data-analytics

April 12, 2025

Q1.Data Cleaning and Imputation Techniques:

Load a dataset with missing values. Apply techniques like mean/mode/median imputation and compare the results.

[9]:
```
!pip install pandas
```

```
Requirement already satisfied: pandas in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (2.2.3)
Requirement already satisfied: numpy>=1.26.0 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from pandas)
(2.2.4)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from pandas)
(2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from pandas)
(2025.2)
Requirement already satisfied: tzdata>=2022.7 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from pandas)
(2025.2)
Requirement already satisfied: six>=1.5 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from python-
dateutil>=2.8.2->pandas) (1.17.0)
```

[6]:
```
import pandas as pd
```

[7]:
```
print(pd)
```

```
<module 'pandas' from 'C:\\Users\\Shinde
Ankita\\AppData\\Local\\Programs\\Python\\Python313\\Lib\\site-
packages\\pandas\\__init__.py'>
```

[12]:
```
!pip install seaborn
```

```
Requirement already satisfied: seaborn in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (0.13.2)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from seaborn)
```

```
(2.2.4)
Requirement already satisfied: pandas>=1.2 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from seaborn)
(2.2.3)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from seaborn)
(3.10.1)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (1.3.1)
Requirement already satisfied: cycler>=0.10 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (4.57.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (1.4.8)
Requirement already satisfied: packaging>=20.0 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (24.2)
Requirement already satisfied: pillow>=8 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from
pandas>=1.2->seaborn) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from
pandas>=1.2->seaborn) (2025.2)
Requirement already satisfied: six>=1.5 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from python-
dateutil>=2.7->matplotlib!=3.6.1,>=3.4->seaborn) (1.17.0)
```

[14]:
```
import seaborn as sns
```

[15]:
```
df = sns.load_dataset('titanic')
```

[16]:
```
print(df.head())
```

```
   survived  pclass     sex   age  sibsp  parch     fare embarked  class  \
```

```
0         0       3     male  22.0      1      0   7.2500        S  Third
1         1       1   female  38.0      1      0  71.2833        C  First
2         1       3   female  26.0      0      0   7.9250        S  Third
3         1       1   female  35.0      1      0  53.1000        S  First
4         0       3     male  35.0      0      0   8.0500        S  Third

     who  adult_male deck  embark_town alive  alone
0    man        True  NaN  Southampton    no  False
1  woman       False    C    Cherbourg   yes  False
2  woman       False  NaN  Southampton   yes   True
3  woman       False    C  Southampton   yes  False
4    man        True  NaN  Southampton    no   True
```

[17]:

```
print(df.isnull().sum())
```

```
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
deck           688
embark_town      2
alive            0
alone            0
dtype: int64
```

[18]:

```
df_age = df[['age']].copy()
print(df_age.describe())
```

```
              age
count  714.000000
mean    29.699118
std     14.526497
min      0.420000
25%     20.125000
50%     28.000000
75%     38.000000
max     80.000000
```

(a) Mean Imputation

[20]:
```
mean_value = df_age['age'].mean()
df_mean = df_age.fillna(mean_value)
```

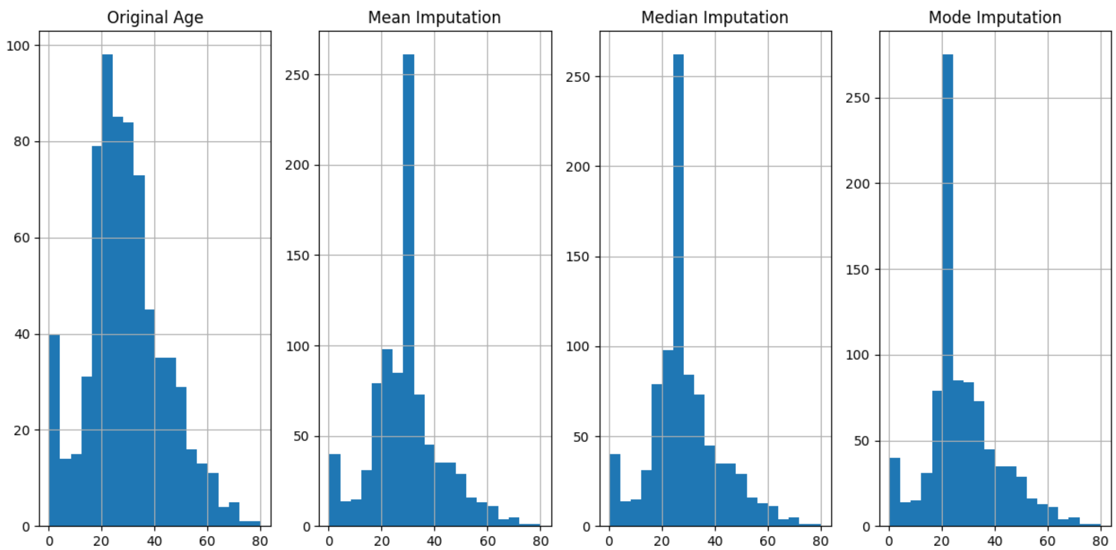
(b) Median Imputation

[21]:
```
median_value = df_age['age'].median()
df_median = df_age.fillna(median_value)
```

Mode Imputation

[22]:
```
mode_value = df_age['age'].mode()[0]
df_mode = df_age.fillna(mode_value)
```

[23]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))

# Original (with missing values)
plt.subplot(1, 4, 1)
df_age['age'].hist(bins=20)
plt.title('Original Age')

# Mean Imputation
plt.subplot(1, 4, 2)
df_mean['age'].hist(bins=20)
plt.title('Mean Imputation')

# Median Imputation
plt.subplot(1, 4, 3)
df_median['age'].hist(bins=20)
plt.title('Median Imputation')

# Mode Imputation
plt.subplot(1, 4, 4)
df_mode['age'].hist(bins=20)
plt.title('Mode Imputation')

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

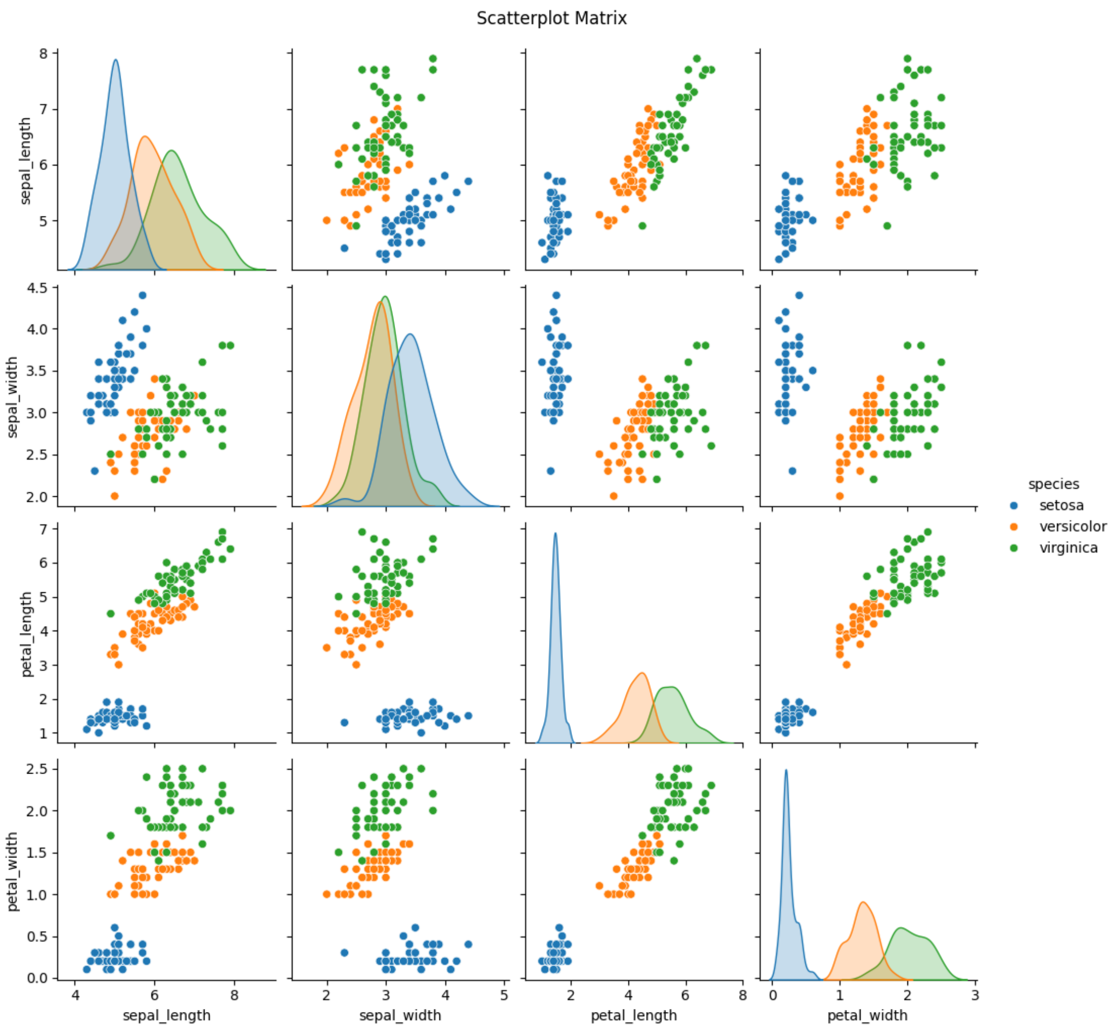


Q2.Data Analysis and Visualization: Use a dataset to: Plot scatterplots for numerical columns.Perform correlation analysis. Apply transformations (e.g., log, square root) and visualize the effect.

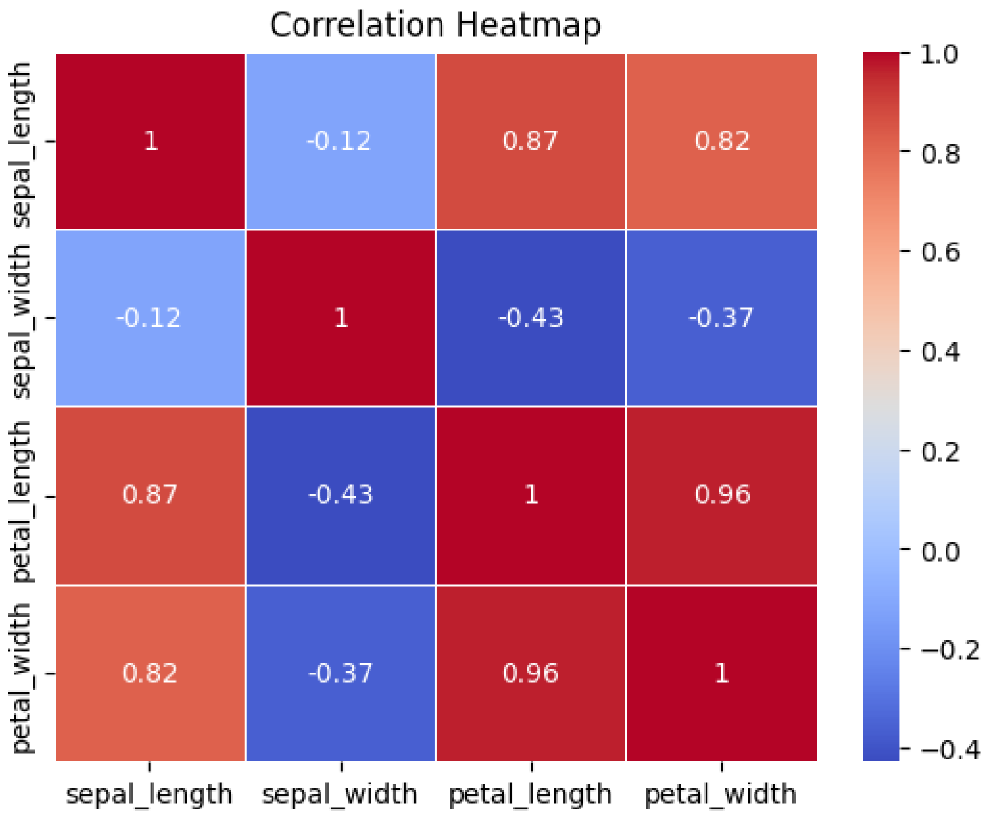
[24]:
```
import seaborn as sns
import pandas as pd

# Load Iris dataset
df = sns.load_dataset('iris')

# Show first few rows
print(df.head())
```

```
   sepal_length  sepal_width  petal_length  petal_width species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa
```

[25]:
```
import matplotlib.pyplot as plt

# Pairplot to plot all scatterplots between numerical columns
sns.pairplot(df, hue='species')
plt.suptitle("Scatterplot Matrix", y=1.02)
plt.show()
```

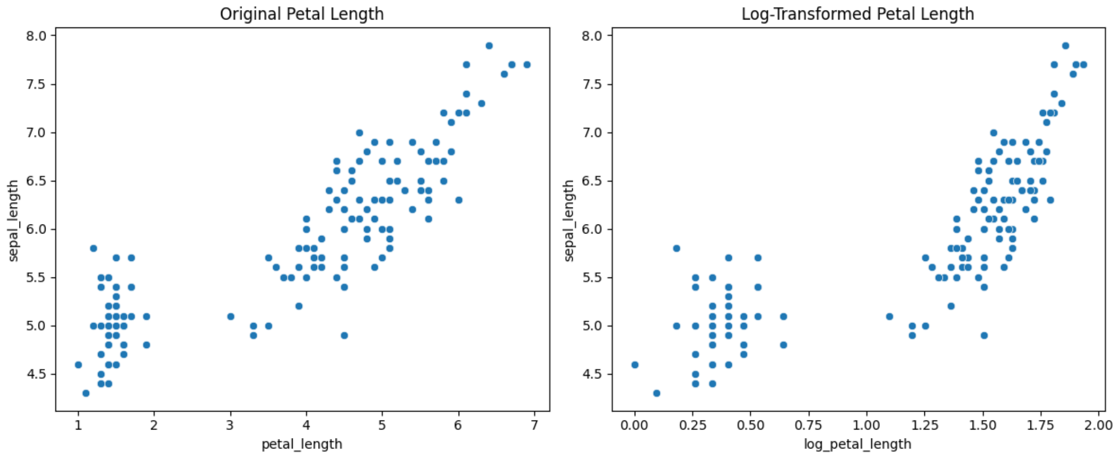


[26]:
```python
# Compute correlation
correlation_matrix = df.corr(numeric_only=True)

# Heatmap
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title("Correlation Heatmap")
plt.show()
```



