1. **Data Cleaning and Imputation Techniques: Load a dataset with missing values. Apply techniques like mean/mode/median imputation and compare the results.**

Code And Output:

```
# imputation_example.py


import pandas as pd

import numpy as np

from sklearn.impute import
SimpleImputer


# Step 1: Create a sample dataset

data = {

   'Age': [25, 27, np.nan, 29, 30, np.nan,
40],

   'Salary': [932, 54000, 58000, np.nan,
62000, 64000, np.nan],

   'Department': ['HR', 'IT', 'IT', np.nan,
'HR', 'Finance', np.nan]

}


df = pd.DataFrame(data)

print("Original Data with Missing
Values:\n")

print(df)

# Step 2: Mean Imputation for Age and
Salary

mean_imputer =
SimpleImputer(strategy='mean')

df_mean = df.copy()

df_mean[['Age', 'Salary']] =
mean_imputer.fit_transform(df_mean[['
Age', 'Salary']])

print("\nAfter Mean Imputation
(Numerical):\n")

print(df_mean)


# Step 3: Median Imputation for Age
and Salary

median_imputer =
SimpleImputer(strategy='median')

df_median = df.copy()

df_median[['Age', 'Salary']] =
median_imputer.fit_transform(df_media
n[['Age', 'Salary']])

print("\nAfter Median Imputation
(Numerical):\n")

print(df_median)
```

# Step 4: Mode Imputation for Department

mode_imputer = SimpleImputer(strategy='most_frequent')

df_mode = df.copy()

df_mode[['Department']] = mode_imputer.fit_transform(df_mode[['Department']])

print("\nAfter Mode Imputation (Categorical):\n")

print(df_mode)

# Optional: Combined example

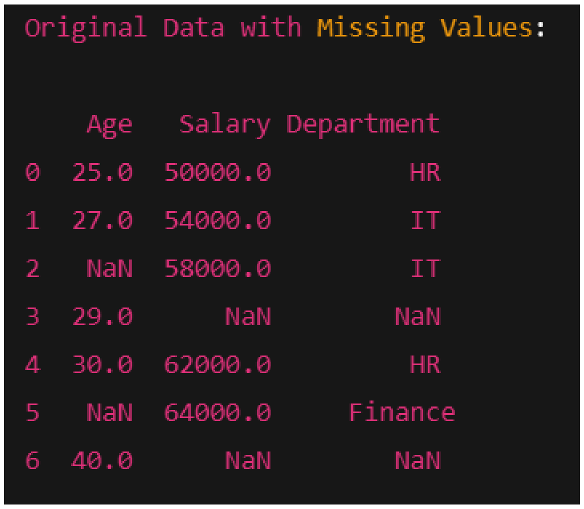
combined_df = df.copy()

combined_df[['Age', 'Salary']] = mean_imputer.fit_transform(combined_df[['Age', 'Salary']])

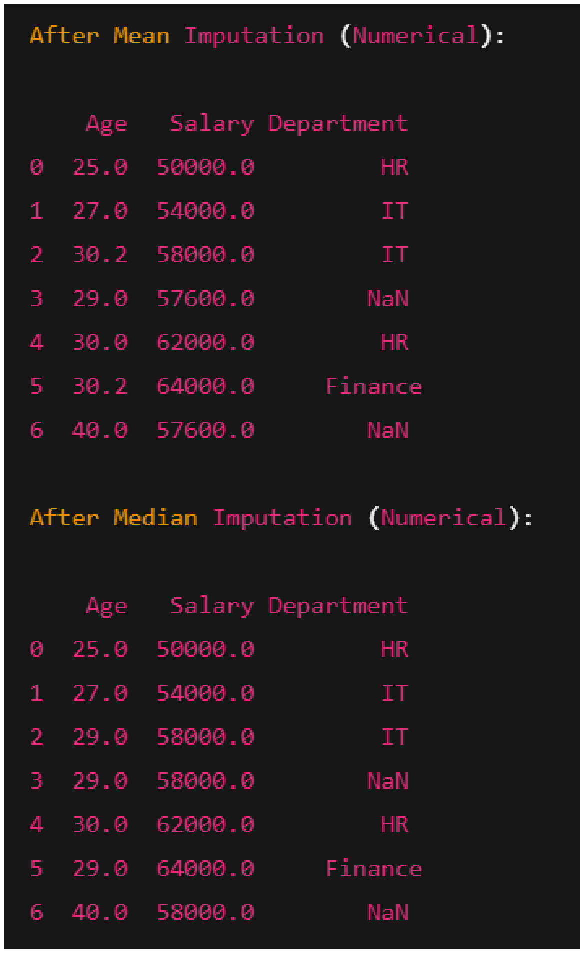
combined_df[['Department']] = mode_imputer.fit_transform(combined_df[['Department']])

print("\nAfter Combined Imputation:\n")

print(combined_df)



```
Original Data with Missing Values:

    Age   Salary Department
0  25.0  50000.0         HR
1  27.0  54000.0         IT
2   NaN  58000.0         IT
3  29.0      NaN        NaN
4  30.0  62000.0         HR
5   NaN  64000.0    Finance
6  40.0      NaN        NaN
```



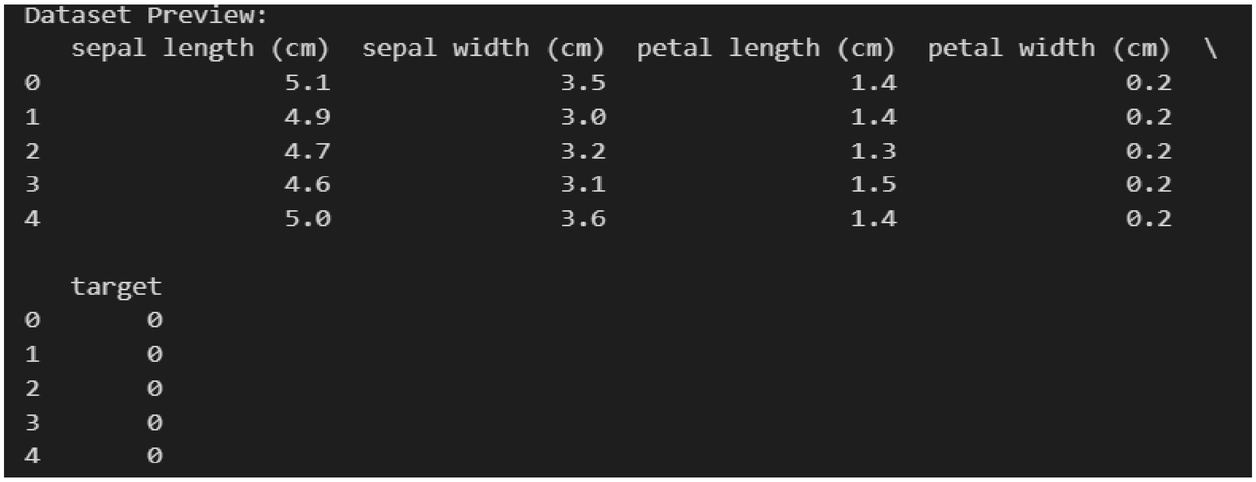
```
After Mean Imputation (Numerical):

    Age   Salary Department
0  25.0  50000.0         HR
1  27.0  54000.0         IT
2  30.2  58000.0         IT
3  29.0  57600.0        NaN
4  30.0  62000.0         HR
5  30.2  64000.0    Finance
6  40.0  57600.0        NaN

After Median Imputation (Numerical):

    Age   Salary Department
0  25.0  50000.0         HR
1  27.0  54000.0         IT
2  29.0  58000.0         IT
3  29.0  58000.0        NaN
4  30.0  62000.0         HR
5  29.0  64000.0    Finance
6  40.0  58000.0        NaN
```

2. **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.**

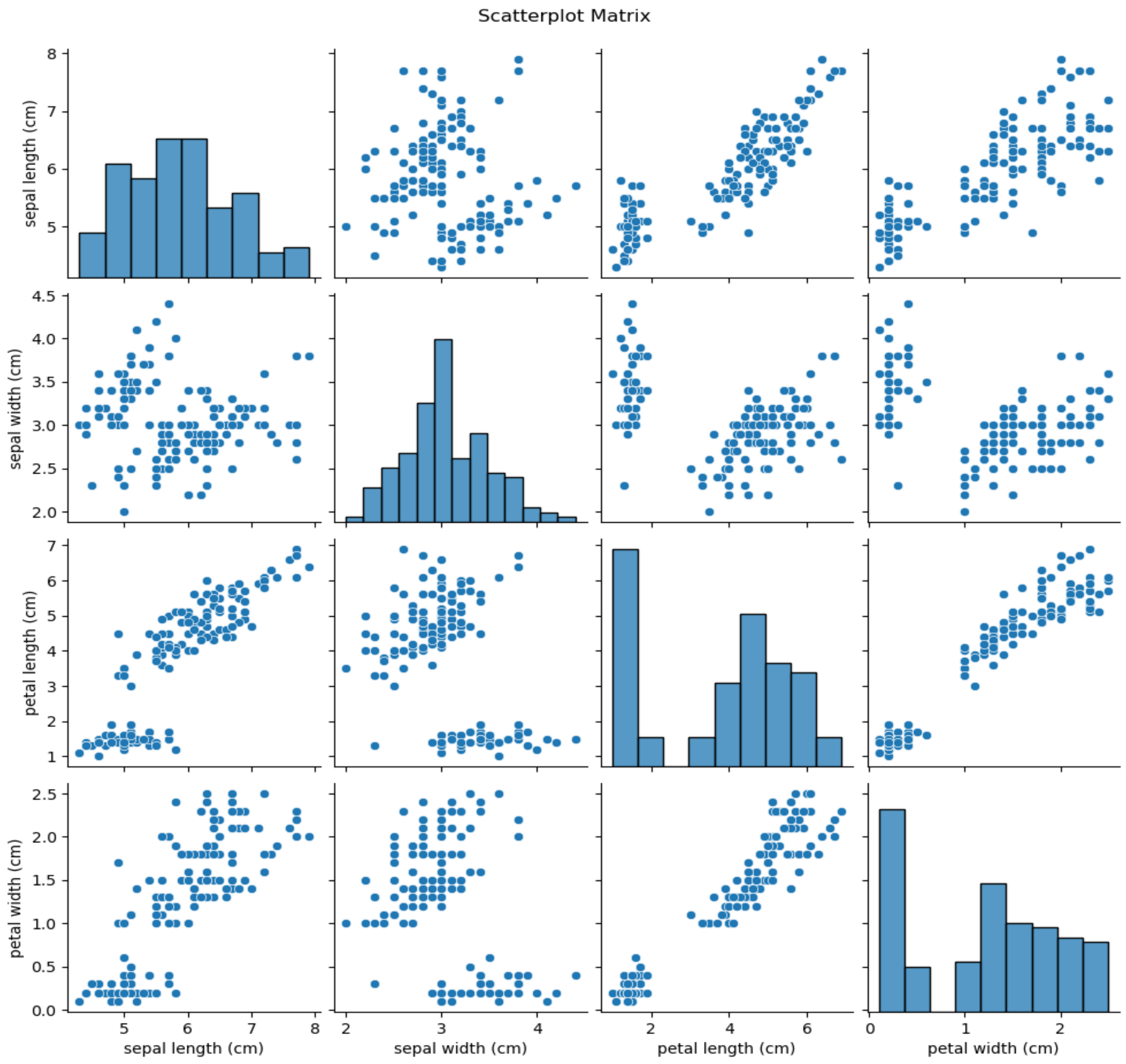
Code And Output:

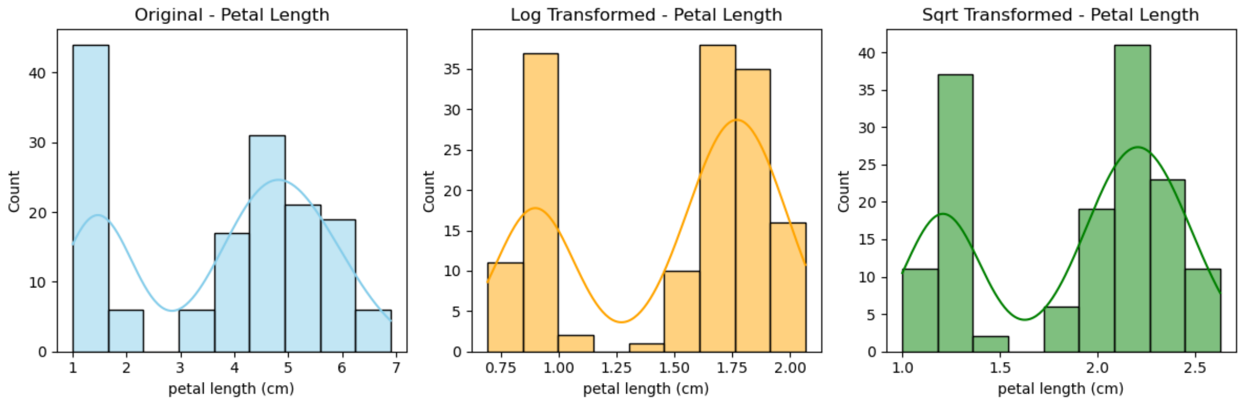
```
# data_analysis.py


import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import numpy as np

from sklearn.datasets import load_iris


# Step 1: Load the Iris dataset

iris = load_iris()

df = pd.DataFrame(iris.data,
columns=iris.feature_names)

df['target'] = iris.target

print("Dataset Preview:")

print(df.head())


# Step 2: Scatterplot matrix

sns.pairplot(df.iloc[:, :-1])  # Exclude
target

plt.suptitle("Scatterplot Matrix", y=1.02)

plt.show()


# Step 3: Correlation analysis

correlation_matrix = df.iloc[:, :-1].corr()

print("\nCorrelation Matrix:")

print(correlation_matrix)


# Visualize correlation matrix

sns.heatmap(correlation_matrix,
annot=True, cmap='coolwarm')

plt.title("Correlation Heatmap")

plt.show()


# Step 4: Apply transformations (Log and
Square Root)

df_log = df.copy()

df_sqrt = df.copy()


for col in df.columns[:-1]:  # Skip target
column

   df_log[col] = np.log(df[col] + 1)  #
Avoid log(0)

   df_sqrt[col] = np.sqrt(df[col])


# Step 5: Visualize transformation
effects for one column (example: petal
length)
```

