Program: 1a

```python
import matplotlib.pyplot as plt
from google.colab import drive

drive.mount('/content/drive')

def visualize_find_s_algorithm(positive_examples):
    hypothesis = ['0'] * len(positive_examples[0])

    visualization = []

    for idx, example in enumerate(positive_examples, start=1):
        for i in range(len(example)):
            if hypothesis[i] == '0':
                hypothesis[i] = example[i]
            elif hypothesis[i] != example[i]:
                hypothesis[i] = '?'

        visualization.append((idx, " ".join(hypothesis)))

    return visualization

positive_examples_find_s = [
    ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same'],
    ['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same'],
    ['Rainy', 'Cold', 'High', 'Strong', 'Cool', 'Change'],
    ['Sunny', 'Hot', 'High', 'Strong', 'Cool', 'Change']
]

hypothesis_find_s_visualization =
visualize_find_s_algorithm(positive_examples_find_s)
plt.figure(figsize=(12, 6))
plt.plot([point[0] for point in hypothesis_find_s_visualization],
[point[1] for point in hypothesis_find_s_visualization], marker='o')
plt.title('FIND-S Algorithm Visualization')
plt.xlabel('Example Index')
plt.ylabel('Hypothesis')
plt.xticks(range(1, len(positive_examples_find_s) + 1))
plt.grid(True)
plt.savefig('/content/drive/MyDrive/ML Lab/ExNo02/FIND-S Algorithm
Visualization.png', dpi=500)
plt.show()
```

Program: 1b

```
import matplotlib.pyplot as plt
def visualize_candidate_elimination_algorithm(positive_examples,
negative_examples):
    specific_hypothesis = ['0'] * len(positive_examples[0])
general_hypothesis = ['?'] * len(positive_examples[0])
    visualization_specific = []
    visualization_general = []
    for idx, example in enumerate(positive_examples, start=1):
        for i in range(len(example)):
            if specific_hypothesis[i] == '0':
                specific_hypothesis[i] = example[i]
            elif specific_hypothesis[i] != example[i]:
                specific_hypothesis[i] = '?'

        for i in range(len(example)):
            if example[i] != specific_hypothesis[i]:
                general_hypothesis[i] = specific_hypothesis[i]
        visualization_specific.append((idx, "
".join(specific_hypothesis)))
        visualization_general.append((idx, "
".join(general_hypothesis)))
    for example in negative_examples:
        for i in range(len(example)):
            if example[i] != specific_hypothesis[i]:
                general_hypothesis[i] = '?'
visualization_general.append((idx + 1, " ".join(general_hypothesis)))

    return visualization_specific, visualization_general
positive_examples_candidate = [
    ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same'],
    ['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same'],
    ['Rainy', 'Cold', 'Normal', 'Strong', 'Cool', 'Change'],
    ['Sunny', 'Hot', 'High', 'Strong', 'Cool', 'Change']
]
negative_examples_candidate = [
    ['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change'],
    ['Sunny', 'Warm', 'Normal', 'Weak', 'Cool', 'Same']
]
specific, general =
visualize_candidate_elimination_algorithm(positive_examples_candidate,
negative_examples_candidate)

plt.figure(figsize=(12, 6))
plt.plot([point[0] for point in specific], [point[1] for point in
specific], marker='o', label='Specific Hypothesis')
plt.plot([point[0] for point in general], [point[1] for point in
general], marker='x', linestyle='--', label='General Hypothesis')
```

```
plt.title('Candidate-Elimination Algorithm Visualization')
plt.xlabel('Example Index')
plt.ylabel('Hypothesis')
plt.legend()
plt.xticks(range(1, len(positive_examples_candidate) + 2))
plt.grid(True)
plt.savefig('/content/drive/MyDrive/ML Lab/ExNo02/Candidate-Elimination
Algorithm Visualization.png', dpi=500)
plt.show()
```

Program: 3a

```
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.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

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

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

# Apply PCA for dimensionality reduction
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

# Plot the PCA-transformed training data
plt.figure(figsize=(8, 6))
colors = ['red', 'green', 'blue']
for i in range(3):
    plt.scatter(X_train_pca[y_train == i, 0], X_train_pca[y_train == i, 1], label=f'Class {i}',
color=colors[i])

plt.title('PCA-transformed Training Data')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.show()

# Train a classifier on the PCA-transformed data
knn_classifier = KNeighborsClassifier(n_neighbors=3)
knn_classifier.fit(X_train_pca, y_train)

# Make predictions on the PCA-transformed test data
y_pred = knn_classifier.predict(X_test_pca)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy on the test set: {accuracy:.2f}')

# Plot the PCA-transformed test data with predictions
plt.figure(figsize=(8, 6))
for i in range(3):
    plt.scatter(X_test_pca[y_test == i, 0], X_test_pca[y_test == i, 1], label=f'Actual Class {i}',
color=colors[i], alpha=0.7)

plt.scatter(X_test_pca[:, 0], X_test_pca[:, 1], c=y_pred, marker='x', cmap='viridis',
label='Predictions')
plt.title('PCA-transformed Test Data with Predictions')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.show()
```

Program:3b

```
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.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

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

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

# Apply Linear Discriminant Analysis (LDA) for dimensionality reduction
lda = LinearDiscriminantAnalysis(n_components=2)
X_train_lda = lda.fit_transform(X_train, y_train)
X_test_lda = lda.transform(X_test)

# Plot the LDA-transformed training data
plt.figure(figsize=(8, 6))
colors = ['red', 'green', 'blue']
for i in range(3):
    plt.scatter(X_train_lda[y_train == i, 0], X_train_lda[y_train == i, 1], label=f'Class {i}',
color=colors[i])

plt.title('LDA-transformed Training Data')
plt.xlabel('LDA Component 1')
plt.ylabel('LDA Component 2')
plt.legend()
plt.show()

# Train a classifier on the LDA-transformed data
knn_classifier = KNeighborsClassifier(n_neighbors=3)
knn_classifier.fit(X_train_lda, y_train)

# Make predictions on the LDA-transformed test data
y_pred = knn_classifier.predict(X_test_lda)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy on the test set: {accuracy:.2f}')

# Plot the LDA-transformed test data with predictions
plt.figure(figsize=(8, 6))
for i in range(3)
```

```
plt.scatter(X_test_lda[y_test == i, 0], X_test_lda[y_test == i, 1], label=f'Actual Class {i}',
color=colors[i], alpha=0.7)

plt.scatter(X_test_lda[:, 0], X_test_lda[:, 1], c=y_pred, marker='x', cmap='viridis',
label='Predictions')
plt.title('LDA-transformed Test Data with Predictions')
plt.xlabel('LDA Component 1')
plt.ylabel('LDA Component 2')
plt.legend()
plt.show()
```

Program:5

```
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score, precision_score,
recall_score, confusion_matrix, roc_curve, auc, precision_recall_curve
import matplotlib.pyplot as plt

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

# Preprocess the data
titanic.dropna(subset=['age', 'embarked'], inplace=True)
X = titanic[['pclass', 'sex', 'age', 'sibsp', 'parch', 'fare', 'embarked']]
X = pd.get_dummies(X, columns=['sex', 'embarked'], drop_first=True)
y = titanic['survived']

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

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

# Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train, y_train)
y_pred_rf = rf_classifier.predict(X_test)

# Calculate metrics for Decision Tree
classification_rep_dt = classification_report(y_test, y_pred_dt)
accuracy_dt = accuracy_score(y_test, y_pred_dt)
precision_dt = precision_score(y_test, y_pred_dt)
recall_dt = recall_score(y_test, y_pred_dt)

print(f'Accuracy: {accuracy_dt}')
print(f'Precision: {precision_dt}')
print(f'Recall: {recall_dt}')


# Confusion Matrix for Decision Tree
cm_dt = confusion_matrix(y_test, y_pred_dt)
sns.heatmap(cm_dt, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Confusion Matrix - Decision Tree')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

print('Confusion Matrix:\n', cm_dt)

# Precision-Recall Curve for Decision Tree
precision_dt, recall_dt, _ = precision_recall_curve(y_test,
dt_classifier.predict_proba(X_test)[:, 1])
plt.figure(figsize=(8, 6))
plt.plot(recall_dt, precision_dt, color='blue', label='Decision Tree')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve - Decision Tree')
plt.legend()
plt.show()

# Calculate metrics for Random Forest
classification_rep_rf = classification_report(y_test, y_pred_rf)
accuracy_rf = accuracy_score(y_test, y_pred_rf)
precision_rf = precision_score(y_test, y_pred_rf)
recall_rf = recall_score(y_test, y_pred_rf)

print(f'Accuracy: {accuracy_rf}')
print(f'Precision: {precision_rf}')
print(f'Recall: {recall_rf}')

# Confusion Matrix for Random Forest
cm_rf = confusion_matrix(y_test, y_pred_rf)
sns.heatmap(cm_rf, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Confusion Matrix - Random Forest')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
print('Confusion Matrix:\n', cm_rf)
# Precision-Recall Curve for Random Forest
precision_rf, recall_rf, _ = precision_recall_curve(y_test,
rf_classifier.predict_proba(X_test)[:, 1])
plt.figure(figsize=(8, 6))
plt.plot(recall_rf, precision_rf, color='green', label='Random Forest')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve - Random Forest')
plt.legend()
plt.show()
```