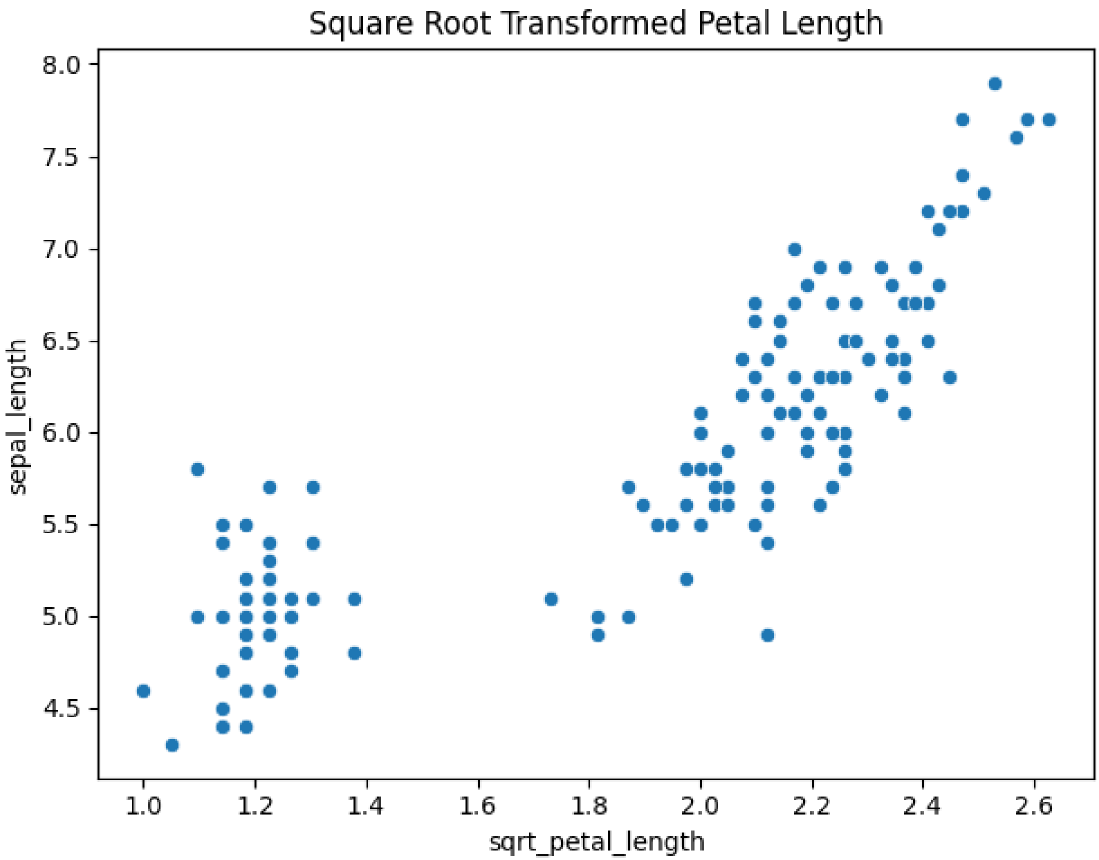
[27]:

```python
import numpy as np

df['log_petal_length'] = np.log(df['petal_length'])

# Scatterplot before and after transformation
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
sns.scatterplot(x='petal_length', y='sepal_length', data=df)
plt.title("Original Petal Length")

plt.subplot(1, 2, 2)
sns.scatterplot(x='log_petal_length', y='sepal_length', data=df)
plt.title("Log-Transformed Petal Length")

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

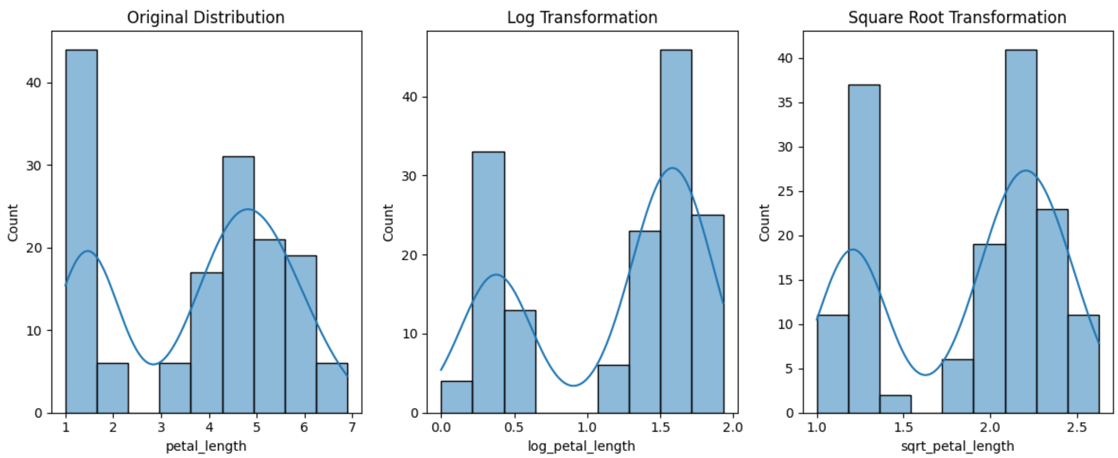


[28]:

```
df['sqrt_petal_length'] = np.sqrt(df['petal_length'])

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
sns.scatterplot(x='sqrt_petal_length', y='sepal_length', data=df)
plt.title("Square Root Transformed Petal Length")

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



[29]:
```
plt.figure(figsize=(12, 5))

# Original
plt.subplot(1, 3, 1)
sns.histplot(df['petal_length'], kde=True)
plt.title("Original Distribution")

# Log Transformed
plt.subplot(1, 3, 2)
sns.histplot(df['log_petal_length'], kde=True)
plt.title("Log Transformation")

# Sqrt Transformed
plt.subplot(1, 3, 3)
sns.histplot(df['sqrt_petal_length'], kde=True)
plt.title("Square Root Transformation")

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



Q3.Encoding Methods:

Encode a categorical dataset using One-Hot Encoding and Label Encoding. Compare the effect of both methods on a machine-learning model.

Step 1: Load and Preprocess Titanic Dataset

[30]:
```python
import seaborn as sns
import pandas as pd

# Load dataset
df = sns.load_dataset('titanic')

# Keep relevant columns
df = df[['survived', 'sex', 'embarked', 'pclass', 'age']]

# Drop rows with missing values
df.dropna(inplace=True)

# Show dataset
print(df.head())
```

```
   survived     sex embarked  pclass   age
0         0    male        S       3  22.0
1         1  female        C       1  38.0
2         1  female        S       3  26.0
3         1  female        S       1  35.0
4         0    male        S       3  35.0
```

Step 2: Label Encoding

[34]:
```
!pip install scikit-learn
```

```
Collecting scikit-learn
```

```
  Downloading scikit_learn-1.6.1-cp313-cp313-win_amd64.whl.metadata (15 kB)
Requirement already satisfied: numpy>=1.19.5 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from scikit-
learn) (2.2.4)
Collecting scipy>=1.6.0 (from scikit-learn)
  Downloading scipy-1.15.2-cp313-cp313-win_amd64.whl.metadata (60 kB)
Collecting joblib>=1.2.0 (from scikit-learn)
  Downloading joblib-1.4.2-py3-none-any.whl.metadata (5.4 kB)
Collecting threadpoolctl>=3.1.0 (from scikit-learn)
  Downloading threadpoolctl-3.6.0-py3-none-any.whl.metadata (13 kB)
Downloading scikit_learn-1.6.1-cp313-cp313-win_amd64.whl (11.1 MB)
   ---------------------------------------- 0.0/11.1 MB ? eta -:--:--
    --------------------------------------- 0.3/11.1 MB ? eta -:--:--
   -- ------------------------------------- 0.8/11.1 MB 2.6 MB/s eta 0:00:04
   ---- ----------------------------------- 1.3/11.1 MB 2.8 MB/s eta 0:00:04
   ------- -------------------------------- 2.1/11.1 MB 2.8 MB/s eta 0:00:04
   ---------- ----------------------------- 2.9/11.1 MB 3.1 MB/s eta 0:00:03
   -------------- ------------------------- 3.9/11.1 MB 3.5 MB/s eta 0:00:03
   ----------------- ---------------------- 4.7/11.1 MB 3.5 MB/s eta 0:00:02
   --------------------- ------------------ 6.0/11.1 MB 3.9 MB/s eta 0:00:02
   -------------------------- ------------- 7.3/11.1 MB 4.2 MB/s eta 0:00:01
   -------------------------------- ------- 8.9/11.1 MB 4.5 MB/s eta 0:00:01
   -------------------------------------- - 10.7/11.1 MB 4.9 MB/s eta 0:00:01
   ---------------------------------------- 11.1/11.1 MB 4.9 MB/s eta 0:00:00
Downloading joblib-1.4.2-py3-none-any.whl (301 kB)
Downloading scipy-1.15.2-cp313-cp313-win_amd64.whl (41.0 MB)
   ---------------------------------------- 0.0/41.0 MB ? eta -:--:--
   - -------------------------------------- 1.3/41.0 MB 7.6 MB/s eta 0:00:06
   -- ------------------------------------- 2.6/41.0 MB 6.8 MB/s eta 0:00:06
   ---- ----------------------------------- 4.2/41.0 MB 7.1 MB/s eta 0:00:06
   ----- ---------------------------------- 6.0/41.0 MB 7.4 MB/s eta 0:00:05
   ------- -------------------------------- 7.3/41.0 MB 7.2 MB/s eta 0:00:05
   -------- ------------------------------- 8.7/41.0 MB 7.0 MB/s eta 0:00:05
   ---------- ----------------------------- 10.5/41.0 MB 7.0 MB/s eta 0:00:05
   ------------ --------------------------- 12.3/41.0 MB 7.1 MB/s eta 0:00:05
   ------------- -------------------------- 13.9/41.0 MB 7.3 MB/s eta 0:00:04
   --------------- ------------------------ 15.5/41.0 MB 7.4 MB/s eta 0:00:04
   ---------------- ----------------------- 17.0/41.0 MB 7.4 MB/s eta 0:00:04
   ----------------- ---------------------- 18.1/41.0 MB 7.2 MB/s eta 0:00:04
   ------------------ --------------------- 19.4/41.0 MB 7.0 MB/s eta 0:00:04
   ------------------- -------------------- 19.9/41.0 MB 6.8 MB/s eta 0:00:04
   --------------------- ------------------ 21.8/41.0 MB 6.8 MB/s eta 0:00:03
   ---------------------- ----------------- 23.3/41.0 MB 6.9 MB/s eta 0:00:03
   ------------------------ --------------- 25.2/41.0 MB 7.0 MB/s eta 0:00:03
   ------------------------- -------------- 26.2/41.0 MB 6.9 MB/s eta 0:00:03
   -------------------------- ------------- 27.5/41.0 MB 6.9 MB/s eta 0:00:02
   ---------------------------- ----------- 29.1/41.0 MB 6.9 MB/s eta 0:00:02
   ----------------------------- ---------- 30.7/41.0 MB 6.9 MB/s eta 0:00:02
```

```
   ------------------------------ -------- 32.2/41.0 MB 6.9 MB/s eta 0:00:02
   ------------------------------- ------- 33.3/41.0 MB 6.9 MB/s eta 0:00:02
   -------------------------------- ------ 34.3/41.0 MB 6.8 MB/s eta 0:00:01
   --------------------------------- ----- 35.4/41.0 MB 6.7 MB/s eta 0:00:01
   ----------------------------------- --- 37.0/41.0 MB 6.7 MB/s eta 0:00:01
   ------------------------------------ -- 38.5/41.0 MB 6.8 MB/s eta 0:00:01
   ------------------------------------- - 39.1/41.0 MB 6.7 MB/s eta 0:00:01
   ------------------------------------- - 39.6/41.0 MB 6.6 MB/s eta 0:00:01
   --------------------------------------  40.4/41.0 MB 6.5 MB/s eta 0:00:01
   --------------------------------------  40.9/41.0 MB 6.3 MB/s eta 0:00:01
   --------------------------------------- 41.0/41.0 MB 6.3 MB/s eta 0:00:00
Downloading threadpoolctl-3.6.0-py3-none-any.whl (18 kB)
Installing collected packages: threadpoolctl, scipy, joblib, scikit-learn
Successfully installed joblib-1.4.2 scikit-learn-1.6.1 scipy-1.15.2
threadpoolctl-3.6.0
```

[35]:

```python
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

[37]:

```python
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Copy dataset
df_label = df.copy()

# Apply Label Encoding to categorical columns
le = LabelEncoder()
df_label['sex'] = le.fit_transform(df_label['sex'])        # male = 1, female =␣
 ↪0
df_label['embarked'] = le.fit_transform(df_label['embarked'])

# Train-Test Split
X_label = df_label.drop('survived', axis=1)
y_label = df_label['survived']
X_train, X_test, y_train, y_test = train_test_split(X_label, y_label,␣
 ↪test_size=0.2, random_state=42)

# Model
model_label = RandomForestClassifier(random_state=42)
model_label.fit(X_train, y_train)
pred_label = model_label.predict(X_test)

# Accuracy
```

```
acc_label = accuracy_score(y_test, pred_label)
print("Label Encoding Accuracy:", acc_label)
```

```
Label Encoding Accuracy: 0.7692307692307693
```

Step 3: One-Hot Encoding

1 Copy dataset

df_onehot = df.copy()

2 Apply One-Hot Encoding

df_onehot = pd.get_dummies(df_onehot, columns=['sex', 'embarked'], drop_first=True)

3 Train-Test Split

X_onehot = df_onehot.drop('survived', axis=1) y_onehot = df_onehot['survived'] X_train_oh, X_test_oh, y_train_oh, y_test_oh = train_test_split(X_onehot, y_onehot, test_size=0.2, random_state=42)

4 Model

model_onehot = RandomForestClassifier(random_state=42) model_onehot.fit(X_train_oh, y_train_oh) pred_onehot = model_onehot.predict(X_test_oh)

5 Accuracy

acc_onehot = accuracy_score(y_test_oh, pred_onehot) print("One-Hot Encoding Accuracy:", acc_onehot)

Q4. Outlier Detection:

Use the Isolation Forest algorithm to detect and visualize outliers in a dataset.

