TASK 1

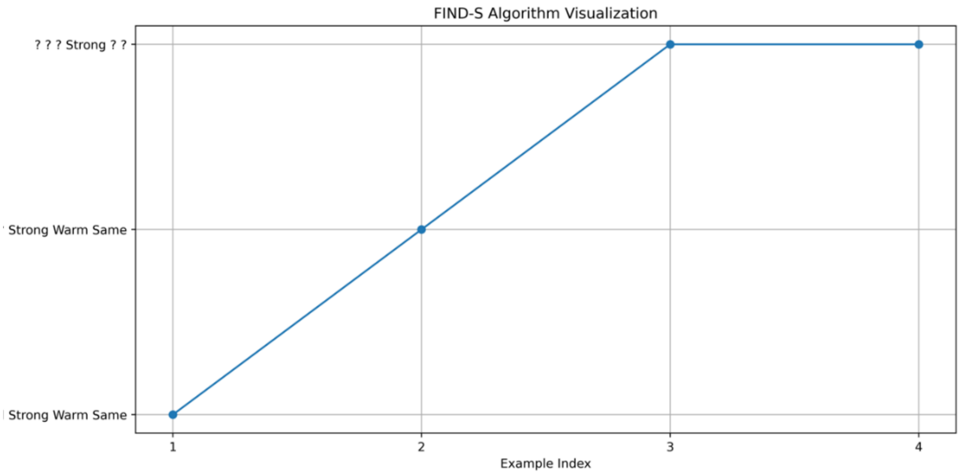
FIND-S ALGORITHM

Program :

```
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()
```

Output :



CANDIDATE-ELIMINATIO ALGORITHM

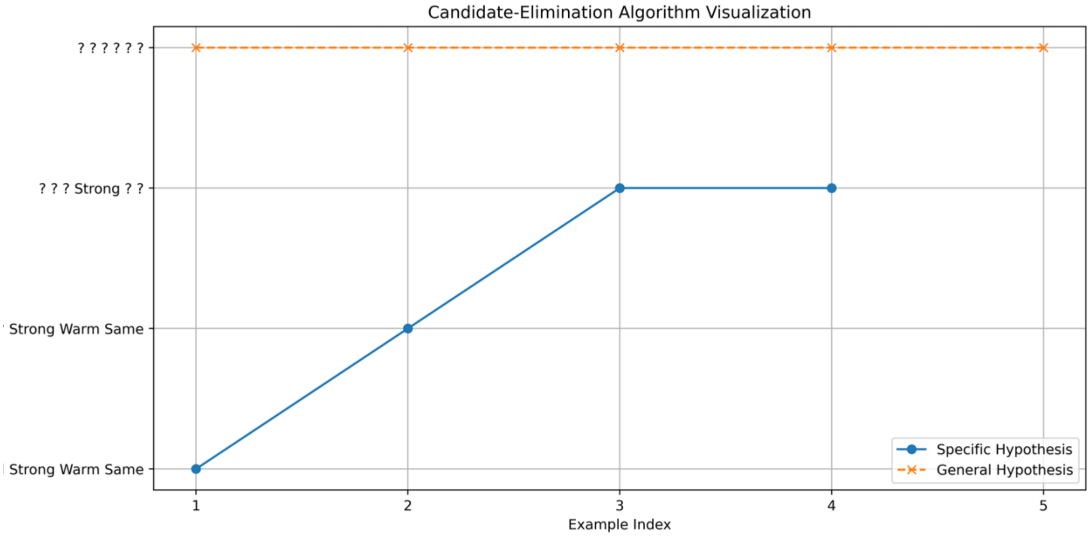
Program :

```
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()
```

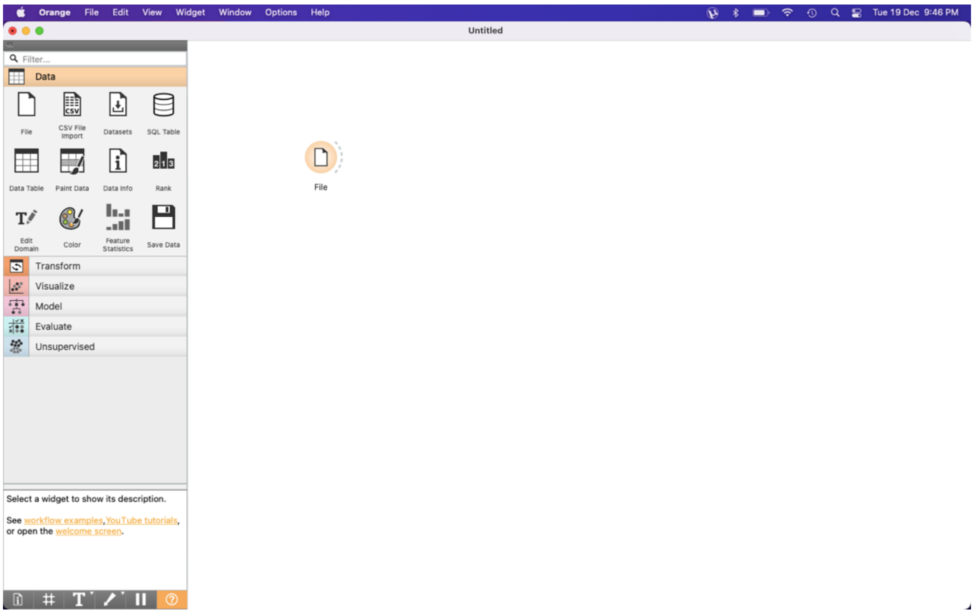
Output:



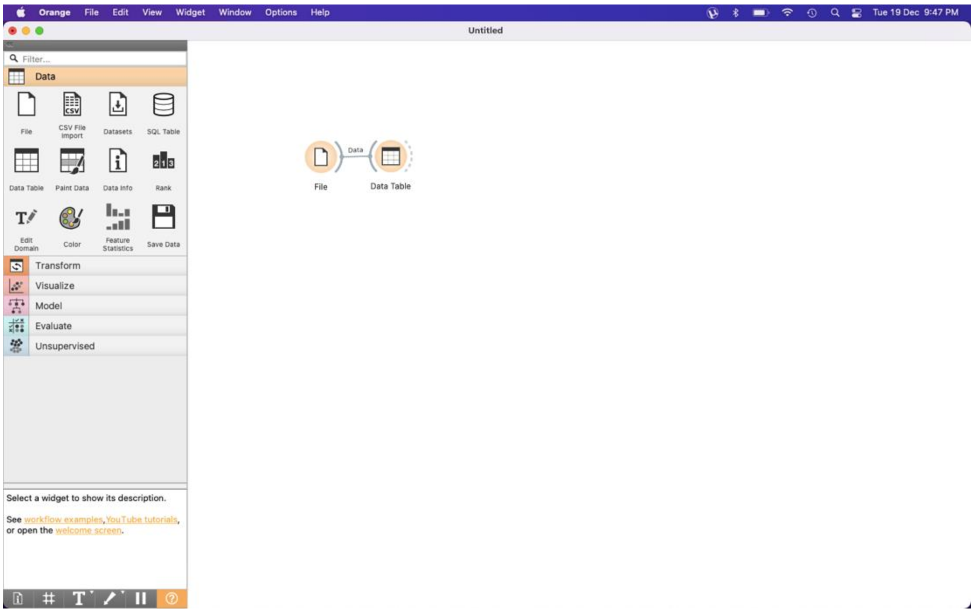
TASK 2

DATA PREPROCESSING AND VISUALIZATION TECHNIQUES

Step 1: Load the Dataset -> Drag and drop the "File" widget onto the canvas

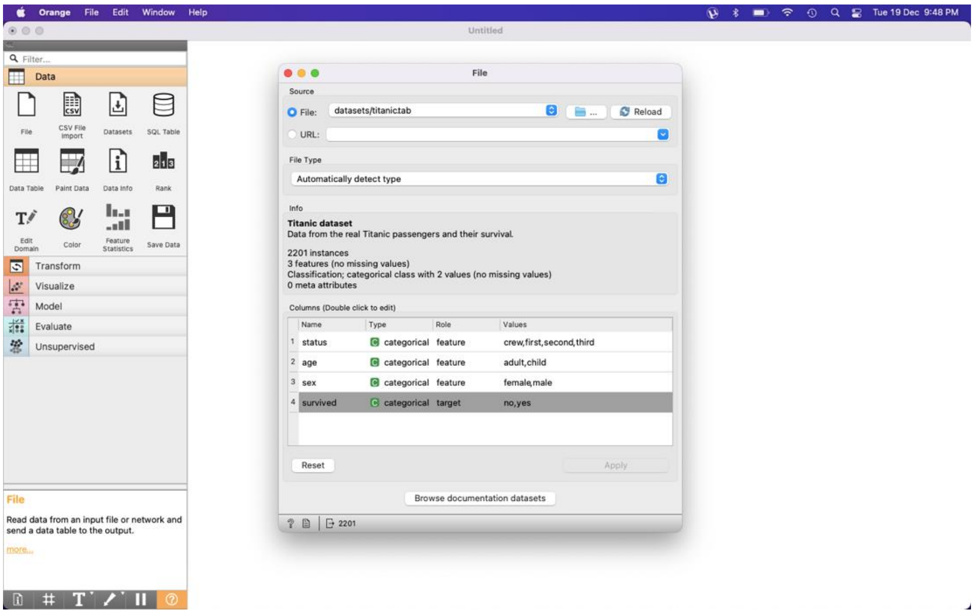


Step 2: Connect the "File" widget to the "Data Table" widget.



Step3: Load your customer dataset using the "Browse" button in the "File" widget.

(titanic dataset loaded here)

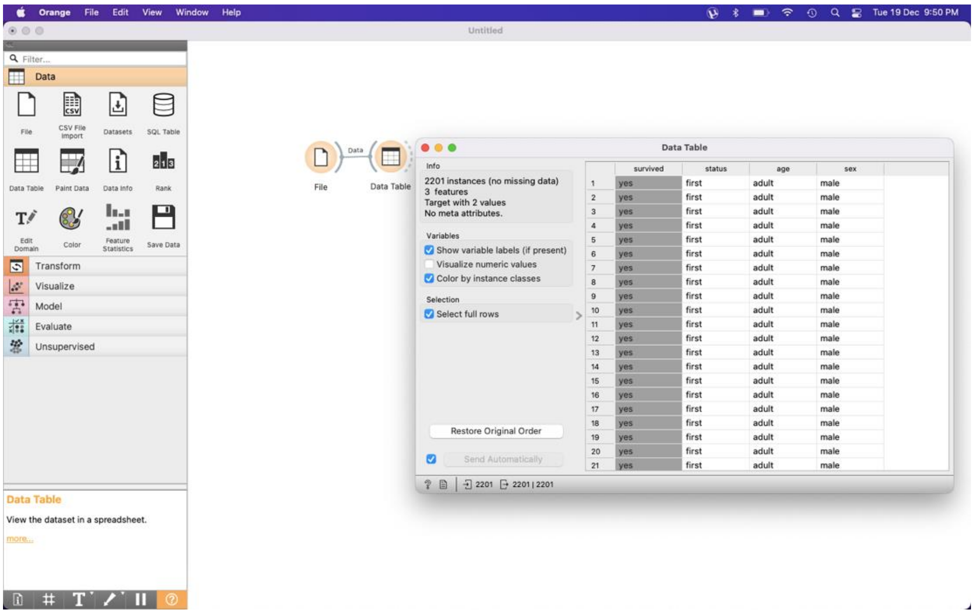


Step 4: Data Pre-processing

Drag and drop the "Data Table" widget onto the canvas.