Program:7a

```
# Install required libraries
!pip install -q pandas numpy matplotlib scikit-learn

# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans, AffinityPropagation, Birch
from sklearn.metrics import silhouette_score, davies_bouldin_score, calinski_harabasz_score
from sklearn.datasets import load_iris

# Load Iris dataset
iris = load_iris()
data = pd.DataFrame(data= np.c_[iris['data'], iris['target']], columns= iris['feature_names'] +
['target'])

# Select relevant features for clustering
selected_features = ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width
(cm)']
X = data[selected_features]

# K-means clustering function
def kmeans_clustering(X, n_clusters=3):
   model = KMeans(n_clusters=n_clusters, random_state=42)
   labels = model.fit_predict(X)
   return labels

# Affinity Propagation clustering function
def affinity_propagation_clustering(X):
   model = AffinityPropagation()
   labels = model.fit_predict(X)
   return labels

# Birch clustering function
def birch_clustering(X, n_clusters=3):
   model = Birch(n_clusters=n_clusters)
   labels = model.fit_predict(X)
   return labels

# Function to evaluate clustering metrics
def evaluate_clustering(X, labels, algorithm):
   silhouette = silhouette_score(X, labels)
   db_index = davies_bouldin_score(X, labels)
   ch_index = calinski_harabasz_score(X, labels)

   print(f'Evaluation Metrics for {algorithm}:')
   print(f'Silhouette Score: {silhouette}')
   print(f'Davies-Bouldin Index: {db_index}')
```

```
    print(f'Calinski-Harabasz Index: {ch_index}\n')

# Function to plot clusters
def plot_clusters(X, labels, algorithm):
    plt.scatter(X.iloc[:, 0], X.iloc[:, 1], c=labels, cmap='viridis', marker='o', edgecolors='k')
    plt.title(f'{algorithm} Clustering')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.show()

# Apply K-means clustering
kmeans_labels = kmeans_clustering(X)
evaluate_clustering(X, kmeans_labels, 'K-Means Clustering')
plot_clusters(X, kmeans_labels, 'K-Means')

# Apply Affinity Propagation clustering
affinity_labels = affinity_propagation_clustering(X)
evaluate_clustering(X, affinity_labels, 'Affinity Propagation')
plot_clusters(X, affinity_labels, 'Affinity Propagation')

# Apply Birch clustering
birch_labels = birch_clustering(X)
evaluate_clustering(X, birch_labels, 'Birch Clustering')
plot_clusters(X, birch_labels, 'Birch')
```

Program:7b

```
# Install required libraries
!pip install -q pandas numpy matplotlib scikit-learn

# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering, KMeans
from sklearn.metrics import silhouette_score, davies_bouldin_score, calinski_harabasz_score
from scipy.cluster.hierarchy import dendrogram, linkage

# Load CC General dataset
# Replace 'CC_general.csv' with the actual file path of the CC General dataset
cc_data = pd.read_csv('/content/CC GENERAL.csv')

# Drop non-numeric columns and handle missing values (customize based on your dataset)
X = cc_data.drop(['CUST_ID', 'TENURE'], axis=1).fillna(0)

# Agglomerative Hierarchical Clustering function
def hierarchical_clustering(X, n_clusters=4, method='ward', metric='euclidean'):
    model = AgglomerativeClustering(n_clusters=n_clusters, linkage=method,
affinity=metric)
    labels = model.fit_predict(X)
    return labels

# K-means clustering function
def kmeans_clustering(X, n_clusters=4):
    model = KMeans(n_clusters=n_clusters, random_state=42)
    labels = model.fit_predict(X)
    return labels

# Function to evaluate clustering metrics
def evaluate_clustering(X, labels, algorithm):
    silhouette = silhouette_score(X, labels)
    db_index = davies_bouldin_score(X, labels)
    ch_index = calinski_harabasz_score(X, labels)

    print(f'Evaluation Metrics for {algorithm}:')
    print(f'Silhouette Score: {silhouette}')
    print(f'Davies-Bouldin Index: {db_index}')
    print(f'Calinski-Harabasz Index: {ch_index}\n')

# Function to visualize hierarchical clustering dendrogram
def hierarchical_dendrogram(X, method='ward', metric='euclidean'):
    linkage_matrix = linkage(X, method=method, metric=metric)
    dendrogram(linkage_matrix)
    plt.title(f'Hierarchical Clustering - Method: {method}, Metric: {metric}')
    plt.show()
```

```
# Function to plot K-means clusters
def plot_kmeans_clusters(X, labels):
    plt.scatter(X.iloc[:, 0], X.iloc[:, 1], c=labels, cmap='viridis', marker='o', edgecolors='k')
    plt.title('K-Means Clustering')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.show()

# Apply Agglomerative Hierarchical Clustering
hierarchical_labels = hierarchical_clustering(X)
evaluate_clustering(X, hierarchical_labels, 'Agglomerative Hierarchical Clustering')
hierarchical_dendrogram(X)

# Apply K-means clustering
kmeans_labels = kmeans_clustering(X)
evaluate_clustering(X, kmeans_labels, 'K-Means Clustering')
plot_kmeans_clusters(X, kmeans_labels)
```

Program:8

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import fashion_mnist
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Load and preprocess the Fashion MNIST dataset
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

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

# Flatten the images to one-dimensional arrays
train_images = train_images.reshape((60000, 28 * 28))
test_images = test_images.reshape((10000, 28 * 28))

# Split the data into training and validation sets
train_images, val_images, train_labels, val_labels = train_test_split(
    train_images, train_labels, test_size=0.2, random_state=42
)

# Standardize the features
scaler = StandardScaler()
train_images = scaler.fit_transform(train_images)
val_images = scaler.transform(val_images)
test_images = scaler.transform(test_images)

# Build the neural network model
model = models.Sequential()
model.add(layers.Dense(128, activation='relu', input_shape=(28 * 28,)))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(10, activation='softmax'))

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

# Train the model
history = model.fit(train_images, train_labels, epochs=10, validation_data=(val_images,
val_labels))

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

# Plot the training and validation accuracy over epochs
```

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

def display_images(images, labels, predictions=None):
    plt.figure(figsize=(10, 4))

    for i in range(5):  # Displaying 5 examples
        plt.subplot(2, 5, i + 1)
        plt.imshow(images[i].reshape(28, 28), cmap='gray')
        plt.title(f"Digit: {np.argmax(labels[i])}")
        plt.axis('off')

        if predictions is not None:
            plt.subplot(2, 5, i + 6)
            plt.bar(range(10), predictions[i])
            plt.title(f"Prediction: {np.argmax(predictions[i])}")
            plt.xticks(range(10))

    plt.show()
display_images(test_images, test_labels)

subset_indices = np.random.choice(len(test_images), size=5, replace=False)
subset_images = test_images[subset_indices]
subset_labels = test_labels[subset_indices]
predictions = model.predict(subset_images)

display_images(subset_images, subset_labels, predictions)
```

Program:9

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Perceptron
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, auc,
precision_recall_curve
import seaborn as sns
import matplotlib.pyplot as plt
from google.colab import drive
drive.mount('/content/drive')

file_path = '/content/drive/MyDrive/ML Lab/ExNo08/healthcare-dataset-stroke-data.csv'

data = pd.read_csv(file_path)

print(data.head())

data = data.fillna(data.mean())

X = data[['age', 'hypertension', 'heart_disease', 'avg_glucose_level', 'bmi']]
y = data['stroke']

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

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

perceptron = Perceptron(max_iter=100, eta0=0.1, random_state=42)

perceptron.fit(X_train, y_train)

y_pred = perceptron.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')

final_weights = perceptron.coef_
print(f'Final Weights: {final_weights}')

conf_matrix = confusion_matrix(y_test, y_pred)

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

plt.subplot(1, 3, 1)
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['No Stroke',
'Stroke'], yticklabels=['No Stroke', 'Stroke'])
```