Step 1: Import Libraries & Load Dataset

[40]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.ensemble import IsolationForest

# Load the Iris dataset
df = sns.load_dataset('iris')

# We'll only use numerical features
df = df[['sepal_length', 'sepal_width']]
```

```python
# Display the data
print(df.head())
```

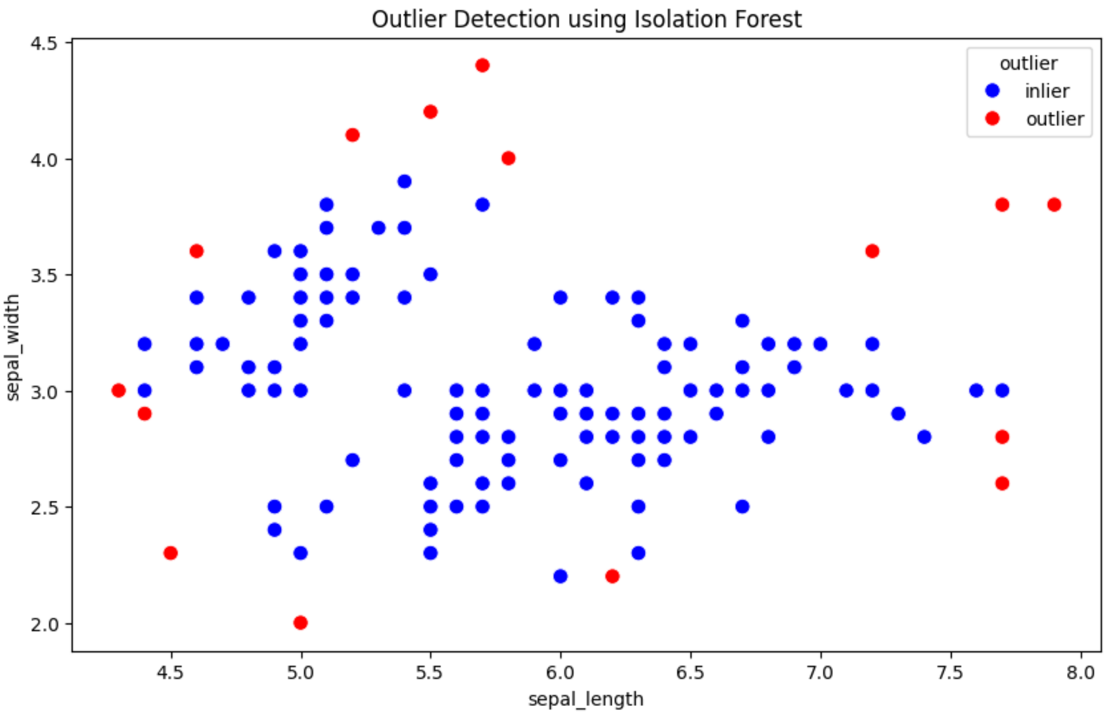
```
   sepal_length  sepal_width
0           5.1          3.5
1           4.9          3.0
2           4.7          3.2
3           4.6          3.1
4           5.0          3.6
```

Step 2: Apply Isolation Forest

[41]:
```python
# Initialize the model
iso = IsolationForest(contamination=0.1, random_state=42)  # 10% outliers

# Fit and predict
df['outlier'] = iso.fit_predict(df)

# Map results: -1 is outlier, 1 is inlier
df['outlier'] = df['outlier'].map({1: 'inlier', -1: 'outlier'})
```

Step 3: Visualize Outliers

[42]:
```python
# Scatter plot showing outliers
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='sepal_length', y='sepal_width', hue='outlier',␣
 ↪palette={'inlier': 'blue', 'outlier': 'red'}, s=70)
plt.title("Outlier Detection using Isolation Forest")
plt.show()
```



Q5.Predictive Power Score (PPS):

Calculate the PPS for a dataset and interpret which variables are most predictive.

[ ]:
```
!pip install ppscore
```

[ ]:
```
import ppscore as pps
```

[ ]:
```
import pandas as pd
import seaborn as sns
import ppscore as pps

# Load dataset
df = sns.load_dataset("titanic")

# Clean up: drop rows with NaNs and irrelevant columns
df = df[['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'fare',␣
↪'embarked']].dropna()
```

[ ]:
```
# PPS matrix: predictive power of each feature → target
pps_matrix = pps.matrix(df)

# Show top predictive features
pps_matrix_filtered = pps_matrix[pps_matrix['y'] == 'survived'].
↪sort_values('ppscore', ascending=False)
```

```python
print(pps_matrix_filtered[['x', 'y', 'ppscore']])
```

[ ]:
```python
# Top features predictive of 'survived'
top_predictors = pps_matrix[pps_matrix['y'] == 'survived'].
    ↪sort_values('ppscore', ascending=False)
print(top_predictors[['x', 'ppscore']])
```

[ ]:
```python
import pandas as pd

# Replace 'your_file.csv' with your actual file path
df = pd.read_csv("sample.csv")

# Preview it
df.head()
```

[ ]:
```python
!pip install ppscore
import ppscore as pps
```

[ ]:
```python
# Calculate PPS matrix
pps_matrix = pps.matrix(df)

# See top predictors for a specific column
target_column = 'your_target_column'  # <- Replace this
pps_matrix[pps_matrix['y'] == target_column].sort_values('ppscore',␣
    ↪ascending=False)
```

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

# Pivot for heatmap
pps_heatmap = pps_matrix.pivot(index='x', columns='y', values='ppscore')

plt.figure(figsize=(12, 6))
sns.heatmap(pps_heatmap, annot=True, cmap="YlGnBu")
plt.title("Predictive Power Score Matrix")
plt.show()
```

Q6.Simple and Multiple Linear Regression:

Implement simple and multiple linear regression on a dataset. Evaluate the model's performance using R- squared and mean squared error (MSE).

Step 1: Import Libraries & Dataset

[4]:
```python
import pandas as pd
import numpy as np
import seaborn as sns
```

```python
import matplotlib.pyplot as plt

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error

# Load dataset (we'll use 'mpg' from seaborn as it's perfect)
df = sns.load_dataset('mpg').dropna()

# Preview
df.head()
```

[4]:

```
    mpg  cylinders  displacement  horsepower  weight  acceleration  \
0  18.0          8         307.0       130.0    3504          12.0
1  15.0          8         350.0       165.0    3693          11.5
2  18.0          8         318.0       150.0    3436          11.0
3  16.0          8         304.0       150.0    3433          12.0
4  17.0          8         302.0       140.0    3449          10.5

   model_year origin                       name
0          70    usa  chevrolet chevelle malibu
1          70    usa          buick skylark 320
2          70    usa         plymouth satellite
3          70    usa              amc rebel sst
4          70    usa                ford torino
```

Step 2: Simple Linear Regression

[5]:

```python
# Simple Linear Regression
X_simple = df[['horsepower']]
y = df['mpg']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_simple, y, test_size=0.2,␣
↪random_state=42)

# Model
model_simple = LinearRegression()
model_simple.fit(X_train, y_train)

# Predict
y_pred = model_simple.predict(X_test)

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

```
print("Simple Linear Regression")
print("R-squared:", round(r2, 3))
print("Mean Squared Error:", round(mse, 3))
```

```
Simple Linear Regression
R-squared: 0.566
Mean Squared Error: 22.153
```

Step 3: Multiple Linear Regression

[6]:
```
# Multiple Linear Regression
features = ['horsepower', 'weight', 'acceleration', 'displacement']
X_multi = df[features]

# Split
X_train, X_test, y_train, y_test = train_test_split(X_multi, y, test_size=0.2,
 ↪random_state=42)

# Model
model_multi = LinearRegression()
model_multi.fit(X_train, y_train)

# Predict
y_pred_multi = model_multi.predict(X_test)

# Evaluation
r2_multi = r2_score(y_test, y_pred_multi)
mse_multi = mean_squared_error(y_test, y_pred_multi)

print("\nMultiple Linear Regression")
print("R-squared:", round(r2_multi, 3))
print("Mean Squared Error:", round(mse_multi, 3))
```

```
Multiple Linear Regression
R-squared: 0.646
Mean Squared Error: 18.066
```

Q7.Build a logistic regression model to classify binary outcomes (e.g., predicting if a customer will buy a product). Evaluate the model using confusion matrix metrics.

Step 1: Import Libraries & Load Dataset

[7]:
```
import pandas as pd
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
```

```
# Load Titanic dataset
df = sns.load_dataset('titanic')

# Select useful features and drop missing data
df = df[['survived', 'pclass', 'sex', 'age', 'fare']].dropna()
df['sex'] = df['sex'].map({'male': 0, 'female': 1})  # Encode 'sex'
```

Step 2: Split Data

[8]:
```
# Features and target
X = df[['pclass', 'sex', 'age', 'fare']]
y = df['survived']

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

Step 3: Train Logistic Regression

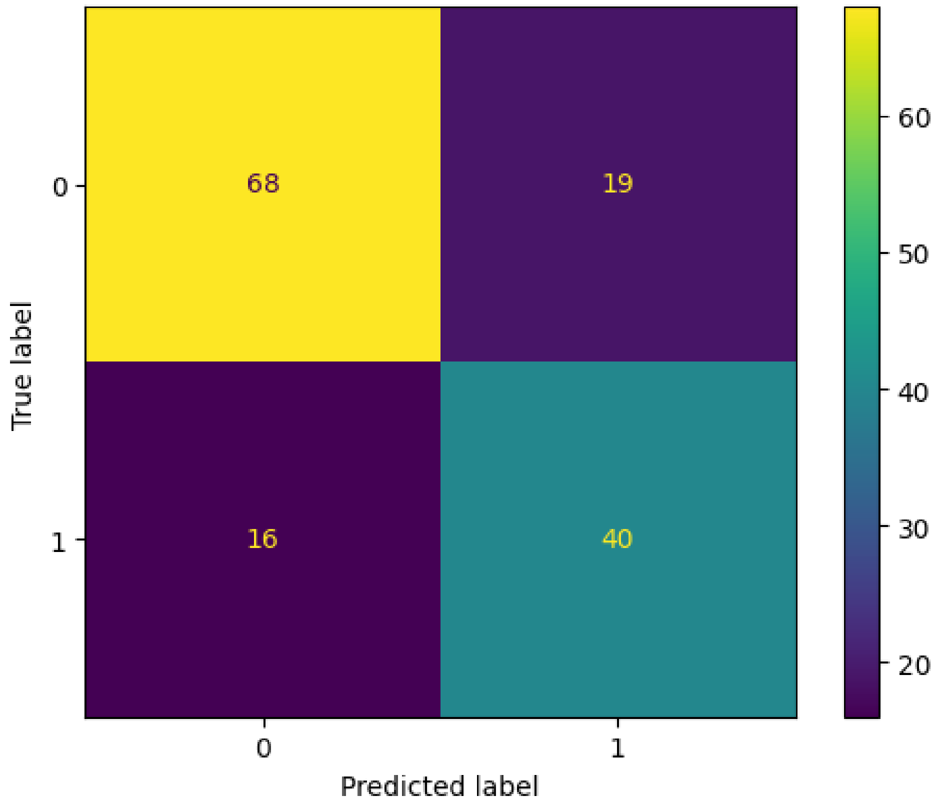
[10]:
```
model = LogisticRegression()
model.fit(X_train, y_train)

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

Step 4: Evaluate Using Confusion Matrix

[11]:
```
from sklearn.metrics import ConfusionMatrixDisplay

# Confusion Matrix & Classification Report
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

# Optional visualization
ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
```

```
[[68 19]
 [16 40]]
              precision    recall  f1-score   support

           0       0.81      0.78      0.80        87
           1       0.68      0.71      0.70        56

    accuracy                           0.76       143
   macro avg       0.74      0.75      0.75       143
weighted avg       0.76      0.76      0.76       143
```

[11]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1d2eb886900>



Q8.Clustering Techniques: Perform K-Means and hierarchical clustering on a dataset. Visualize the clusters and interpret the results.

Step 1: Import Libraries & Load Data

[12]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from scipy.cluster.hierarchy import linkage, dendrogram, fcluster
from sklearn.preprocessing import StandardScaler

# Load Iris dataset
df = sns.load_dataset("iris")
df.head()
```

[12]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa

3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Step 2: Prepare Features (Drop the label)

