1.Simple python program using conditional statements,looping,performing operations such as insert,update,delete,display,sorting and searching on data types like List,Tuples,set ,dicitionary

```
def list_operations():
    my_list = []
    print("\n--- List Operations ---")
    print("1. Insert")
    print("2. Update")
    print("3. Delete")
    print("4. Display")
    print("5. Sort")
    print("6. Search")
    print("7. Exit")

    while True:
        choice = input("Enter choice: ")

        if choice == '1':
            value = input("Enter value to insert: ")
            my_list.append(value)
        elif choice == '2':
            old = input("Enter value to update: ")
            if old in my_list:
                new = input("Enter new value: ")
                my_list[my_list.index(old)] = new
            else:
                print("Value not found.")
        elif choice == '3':
            value = input("Enter value to delete: ")
            if value in my_list:
                my_list.remove(value)
            else:
                print("Value not found.")
        elif choice == '4':
            print("List contents:", my_list)
        elif choice == '5':
            my_list.sort()
            print("Sorted List:", my_list)
        elif choice == '6':
            search = input("Enter value to search: ")
            print("Found!" if search in my_list else "Not found.")
        elif choice == '7':
            break
        else:
            print("Invalid choice.")
```

```python
def tuple_operations():
    my_tuple = ("apple", "banana", "cherry")
    print("\n--- Tuple Operations (Immutable) ---")
    print("Original Tuple:", my_tuple)

    while True:
        print("\n1. Display")
        print("2. Search")
        print("3. Convert to List and Add Item")
        print("4. Exit")
        choice = input("Enter choice: ")

        if choice == '1':
            print("Tuple Contents:", my_tuple)
        elif choice == '2':
            item = input("Enter item to search: ")
            print("Found!" if item in my_tuple else "Not found.")
        elif choice == '3':
            item = input("Enter item to add: ")
            temp = list(my_tuple)
            temp.append(item)
            my_tuple = tuple(temp)
            print("Updated Tuple:", my_tuple)
        elif choice == '4':
            break
        else:
            print("Invalid choice.")


def set_operations():
    my_set = set()
    print("\n--- Set Operations ---")
    print("1. Insert")
    print("2. Delete")
    print("3. Display")
    print("4. Search")
    print("5. Exit")

    while True:
        choice = input("Enter choice: ")

        if choice == '1':
            value = input("Enter value to insert: ")
```

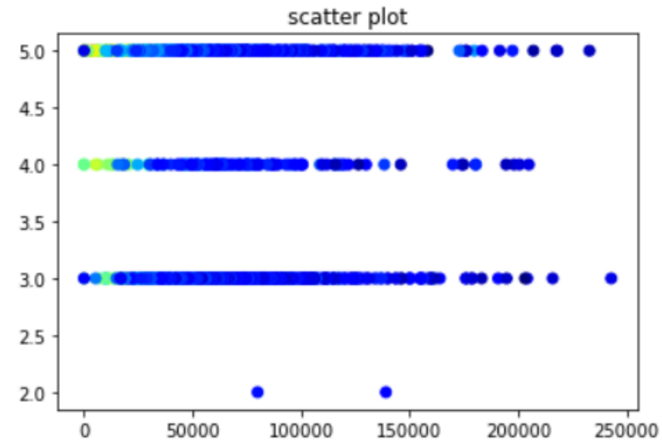
```
        my_set.add(value)
    elif choice == '2':
        value = input("Enter value to delete: ")
        my_set.discard(value)
    elif choice == '3':
        print("Set contents:", my_set)
    elif choice == '4':
        value = input("Enter value to search: ")
        print("Found!" if value in my_set else "Not found.")
    elif choice == '5':
        break
    else:
        print("Invalid choice.")


def dict_operations():
  my_dict = {}
  print("\n--- Dictionary Operations ---")
  print("1. Insert")
  print("2. Update")
  print("3. Delete")
  print("4. Display")
  print("5. Search")
  print("6. Exit")

  while True:
    choice = input("Enter choice: ")

    if choice == '1':
      key = input("Enter key: ")
      value = input("Enter value: ")
      my_dict[key] = value
    elif choice == '2':
      key = input("Enter key to update: ")
      if key in my_dict:
        value = input("Enter new value: ")
        my_dict[key] = value
      else:
        print("Key not found.")
    elif choice == '3':
      key = input("Enter key to delete: ")
      if key in my_dict:
        del my_dict[key]
      else:
        print("Key not found.")
```

```
        elif choice == '4':
            print("Dictionary contents:", my_dict)
        elif choice == '5':
            key = input("Enter key to search: ")
            print("Found!" if key in my_dict else "Not found.")
        elif choice == '6':
            break
        else:
            print("Invalid choice.")


# Main Program Loop
print("\n===== Main Menu =====")
print("1. List")
print("2. Tuple")
print("3. Set")
print("4. Dictionary")
print("5. Exit")

while True:
    main_choice = input("Enter Choice: ")

    if main_choice == '1':
        list_operations()
    elif main_choice == '2':
        tuple_operations()
    elif main_choice == '3':
        set_operations()
    elif main_choice == '4':
        dict_operations()
    elif main_choice == '5':
        print("Exiting Program.")
        break
    else:
        print("Invalid choice. Try again.")
```

