EX 1: Student Information using Lists and Tuples

Aim:

To create Python lists or tuples to store the student details (rollno, name, dept, branch, percentage of mark) and print the values.

PROCEDURE:

1. Start the Python program.
2. Create a list or tuple of student records where each student detail contains (Roll Number, Name, Department, Branch, Percentage).
3. Display student details.
4. Stop the program.

Program:

```
# Creating list of students - (rollno, name, dept, branch, percentage)

students = [
                [101, "Ranji", "CSE", "Data Science", 85.6],
                [102, "Ravi", "CSE", "Machine Learning", 91.2],
                [103, "Rohu", "IT", "AI", 78.5],
                [104, "Kousi", "ECE", "Embedded", 88.4],
                [105, "Azhaghu", "EEE", "Robotics", 82.9]
            ]
```

```
# Printing student details

print("Student Details:\n")
for s in students:
    print("Roll No   :", s[0])
    print("Name      :", s[1])
    print("Dept      :", s[2])
    print("Branch    :", s[3])
    print("Percent   :", s[4])
    print("---------------------")
```

```python
# Creating tuple of students - (rollno, name, dept, branch, percentage)

students = (
                (101, "Ranji", "CSE", "Data Science", 85.6),
                (102, "Ravi", "CSE", "Machine Learning", 91.2),
                (103, "Rohu", "IT", "AI", 78.5),
                (104, "Kousi", "ECE", "Embedded", 88.4),
                (105, "Azhaghu", "EEE", "Robotics", 82.9)
            )
```

```python
# Printing student details

print("Student Details:\n")
for s in students:
    print("Roll No   :", s[0])
    print("Name      :", s[1])
    print("Dept      :", s[2])
    print("Branch    :", s[3])
    print("Percent   :", s[4])
    print("---------------------")
```

```
Student Details:

Roll No    : 101
Name       : Ranji
Dept       : CSE
Branch     : Data Science
Percent    : 85.6
---------------------
Roll No    : 102
Name       : Ravi
Dept       : CSE
Branch     : Machine Learning
Percent    : 91.2
---------------------
Roll No    : 103
Name       : Rohu
Dept       : IT
Branch     : AI
Percent    : 78.5
---------------------
Roll No    : 104
Name       : Kousi
Dept       : ECE
Branch     : Embedded
Percent    : 88.4
```

Result:

Thus, the Python program using lists and tuples to store and display the student details was executed successfully.

EX 2: NumPy Array Creation and Slicing using Lists and Tuples

Aim:

To create Python list and tuple, convert them into NumPy arrays, print its elements and slice the NumPy array into three parts.

PROCEDURE:

1. Start the Python program.
2. Create a Python list and tuple.
3. Convert the list and tuple into NumPy array and print the values.
4. Slice the NumPy array into three portions and print each slice.
5. Stop the program.

Program:

Creating List:

```
import numpy as np

students = [101, 102, 103, 104, 105, 106]
print("Original Python List:")
print(students)

# Convert list into NumPy array

arr1 = np.array(students)
print("\nNumPy Array values:")
print(arr1)

# Slice NumPy array into 3 equal parts

print("\nArray Slice 1:")
print(arr1[:2])

print("\nArray Slice 2:")
print(arr1[2:4])

print("\nArray Slice 3:")
print(arr1[4:])
```

Output:

```
Original Python List:
[101, 102, 103, 104, 105, 106]

NumPy Array values:
[101 102 103 104 105 106]

Array Slice 1:
[101 102]

Array Slice 2:
[103 104]

Array Slice 3:
[105 106]
```

Creating Tuple:

```python
students = (101, 102, 103, 104, 105, 106)
print("Original Python Tuple:")
print(students)

# Convert tuple into NumPy array

arr1 = np.array(students)
print("\nNumPy Array values:")
print(arr1)

# Slice NumPy array into 3 equal parts

print("\nArray Slice 1:")
print(arr1[:2])

print("\nArray Slice 2:")
print(arr1[2:4])

print("\nArray Slice 3:")
print(arr1[4:])
```

Output:

```
Original Python Tuple:
(101, 102, 103, 104, 105, 106)

NumPy Array values:
[101 102 103 104 105 106]

Array Slice 1:
[101 102]

Array Slice 2:
[103 104]

Array Slice 3:
[105 106]
```

Result:

Thus, the Python program to convert list and tuple into NumPy array and slice the array into three parts was executed successfully.

EX 3: NumPy Array Inspection

Aim:

To load class marklist data from a CSV file into a NumPy array and perform array inspection operations such as len(), ndim, size, dtype, shape and info().

PROCEDURE:

1. Start the Python program.
2. Read the CSV file **marklist.csv** containing student records using NumPy loadtxt() function.
3. Display the loaded NumPy array.
4. Perform array inspection operations:

len()

1. gives the total number of rows in a NumPy array.
2. Syntax: **len(array_name)**
3. If the csv file contains 20 student records it returns 20

ndim()

1. ndim means Number of Dimensions
2. represents how many dimensions the array has.
   - A 1D array → ndim = 1
   - A 2D table → ndim = 2
   - A 3D cube → ndim = 3
3. Syntax: **array_name.ndim**
4. Marklist.csv has rows and columns so ndim() gives 2

shape()

1. Shape means array structure in terms of rows and columns.
2. It gives how many rows and columns the array has.
3. Format: (rows, columns)
4. Syntax: **arrayname.shape**
5. If the csv file contains 20 student records and 3 fields per entry it gives (20,3)

size()

1. gives the total number of elements in a NumPy array.
2. Syntax: **arrayname.size**
3. Formula: **Size = rows x columns**
4. If the csv file contains 20 rows and 3 columns then it gives 60

dtype()

1. gives the data type stored inside the array (int, float, string, boolean etc.).
2. Syntax: **arrayname.dtype**
3. Data Type (dtype): <U9 means Unicode string data, with each element having a maximum length of 9 characters.

info()

1. It is a NumPy utility function that displays information about array structure, datatype, shape, memory information and description.

2. Syntax: **np.info(array_name)**

3. np.info(array_name) gives

   class: ndarray - data we loaded is stored as a NumPy array.

   strides - how many bytes we need to move in memory to go to the next row or next element.