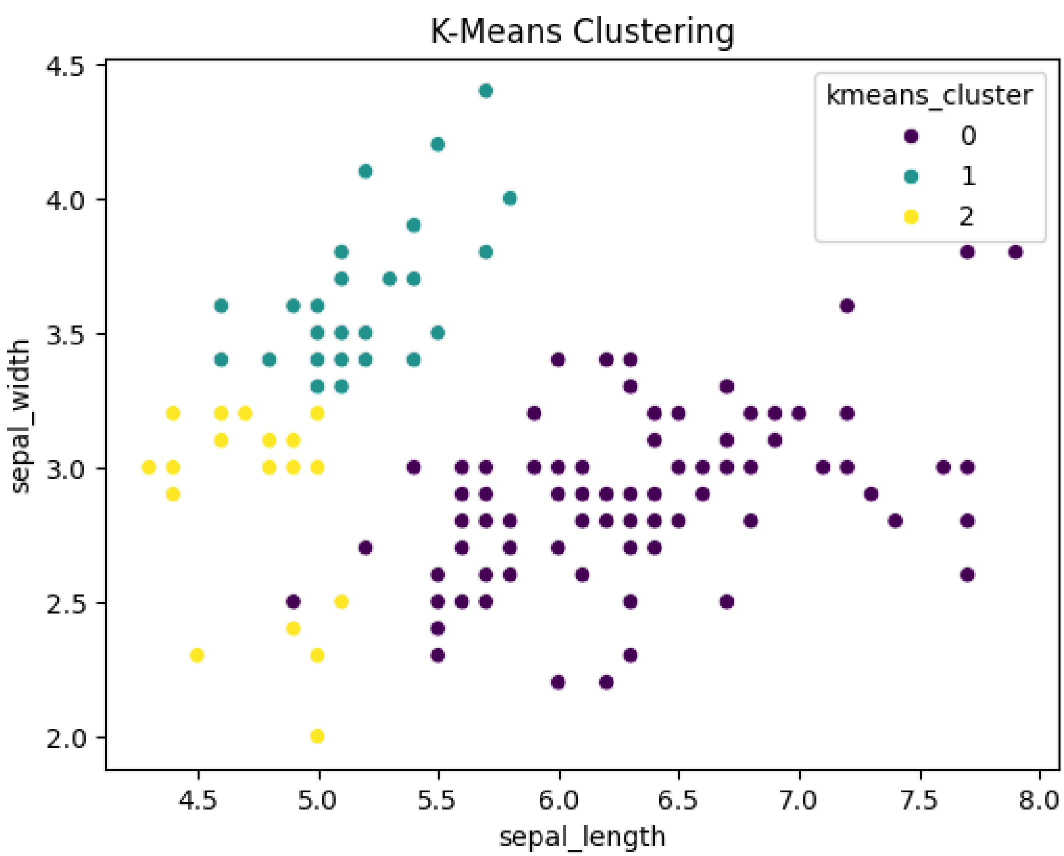
[13]:
```python
X = df.drop('species', axis=1)

# Standardize features (important for distance-based algorithms)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Step 3: Apply K-Means

[14]:
```python
kmeans = KMeans(n_clusters=3, random_state=42)
df['kmeans_cluster'] = kmeans.fit_predict(X_scaled)
```

Step 4: Visualize K-Means Clusters

[15]:
```python
sns.scatterplot(data=df, x='sepal_length', y='sepal_width',␣
 ↪hue='kmeans_cluster', palette='viridis')
plt.title("K-Means Clustering")
plt.show()
```

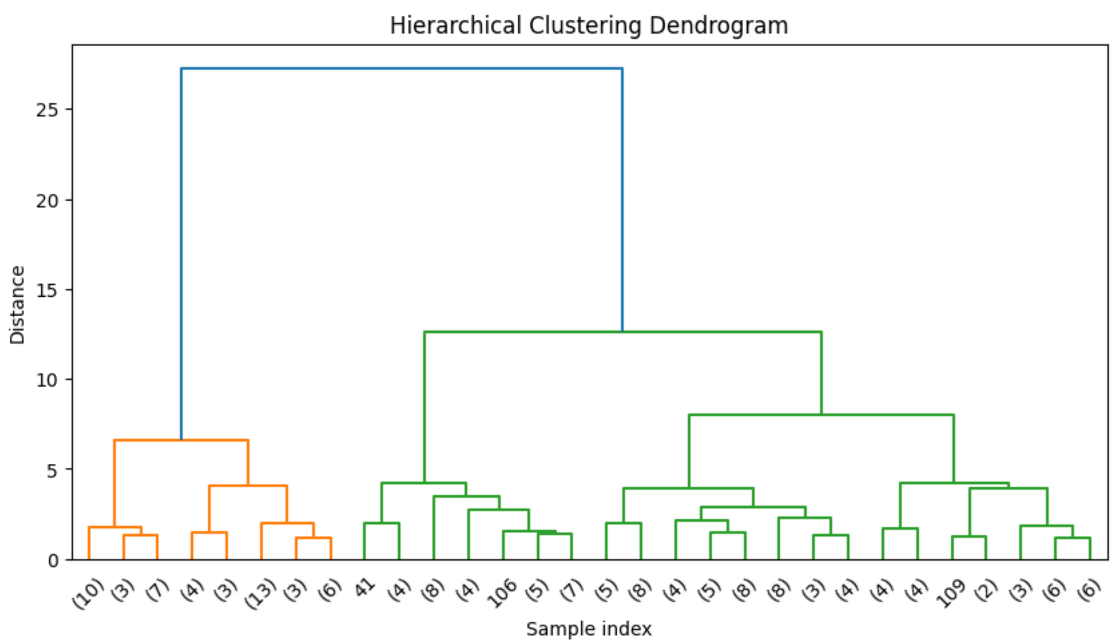


Hierarchical Clustering Step 5: Apply Hierarchical Clustering

[16]:
```python
linkage_matrix = linkage(X_scaled, method='ward')
```

Step 6: Dendrogram

[18]:
```python
plt.figure(figsize=(10, 5))
dendrogram(linkage_matrix, truncate_mode='lastp', p=30)
plt.title("Hierarchical Clustering Dendrogram")
plt.xlabel("Sample index")
plt.ylabel("Distance")
plt.show()
```

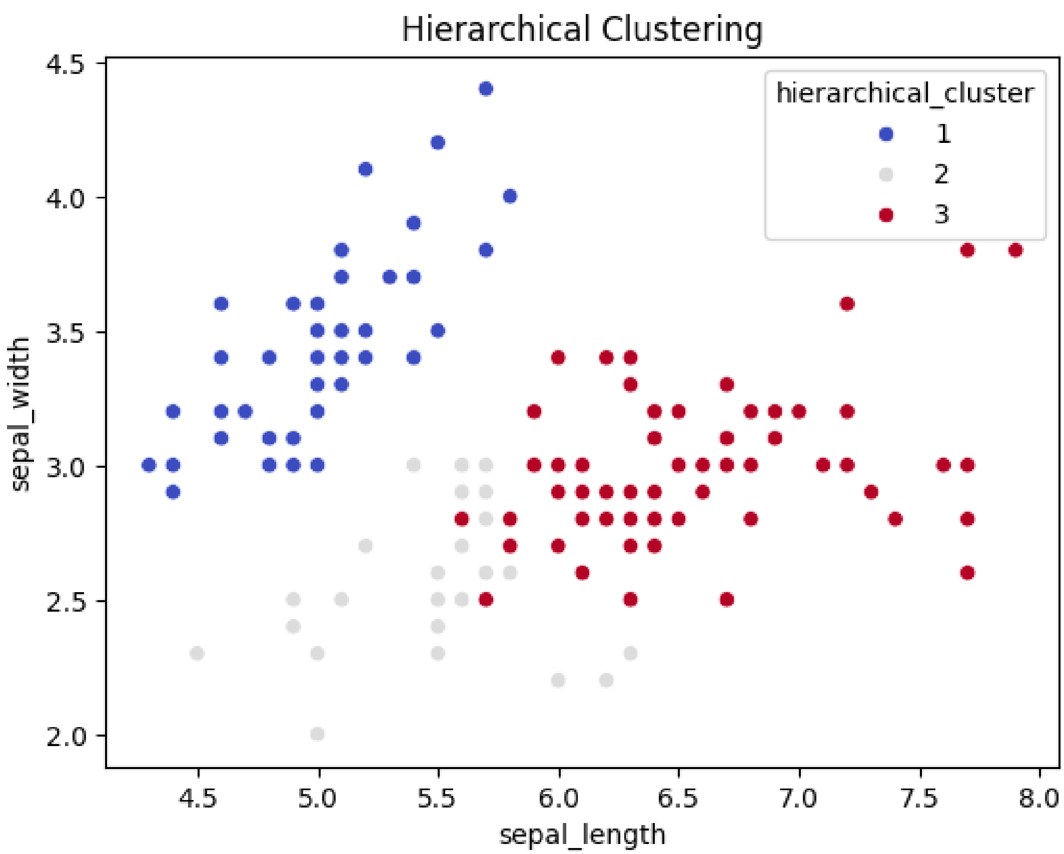


Step 7: Assign Cluster Labels

[19]:
```python
df['hierarchical_cluster'] = fcluster(linkage_matrix, t=3, criterion='maxclust')
```

Step 8: Visualize Hierarchical Clusters

[20]:
```python
sns.scatterplot(data=df, x='sepal_length', y='sepal_width', hue='hierarchical_cluster', palette='coolwarm')
plt.title("Hierarchical Clustering")
plt.show()
```



Q9.Principal Component Analysis (PCA):

Apply PCA on a high-dimensional dataset. Reduce the dimensions and visualize the transformed data.

Step 1: Import Libraries & Load Data

[21]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Load dataset
df = sns.load_dataset('iris')

# Separate features and label
X = df.drop('species', axis=1)
y = df['species']
```

Step 2: Standardize the Data

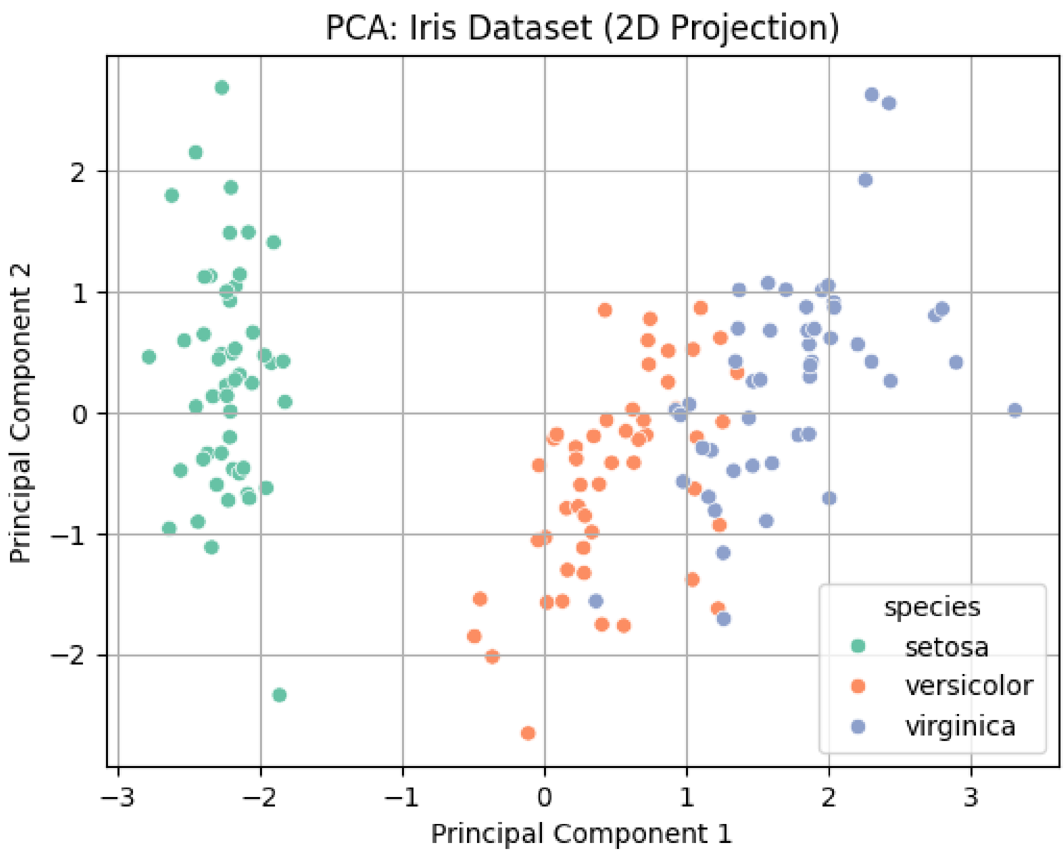
[22]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Step 3: Apply PCA

[23]:
```
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Convert to DataFrame
df_pca = pd.DataFrame(X_pca, columns=['PC1', 'PC2'])
df_pca['species'] = y
```

Step 4: Visualize PCA-Reduced Data

[25]:
```
sns.scatterplot(data=df_pca, x='PC1', y='PC2', hue='species', palette='Set2')
plt.title("PCA: Iris Dataset (2D Projection)")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.grid(True)
plt.show()
```



Unit 3: Specialized Applications in AI and ML Market Basket Analysis: Q10.Implement Association Rule Mining using the Apriori algorithm. Identify frequent itemsets and generate association rules for a transactional dataset.