```
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')

plt.subplot(1, 3, 2)
fpr, tpr, thresholds = roc_curve(y_test, perceptron.decision_function(X_test))
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')

plt.subplot(1, 3, 3)
precision, recall, _ = precision_recall_curve(y_test, perceptron.decision_function(X_test))
plt.plot(recall, precision, color='green', lw=2, label='Precision-Recall curve')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')

plt.tight_layout()
plt.savefig('/content/drive/MyDrive/ML Lab/ExNo10/Figure.png',dpi=500)
plt.show()
```

PROGRAM:6

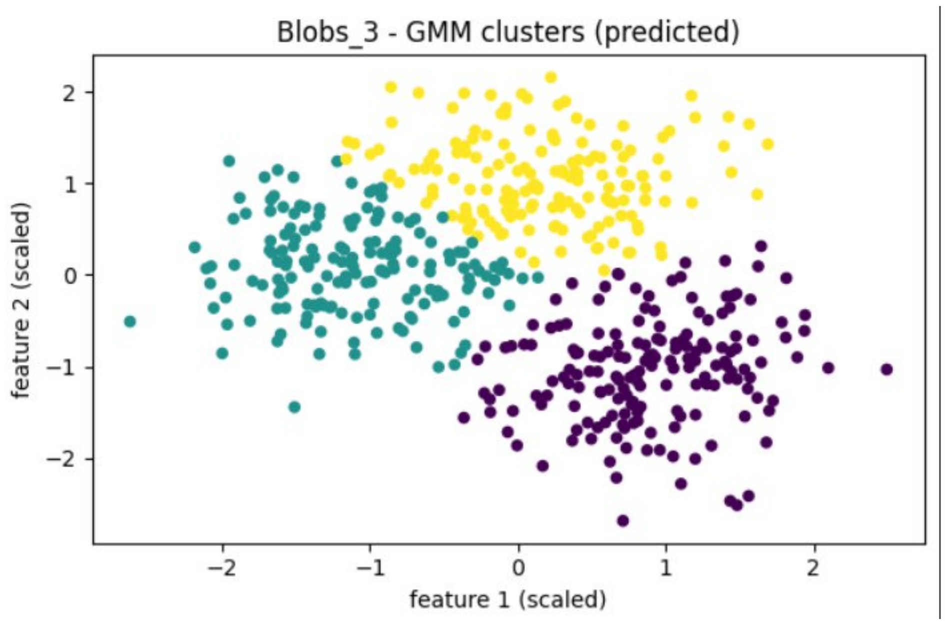
```
# Gaussian Mixture Model classifier evaluation on several datasets
# Run in Google Colab / local Python environment
# Requirements: scikit-learn, scipy, pandas (optional)
# pip install scikit-learn scipy pandas

import numpy as np
import pandas as pd
from sklearn import datasets
from sklearn.mixture import GaussianMixture
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.datasets import make_blobs
from scipy.optimize import linear_sum_assignment
from sklearn.metrics import confusion_matrix

def best_label_mapping(y_true, y_pred):
    """
    Map cluster labels (y_pred) to true labels (y_true) using the Hungarian algorithm.
    Returns mapped predictions.
    """
    cm = confusion_matrix(y_true, y_pred)
    # We want to maximize trace after permuting columns: convert to cost by negation
    row_ind, col_ind = linear_sum_assignment(cm.max() - cm)
    mapping = {}
    for r, c in zip(row_ind, col_ind):
        mapping[c] = r  # map predicted label c -> true label r
    # create mapped prediction array
    y_pred_mapped = np.array([mapping.get(lbl, -1) for lbl in y_pred])
    return y_pred_mapped, mapping, cm

def evaluate_gmm_classifier(X, y, n_components=None, test_size=0.3,
random_state=42, scale=True, covariance_type='full'):
    """
    Fits GaussianMixture on training set and evaluates on test set.
    Returns accuracy, report and details.
    """
    if n_components is None:
        n_components = len(np.unique(y))
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, stratify=y,
random_state=random_state)

    if scale:
        scaler = StandardScaler()
        X_train = scaler.fit_transform(X_train)
        X_test = scaler.transform(X_test)

    gmm = GaussianMixture(n_components=n_components,
covariance_type=covariance_type, random_state=random_state)
```

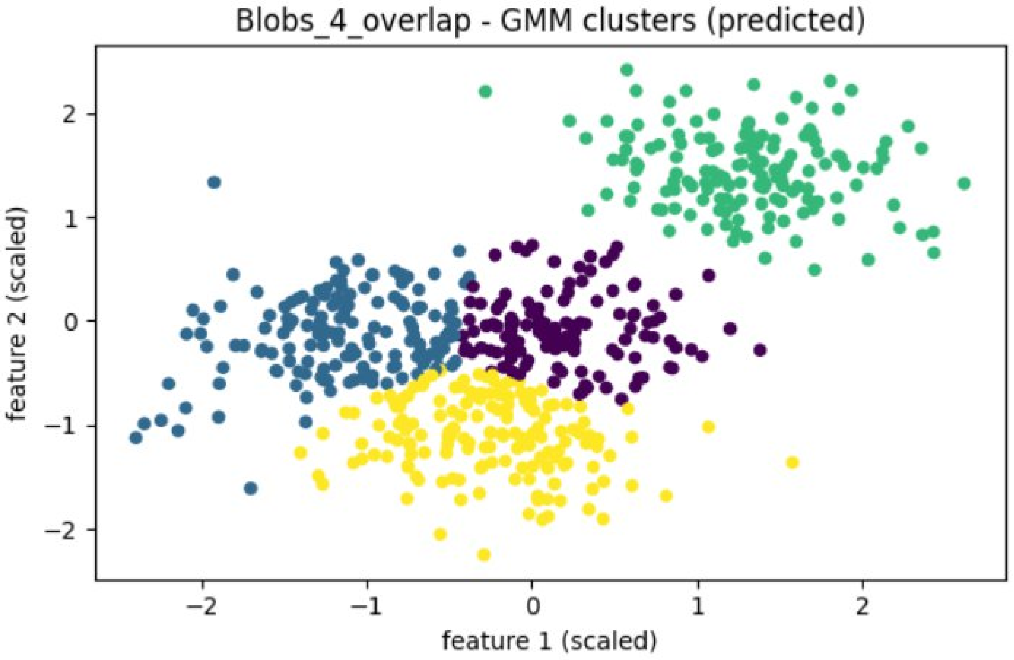
```
    gmm.fit(X_train)

    # For GMM as classifier: predict cluster assignments
    y_pred_test = gmm.predict(X_test)
    y_pred_train = gmm.predict(X_train)

    # Map clusters to actual labels for test set
    y_pred_test_mapped, mapping, cm = best_label_mapping(y_test, y_pred_test)

    acc = accuracy_score(y_test, y_pred_test_mapped)
    report = classification_report(y_test, y_pred_test_mapped, zero_division=0)
    return {
        'accuracy': acc,
        'report': report,
        'mapping': mapping,
        'confusion_matrix': cm,
        'gmm_model': gmm,
        'y_test': y_test,
        'y_pred_test': y_pred_test,
        'y_pred_test_mapped': y_pred_test_mapped
    }

def run_on_datasets(datasets_list):
    results = {}
    for name, (X, y) in datasets_list.items():
        print(f"\n=== Dataset: {name} | samples: {X.shape[0]} features: {X.shape[1]}
classes: {len(np.unique(y))} ===")
        res = evaluate_gmm_classifier(X, y, n_components=len(np.unique(y)),
test_size=0.3, random_state=42, scale=True)
        print(f"Accuracy: {res['accuracy']:.4f}")
        print("Mapping (pred_cluster -> true_label):", res['mapping'])
        print("Confusion matrix (rows=true labels, cols=pred clusters):\n",
res['confusion_matrix'])
        print("Classification report:\n", res['report'])
        results[name] = res
    return results

# Prepare datasets
datasets_list = {}

# 1) Iris
iris = datasets.load_iris()
datasets_list['Iris'] = (iris.data, iris.target)

# 2) Wine
wine = datasets.load_wine()
datasets_list['Wine'] = (wine.data, wine.target)

# 3) Synthetic blobs (3 clusters)
```

```
Xb, yb = make_blobs(n_samples=500, centers=3, n_features=2, cluster_std=1.0,
random_state=0)
datasets_list['Blobs_3'] = (Xb, yb)

# 4) Synthetic blobs (4 clusters, overlapping)
Xb4, yb4 = make_blobs(n_samples=600, centers=4, n_features=2, cluster_std=2.0,
random_state=1)
datasets_list['Blobs_4_overlap'] = (Xb4, yb4)

# 5) Optionally: load your custom CSV dataset (uncomment and edit path)
# df = pd.read_csv('/path/to/your.csv')
# X_custom = df.drop('target_column', axis=1).values
# y_custom = df['target_column'].values
# datasets_list['Custom'] = (X_custom, y_custom)

