the classification report displays precision, recall, and F1-score.

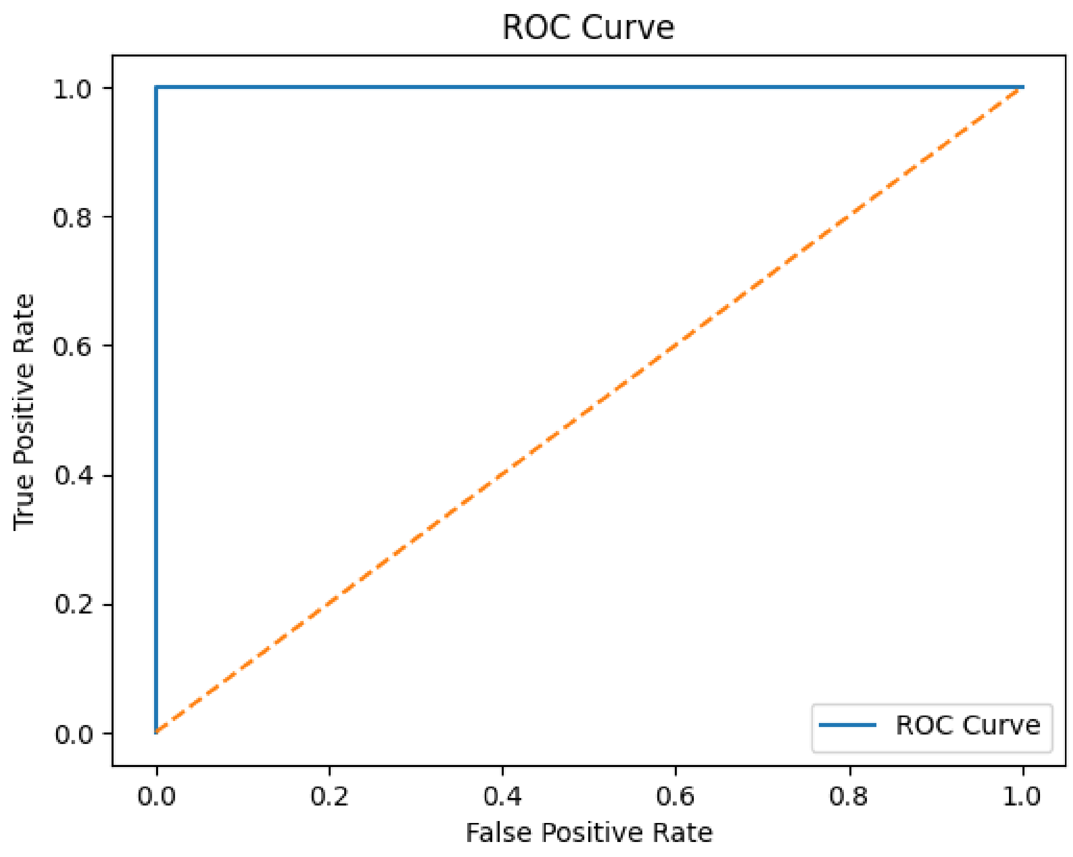
[10]:
```
from sklearn.metrics import confusion_matrix, classification_report

print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Confusion Matrix:
 [[12  0]
 [ 0  8]]

Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        12
           1       1.00      1.00      1.00         8

    accuracy                           1.00        20
   macro avg       1.00      1.00      1.00        20
weighted avg       1.00      1.00      1.00        20
```

1.4 3. ROC Curve and AUC Score

The ROC curve plots the True Positive Rate against the False Positive Rate. AUC measures the overall performance of the classifier.

[11]:
```
from sklearn.metrics import roc_curve, roc_auc_score

# Probability estimates
y_prob = log_reg.predict_proba(X_test)[:, 1]

# ROC curve
fpr, tpr, _ = roc_curve(y_test, y_prob)

plt.plot(fpr, tpr, label="ROC Curve")
plt.plot([0,1], [0,1], linestyle='--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()

# AUC score
print("AUC Score:", roc_auc_score(y_test, y_prob))
```



```
AUC Score: 1.0
```

1.5 4. Multiclass Classification using Logistic Regression

Logistic regression can also be applied to multiclass classification problems using One-vs-Rest strategy.

[13]:
```
# Multiclass classification
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)

multi_log = LogisticRegression(max_iter=200)
multi_log.fit(X_train, y_train)

y_multi_pred = multi_log.predict(X_test)

print("Multiclass Accuracy:", accuracy_score(y_test, y_multi_pred))
```

```
Multiclass Accuracy: 1.0
```

1.6 5. Predicting New Data

The trained logistic regression model is used to predict the class of new input samples.

[14]:
```
# New sample prediction
new_sample = [[5.1, 3.5, 1.4, 0.2]]
predicted_class = multi_log.predict(new_sample)

print("Predicted Class:", predicted_class)
```

```
Predicted Class: [0]
```

1.7 Result

Logistic regression models were successfully implemented for binary and multiclass classification. Model performance was evaluated using accuracy, confusion matrix, classification report, ROC curve, and AUC score.