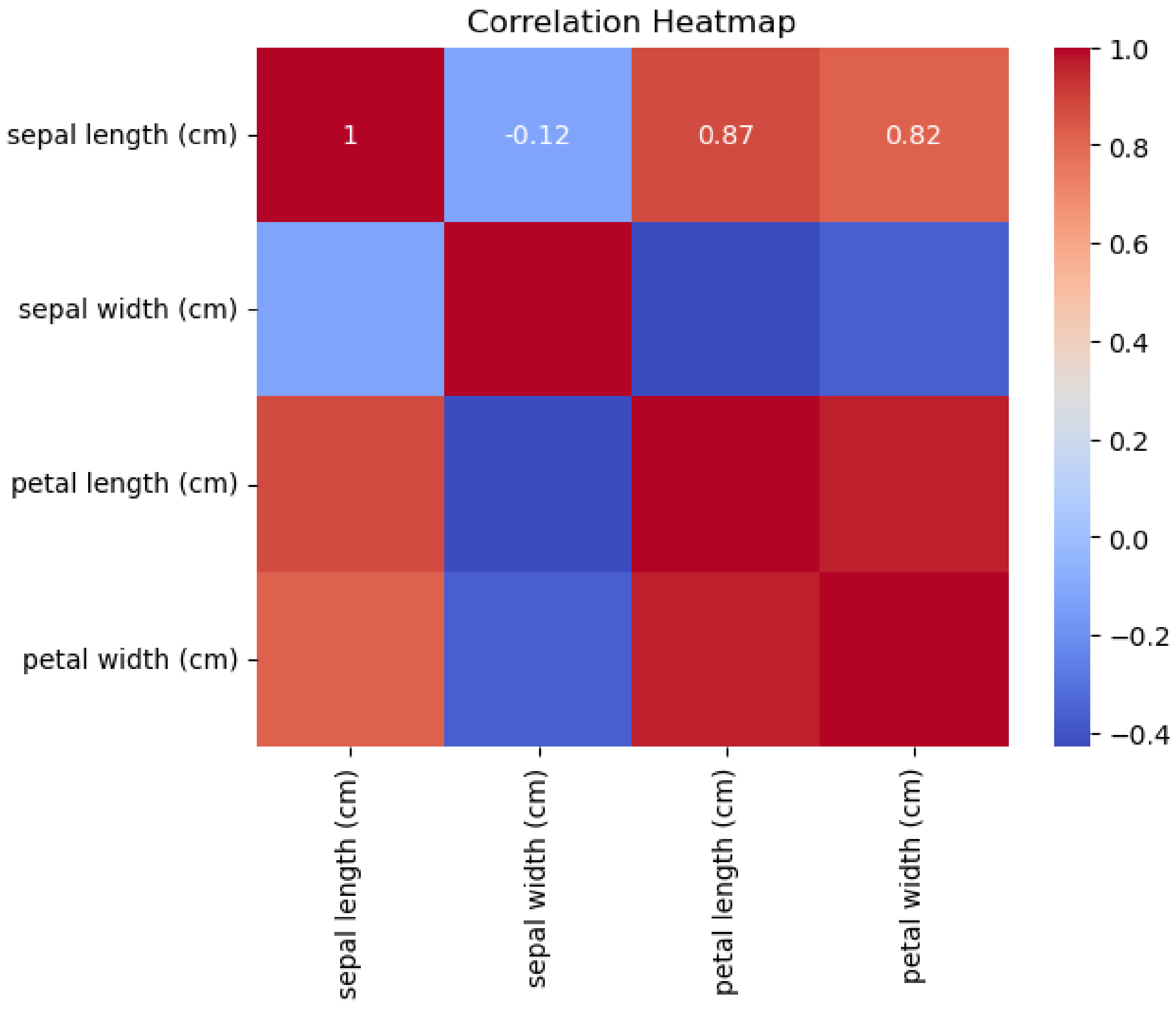
```
plt.figure(figsize=(12, 4))


plt.subplot(1, 3, 1)

sns.histplot(df['petal length (cm)'],
kde=True, color='skyblue')

plt.title("Original - Petal Length")


plt.subplot(1, 3, 2)

sns.histplot(df_log['petal length (cm)'],
kde=True, color='orange')

plt.title("Log Transformed - Petal
Length")


plt.subplot(1, 3, 3)

sns.histplot(df_sqrt['petal length (cm)'],
kde=True, color='green')

plt.title("Sqrt Transformed - Petal
Length")


plt.tight_layout()

plt.show()
```



```
Dataset Preview:
   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  
```




Original - Petal Length
Log Transformed - Petal Length
Sqrt Transformed - Petal Length
Count
petal length (cm)
0
10
20
30
40
1
2
3
4
5
6
7
0.75
1.00
1.25
1.50
1.75
2.00
1.0
1.5
2.0
2.5


Correlation Heatmap
sepal length (cm)
sepal width (cm)
petal length (cm)
petal width (cm)
1
-0.12
0.87
0.82
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

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

Code And Output :

```
import pandas as pd

from sklearn.preprocessing import
LabelEncoder, OneHotEncoder

from sklearn.linear_model import
LogisticRegression

from sklearn.model_selection
import train_test_split

from sklearn.metrics import
accuracy_score


# Step 1: Load data

df = pd.read_csv("dataaaa.csv")

print("Original Data:\n", df, "\n")


# Separate features and target

X = df[['Color', 'Size']]

y = df['Class']


# Step 2: Label Encoding

le_color = LabelEncoder()

le_size = LabelEncoder()


X_label_encoded = X.copy()

X_label_encoded['Color'] =
le_color.fit_transform(X['Color'])

X_label_encoded['Size'] =
le_size.fit_transform(X['Size'])


print("Label Encoded:\n",
X_label_encoded, "\n")


# Train model with Label Encoding

X_train, X_test, y_train, y_test =
train_test_split(X_label_encoded,
y, test_size=0.4, random_state=0)

model_label = LogisticRegression()

model_label.fit(X_train, y_train)

pred_label =
model_label.predict(X_test)
```

acc_label = accuracy_score(y_test, pred_label)

# Step 3: One-Hot Encoding

X_onehot = pd.get_dummies(X)

print("One-Hot Encoded:\n", X_onehot, "\n")

# Train model with One-Hot Encoding

X_train_oh, X_test_oh, y_train_oh, y_test_oh = train_test_split(X_onehot, y, test_size=0.4, random_state=0)

model_onehot = LogisticRegression()

model_onehot.fit(X_train_oh, y_train_oh)

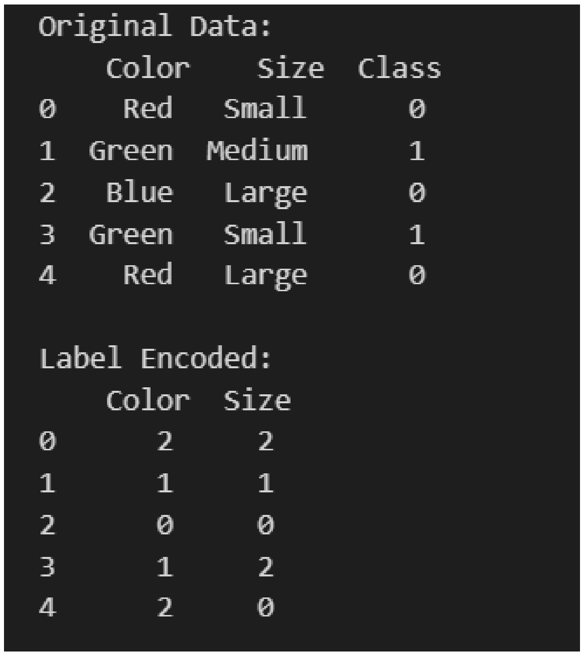
pred_onehot = model_onehot.predict(X_test_oh)

acc_onehot = accuracy_score(y_test_oh, pred_onehot)

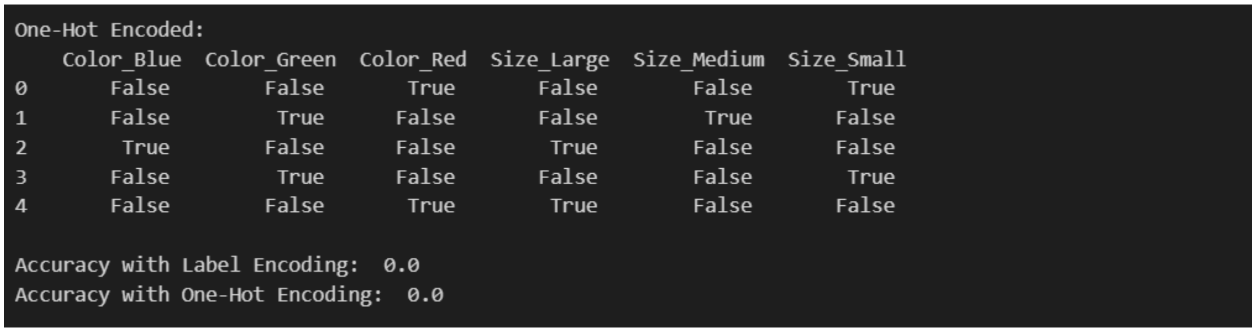
# Step 4: Print Results

print("Accuracy with Label Encoding: ", acc_label)

print("Accuracy with One-Hot Encoding: ", acc_onehot)



```
Original Data:
   Color    Size  Class
0    Red   Small      0
1  Green  Medium      1
2   Blue   Large      0
3  Green   Small      1
4    Red   Large      0

Label Encoded:
   Color  Size
0      2     2
1      1     1
2      0     0
3      1     2
4      2     0
```

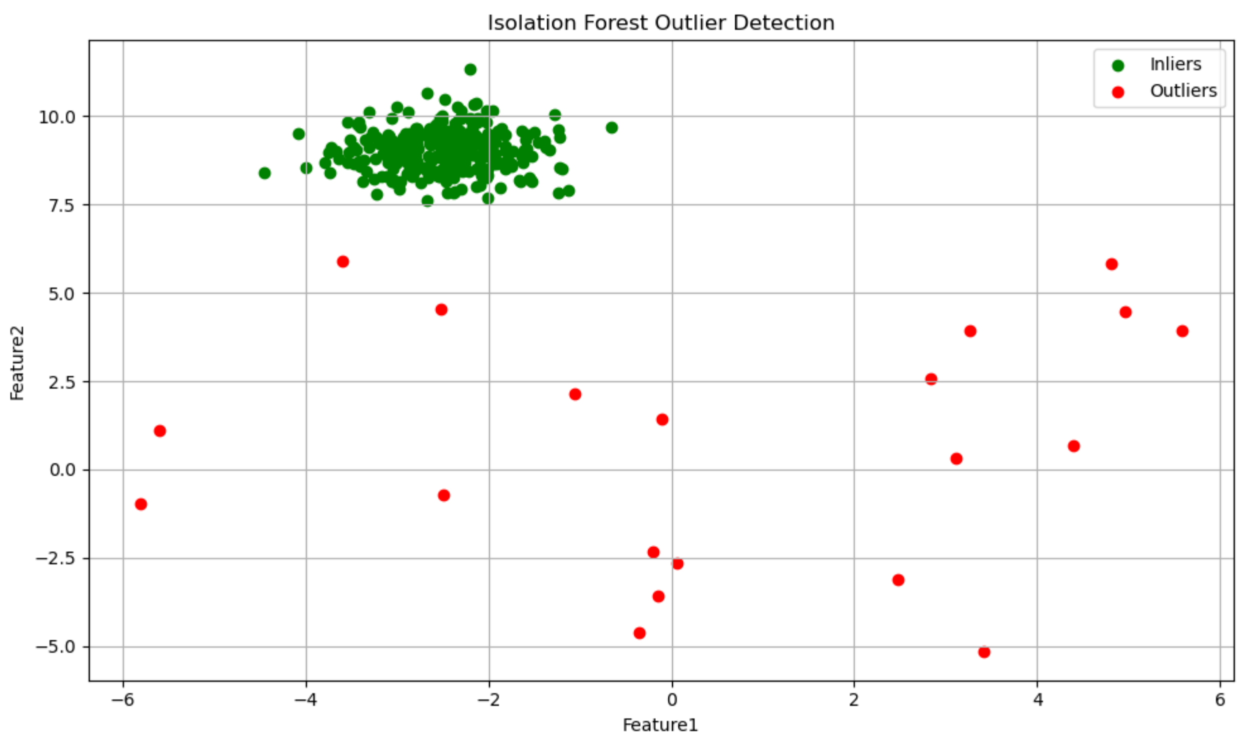

```
One-Hot Encoded:
   Color_Blue  Color_Green  Color_Red  Size_Large  Size_Medium  Size_Small
0       False        False       True       False        False        True
1       False         True      False       False         True       False
2        True        False      False        True        False       False
3       False         True      False       False        False        True
4       False        False       True        True        False       False

Accuracy with Label Encoding:  0.0
Accuracy with One-Hot Encoding:  0.0
```

4. Outlier Detection: Use the Isolation Forest algorithm to detect and visualize outliers in a dataset.

Code And Output:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.ensemble import
IsolationForest
from sklearn.datasets import
make_blobs

# Step 1: Generate sample data
X, _ =
make_blobs(n_samples=300,
centers=1, cluster_std=0.60,
random_state=42)

# Inject some outliers manually
outliers =
np.random.uniform(low=-6,
high=6, size=(20, 2))
X_with_outliers = np.vstack((X,
outliers))