# Run experiments
results = run_on_datasets(datasets_list)

# If you want to visualize results for synthetic data (optional)
try:
    import matplotlib.pyplot as plt
    for name in ['Blobs_3', 'Blobs_4_overlap']:
        X, y = datasets_list[name]
        res = results[name]
        gmm = res['gmm_model']
        # scale for plotting
        from sklearn.preprocessing import StandardScaler
        scaler = StandardScaler().fit(X)
        Xs = scaler.transform(X)
        y_pred = gmm.predict(Xs)
        plt.figure(figsize=(6,4))
        plt.scatter(Xs[:,0], Xs[:,1], c=y_pred, s=20)
        plt.title(f"{name} - GMM clusters (predicted)")
        plt.xlabel("feature 1 (scaled)")
        plt.ylabel("feature 2 (scaled)")
        plt.tight_layout()
    plt.show()
except Exception as e:
    print("Plotting skipped or failed:", e)
```

program:4a

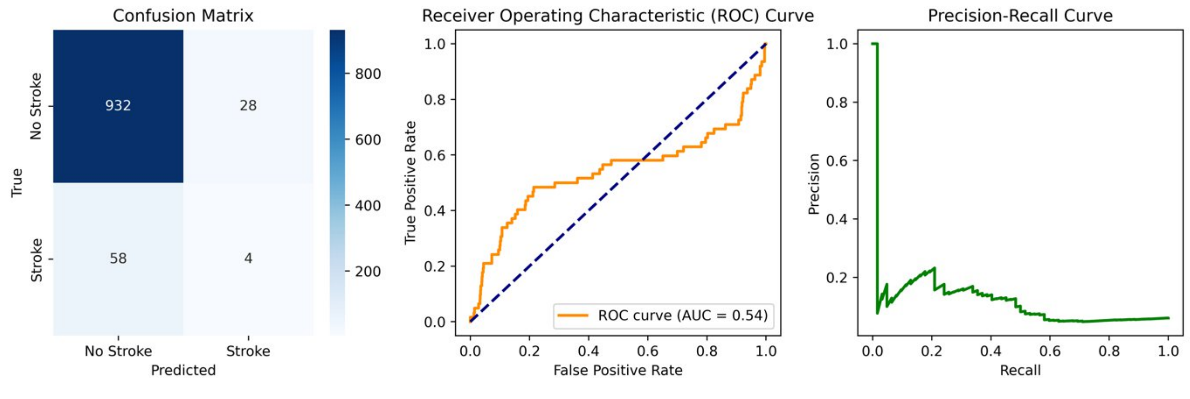
```
!pip install pyfpgrowth

import pyfpgrowth
import matplotlib.pyplot as plt
import seaborn as sns

buying_books_data = [
   ['Book1', 'Book2', 'Book3'],
   ['Book2', 'Book3', 'Book4'],
   ['Book1', 'Book3', 'Book5'],
   ['Book2', 'Book4', 'Book5'],
]

transactions = [tuple(transaction) for transaction in
buying_books_data]

min_support = 2
patterns = pyfpgrowth.find_frequent_patterns(transactions, min_support)

min_confidence = 0.5
rules = pyfpgrowth.generate_association_rules(patterns, min_confidence)