Lab – 06: Decision Trees in Machine Learning

1.1 Aim: To implement and evaluate decision tree models for classification tasks using Python.

1.2 Dataset Used

Iris dataset from sklearn is used for decision tree classification.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score
```

1.3 1. Building a Simple Decision Tree

A decision tree classifier is trained using the Iris dataset.

[2]:
```
# Load dataset
iris = load_iris()
X = iris.data
y = iris.target
```

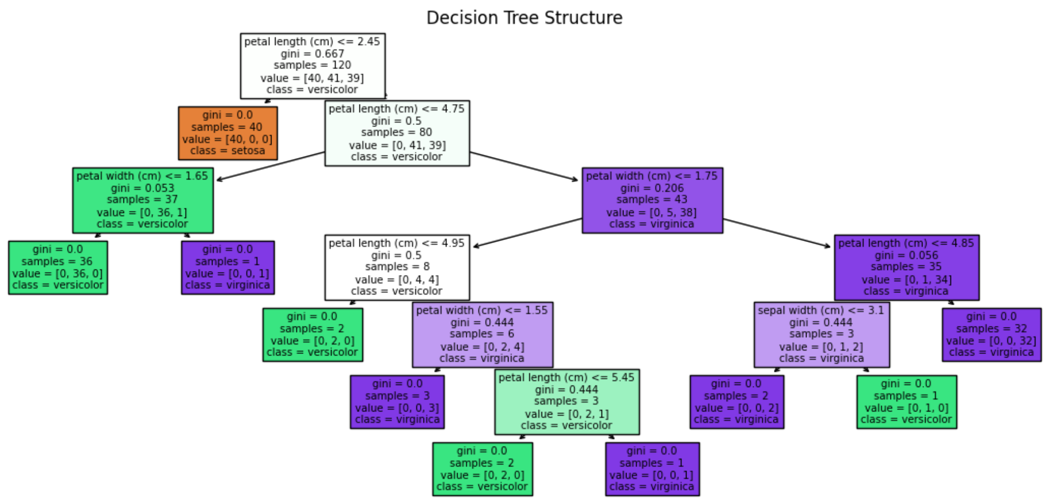
[3]:
```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
```

[4]:
```
# Train decision tree
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)
```

[4]:
```
DecisionTreeClassifier(random_state=42)
```

[5]:
```
# Visualize tree
plt.figure(figsize=(14,6))
plot_tree(dt, feature_names=iris.feature_names,
```

```python
            class_names=iris.target_names, filled=True)
plt.title("Decision Tree Structure")
plt.show()
```



1.4 2. Evaluating Decision Tree Performance

Model performance is evaluated using accuracy, precision, recall, and F1-score.

[6]:
```python
# Predictions
y_pred = dt.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred, average='macro'))
print("Recall:", recall_score(y_test, y_pred, average='macro'))
print("F1 Score:", f1_score(y_test, y_pred, average='macro'))
```

```
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1 Score: 1.0
```

1.5 3. Using Different Splitting Criteria

Decision trees are trained using Gini and Entropy criteria.

[7]:
```python
dt_gini = DecisionTreeClassifier(criterion='gini', random_state=42)
dt_entropy = DecisionTreeClassifier(criterion='entropy', random_state=42)

dt_gini.fit(X_train, y_train)
dt_entropy.fit(X_train, y_train)
```

```
print("Gini Accuracy:", accuracy_score(y_test, dt_gini.predict(X_test)))
print("Entropy Accuracy:", accuracy_score(y_test, dt_entropy.predict(X_test)))
```

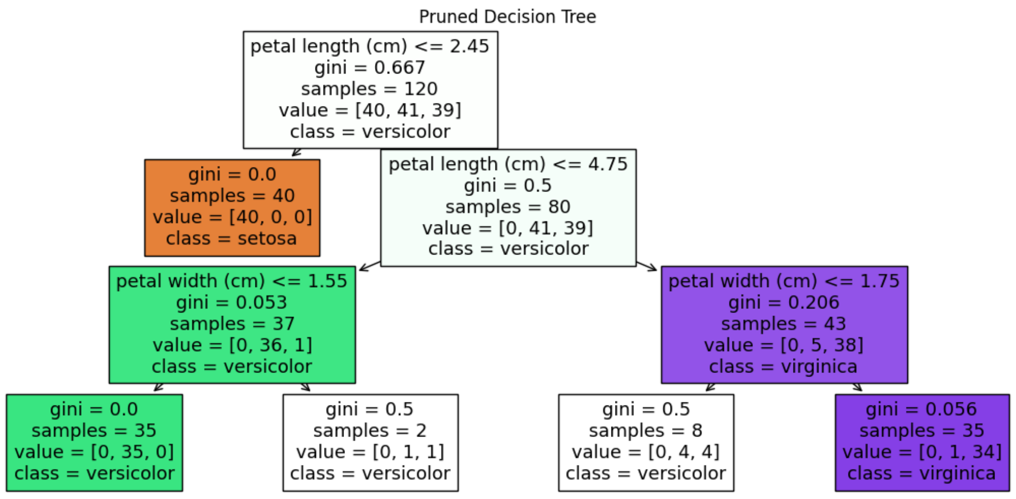
```
Gini Accuracy: 1.0
Entropy Accuracy: 1.0
```

1.6 4. Pruning and Controlling Overfitting

Tree complexity is controlled using pruning parameters.

[8]:
```
dt_pruned = DecisionTreeClassifier(
    max_depth=3,
    min_samples_split=5,
    min_samples_leaf=2,
    random_state=42)

dt_pruned.fit(X_train, y_train)

print("Pruned Tree Accuracy:", accuracy_score(y_test, dt_pruned.
  ↪predict(X_test)))

plt.figure(figsize=(14,6))
plot_tree(dt_pruned, feature_names=iris.feature_names,
          class_names=iris.target_names, filled=True)
plt.title("Pruned Decision Tree")
plt.show()
```

