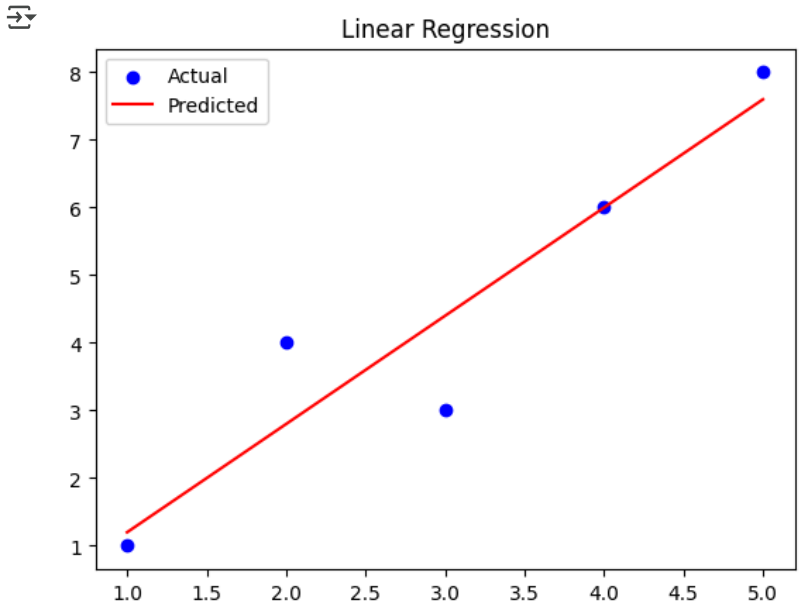
```
# Program1: Write a python program to build linear regression model for the data and plot the fitted line.

from sklearn.linear_model import LinearRegression
import numpy as np
import matplotlib.pyplot as plt

# Sample data

X = np.array([[1], [2], [3], [4], [5]])
y = np.array([1, 4, 3, 6, 8])

# Model

model = LinearRegression()
model.fit(X, y)

# Prediction
y_pred = model.predict(X)

# Plot

plt.scatter(X, y, color='blue', label='Actual')
plt.plot(X, y_pred, color='red', label='Predicted')
plt.title('Linear Regression')
plt.legend()
plt.show()
```

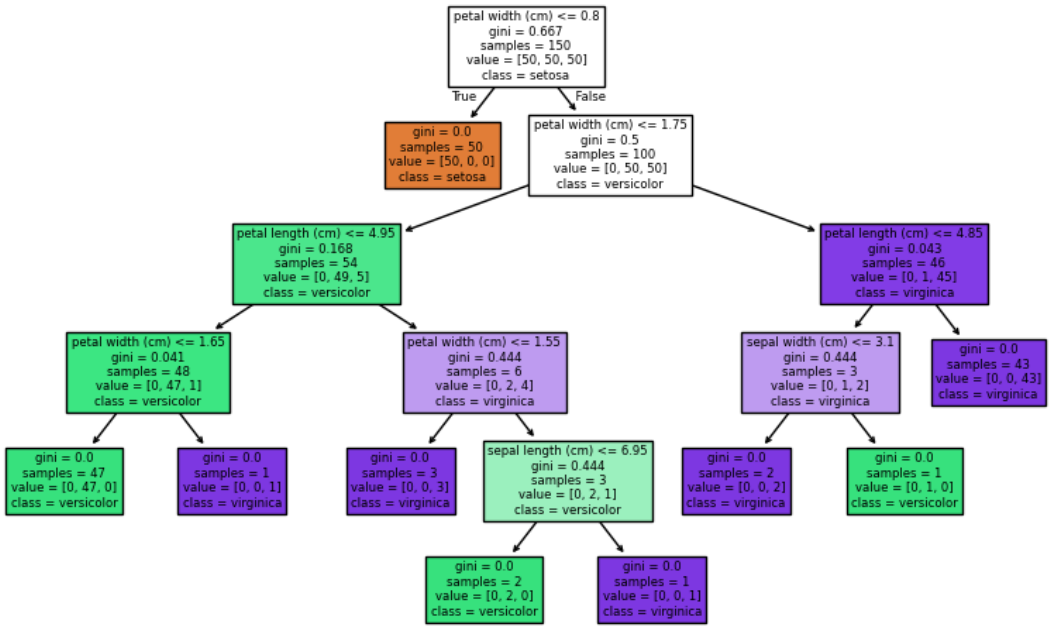



```
# Program2: Write a python program to build the linear regression model to predict the 7th and 8th month data using 5 months sal

import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Months (1 to 5)
months = np.array([1, 2, 3, 4, 5]).reshape(-1, 1)

# Sales for these months (you can replace with actual data)
sales = np.array([150, 200, 250, 300, 330])

# Create and train model
model = LinearRegression()
model.fit(months, sales)

# Predict sales for 7th and 8th month
future_months = np.array([7, 8]).reshape(-1, 1)
predicted_sales = model.predict(future_months)

print(f"Predicted Sales for
print(f"Predicted Sales for

# Plotting
```

```
plt.scatter(months, sales, color='blue', label='Actual Sales')
plt.plot(months, model.predict(months), color='green', label='Regression Line')
plt.scatter(future_months, predicted_sales, color='red', label='Predicted Sales')
plt.xlabel('Month')
plt.ylabel('Sales')
plt.title('Sales Prediction using Linear Regression')
plt.legend()
plt.grid(True)
plt.show()
```

```
Predicted Sales for 7th Month: 430.00
Predicted Sales for 8th Month: 476.00
```


```
# Program3: Write a python program to implement decision tree classifier for the iris dataset and visualize the tree.

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

# Load dataset
iris = load_iris()
clf = DecisionTreeClassifier()
clf = clf.fit(iris.data, iris.target)

# Visualize the tree
plt.figure(figsize=(10, 6))
tree.plot_tree(clf, filled=True, feature_names=iris.feature_names, class_names=iris.target_names)
plt.show()
```



