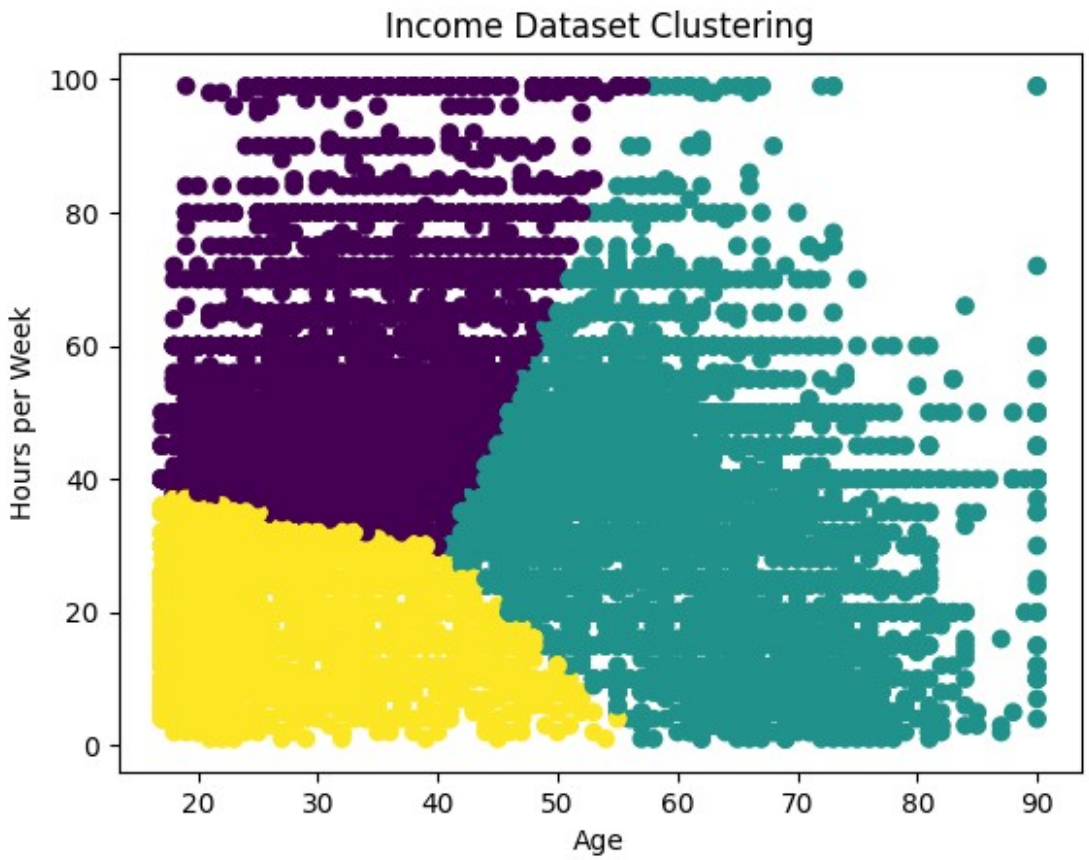
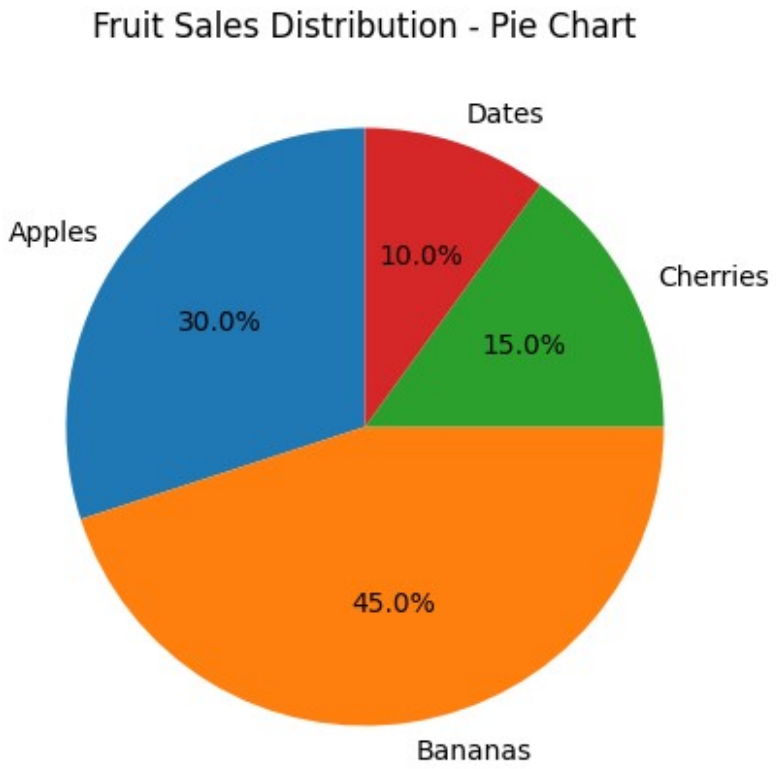
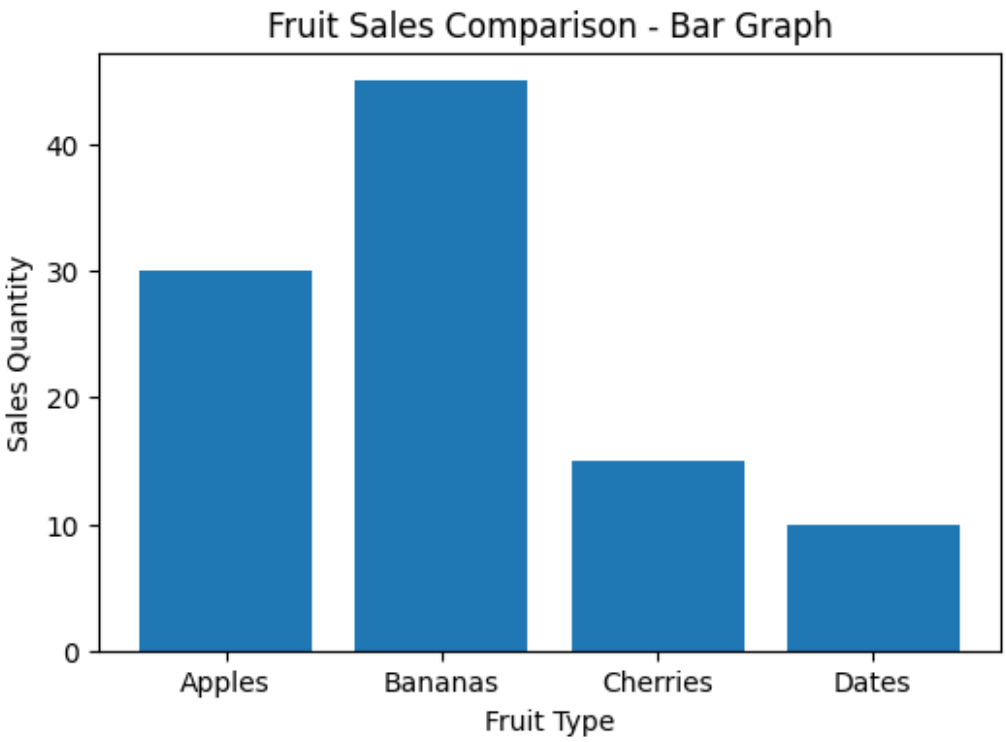
```python
# □ Practical 1: Implement clustering algorithm on given “income.csv” dataset and display it using scatter plot
from google.colab import files
uploaded = files.upload()

import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Load dataset
data = pd.read_csv("income.csv")

# Check column names
print("Columns:", data.columns)

# Perform clustering on age and hours_per_week (you can change columns if needed)
kmeans = KMeans(n_clusters=3)
data['cluster'] = kmeans.fit_predict(data[['age', 'hours_per_week']])

# Display scatter plot
plt.scatter(data['age'], data['hours_per_week'], c=data['cluster'])
plt.xlabel('Age')
plt.ylabel('Hours per Week')
plt.title('Income Dataset Clustering')
plt.show()
```

```
<IPython.core.display.HTML object>

Saving income.csv to income.csv
Columns: Index(['age', 'fnlwgt', 'education_num', 'capital_gain', 'capital_loss',
       'hours_per_week', 'income_level'],
      dtype='object')
```