   Itemsize - Each element in the array occupies how many bytes of memory.

   aligned: True - means the data in memory is stored in a proper aligned manner.

   contiguous: True - Array values are stored continuously in memory without gaps.

   fortran: False - tells the memory layout is row-major order, not column-major order.

   data pointer: 0x27385518a70 - memory address where the array is stored in the computer.

   byteorder - Data is stored using little-endian byte order or big endian byte order

   byteswap: False - NumPy did not change the original data format stored in memory. It was not modified internally.

5. Stop the program.

Program:

```
# Load data from CSV file into numpy array

import numpy as np
data = np.loadtxt("marklist.csv", delimiter=",", dtype=str, skiprows=1)
print("Marklist Array:\n")
print(data)
```

```
Marklist Array:

[['101' 'Arun' '85']
 ['102' 'Bharath' '78']
 ['103' 'Divya' '92']
 ['104' 'Kavya' '88']
 ['105' 'Vijay' '76']
 ['106' 'Rohit' '90']
 ['107' 'Ramya' '81']
 ['108' 'Sanjay' '73']
 ['109' 'Priya' '94']
 ['110' 'Karan' '69']
 ['111' 'Sneha' '87']
 ['112' 'Varun' '82']
 ['113' 'Aishwarya' '77']
 ['114' 'Manoj' '80']
 ['115' 'Pooja' '95']
```

```
# NumPy Array Inspection operations

print("\nLength of array:", len(data))
print("Number of Dimensions:", data.ndim)
print("Shape of Array:", data.shape)
print("Total Size:", data.size)
print("Data Type:", data.dtype)
```

```
Length of array: 20
Number of Dimensions: 2
Shape of Array: (20, 3)
Total Size: 60
Data Type: <U9
```

```
np.info(data)
```

```
class:  ndarray
shape:  (20, 3)
strides:  (108, 36)
itemsize:  36
aligned:  True
contiguous:  True
fortran:  False
data pointer: 0x27385518a70
byteorder:  little
byteswap:  False
type: <U9
```

Result:

Thus, the class marklist CSV data was successfully loaded into a NumPy array and the array properties such as length, dimensions, size, datatype, and shape were inspected and displayed.

EX 4: Statistical Operations using Pandas

Aim:

To load a dataset into a Pandas DataFrame and perform the following statistical functions: min(), max(), cumsum(), mean(), median(), std(), corrcoef().

PROCEDURE:

1. Import Pandas and NumPy libraries
2. Load the CSV file into a DataFrame
3. Display the DataFrame
4. Apply statistical functions on the DataFrame

Statistical Functions:

Assume the dataset values are:

$$x_1, x_2, x_3, \dots, x_n$$

1. min() – Minimum

$$\min(x) = \text{smallest value among } x_1, x_2, \dots, x_n$$

2. max() – Maximum

$$\max(x) = \text{largest value among } x_1, x_2, \dots, x_n$$

3. cumsum() – Cumulative Sum

$$\textbf{cumsum}_{\mathbf{k}} = \sum_{\mathbf{i=1}}^{\mathbf{k}} \mathbf{x_i}$$

It adds values step by step

4. mean() – Average

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

5. median()

1.Values must be arranged in ascending order.

2.For odd number of values:

$$\textbf{Median} = x_{\frac{n+1}{2}}$$

3.For even number of values:

$$\textbf{Median} = \frac{x_{\frac{n}{2}} + x_{\frac{n}{2}+1}}{2}$$

6. std() – Standard Deviation

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}$$

It measures spread of data from the mean

7. corrcoef() – Correlation Coefficient

$$r = \frac{\sum(x - \bar{x})(y - \bar{y})}{\sqrt{\sum(x - \bar{x})^2 \sum(y - \bar{y})^2}}$$

It measures relationship between two variables.

- $r = +1$ → Perfect positive correlation
- $r = 0$ → No correlation
- $r = -1$ → Perfect negative correlation

Program:

```
import pandas as pd
import numpy as np

df = pd.read_csv("marks.csv")
print(df)
```

```
   Maths  Physics  Chemistry
0     78       85         80
1     88       79         90
2     92       91         89
3     85       87         84
4     90       88         92
```

```
# Statistical functions

print("\nMinimum values:\n", df.min())

print("\nMaximum values:\n", df.max())

print("\nCumulative Sum:\n", df.cumsum())

print("\nMean:\n", df.mean())

print("\nMedian:\n", df.median())

print("\nStandard Deviation:\n", df.std())

print("\nCorrelation Coefficient:\n", np.corrcoef(df.T))
```

Output:

```
Minimum values:
 Maths        78
Physics      79
Chemistry    80
dtype: int64

Maximum values:
 Maths        92
Physics      91
Chemistry    92
dtype: int64

Cumulative Sum:
    Maths  Physics  Chemistry
0      78       85         80
1     166      164        170
2     258      255        259
3     343      342        343
4     433      430        435
```

```
Mean:
 Maths        86.6
Physics      86.0
Chemistry    87.0
dtype: float64

Median:
 Maths        88.0
Physics      87.0
Chemistry    89.0
dtype: float64

Standard Deviation:
 Maths        5.458938
Physics      4.472136
Chemistry    4.898979
dtype: float64

Correlation Coefficient:
 [[1.         0.31745237 0.90677177]
 [0.31745237 1.         0.03423266]
 [0.90677177 0.03423266 1.        ]]
```

Result:

Thus, the given dataset was successfully loaded into a Pandas DataFrame and all required statistical functions were executed successfully.

EX 5: Data Cleaning with Pandas

Aim:

To load a data into a pandas data frame, list out number of missing values in each column and fill the missing values with suitable default value.