2.Visualize the n-dimensional data using :

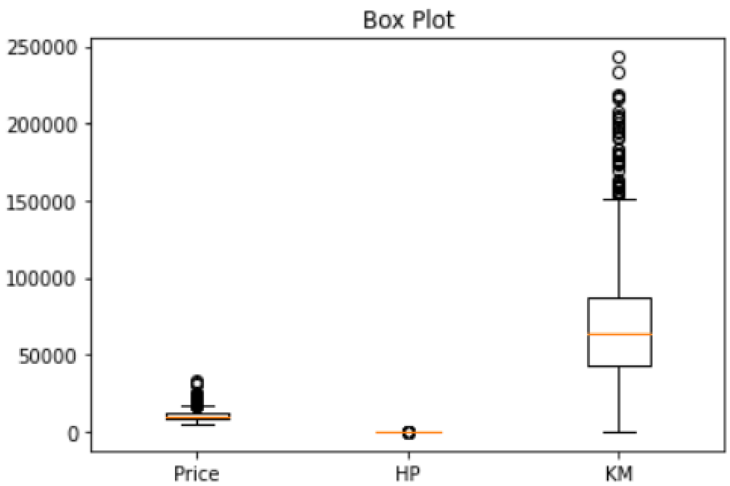
a)Scatter plot

```
import pandas as pd
import matplotlib.pyplot as plt
data = pd.read_csv("ToyotaCorolla.csv")
x=data['KM']
y=data['Doors']
plt.scatter(x,y,c=data['Price'],cmap="jet")
plt.title("scatter plot")
plt.show()
```



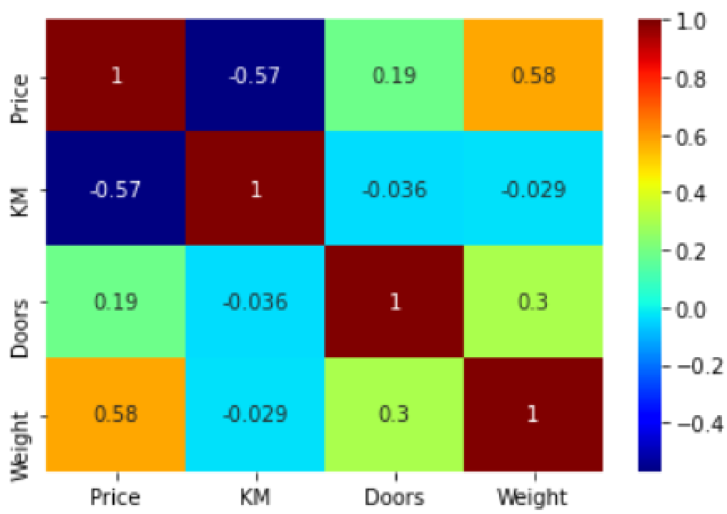
2.b)Box plot

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
data = pd.read_csv("ToyotaCorolla.csv")

