Institute of Distance and Open Learning

Vidya Nagari, Kalina, Santacruz East – 400098.

A Practical Journal Submitted in fulfillment of the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

YEAR 2024-25

Part - II

Semester - IV

Subject Code- 91955

Subject Name- Business Intelligence and Big Data Analytics III

BY

RAINA RIZWANULLAH KHAN

Seat Number: 2700155

Application ID: 42662

Institute of Distance and Open Learning (IDOL)

University of Mumbai

Certificate

This is to certify that **Miss Raina Rizwanullah Khan** student of Masters of Computer Science, Part 2, Semester 4 has completed the specified term work in the subject of **Business Intelligence and Big Data Analytics III** in satisfactorily manner within this institute as laid down by University of Mumbai during the academic year 2024 to 2025.

M.Sc. CS Coordinator **Examiner**

Date:30-06-2025 **Guide**

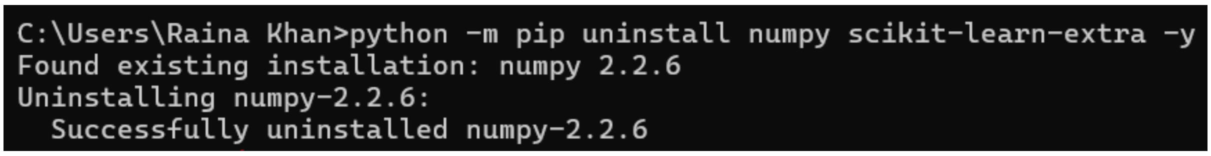
INDEX

Practical No 1

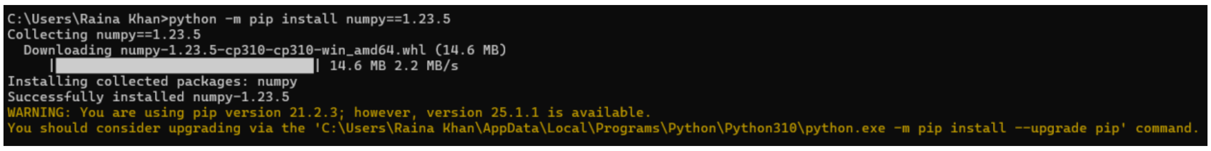
Aim: Pre-process the given data set and hence apply clustering techniques like KMeans, K-Medoids. Interpre t the result.

Code:

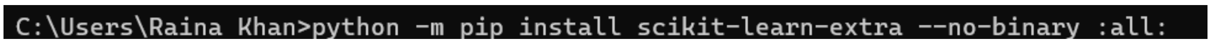
Install python on the system then run this code in cmd py -m pip install pandas numpy matplotlib scikit-learn scikit-learn-extra to install all the required packages.



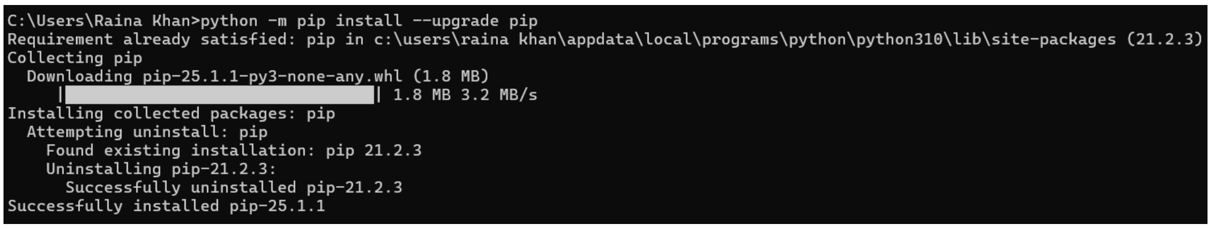
```
C:\Users\Raina Khan>python -m pip uninstall numpy scikit-learn-extra -y
Found existing installation: numpy 2.2.6
Uninstalling numpy-2.2.6:
  Successfully uninstalled numpy-2.2.6
```



```
C:\Users\Raina Khan>python -m pip install numpy==1.23.5
Collecting numpy==1.23.5
  Downloading numpy-1.23.5-cp310-cp310-win_amd64.whl (14.6 MB)
     |████████████████████████████████| 14.6 MB 2.2 MB/s
Installing collected packages: numpy
Successfully installed numpy-1.23.5
WARNING: You are using pip version 21.2.3; however, version 25.1.1 is available.
You should consider upgrading via the 'C:\Users\Raina Khan\AppData\Local\Programs\Python\Python310\python.exe -m pip install --upgrade pip' command.
```



```
C:\Users\Raina Khan>python -m pip install scikit-learn-extra --no-binary :all:
```