```
Pruned Tree Accuracy: 1.0
```



1.7 5. Predicting New Data

The trained decision tree is used to predict the class of new input samples.

[9]:
```
new_sample = [[5.1, 3.5, 1.4, 0.2]]
prediction = dt_pruned.predict(new_sample)

print("Predicted Class:", iris.target_names[prediction[0]])
```

```
Predicted Class: setosa
```

1.8 Result

Decision tree models were successfully implemented and evaluated. Pruning helped reduce overfitting while maintaining good accuracy.

Lab – 07: Random Forests and Ensemble Methods in Machine Learning

1.1 Aim: To implement and evaluate ensemble learning techniques using Random Forests for classification and regression.

1.2 1. Random Forest Classifier

[3]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris, fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, mean_squared_error, r2_score
import pandas as pd

# Classification dataset
iris = load_iris()
X = iris.data
y = iris.target
```

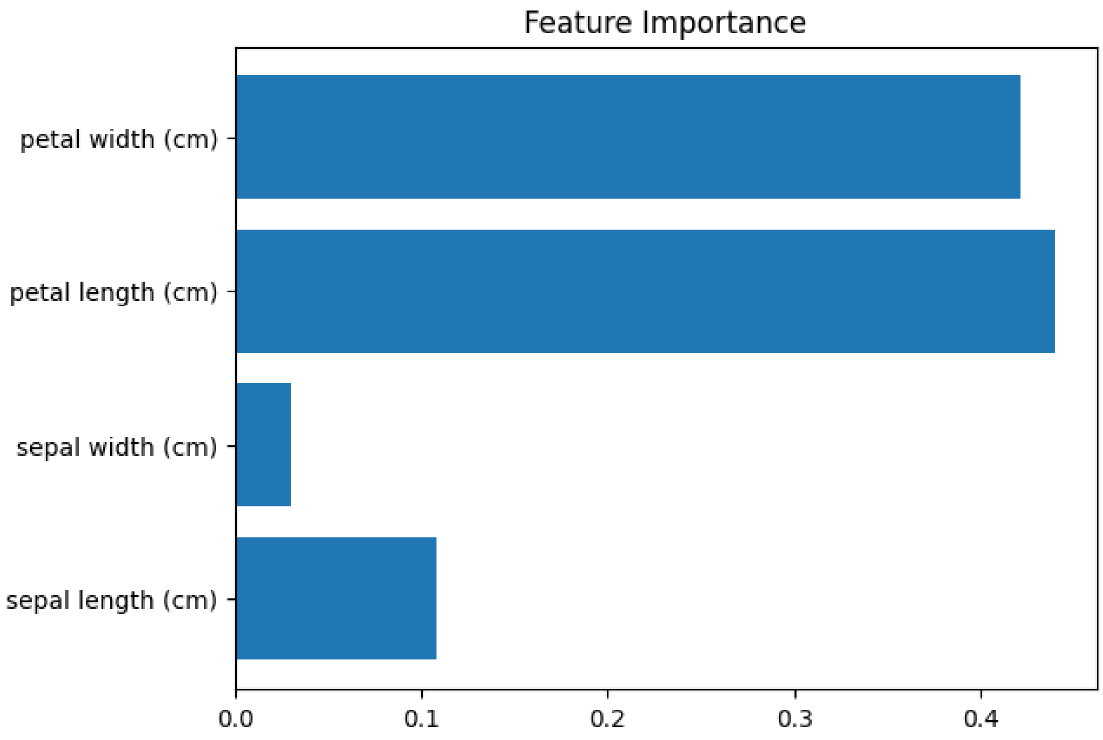
[4]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)

rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

rf_pred = rf.predict(X_test)
print("Random Forest Accuracy:", accuracy_score(y_test, rf_pred))
```

```
Random Forest Accuracy: 1.0
```

1.3 2. Feature Importance Analysis

[5]:
```
importances = rf.feature_importances_
for f, i in zip(iris.feature_names, importances):
    print(f, ":", i)

plt.barh(iris.feature_names, importances)
plt.title("Feature Importance")
plt.show()
```

```
sepal length (cm) : 0.10809762464246378
sepal width (cm) : 0.030386812473242528
petal length (cm) : 0.43999397414456937
petal width (cm) : 0.4215215887397244
```



1.4 3. Hyperparameter Tuning

[6]:
```
rf_tuned = RandomForestClassifier(
    n_estimators=200, max_depth=5,
    min_samples_split=5, min_samples_leaf=3,
    random_state=42)

rf_tuned.fit(X_train, y_train)
```

```
print("Tuned RF Accuracy:", accuracy_score(y_test, rf_tuned.predict(X_test)))
```