#box plot
plt.title('Box Plot')
plt.boxplot([data["Price"],data["HP"],data["KM"]])

plt.xticks([1,2,3],["Price","HP","KM"])

plt.show()
```



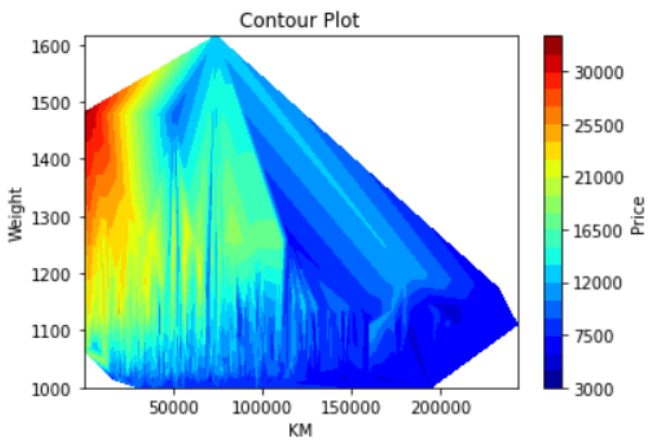
2.c)Heat map

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
data = pd.read_csv("ToyotaCorolla.csv")

#heat map
sns.heatmap(data[["Price","KM","Doors", "Weight"]].corr(),cmap='jet',annot=True)
plt.show()
```



2.d)Contour plot

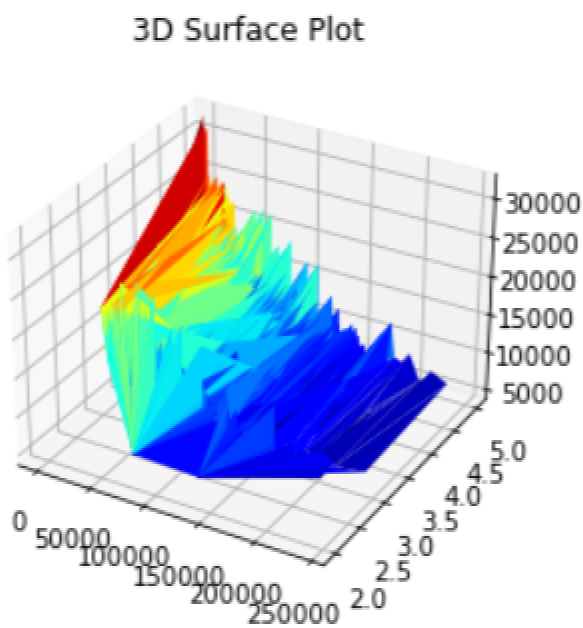
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
data = pd.read_csv("ToyotaCorolla.csv")

#contour plot
x = data['KM']
y = data['Weight']
z = data['Price']

plt.tricontourf(x, y, z, levels=20, cmap='jet')
plt.colorbar(label='Price')
plt.xlabel('KM')
plt.ylabel('Weight')
plt.title('Contour Plot')
plt.show()
```



2.e) 3D surface plot

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
data = pd.read_csv("ToyotaCorolla.csv")

#3d surface plot
x = data['KM']
y = data['Doors']
z = data['Price']
```

```python
ax = plt.axes(projection='3d')
ax.plot_trisurf(x,y,z,cmap="jet")
ax.set_title("3D Surface Plot")

plt.show()
```



3.Write a program to implement hill climbing algorithm

```python
def hill_climbing(graph, start, goal, heuristic):
    current = start
    path = [current]

    while current != goal:
        neighbors = graph[current]
        if not neighbors:
            print(f"No more neighbors to explore from {current}. Stuck at local maxima.")
            return path

        # Choose neighbor with lowest heuristic value (best next move)
        next_node = min(neighbors, key=lambda x: heuristic[x])
```

```
        # If no improvement, return current path (local maxima)
        if heuristic[next_node] >= heuristic[current]:
            print(f"Reached local maxima at {current}. Stopping search.")
            return path

        current = next_node
        path.append(current)

    return path


graph = {
    'A': ['B', 'C', 'D'],
    'B': ['A', 'E'],
    'C': ['A', 'E', 'D', 'F'],
    'D': ['A', 'F', 'C'],
    'E': ['B', 'C', 'H'],
    'F': ['G', 'C', 'D'],
    'G': [],
    'H': ['E', 'G']
}

start = 'A'
goal = 'G'

heuristic = {
    'A': 40,
    'B': 32,
    'C': 25,
    'D': 35,
    'E': 19,
    'F': 17,
    'G': 0,
    'H': 10
}

result = hill_climbing(graph, start, goal, heuristic)

if result and result[-1] == goal:
    print(f"Path found from {start} to {goal}: {result}")
else:
    print(f"No path found from {start} to {goal}. Reached: {result[-1]}")
```

```
Path found from A to G: ['A', 'C', 'F', 'G']
```