#Program 4: Write a python program to implement Naive bayesian classifier for the sample dataset given with attributes GPA, Intr

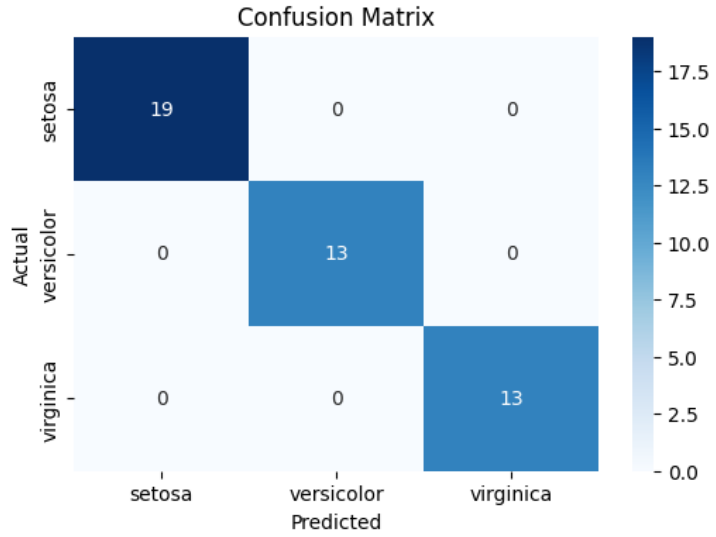
```
#import the necessary libraries
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import CategoricalNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Sample dataset
data = {
    'GPA': ['High', 'Low', 'Medium', 'High', 'Low', 'Medium', 'High'],
    'Internships': ['Yes', 'No', 'Yes', 'No', 'No', 'No', 'Yes'],
    'Projects': ['Good', 'Poor', 'Good', 'Average', 'Poor', 'Average', 'Good'],
    'Communication': ['Good', 'Poor', 'Average', 'Good', 'Poor', 'Average', 'Good'],
    'JobOffer': ['Yes', 'No', 'Yes', 'Yes', 'No', 'No', 'Yes']
}

df = pd.DataFrame(data)

# Encode categorical features
le = LabelEncoder()
for col in df.columns:
    df[col] = le.fit_transform(df[col])

# Features and target
X = df.drop('JobOffer', axis=1)
y = df['JobOffer']

# Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Naive Bayes for categorical data
model = CategoricalNB()
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

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

# Predict for new student
# GPA=High, Internship=Yes, Project=Good, Communication=Good
sample = pd.DataFrame([[2, 1, 1, 1]])  # You must encode the same way
prediction = model.predict(sample)
print("Job Offer Prediction (0=No, 1=Yes):", prediction[0])
```

```
Accuracy: 0.6666666666666666
Job Offer Prediction (0=No, 1=Yes): 1
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names,
  warnings.warn(
```

```
# Write a python program to implement KNN classifier for the iris dataset. Calculate accuracy, precision, F-score and Display th

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, precision_score, confusion_matrix, f1_score, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset (replace this with your own dataset if needed)
data = load_iris()
X = data.data
y = data.target

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

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

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

# Accuracy
acc = accuracy_score(y_test, y_pred)
print("Accuracy:", acc)

# Precision (average='macro' for multi-class)
prec = precision_score(y_test, y_pred, average='macro')
print("Precision:", prec)

# F1 Score
f1 = f1_score(y_test, y_pred, average='macro')
print("F1 Score:", f1)

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)

# Display Confusion Matrix as Heatmap
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='d', xticklabels=data.target_names, yticklabels=data.target_names)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Detailed classification report
print("\nClassification Report:\n", classification_report(y_test, y_pred, target_names=data.target_names))
```

```
Accuracy: 1.0
Precision: 1.0
F1 Score: 1.0
Confusion Matrix:
 [[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
```