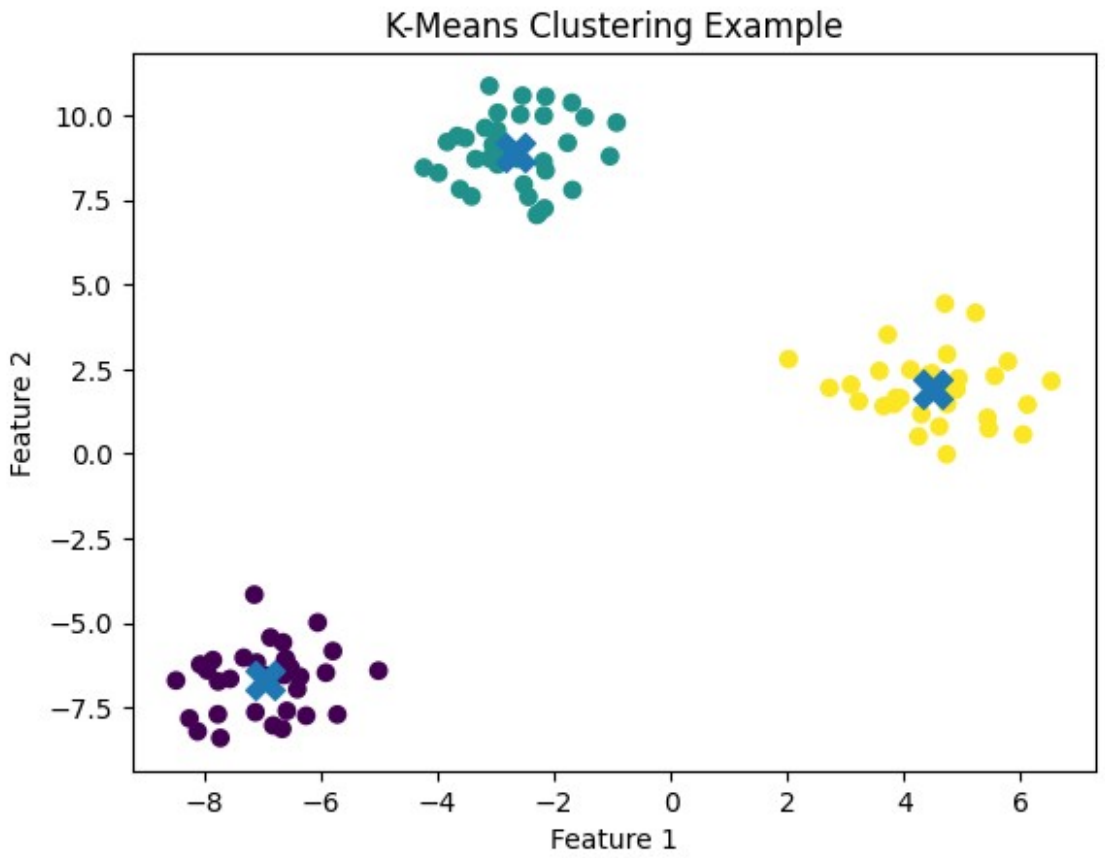



```python
# 🧪 Practical 2: Implement Data Visualization using your own data
# A. Draw a Pie Graph
# B. Draw a Bar Graph

import matplotlib.pyplot as plt

# Sample data (you can replace with your own)
categories = ['Apples', 'Bananas', 'Cherries', 'Dates']
values = [30, 45, 15, 10]

# --- A. Pie Chart ---
plt.figure(figsize=(5,5))
plt.pie(values, labels=categories, autopct='%1.1f%%', startangle=90)
plt.title("Fruit Sales Distribution - Pie Chart")
plt.show()

# --- B. Bar Graph ---
plt.figure(figsize=(6,4))
plt.bar(categories, values)
plt.xlabel('Fruit Type')
plt.ylabel('Sales Quantity')
plt.title("Fruit Sales Comparison - Bar Graph")
plt.show()
```