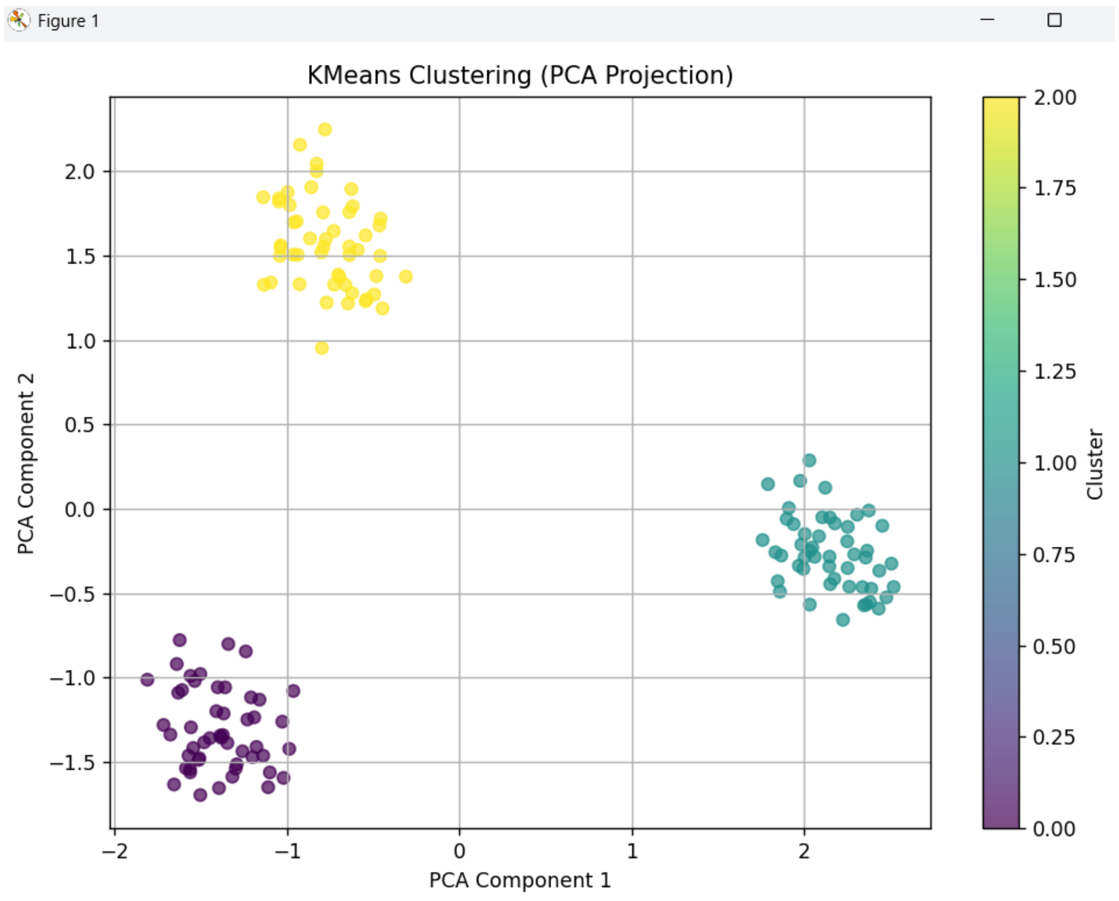
```
C:\Users\Raina Khan>python -m pip install --upgrade pip
Requirement already satisfied: pip in c:\users\raina khan\appdata\local\programs\python\python310\lib\site-packages (21.2.3)
Collecting pip
  Downloading pip-25.1.1-py3-none-any.whl (1.8 MB)
     |████████████████████████████████| 1.8 MB 3.2 MB/s
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 21.2.3
    Uninstalling pip-21.2.3:
      Successfully uninstalled pip-21.2.3
Successfully installed pip-25.1.1
```

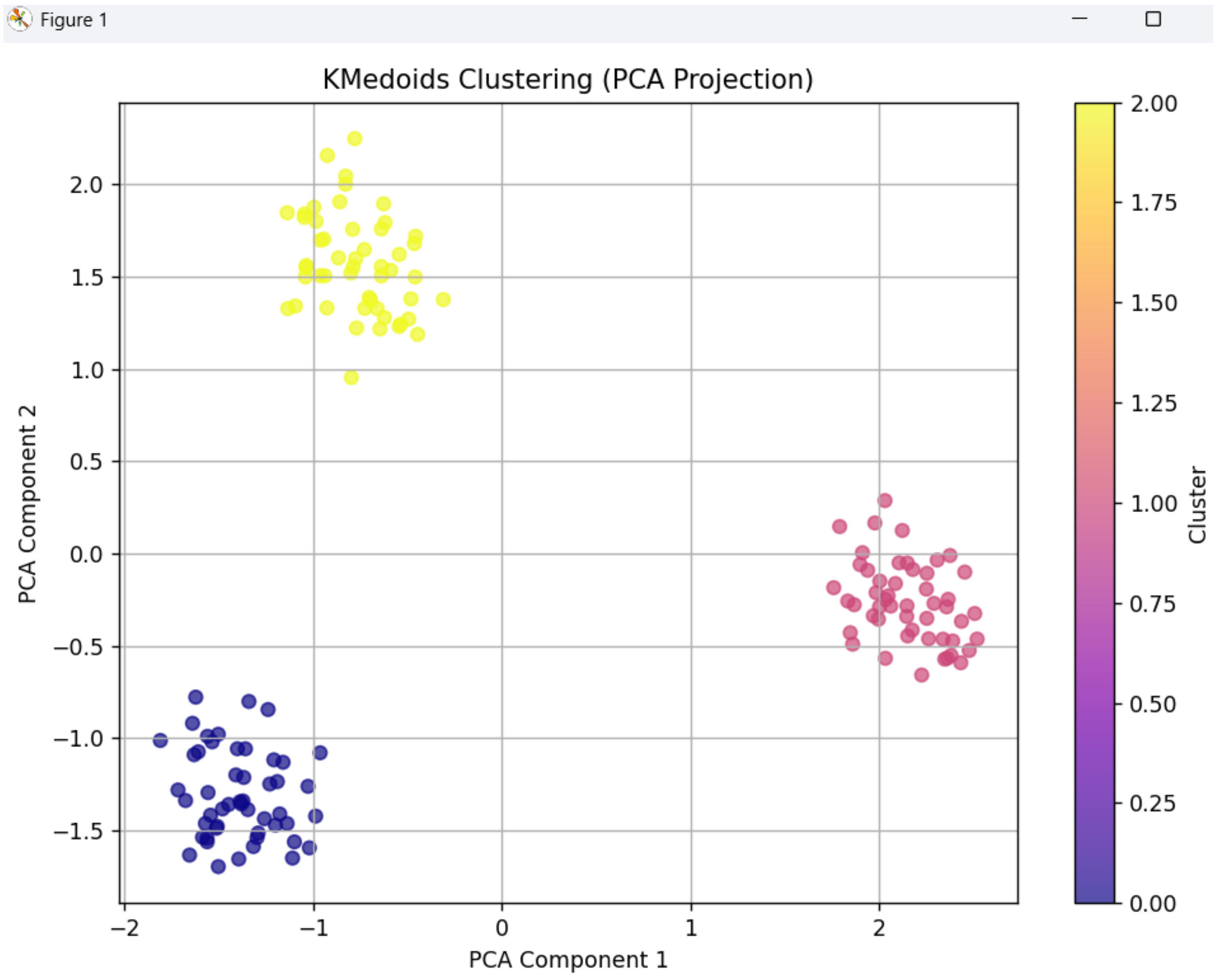
```
# Install required packages first via Command Prompt (if not installed):
# python -m pip install pandas numpy matplotlib scikit-learn scikit-learn-extra


# --- Start of Code ---
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn_extra.cluster import KMedoids
from sklearn.decomposition import PCA
```