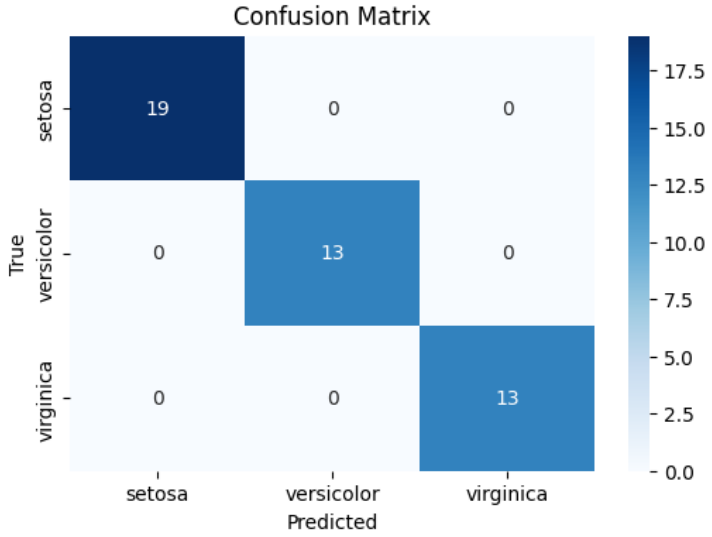



```
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

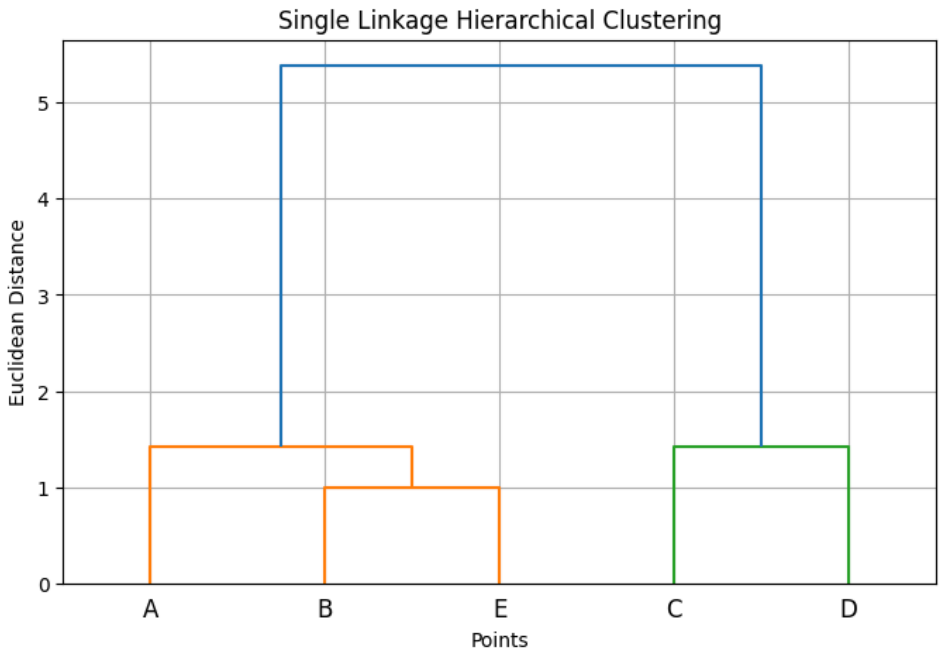
```
# Program 6:Write a python program to implement SVM classifier for the iris dataset. Calculate accuracy, precision, F-score and

from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset (use your own dataset here)
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create SVM classifier
svm = SVC(kernel='linear')  # You can change to 'rbf' or 'poly' as needed
svm.fit(X_train, y_train)

# Make predictions
y_pred = svm.predict(X_test)

# Evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')  # or 'micro', 'weighted'
recall = recall_score(y_test, y_pred, average='macro')
f1 = f1_score(y_test, y_pred, average='macro')
conf_matrix = confusion_matrix(y_test, y_pred)

# Display results
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
# Plot confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=iris.target_names,
            yticklabels=iris.target_names)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