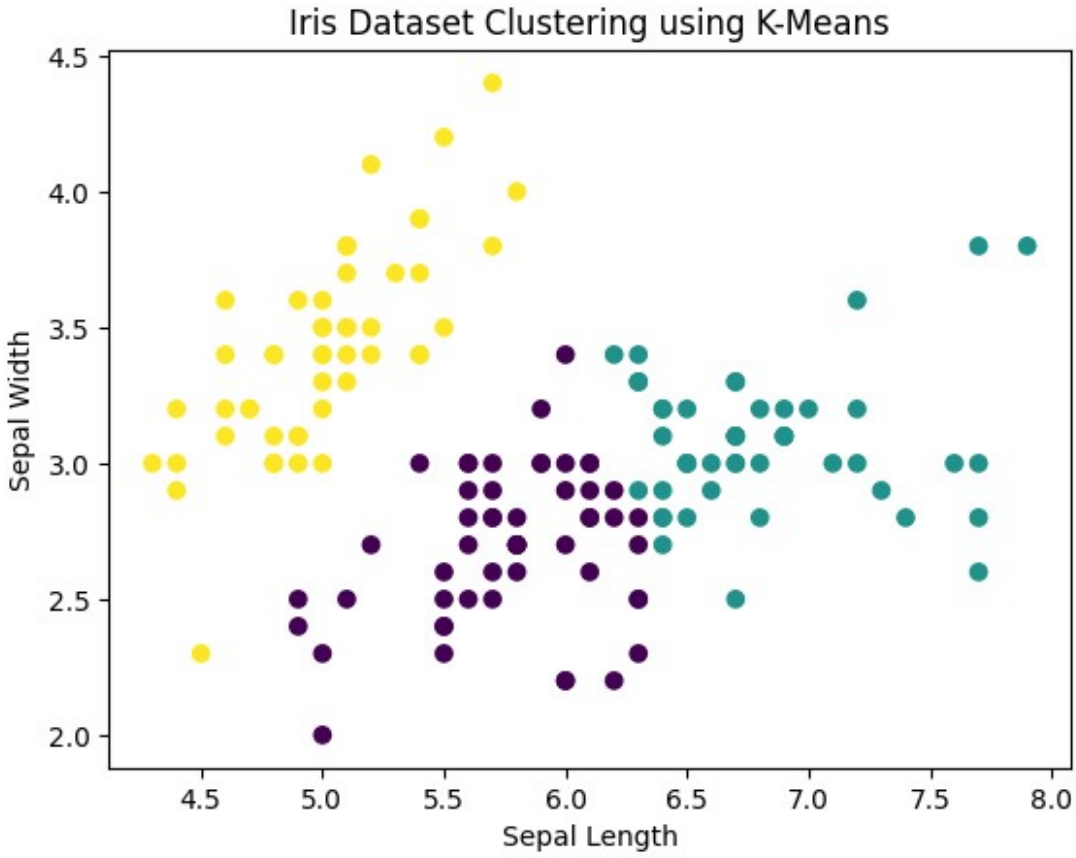

Fruit Sales Distribution - Pie Chart
Dates
Apples
Cherries
10.0%
30.0%
15.0%
45.0%
Bananas


Fruit Sales Comparison - Bar Graph
Sales Quantity
0
10
20
30
40
Apples
Bananas
Cherries
Dates
Fruit Type