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs

# 1. Generate synthetic dataset (150 samples, 4 features, 3 clusters)
X, _ = make_blobs(n_samples=150, centers=3, n_features=4,
random_state=42)
df = pd.DataFrame(X, columns=['Feature1', 'Feature2', 'Feature3', 'Feature4'])

# 2. Pre-processing: Standardize the data
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df)

# 3. Apply KMeans clustering
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(df_scaled)
df['KMeans_Cluster'] = kmeans.labels_

# Output KMeans results
print("KMeans Cluster Centers:\n", kmeans.cluster_centers_)

# 4. Apply KMedoids clustering
kmedoids = KMedoids(n_clusters=3, random_state=42)
kmedoids.fit(df_scaled)
df['KMedoids_Cluster'] = kmedoids.labels_

# Output KMedoids results
print("KMedoids Cluster Medoids (indices):", kmedoids.medoid_indices_)
```

```
# 5. Reduce dimensions for visualization
pca = PCA(n_components=2)
df_pca = pca.fit_transform(df_scaled)

# 6. Visualize KMeans clusters
plt.figure(figsize=(8, 6))
plt.scatter(df_pca[:, 0], df_pca[:, 1], c=df['KMeans_Cluster'], cmap='viridis',
alpha=0.7)
plt.title('KMeans Clustering (PCA Projection)')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.colorbar(label='Cluster')
plt.grid(True)
plt.tight_layout()
plt.show()

# 7. Visualize KMedoids clusters
plt.figure(figsize=(8, 6))
plt.scatter(df_pca[:, 0], df_pca[:, 1], c=df['KMedoids_Cluster'], cmap='plasma',
alpha=0.7)
plt.title('KMedoids Clustering (PCA Projection)')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.colorbar(label='Cluster')
plt.grid(True)
plt.tight_layout()
plt.show()


# Optional: Save result
# df.to_csv("clustered_result.csv", index=False)
# --- End of Code ---
```

Output:





```
= RESTART: C:/Users/Raina Khan/AppData/Local/Programs/Python/Python310/BI Pract
1.py
KMeans Cluster Centers:
 [[-1.16516133 -1.33142953 -0.62509016  0.34036025]
 [-0.04419554  1.03916665  1.3942501  -1.29887211]
 [ 1.20935687  0.29226288 -0.76915994  0.95851186]]
KMedoids Cluster Medoids (indices): [133 144  48]
```

Practical No 2

Aim: Pre-process the given data set and hence apply partition clustering algorithms. Interpret the result.

Code:

```
import pandas as pd
import numpy as np
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

# Step 1: Generate synthetic dataset
X, _ = make_blobs(n_samples=100, centers=3, n_features=3,
random_state=42)
df = pd.DataFrame(X, columns=['Feature1', 'Feature2', 'Feature3'])

# Step 2: Preprocess (scale) the data
scaler = StandardScaler()
scaled_features = scaler.fit_transform(df)
df_scaled = pd.DataFrame(scaled_features, columns=df.columns)

# Step 3: Apply K-Means clustering
num_clusters = 3
kmeans = KMeans(n_clusters=num_clusters, random_state=42, n_init=10)
kmeans.fit(df_scaled)

# Step 4: Add cluster labels
df['Cluster'] = kmeans.labels_
```

```
# Step 5: Print number of data points in each cluster
print("\033[95mNumber of data points in each cluster:\033[0m")
print(df['Cluster'].value_counts())

# Step 6: Print cluster centroids
print("\n\033[95mCluster centroids:\033[0m")
centroids = pd.DataFrame(kmeans.cluster_centers_, columns=df.columns[:-1])
print(centroids)

# Step 7: Print mean values of features across clusters
print("\n\033[95mMean values of features across clusters:\033[0m")
cluster_means = df.groupby('Cluster').mean(numeric_only=True)
print(cluster_means)
```

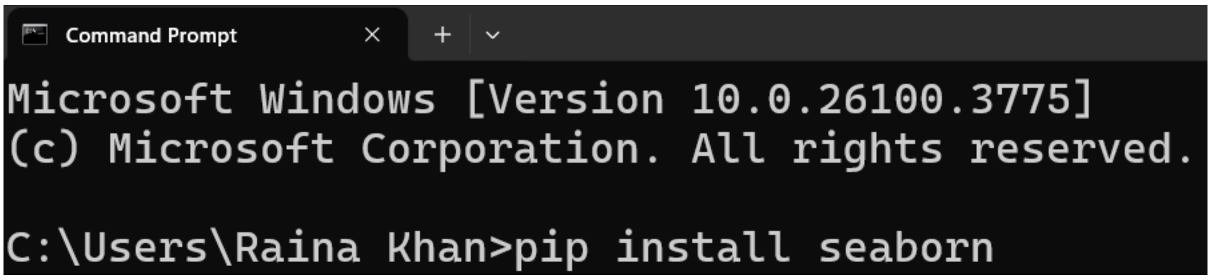
Output:

```
= RESTART: C:/Users/Raina Khan/AppData/Local/Programs/Python/Python310/BI Pract2
.py
[95mNumber of data points in each cluster:[0m
Cluster
1    34
2    33
0    33
Name: count, dtype: int64

[95mCluster centroids:[0m
   Feature1  Feature2  Feature3
0  1.141480 -1.407411 -1.358115
1  0.110490  0.789528  0.929131
2 -1.255318  0.593958  0.400828

[95mMean values of features across clusters:[0m
         Feature1  Feature2  Feature3
Cluster
0        1.951547 -6.481144 -6.634443
1       -2.645974  8.887457  4.697897
2       -8.736560  7.519361  2.080378
```

Practical No 3

Aim: Pre-process the given data set and hence apply hierarchical algorithms and density-based clustering techniques. Interpret the result.

Code:



```
Microsoft Windows [Version 10.0.26100.3775]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Raina Khan>pip install seaborn
```

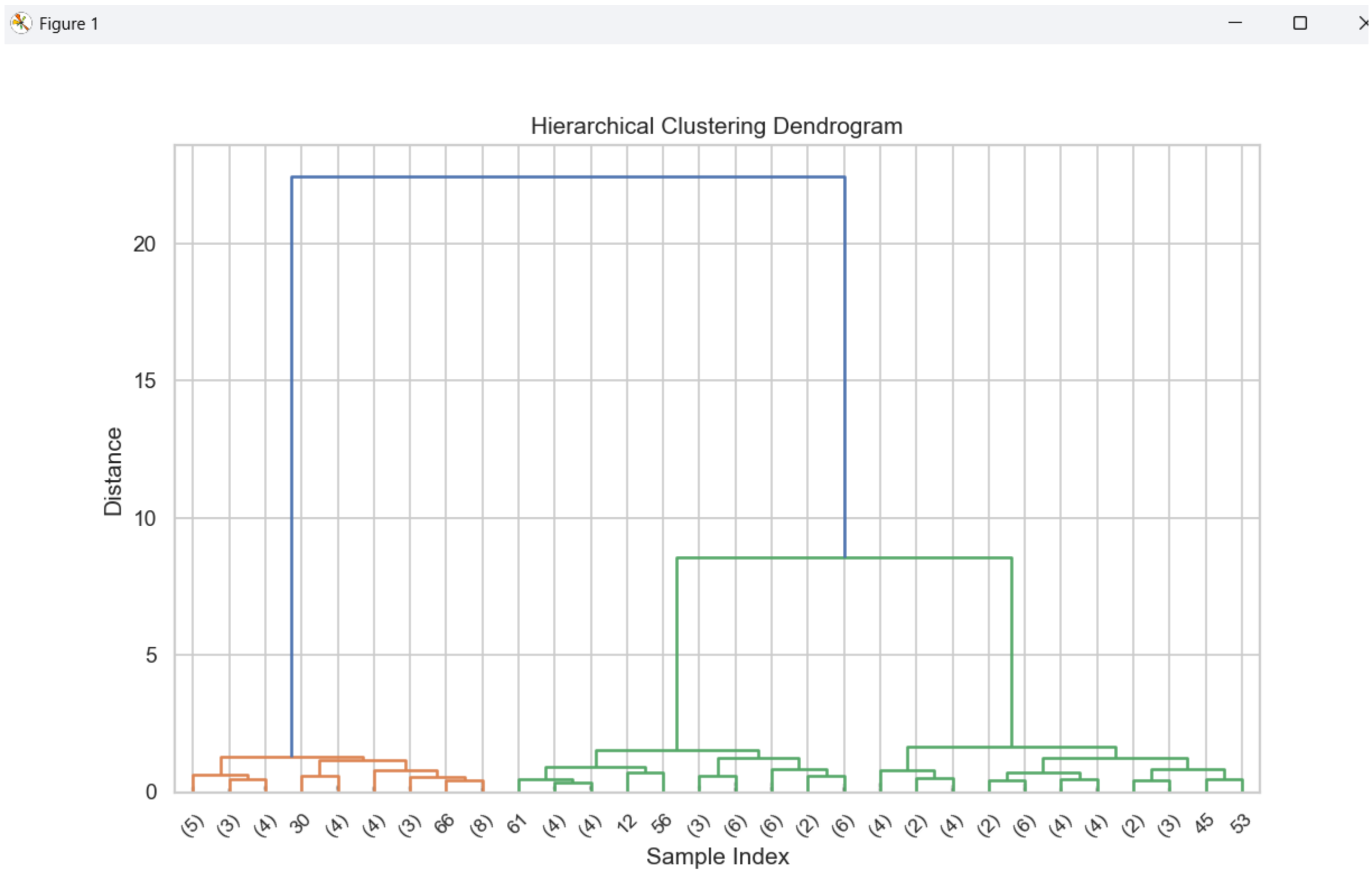
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import DBSCAN
from scipy.cluster.hierarchy import linkage, dendrogram, fcluster
import matplotlib.pyplot as plt
import seaborn as sns

# Optional: Set style for plots
sns.set(style="whitegrid")

# Step 1: Load dataset
# If you don't have a file, you can generate synthetic data using make_blobs
(see below)
# df = pd.read_csv("your_dataset.csv")
from sklearn.datasets import make_blobs
X, _ = make_blobs(n_samples=100, centers=3, n_features=3,
random_state=42)
```

```
df = pd.DataFrame(X, columns=["Feature1", "Feature2", "Feature3"])

# Step 2: Handle missing values (drop rows with NaNs)
df.dropna(inplace=True)

# Step 3: Scale the data
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df)

# --------------------------------------
# Step 4: Hierarchical Clustering
# --------------------------------------
linked = linkage(scaled_data, method='ward')

# Plot the dendrogram
plt.figure(figsize=(10, 6))
dendrogram(linked, truncate_mode='lastp', p=30, leaf_rotation=45.,
leaf_font_size=10., show_contracted=True)
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()

# Assign cluster labels (e.g., k=3)
hc_labels = fcluster(linked, t=3, criterion='maxclust')
df['HC_Cluster'] = hc_labels
```

```
# -------------------------------------
# Step 5: DBSCAN Clustering
# -------------------------------------
db = DBSCAN(eps=0.5, min_samples=5)
db_labels = db.fit_predict(scaled_data)
df['DBSCAN_Cluster'] = db_labels

# -------------------------------------
# Step 6: Visualize Clusters
# -------------------------------------
# Visualize Hierarchical Clusters
plt.figure(figsize=(8, 5))
sns.scatterplot(data=df, x="Feature1", y="Feature2", hue="HC_Cluster",
palette="Set2")
plt.title("Hierarchical Clustering Result")
plt.show()

# Visualize DBSCAN Clusters
plt.figure(figsize=(8, 5))
sns.scatterplot(data=df, x="Feature1", y="Feature2", hue="DBSCAN_Cluster",
palette="Set1")
plt.title("DBSCAN Clustering Result")
plt.show()

# -------------------------------------
# Step 7: Interpretation Guide
# -------------------------------------
```

```
print("\nInterpretation Guide:")
print("- Hierarchical clustering builds a tree (dendrogram). Cut the tree to
decide number of clusters.")
print("- DBSCAN identifies clusters based on density. Noise points (outliers) are
labeled as -1.")
print("- You can adjust `eps` and `min_samples` to tune DBSCAN for your
dataset.")
```

Output:



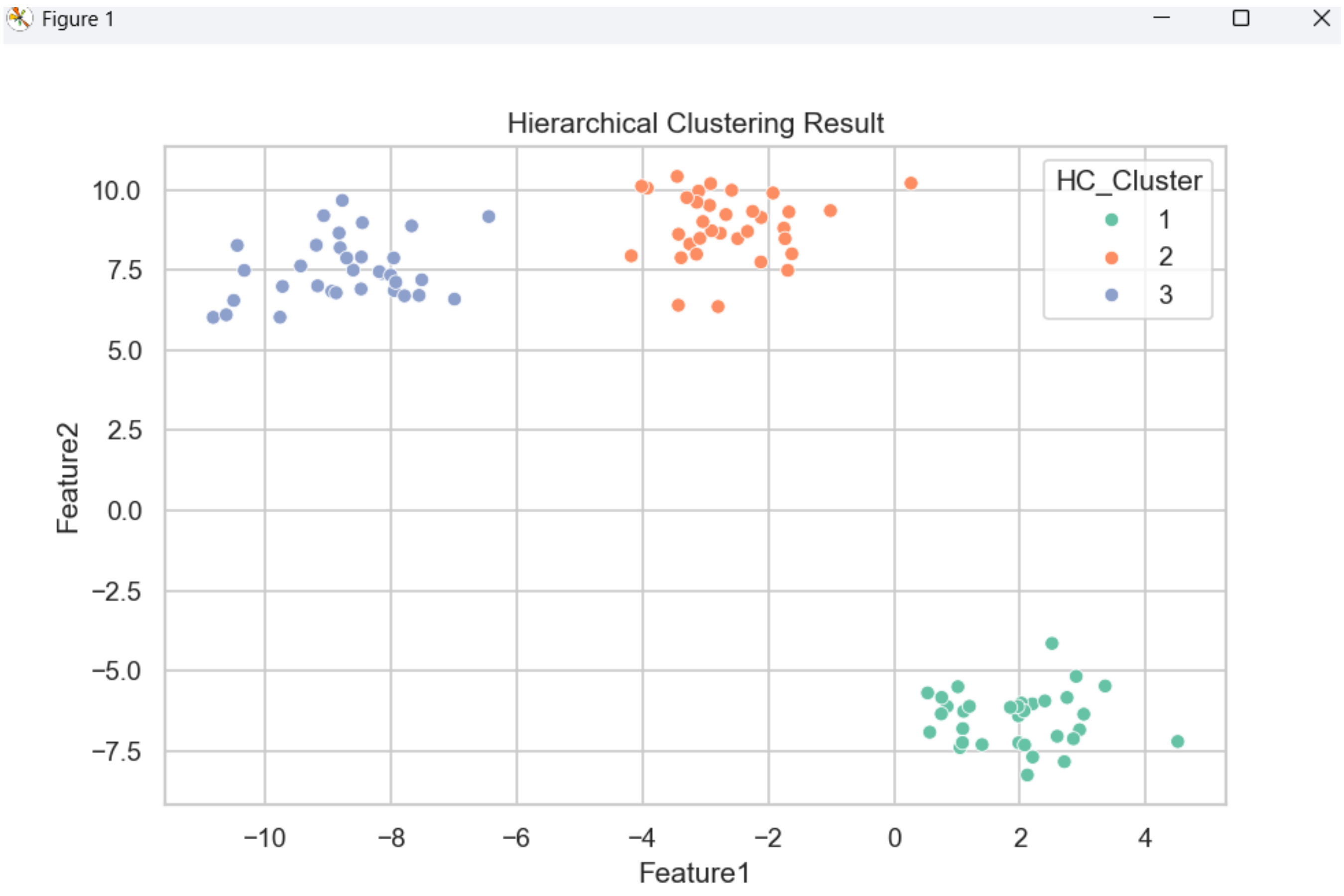


Figure 1
Hierarchical Clustering Result
HC_Cluster
1
2
3
Feature2
Feature1
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10
−8
−6
−4
−2
0
2
4