PROCEDURE:

1. **Load the CSV file:**

[1]:
```
import pandas as pd
df = pd.read_csv('students_data.csv')
df
```

[1]:

	ID	Name	Age	Salary	Department
0	1	Alice	25.0	50000.0	HR
1	2	Bob	30.0	NaN	IT
2	3	Henry	27.0	62000.0	Finance
3	4	David	28.0	55000.0	IT
4	5	Eve	26.0	52000.0	NaN
5	6	John	27.0	54000.0	Operations
6	1	Alice	25.0	50000.0	HR
7	8	Grace	29.0	58000.0	Marketing
8	3	Henry	27.0	62000.0	Finance

2. Finding the null(missing) values:

df.isnull() -> To check where null exists and returns true in each cell in a dataframe

df.isnull().sum() -> Total number of null values in each column

df.isnull().sum().sum() -> Total number of null values in entire DataFrame

[4]:
```
#To check where null exists and returns true in each cell in a dataframe
df.isnull()
```

[4]:

	ID	Name	Age	Salary	Department
0	False	False	False	False	False
1	False	False	False	True	False
2	False	False	False	False	False
3	False	False	False	False	False
4	False	False	False	False	True
5	False	False	False	False	False
6	False	False	False	False	False
7	False	False	False	False	False
8	False	False	False	False	False

[5]:
```
#Total number of null values in each column
df.isnull().sum()
```

[5]:
```
ID            0
Name          0
Age           0
Salary        1
Department    1
dtype: int64
```

[6]:
```
#Total number of null values in entire DataFrame
df.isnull().sum().sum()
```

[6]:
```
2
```

3. Handle Missing values:

df.fillna(value)

value can be:

- a constant (e.g., 0, 'Unknown')
- a calculation (e.g., column mean or median)
- a method (e.g., 'ffill', 'bfill') ffill -forward fill, bfill-backward fill

fillna stands for fill + na -> "fill na(Not Available) values"

```
#Fills missing values in the Salary column with the mean salary.
df['Salary'] = df['Salary'].fillna(df['Salary'].mean())
df
```

	ID	Name	Age	Salary	Department
0	1	Alice	25.0	50000.0	HR
1	2	Bob	30.0	56000.0	IT
2	3	Henry	27.0	62000.0	Finance
3	4	David	28.0	55000.0	IT
4	5	Eve	26.0	52000.0	NaN
5	6	John	27.0	54000.0	Operations
6	1	Alice	25.0	50000.0	HR
7	8	Grace	29.0	58000.0	Marketing
8	3	Henry	27.0	62000.0	Finance
9	10	Ivy	31.0	61000.0	Marketing

```
#Replaces missing values in the Department column with the text "Unknown"
df['Department'] = df['Department'].fillna('Unknown')
df
```

	ID	Name	Age	Salary	Department
0	1	Alice	25.0	50000.0	HR
1	2	Bob	30.0	56000.0	IT
2	3	Henry	27.0	62000.0	Finance
3	4	David	28.0	55000.0	IT
4	5	Eve	26.0	52000.0	Unknown
5	6	John	27.0	54000.0	Operations
6	1	Alice	25.0	50000.0	HR
7	8	Grace	29.0	58000.0	Marketing
8	3	Henry	27.0	62000.0	Finance
9	10	Ivy	31.0	61000.0	Marketing

4. Save a final dataframe to a new csv file:

```
#Saves the cleaned DataFrame to a new CSV file named students_cleaned.csv
df.to_csv("students_cleaned.csv", index=False)
```

Result:

Thus the experiment successfully generates the cleaned dataset contains no missing values.

EX 6: CSV Merge and Duplicate Removal

Aim:

To load two CSV files into two DataFrames (d1, d2), combine both DataFrames, find and remove duplicate rows, and rename the index values using Python Pandas.

PROCEDURE:

1. Import the Pandas library.
2. Load the first CSV file into DataFrame df1.
3. Load the second CSV file into DataFrame df2.
4. Combine both DataFrames using concat().
5. Identify and remove duplicate rows using drop_duplicates().
6. Rename / reset the index using reset_index().
7. Display the final DataFrame.

Program

```
# Load two CSV files into DataFrames

import pandas as pd
df1 = pd.read_csv("file1.csv")
df2 = pd.read_csv("file2.csv")
print(df1)
print("******************")
print(df2)
```

```
    ID   Name  Marks
0  101   Ravi     85
1  102    Anu     90
2  103  Kumar     78
3  104  Priya     88
******************
    ID    Name  Marks
0  103   Kumar     78
1  104   Priya     88
2  105  Suresh     92
3  106   Meena     80
```

```python
# Combine both DataFrames

combined_df = pd.concat([df1, df2])
print(combined_df)
```

```
    ID    Name  Marks
0  101    Ravi     85
1  102     Anu     90
2  103   Kumar     78
3  104   Priya     88
0  103   Kumar     78
1  104   Priya     88
2  105  Suresh     92
3  106   Meena     80
```

```python
# Remove duplicate rows

unique_df = combined_df.drop_duplicates()
print(unique_df)
```

```
    ID    Name  Marks
0  101    Ravi     85
1  102     Anu     90
2  103   Kumar     78
3  104   Priya     88
2  105  Suresh     92
3  106   Meena     80
```

```python
# Rename (reset) index
final_df = unique_df.reset_index(drop=True)
print("Final DataFrame after removing duplicates and renaming index:")
print(final_df)
```

```
Final DataFrame after removing duplicates and renaming index:
    ID    Name  Marks
0  101    Ravi     85
1  102     Anu     90
2  103   Kumar     78
3  104   Priya     88
4  105  Suresh     92
5  106   Meena     80
```

Result:

Thus, two CSV files were loaded into DataFrames, combined successfully, duplicate rows were removed, and the index was renamed using Pandas.