Connect the "Data Table" widget to the "File" widget.

Use the "Data Table" widget to inspect your dataset. Address any missing values or outliers.

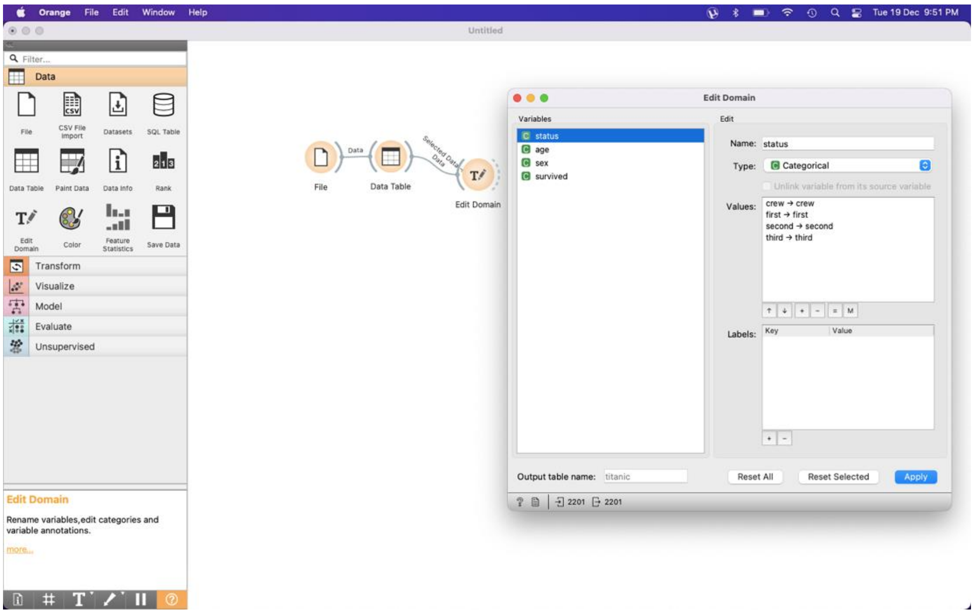


Step 5: Encode Categorical Variables

Drag and drop the "Edit Domain" widget onto the canvas.

Connect the "Edit Domain" widget to the "Data Table" widget.

In the "Edit Domain" widget, select the categorical variables and choose the appropriate encoding method.

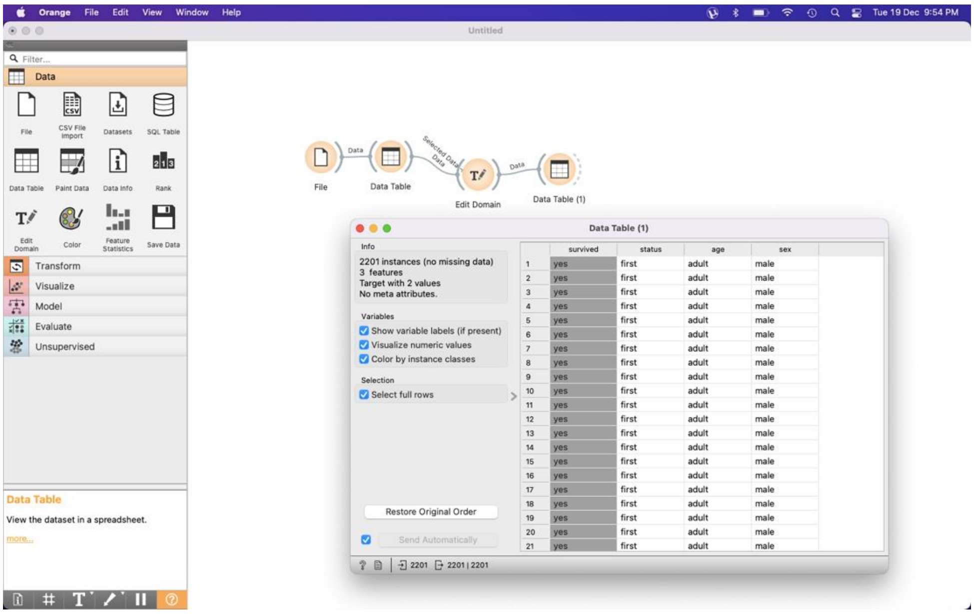


Step 6: Data Analysis

Drag and drop the "Data Table" widget onto the canvas.

Connect the "Data Table" widget to the "Edit Domain" widget.

Use the "Data Table" widget to explore basic statistics and demographics of your customers.

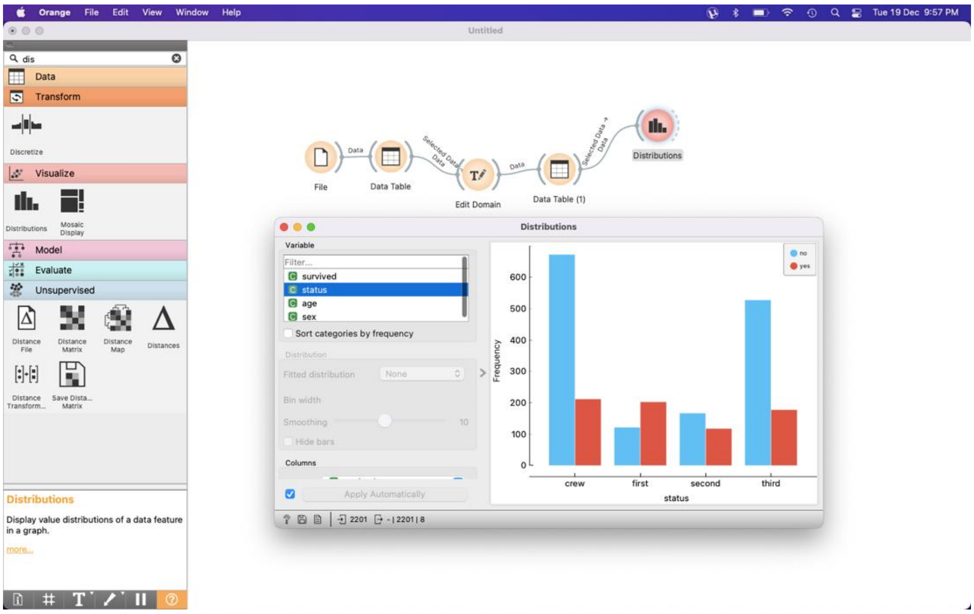


Step 7: Data Visualization

Drag and drop the "Distributions" widget onto the canvas.

Connect the "Distributions" widget to the "Edit Domain" widget.

In the "Distributions" widget, select variables like age, income, and spending to plot histograms and visualize distributions.

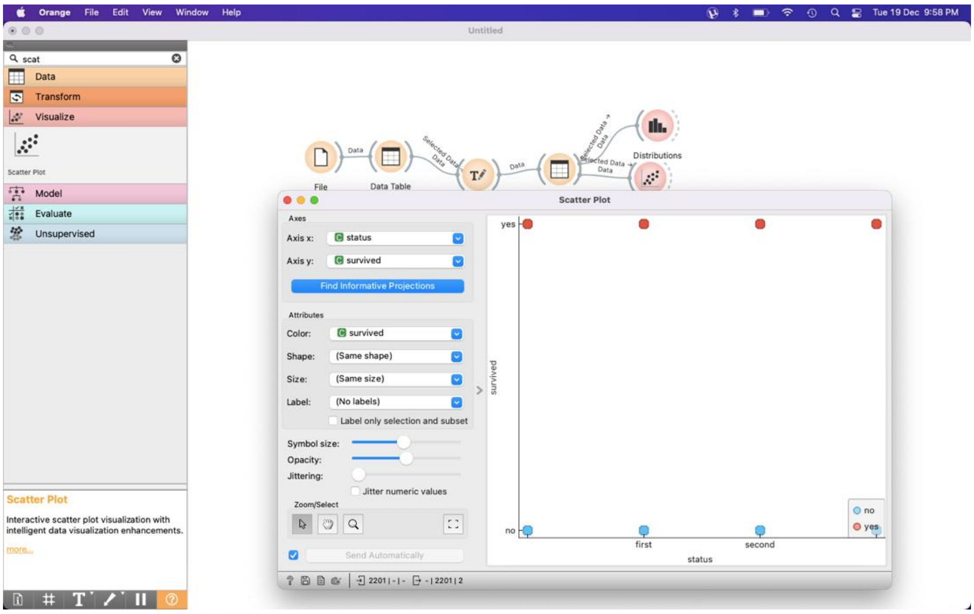


For visualizing relationships between income and spending, you can use the "Scatter Plot" widget.

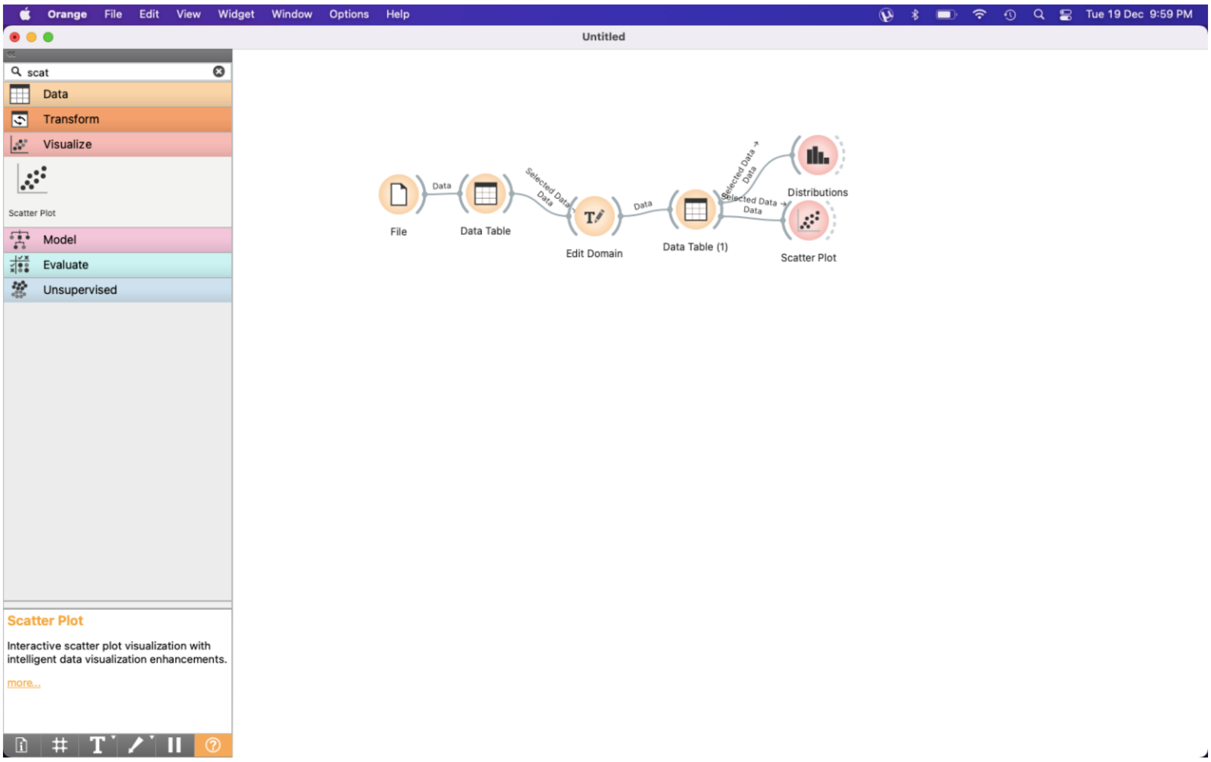
Drag and drop the "Scatter Plot" widget onto the canvas.