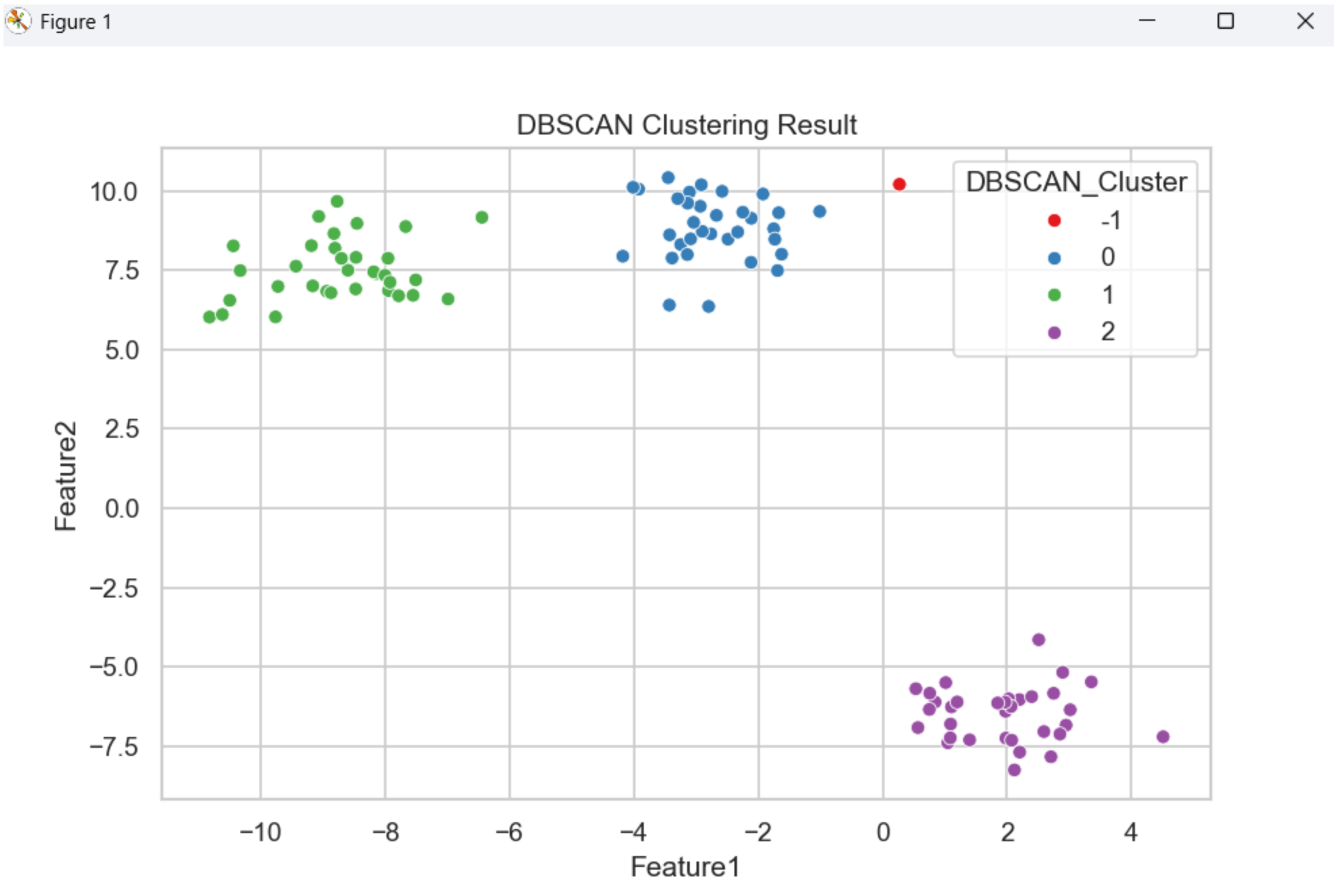


Figure 1
DBSCAN Clustering Result
DBSCAN_Cluster
-1
0
1
2
Feature2
Feature1
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10
−8
−6
−4
−2
0
2
4

Practical No 4

Aim: Pre-process the given data set and hence classify the resultant data set using tree classification techniques. Interpret the result.

Code:

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score
import matplotlib.pyplot as plt

# Step 1: Generate a synthetic dataset (4 features, 2 classes)
from sklearn.datasets import make_classification

X_sample, y_sample = make_classification(
   n_samples=220,
   n_features=4,
   n_informative=3,
   n_redundant=0,
   n_repeated=0,
   n_classes=2,
   random_state=42
)

# Step 2: Convert to DataFrame
```

```
df = pd.DataFrame(X_sample, columns=["Feature1", "Feature2", "Feature3",
"Feature4"])
df["target"] = y_sample

# Step 3: Preprocessing (optional: drop NA if real dataset is used)
df.dropna(inplace=True)
le = LabelEncoder()
df["target"] = le.fit_transform(df["target"])

# Step 4: Split dataset into features and labels
X = df.drop("target", axis=1)
y = df["target"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# ------------------------
# Decision Tree Classifier
# ------------------------
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train, y_train)

# Evaluate
dt_predictions = dt_classifier.predict(X_test)
dt_accuracy = accuracy_score(y_test, dt_predictions)
print("Decision Tree Accuracy:", round(dt_accuracy, 2))
print(classification_report(y_test, dt_predictions))
```

```
# Plot Decision Tree
plt.figure(figsize=(12, 8))
plot_tree(dt_classifier, filled=True, feature_names=X.columns,
class_names=["Class 0", "Class 1"])
plt.title("Decision Tree")
plt.show()

# -------------------------
# Random Forest Classifier
# -------------------------
rf_classifier = RandomForestClassifier(random_state=42)
rf_classifier.fit(X_train, y_train)

# Evaluate
rf_predictions = rf_classifier.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_predictions)
print("Random Forest Accuracy:", round(rf_accuracy, 2))
print(classification_report(y_test, rf_predictions))
```

Output:

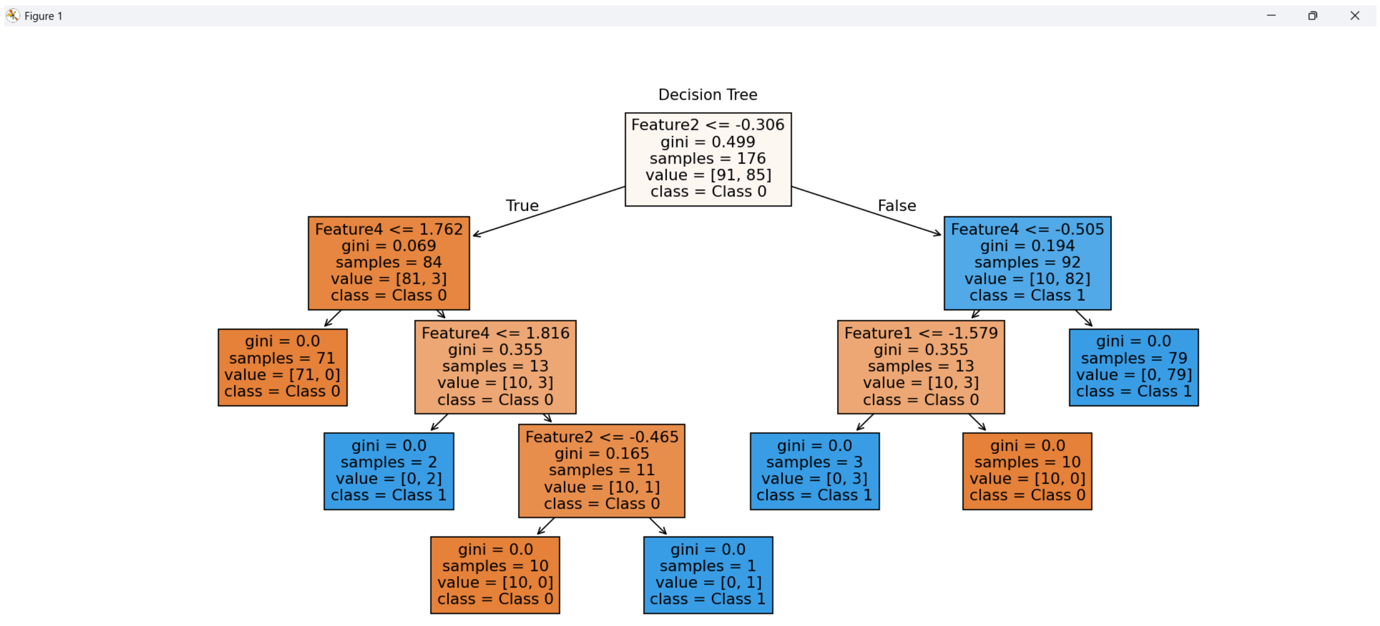


Figure 1
Decision Tree
Feature2 <= -0.306 gini = 0.499 samples = 176 value = [91, 85] class = Class 0
True
False
Feature4 <= 1.762 gini = 0.069 samples = 84 value = [81, 3] class = Class 0
Feature4 <= -0.505 gini = 0.194 samples = 92 value = [10, 82] class = Class 1
gini = 0.0 samples = 71 value = [71, 0] class = Class 0
Feature4 <= 1.816 gini = 0.355 samples = 13 value = [10, 3] class = Class 0
Feature1 <= -1.579 gini = 0.355 samples = 13 value = [10, 3] class = Class 0
gini = 0.0 samples = 79 value = [0, 79] class = Class 1
gini = 0.0 samples = 2 value = [0, 2] class = Class 1
Feature2 <= -0.465 gini = 0.165 samples = 11 value = [10, 1] class = Class 0
gini = 0.0 samples = 3 value = [0, 3] class = Class 1
gini = 0.0 samples = 10 value = [10, 0] class = Class 0
gini = 0.0 samples = 10 value = [10, 0] class = Class 0
gini = 0.0 samples = 1 value = [0, 1] class = Class 1