[26]:
```
!pip install mlxtend
```

```
Collecting mlxtend
  Downloading mlxtend-0.23.4-py3-none-any.whl.metadata (7.3 kB)
Requirement already satisfied: scipy>=1.2.1 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from mlxtend)
(1.15.2)
Requirement already satisfied: numpy>=1.16.2 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from mlxtend)
(2.2.4)
Requirement already satisfied: pandas>=0.24.2 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from mlxtend)
(2.2.3)
Requirement already satisfied: scikit-learn>=1.3.1 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from mlxtend)
(1.6.1)
Requirement already satisfied: matplotlib>=3.0.0 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from mlxtend)
(3.10.1)
Requirement already satisfied: joblib>=0.13.2 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from mlxtend)
(1.4.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from
matplotlib>=3.0.0->mlxtend) (1.3.1)
Requirement already satisfied: cycler>=0.10 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from
matplotlib>=3.0.0->mlxtend) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from
matplotlib>=3.0.0->mlxtend) (4.57.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from
matplotlib>=3.0.0->mlxtend) (1.4.8)
Requirement already satisfied: packaging>=20.0 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from
matplotlib>=3.0.0->mlxtend) (24.2)
Requirement already satisfied: pillow>=8 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from
matplotlib>=3.0.0->mlxtend) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from
matplotlib>=3.0.0->mlxtend) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\shinde
```

```
ankita\appdata\local\programs\python\python313\lib\site-packages (from
matplotlib>=3.0.0->mlxtend) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from
pandas>=0.24.2->mlxtend) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from
pandas>=0.24.2->mlxtend) (2025.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from scikit-
learn>=1.3.1->mlxtend) (3.6.0)
Requirement already satisfied: six>=1.5 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from python-
dateutil>=2.7->matplotlib>=3.0.0->mlxtend) (1.17.0)
Downloading mlxtend-0.23.4-py3-none-any.whl (1.4 MB)
   ---------------------------------------- 0.0/1.4 MB ? eta -:--:--
   ----------------------- ---------------- 0.8/1.4 MB 8.0 MB/s eta 0:00:01
   ---------------------------------------- 1.4/1.4 MB 6.7 MB/s eta 0:00:00
Installing collected packages: mlxtend
Successfully installed mlxtend-0.23.4
```

[27]:

```python
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder

# Sample transactional data
transactions = [
    ['milk', 'bread', 'butter'],
    ['bread', 'diapers', 'beer', 'eggs'],
    ['milk', 'diapers', 'beer', 'cola'],
    ['bread', 'milk', 'diapers', 'beer'],
    ['bread', 'milk', 'diapers', 'cola']
]

# Convert to one-hot encoding format
te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
df = pd.DataFrame(te_ary, columns=te.columns_)
df.head()
```

[27]:

	beer	bread	butter	cola	diapers	eggs	milk
0	False	True	True	False	False	False	True
1	True	True	False	False	True	True	False
2	True	False	False	True	True	False	True
3	True	True	False	False	True	False	True
4	False	True	False	True	True	False	True

[28]:
```python
from mlxtend.frequent_patterns import apriori

# Get frequent itemsets
frequent_itemsets = apriori(df, min_support=0.6, use_colnames=True)
frequent_itemsets
```

[28]:
```
   support          itemsets
0      0.6            (beer)
1      0.8           (bread)
2      0.8         (diapers)
3      0.8            (milk)
4      0.6   (diapers, beer)
5      0.6  (bread, diapers)
6      0.6     (milk, bread)
7      0.6   (milk, diapers)
```

[29]:
```python
from mlxtend.frequent_patterns import association_rules

# Generate rules
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1.0)
rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']]
```

[29]:
```
  antecedents consequents  support  confidence  lift
0   (diapers)      (beer)      0.6        0.75  1.25
1      (beer)   (diapers)      0.6        1.00  1.25
```

Q11.Recommendation Systems:

Build a collaborative filtering recommendation system using a movie or product dataset. Compare results using user-based and item-based filtering.

[31]:
```python
import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity

# Sample ratings data (user-movie matrix)
ratings_dict = {
    'User': ['A', 'A', 'A', 'B', 'B', 'C', 'C', 'D', 'E'],
    'Movie': ['Titanic', 'Avatar', 'Avengers', 'Titanic', 'Avengers', 'Avatar',␣
↪'Avengers', 'Titanic', 'Avatar'],
    'Rating': [5, 3, 4, 4, 5, 2, 4, 5, 3]
}

df = pd.DataFrame(ratings_dict)
ratings_matrix = df.pivot_table(index='User', columns='Movie', values='Rating').
↪fillna(0)
ratings_matrix
```

[31]:
```
Movie  Avatar  Avengers  Titanic
User
A         3.0       4.0      5.0
B         0.0       5.0      4.0
C         2.0       4.0      0.0
D         0.0       0.0      5.0
E         3.0       0.0      0.0
```

[32]:
```
# Compute cosine similarity between users
user_similarity = cosine_similarity(ratings_matrix)
user_sim_df = pd.DataFrame(user_similarity, index=ratings_matrix.index,␣
 ↪columns=ratings_matrix.index)
user_sim_df
```

[32]:
```
User         A         B         C         D         E
User
A     1.000000  0.883452  0.695701  0.707107  0.424264
B     0.883452  1.000000  0.698430  0.624695  0.000000
C     0.695701  0.698430  1.000000  0.000000  0.447214
D     0.707107  0.624695  0.000000  1.000000  0.000000
E     0.424264  0.000000  0.447214  0.000000  1.000000
```

[33]:
```
# Transpose and compute cosine similarity between movies
item_similarity = cosine_similarity(ratings_matrix.T)
item_sim_df = pd.DataFrame(item_similarity, index=ratings_matrix.columns,␣
 ↪columns=ratings_matrix.columns)
item_sim_df
```

[33]:
```
Movie       Avatar  Avengers   Titanic
Movie
Avatar    1.000000  0.564782  0.393648
Avengers  0.564782  1.000000  0.652155
Titanic   0.393648  0.652155  1.000000
```

[34]:
```
similar_users = user_sim_df['C'].sort_values(ascending=False)[1:]
top_user = similar_users.index[0]
recommendation = ratings_matrix.loc[top_user][ratings_matrix.loc['C'] == 0].
 ↪sort_values(ascending=False)
recommendation
```

[34]:
```
Movie
Titanic    4.0
Name: B, dtype: float64
```

[35]:
```
item_sim_df['Avatar'].sort_values(ascending=False)[1:]
```

[35]:
```
Movie
Avengers    0.564782
Titanic     0.393648
Name: Avatar, dtype: float64
```

Q12.Tree-Based Feature Engineering

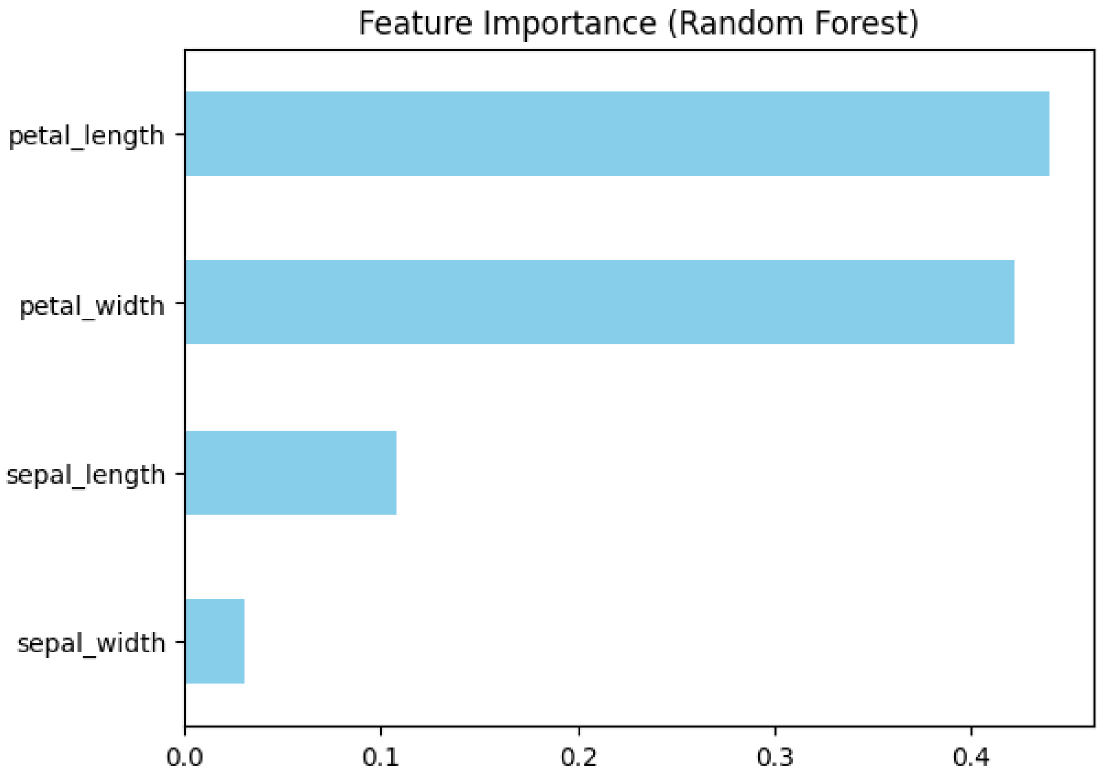
Apply tree-based methods to rank feature importance in a dataset. Use the results to train a simplified model.

[36]:
```python
import pandas as pd
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Load Iris dataset
df = sns.load_dataset("iris")

# Features and target
X = df.drop("species", axis=1)
y = df["species"]

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[37]:
```python
# Train a random forest
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Get feature importances
importances = rf.feature_importances_
feature_ranking = pd.Series(importances, index=X.columns).
↪sort_values(ascending=False)
print(feature_ranking)
```

```
petal_length    0.439994
petal_width     0.421522
sepal_length    0.108098
sepal_width     0.030387
dtype: float64
```

[38]:
```python
feature_ranking.plot(kind='barh', title='Feature Importance (Random Forest)',␣
↪color='skyblue')
plt.gca().invert_yaxis()
plt.show()
```



[39]:
```python
top_features = feature_ranking.head(2).index.tolist()

# Train with only top 2 features
X_train_simple = X_train[top_features]
X_test_simple = X_test[top_features]

rf_simple = RandomForestClassifier(n_estimators=100, random_state=42)
rf_simple.fit(X_train_simple, y_train)

# Evaluate
y_pred_simple = rf_simple.predict(X_test_simple)
accuracy = accuracy_score(y_test, y_pred_simple)
print("Simplified Model Accuracy:", round(accuracy * 100, 2), "%")
```

```
Simplified Model Accuracy: 100.0 %
```

Q13.Recursive Feature Elimination (RFE)

Perform feature selection using RFE. Evaluate the performance of a machine-learning model before and after feature selection.

[40]:
```python
import pandas as pd
import seaborn as sns
```

```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.feature_selection import RFE

# Load dataset
df = sns.load_dataset('iris')

# Prepare features and target
X = df.drop('species', axis=1)
y = df['species']

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

[41]:
```python
model = LogisticRegression(max_iter=200)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

# Accuracy before feature selection
base_acc = accuracy_score(y_test, y_pred)
print("Accuracy (All Features):", round(base_acc * 100, 2), "%")
```

```
Accuracy (All Features): 100.0 %
```

[42]:
```python
# Recursive Feature Elimination
selector = RFE(estimator=LogisticRegression(max_iter=200), n_features_to_select=2)
selector.fit(X_train, y_train)

# Get selected features
selected_features = X.columns[selector.support_]
print("Selected Features by RFE:", selected_features.tolist())
```

```
Selected Features by RFE: ['petal_length', 'petal_width']
```

[43]:
```python
# Train with selected features only
X_train_rfe = X_train[selected_features]
X_test_rfe = X_test[selected_features]

model_rfe = LogisticRegression(max_iter=200)
model_rfe.fit(X_train_rfe, y_train)
y_pred_rfe = model_rfe.predict(X_test_rfe)

# Accuracy after RFE
rfe_acc = accuracy_score(y_test, y_pred_rfe)
```

```
print("Accuracy (After RFE):", round(rfe_acc * 100, 2), "%")
```

```
Accuracy (After RFE): 100.0 %
```

Q14.Train-Test Split and Cross-Validation

Split a dataset into train-test sets. Use Shuffle Cross- Validation to evaluate a model and compare the results.

[45]:
```
import pandas as pd
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, ShuffleSplit,␣
↪cross_val_score
from sklearn.metrics import accuracy_score

# Load dataset
df = sns.load_dataset('iris')

# Features and target
X = df.drop('species', axis=1)
y = df['species']
```

[46]:
```
# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Train model
model = LogisticRegression(max_iter=200)
model.fit(X_train, y_train)

# Evaluate on test set
y_pred = model.predict(X_test)
acc_test = accuracy_score(y_test, y_pred)
print("Accuracy (Train-Test Split):", round(acc_test * 100, 2), "%")
```

```
Accuracy (Train-Test Split): 100.0 %
```

[47]:
```
from sklearn.model_selection import ShuffleSplit

# Set up ShuffleSplit
shuffle_split = ShuffleSplit(n_splits=5, test_size=0.2, random_state=42)

# Cross-validation scores
cv_scores = cross_val_score(model, X, y, cv=shuffle_split)

print("Cross-Validation Scores:", cv_scores)
print("Mean CV Accuracy:", round(cv_scores.mean() * 100, 2), "%")
```

```
Cross-Validation Scores: [1.         0.96666667 0.96666667 0.93333333
0.93333333]
Mean CV Accuracy: 96.0 %
```

Q15.Bagging and Random Forest

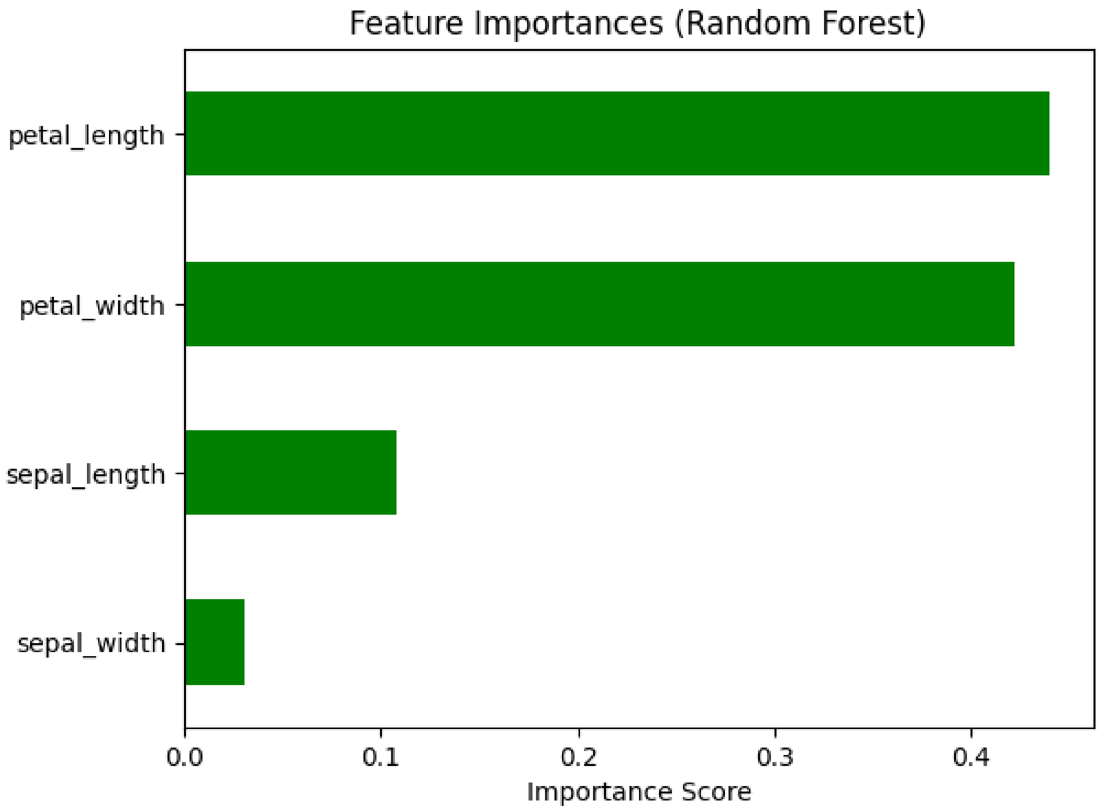
Build and evaluate a Random Forest model. Visualize the decision trees and feature importance.