```
Tuned RF Accuracy: 1.0
```

1.5 4. Random Forest Regression

[7]:
```
housing = fetch_california_housing()
Xh = housing.data
yh = housing.target

Xh_train, Xh_test, yh_train, yh_test = train_test_split(
    Xh, yh, test_size=0.2, random_state=42)

rf_reg = RandomForestRegressor(random_state=42)
rf_reg.fit(Xh_train, yh_train)

yh_pred = rf_reg.predict(Xh_test)

print("MSE:", mean_squared_error(yh_test, yh_pred))
print("RMSE:", np.sqrt(mean_squared_error(yh_test, yh_pred)))
print("R2 Score:", r2_score(yh_test, yh_pred))
```

```
MSE: 0.2553684927247781
RMSE: 0.5053399773665033
R2 Score: 0.8051230593157366
```

1.6 5. Comparison with Decision Tree

[8]:
```
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)

print("Decision Tree Accuracy:", accuracy_score(y_test, dt.predict(X_test)))
print("Random Forest Accuracy:", accuracy_score(y_test, rf_pred))
```

```
Decision Tree Accuracy: 1.0
Random Forest Accuracy: 1.0
```

Lab – 08: Support Vector Machine (SVM) in Machine Learning

2.1 Aim: To implement and evaluate SVM models for classification tasks.

2.2 1. Binary Classification using SVM

[16]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris, fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, mean_squared_error, r2_score
import pandas as pd


# dataset

iris = load_iris()
X = iris.data
y = iris.target
```

[17]:
```
from sklearn.svm import SVC

# Binary dataset
X_bin = X[y != 2]
y_bin = y[y != 2]
```

[18]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X_bin, y_bin, test_size=0.2, random_state=42)

svm_linear = SVC(kernel='linear')
svm_linear.fit(X_train, y_train)

print("Linear SVM Accuracy:", accuracy_score(y_test, svm_linear.
 ↪predict(X_test)))
```

```
Linear SVM Accuracy: 1.0
```

2.3 2. Different Kernels

[19]:
```
for kernel in ['linear', 'poly', 'rbf']:
    svm = SVC(kernel=kernel)
    svm.fit(X_train, y_train)
    print(kernel, "kernel accuracy:", accuracy_score(y_test, svm.
 ↪predict(X_test)))
```

```
linear kernel accuracy: 1.0
poly kernel accuracy: 1.0
rbf kernel accuracy: 1.0
```

2.4 3. Hyperparameter Tuning

[20]:
```
svm_tuned = SVC(kernel='rbf', C=10, gamma=0.1)
svm_tuned.fit(X_train, y_train)

print("Tuned SVM Accuracy:", accuracy_score(y_test, svm_tuned.predict(X_test)))
```

```
Tuned SVM Accuracy: 1.0
```

2.5 4. Multiclass Classification

[21]:
```
svm_multi = SVC(kernel='rbf', decision_function_shape='ovr')
svm_multi.fit(X, y)

print("Multiclass SVM Accuracy:", accuracy_score(y, svm_multi.predict(X)))
```

```
Multiclass SVM Accuracy: 0.9733333333333334
```

Lab – 09: K-Nearest Neighbors (KNN) in Machine Learning

3.1 Aim: To implement and evaluate KNN models for classification and regression.

3.2 1. KNN Classification

[24]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris, fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, mean_squared_error, r2_score
import pandas as pd

# dataset

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, random_state=42)
```

[25]:
```
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

print("KNN Accuracy:", accuracy_score(y_test, knn.predict(X_test)))
```

```
KNN Accuracy: 1.0
```

3.3 2. Choosing Different K Values

[26]:
```
for k in [3,5,7]:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    print("K =", k, "Accuracy:", accuracy_score(y_test, knn.predict(X_test)))
```

```
K = 3 Accuracy: 1.0
K = 5 Accuracy: 1.0
K = 7 Accuracy: 0.9666666666666667
```

3.4 3. Distance Metrics

[27]:
```
for m in ['euclidean', 'manhattan', 'minkowski']:
    knn = KNeighborsClassifier(metric=m)
    knn.fit(X_train, y_train)
    print(m, "Accuracy:", accuracy_score(y_test, knn.predict(X_test)))
```

```
euclidean Accuracy: 1.0
manhattan Accuracy: 1.0
minkowski Accuracy: 1.0
```

3.5 4. KNN Regression

[28]:
```
knn_reg = KNeighborsRegressor()
knn_reg.fit(Xh_train, yh_train)

yh_knn = knn_reg.predict(Xh_test)

print("RMSE:", np.sqrt(mean_squared_error(yh_test, yh_knn)))
print("R2:", r2_score(yh_test, yh_knn))
```

```
RMSE: 1.0576778270706204
R2: 0.14631049965900345
```