EX 7: Visualization and Mean Feature Analysis of Iris Setosa

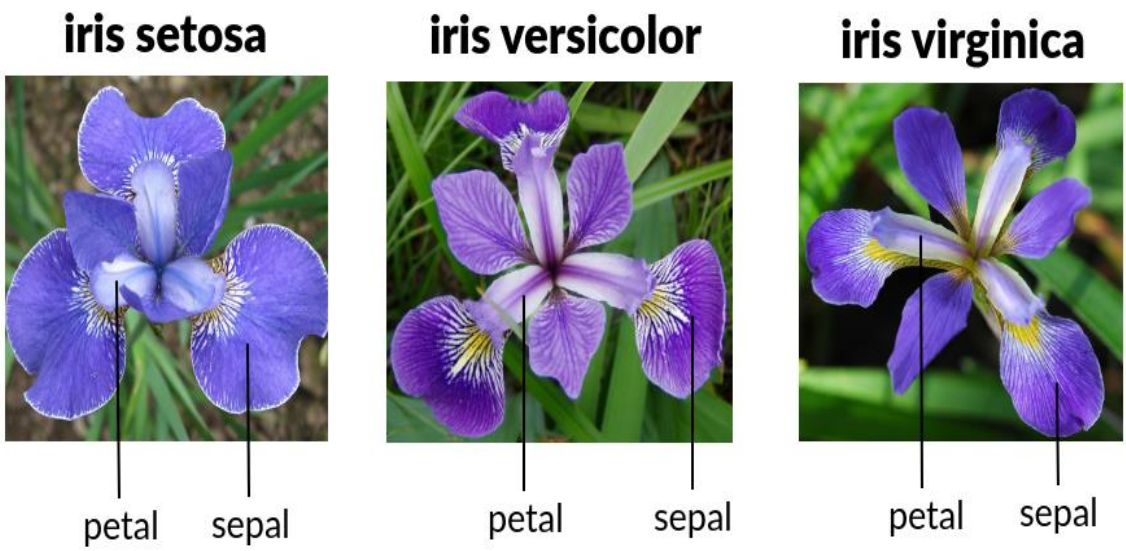
Aim:

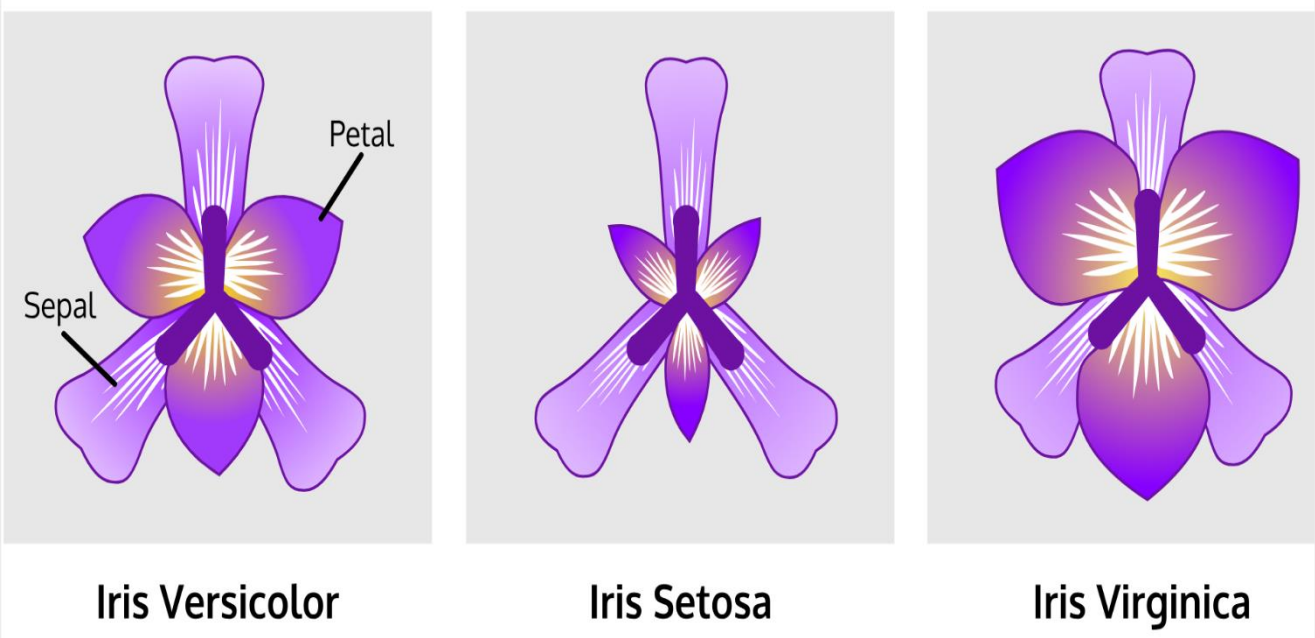
To Load the Iris dataset and visualize the average value of each feature for the Iris Setosa class using a bar chart.

PROCEDURE:

- Load the Iris dataset
- Select observations belonging to Setosa
- Compute mean of each feature
- Plot a bar chart with suitable colour and linewidth