Connect the "Scatter Plot" widget to the "Edit Domain" widget.

Choose "Income" as the X-axis variable and "Spending" as the Y-axis variable.



Final Design:



TASK-3

PCA ALGORITHM

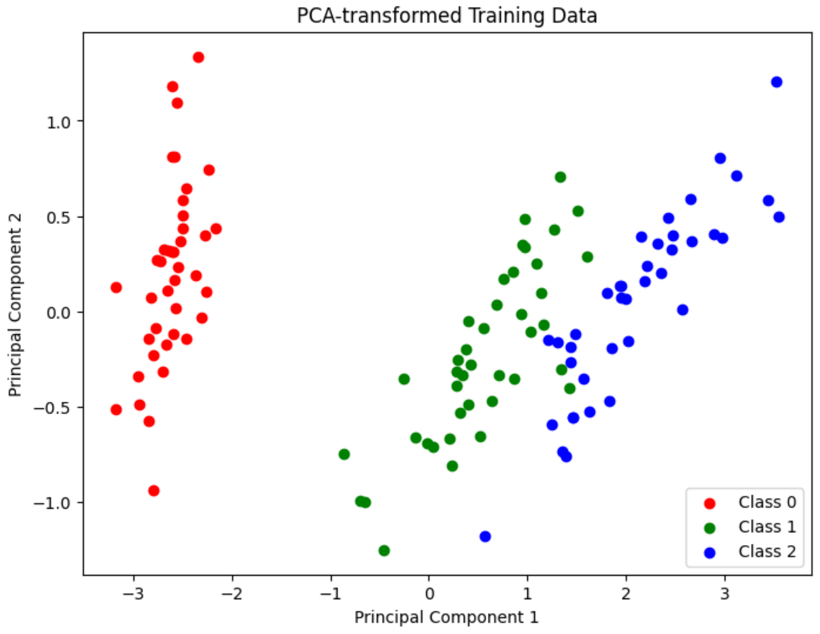
Program :

```
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):
```

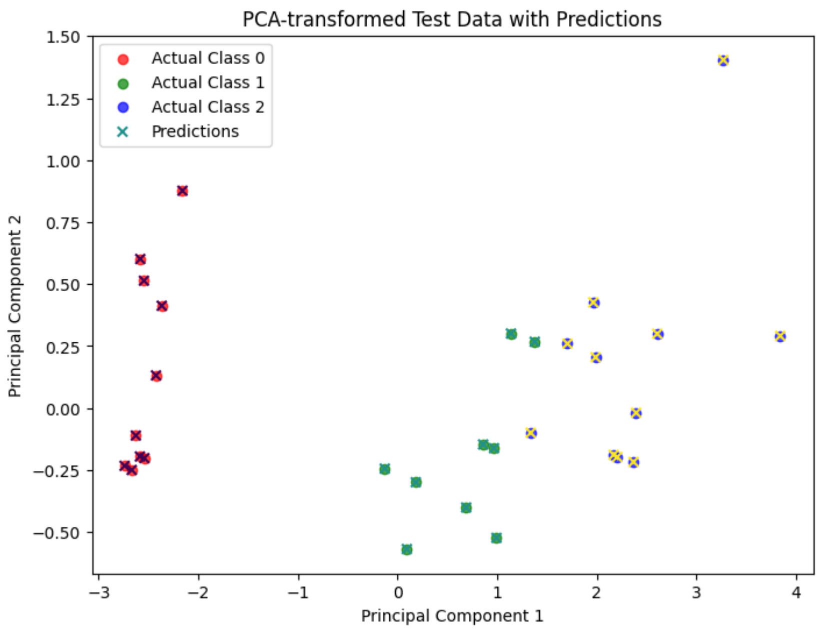
```
    plt.scatter(X_train_pca[y_train == i, 0], X_train_pca[y_train == i, 1], label=f'Class {i}', color=colors[i])

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()
```

Output :




```
Accuracy on the test set: 1.00
```

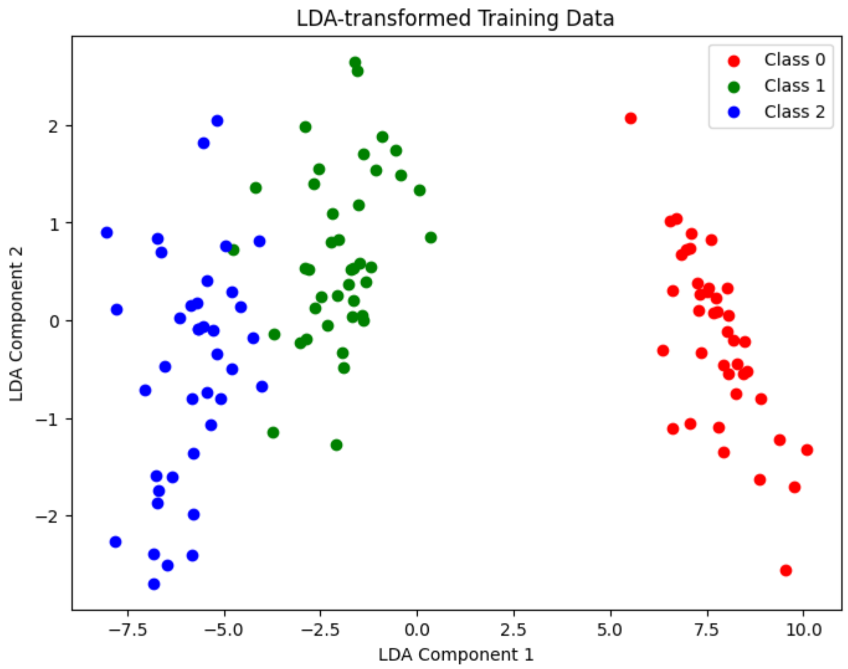


LDA ALGORITHM

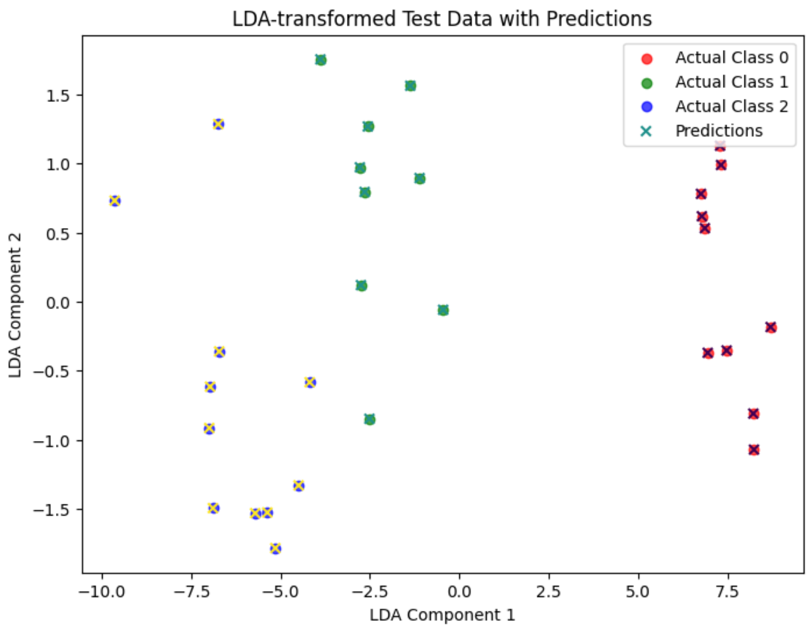
```
import numpy as np

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()
```

Output :




```
Accuracy on the test set: 1.00
```



TASK 4

Program:

```
!pip install mlxtend
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules
# Sample dataset (replace this with your actual dataset)
buying_books_data = [
['Book1', 'Book2', 'Book3'],
['Book2', 'Book3', 'Book4'],
['Book1', 'Book3', 'Book5'],
['Book2', 'Book4', 'Book5'],
]
# Convert the dataset to a one-hot encoded format
te = TransactionEncoder()
te_ary = te.fit(buying_books_data).transform(buying_books_data)
df_buying_books = pd.DataFrame(te_ary, columns=te.columns_)
# Apply the Apriori algorithm
min_support = 0.2  # Adjust as needed
frequent_itemsets = apriori(df_buying_books, min_support=min_support,
use_colnames=True)
# Generate association rules
min_confidence = 0.5  # Adjust as needed
rules = association_rules(frequent_itemsets, metric='confidence',
min_threshold=min_confidence)
# Display frequent itemsets
print("Frequent Itemsets:")
print(frequent_itemsets)

# Display association rules
```

```
print("\nAssociation Rules:")
print(rules)
```

Output :

```
Frequent Itemsets:
   support            itemsets
0     0.50             (Book1)
1     0.75             (Book2)
2     0.75             (Book3)
3     0.50             (Book4)
4     0.50             (Book5)
5     0.25       (Book2, Book1)
6     0.50       (Book3, Book1)
7     0.25       (Book5, Book1)
8     0.50       (Book3, Book2)
9     0.50       (Book4, Book2)
10    0.25       (Book5, Book2)
11    0.25       (Book4, Book3)
12    0.25       (Book5, Book3)
13    0.25       (Book5, Book4)
14    0.25  (Book3, Book2, Book1)
15    0.25  (Book5, Book3, Book1)
16    0.25  (Book4, Book3, Book2)
17    0.25  (Book4, Book5, Book2)

Association Rules:
     antecedents     consequents  antecedent support  consequent
support  \
0        (Book1)       (Book2)             0.50
```