print("Frequent Itemsets:")
print(patterns)
print("\nAssociation Rules:")
print(rules)
itemset_labels = [', '.join(map(str, itemset)) for itemset in
patterns.keys()]
plt.figure(figsize=(25, 14))
plt.subplot(2, 2, 1)
plt.barh(itemset_labels, list(patterns.values()))
plt.xlabel('Support')
plt.ylabel('Itemsets')
plt.title('Frequent Itemsets')
plt.subplot(2, 2, 2)
sns.histplot(list(patterns.values()), bins=10, kde=True)
plt.xlabel('Support')
plt.ylabel('Frequency')
plt.title('Support Distribution')
```

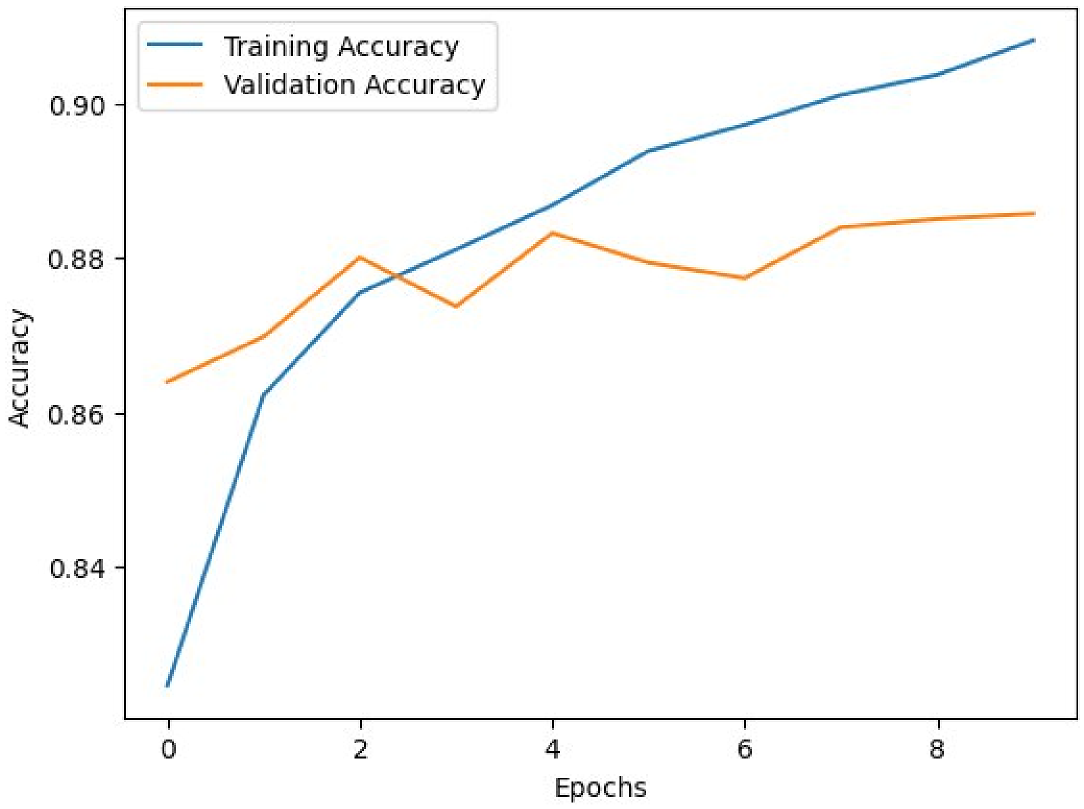