Iris flower



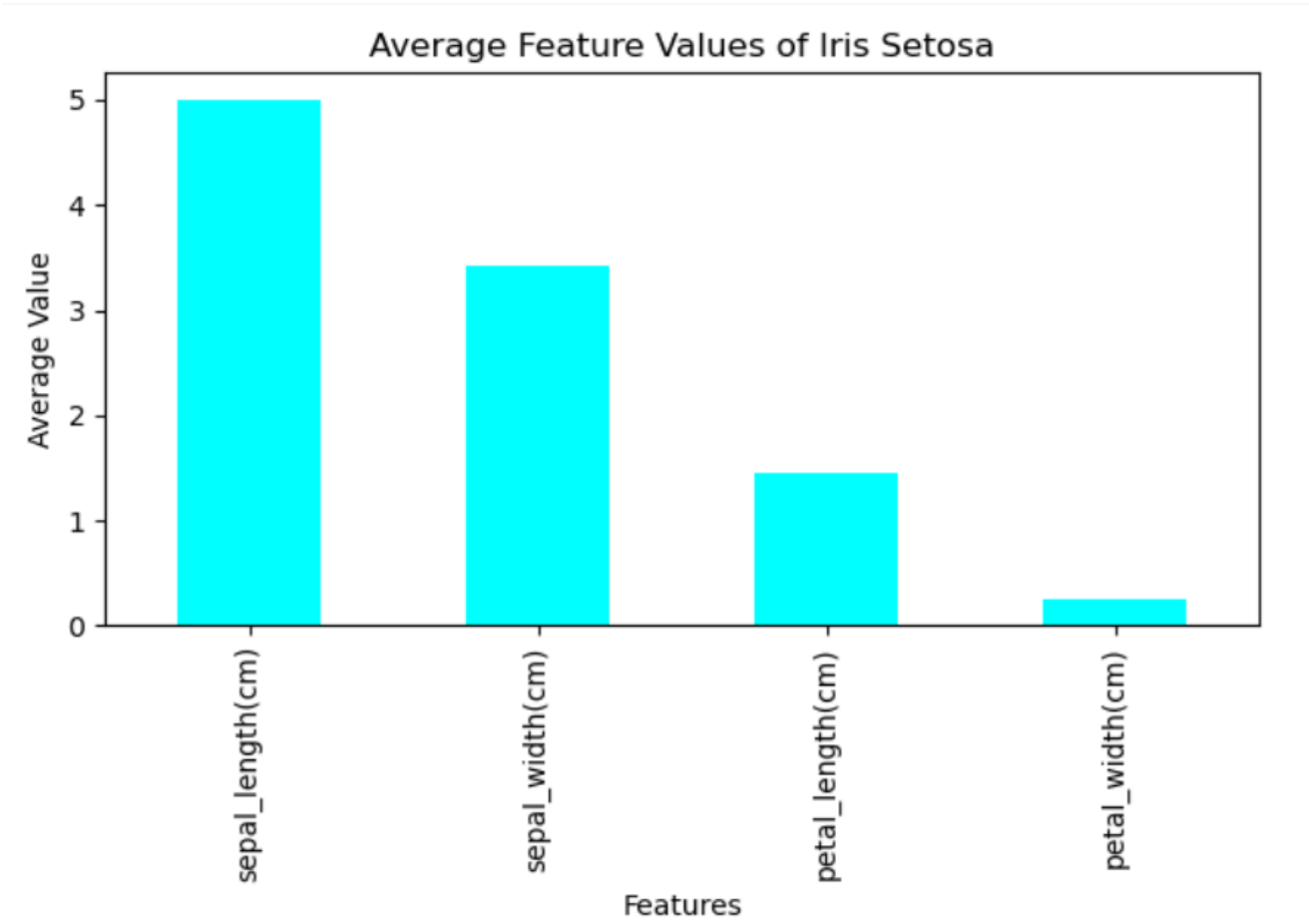


In iris flowers, sepals are large and colourful, while petals are smaller and upright, unlike typical flowers where petals are colourful and sepals are green.

Program

```
import pandas as pd
import matplotlib.pyplot as plt

# Load Iris dataset from CSV file
df = pd.read_csv("iris_dataset.csv")

# Filter Setosa class
setosa = df[df['species'] == 'setosa']

# Calculate average value of each feature
feature_mean = setosa[['sepal_length(cm)','sepal_width(cm)','petal_length(cm)','petal_width(cm)']].mean()

# Plot bar chart
plt.figure()
feature_mean.plot(kind='bar', linewidth=2, color='cyan')
plt.title("Average Feature Values of Iris Setosa")
plt.xlabel("Features")
plt.ylabel("Average Value")
plt.tight_layout()
plt.show()
```

Output:



Observation (Iris Setosa – Mean Values)

- Sepal length has the highest average value
- Sepal width is moderate
- Petal length is significantly smaller than sepals
- Petal width has the lowest average value

This shows that Iris Setosa has short and narrow petals compared to its sepals, which is a key distinguishing characteristic of this species.

Result:

The Iris dataset was successfully loaded and the Setosa class was isolated. The mean values of the four features—sepal length, sepal width, petal length, and petal width—were calculated and visualized using a bar chart.

EX 8: Multivariate Analysis using Pairplot on Iris Dataset

Aim:

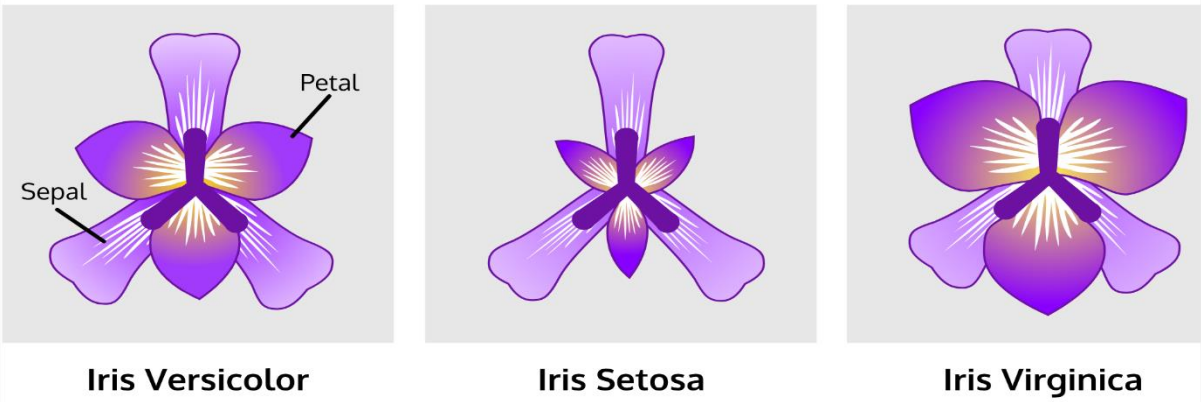
To load the Iris dataset, analyze the relationships between all feature columns using a pairplot for multivariate analysis, and save the generated plot as a JPEG file using Python.