# Step 2: Apply Isolation Forest
clf =
IsolationForest(contamination=
0.06, random_state=42)
clf.fit(X_with_outliers)
pred =
clf.predict(X_with_outliers)

# -1 = outlier, 1 = inlier
X_df =
pd.DataFrame(X_with_outliers,
columns=['Feature1',
'Feature2'])
X_df['Outlier'] = pred

# Step 3: Visualize
```

```
plt.figure(figsize=(10, 6))
plt.scatter(X_df[X_df['Outlier']
== 1]['Feature1'],
X_df[X_df['Outlier'] ==
1]['Feature2'],
            color='green',
label='Inliers')
plt.scatter(X_df[X_df['Outlier']
== -1]['Feature1'],
X_df[X_df['Outlier'] == -
1]['Feature2'],
            color='red',
label='Outliers')
plt.title('Isolation Forest Outlier
Detection')
plt.xlabel('Feature1')
plt.ylabel('Feature2')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# Step 4: Print output
num_outliers = sum(pred == -1)
print(f"\n🔍 Total Outliers
Detected: {num_outliers} out
of {len(pred)} samples")
```



5. **Predictive Power Score (PPS): Calculate the PPS for a dataset and interpret which variables are most predictive.**

Code And Output :

```
import pandas as pd
import seaborn as sns
import ppscore as pps
import matplotlib.pyplot as plt

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

# Calculate PPS for all column pairs
pps_matrix = pps.matrix(df)[['x', 'y',
'ppscore']]
pps_matrix =
pps_matrix[pps_matrix['ppscore'] > 0]  #
Filter out zero PPS

# Show top predictors
top_predictors =
pps_matrix.sort_values(by='ppscore',
ascending=False).head(10)
print("\n🔍 Top Predictive
Relationships:\n")
```

print(top_predictors.to_string(index=False))

# Visualize as heatmap

pps_heatmap = pps.predictors(df,
'survived')

plt.figure(figsize=(8, 4))

plt.barh(pps_heatmap['x'],
pps_heatmap['ppscore'], color='steelblue')

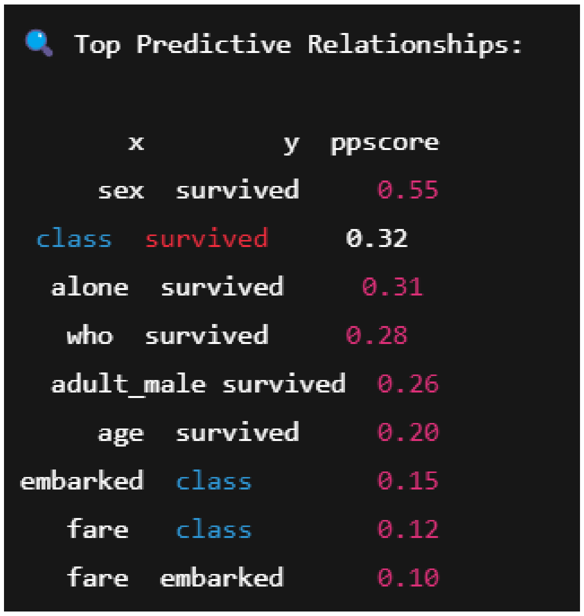
plt.xlabel("PPS")

plt.title("Predictive Power Score for
Predicting 'Survived'")

plt.grid(True)

plt.tight_layout()

plt.show()



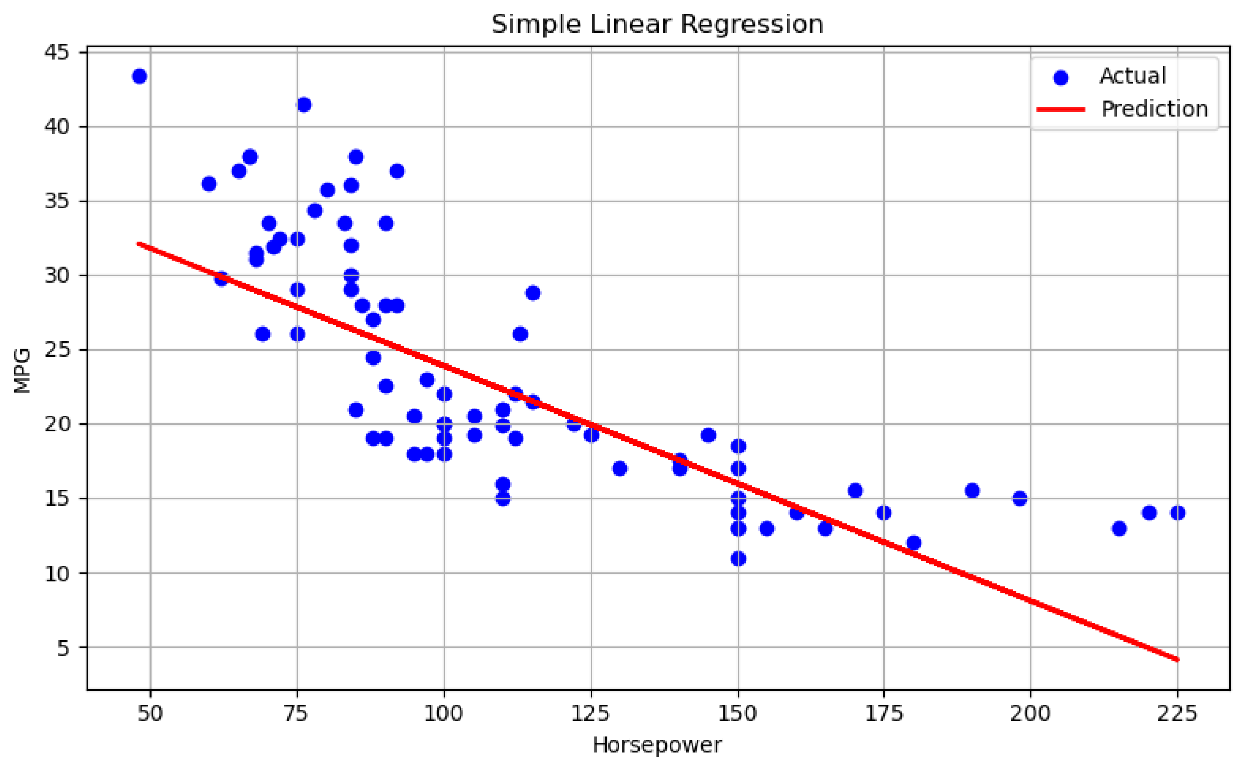
```
🔍 Top Predictive Relationships:

       x          y  ppscore
     sex   survived     0.55
 class   survived     0.32
   alone   survived     0.31
     who   survived     0.28
   adult_male survived  0.26
     age   survived     0.20
embarked   class        0.15
    fare   class        0.12
    fare   embarked     0.10
```

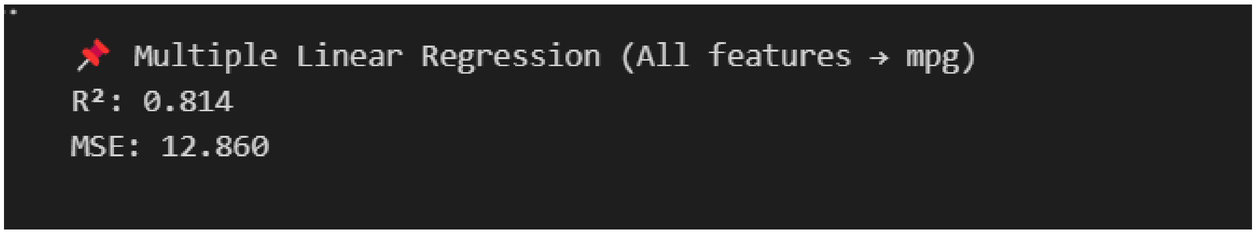
6. **Simple and Multiple Linear Regression: Implement simple and multiple linear regression on a dataset. Evaluate the model's performance using R-squared and mean squared error (MSE).**

Code And Output:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import
LinearRegression
from sklearn.metrics import
r2_score, mean_squared_error
from sklearn.model_selection
import train_test_split

# Load sample dataset
df =
sns.load_dataset("mpg").dropna()

# -------------------- SIMPLE LINEAR
REGRESSION --------------------
print("\n📌 Simple Linear
Regression (horsepower → mpg)")

# Feature and Target
X_simple = df[['horsepower']]
y = df['mpg']

# Train-test split
X_train_s, X_test_s, y_train_s,
y_test_s = train_test_split(X_simple,
y, test_size=0.2, random_state=1)

# Train the model
model_simple = LinearRegression()
model_simple.fit(X_train_s,
y_train_s)

# Predict
y_pred_s =
model_simple.predict(X_test_s)

# Evaluation
r2_s = r2_score(y_test_s, y_pred_s)
mse_s =
mean_squared_error(y_test_s,
y_pred_s)

print(f"R²: {r2_s:.3f}")
print(f"MSE: {mse_s:.3f}")

# Plotting
plt.figure(figsize=(8, 5))
plt.scatter(X_test_s, y_test_s,
color='blue', label='Actual')
plt.plot(X_test_s, y_pred_s,
color='red', linewidth=2,
label='Prediction')
plt.title("Simple Linear Regression")
plt.xlabel("Horsepower")
plt.ylabel("MPG")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
# -------------------- MULTIPLE LINEAR
REGRESSION --------------------
print("\n📌 Multiple Linear
Regression (All features → mpg)")

# Select numerical features only
X_multi =
df.select_dtypes(include=['float64',
'int64']).drop(columns=['mpg'])
y = df['mpg']

# Train-test split
X_train_m, X_test_m, y_train_m,
y_test_m = train_test_split(X_multi,
y, test_size=0.2, random_state=1)

# Train the model
model_multi = LinearRegression()
model_multi.fit(X_train_m,
y_train_m)

# Predict
y_pred_m =
model_multi.predict(X_test_m)

# Evaluation
r2_m = r2_score(y_test_m,
y_pred_m)
mse_m =
mean_squared_error(y_test_m,
y_pred_m)

print(f"R²: {r2_m:.3f}")
print(f"MSE: {mse_m:.3f}")
```





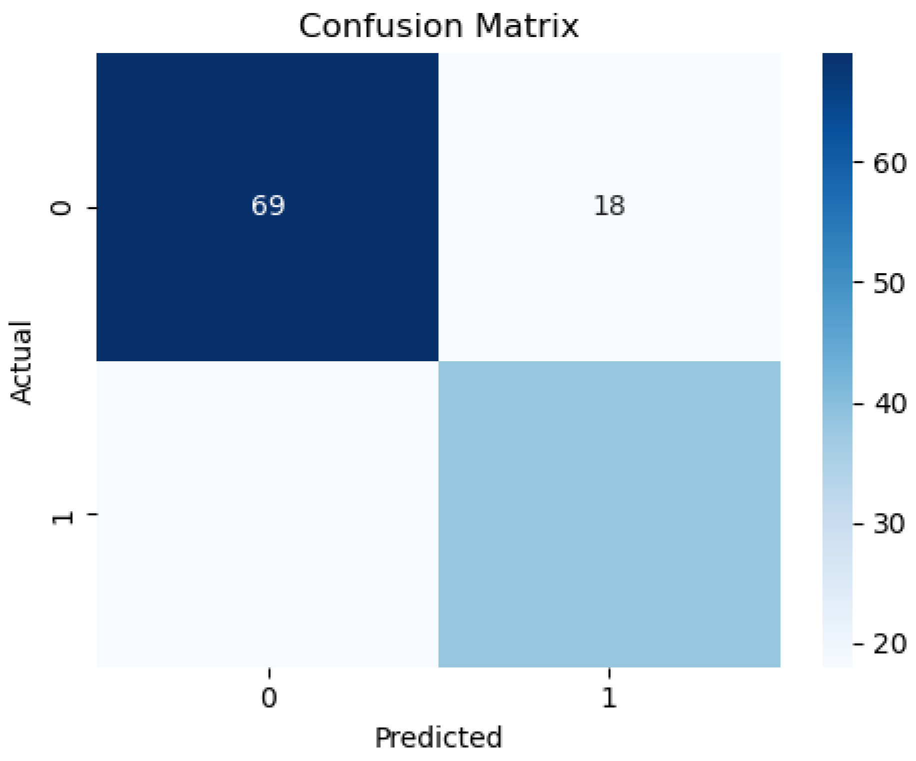
```
📌 Multiple Linear Regression (All features → mpg)
R²: 0.814
MSE: 12.860
```