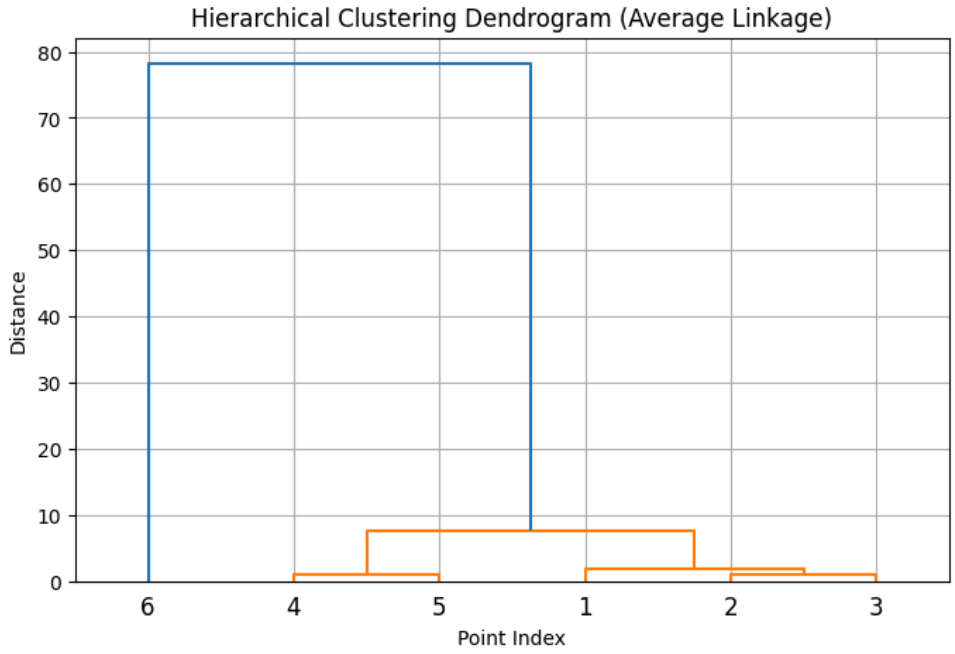
```
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1 Score: 1.0

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```




```
# Program 7: Write a python program to impliment single linkage hierarchical clustering algorithm and show the dendrogram.
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import linkage, dendrogram

# Define points
points = np.array([
    [1, 2],  # A
    [2, 3],  # B
    [5, 8],  # C
    [6, 9],  # D
    [3, 3],  # E
])

labels = ['A', 'B', 'C', 'D', 'E']

# Apply Single Linkage Clustering
linked = linkage(points, method='single')

# Plot dendrogram
plt.figure(figsize=(8, 5))
dendrogram(linked, labels=labels, distance_sort='ascending')
plt.title('Single Linkage Hierarchical Clustering')
plt.xlabel('Points')
plt.ylabel('Euclidean Distance')
plt.grid(True)
plt.show()
```




```
# Program 8: Write a python program to impliment average linkage hierarchical clustering algorithm and show the dendrogram.

import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage

# Sample data: array of 2D points
points = np.array([
    [1, 2],
    [2, 3],
    [3, 3],
    [8, 7],
    [8, 8],
    [25, 80]
])

# Apply average linkage hierarchical clustering
linked = linkage(points, method='average')

# Plot dendrogram
plt.figure(figsize=(8, 5))
dendrogram(linked,
           labels=np.arange(1, len(points)+1),
           distance_sort='ascending',
           show_leaf_counts=True)
plt.title('Hierarchical Clustering Dendrogram (Average Linkage)')
plt.xlabel('Point Index')
plt.ylabel('Distance')
plt.grid(True)
plt.show()
```




```
# Program 9:Write a python program to impliment complete linkage hierarchical clustering algorithm and show the dendrogram.
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage

# Sample data points
points = np.array([
    [1, 2],
    [2, 3],
    [3, 3],
    [5, 8],
    [6, 8],
    [7, 9]
])

# Perform hierarchical clustering using complete linkage
linked = linkage(points, method='complete')

# Plot the dendrogram
plt.figure(figsize=(8, 5))
dendrogram(linked, labels=range(1, len(points)+1))
plt.title('Hierarchical Clustering Dendrogram (Complete Linkage)')
plt.xlabel('Data Point Index')
plt.ylabel('Distance')
plt.grid(True)
plt.show()
```


```
# Program 10: Write a python program to impliment Kmeans clustering algorithm.

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

# Generate sample data
X, _ = make_blobs(n_samples=300, centers=3, cluster_std=0.60, random_state=0)

# Apply K-Means clustering
kmeans = KMeans(n_clusters=3, random_state=0)
kmeans.fit(X)
```