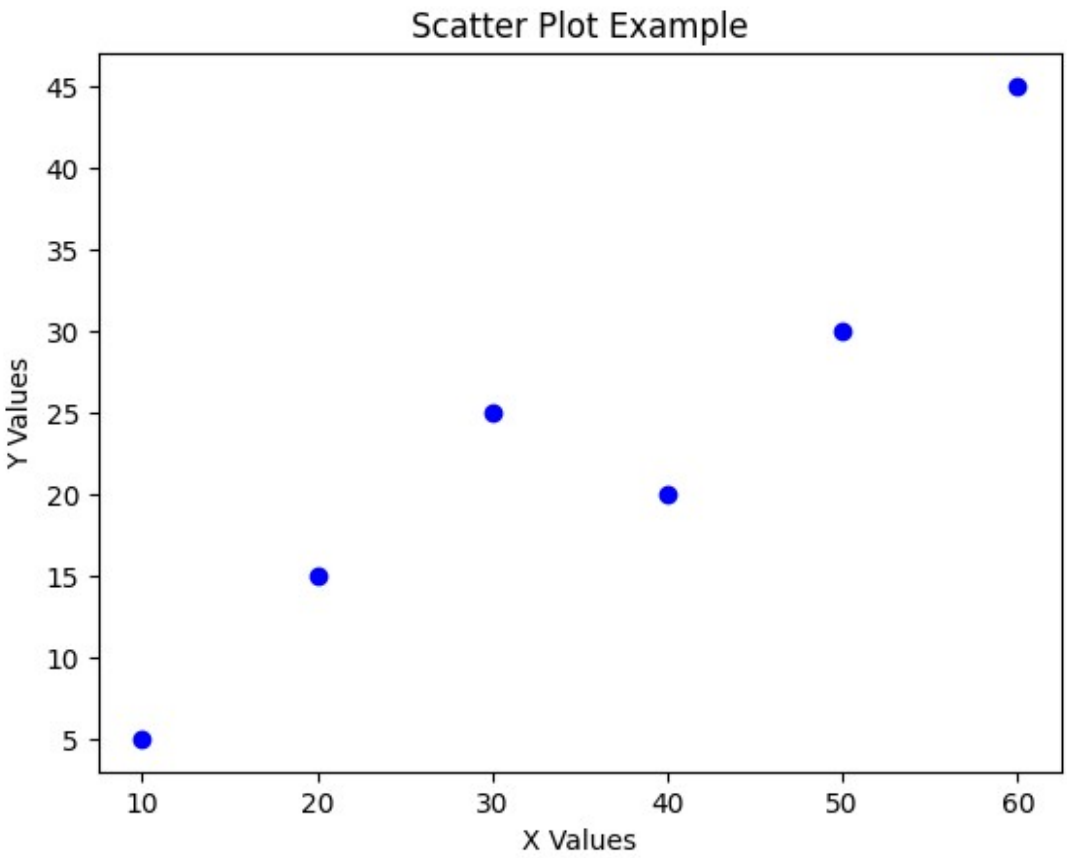
```python
# ✅ Practical 3: Implement K-Means Clustering techniques using any data analytics tool

import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

# Create sample data
X, y = make_blobs(n_samples=100, centers=3, random_state=42)

# Apply K-Means
kmeans = KMeans(n_clusters=3)
y_pred = kmeans.fit_predict(X)

# Plot the clusters
plt.scatter(X[:,0], X[:,1], c=y_pred)
plt.scatter(kmeans.cluster_centers_[:,0],
kmeans.cluster_centers_[:,1], s=200, marker='X')
plt.title("K-Means Clustering Example")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```

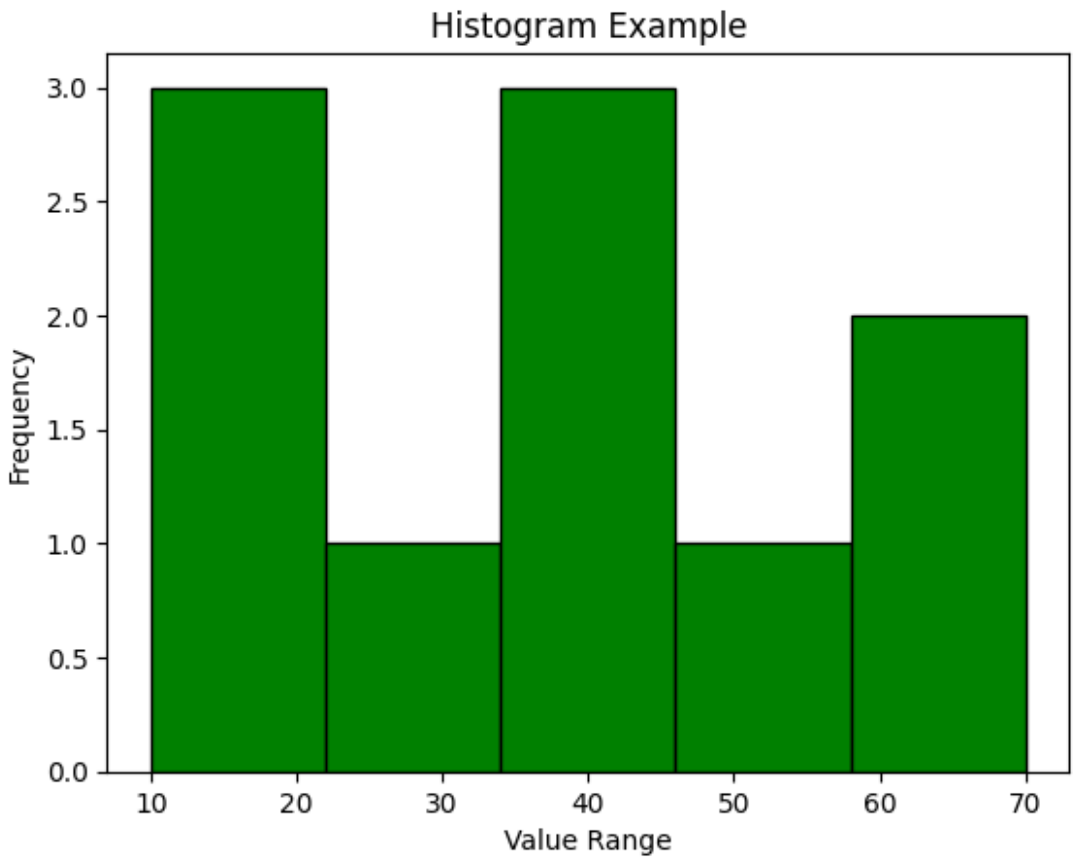