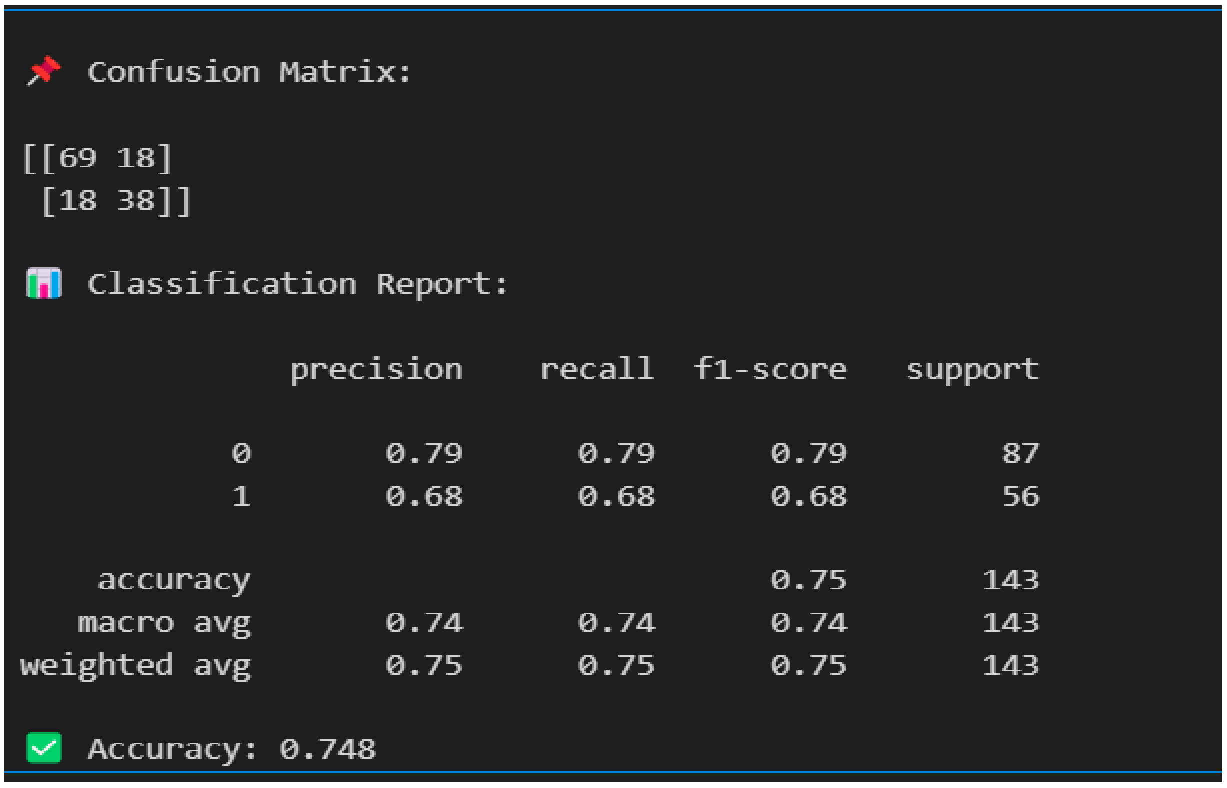
7. **Logistic Regression: Build a logistic regression model to classify binary outcomes (e.g., predicting if a customer will buy a product). Evaluate the model using confusion matrix metrics.**

Code And Output:

```
import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
```

```
# Load dataset

df = sns.load_dataset('titanic')


# Select relevant columns and drop missing values

df = df[['survived', 'sex', 'age', 'fare']].dropna()


# Convert categorical variable to numeric

df['sex'] = df['sex'].map({'male': 0, 'female': 1})


# Features and target

X = df[['sex', 'age', 'fare']]

y = df['survived']


# Train-test split

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


# Create and train model

model = LogisticRegression()

model.fit(X_train, y_train)


# Predict

y_pred = model.predict(X_test)


# Evaluation

print("\n📌 Confusion Matrix:\n")

print(confusion_matrix(y_test, y_pred))


print("\n📊 Classification Report:\n")

print(classification_report(y_test, y_pred))


print(f"✅ Accuracy: {accuracy_score(y_test, y_pred):.3f}")


# Optional: Visualize confusion matrix

import seaborn as sns

cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(5,4))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.tight_layout()

plt.show()
```





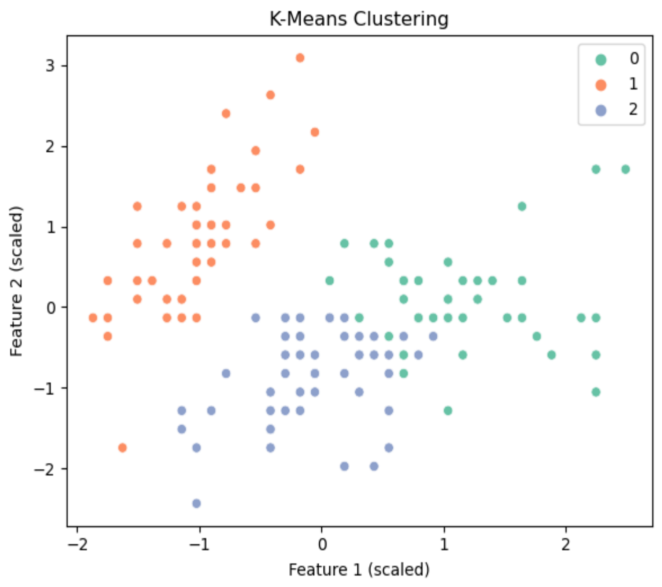
```
📌 Confusion Matrix:

[[69 18]
 [18 38]]

📊 Classification Report:

              precision    recall  f1-score   support

           0       0.79      0.79      0.79        87
           1       0.68      0.68      0.68        56

    accuracy                           0.75       143
   macro avg       0.74      0.74      0.74       143
weighted avg       0.75      0.75      0.75       143

✅ Accuracy: 0.748
```

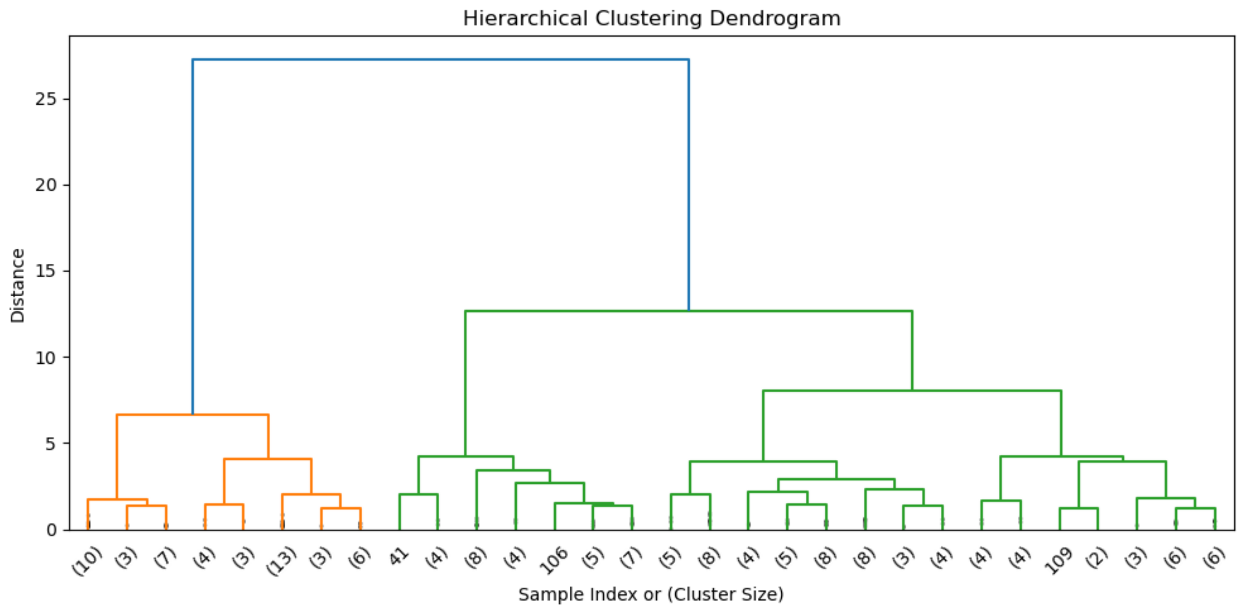
8. Clustering Techniques: Perform K-Means and hierarchical clustering on a dataset. Visualize the clusters and interpret the results.

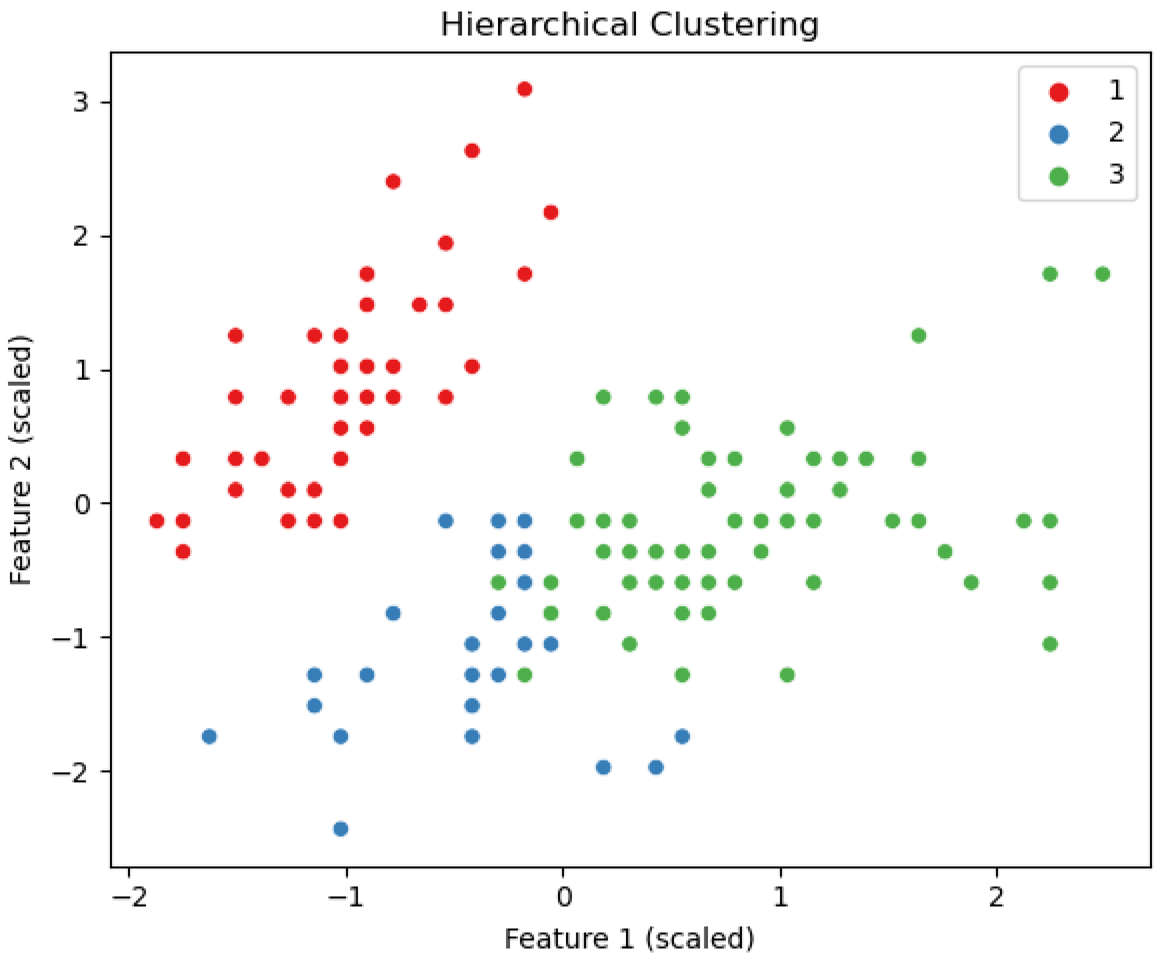
Code And Output :

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import
StandardScaler
from scipy.cluster.hierarchy import
linkage, dendrogram, fcluster
from sklearn.datasets import load_iris

# Load iris dataset
iris = load_iris()
df = pd.DataFrame(iris.data,
columns=iris.feature_names)

# Standardize the data
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df)

# -------------------- K-MEANS CLUSTERING
--------------------
kmeans = KMeans(n_clusters=3,
random_state=42)
kmeans_labels =
kmeans.fit_predict(scaled_data)

# Add cluster labels to original data
df['KMeans_Cluster'] = kmeans_labels

# Visualize K-Means clusters
plt.figure(figsize=(6, 5))
sns.scatterplot(x=scaled_data[:, 0],
y=scaled_data[:, 1], hue=kmeans_labels,
palette='Set2')
plt.title("K-Means Clustering")
plt.xlabel('Feature 1 (scaled)')
plt.ylabel('Feature 2 (scaled)')
plt.tight_layout()
plt.show()

# -------------------- HIERARCHICAL
CLUSTERING --------------------
linkage_matrix = linkage(scaled_data,
method='ward')

# Plot dendrogram
```

```
plt.figure(figsize=(10, 5))

dendrogram(linkage_matrix,
truncate_mode='lastp', p=30,
leaf_rotation=45., leaf_font_size=10.,
show_contracted=True)

plt.title("Hierarchical Clustering
Dendrogram")

plt.xlabel("Sample Index or (Cluster
Size)")

plt.ylabel("Distance")

plt.tight_layout()

plt.show()