Lab – 10: Unsupervised Learning – Clustering in Machine Learning

4.1 Aim: To implement and evaluate clustering algorithms.

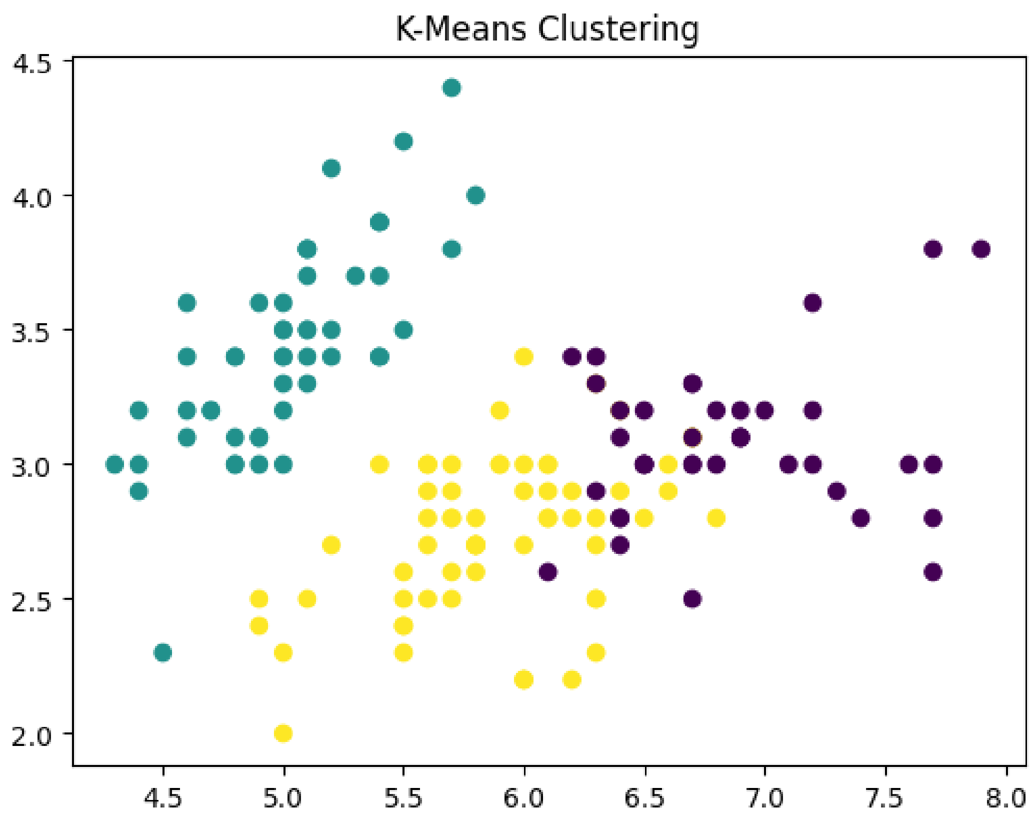
4.2 1. K-Means Clustering

[29]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris, fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.tree import DecisionTreeClassifier
```

```python
from sklearn.metrics import accuracy_score, mean_squared_error, r2_score
import pandas as pd

# dataset

iris = load_iris()
X = iris.data
y = iris.target
```

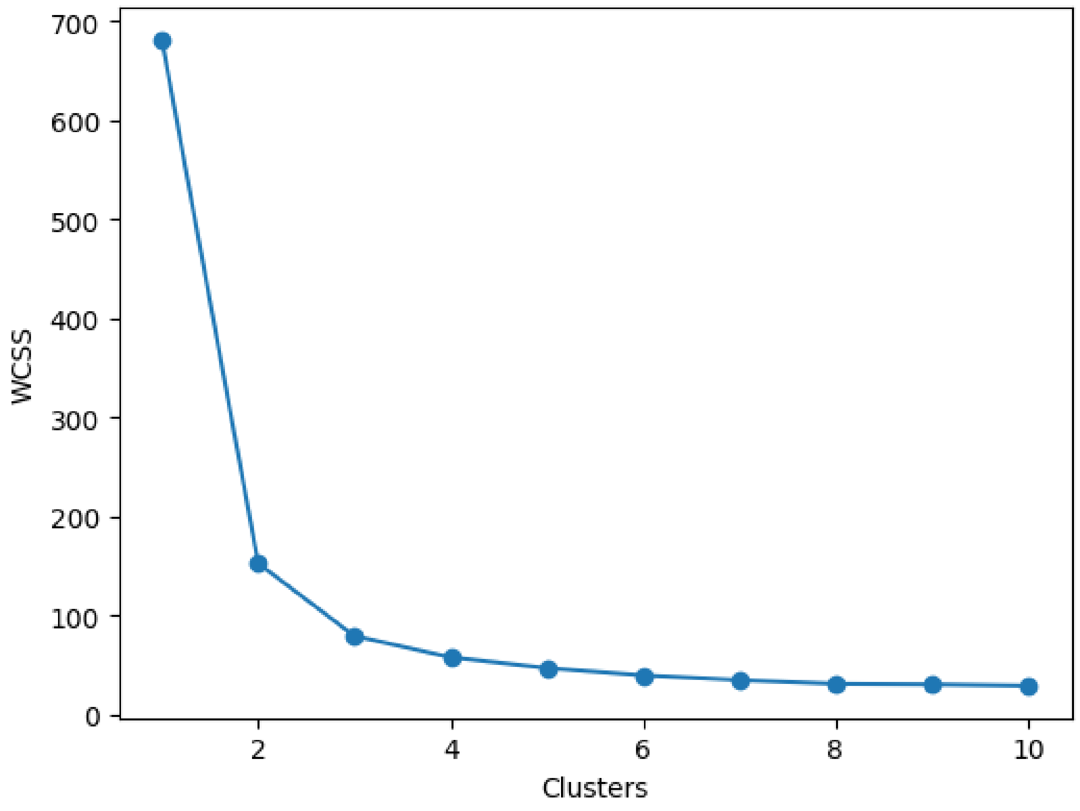
[30]:

```python
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, davies_bouldin_score

kmeans = KMeans(n_clusters=3, random_state=42)
labels = kmeans.fit_predict(X)

plt.scatter(X[:,0], X[:,1], c=labels)
plt.title("K-Means Clustering")
plt.show()
```



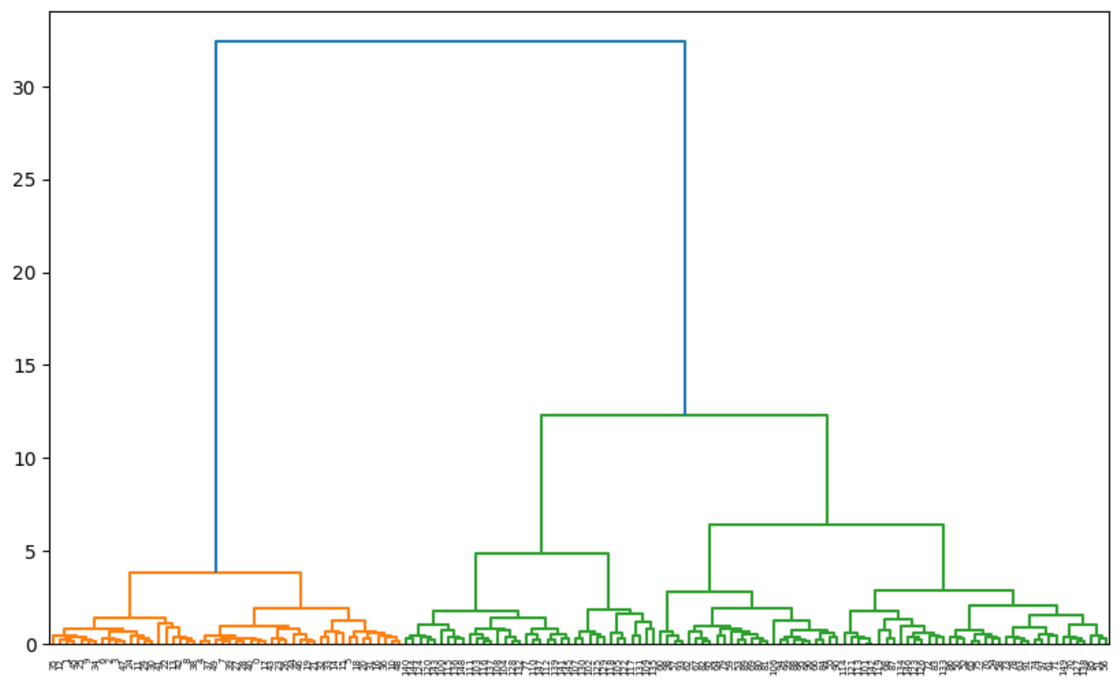
4.3 2. Elbow Method

[31]:
```
wcss = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

plt.plot(range(1,11), wcss, marker='o')
plt.xlabel("Clusters")
plt.ylabel("WCSS")
plt.show()
```



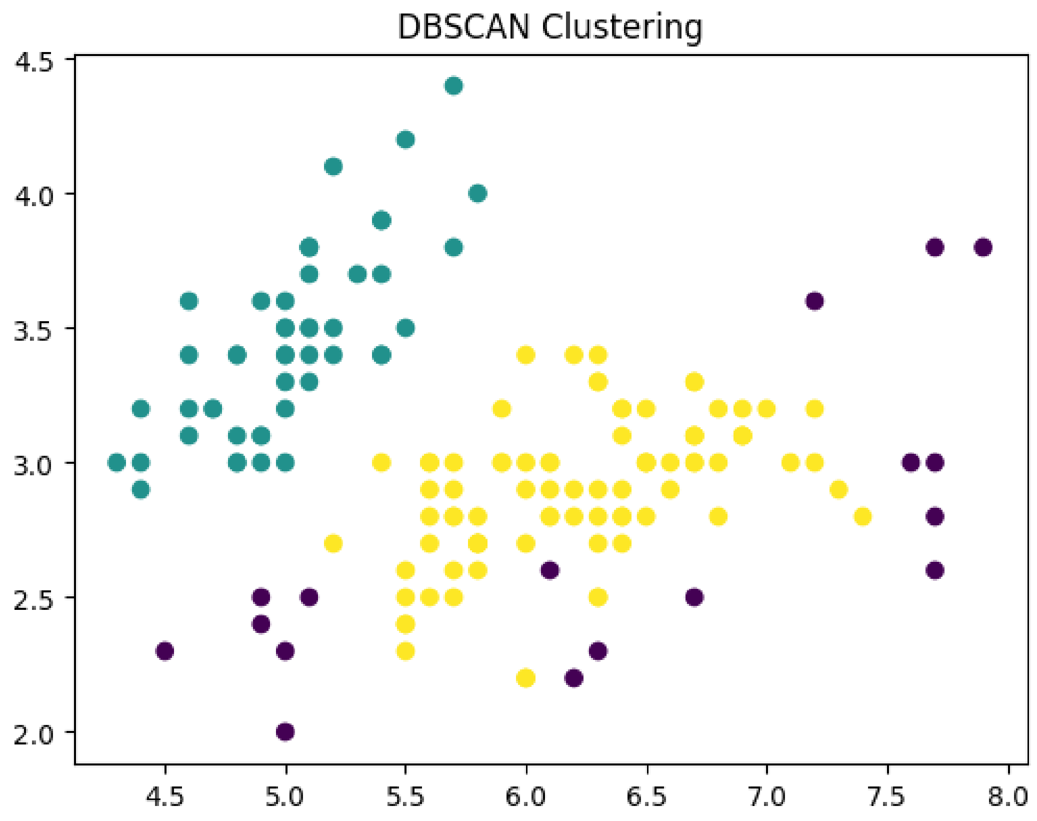
4.4 3. Hierarchical Clustering

[32]:
```
from scipy.cluster.hierarchy import dendrogram, linkage