4.a)Write a program to implement the Best First Search (BFS) algorithm.

```python
def best_first_search(graph,start,goal,heuristic, path=[]):
    open_list = [(0,start)]
    closed_list = set()
    closed_list.add(start)

    while open_list:
        open_list.sort(key = lambda x: heuristic[x[1]], reverse=True)
        cost, node = open_list.pop()
        path.append(node)

        if node==goal:
            return cost, path

        closed_list.add(node)
        for neighbour, neighbour_cost in graph[node]:
            if neighbour not in closed_list:
                closed_list.add(neighbour)
                open_list.append((cost+neighbour_cost, neighbour))

    return None


graph = {
    'A': [('B', 11), ('C', 14), ('D',7)],
    'B': [('A', 11), ('E', 15)],
    'C': [('A', 14), ('E', 8), ('D',18), ('F',10)],
    'D': [('A', 7), ('F', 25), ('C',18)],
    'E': [('B', 15), ('C', 8), ('H',9)],
    'F': [('G', 20), ('C', 10), ('D',25)],
    'G': [],
    'H': [('E',9), ('G',10)]
}

start = 'A'
goal = 'G'

heuristic = {
    'A': 40,
    'B': 32,
    'C': 25,
    'D': 35,
    'E': 19,
    'F': 17,
    'G': 0,
    'H': 10
```

```python
}

result = best_first_search(graph, start, goal, heuristic)

if result:
    print(f"Minimum cost path from {start} to {goal} is {result[1]}")
    print(f"Cost: {result[0]}")
else:
    print(f"No path from {start} to {goal}")
```

```
Minimum cost path from A to G is ['A', 'C', 'F', 'G']
Cost: 44
```

4.b)Write a program to implement A* algorithm.

```python
def h(n):
    H = {'A': 3, 'B': 4, 'C': 2, 'D': 6, 'G': 0, 'S': 5}
    return H[n]

def a_star_algorithm(graph, start, goal):

    open_list = [start]
    closed_list = set()

    g = {start:0}

    parents = {start:start}

    while open_list:

        open_list.sort(key=lambda v: g[v] + h(v), reverse=True)
        n = open_list.pop()

        # If node is goal then construct the path and return
        if n == goal:
            reconst_path = []

            while parents[n] != n:
                reconst_path.append(n)
                n = parents[n]

            reconst_path.append(start)
            reconst_path.reverse()

            print(f'Path found: {reconst_path}')
```

```
            return reconst_path

        for (m, weight) in graph[n]:
        # if m is first visited, add it to open_list and note its parent
            if m not in open_list and m not in closed_list:
                open_list.append(m)
                parents[m] = n
                g[m] = g[n] + weight

            # otherwise, check if it's quicker to first visit n, then m
            # and if it is, update parent and g data
            # and if the node was in the closed_list, move it to open_list
            else:
                if g[m] > g[n] + weight:
                    g[m] = g[n] + weight
                    parents[m] = n

                    if m in closed_list:
                        closed_list.remove(m)
                        open_list.append(m)

        # Node's neighbours are visited. Now put node to closed list.
        closed_list.add(n)

    print('Path does not exist!')
    return None


graph = {
    'S': [('A', 1), ('G', 10)],
    'A': [('B', 2), ('C', 1)],
    'B': [('D', 5)],
    'C': [('D', 3),('G', 4)],
    'D': [('G', 2)]
}

a_star_algorithm(graph, 'S', 'G')
```

```
Path found: ['S', 'A', 'C', 'G']
```

5.Write a program to implement Min-Max algorithm.

```
def minmax(depth, nodeIndex, maximizingPlayer, values, alpha, beta):
    if depth == 3:
        return values[nodeIndex]

    if maximizingPlayer:
        best = float('-inf')
        for i in range(2):
            val = minmax(depth + 1, nodeIndex * 2 + i, False, values, alpha, beta)
            best = max(best, val)
        return best
    else:
        best = float('inf')
        for i in range(2):
            val = minmax(depth + 1, nodeIndex * 2 + i, True, values, alpha, beta)
            best = min(best, val)
        return best

# Example tree with depth 3 and 8 terminal nodes
values = [3, 5, 2, 9, 12, 5, 23, 23]

# Start the Min-Max algorithm
result = minmax(0, 0, True, values, float('-inf'), float('inf'))
print("The optimal value is:", result)
```

```
The optimal value is: 12
```