# Assign cluster labels from dendrogram

hier_labels = fcluster(linkage_matrix,
t=3, criterion='maxclust')

df['Hierarchical_Cluster'] = hier_labels


# Visualize Hierarchical Clustering

plt.figure(figsize=(6, 5))

sns.scatterplot(x=scaled_data[:, 0],
y=scaled_data[:, 1], hue=hier_labels,
palette='Set1')

plt.title("Hierarchical Clustering")

plt.xlabel('Feature 1 (scaled)')

plt.ylabel('Feature 2 (scaled)')

plt.tight_layout()

plt.show()
```




Hierarchical Clustering Dendrogram
Distance
25
20
15
10
5
0
(10) (3) (7) (4) (3) (13) (3) (6) 41 (4) (8) (4) 106 (5) (7) (5) (8) (4) (5) (8) (8) (3) (4) (4) (4) 109 (2) (3) (6) (6)
Sample Index or (Cluster Size)

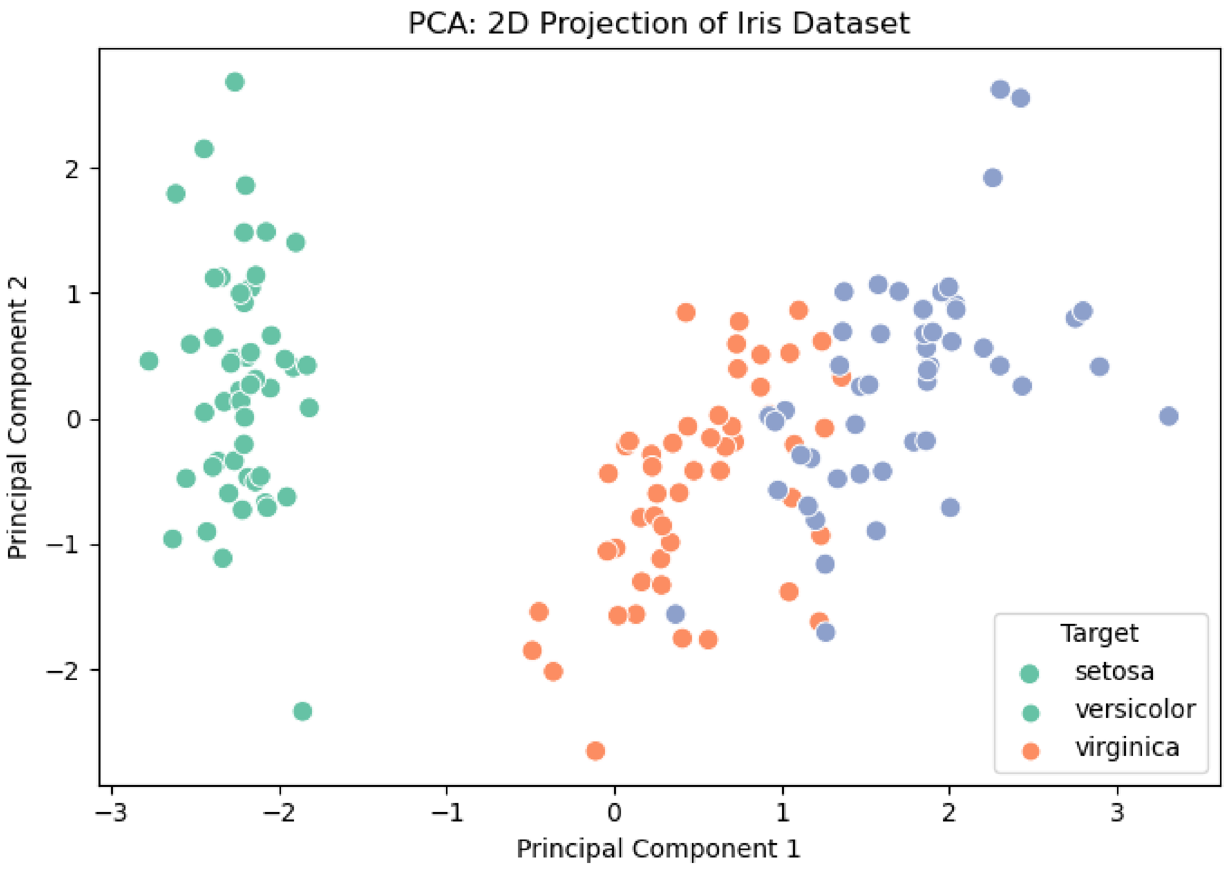

Hierarchical Clustering
Feature 2 (scaled)
Feature 1 (scaled)
3
2
1
0
−1
−2
0
1
2
● 1
● 2
● 3

9. **Principal Component Analysis (PCA): Apply PCA on a high- dimensional dataset. Reduce the dimensions and visualize the transformed data.**

Code And Output:

```
import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.datasets import load_iris

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA


# Load the dataset

iris = load_iris()

X = iris.data

y = iris.target

feature_names = iris.feature_names

target_names = iris.target_names


# Standardize the data

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)


# Apply PCA

pca = PCA(n_components=2)  # Reduce to 2 dimensions

X_pca = pca.fit_transform(X_scaled)


# Create a DataFrame with PCA results

pca_df = pd.DataFrame(data=X_pca, columns=['PC1', 'PC2'])

pca_df['Target'] = y


# Visualize the PCA results

plt.figure(figsize=(7, 5))

sns.scatterplot(data=pca_df, x='PC1', y='PC2', hue='Target', palette='Set2', s=70)

plt.title("PCA: 2D Projection of Iris Dataset")

plt.xlabel("Principal Component 1")

plt.ylabel("Principal Component 2")

plt.legend(title='Target', labels=target_names)

plt.tight_layout()

plt.show()


# Explained variance
```

```
print("🔍 Explained Variance Ratio:")
for i, ratio in enumerate(pca.explained_variance_ratio_):
print(f"PC{i+1}: {ratio:.2f}")
```

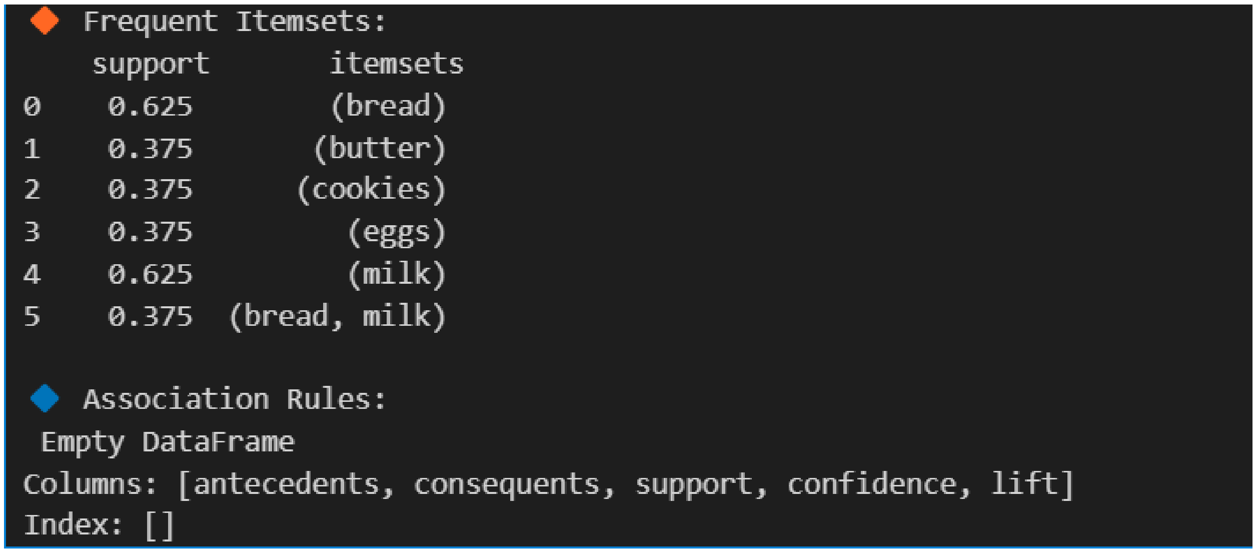


```
🔍 Explained Variance Ratio:
PC1: 0.73
PC2: 0.23
```

10. Market Basket Analysis: Implement Association Rule Mining using the Apriori algorithm. Identify frequent itemsets and generate association rules for a transactional dataset.

Code And Output:

```
import pandas as pd

from mlxtend.preprocessing import
TransactionEncoder

from mlxtend.frequent_patterns import
apriori, association_rules


# Step 1: Prepare dataset

transactions = [

   [‘curd', 'bread', 'eggs'],

   ['Curd', 'bread'],

   ['vadapav', 'cookies'],

   ['Kamlesh', 'butter'],

   ['milk', 'bread', 'butter', 'cookies'],

   ['eggs', 'bread'],

   ['milk', 'eggs'],

   ['cookies', 'butter'],

]


# Step 2: One-hot encode the
transactions

te = TransactionEncoder()

te_ary =
te.fit(transactions).transform(transactio
ns)

df = pd.DataFrame(te_ary,
columns=te.columns_)


# Step 3: Find frequent itemsets

frequent_itemsets = apriori(df,
min_support=0.3, use_colnames=True)


# Step 4: Generate association rules

rules =
association_rules(frequent_itemsets,
metric="lift", min_threshold=1.0)


# Display results

print("� Frequent Itemsets:\n",
frequent_itemsets)

print("\n� Association Rules:\n",
rules[['antecedents', 'consequents',
'support', 'confidence', 'lift']])
```

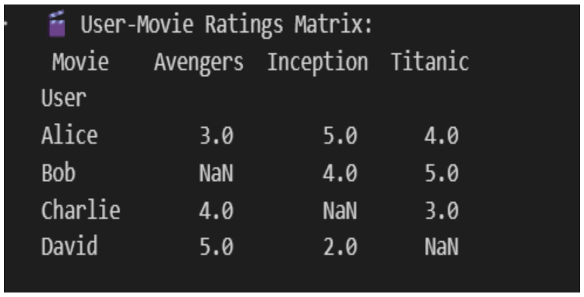


```
🔶 Frequent Itemsets:
    support       itemsets
0    0.625         (bread)
1    0.375        (butter)
2    0.375       (cookies)
3    0.375          (eggs)
4    0.625          (milk)
5    0.375  (bread, milk)

🔷 Association Rules:
 Empty DataFrame
Columns: [antecedents, consequents, support, confidence, lift]
Index: []
```

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

Code And Output:

```
import pandas as pd

# Sample user-movie rating matrix
data = {
   'User': ['Kamlesh', 'Kaivalya', 'Bitch', 'Bob', 'Bobade', 'Charlie', 'Charlie', 'David', 'David'],
   'Movie': ['Inception', 'Avengers', 'Titanic', 'Inception', 'Titanic', 'Avengers', 'Titanic', 'Inception', 'Avengers'],
   'Rating': [5, 3, 4, 4, 5, 4, 3, 2, 5]
}

df = pd.DataFrame(data)

# Pivot to create user-movie matrix
ratings = df.pivot_table(index='User', columns='Movie', values='Rating')
print("🎬 User-Movie Ratings Matrix:\n", ratings)
```



```
🎬 User-Movie Ratings Matrix:
 Movie    Avengers  Inception  Titanic
User
Alice          3.0        5.0      4.0
Bob            NaN        4.0      5.0
Charlie        4.0        NaN      3.0
David          5.0        2.0      NaN
```

Step 2: Apply Collaborative Filtering

from sklearn.metrics.pairwise import cosine_similarity

import numpy as np

# Fill NaNs with 0 for similarity calculations

ratings_filled = ratings.fillna(0)

# ----- USER-BASED Collaborative Filtering -----

user_similarity = cosine_similarity(ratings_filled)

user_sim_df = pd.DataFrame(user_similarity, index=ratings.index, columns=ratings.index)

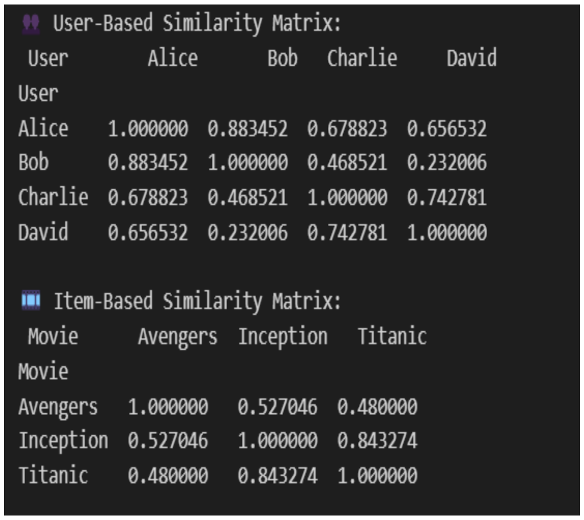
print("\n👥 User-Based Similarity Matrix:\n", user_sim_df)

# ----- ITEM-BASED Collaborative Filtering -----

item_similarity = cosine_similarity(ratings_filled.T)

item_sim_df = pd.DataFrame(item_similarity, index=ratings.columns, columns=ratings.columns)

print("\n🎞 Item-Based Similarity Matrix:\n", item_sim_df)



```
👥 User-Based Similarity Matrix:
 User        Alice       Bob   Charlie     David
User
Alice    1.000000  0.883452  0.678823  0.656532
Bob      0.883452  1.000000  0.468521  0.232006
Charlie  0.678823  0.468521  1.000000  0.742781
David    0.656532  0.232006  0.742781  1.000000

🎞 Item-Based Similarity Matrix:
 Movie      Avengers  Inception   Titanic
Movie
Avengers   1.000000   0.527046  0.480000
Inception  0.527046   1.000000  0.843274
Titanic    0.480000   0.843274  1.000000
```

🔍 **Step 3: Recommend Movies (Example for 'Alice')**

# Let's find similar users to Alice

similar_users = user_sim_df['Alice'].sort_values(ascending=False)[1:]

print("\nTop similar users to Alice:\n", similar_users)

# Recommend movies Alice hasn't rated based on similar users

alice_ratings = ratings.loc['Alice']

unrated_by_alice = alice_ratings[alice_ratings.isnull()]

# Predict using a weighted sum of ratings from similar users

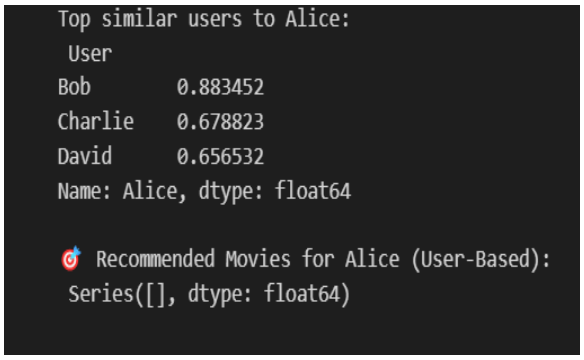
weighted_scores = ratings.loc[similar_users.index].T.dot(similar_users)

sum_of_weights = similar_users.sum()

predicted_ratings = weighted_scores / sum_of_weights

predicted_ratings = predicted_ratings[unrated_by_alice.index]

print("\n🎯 Recommended Movies for Alice (User-Based):\n", predicted_ratings.sort_values(ascending=False))

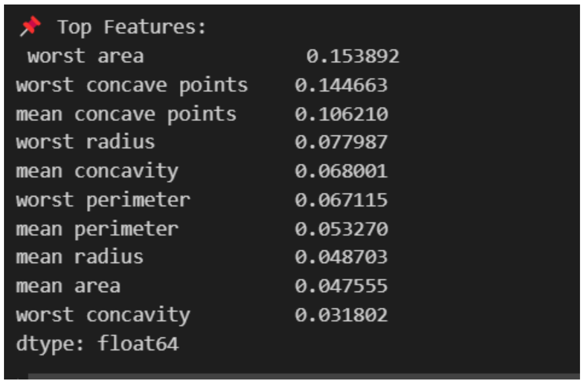