```
= RESTART: C:/Users/Raina Khan/AppData/Local/Programs/Python/Python310/BI Pract
4.py
Decision Tree Accuracy: 0.93
              precision    recall  f1-score   support

           0       0.90      0.95      0.93        20
           1       0.96      0.92      0.94        24

    accuracy                           0.93        44
   macro avg       0.93      0.93      0.93        44
weighted avg       0.93      0.93      0.93        44

Random Forest Accuracy: 0.93
              precision    recall  f1-score   support

           0       0.95      0.90      0.92        20
           1       0.92      0.96      0.94        24

    accuracy                           0.93        44
   macro avg       0.93      0.93      0.93        44
weighted avg       0.93      0.93      0.93        44
```

Practical No 5

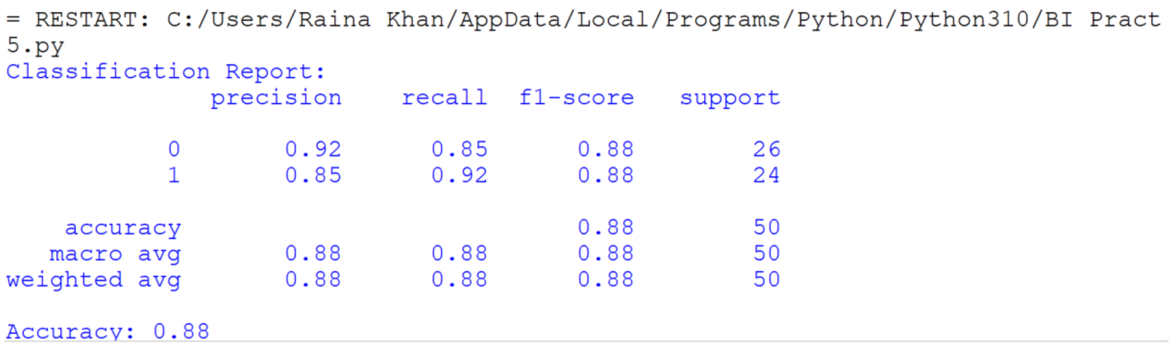
Aim: Pre-process the given data set and hence classify the resultant data set using Statistical based classifiers. Interpret the result.

Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report, accuracy_score
from sklearn.datasets import make_classification

# Generate synthetic dataset
X, y = make_classification(
    n_samples=250,
    n_features=5,
    n_informative=3,
    n_redundant=0,
    n_classes=2,
    random_state=42
)

# Convert to DataFrame for consistency
df = pd.DataFrame(X, columns=[f'Feature{i}' for i in range(1, 6)])
df['target'] = y

# Step 1: Separate features and target
X = df.drop('target', axis=1)
```

```
y = df['target']

# Step 2: Normalize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 3: Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42
)

# Step 4: Train Naive Bayes classifier
classifier = GaussianNB()
classifier.fit(X_train, y_train)

# Step 5: Evaluate
y_pred = classifier.predict(X_test)
print("Classification Report:")
print(classification_report(y_test, y_pred))
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

Output:



```
= RESTART: C:/Users/Raina Khan/AppData/Local/Programs/Python/Python310/BI Pract
5.py
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.85      0.88        26
           1       0.85      0.92      0.88        24

    accuracy                           0.88        50
   macro avg       0.88      0.88      0.88        50
weighted avg       0.88      0.88      0.88        50