linked = linkage(X, method='ward')
plt.figure(figsize=(10,6))
dendrogram(linked)
plt.show()
```



4.5 4. DBSCAN

[33]:
```
from sklearn.cluster import DBSCAN

db = DBSCAN(eps=0.5, min_samples=5)
db_labels = db.fit_predict(X)

plt.scatter(X[:,0], X[:,1], c=db_labels)
plt.title("DBSCAN Clustering")
plt.show()
```



4.6 5. Clustering Evaluation

[34]:
```
print("Silhouette:", silhouette_score(X, labels))
print("Davies-Bouldin:", davies_bouldin_score(X, labels))
```

```
Silhouette: 0.551191604619592
Davies-Bouldin: 0.6660385791628493
```

5 Lab – 11: Dimensionality Reduction in Machine Learning

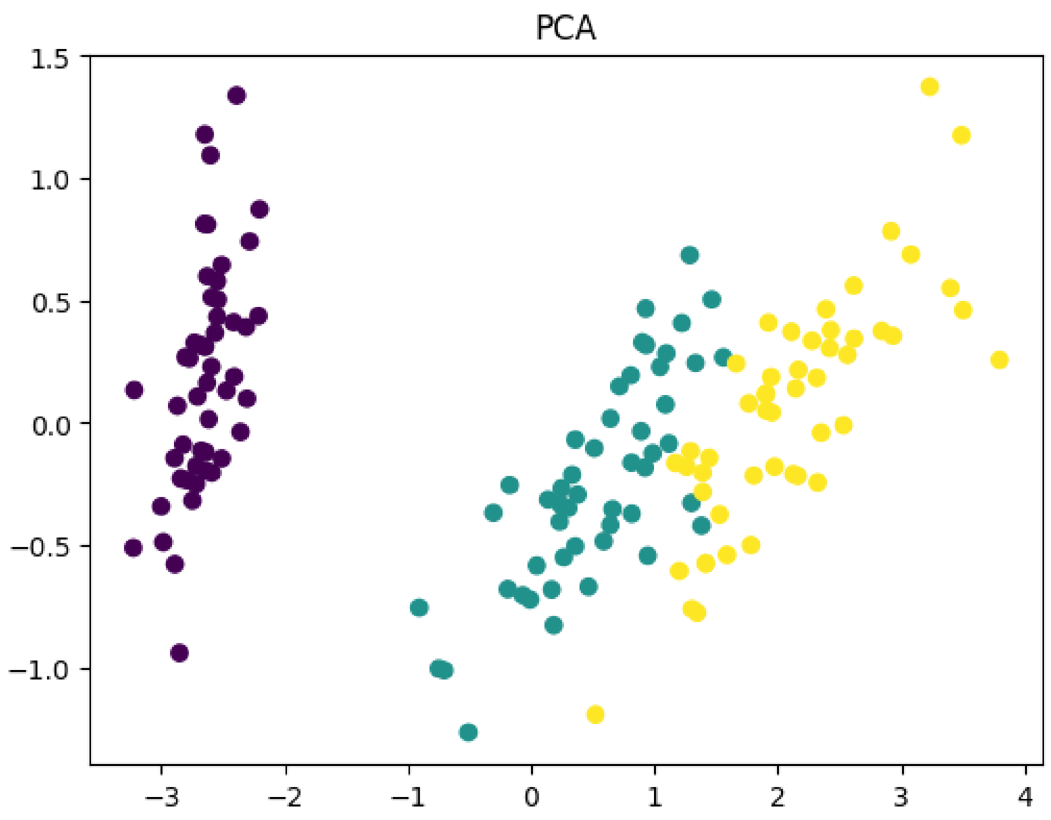
5.1 Aim: To reduce feature space while retaining important information.

5.2 PCA and t-SNE

[35]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris, fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, mean_squared_error, r2_score
```