[48]:
```python
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Load the Iris dataset
df = sns.load_dataset('iris')
X = df.drop('species', axis=1)
y = df['species']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

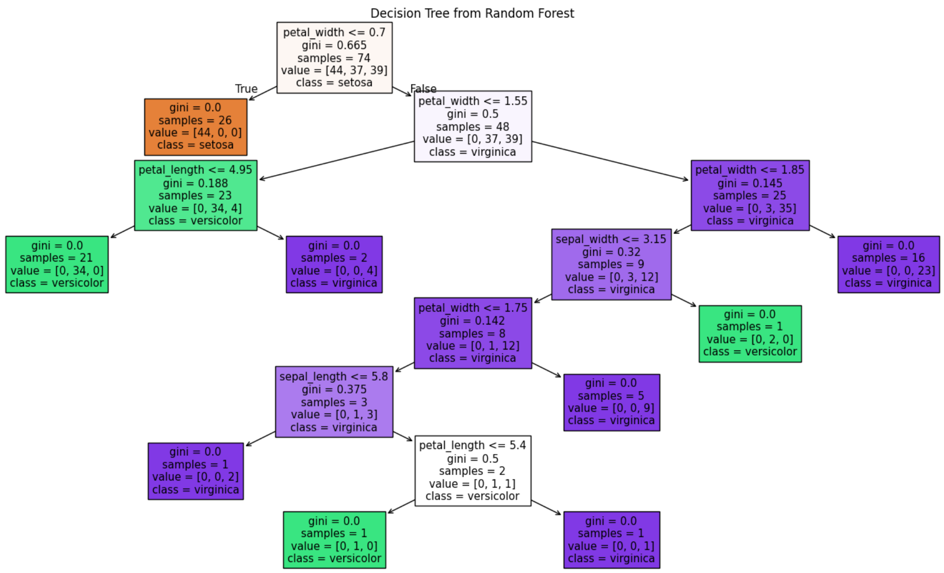
[49]:
```python
# Train a Random Forest classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Predict and evaluate
y_pred = rf.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print("Random Forest Accuracy:", round(acc * 100, 2), "%")
```

```
Random Forest Accuracy: 100.0 %
```

[50]:
```python
import matplotlib.pyplot as plt

# Get and plot feature importances
importances = rf.feature_importances_
feat_names = X.columns
feat_imp_df = pd.Series(importances, index=feat_names).
↪sort_values(ascending=True)

# Plot
feat_imp_df.plot(kind='barh', title='Feature Importances (Random Forest)',␣
↪color='green')
plt.xlabel('Importance Score')
plt.show()
```



[51]:
```
from sklearn.tree import plot_tree

# Visualize one decision tree from the forest
plt.figure(figsize=(20, 10))
plot_tree(rf.estimators_[0], feature_names=feat_names, class_names=rf.classes_,
    filled=True)
plt.title("Decision Tree from Random Forest")
plt.show()
```



Q16.Boosting Methods

Implement AdaBoost and XGBoost on a classification task. Compare their accuracy and runtime performance.

[52]:
```
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import time

# Load Iris dataset
df = sns.load_dataset('iris')
X = df.drop('species', axis=1)
y = df['species']

# Encode target labels
y = y.astype('category').cat.codes  # Converts species into numeric values

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

[53]:
```
from sklearn.ensemble import AdaBoostClassifier

# Train AdaBoost
start_time = time.time()
ada = AdaBoostClassifier(n_estimators=50, random_state=42)
ada.fit(X_train, y_train)
```

```
ada_time = time.time() - start_time

# Predict & evaluate
y_pred_ada = ada.predict(X_test)
acc_ada = accuracy_score(y_test, y_pred_ada)
print("AdaBoost Accuracy:", round(acc_ada * 100, 2), "%")
print("AdaBoost Training Time:", round(ada_time, 4), "seconds")
```

```
AdaBoost Accuracy: 93.33 %
AdaBoost Training Time: 0.1158 seconds
```

[54]:
```
from xgboost import XGBClassifier

# Train XGBoost
start_time = time.time()
xgb = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss')
xgb.fit(X_train, y_train)
xgb_time = time.time() - start_time

# Predict & evaluate
y_pred_xgb = xgb.predict(X_test)
acc_xgb = accuracy_score(y_test, y_pred_xgb)
print("XGBoost Accuracy:", round(acc_xgb * 100, 2), "%")
print("XGBoost Training Time:", round(xgb_time, 4), "seconds")
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[54], line 1
----> 1 from xgboost import XGBClassifier
      3 # Train XGBoost
      4 start_time = time.time()

ModuleNotFoundError: No module named 'xgboost'
```

!pip install xgboost

[57]:
```
from xgboost import XGBClassifier

# Train XGBoost
start_time = time.time()
xgb = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss')
xgb.fit(X_train, y_train)
xgb_time = time.time() - start_time

# Predict & evaluate
y_pred_xgb = xgb.predict(X_test)
acc_xgb = accuracy_score(y_test, y_pred_xgb)
```

```
print("XGBoost Accuracy:", round(acc_xgb * 100, 2), "%")
print("XGBoost Training Time:", round(xgb_time, 4), "seconds")
```

```
XGBoost Accuracy: 100.0 %
XGBoost Training Time: 0.0595 seconds
```

```
C:\Users\Shinde Ankita\AppData\Local\Programs\Python\Python313\Lib\site-
packages\xgboost\training.py:183: UserWarning: [13:03:43] WARNING: C:\actions-
runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
```

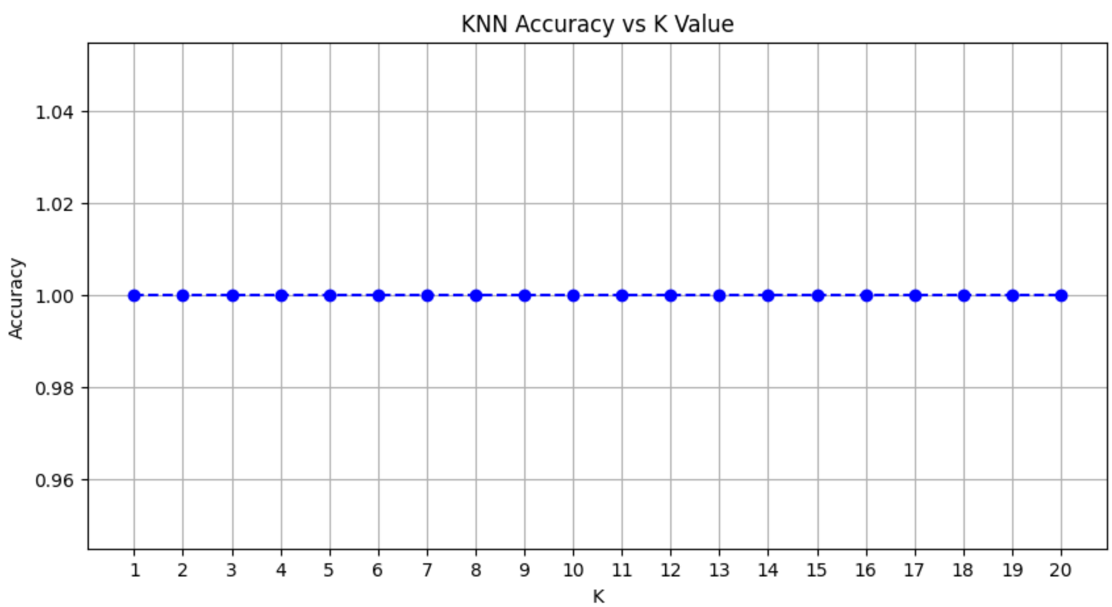
Q17.K-Nearest Neighbors (KNN)

Implement a KNN classifier for a classification task. Experiment with different values of K and analyze their impact on accuracy.

[59]:
```
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

# Load dataset
df = sns.load_dataset("iris")
X = df.drop("species", axis=1)
y = df["species"]

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Feature scaling (important for KNN!)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[60]:
```
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt

k_values = list(range(1, 21))
accuracies = []

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_scaled, y_train)
    y_pred = knn.predict(X_test_scaled)
    acc = accuracy_score(y_test, y_pred)
```

```
    accuracies.append(acc)

# Plot accuracy vs K
plt.figure(figsize=(10, 5))
plt.plot(k_values, accuracies, marker='o', linestyle='--', color='blue')
plt.title('KNN Accuracy vs K Value')
plt.xlabel('K')
plt.ylabel('Accuracy')
plt.xticks(k_values)
plt.grid(True)
plt.show()
```



Q18.Support Vector Machines (SVM)

Train an SVM model with both linear and RBF kernels on a dataset. Visualize the decision boundaries.

[61]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load dataset
df = sns.load_dataset("iris")
```

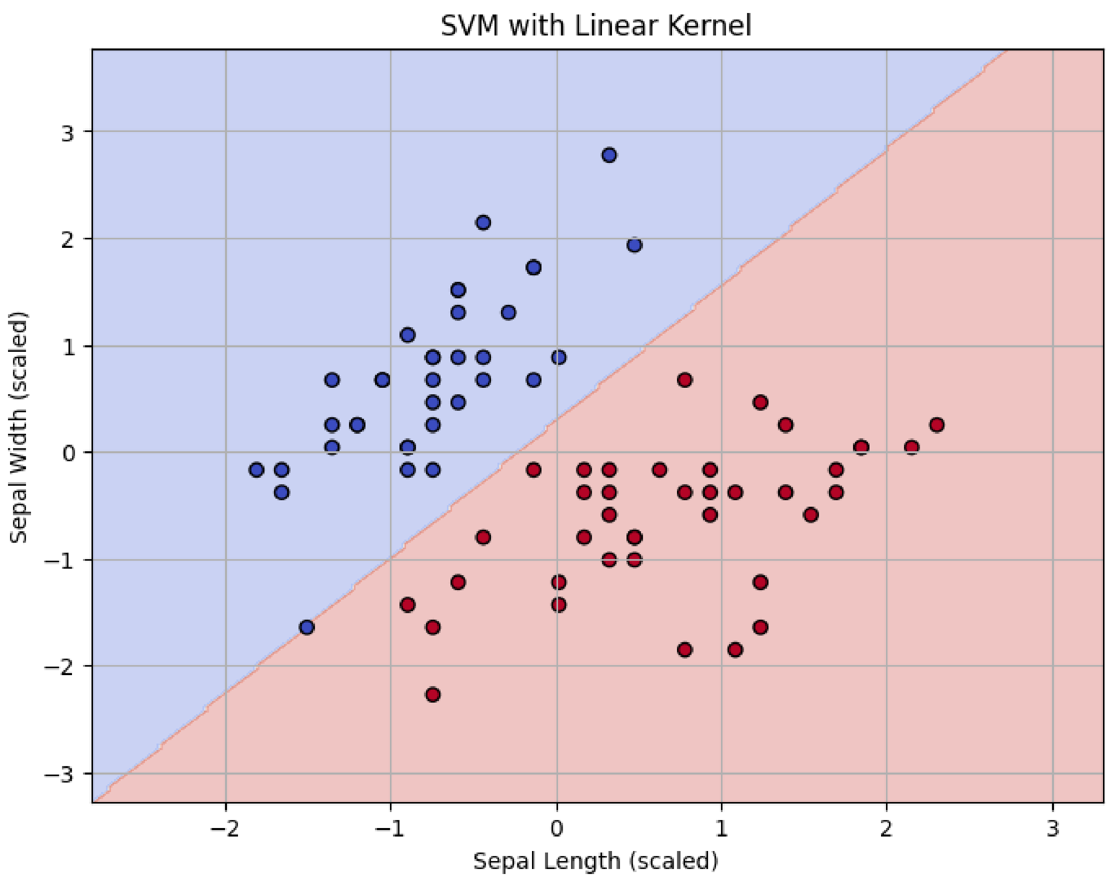
```
# Use only 2 classes: setosa and versicolor
df = df[df['species'].isin(['setosa', 'versicolor'])]

# Use 2 features for visualization
X = df[['sepal_length', 'sepal_width']]
y = df['species']

# Encode labels
y = y.astype('category').cat.codes  # 0=setosa, 1=versicolor

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

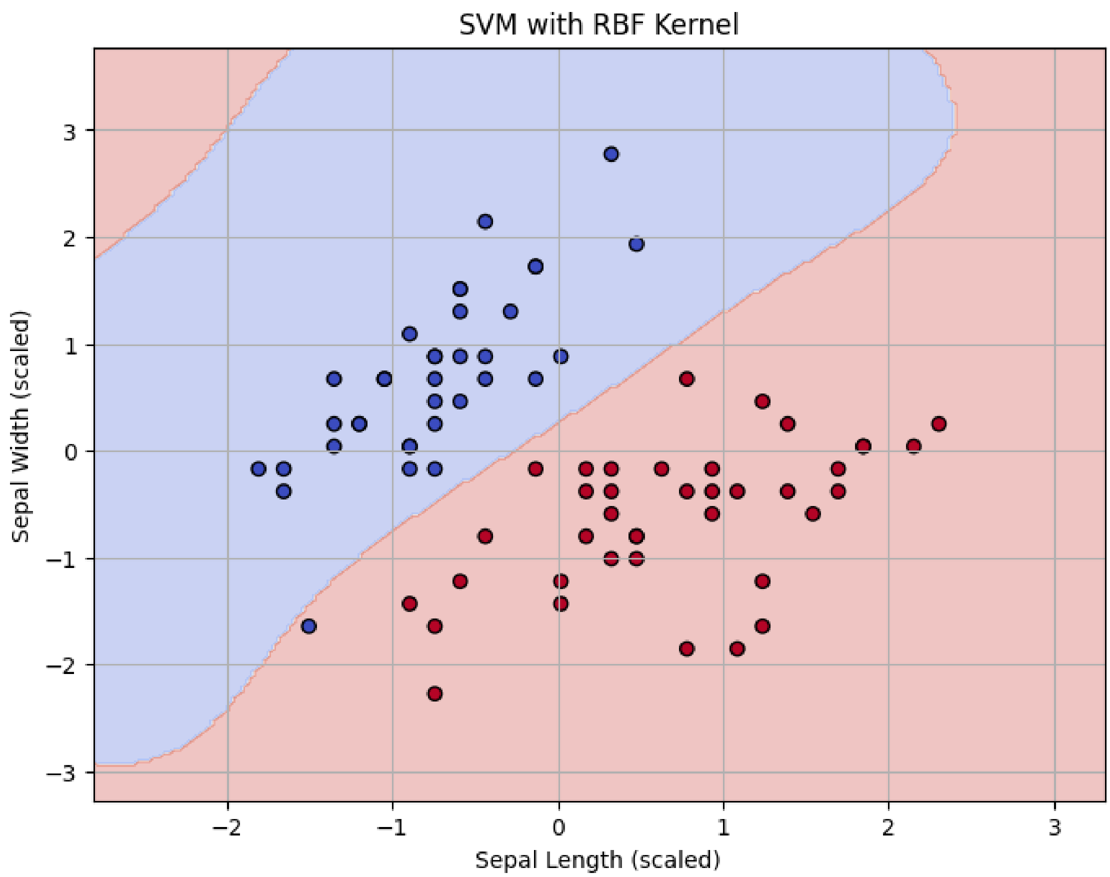
[62]:
```
# Linear kernel
svm_linear = SVC(kernel='linear')
svm_linear.fit(X_train_scaled, y_train)