```python
# 🔹 Practical 4: Implement clustering algorithm and display it using scatter plot on Iris dataset

from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Load iris dataset
iris = load_iris()
X = iris.data[:, :2]  # using first two columns for 2D plotting

# Apply K-Means clustering
kmeans = KMeans(n_clusters=3)
y_pred = kmeans.fit_predict(X)

# Display scatter plot
plt.scatter(X[:,0], X[:,1], c=y_pred)
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.title('Iris Dataset Clustering using K-Means')
plt.show()
```

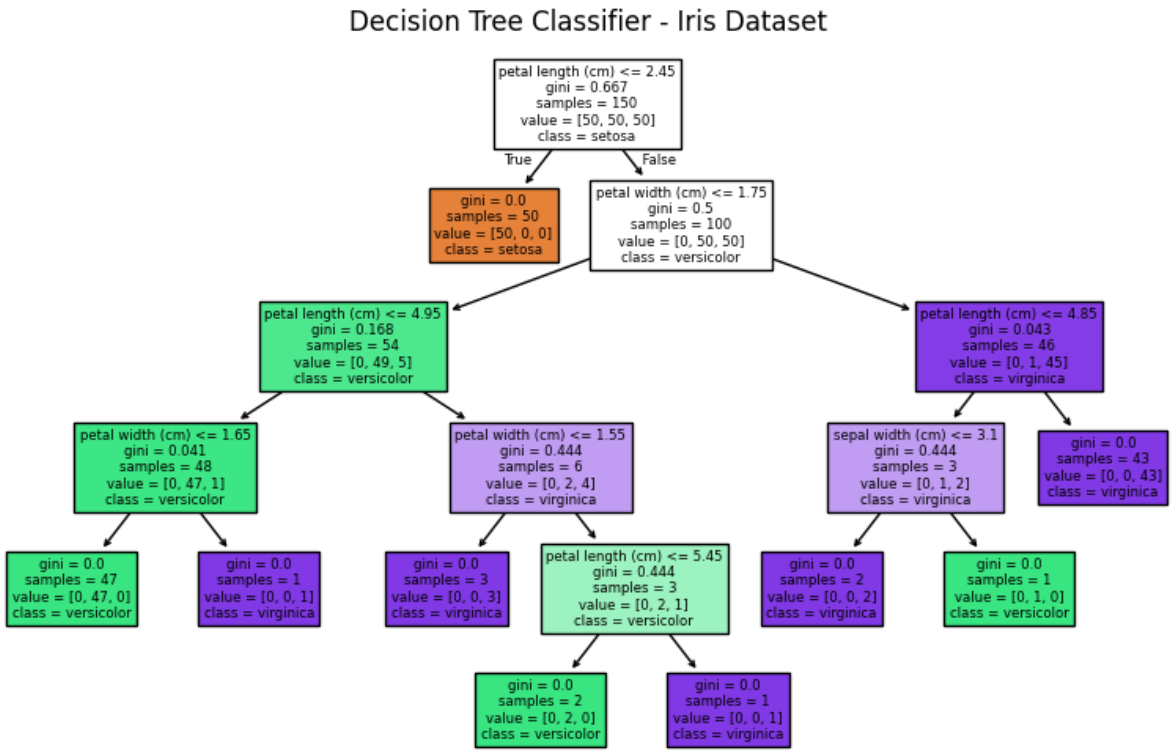