0.75

1 (Book3) (Book1) 0.75

0.50

2 (Book1) (Book3) 0.50

0.75

3 (Book5) (Book1) 0.50

0.50

4 (Book1) (Book5) 0.50

0.50

5 (Book3) (Book2) 0.75

0.75

6 (Book2) (Book3) 0.75

0.75

7 (Book4) (Book2) 0.50

0.75

8 (Book2) (Book4) 0.75

0.50

9 (Book5) (Book2) 0.50

0.75

10 (Book4) (Book3) 0.50

0.75

11 (Book5) (Book3) 0.50

0.75

12 (Book5) (Book4) 0.50

0.50

13 (Book4) (Book5) 0.50

0.50

14 (Book2, Book3) (Book1) 0.50

0.50

15 (Book3, Book1) (Book2) 0.50

0.75

16 (Book2, Book1) (Book3) 0.25

0.75

17 (Book1) (Book2, Book3) 0.50

0.50

18 (Book5, Book3) (Book1) 0.25

0.50

19 (Book5, Book1) (Book3) 0.25

0.75

20 (Book3, Book1) (Book5) 0.50

0.50

21 (Book5) (Book3, Book1) 0.50

0.50

22 (Book1) (Book5, Book3) 0.50

0.25

23 (Book3, Book4) (Book2) 0.25

0.75

24 (Book2, Book4) (Book3) 0.50

0.75

25 (Book2, Book3) (Book4) 0.50

0.50

26 (Book4) (Book2, Book3) 0.50

0.50

27 (Book5, Book4) (Book2) 0.25

0.75

28 (Book2, Book4) (Book5) 0.50

0.50

29 (Book5, Book2) (Book4) 0.25

0.50

30 (Book4) (Book5, Book2) 0.50

0.25

31 (Book5) (Book2, Book4) 0.50

0.50

	support	confidence	lift	leverage	conviction	zhangs_metric
0	0.25	0.500000	0.666667	-0.1250	0.50	-0.500000
1	0.50	0.666667	1.333333	0.1250	1.50	1.000000
2	0.50	1.000000	1.333333	0.1250	inf	0.500000
3	0.25	0.500000	1.000000	0.0000	1.00	0.000000
4	0.25	0.500000	1.000000	0.0000	1.00	0.000000
5	0.50	0.666667	0.888889	-0.0625	0.75	-0.333333
6	0.50	0.666667	0.888889	-0.0625	0.75	-0.333333
7	0.50	1.000000	1.333333	0.1250	inf	0.500000
8	0.50	0.666667	1.333333	0.1250	1.50	1.000000
9	0.25	0.500000	0.666667	-0.1250	0.50	-0.500000
10	0.25	0.500000	0.666667	-0.1250	0.50	-0.500000
11	0.25	0.500000	0.666667	-0.1250	0.50	-0.500000
12	0.25	0.500000	1.000000	0.0000	1.00	0.000000
13	0.25	0.500000	1.000000	0.0000	1.00	0.000000
14	0.25	0.500000	1.000000	0.0000	1.00	0.000000
15	0.25	0.500000	0.666667	-0.1250	0.50	-0.500000
16	0.25	1.000000	1.333333	0.0625	inf	0.333333
17	0.25	0.500000	1.000000	0.0000	1.00	0.000000
18	0.25	1.000000	2.000000	0.1250	inf	0.666667
19	0.25	1.000000	1.333333	0.0625	inf	0.333333
20	0.25	0.500000	1.000000	0.0000	1.00	0.000000
21	0.25	0.500000	1.000000	0.0000	1.00	0.000000
22	0.25	0.500000	2.000000	0.1250	1.50	1.000000
23	0.25	1.000000	1.333333	0.0625	inf	0.333333
24	0.25	0.500000	0.666667	-0.1250	0.50	-0.500000
25	0.25	0.500000	1.000000	0.0000	1.00	0.000000
26	0.25	0.500000	1.000000	0.0000	1.00	0.000000
27	0.25	1.000000	1.333333	0.0625	inf	0.333333

28 0.25 0.500000 1.000000 0.0000 1.00 0.000000

29 0.25 1.000000 2.000000 0.1250 inf 0.666667

30 0.25 0.500000 2.000000 0.1250 1.50 1.000000

31 0.25 0.500000 1.000000 0.0000 1.00 0.000000 dataset) buying_books_data = [['Book1', 'Book2', 'Book3'], ['Book2', 'Book3', 'Book4'], ['Book1', 'Book3', 'Book5'], ['Book2', 'Book4', 'Book5'],] # Convert the dataset to a one-hot encoded format te = TransactionEncoder() te_ary = te.fit(buying_books_data).transform(buying_books_data) df_buying_books = pd.DataFrame(te_ary, columns=te.columns_) # Apply the Apriori algorithm min_support = 0.2 # Adjust as needed frequent_itemsets = apriori(df_buying_books, min_support=min_support, use_colnames=True) # Generate association rules min_confidence = 0.5 # Adjust as needed rules = association_rules(frequent_itemsets, metric='confidence', min_threshold=min_confidence) # Display frequent itemsets print("Frequent Itemsets:") print(frequent_itemsets) # Display association rules print("\nAssociation Rules:") print(rules) OUTPUT: Frequent Itemsets: support itemsets 0 0.50 (Book1) 1 0.75 (Book2) 2 0.75 (Book3) 3 0.50 (Book4) 4 0.50 (Book5) 5 0.25 (Book2, Book1) 6 0.50 (Book3, Book1) 7 0.25 (Book5, Book1) 8 0.50 (Book3, Book2) 9 0.50 (Book4, Book2) 10 0.25 (Book5, Book2) 11 0.25 (Book4, Book3) 12 0.25 (Book5, Book3) 13 0.25 (Book5, Book4) 14 0.25 (Book3, Book2, Book1) 15 0.25 (Book5, Book3, Book1) 16 0.25 (Book4, Book3, Book2) 17 0.25 (Book4, Book5, Book2) Association Rules: antecedents consequents antecedent support consequent support \ 0 (Book1) (Book2) 0.50 0.75 1 (Book3) (Book1) 0.75 0.50 2 (Book1) (Book3) 0.50 0.75 3 (Book5) (Book1) 0.50 0.50 4 (Book1) (Book5) 0.50 0.50 5 (Book3) (Book2) 0.75 0.75 6 (Book2) (Book3) 0.75 0.75 7 (Book4) (Book2) 0.50 0.75 8 (Book2) (Book4) 0.75 0.50 9 (Book5) (Book2) 0.50 0.75 10 (Book4) (Book3) 0.50 0.75 11 (Book5) (Book3) 0.50 0.75 12 (Book5) (Book4) 0.50 0.50 13 (Book4) (Book5) 0.50 0.50 14 (Book2, Book3) (Book1) 0.50 0.50 15 (Book3, Book1) (Book2) 0.50 0.75 16 (Book2, Book1) (Book3) 0.25 0.75 17 (Book1) (Book2, Book3) 0.50 0.50 18 (Book5, Book3) (Book1) 0.25 0.50 19 (Book5, Book1) (Book3) 0.25 0.75 20 (Book3, Book1) (Book5) 0.50 0.50 21 (Book5) (Book3, Book1) 0.50 0.50 22 (Book1) (Book5, Book3) 0.50 0.25 23 (Book3, Book4) (Book2) 0.25 0.75 24 (Book2, Book4) (Book3) 0.50 0.75 25 (Book2, Book3) (Book4) 0.50 0.50 26 (Book4) (Book2, Book3) 0.50 0.50 27 (Book5, Book4) (Book2) 0.25 0.75 28 (Book2, Book4) (Book5) 0.50 0.50 29 (Book5, Book2) (Book4) 0.25 0.50 30 (Book4) (Book5, Book2) 0.50 0.25 31 (Book5) (Book2, Book4) 0.50 0.50 support confidence lift leverage conviction zhangs_metric 0 0.25 0.500000 0.666667 -0.1250 0.50 -0.500000 1 0.50 0.666667 1.333333 0.1250 1.50 1.000000 2 0.50 1.000000 1.333333 0.1250 inf 0.500000 3 0.25 0.500000 1.000000 0.0000 1.00 0.000000 4 0.25 0.500000 1.000000 0.0000 1.00 0.000000 5 0.50 0.666667 0.888889 -0.0625 0.75 -0.333333 6 0.50 0.666667 0.888889 -0.0625 0.75 -0.333333 7 0.50 1.000000 1.333333 0.1250 inf 0.500000 8 0.50 0.666667 1.333333 0.1250 1.50 1.000000 9 0.25 0.500000 0.666667 -0.1250 0.50 -0.500000 10 0.25 0.500000 0.666667 -0.1250 0.50 -0.500000 11 0.25 0.500000 0.666667 -0.1250 0.50 -0.500000 12 0.25 0.500000 1.000000 0.0000 1.00 0.000000 13 0.25 0.500000 1.000000 0.0000 1.00 0.000000 14 0.25 0.500000 1.000000 0.0000 1.00 0.000000 15 0.25 0.500000 0.666667 -0.1250 0.50 -0.500000 16 0.25 1.000000 1.333333 0.0625 inf 0.333333 17 0.25 0.500000 1.000000 0.0000 1.00 0.000000 18 0.25 1.000000 2.000000 0.1250 inf 0.666667 19 0.25 1.000000 1.333333 0.0625 inf 0.333333 20 0.25 0.500000 1.000000 0.0000 1.00 0.000000 21 0.25 0.500000 1.000000 0.0000 1.00 0.000000 22 0.25 0.500000 2.000000 0.1250 1.50 1.000000 23 0.25 1.000000 1.333333 0.0625 inf 0.333333 24 0.25 0.500000 0.666667 -0.1250 0.50 -0.500000 25 0.25 0.500000 1.000000 0.0000 1.00 0.000000 26 0.25 0.500000 1.000000 0.0000 1.00 0.000000 27 0.25 1.000000 1.333333 0.0625 inf 0.333333 28 0.25 0.500000 1.000000 0.0000 1.00 0.000000 29 0.25 1.000000 2.000000 0.1250 inf 0.666667 30 0.25 0.500000 2.000000 0.1250 1.50 1.000000 31 0.25 0.500000 1.000000 0.0000 1.00 0.000000

FP GROWTH ALGORITHM :

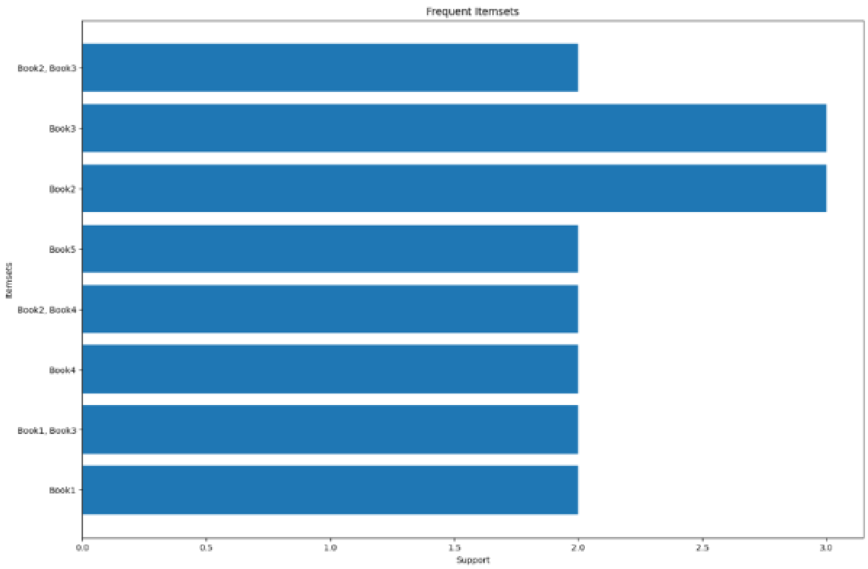
Program :

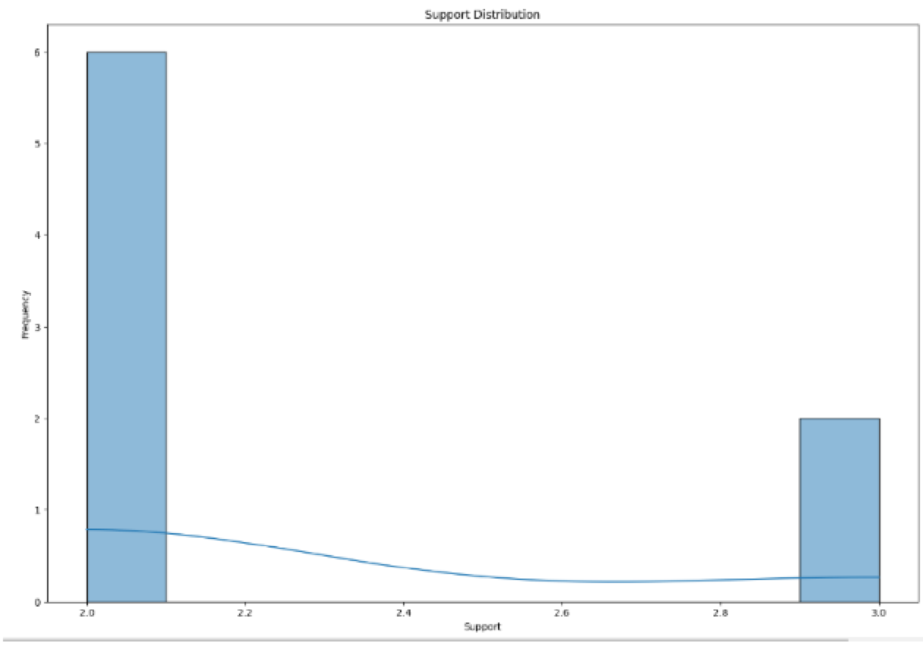
```
!pip install pyfpgrowth
import pyfpgrowth
# Sample dataset (replace this with your actual dataset)
buying_books_data = [
['Book1', 'Book2', 'Book3'],
['Book2', 'Book3', 'Book4'],
['Book1', 'Book3', 'Book5'],
['Book2', 'Book4', 'Book5'],
]
# Convert the dataset to a list of transactions
transactions = [tuple(transaction) for transaction in
buying_books_data]
# Apply the FP-growth algorithm
min_support = 2 # Adjust as needed
patterns = pyfpgrowth.find_frequent_patterns(transactions,
min_support)
# Generate association rules
min_confidence = 0.5 # Adjust as needed
rules = pyfpgrowth.generate_association_rules(patterns,
min_confidence)
# Display frequent itemsets
print("Frequent Itemsets:")
print(patterns)

# Display association rules
print("\nAssociation Rules:")
print(rules)
itemset_labels = [', '.join(map(str, itemset)) for itemset in
patterns.keys()]
plt.figure(figsize=(36, 24))
plt.subplot(2, 2, 1)
plt.barh(itemset_labels, list(patterns.values()))
plt.xlabel('Support')
plt.ylabel('Itemsets')
plt.title('Frequent Itemsets')
import seaborn as sns
plt.subplot(2, 2, 2)
sns.histplot(list(patterns.values()), bins=10, kde=True)
plt.xlabel('Support')
plt.ylabel('Frequency')
plt.title('Support Distribution')
```