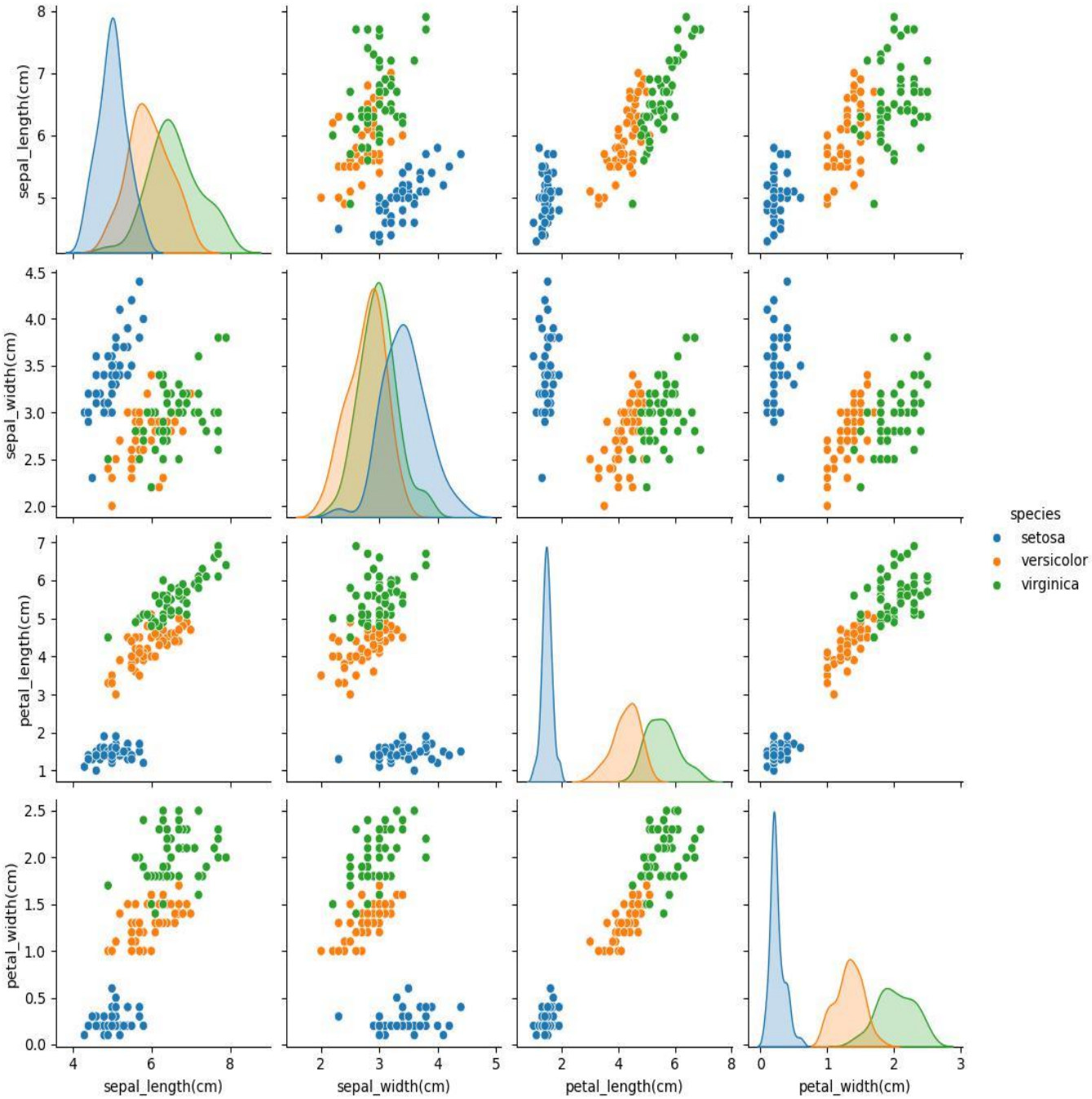
PROCEDURE:

1. Import the required libraries such as Pandas, Seaborn, and Matplotlib.
2. Load the Iris dataset into a Pandas DataFrame.
3. Use Seaborn's pairplot() function to visualize relationships among all columns.
4. Customize the pairplot using hue for species classification.
5. Identify and remove duplicate rows using drop_duplicates().
6. Save the generated pairplot as a JPEG file.
7. Display the plot output.