```python
# □ Practical 5: Implement Data Visualization using your own data
# A. Scatter Plot
# B. Histogram

import matplotlib.pyplot as plt

# Sample data (you can replace with your own)
x = [10, 20, 30, 40, 50, 60]
y = [5, 15, 25, 20, 30, 45]

# --- A. Scatter Plot ---
plt.scatter(x, y, color='blue')
plt.title("Scatter Plot Example")
plt.xlabel("X Values")
plt.ylabel("Y Values")
plt.show()

# --- B. Histogram ---
data = [10, 20, 20, 30, 40, 40, 40, 50, 60, 70]
plt.hist(data, bins=5, color='green', edgecolor='black')
plt.title("Histogram Example")
plt.xlabel("Value Range")
plt.ylabel("Frequency")
plt.show()
```


Scatter Plot Example
Y Values
X Values
45
40
35
30
25
20
15
10
5
10
20
30
40
50
60




```python
# 🔹 Practical 6: Implement Decision Tree Classifier on Iris Dataset

from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt

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

# Create and train model
model = DecisionTreeClassifier()
model.fit(X, y)

# Plot decision tree
plt.figure(figsize=(10,6))
plot_tree(model, filled=True, feature_names=iris.feature_names,
class_names=iris.target_names)
plt.title("Decision Tree Classifier - Iris Dataset")
plt.show()
```




```
# 🔹 Practical 7: Implement KNN Classification techniques using any data analytics tool

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

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

# Split 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)

# Create and train KNN model
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# Predict and check accuracy
y_pred = knn.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))

Accuracy: 1.0
```

```python
import warnings
warnings.filterwarnings('ignore')

import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules

data = pd.DataFrame({
    'milk': [1,1,0,1],
    'bread': [1,0,1,1],
    'butter': [0,1,1,1]
})
data = data.astype(bool)

freq = apriori(data, min_support=0.5, use_colnames=True)
rules = association_rules(freq, metric="support", min_threshold=0.5)
print(rules[['antecedents','consequents','support']])
```

```
  antecedents consequents  support
0      (milk)     (bread)      0.5
1     (bread)      (milk)      0.5
2      (milk)    (butter)      0.5
3    (butter)      (milk)      0.5
4    (butter)     (bread)      0.5
5     (bread)    (butter)      0.5
```

```python
# □ Practical 9: Implement Apriori Association Rule Mining techniques
using any data analytics tool

import warnings
warnings.filterwarnings('ignore')   # To hide unnecessary warnings

import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules

# Sample dataset (1 = item bought, 0 = not bought)
data = pd.DataFrame({
    'milk':   [1, 1, 0, 1],
    'bread':  [1, 0, 1, 1],
    'butter': [0, 1, 1, 1],
    'jam':    [1, 0, 1, 0]
})

# Convert data to boolean
data = data.astype(bool)

# Apply Apriori algorithm to find frequent itemsets
frequent_items = apriori(data, min_support=0.5, use_colnames=True)
print("Frequent Itemsets:\n", frequent_items)

# Generate association rules based on Lift metric
rules = association_rules(frequent_items, metric="lift",
```

```
min_threshold=1)
print("\nAssociation Rules:\n", rules[['antecedents', 'consequents',
'support', 'confidence', 'lift']])
```

```
Frequent Itemsets:
    support         itemsets
0     0.75           (milk)
1     0.75          (bread)
2     0.75         (butter)
3     0.50            (jam)
4     0.50    (milk, bread)
5     0.50   (milk, butter)
6     0.50  (butter, bread)
7     0.50     (jam, bread)

Association Rules:
   antecedents consequents  support  confidence      lift
0       (jam)     (bread)      0.5    1.000000  1.333333
1     (bread)       (jam)      0.5    0.666667  1.333333
```

```
# 🔹 Practical 10: Implement Naive Bayes Classification techniques
using any data analytics tool

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score,confusion_matrix,
ConfusionMatrixDisplay

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

# Split 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)

# Create and train Naive Bayes model
model = GaussianNB()
model.fit(X_train, y_train)

# Predict on test data
y_pred = model.predict(X_test)

# Display accuracy
print("Accuracy:", accuracy_score(y_test, y_pred))

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
```

```
display_labels=iris.target_names)
disp.plot(cmap='Blues')
plt.title("Confusion Matrix - Naive Bayes Classifier (Iris Dataset)")
plt.show()
```

```
Accuracy: 1.0
```