OUTPUT:

```
Frequent Itemsets:
{('Book1',): 2, ('Book1', 'Book3'): 2, ('Book4',): 2,
('Book2', 'Book4'): 2, ('Book5',): 2, ('Book2',): 3,
('Book3',): 3, ('Book2', 'Book3'): 2}
```

Association Rules:
{('Book1',): (('Book3',), 1.0), ('Book3',): (('Book2',), 0.6666666666666666), ('Book2',): (('Book3',), 0.6666666666666666), ('Book4',): (('Book2',), 1.0)}





TASK 5

CLASSIFICATION

Program :

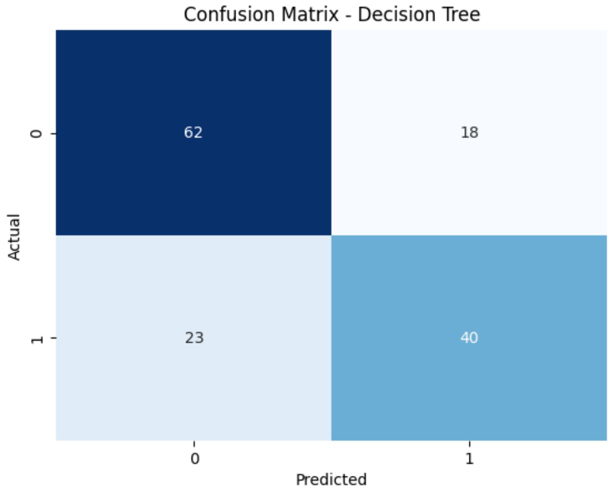
```
import pandas as pd

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')
```

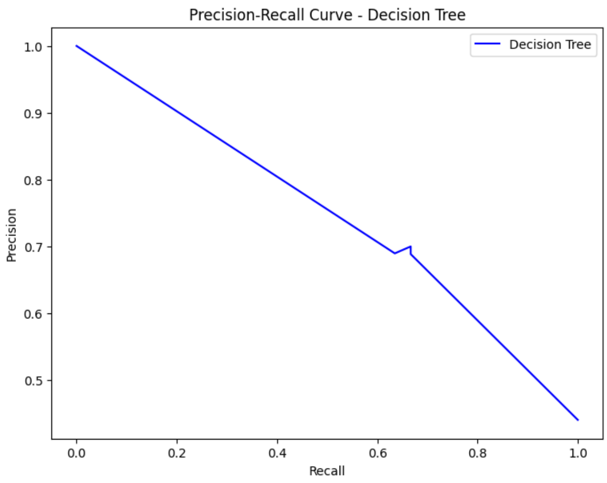
```
plt.legend()
plt.show()

# 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()

print()
```

Output :

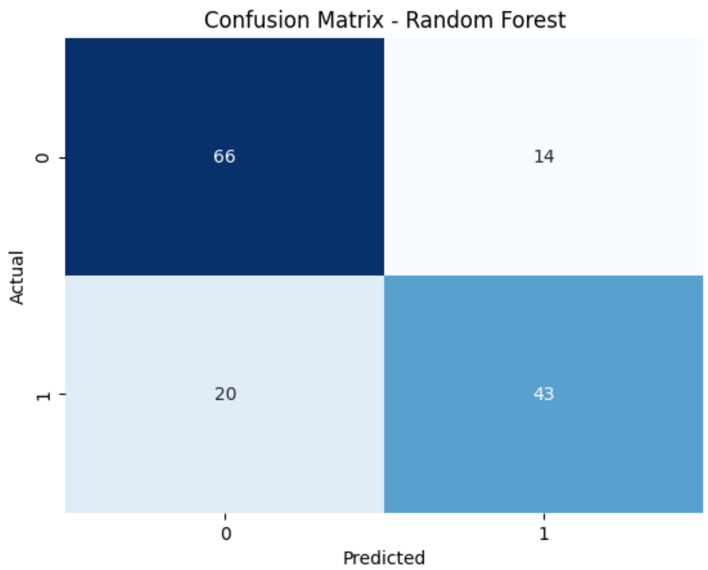
```
Accuracy: 0.7132867132867133

Precision: 0.6896551724137931

Recall: 0.6349206349206349
```



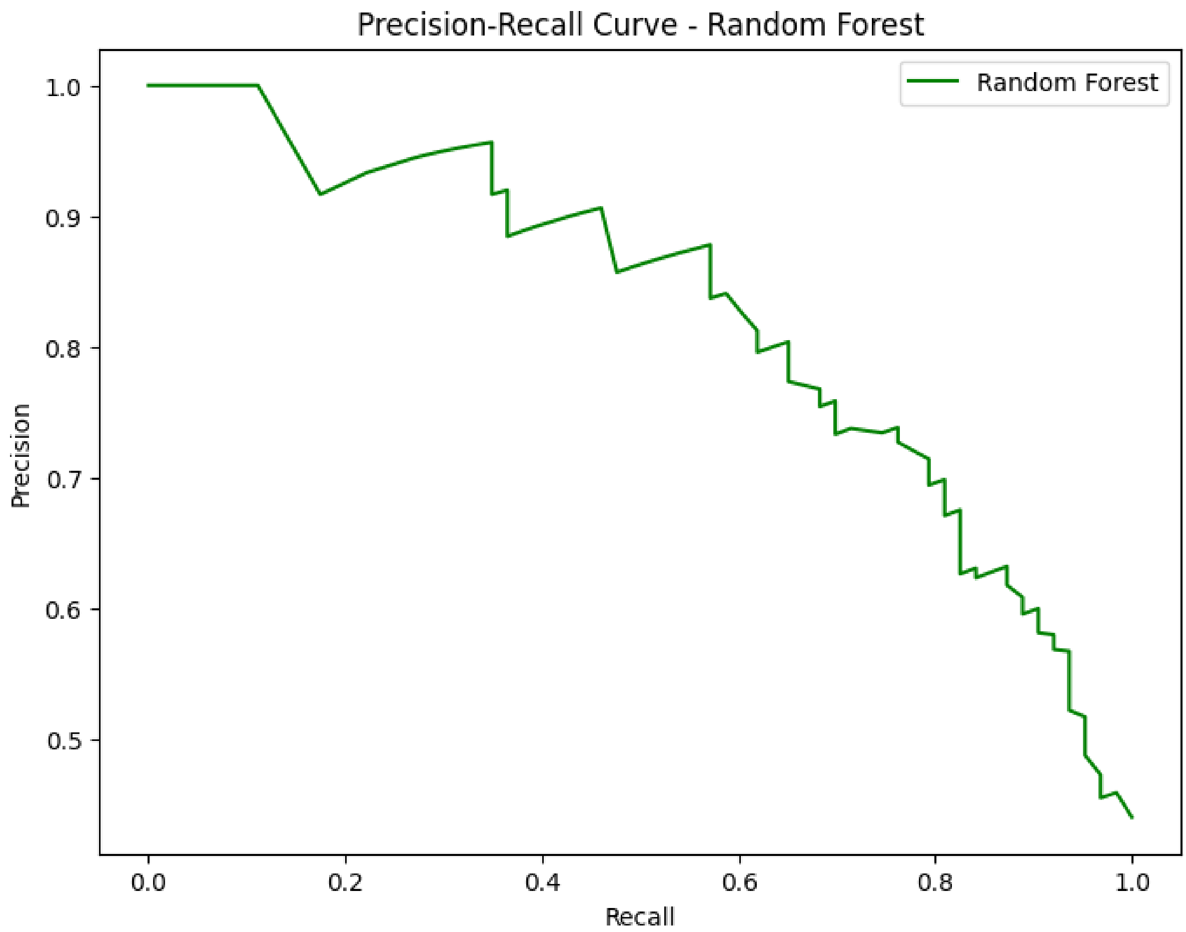

```
Confusion Matrix:

[[62 18]

 [23 40]]
```




```
Accuracy: 0.7622377622377622

Precision: 0.7543859649122807

Recall: 0.6825396825396826
```




```
Confusion Matrix:
 [[66 14]
 [20 43]]
```



TASK-6

Program :

```
# 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)
```

OUTPUT:

```
=== Dataset: Iris | samples: 150 features: 4 classes: 3 ===
Accuracy: 0.8000
Mapping (pred_cluster -> true_label): {np.int64(0): np.int64(0), np.int64(2): np.int64(1), np.int64(1): np.int64(2)}
Confusion matrix (rows=true labels, cols=pred clusters):
 [[15  0  0]
 [ 0  1 14]
 [ 0  7  8]]
Classification report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        15
           1       0.64      0.93      0.76        15
           2       0.88      0.47      0.61        15

    accuracy                           0.80        45
   macro avg       0.84      0.80      0.79        45
weighted avg       0.84      0.80      0.79        45
```

=== Dataset: Wine | samples: 178 features: 13 classes: 3 ===

Accuracy: 0.9630

Mapping (pred_cluster -> true_label): {np.int64(1): np.int64(0), np.int64(2): np.int64(1), np.int64(0): np.int64(2)}

Confusion matrix (rows=true labels, cols=pred clusters):

```
[[ 0 18  0]
 [ 2  0 19]
 [15  0  0]]
```

Classification report:

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        18
           1       1.00      0.90      0.95        21
           2       0.88      1.00      0.94        15

    accuracy                           0.96        54
   macro avg       0.96      0.97      0.96        54
weighted avg       0.97      0.96      0.96        54
```

=== Dataset: Blobs_3 | samples: 500 features: 2 classes: 3 ===

Accuracy: 0.8933

Mapping (pred_cluster -> true_label): {np.int64(2): np.int64(0), np.int64(0): np.int64(1), np.int64(1): np.int64(2)}

Confusion matrix (rows=true labels, cols=pred clusters):

```
[[ 5  6 39]
 [49  1  0]
 [ 1 46  3]]
```

Classification report:

```
              precision    recall  f1-score   support
```

0	0.93	0.78	0.85	50
1	0.89	0.98	0.93	50
2	0.87	0.92	0.89	50
accuracy			0.89	150
macro avg	0.90	0.89	0.89	150
weighted avg	0.90	0.89	0.89	150

=== Dataset: Blobs_4_overlap | samples: 600 features: 2 classes: 4 ===

Accuracy: 0.8167

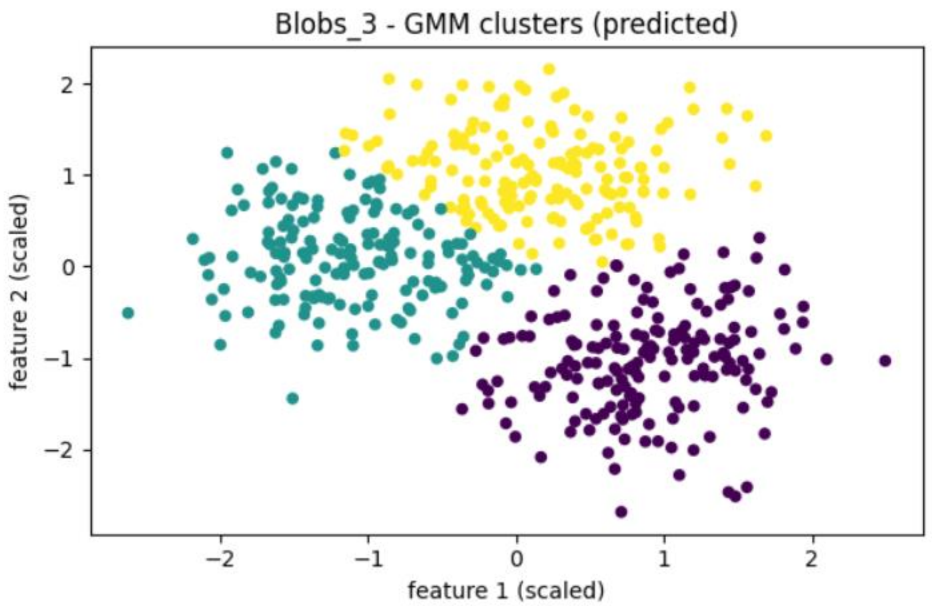
Mapping (pred_cluster -> true_label): {np.int64(2): np.int64(0), np.int64(1): np.int64(1), np.int64(3): np.int64(2), np.int64(0): np.int64(3)}

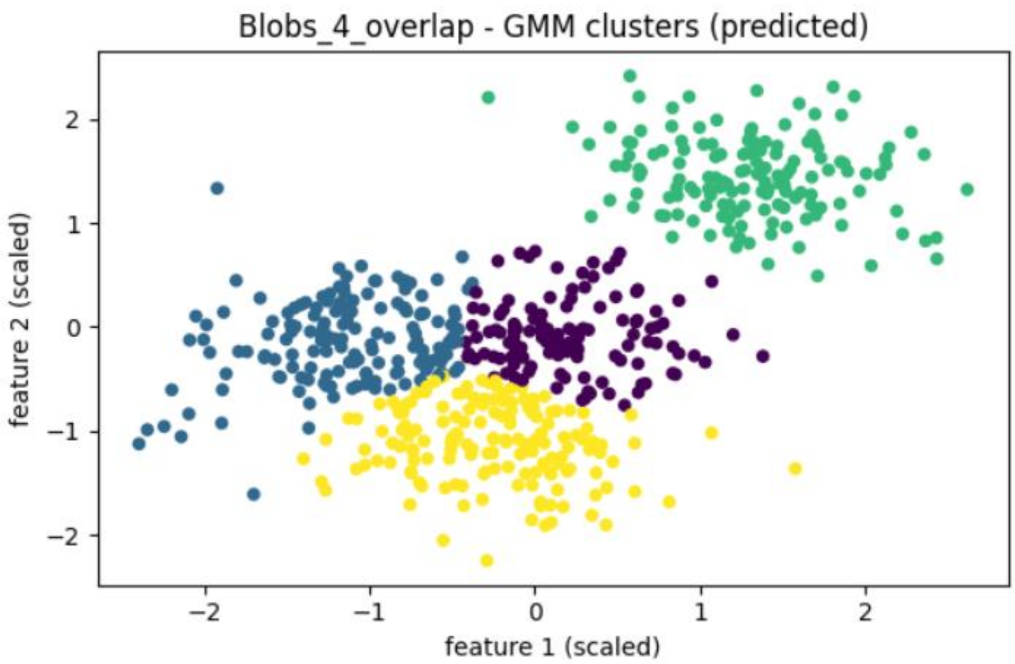
Confusion matrix (rows=true labels, cols=pred clusters):

```
[[ 1  0 44  0]
 [ 6 35  0  4]
 [ 2  3  0 40]
 [28 10  1  6]]
```

Classification report:

	precision	recall	f1-score	support
0	0.98	0.98	0.98	45
1	0.73	0.78	0.75	45
2	0.80	0.89	0.84	45
3	0.76	0.62	0.68	45
accuracy			0.82	180
macro avg	0.82	0.82	0.81	180
weighted avg	0.82	0.82	0.81	180


Blobs_3 - GMM clusters (predicted)
feature 2 (scaled)
feature 1 (scaled)
2
1
0
−1
−2
−2
−1
0
1
2


Blobs_4_overlap - GMM clusters (predicted)
feature 2 (scaled)
feature 1 (scaled)
2
1
0
−1
−2
−2
−1
0
1
2

TASK 7

PARTITIONED CLUSTERING

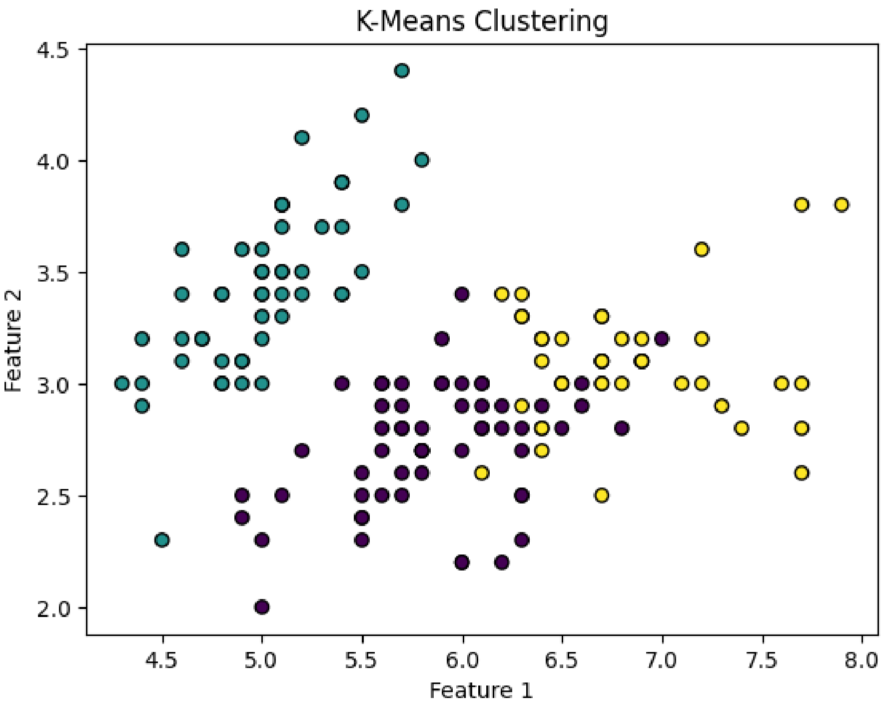
Program :

```
# 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
```

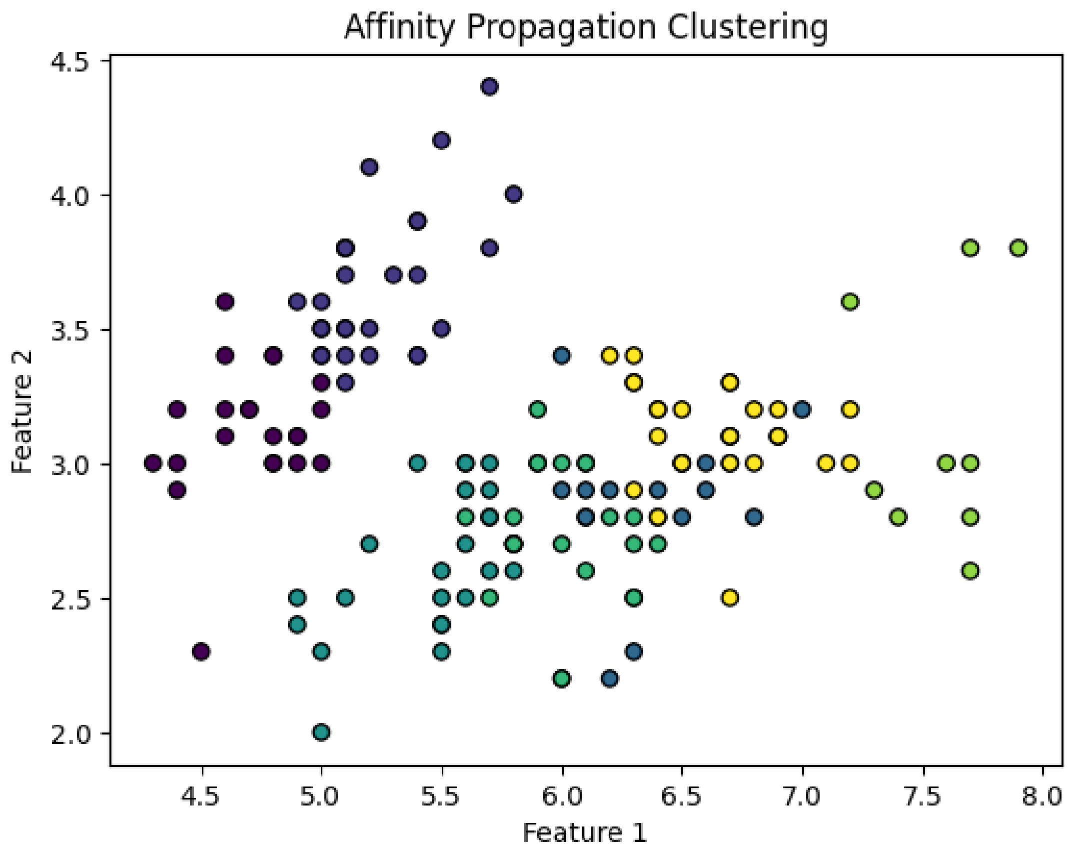
```python
# Affinity Propagation clustering function
def affinity_propagation_clustering(X):
    model = AffinityPropagation()
    labels = model.fit_predict(X)
    return labels

# 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()
```

```python
# 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')
```

```
Evaluation Metrics for K-Means Clustering:
Silhouette Score: 0.5528190123564095
Davies-Bouldin Index: 0.6619715465007465
Calinski-Harabasz Index: 561.62775662962

/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto'
in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```




```
Evaluation Metrics for Affinity Propagation:

Silhouette Score: 0.3474081937055608

Davies-Bouldin Index: 0.9853972233056473

Calinski-Harabasz Index: 443.79711286686637
```