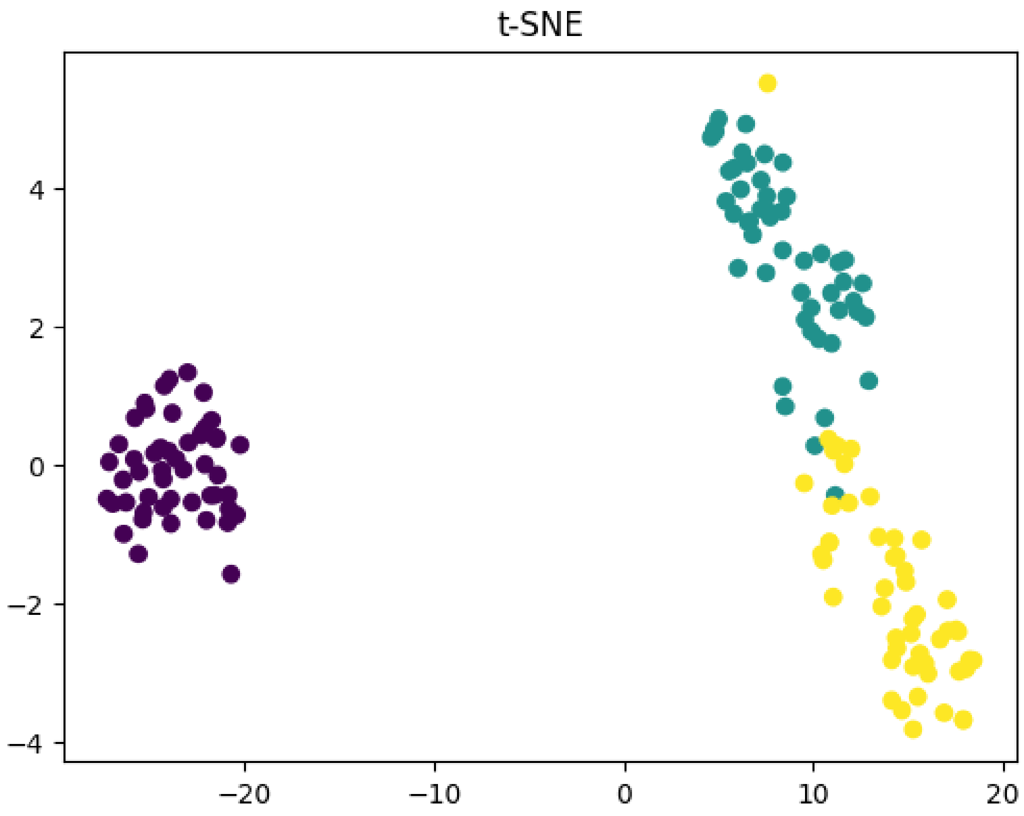
```
import pandas as pd

# dataset

iris = load_iris()
X = iris.data
y = iris.target
```

[36]:
```
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

plt.scatter(X_pca[:,0], X_pca[:,1], c=y)
plt.title("PCA")
plt.show()
```



[37]:
```
tsne = TSNE(n_components=2, random_state=42)
X_tsne = tsne.fit_transform(X)
```

```
plt.scatter(X_tsne[:,0], X_tsne[:,1], c=y)
plt.title("t-SNE")
plt.show()
```



Lab – 12: Model Evaluation and Cross-Validation

6.1 Aim: To evaluate models using validation techniques.

6.2 Cross-Validation and Grid Search

[39]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris, fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, mean_squared_error, r2_score
import pandas as pd

# dataset
```

```
iris = load_iris()
X = iris.data
y = iris.target
```

[40]:
```
from sklearn.model_selection import KFold, StratifiedKFold, cross_val_score,␣
 ↪GridSearchCV
from sklearn.linear_model import LogisticRegression
```

[41]:
```
log = LogisticRegression(max_iter=200)

kf = KFold(n_splits=5)
scores = cross_val_score(log, X, y, cv=kf)
print("K-Fold Accuracy:", scores.mean())

skf = StratifiedKFold(n_splits=5)
scores = cross_val_score(log, X, y, cv=skf)
print("Stratified K-Fold Accuracy:", scores.mean())

param_grid = {'C':[0.1,1,10]}
grid = GridSearchCV(log, param_grid, cv=5)
grid.fit(X, y)

print("Best Params:", grid.best_params_)
```

```
K-Fold Accuracy: 0.9266666666666665
Stratified K-Fold Accuracy: 0.9733333333333334
Best Params: {'C': 1}
```

[42]:
```
kf = KFold(n_splits=5)
scores = cross_val_score(log, X, y, cv=kf)
print("K-Fold Accuracy:", scores.mean())
```

```
K-Fold Accuracy: 0.9266666666666665
```

[43]:
```
skf = StratifiedKFold(n_splits=5)
scores = cross_val_score(log, X, y, cv=skf)
print("Stratified K-Fold Accuracy:", scores.mean())
```

```
Stratified K-Fold Accuracy: 0.9733333333333334
```

[44]:
```
param_grid = {'C':[0.1,1,10]}
grid = GridSearchCV(log, param_grid, cv=5)
grid.fit(X, y)
```

[44]:
```
GridSearchCV(cv=5, estimator=LogisticRegression(max_iter=200),
             param_grid={'C': [0.1, 1, 10]})
```

[45]:
```
print("Best Params:", grid.best_params_)
```

```
Best Params: {'C': 1}
```