Program

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv("iris_dataset.csv")
print(df)
sns.pairplot(df, hue="species")
plt.savefig("iris_pairplot.jpg")
plt.show()
```


Petal
Sepal
Iris Versicolor
Iris Setosa
Iris Virginica


sepal_length(cm)
sepal_width(cm)
petal_length(cm)
petal_width(cm)
species
setosa
versicolor
virginica

Pairplot:

- It shows relationships between every pair of features in the Iris dataset.
- Features used:
  - Sepal length
  - Sepal width
  - Petal length
  - Petal width
- Different colors represent different species:
  - Blue → Setosa
  - Orange → Versicolor
  - Green → Virginica

Components of Pairplot:

1. **Diagonal plots (distribution plots):**

- These show data distribution of each single feature.
- It shows
  - Sepal length and Sepal width distribution for all three species
  - Petal length and Petal width distribution for all three species
- Observation:
  - Setosa is clearly separated in petal length and petal width.
  - Versicolor and Virginica show some overlap.

2. Off-diagonal plots (scatter plots):

- These show relationship between two different features.
- It shows
  - X-axis → one feature
  - Y-axis → another feature
  - Each dot → one flower sample
- Purpose:
  - To observe correlation and class separation between two features.
  - Detect overlapping classes.

Analysis of Feature Relationships

1. Sepal Length vs Sepal Width

- Points are widely scattered.
- No clear linear pattern → weak correlation.
- Species overlap significantly.
- Conclusion: Sepal features alone are not reliable for classification.

2. Sepal Length vs Petal Length

- Clear upward trend → positive correlation.
- Setosa points lie far apart from others.
- Versicolor and Virginica show partial overlap.

  Conclusion: Petal length improves class separability.

3. Sepal Length vs Petal Width

- Moderate to strong positive correlation.
- Setosa is clearly isolated.
- Versicolor and Virginica still overlap.

  Conclusion: Useful feature pair for classification.

4. Sepal Width vs Petal Length

- Setosa forms a tight, separate cluster.
- Other two species spread along increasing petal length.

  Conclusion: Petal length dominates species separation.

5. Sepal Width vs Petal Width

- Clear grouping of Setosa.
- Overlap between Versicolor and Virginica.

  Conclusion: Sepal width alone is weak, but petal width helps.

6. Petal Length vs Petal Width

- Strong linear relationship.
- Three visible clusters:
  - Setosa → completely separated
  - Versicolor → middle cluster
  - Virginica → upper cluster

Conclusion: Best feature combination for ML classification.

Interpretation:

This pairplot visually demonstrates the relationships among all features of the Iris dataset and shows that petal features provide better class separation than sepal features.

Result:

Thus, the Iris dataset was successfully loaded, relationships between all feature columns were visualized using a pairplot for multivariate analysis, and the plot was saved as a JPEG file.

EX 9: Clustering using Iris Dataset and Analysis using Decision Tree

Aim:

To perform clustering on the Iris dataset using K-Means and analyze the clustered data using a Decision Tree model.

PROCEDURE:

1. Import the required libraries such as Pandas, Sklearn, and Matplotlib.
2. Load the Iris dataset into a Pandas DataFrame.
3. Apply K-Means clustering with k = 3.
4. Assign cluster labels to the dataset.
5. Train a Decision Tree using cluster labels.
6. Visualize the Decision Tree.
7. Analyze feature-based splits.

Algorithm:

ID3 Algorithm (Iterative Dichotomiser 3):

uses Information Gain based on Entropy to build the decision tree.

Entropy: Measures impurity or disorder of a dataset.

$$\textbf{Entropy}(S) = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

where S = current dataset

p_i = probability of class i in dataset S

n = Number of unique classes in the dataset

Information Gain (IG): Reduction in entropy after splitting.

$$\mathbf{IG}(\mathbf{S},\mathbf{A}) = \textbf{Entropy}(S) - \sum_{v \in \textbf{Values}\,(A)} \frac{|S_v|}{|S|}\,\textbf{Entropy}(S_v)$$

where $\mathbf{IG}(\mathbf{S},\mathbf{A})$ = Information Gain from the dataset S using attribute A.

Entropy(S) = Entropy of the original dataset

S_v = The subset of dataset(S) where the attribute A has value v

Entrophy(S_v) = The entropy of the subset dataset.

|S| = Total number of samples in the dataset

|Sv| = Total number of samples in subset dataset

v ∈ Values(A) = v is the set of all values of the attribute A.

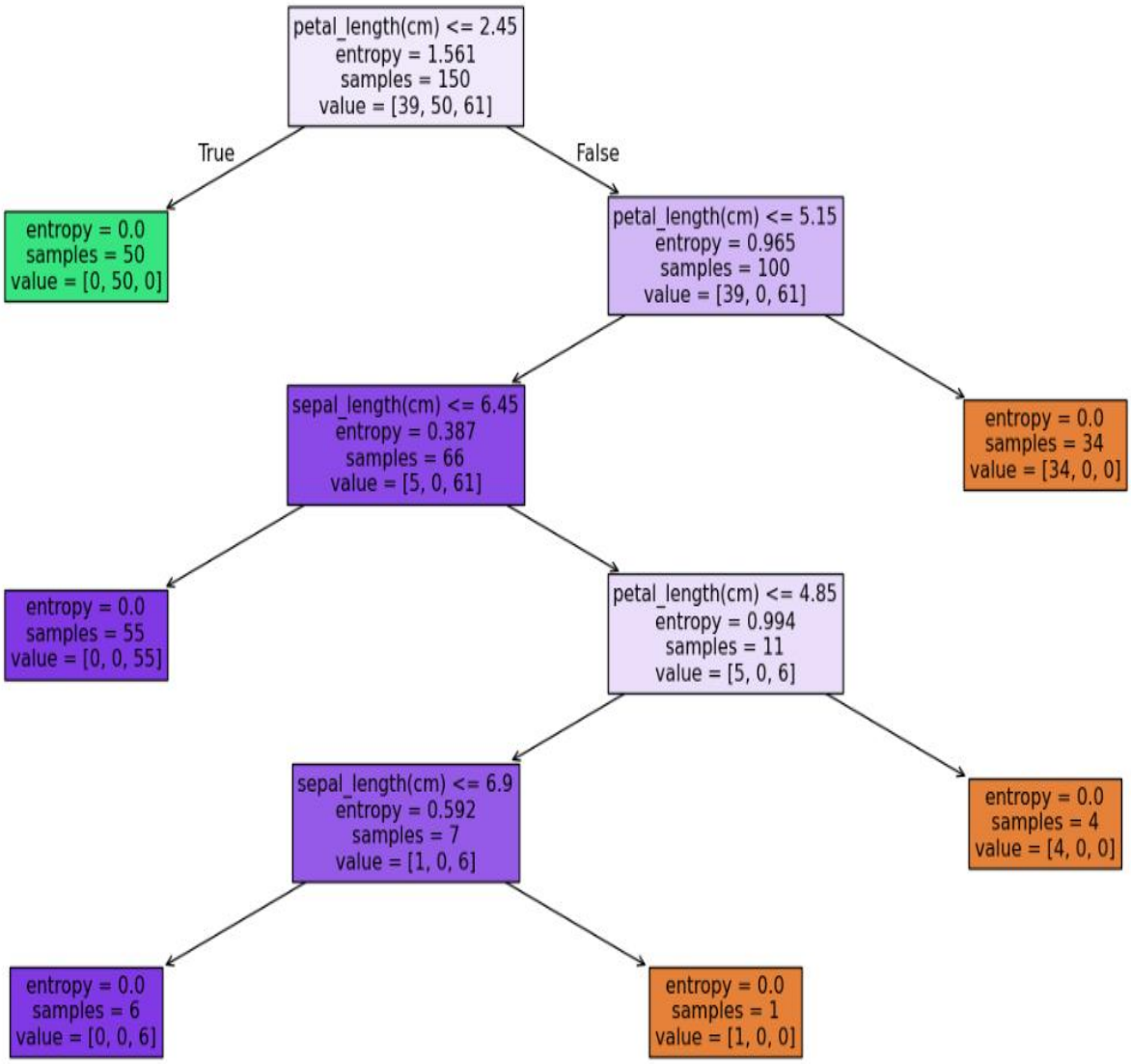
Program

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.tree import DecisionTreeClassifier, plot_tree

# Load Iris dataset from CSV file
df = pd.read_csv("iris_dataset.csv")

# Select feature columns
X = df.drop("species", axis=1)

# Apply K-Means clustering
kmeans = KMeans(n_clusters=3, random_state=42)
cluster_labels = kmeans.fit_predict(X)

# Train Decision Tree using cluster labels
dt = DecisionTreeClassifier(criterion="entropy")
dt.fit(X, cluster_labels)

# Plot Decision Tree
plt.figure(figsize=(20, 10))
plot_tree(dt, feature_names=X.columns, filled=True,fontsize=12)
plt.show()
```

Analysis:

- K-Means algorithm groups the Iris dataset into three clusters based on similarity of features.
- Each cluster represents a group of flowers with similar measurements.
- Decision Tree model is applied to the clustered data to explain how clusters are formed.
- The tree uses simple if–else rules to separate the clusters.
- Petal length and petal width are the most important features for cluster separation.
- Clusters are divided using feature threshold values at each node of the tree.