# RBF kernel
svm_rbf = SVC(kernel='rbf')
svm_rbf.fit(X_train_scaled, y_train)

# Accuracy
acc_linear = accuracy_score(y_test, svm_linear.predict(X_test_scaled))
acc_rbf = accuracy_score(y_test, svm_rbf.predict(X_test_scaled))

print("Linear SVM Accuracy:", round(acc_linear * 100, 2), "%")
print("RBF SVM Accuracy:", round(acc_rbf * 100, 2), "%")
```

```
Linear SVM Accuracy: 100.0 %
RBF SVM Accuracy: 100.0 %
```

[63]:
```
import numpy as np

def plot_decision_boundary(model, title):
    x_min, x_max = X_train_scaled[:, 0].min() - 1, X_train_scaled[:, 0].max() +
 ↪1
    y_min, y_max = X_train_scaled[:, 1].min() - 1, X_train_scaled[:, 1].max() +
 ↪1
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 200),
                         np.linspace(y_min, y_max, 200))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
```

```
    plt.figure(figsize=(8, 6))
    plt.contourf(xx, yy, Z, alpha=0.3, cmap=plt.cm.coolwarm)
    plt.scatter(X_train_scaled[:, 0], X_train_scaled[:, 1], c=y_train, cmap=plt.
↪cm.coolwarm, edgecolors='k')
    plt.title(title)
    plt.xlabel("Sepal Length (scaled)")
    plt.ylabel("Sepal Width (scaled)")
    plt.grid(True)
    plt.show()

# Plot both
plot_decision_boundary(svm_linear, "SVM with Linear Kernel")
plot_decision_boundary(svm_rbf, "SVM with RBF Kernel")
```





Q19.Regularization Techniques

Use Lasso and Ridge regression on a dataset. Analyze how they handle multicollinearity and reduce model complexity.

[65]:
```
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Lasso, Ridge
from sklearn.metrics import mean_squared_error

# Load the California housing dataset
housing = fetch_california_housing()
X = pd.DataFrame(housing.data, columns=housing.feature_names)
y = housing.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

```python
# Lasso Regression
lasso = Lasso(alpha=0.1)  # Regularization parameter
lasso.fit(X_train, y_train)

# Ridge Regression
ridge = Ridge(alpha=0.1)  # Regularization parameter
ridge.fit(X_train, y_train)

# Predictions
y_pred_lasso = lasso.predict(X_test)
y_pred_ridge = ridge.predict(X_test)

# Evaluate the models
lasso_rmse = np.sqrt(mean_squared_error(y_test, y_pred_lasso))
ridge_rmse = np.sqrt(mean_squared_error(y_test, y_pred_ridge))

print(f"Lasso RMSE: {lasso_rmse}")
print(f"Ridge RMSE: {ridge_rmse}")

# Compare the coefficients
print("\nLasso Coefficients:")
print(lasso.coef_)

print("\nRidge Coefficients:")
print(ridge.coef_)
```

```
Lasso RMSE: 0.7832697618354822
Ridge RMSE: 0.7455754517896762

Lasso Coefficients:
[ 3.92693362e-01  1.50810624e-02 -0.00000000e+00  0.00000000e+00
  1.64168387e-05 -3.14918929e-03 -1.14291203e-01 -9.93076483e-02]

Ridge Coefficients:
[ 4.48658477e-01  9.72442833e-03 -1.23292361e-01  7.82971747e-01
 -2.02924019e-06 -3.52627239e-03 -4.19791946e-01 -4.33705352e-01]
```

Q20.Introduction to Neural Networks

Build a simple Artificial Neural Network (ANN) to classify data. Use optimization algorithms (Gradient Descent, SGD) and visualize the loss during training.

[67]:
```
!pip install torch torchvision matplotlib
```

```
Collecting torch
  Downloading torch-2.6.0-cp313-cp313-win_amd64.whl.metadata (28 kB)
Collecting torchvision
  Downloading torchvision-0.21.0-cp313-cp313-win_amd64.whl.metadata (6.3 kB)
```

```
Requirement already satisfied: matplotlib in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (3.10.1)
Collecting filelock (from torch)
  Downloading filelock-3.18.0-py3-none-any.whl.metadata (2.9 kB)
Requirement already satisfied: typing-extensions>=4.10.0 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from torch)
(4.13.2)
Collecting networkx (from torch)
  Downloading networkx-3.4.2-py3-none-any.whl.metadata (6.3 kB)
Requirement already satisfied: jinja2 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from torch)
(3.1.6)
Collecting fsspec (from torch)
  Downloading fsspec-2025.3.2-py3-none-any.whl.metadata (11 kB)
Requirement already satisfied: setuptools in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from torch)
(78.1.0)
Collecting sympy==1.13.1 (from torch)
  Downloading sympy-1.13.1-py3-none-any.whl.metadata (12 kB)
Collecting mpmath<1.4,>=1.1.0 (from sympy==1.13.1->torch)
  Downloading mpmath-1.3.0-py3-none-any.whl.metadata (8.6 kB)
Requirement already satisfied: numpy in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from
torchvision) (2.2.4)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from
torchvision) (11.1.0)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from
matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from
matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from
matplotlib) (4.57.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from
matplotlib) (1.4.8)
Requirement already satisfied: packaging>=20.0 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from
matplotlib) (24.2)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from
matplotlib) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from
matplotlib) (2.9.0.post0)
```

```
Requirement already satisfied: six>=1.5 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from python-
dateutil>=2.7->matplotlib) (1.17.0)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\shinde
ankita\appdata\local\programs\python\python313\lib\site-packages (from
jinja2->torch) (3.0.2)
Downloading torch-2.6.0-cp313-cp313-win_amd64.whl (204.1 MB)
   ---------------------------------------- 0.0/204.1 MB ? eta -:--:--
   ---------------------------------------- 0.5/204.1 MB 4.9 MB/s eta 0:00:42
   ---------------------------------------- 2.1/204.1 MB 7.2 MB/s eta 0:00:29
    --------------------------------------- 3.9/204.1 MB 7.6 MB/s eta 0:00:27
   - -------------------------------------- 5.2/204.1 MB 7.3 MB/s eta 0:00:28
   - -------------------------------------- 6.8/204.1 MB 7.5 MB/s eta 0:00:27
   - -------------------------------------- 8.7/204.1 MB 7.7 MB/s eta 0:00:26
   -- ------------------------------------- 10.2/204.1 MB 7.6 MB/s eta 0:00:26
   -- ------------------------------------- 11.8/204.1 MB 7.7 MB/s eta 0:00:25
   -- ------------------------------------- 13.6/204.1 MB 7.8 MB/s eta 0:00:25
   -- ------------------------------------- 15.2/204.1 MB 7.8 MB/s eta 0:00:25
   --- ------------------------------------ 17.0/204.1 MB 7.8 MB/s eta 0:00:24
   --- ------------------------------------ 18.6/204.1 MB 7.9 MB/s eta 0:00:24
   ---- ----------------------------------- 20.4/204.1 MB 7.9 MB/s eta 0:00:24
   ---- ----------------------------------- 22.3/204.1 MB 8.0 MB/s eta 0:00:23
   ---- ----------------------------------- 24.1/204.1 MB 8.0 MB/s eta 0:00:23
   ---- ----------------------------------- 25.4/204.1 MB 7.9 MB/s eta 0:00:23
   ----- ---------------------------------- 27.3/204.1 MB 7.9 MB/s eta 0:00:23
   ----- ---------------------------------- 28.8/204.1 MB 7.9 MB/s eta 0:00:23
   ------ --------------------------------- 30.7/204.1 MB 7.9 MB/s eta 0:00:22
   ------ --------------------------------- 32.5/204.1 MB 8.0 MB/s eta 0:00:22
   ------ --------------------------------- 34.1/204.1 MB 8.0 MB/s eta 0:00:22
   ------ --------------------------------- 35.7/204.1 MB 8.0 MB/s eta 0:00:22
   ------- -------------------------------- 37.0/204.1 MB 7.9 MB/s eta 0:00:22
   ------- -------------------------------- 38.8/204.1 MB 7.9 MB/s eta 0:00:21
   ------- -------------------------------- 40.4/204.1 MB 7.9 MB/s eta 0:00:21
   -------- ------------------------------- 41.9/204.1 MB 7.9 MB/s eta 0:00:21
   -------- ------------------------------- 43.8/204.1 MB 7.9 MB/s eta 0:00:21
   -------- ------------------------------- 45.4/204.1 MB 7.9 MB/s eta 0:00:21
   --------- ------------------------------ 47.2/204.1 MB 7.9 MB/s eta 0:00:20
   --------- ------------------------------ 48.5/204.1 MB 7.9 MB/s eta 0:00:20
   --------- ------------------------------ 50.3/204.1 MB 7.9 MB/s eta 0:00:20
   ---------- ----------------------------- 51.4/204.1 MB 7.9 MB/s eta 0:00:20
   ---------- ----------------------------- 52.2/204.1 MB 7.7 MB/s eta 0:00:20
   ---------- ----------------------------- 53.5/204.1 MB 7.7 MB/s eta 0:00:20
   ---------- ----------------------------- 55.1/204.1 MB 7.6 MB/s eta 0:00:20
   ----------- ---------------------------- 56.6/204.1 MB 7.6 MB/s eta 0:00:20
   ----------- ---------------------------- 58.2/204.1 MB 7.6 MB/s eta 0:00:20
   ----------- ---------------------------- 60.0/204.1 MB 7.6 MB/s eta 0:00:19
   ------------ --------------------------- 61.9/204.1 MB 7.6 MB/s eta 0:00:19
   ------------ --------------------------- 63.4/204.1 MB 7.7 MB/s eta 0:00:19
```

```
------------ --------------------------- 65.0/204.1 MB 7.6 MB/s eta 0:00:19
------------- -------------------------- 66.8/204.1 MB 7.7 MB/s eta 0:00:18
------------- -------------------------- 68.4/204.1 MB 7.7 MB/s eta 0:00:18
------------- -------------------------- 70.0/204.1 MB 7.7 MB/s eta 0:00:18
-------------- ------------------------- 71.8/204.1 MB 7.7 MB/s eta 0:00:18
-------------- ------------------------- 73.7/204.1 MB 7.7 MB/s eta 0:00:17
-------------- ------------------------- 75.5/204.1 MB 7.7 MB/s eta 0:00:17
--------------- ------------------------ 76.8/204.1 MB 7.7 MB/s eta 0:00:17
--------------- ------------------------ 77.1/204.1 MB 7.7 MB/s eta 0:00:17
--------------- ------------------------ 78.1/204.1 MB 7.5 MB/s eta 0:00:17
--------------- ------------------------ 79.7/204.1 MB 7.5 MB/s eta 0:00:17
--------------- ------------------------ 81.5/204.1 MB 7.6 MB/s eta 0:00:17
---------------- ----------------------- 83.1/204.1 MB 7.6 MB/s eta 0:00:16
---------------- ----------------------- 84.7/204.1 MB 7.6 MB/s eta 0:00:16
---------------- ----------------------- 86.2/204.1 MB 7.6 MB/s eta 0:00:16
----------------- ---------------------- 87.8/204.1 MB 7.6 MB/s eta 0:00:16
----------------- ---------------------- 89.7/204.1 MB 7.6 MB/s eta 0:00:16
----------------- ---------------------- 91.5/204.1 MB 7.6 MB/s eta 0:00:15
------------------ --------------------- 93.1/204.1 MB 7.6 MB/s eta 0:00:15
------------------ --------------------- 94.9/204.1 MB 7.6 MB/s eta 0:00:15
------------------ --------------------- 96.5/204.1 MB 7.6 MB/s eta 0:00:15
------------------- -------------------- 98.3/204.1 MB 7.6 MB/s eta 0:00:14
------------------- -------------------- 99.9/204.1 MB 7.6 MB/s eta 0:00:14
------------------- -------------------- 101.7/204.1 MB 7.7 MB/s eta 0:00:14
-------------------- ------------------- 103.3/204.1 MB 7.7 MB/s eta 0:00:14
-------------------- ------------------- 104.9/204.1 MB 7.7 MB/s eta 0:00:13
-------------------- ------------------- 106.7/204.1 MB 7.7 MB/s eta 0:00:13
-------------------- ------------------- 107.0/204.1 MB 7.7 MB/s eta 0:00:13
--------------------- ------------------ 108.5/204.1 MB 7.6 MB/s eta 0:00:13
--------------------- ------------------ 110.4/204.1 MB 7.6 MB/s eta 0:00:13
--------------------- ------------------ 112.2/204.1 MB 7.6 MB/s eta 0:00:13
---------------------- ----------------- 114.0/204.1 MB 7.6 MB/s eta 0:00:12
---------------------- ----------------- 115.6/204.1 MB 7.6 MB/s eta 0:00:12
----------------------- ---------------- 117.4/204.1 MB 7.6 MB/s eta 0:00:12
----------------------- ---------------- 119.0/204.1 MB 7.6 MB/s eta 0:00:12
----------------------- ---------------- 120.8/204.1 MB 7.6 MB/s eta 0:00:11
------------------------ --------------- 122.7/204.1 MB 7.7 MB/s eta 0:00:11
------------------------ --------------- 124.3/204.1 MB 7.7 MB/s eta 0:00:11
------------------------ --------------- 125.8/204.1 MB 7.7 MB/s eta 0:00:11
------------------------ --------------- 127.4/204.1 MB 7.7 MB/s eta 0:00:11
------------------------- -------------- 129.2/204.1 MB 7.7 MB/s eta 0:00:10
------------------------- -------------- 130.8/204.1 MB 7.7 MB/s eta 0:00:10
------------------------- -------------- 132.6/204.1 MB 7.7 MB/s eta 0:00:10
-------------------------- ------------- 134.2/204.1 MB 7.7 MB/s eta 0:00:10
-------------------------- ------------- 135.8/204.1 MB 7.7 MB/s eta 0:00:09
-------------------------- ------------- 137.6/204.1 MB 7.7 MB/s eta 0:00:09
--------------------------- ------------ 139.5/204.1 MB 7.7 MB/s eta 0:00:09
--------------------------- ------------ 141.0/204.1 MB 7.7 MB/s eta 0:00:09
```

```
   --------------------------- ------------ 142.9/204.1 MB 7.7 MB/s eta 0:00:08
   ---------------------------- ----------- 144.4/204.1 MB 7.7 MB/s eta 0:00:08
   ---------------------------- ----------- 146.3/204.1 MB 7.7 MB/s eta 0:00:08
   ---------------------------- ----------- 147.6/204.1 MB 7.7 MB/s eta 0:00:08
   ----------------------------- ---------- 148.9/204.1 MB 7.7 MB/s eta 0:00:08
   ----------------------------- ---------- 150.2/204.1 MB 7.7 MB/s eta 0:00:08
   ----------------------------- ---------- 151.8/204.1 MB 7.7 MB/s eta 0:00:07
   ------------------------------ --------- 153.4/204.1 MB 7.7 MB/s eta 0:00:07
   ------------------------------ --------- 154.7/204.1 MB 7.7 MB/s eta 0:00:07
   ------------------------------ --------- 156.0/204.1 MB 7.6 MB/s eta 0:00:07
   ------------------------------ --------- 157.3/204.1 MB 7.6 MB/s eta 0:00:07
   ------------------------------ --------- 157.3/204.1 MB 7.6 MB/s eta 0:00:07
   ------------------------------- -------- 158.3/204.1 MB 7.5 MB/s eta 0:00:07
   ------------------------------- -------- 159.4/204.1 MB 7.5 MB/s eta 0:00:06
   ------------------------------- -------- 161.2/204.1 MB 7.5 MB/s eta 0:00:06
   ------------------------------- -------- 162.8/204.1 MB 7.5 MB/s eta 0:00:06
   -------------------------------- ------- 164.4/204.1 MB 7.5 MB/s eta 0:00:06
   -------------------------------- ------- 166.2/204.1 MB 7.5 MB/s eta 0:00:06
   -------------------------------- ------- 167.8/204.1 MB 7.5 MB/s eta 0:00:05
   --------------------------------- ------ 169.6/204.1 MB 7.5 MB/s eta 0:00:05
   --------------------------------- ------ 171.2/204.1 MB 7.5 MB/s eta 0:00:05
   --------------------------------- ------ 172.8/204.1 MB 7.5 MB/s eta 0:00:05
   ---------------------------------- ----- 174.6/204.1 MB 7.5 MB/s eta 0:00:04
   ---------------------------------- ----- 176.2/204.1 MB 7.5 MB/s eta 0:00:04
   ---------------------------------- ----- 178.0/204.1 MB 7.5 MB/s eta 0:00:04
   ----------------------------------- ---- 179.8/204.1 MB 7.5 MB/s eta 0:00:04
   ----------------------------------- ---- 181.7/204.1 MB 7.6 MB/s eta 0:00:03
   ----------------------------------- ---- 183.0/204.1 MB 7.5 MB/s eta 0:00:03
   ------------------------------------ --- 184.8/204.1 MB 7.6 MB/s eta 0:00:03
   ------------------------------------ --- 186.4/204.1 MB 7.6 MB/s eta 0:00:03
   ------------------------------------ --- 188.2/204.1 MB 7.6 MB/s eta 0:00:03
   ------------------------------------- -- 190.1/204.1 MB 7.6 MB/s eta 0:00:02
   ------------------------------------- -- 191.6/204.1 MB 7.6 MB/s eta 0:00:02
   ------------------------------------- -- 192.9/204.1 MB 7.6 MB/s eta 0:00:02
   -------------------------------------- - 194.8/204.1 MB 7.6 MB/s eta 0:00:02
   -------------------------------------- - 196.3/204.1 MB 7.6 MB/s eta 0:00:02
   -------------------------------------- - 198.2/204.1 MB 7.6 MB/s eta 0:00:01
   ---------------------------------------  200.0/204.1 MB 7.6 MB/s eta 0:00:01
   ---------------------------------------  201.9/204.1 MB 7.6 MB/s eta 0:00:01
   ---------------------------------------  203.2/204.1 MB 7.6 MB/s eta 0:00:01
   ---------------------------------------  203.9/204.1 MB 7.6 MB/s eta 0:00:01
   ---------------------------------------- 204.1/204.1 MB 7.5 MB/s eta 0:00:00
Downloading sympy-1.13.1-py3-none-any.whl (6.2 MB)
   ---------------------------------------- 0.0/6.2 MB ? eta -:--:--
   ---------- ----------------------------- 1.6/6.2 MB 8.5 MB/s eta 0:00:01
   ---------------------- ----------------- 3.4/6.2 MB 8.5 MB/s eta 0:00:01
   --------------------------------- ------ 5.2/6.2 MB 8.4 MB/s eta 0:00:01
   ---------------------------------------- 6.2/6.2 MB 7.8 MB/s eta 0:00:00
```

```
Downloading torchvision-0.21.0-cp313-cp313-win_amd64.whl (1.6 MB)
   ---------------------------------------- 0.0/1.6 MB ? eta -:--:--
   --------------------------------- ------ 1.3/1.6 MB 8.2 MB/s eta 0:00:01
   ---------------------------------------- 1.6/1.6 MB 7.0 MB/s eta 0:00:00
Downloading filelock-3.18.0-py3-none-any.whl (16 kB)
Downloading fsspec-2025.3.2-py3-none-any.whl (194 kB)
Downloading networkx-3.4.2-py3-none-any.whl (1.7 MB)
   ---------------------------------------- 0.0/1.7 MB ? eta -:--:--
   ------------------------------------ --- 1.6/1.7 MB 8.7 MB/s eta 0:00:01
   ---------------------------------------- 1.7/1.7 MB 8.3 MB/s eta 0:00:00
Downloading mpmath-1.3.0-py3-none-any.whl (536 kB)
   ---------------------------------------- 0.0/536.2 kB ? eta -:--:--
   ---------------------------------------- 536.2/536.2 kB 7.6 MB/s eta 0:00:00
Installing collected packages: mpmath, sympy, networkx, fsspec, filelock, torch,
torchvision
Successfully installed filelock-3.18.0 fsspec-2025.3.2 mpmath-1.3.0
networkx-3.4.2 sympy-1.13.1 torch-2.6.0 torchvision-0.21.0
```