```
Top similar users to Alice:
 User
Bob        0.883452
Charlie    0.678823
David      0.656532
Name: Alice, dtype: float64

🎯 Recommended Movies for Alice (User-Based):
 Series([], dtype: float64)
```

12. Tree-Based Feature Engineering: Apply tree-based methods to rank feature importance in a dataset. Use the results to train a simplified model.

Code And Output:

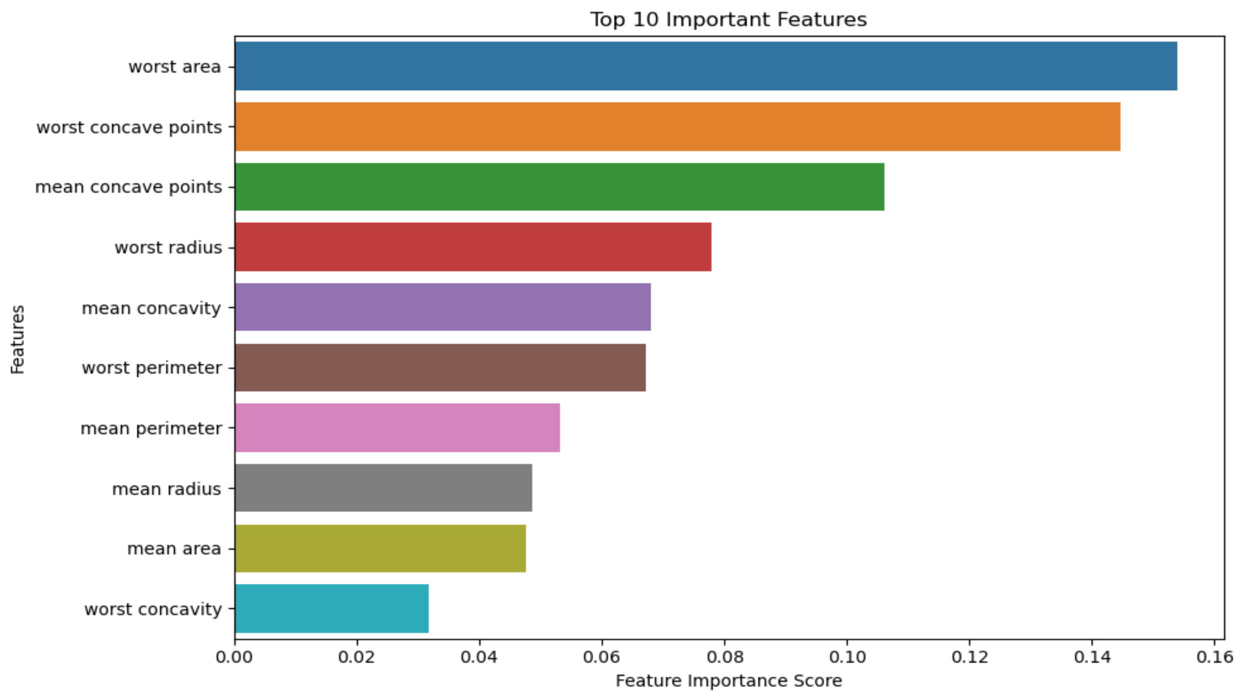
```
from sklearn.datasets import load_breast_cancer

from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns


# Load dataset

data = load_breast_cancer()

X = pd.DataFrame(data.data, columns=data.feature_names)

y = pd.Series(data.target)


print("📊 Dataset Shape:", X.shape)


# Split data

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

# Train Random Forest

model = RandomForestClassifier(n_estimators=100, random_state=42)

model.fit(X_train, y_train)


# Get feature importances

importances = model.feature_importances_

feature_ranks = pd.Series(importances, index=X.columns).sort_values(ascending=False)


print("\n📌 Top Features:\n", feature_ranks.head(10))
```

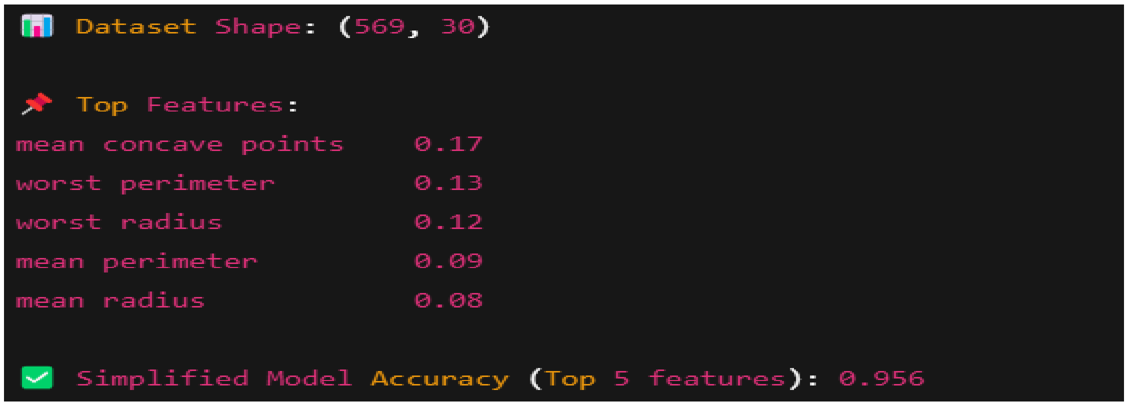


```
📌 Top Features:
 worst area              0.153892
worst concave points    0.144663
mean concave points     0.106210
worst radius            0.077987
mean concavity          0.068001
worst perimeter         0.067115
mean perimeter          0.053270
mean radius             0.048703
mean area               0.047555
worst concavity         0.031802
dtype: float64
```

Visualize Feature Importances

```
plt.show()

plt.figure(figsize=(10, 6))

sns.barplot(x=feature_ranks.head(10),
y=feature_ranks.head(10).index)

plt.title("Top 10 Important Features")

plt.xlabel("Feature Importance Score")

plt.ylabel("Features")

plt.tight_layout()
```





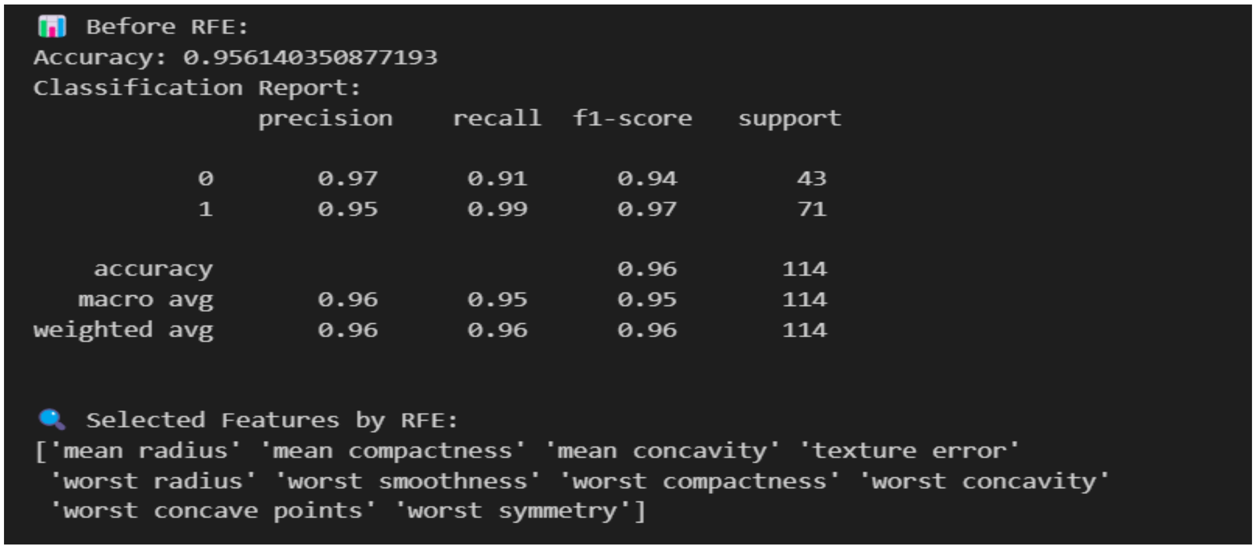
```
📊 Dataset Shape: (569, 30)

📌 Top Features:
mean concave points     0.17
worst perimeter         0.13
worst radius            0.12
mean perimeter          0.09
mean radius             0.08

✅ Simplified Model Accuracy (Top 5 features): 0.956
```

13. Recursive Feature Elimination (RFE): Perform feature selection using RFE. Evaluate the performance of a machine learning model before and after feature selection.

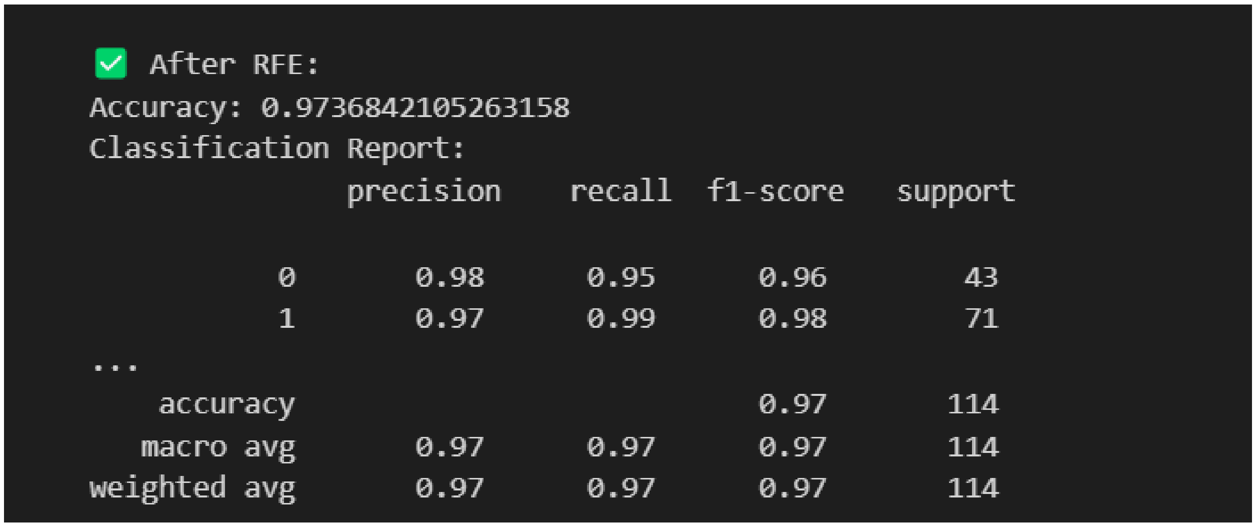
Code And Output:

```
from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
import pandas as pd

# Load dataset
data = load_breast_cancer()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.Series(data.target)

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

# Initialize Logistic Regression
model = LogisticRegression(max_iter=10000, solver='liblinear')

# -------- Before RFE --------
model.fit(X_train, y_train)
y_pred_full = model.predict(X_test)
full_acc = accuracy_score(y_test, y_pred_full)
print("📊 Full Model Accuracy (All features):", full_acc)

# -------- Apply RFE --------
rfe = RFE(estimator=model, n_features_to_select=10)
rfe.fit(X_train, y_train)

# Transform datasets
X_train_rfe = rfe.transform(X_train)
X_test_rfe = rfe.transform(X_test)

# -------- After RFE --------
```

```
model.fit(X_train_rfe, y_train)

y_pred_rfe = model.predict(X_test_rfe)

rfe_acc = accuracy_score(y_test,
y_pred_rfe)

print("📌 RFE Model Accuracy (Top 10
features):", rfe_acc)

# Show selected features

selected_features =
X.columns[rfe.support_]

print("\n✅ Selected Features by RFE:")

print(selected_features)
```



```
📊 Before RFE:
Accuracy: 0.956140350877193
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.91      0.94        43
           1       0.95      0.99      0.97        71

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114


🔍 Selected Features by RFE:
['mean radius' 'mean compactness' 'mean concavity' 'texture error'
 'worst radius' 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry']
```

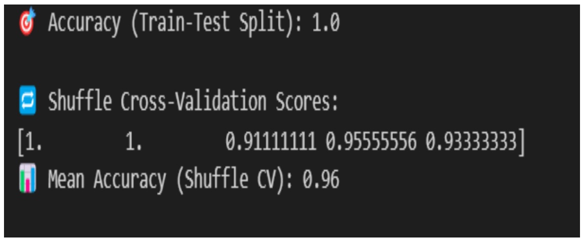


```
✅ After RFE:
Accuracy: 0.9736842105263158
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.95      0.96        43
           1       0.97      0.99      0.98        71
...
    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114
```

14. 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.

Code And Output:

```
from sklearn.datasets import load_iris

from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import train_test_split, ShuffleSplit, cross_val_score

from sklearn.metrics import accuracy_score


# Step 1: Load dataset

data = load_iris()

X = data.data

y = data.target


# Step 2: Train-Test Split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)


# Step 3: Train and Evaluate using Train-Test Split

model = LogisticRegression(max_iter=200)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("🎯 Accuracy (Train-Test Split):", accuracy_score(y_test, y_pred))