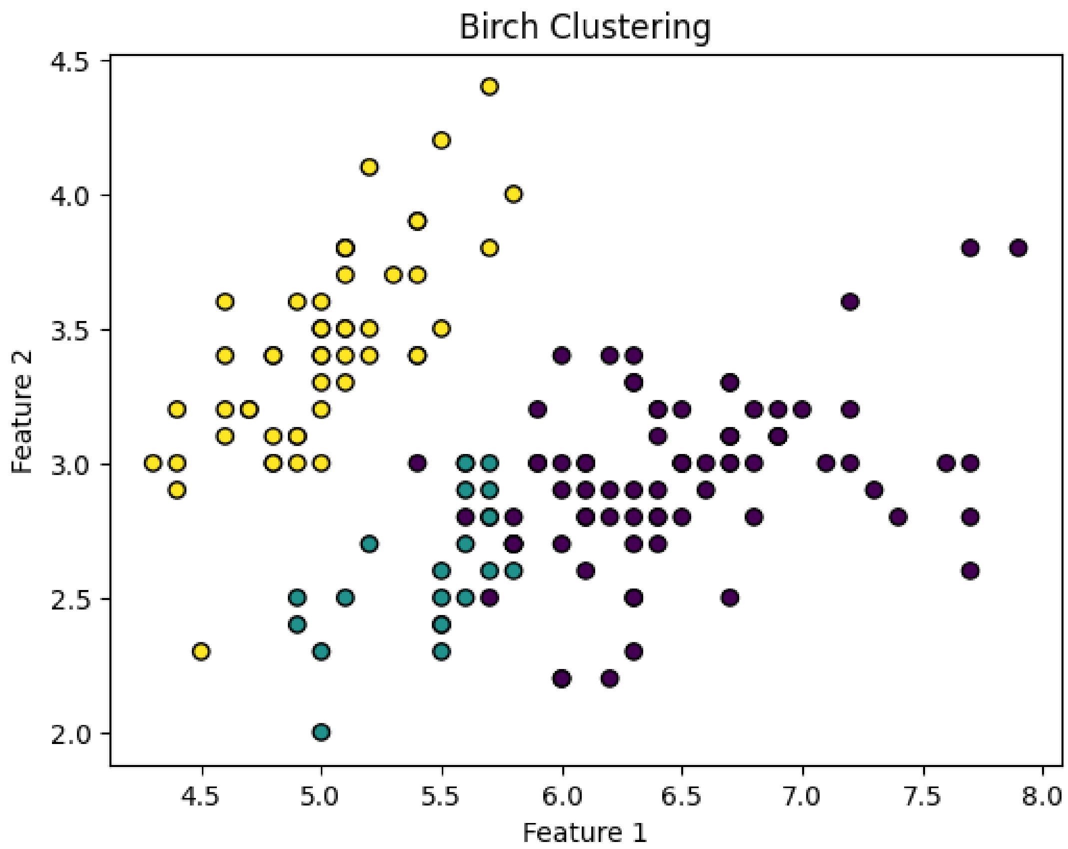


Evaluation Metrics for Birch Clustering:

Silhouette Score: 0.5019524848046077

Davies-Bouldin Index: 0.625830592433168

Calinski-Harabasz Index: 458.47251055625765


Birch 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

HIERARCHICAL CLUSTERING

```
# 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()
```

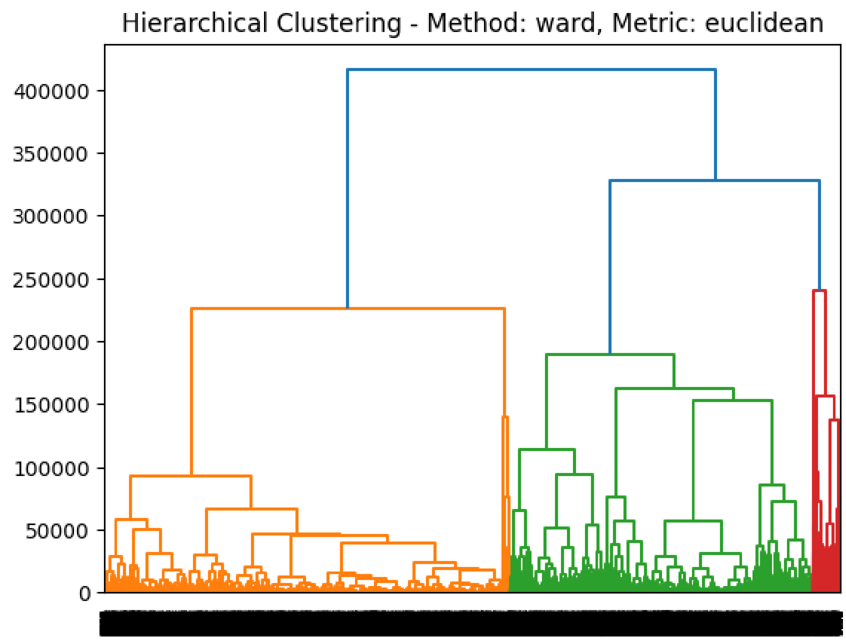
```
# Apply Agglomerative Hierarchical Clustering
hierarchical_labels = hierarchical_clustering(X)
evaluate_clustering(X, hierarchical_labels, 'Agglomerative Hierarchical Clustering')
hierarchical_dendrogram(X)

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

Evaluation Metrics for Agglomerative Hierarchical Clustering:

Silhouette Score: 0.3306184683664866

Davies-Bouldin Index: 1.222874375370533

Calinski-Harabasz Index: 2243.120363690621

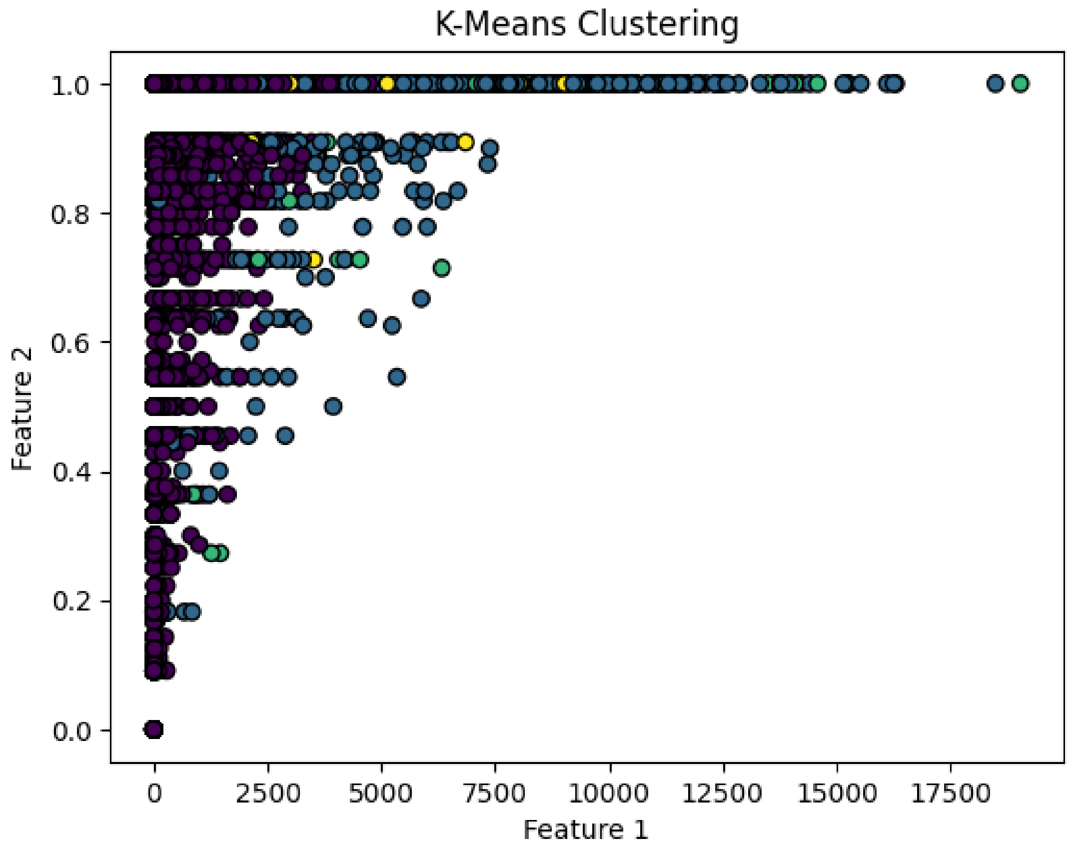


Evaluation Metrics for K-Means Clustering:

Silhouette Score: 0.4665543170797231

Davies-Bouldin Index: 1.1007115307303785

Calinski-Harabasz Index: 2693.6977744231417



TASK 8

BACK PROPAGATION NEURAL NETWORKS

```
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)
```

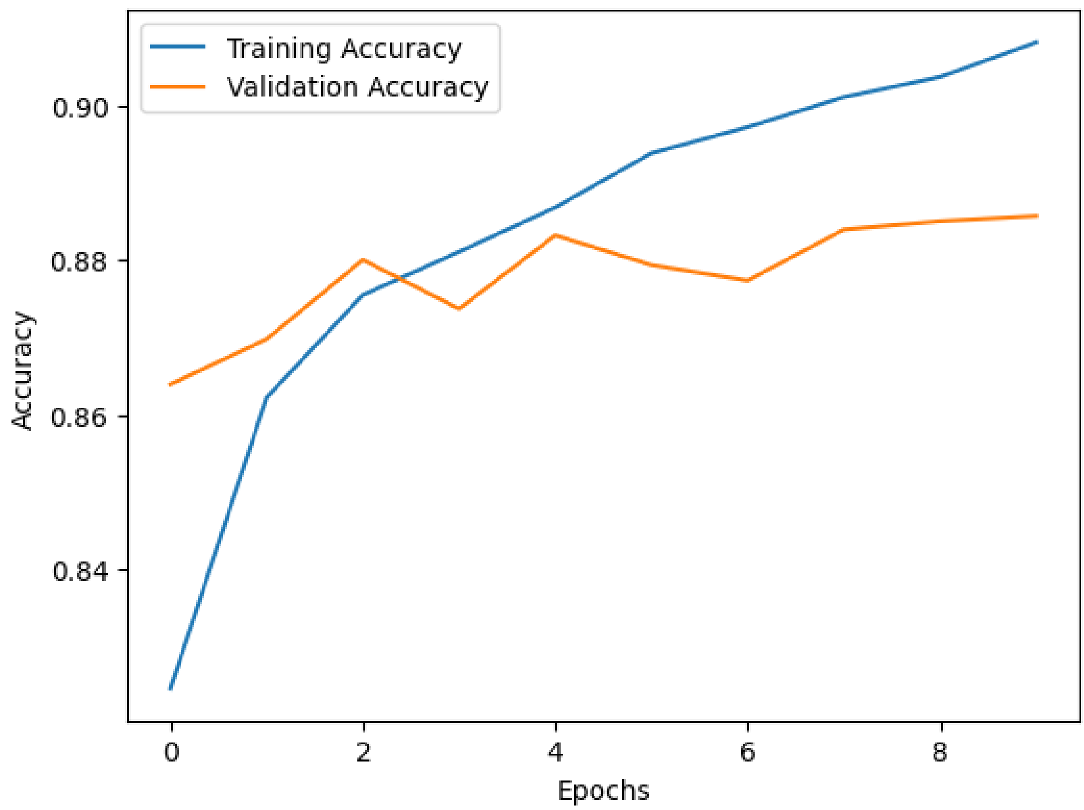
```
# Build the neural network model
model = models.Sequential()
model.add(layers.Dense(128, activation='relu', input_shape=(28 * 28,)))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(10, activation='softmax'))