[75]:
```python
import torch
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
import numpy as np
```

[76]:
```python
# Load the Iris dataset
data = load_iris()
X = data.data
y = data.target

# Standardize the dataset
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Convert to PyTorch tensors
X_train = torch.tensor(X_train, dtype=torch.float32)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.long)
y_test = torch.tensor(y_test, dtype=torch.long)
```

[77]:
```
class SimpleANN(nn.Module):
    def __init__(self):
        super(SimpleANN, self).__init__()
        # Define the layers
        self.layer1 = nn.Linear(4, 10)  # Input layer (4 features to 10 neurons␣
 ↪in the hidden layer)
        self.layer2 = nn.Linear(10, 3)  # Output layer (10 neurons to 3 output␣
 ↪classes)

    def forward(self, x):
        x = torch.relu(self.layer1(x))  # ReLU activation function for hidden␣
 ↪layer
        x = self.layer2(x)  # Output layer (logits)
        return x
```

[78]:
```
# Initialize the model
model = SimpleANN()

# Loss function (Cross Entropy for multi-class classification)
criterion = nn.CrossEntropyLoss()

# Optimizer: Gradient Descent (SGD can be used as well)
optimizer = optim.SGD(model.parameters(), lr=0.01)

# For visualization, we will store the loss values
train_losses = []
```

[79]:
```
epochs = 200
for epoch in range(epochs):
    model.train()  # Set the model to training mode
    optimizer.zero_grad()  # Zero the gradients from previous step

    # Forward pass
    outputs = model(X_train)
    loss = criterion(outputs, y_train)  # Calculate the loss

    # Backward pass and optimization
    loss.backward()
    optimizer.step()

    # Store the loss for visualization
    train_losses.append(loss.item())

    if (epoch + 1) % 20 == 0:
        print(f'Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}')
```

```
Epoch [20/200], Loss: 1.0315
```

```
Epoch [40/200], Loss: 0.9636
Epoch [60/200], Loss: 0.9111
Epoch [80/200], Loss: 0.8685
Epoch [100/200], Loss: 0.8327
Epoch [120/200], Loss: 0.8017
Epoch [140/200], Loss: 0.7744
Epoch [160/200], Loss: 0.7499
Epoch [180/200], Loss: 0.7278
Epoch [200/200], Loss: 0.7075
```

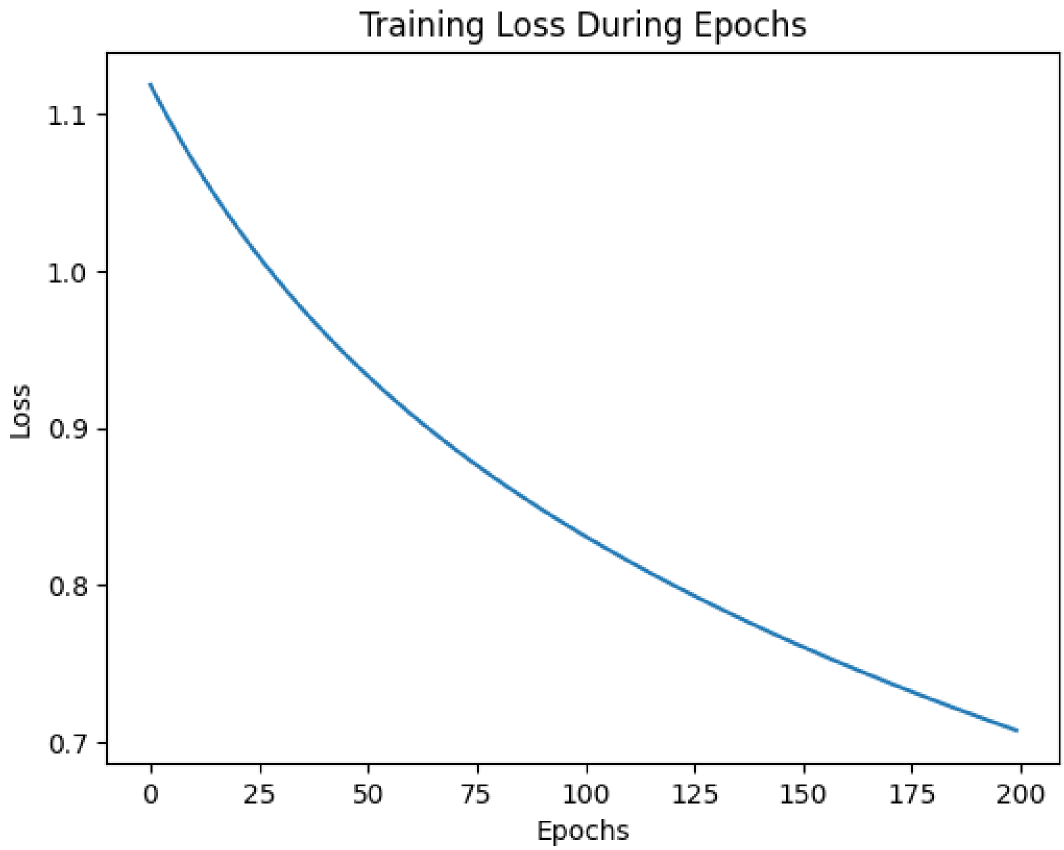
[80]:

```
# Set the model to evaluation mode
model.eval()

# Get the predictions on the test set
with torch.no_grad():
    outputs = model(X_test)
    _, predicted = torch.max(outputs, 1)  # Get the class with the highest␣
 ↪probability
    accuracy = accuracy_score(y_test, predicted)
    print(f'Accuracy on Test Set: {accuracy * 100:.2f}%')
```

```
Accuracy on Test Set: 73.33%
```

[81]:

```
# Plot the training loss
plt.plot(train_losses)
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training Loss During Epochs')
plt.show()
```


Training Loss During Epochs
Loss
1.1
1.0
0.9
0.8
0.7
0
25
50
75
100
125
150
175
200
Epochs

```
[ ]:
```

```
[ ]:
```