# Step 4: Shuffle Cross-Validation

shuffle_split = ShuffleSplit(n_splits=5, test_size=0.3, random_state=42)

cv_scores = cross_val_score(LogisticRegression(max_iter=200), X, y, cv=shuffle_split)


print("\n🔁 Shuffle Cross-Validation Scores:")

print(cv_scores)

print("📊 Mean Accuracy (Shuffle CV):", cv_scores.mean())
```

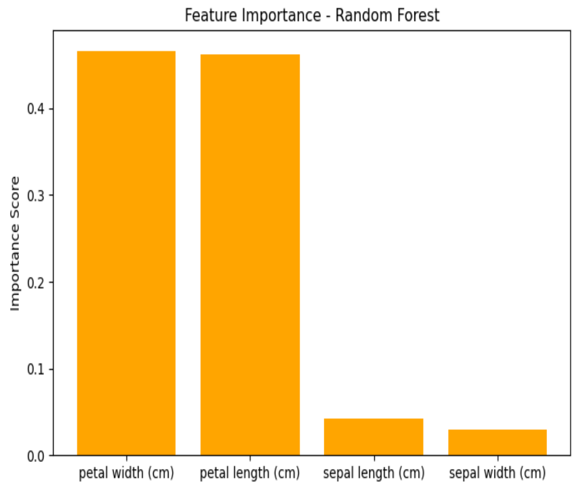


```
🎯 Accuracy (Train-Test Split): 1.0

🔁 Shuffle Cross-Validation Scores:
[1.         1.         0.91111111 0.95555556 0.93333333]
📊 Mean Accuracy (Shuffle CV): 0.96
```

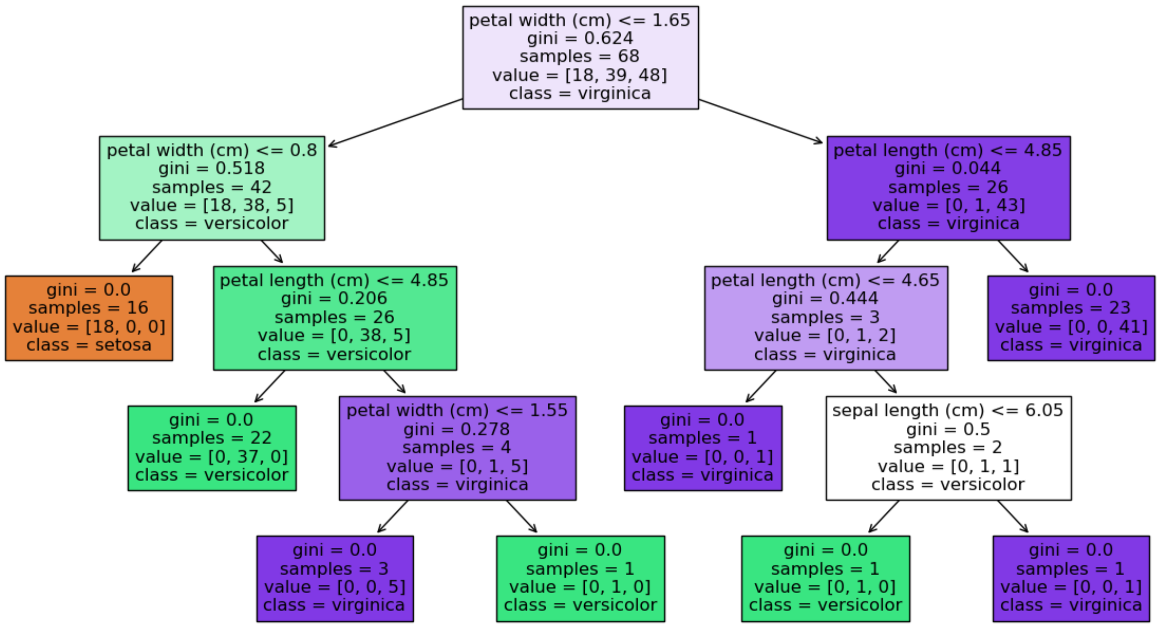
15. Bagging and Random Forest: Build and evaluate a Random Forest model. Visualize the decision trees and feature importance.

Code And Output :

```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier
from sklearn.tree import plot_tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load the dataset
iris = load_iris()
X = iris.data
y = iris.target
feature_names = iris.feature_names
class_names = iris.target_names

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# ---------------------------
# 🌲 Random Forest Classifier
# ---------------------------
rf = RandomForestClassifier(n_estimators=5, random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
print("✅ Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))

# ---------------------------
# 👜 Bagging Classifier (with Decision Trees)
# ---------------------------
from sklearn.tree import DecisionTreeClassifier

bagging = BaggingClassifier(base_estimator=Decisi
onTreeClassifier(), n_estimators=5,
random_state=42)

bagging.fit(X_train, y_train)

y_pred_bag = bagging.predict(X_test)

print("🎒 Bagging Accuracy:",
accuracy_score(y_test, y_pred_bag))


# ---------------------------

# 📊 Feature Importance from Random
Forest

# ---------------------------

importances = rf.feature_importances_

indices = np.argsort(importances)[::-1]


plt.figure(figsize=(8, 5))

plt.title("Feature Importance - Random
Forest")

plt.bar([feature_names[i] for i in
indices], importances[indices],
color='orange')

plt.ylabel("Importance Score")

plt.show()


# ---------------------------

# 🌳 Visualize one Decision Tree from
Random Forest

# ---------------------------

plt.figure(figsize=(15, 8))

plot_tree(rf.estimators_[0], filled=True,
feature_names=feature_names,
class_names=class_names)

plt.title("🌳 Sample Tree from Random
Forest")

plt.show()
```



□ Sample Tree from Random Forest

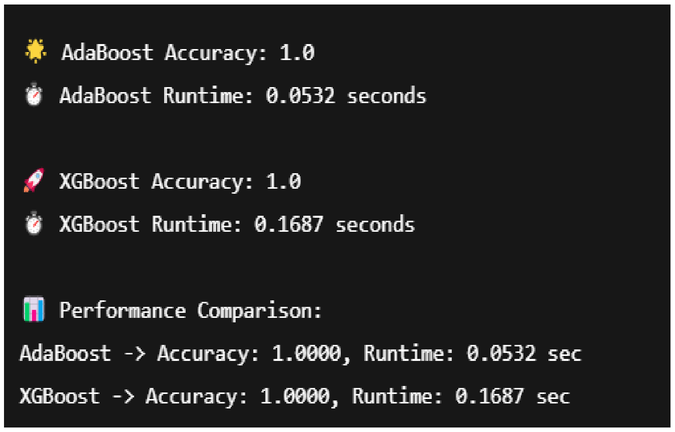


16 Boosting Methods: Implement AdaBoost and XGBoost on classification task. Compare their accuracy and runtime performance.

Code And Output:

```
import time
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.ensemble import AdaBoostClassifier
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

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

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# ------------------------------
# 💥 AdaBoost Classifier
# ------------------------------
start_ada = time.time()
ada_model = AdaBoostClassifier(n_estimators=50, random_state=42)
ada_model.fit(X_train, y_train)
ada_pred = ada_model.predict(X_test)
end_ada = time.time()

ada_accuracy = accuracy_score(y_test, ada_pred)
ada_time = end_ada - start_ada
print("💥 AdaBoost Accuracy:", ada_accuracy)
print("⏱️ AdaBoost Runtime:", ada_time, "seconds")

# ------------------------------
# 🚀 XGBoost Classifier
# ------------------------------
start_xgb = time.time()
xgb_model = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss', n_estimators=50, random_state=42)
```

```
xgb_model.fit(X_train, y_train)

xgb_pred = xgb_model.predict(X_test)

end_xgb = time.time()


xgb_accuracy = accuracy_score(y_test,
xgb_pred)

xgb_time = end_xgb - start_xgb

print("\n🚀 XGBoost Accuracy:",
xgb_accuracy)

print("⏱ XGBoost Runtime:", xgb_time,
"seconds")



# ------------------------------

# 📊 Performance Summary

# ------------------------------

print("\n📊 Performance Comparison:")

print(f"AdaBoost -> Accuracy:
{ada_accuracy:.4f}, Runtime:
{ada_time:.4f} sec")

print(f"XGBoost -> Accuracy:
{xgb_accuracy:.4f}, Runtime:
{xgb_time:.4f} sec")
```

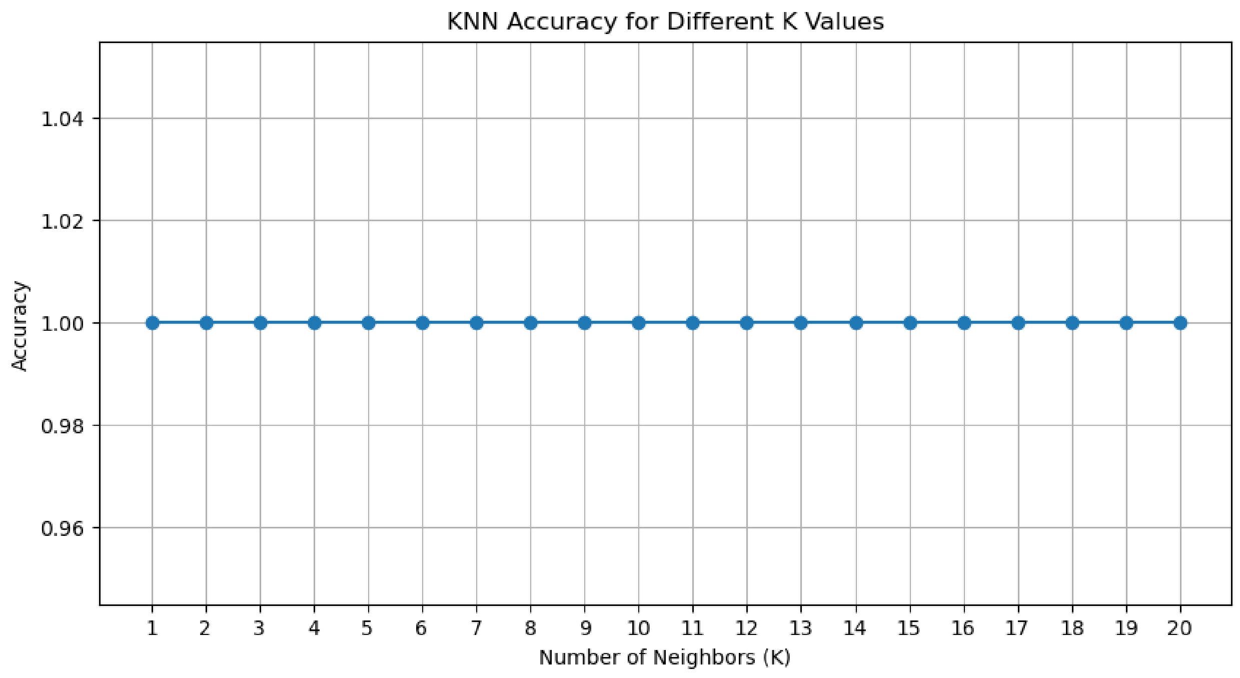


```
🌟 AdaBoost Accuracy: 1.0
⏱ AdaBoost Runtime: 0.0532 seconds

🚀 XGBoost Accuracy: 1.0
⏱ XGBoost Runtime: 0.1687 seconds

📊 Performance Comparison:
AdaBoost -> Accuracy: 1.0000, Runtime: 0.0532 sec
XGBoost -> Accuracy: 1.0000, Runtime: 0.1687 sec
```

17. K-Nearest Neighbors (KNN): Implement a KNN classifier for a classification task. Experiment with different values of K and analyze their impact on accuracy.

Code And OutPut:

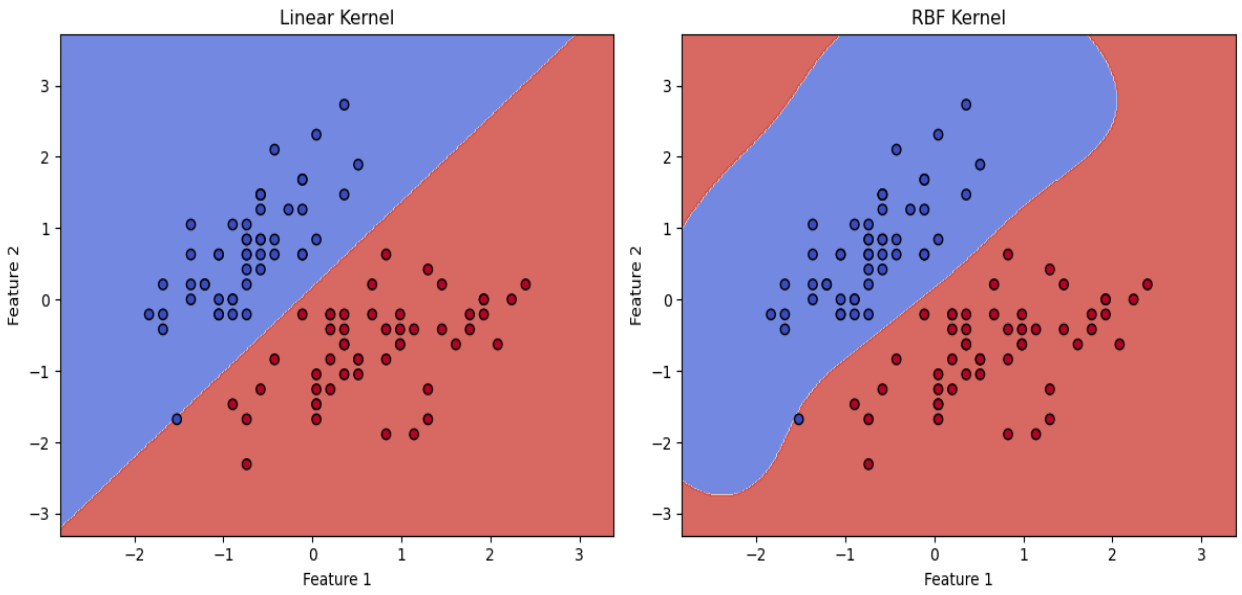
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

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

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Try different values of K
k_values = range(1, 21)
accuracies = []

for k in k_values:
   knn = KNeighborsClassifier(n_neighbors=k)
   knn.fit(X_train, y_train)
   predictions = knn.predict(X_test)
   acc = accuracy_score(y_test, predictions)
   accuracies.append(acc)

# Print accuracy for each K
for k, acc in zip(k_values, accuracies):
   print(f"K={k}: Accuracy={acc:.4f}")

# Plotting accuracy vs K
plt.figure(figsize=(10, 5))
plt.plot(k_values, accuracies, marker='o', linestyle='-')
plt.title('KNN Accuracy for Different K
Values')

plt.xlabel('Number of Neighbors (K)')

plt.ylabel('Accuracy')

plt.xticks(k_values)

plt.grid(True)

plt.show()
```



18. Support Vector Machines (SVM): Train an SVM model with both linear and RBF kernels on a dataset. Visualize the decision boundaries.