# 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)
```

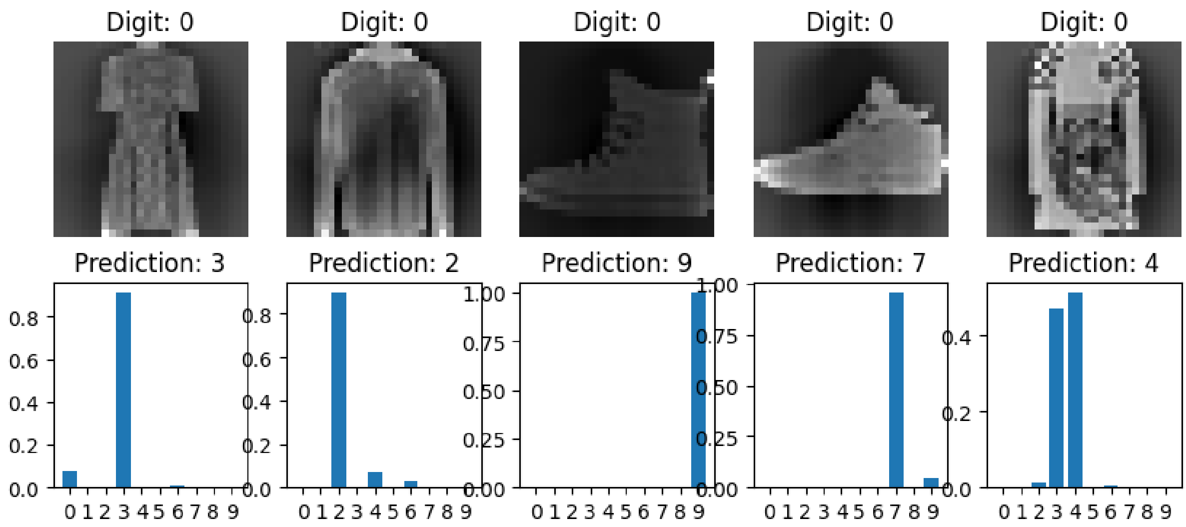
```
        plt.imshow(images[i].reshape(28, 28), cmap='gray')
        plt.title(f"Digit: {np.argmax(labels[i])}")
        plt.axis('off')

        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)
```

Output:

```
Epoch 1/10
1500/1500 [==============================] - 8s 5ms/step - loss: 0.5037
- accuracy: 0.8246 - val_loss: 0.3805 - val_accuracy: 0.8639
Epoch 2/10
1500/1500 [==============================] - 6s 4ms/step - loss: 0.3811
- accuracy: 0.8622 - val_loss: 0.3562 - val_accuracy: 0.8698
Epoch 3/10
1500/1500 [==============================] - 7s 5ms/step - loss: 0.3420
- accuracy: 0.8755 - val_loss: 0.3397 - val_accuracy: 0.8800
Epoch 4/10
```

```
1500/1500 [==============================] - 6s 4ms/step - loss: 0.3203 - accuracy: 0.8810 - val_loss: 0.3510 - val_accuracy: 0.8737

Epoch 5/10

1500/1500 [==============================] - 7s 5ms/step - loss: 0.3050 - accuracy: 0.8868 - val_loss: 0.3312 - val_accuracy: 0.8832

Epoch 6/10

1500/1500 [==============================] - 6s 4ms/step - loss: 0.2877 - accuracy: 0.8938 - val_loss: 0.3533 - val_accuracy: 0.8793

Epoch 7/10

1500/1500 [==============================] - 7s 5ms/step - loss: 0.2757 - accuracy: 0.8972 - val_loss: 0.3518 - val_accuracy: 0.8773

Epoch 8/10

1500/1500 [==============================] - 6s 4ms/step - loss: 0.2655 - accuracy: 0.9010 - val_loss: 0.3478 - val_accuracy: 0.8839

Epoch 9/10

1500/1500 [==============================] - 8s 5ms/step - loss: 0.2596 - accuracy: 0.9036 - val_loss: 0.3402 - val_accuracy: 0.8850

Epoch 10/10

1500/1500 [==============================] - 6s 4ms/step - loss: 0.2457 - accuracy: 0.9081 - val_loss: 0.3493 - val_accuracy: 0.8857

313/313 [==============================] - 1s 2ms/step - loss: 0.3816 - accuracy: 0.8776

Test accuracy: 0.8776000142097473
```


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

TASK 9

NEURAL NETWORKS LEARNING FOR LINEAR AND NON-LINEAR ACTIVATION FUNCTIONS

Program :

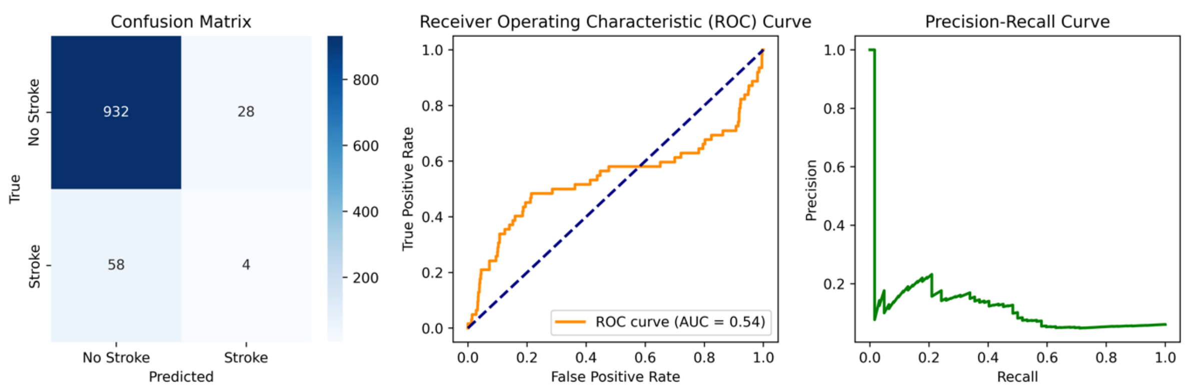
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Perceptron
from sklearn.metrics import accuracy_score, confusion_matrix,
roc_curve, auc, precision_recall_curve
import seaborn 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()
```

```
      id  gender   age  hypertension  heart_disease ever_married  \
0   9046    Male  67.0             0              1          Yes
1  51676  Female  61.0             0              0          Yes
2  31112    Male  80.0             0              1          Yes
3  60182  Female  49.0             0              0          Yes
4   1665  Female  79.0             1              0          Yes


       work_type Residence_type  avg_glucose_level   bmi
smoking_status  \
0        Private          Urban             228.69  36.6  formerly
smoked
```

```
1   Self-employed          Rural             202.21   NaN      never
smoked

2          Private          Rural             105.92   32.5     never
smoked

3          Private          Urban             171.23   34.4
smokes

4   Self-employed          Rural             174.12   24.0     never
smoked


   stroke

0         1

1         1

2         1

3         1

4         1

Accuracy: 0.9158512720156555

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



TASK 10

Program :

```
# Import required libraries

import numpy as np

# Define input features (X) and target output (y)

# Example: OR Gate

X = np.array([[0,0], [0,1], [1,0], [1,1]])

y = np.array([0, 1, 1, 1])

# Initialize weights and learning rate

weights = np.zeros(X.shape[1])

bias = 0

lr = 0.1

# Fixed Increment Learning Algorithm

def activation(x):

   return 1 if x >= 0 else 0

epochs = 10

or epoch in range(epochs):

   error_count = 0

   for i in range(len(X)):

      z = np.dot(X[i], weights) + bias

      output = activation(z)

      error = y[i] - output

      if error != 0:

         weights += lr * error * X[i]

         bias += lr * error

         error_count += 1

   if error_count == 0:

      break

print(" Final Weights:", weights)

print("Final Bias:", bias)
```

OUTPUT :

Final Weights: [0.1 0.1]

Final Bias: -0.1

TASK 11

Program :

```
# Step 1: Install hmmlearn (if not already installed)
!pip install hmmlearn --quiet

# Step 2: Import libraries
import numpy as np
from hmmlearn import hmm

# Step 3: Define hidden states (diseases)
states = ["Flu", "Cold"]
n_states = len(states)

# Step 4: Define observable symptoms
observations = ["Fever", "Cough", "Normal"]
n_observations = len(observations)

# Step 5: Create the HMM model
model = hmm.MultinomialHMM(n_components=n_states, n_iter=100, tol=0.01)

# Step 6: Set initial probabilities (probability of starting with each disease)
model.startprob_ = np.array([0.6, 0.4])  # Flu more likely initially

# Step 7: Set transition probabilities (disease progression)
model.transmat_ = np.array([
    [0.7, 0.3],  # Flu to Flu/Cold
    [0.4, 0.6]   # Cold to Flu/Cold
])

# Step 8: Set emission probabilities (symptoms given disease)
```

```
model.emissionprob_ = np.array([
    [0.6, 0.3, 0.1],  # Flu: Fever, Cough, Normal
    [0.2, 0.5, 0.3]   # Cold: Fever, Cough, Normal
])

# Step 9: Map observations to integers
obs_map = {"Fever": 0, "Cough": 1, "Normal": 2}

# Example observed symptoms sequence
obs_sequence = ["Fever", "Cough", "Normal", "Fever", "Cough"]
obs_seq_int = np.array([[obs_map[sym]] for sym in obs_sequence])

# Step 10: Predict the hidden states (diseases)
logprob, hidden_states = model.decode(obs_seq_int, algorithm="viterbi")

# Step 11: Print results
print("Observed Symptoms: ", obs_sequence)
print("Predicted Disease States: ", [states[s] for s in hidden_states])
```

OUTPUT:

Observed Symptoms: ['Fever', 'Cough', 'Normal', 'Fever', 'Cough']

Predicted Disease States: ['Flu', 'Flu', 'Cold', 'Flu', 'Flu']

TASK 12

Program :

```
# Install required library
!pip install scikit-learn

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Sample text data
docs = ["I love medicine and healthcare",
    "Doctors treat patients",
    "Football is a great sport",
    "Sports improve health",
    "Hospitals save lives",
    "Players train hard"]

# Labels: 1 = Medical, 0 = Sports
labels = [1, 1, 0, 0, 1, 0]

# Convert text to feature vectors
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(docs)

# Train SVM (RBF kernel)
clf = SVC(kernel='rbf', gamma='scale')
clf.fit(X, labels)

# Predict on same data
pred = clf.predict(X)

# Calculate metrics
```

accuracy = accuracy_score(labels, pred)

precision = precision_score(labels, pred)

recall = recall_score(labels, pred)

classification_rate = accuracy * 100

print("Accuracy:", accuracy)

print("Precision:", precision)

print("Recall:", recall)

print("Classification Rate:", classification_rate, "%")

OUTPUT:

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.12/dist-packages (1.6.1)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (2.0.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.16.2)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.5.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (3.6.0)
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
Classification Rate: 100.0 %
```

TASK 13

Program :

```
# Install sklearn (if not already)
!pip install scikit-learn

from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score

# Step 1: Sample data (AND gate)
X = [[0,0], [0,1], [1,0], [1,1]]
y = [0, 0, 0, 1]   # Output of AND gate

# Step 2: Define ANN model
model = MLPClassifier(hidden_layer_sizes=(3,), activation='relu', solver='adam', max_iter=1000)

# Step 3: Train the model
model.fit(X, y)

# Step 4: Predict
pred = model.predict(X)

# Step 5: Evaluate
acc = accuracy_score(y, pred)
print("Input:", X)
print("Predicted Output:", pred)
print("Accuracy:", acc)
```

OUTPUT:

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.12/dist-packages (1.6.1)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (2.0.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.16.2)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.5.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (3.6.0)
Input: [[0, 0], [0, 1], [1, 0], [1, 1]]
Predicted Output: [0 0 0 1]
Accuracy: 1.0
/usr/local/lib/python3.12/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
  warnings.warn(
```