Accuracy: 0.88
```

Practical No 6

Aim: Pre-process the given data set and hence classify the resultant data set using support vector machine. Interpret the result

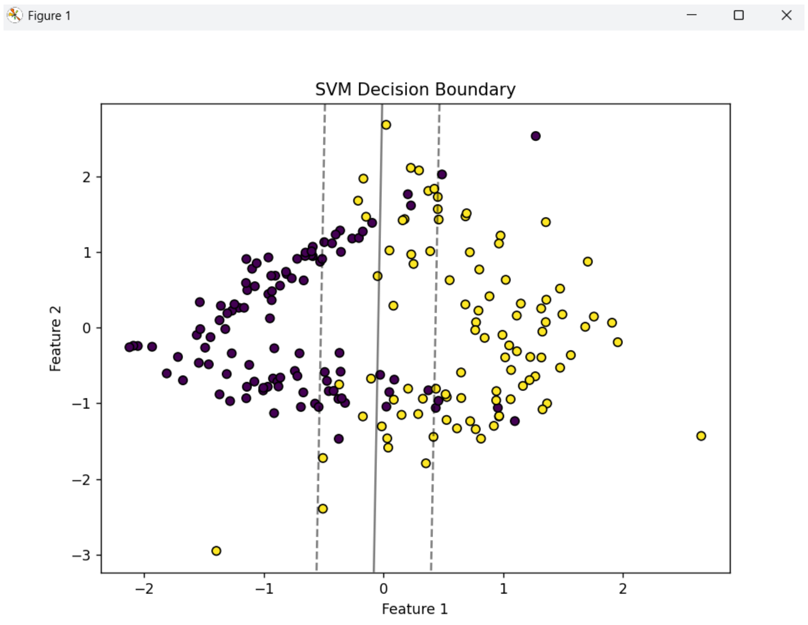
Code:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt

# Step 1: Generate synthetic dataset (replace this with pd.read_csv if you have a real file)
from sklearn.datasets import make_classification
X, y = make_classification(
    n_samples=250,
    n_features=2,  # Set to 2 for plotting decision boundary
    n_informative=2,
    n_redundant=0,
    n_classes=2,
    random_state=42
)

# Step 2: Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
# Step 3: Split into training and testing
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42
)

# Step 4: Train SVM (linear kernel)
svm_model = SVC(kernel='linear')
svm_model.fit(X_train, y_train)

# Step 5: Predict & Evaluate
y_pred = svm_model.predict(X_test)
print("Classification Report:")
print(classification_report(y_test, y_pred))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Step 6: Plot if 2D
if X_train.shape[1] == 2:
    plt.figure(figsize=(8, 6))
    plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap='viridis',
edgecolors='k')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')

    # Plot decision boundary
    ax = plt.gca()
```

```
xlim = ax.get_xlim()
ylim = ax.get_ylim()
xx, yy = np.meshgrid(np.linspace(xlim[0], xlim[1], 50),
                     np.linspace(ylim[0], ylim[1], 50))
Z = svm_model.decision_function(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
ax.contour(xx, yy, Z, colors='k', levels=[-1, 0, 1],
           alpha=0.5, linestyles=['--', '-', '--'])
plt.title("SVM Decision Boundary")
plt.show()
```

Output:



```
= RESTART: C:/Users/Raina Khan/AppData/Local/Programs/Python/Python310/BI Pract
6.py
Classification Report:
              precision    recall  f1-score   support

           0       0.74      0.74      0.74        19
           1       0.84      0.84      0.84        31

    accuracy                           0.80        50
   macro avg       0.79      0.79      0.79        50
weighted avg       0.80      0.80      0.80        50

Confusion Matrix:
[[14  5]
 [ 5 26]]
```

Practical No 7

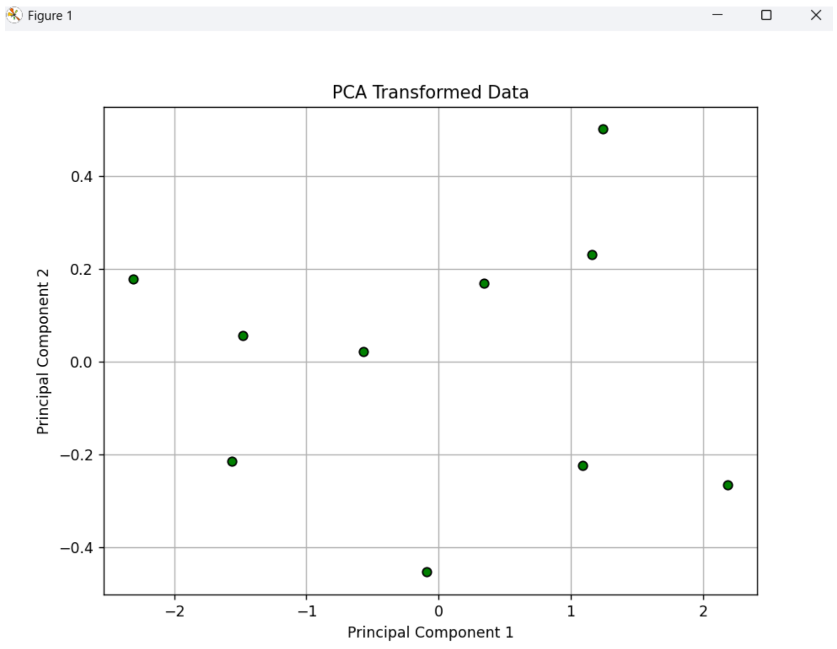
Aim: Write a program to explain different functions of Principal Components.

Code:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler


# Step 1: Create a sample dataset with more meaningful variance

X = np.array([

    [2.5, 2.4],

    [0.5, 0.7],

    [2.2, 2.9],

    [1.9, 2.2],

    [3.1, 3.0],

    [2.3, 2.7],

    [2, 1.6],

    [1, 1.1],

    [1.5, 1.6],

    [1.1, 0.9]

])


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


# Step 2: Standardize the data (very important for PCA)

scaler = StandardScaler()
```

```
X_scaled = scaler.fit_transform(X)

# Step 3: Apply PCA
pca = PCA(n_components=2)  # Keep both components for explanation
X_pca = pca.fit_transform(X_scaled)

# Step 4: Display results
print("\nTransformed Data (PCA Result):\n", X_pca)
print("\nExplained Variance Ratio:", pca.explained_variance_ratio_)
print("Singular Values:", pca.singular_values_)
print("Components (eigenvectors):\n", pca.components_)

# Step 5: Visualize the PCA result
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], color='green', edgecolor='k')
plt.title('PCA Transformed Data')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.grid(True)
plt.show()
```

Output:



```
= RESTART: C:/Users/Raina Khan/AppData/Local/Programs/Python/Python310/BI Pract 7.
py
Original Data:
 [[2.5 2.4]
 [0.5 0.7]
 [2.2 2.9]
 [1.9 2.2]
 [3.1 3. ]
 [2.3 2.7]
 [2.  1.6]
 [1.  1.1]
 [1.5 1.6]
 [1.1 0.9]]
```

```
Transformed Data (PCA Result):
 [[ 1.08643242 -0.22352364]
 [-2.3089372   0.17808082]
 [ 1.24191895  0.501509  ]
 [ 0.34078247  0.16991864]
 [ 2.18429003 -0.26475825]
 [ 1.16073946  0.23048082]
 [-0.09260467 -0.45331721]
 [-1.48210777  0.05566672]
 [-0.56722643  0.02130455]
 [-1.56328726 -0.21536146]]

Explained Variance Ratio: [0.96296464 0.03703536]
Singular Values: [4.38854107 0.86064352]
Components (eigenvectors):
 [[ 0.70710678  0.70710678]
 [-0.70710678  0.70710678]]
```