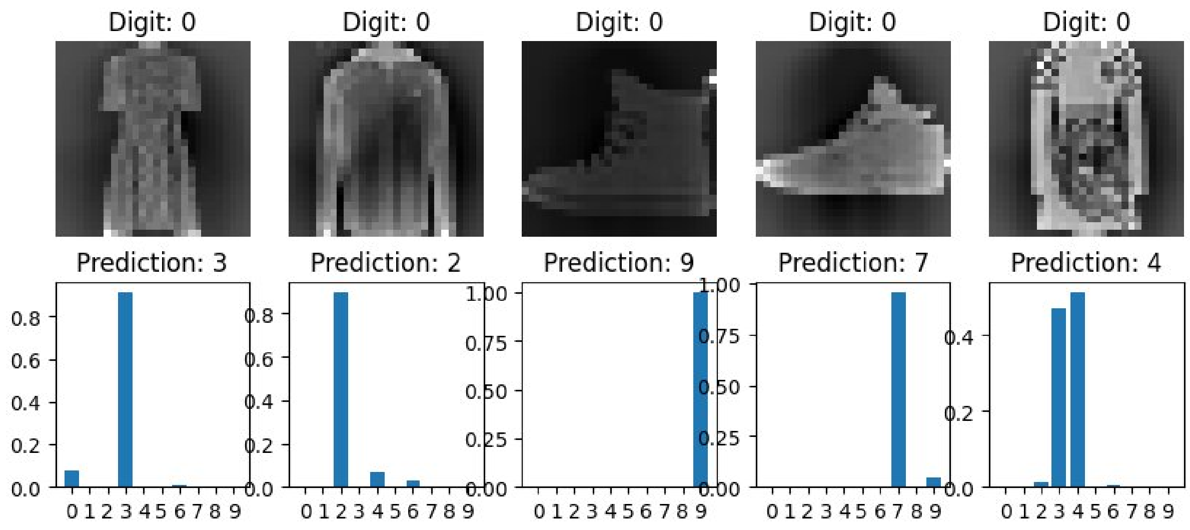


9 output

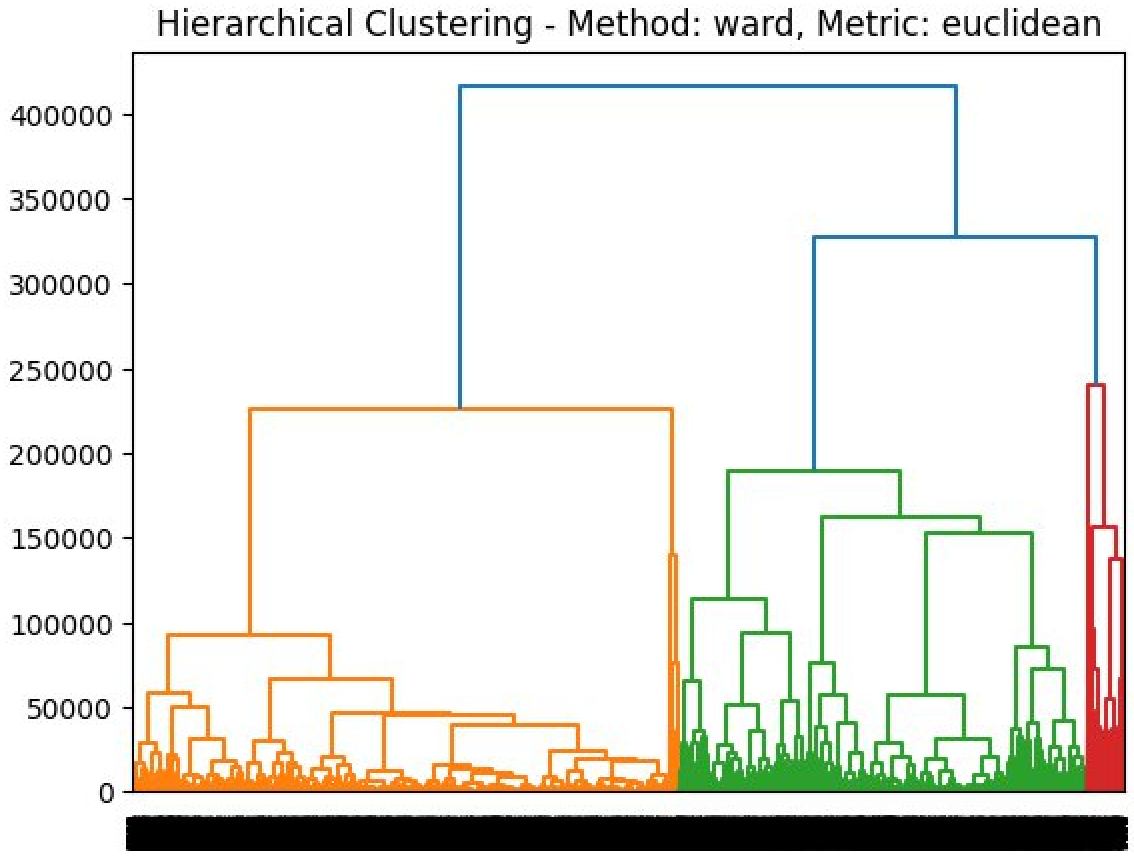
Final Weights: [[0.18583003 -0.42796629 0.188951   -0.19687049 0.06610429]]



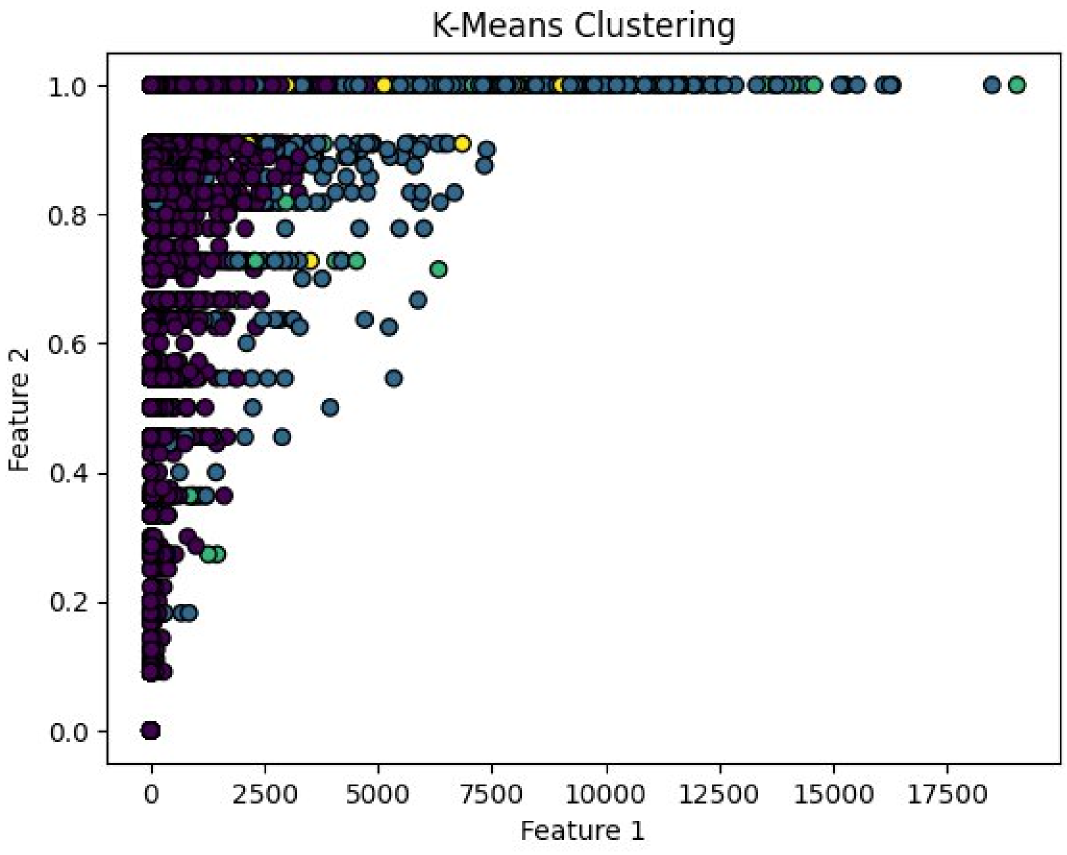

Training Accuracy
Validation Accuracy
Accuracy
Epochs
0.84
0.86
0.88
0.90
0
2
4
6
8


Digit: 0
Digit: 0
Digit: 0
Digit: 0
Digit: 0
Prediction: 3
Prediction: 2
Prediction: 9
Prediction: 7
Prediction: 4

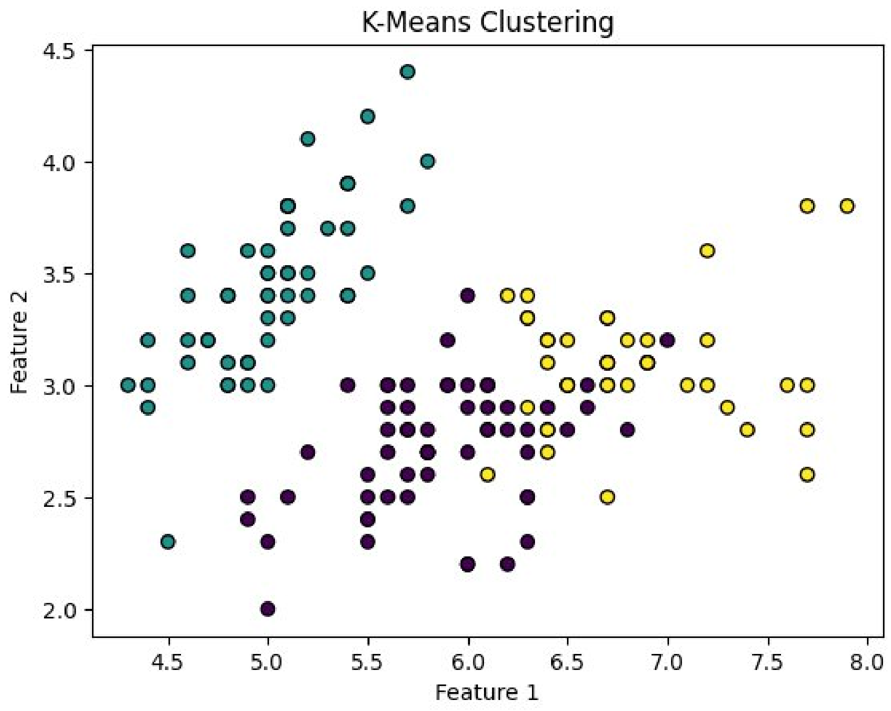
7b

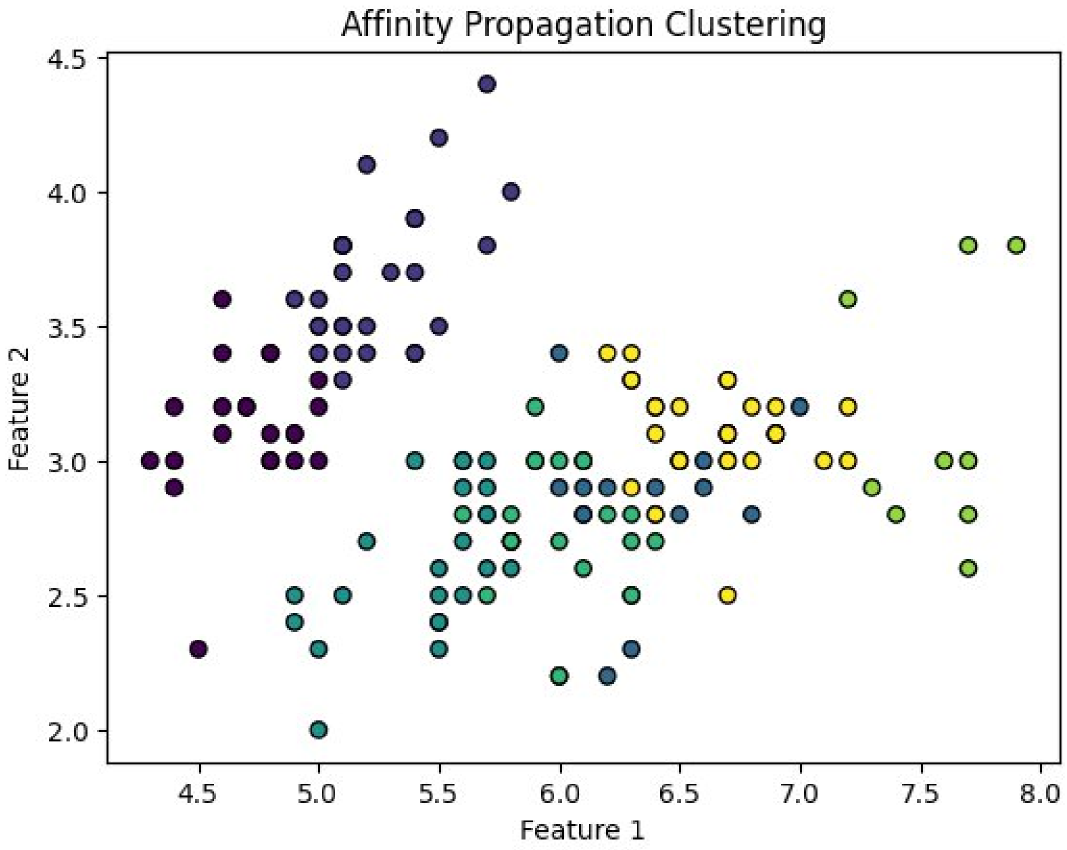

Hierarchical Clustering - Method: ward, Metric: euclidean