Code And Output:

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn import svm, datasets

from sklearn.preprocessing import
StandardScaler


# Load the dataset (2 classes and 2
features for simplicity)

iris = datasets.load_iris()

X = iris.data[:, :2]  # Using only 2
features for visualization

y = iris.target


# Use only 2 classes (binary
classification)

X = X[y != 2]

y = y[y != 2]


# Scale features

scaler = StandardScaler()

X = scaler.fit_transform(X)

# Create a meshgrid for plotting decision
boundaries

h = .02

x_min, x_max = X[:, 0].min() - 1, X[:,
0].max() + 1

y_min, y_max = X[:, 1].min() - 1, X[:,
1].max() + 1

xx, yy = np.meshgrid(np.arange(x_min,
x_max, h),

                     np.arange(y_min, y_max, h))


# Models with Linear and RBF Kernels

models = [

    ('Linear Kernel',
svm.SVC(kernel='linear', C=1.0)),

    ('RBF Kernel', svm.SVC(kernel='rbf',
gamma=0.7, C=1.0))

]


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


for i, (title, clf) in enumerate(models):

    clf.fit(X, y)
    Z = clf.predict(np.c_[xx.ravel(),
yy.ravel()])

    Z = Z.reshape(xx.shape)


    # Plotting

    plt.subplot(1, 2, i + 1)

    plt.contourf(xx, yy, Z,
cmap=plt.cm.coolwarm, alpha=0.8)

    plt.scatter(X[:, 0], X[:, 1], c=y,
cmap=plt.cm.coolwarm, edgecolors='k')

    plt.title(title)

    plt.xlabel('Feature 1')

    plt.ylabel('Feature 2')


plt.tight_layout()

plt.show()
```

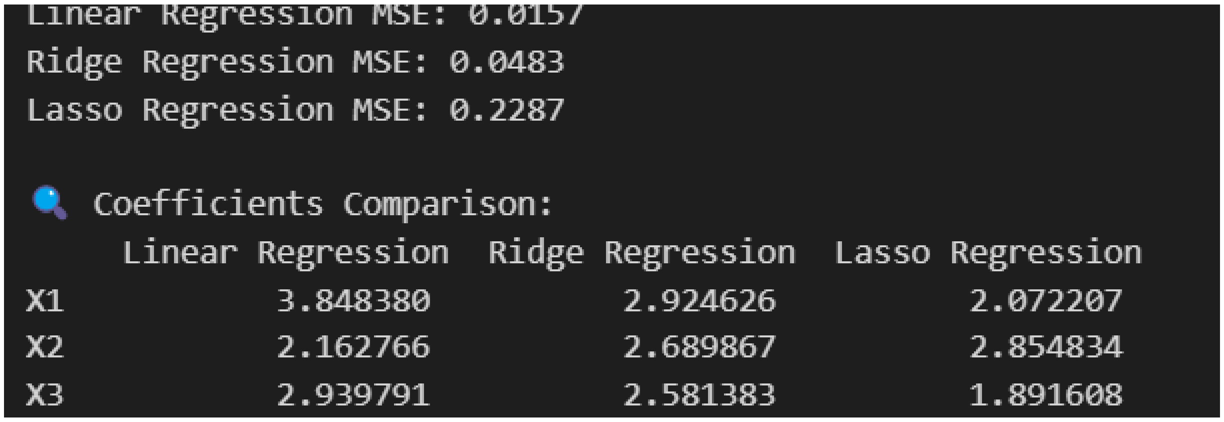


19. Regularization Techniques: Use Lasso and Ridge regression on a dataset. Analyze how they handle multicollinearity and reduce model complexity.

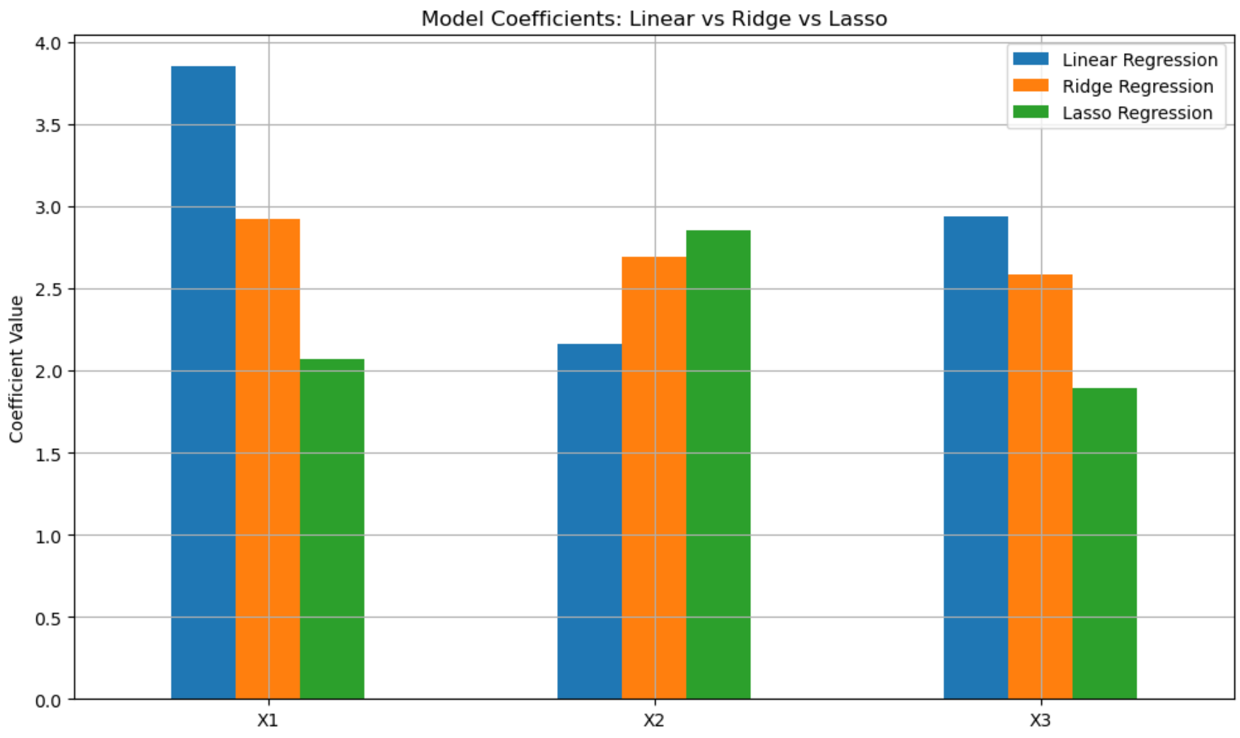
Code And Output:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.linear_model import
LinearRegression, Ridge, Lasso

from sklearn.model_selection import
train_test_split

from sklearn.metrics import
mean_squared_error


# Step 1: Generate synthetic data with
multicollinearity

np.random.seed(0)

n_samples = 100

X1 = np.random.rand(n_samples)

X2 = X1 + np.random.normal(0, 0.1,
n_samples)  # Highly correlated with X1

X3 = np.random.rand(n_samples)


X = np.vstack([X1, X2, X3]).T

y = 4 * X1 + 2 * X2 + 3 * X3 +
np.random.normal(0, 0.1, n_samples)


# Step 2: Split dataset

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


# Step 3: Train models

models = {

   "Linear Regression":
LinearRegression(),

   "Ridge Regression": Ridge(alpha=1.0),

   "Lasso Regression": Lasso(alpha=0.1)

}

coefficients = {}

for name, model in models.items():

   model.fit(X_train, y_train)

   y_pred = model.predict(X_test)

   mse = mean_squared_error(y_test,
y_pred)

   coefficients[name] = model.coef_

   print(f"{name} MSE: {mse:.4f}")


# Step 4: Compare Coefficients

coef_df = pd.DataFrame(coefficients,
index=["X1", "X2", "X3"])
```

```
print("\n🔍 Coefficients
Comparison:\n", coef_df)


# Step 5: Visualize Coefficients

coef_df.plot(kind='bar', figsize=(10, 6))

plt.title("Model Coefficients: Linear vs
Ridge vs Lasso")

plt.ylabel("Coefficient Value")

plt.xticks(rotation=0)

plt.grid(True)

plt.tight_layout()

plt.show()
```



```
Linear Regression MSE: 0.0157
Ridge Regression MSE: 0.0483
Lasso Regression MSE: 0.2287

🔍 Coefficients Comparison:
    Linear Regression  Ridge Regression  Lasso Regression
X1           3.848380          2.924626          2.072207
X2           2.162766          2.689867          2.854834
X3           2.939791          2.581383          1.891608
```

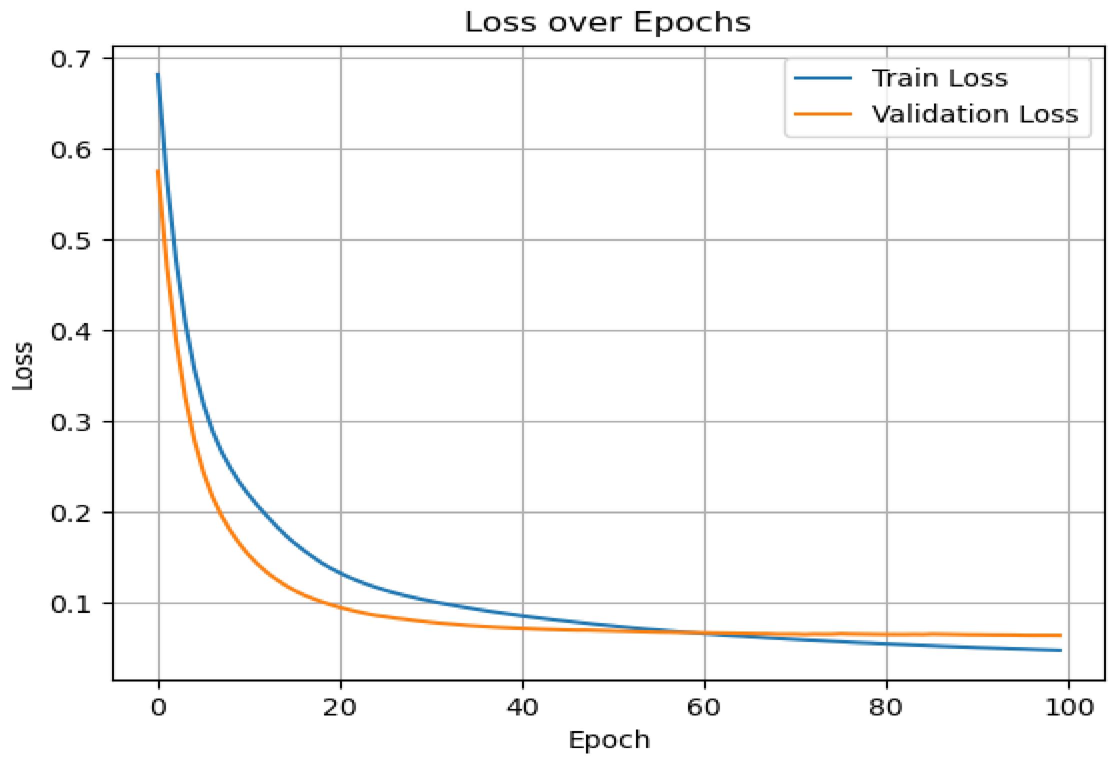


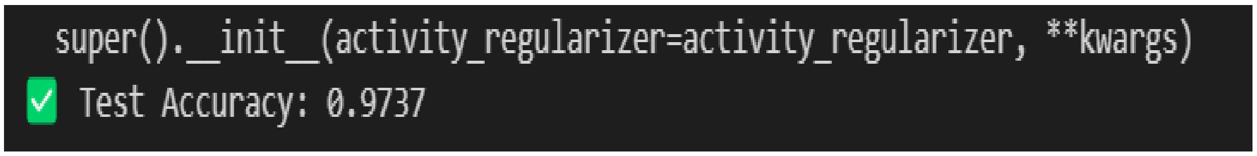
20. Introduction to Neural Networks: Build a simple Artificial Neural Network (ANN) to classify data. Use optimization algorithms (Gradient Descent, SGD) and visualize the loss during training.

Code And Output:

```
# Imports

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import
load_breast_cancer

from sklearn.model_selection import
train_test_split

from sklearn.preprocessing import
StandardScaler

from tensorflow.keras.models import
Sequential

from tensorflow.keras.layers import
Dense

from tensorflow.keras.optimizers import
SGD


# Load dataset

data = load_breast_cancer()

X = data.data

y = data.target


# 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

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)


# Build ANN model

model = Sequential([

   Dense(16, input_shape=(X.shape[1],),
activation='relu'),

   Dense(8, activation='relu'),

   Dense(1, activation='sigmoid')  #
Output layer for binary classification

])


# Compile model

optimizer = SGD(learning_rate=0.01)

model.compile(optimizer=optimizer,
loss='binary_crossentropy',
metrics=['accuracy'])
```

```
# Train model

history = model.fit(X_train, y_train,
validation_data=(X_test, y_test),
epochs=100, batch_size=16, verbose=0)


# Evaluate model

loss, accuracy = model.evaluate(X_test,
y_test, verbose=0)

print(f"✅ Test Accuracy:
{accuracy:.4f}")


# Visualize loss during training

plt.plot(history.history['loss'],
label='Train Loss')

plt.plot(history.history['val_loss'],
label='Validation Loss')

plt.title('Loss over Epochs')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

plt.grid(True)

plt.show()
```





```
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
✅ Test Accuracy: 0.9737
```