Output:



Result:

Thus, clustering was performed using K-Means, and the clustered data was analyzed using a Decision Tree classifier.

EX 10: Clustering using Iris Dataset and Analysis using K-Means Clustering

Aim:

To cluster the Iris dataset using the K-Means algorithm and analyze the cluster formation.

PROCEDURE:

1. Import the required libraries such as Pandas, Sklearn, and Matplotlib.
2. Load the Iris dataset into a Pandas DataFrame.
3. Apply K-Means clustering with k = 3.
4. Assign cluster labels to the dataset.
5. Visualize the clusters.
6. Analyze cluster separation.

K-Means Clustering Algorithm:

Step 1: Initialize

- Choose the number of clusters, k.
- Randomly select k initial centroids (cluster centers) from the dataset.

Step 2: Assign Points to Clusters

- For each data point, calculate the Euclidean distance to each centroid.
- Assign the point to the nearest centroid's cluster.

Step 3: Update Centroids

- After all points are assigned, recalculate each centroid as the mean position of all data points in that cluster.

Step 4: Repeat

- Repeat Step 2 and Step 3 until centroids stop changing significantly or a maximum number of iterations is reached.

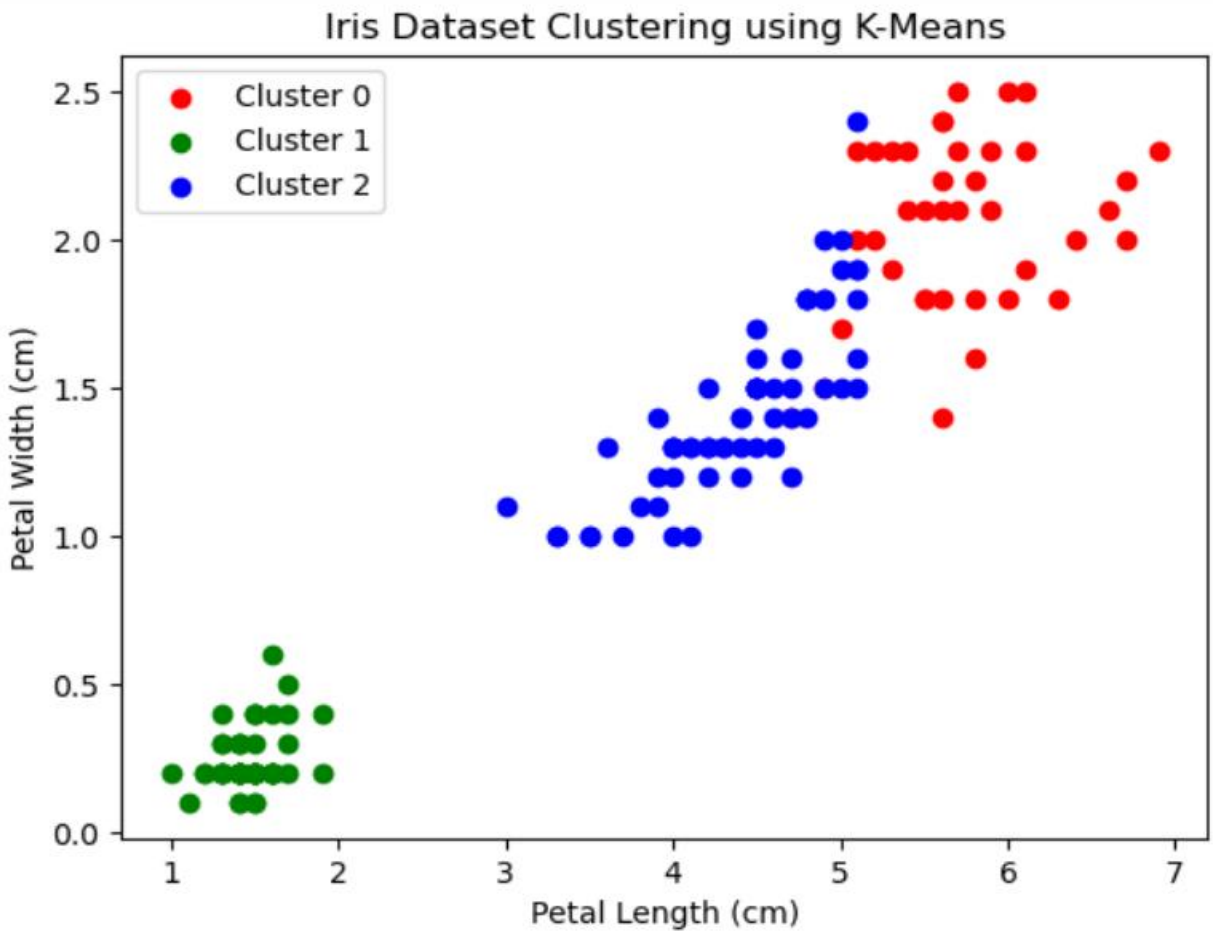
Program

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

# Load Iris dataset
df = pd.read_csv("iris_dataset.csv")

# Select feature columns
X = df.drop("species", axis=1)

# Apply K-Means clustering
kmeans = KMeans(n_clusters=3, random_state=42)
df["cluster"] = kmeans.fit_predict(X)

# Split data by clusters
df0 = df[df.cluster == 0]
df1 = df[df.cluster == 1]
df2 = df[df.cluster == 2]

# Visualize clusters using petal features
plt.scatter(df0["petal_length(cm)"], df0["petal_width(cm)"],color="red", label="Cluster 0")
plt.scatter(df1["petal_length(cm)"], df1["petal_width(cm)"],color="green", label="Cluster 1")
plt.scatter(df2["petal_length(cm)"], df2["petal_width(cm)"],color="blue", label="Cluster 2")

plt.xlabel("Petal Length (cm)")
plt.ylabel("Petal Width (cm)")
plt.title("Iris Dataset Clustering using K-Means")
plt.legend()
plt.show()
```

Analysis

- K-Means groups data based on distance.
- Three clusters are formed.
- Setosa cluster is clearly separated.
- Other two clusters overlap slightly.

Output:



Result:

Thus, the Iris dataset was successfully clustered using the K-Means algorithm.