400000
350000
300000
250000
200000
150000
100000
50000
0

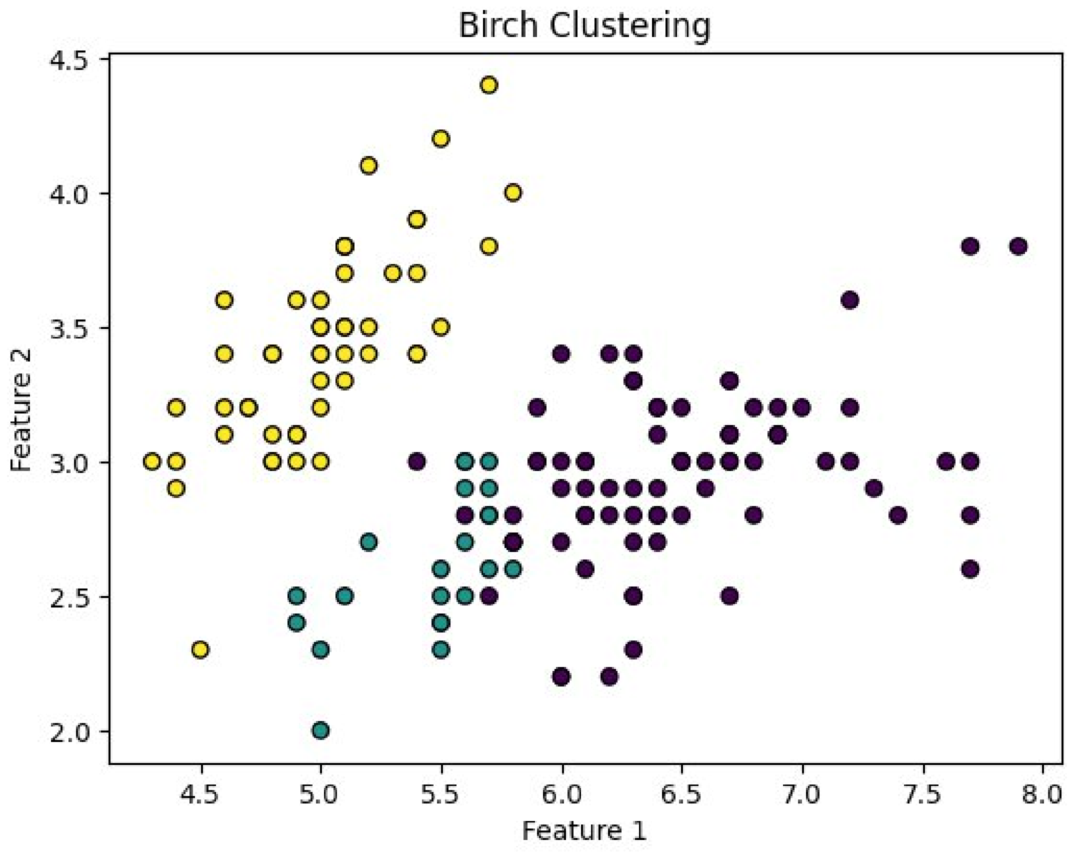
7b


K-Means Clustering
1.0
0.8
0.6
0.4
0.2
0.0
Feature 2
0
2500
5000
7500
10000
12500
15000
17500
Feature 1

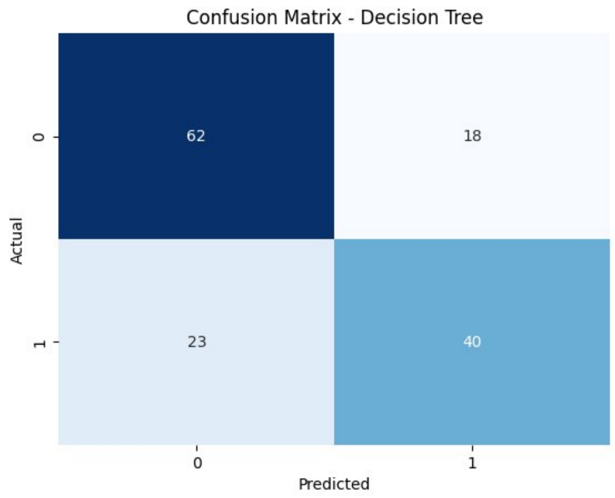
7a


K-Means Clustering
Feature 2
Feature 1
4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0

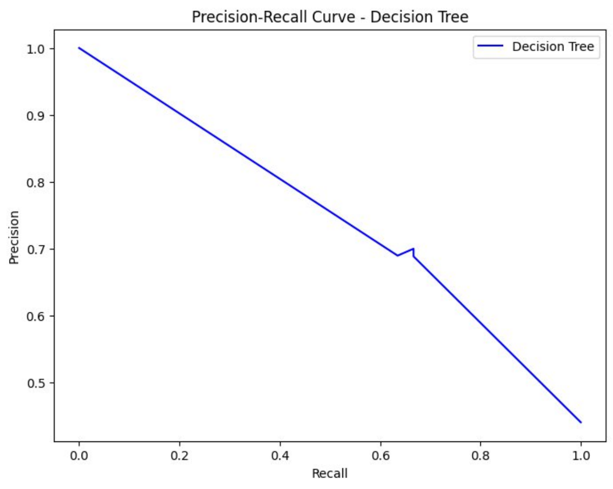

Affinity Propagation Clustering
Feature 2
Feature 1
4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0



Output:5

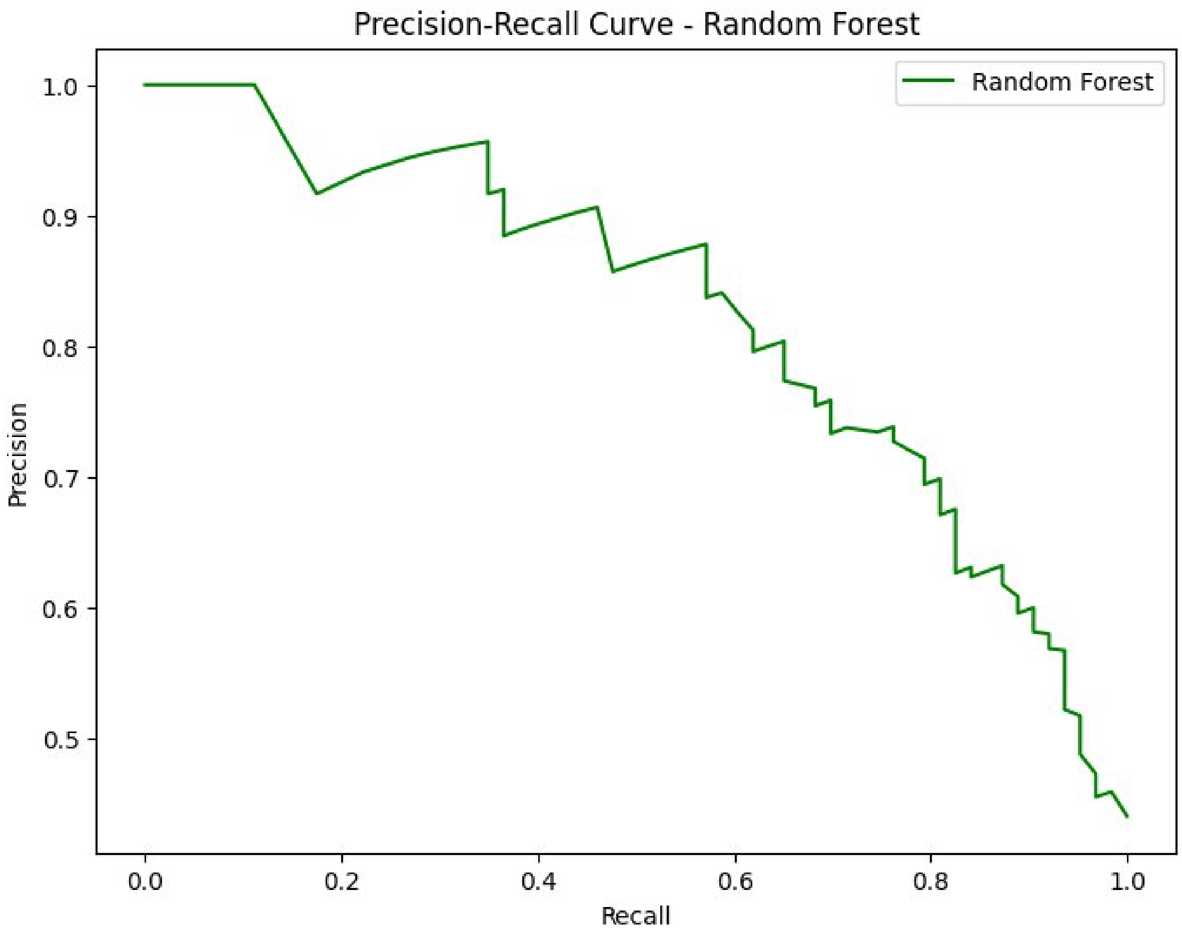


```
Confusion Matrix:
[[62 18]
 [23 40]]
```

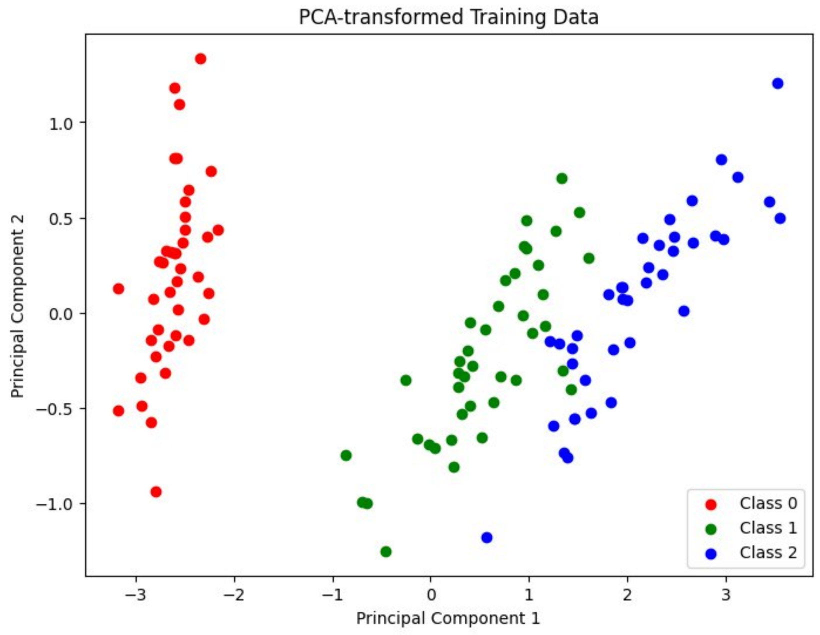


Accuracy: 0.7622377622377622

5

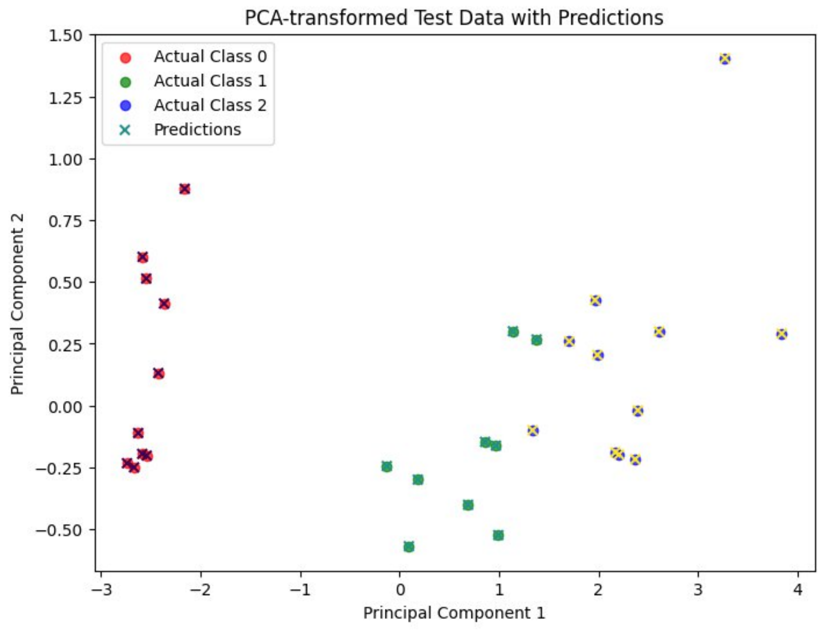


3a

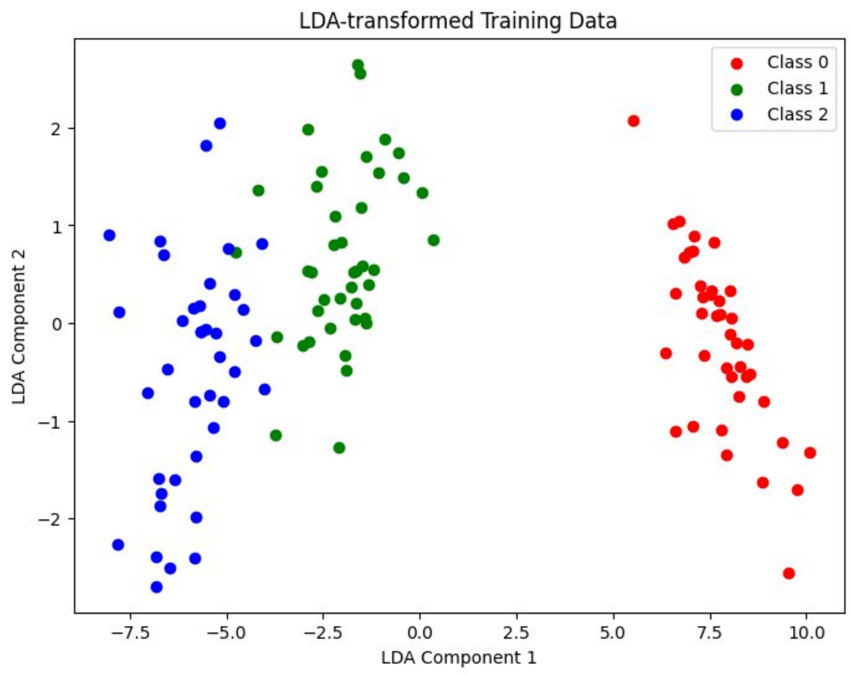


Accuracy on the test set: 1.00

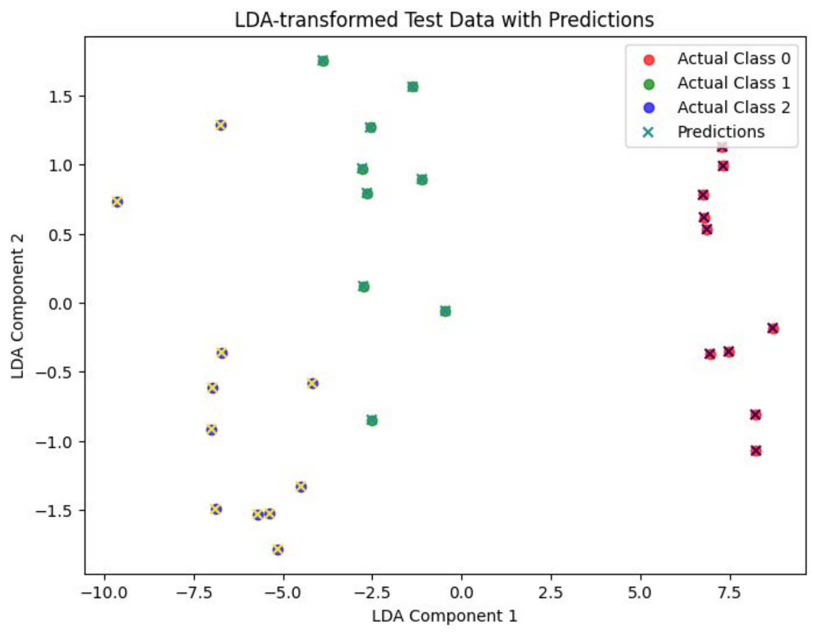
3a



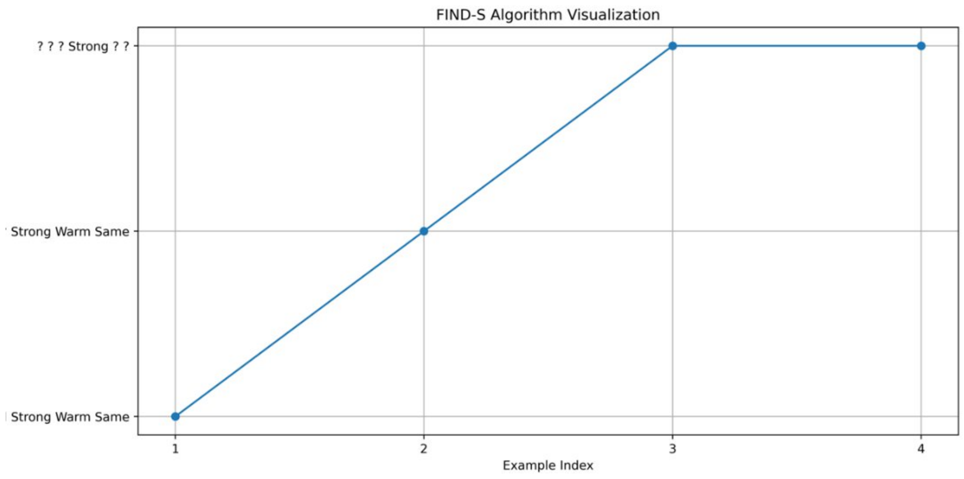
3b


LDA-transformed Training Data
Class 0
Class 1
Class 2
LDA Component 1
LDA Component 2

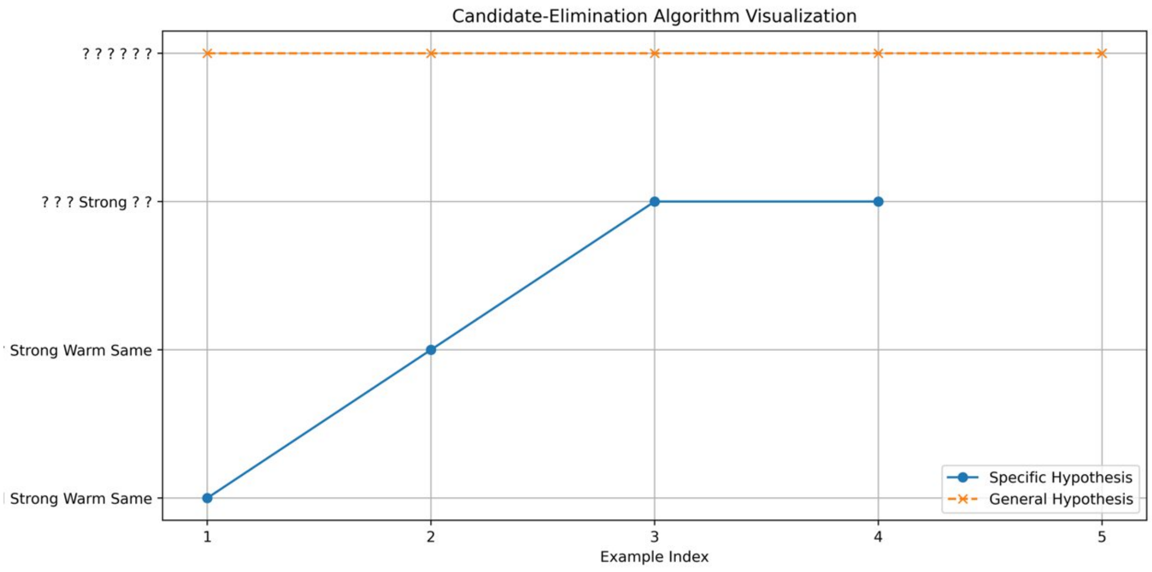
3b

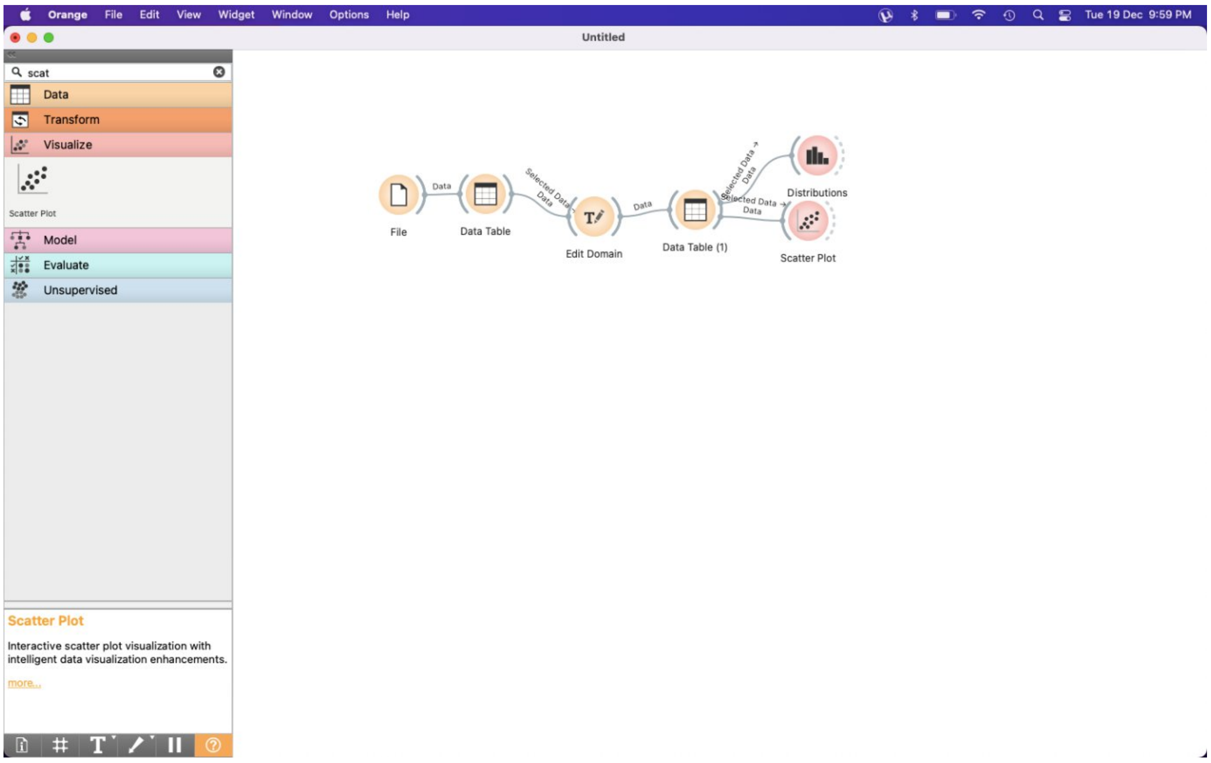

LDA-transformed Test Data with Predictions
Actual Class 0
Actual Class 1
Actual Class 2
Predictions
LDA Component 1
LDA Component 2

Output:1a



Output:1b





Scatter Plot

Interactive scatter plot visualization with intelligent data visualization enhancements.

more...