5 Write a program to implement Alpha-beta pruning algorithm.

```
def alphabeta(depth, nodeIndex, maximizingPlayer, values, alpha, beta):
    if depth == 3:
        return values[nodeIndex]

    if maximizingPlayer:
        best = float('-inf')
        for i in range(2):
            val = alphabeta(depth + 1, nodeIndex * 2 + i, False, values, alpha, beta)
            best = max(best, val)
            alpha = max(alpha, best)
            if beta <= alpha:
                break
        return best
```

```
    else:
        best = float('inf')
        for i in range(2):
            val = alphabeta(depth + 1, nodeIndex * 2 + i, True, values, alpha, beta)
            best = min(best, val)
            beta = min(beta, best)
            if beta <= alpha:
                break
        return best

# Example tree with depth 3 and 8 terminal nodes
values = [3, 5, 2, 9, 12, 5, 23, 23]

# Start the Alpha-Beta Pruning algorithm
result = alphabeta(0, 0, True, values, float('-inf'), float('inf'))
print("The optimal value is:", result)
```

```
The optimal value is: 12
```

6.Write a program to develop the naive bayes classifier based on split up of training and testing dataset as 90-10,70-30.

a) Iris dataset

```
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB

iris = pd.read_csv(r"Iris.csv")
iris.head()

X = iris.iloc[:, :-1].values
y = iris.iloc[:, -1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

classifier = GaussianNB()
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)
```

```
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
print('accuracy is', accuracy_score(y_test, y_pred))
```

```
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       1.00      1.00      1.00        13
 Iris-virginica       1.00      1.00      1.00         6

       accuracy                           1.00        30
      macro avg       1.00      1.00      1.00        30
   weighted avg       1.00      1.00      1.00        30

[[11  0  0]
 [ 0 13  0]
 [ 0  0  6]]
accuracy is 1.0
```

6.b) Titanic dataset

```
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder

# Load the dataset
df = pd.read_csv(r"Titanic-Dataset.csv")
df = df[['Survived', 'Pclass', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']]

# Handle missing values
imputer = SimpleImputer(strategy='median')
df[['Age', 'Fare']] = imputer.fit_transform(df[['Age', 'Fare']])

df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)
df['Embarked'] = LabelEncoder().fit_transform(df['Embarked'])

# Split the data into train and test sets
X = df.drop('Survived', axis=1)
y = df['Survived']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```python
# Initialize and fit the Gaussian Naive Bayes classifier
classifier = GaussianNB()
classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = classifier.predict(X_test)

# Evaluate the model
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Confusion Matrix:
 [[86 19]
 [37 37]]
Accuracy: 0.6871508379888268
```

7.Write a program to develop the KNN classifier for the k values as 3,5,7 based on split up of training and testing dataset as 90-10,70-30,

a)Glass dataset

```python
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

df = pd.read_csv('glass.csv')
y = df['Type'].values
X = df.drop('Type', axis=1).values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

# Using scikit-learn with Euclidean distance
clf_euclidean = KNeighborsClassifier(n_neighbors=3, metric='euclidean')
clf_euclidean.fit(X_train, y_train)
predictions_euclidean = clf_euclidean.predict(X_test)
accuracy_euclidean = accuracy_score(y_test, predictions_euclidean)
print("Accuracy with Euclidean distance:", accuracy_euclidean)
```

```
# Using scikit-learn with Manhattan distance
clf_manhattan = KNeighborsClassifier(n_neighbors=3, metric='manhattan')
clf_manhattan.fit(X_train, y_train)
predictions_manhattan = clf_manhattan.predict(X_test)
accuracy_manhattan = accuracy_score(y_test, predictions_manhattan)
print("Accuracy with Manhattan distance:", accuracy_manhattan)
```

```
Accuracy with Euclidean distance: 0.6153846153846154
Accuracy with Manhattan distance: 0.6461538461538462
```

7.b)Fruit dataset

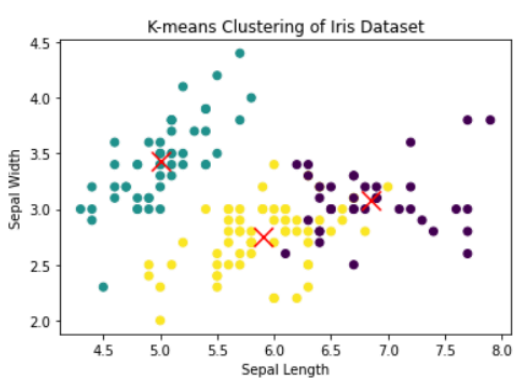
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Load dataset
df = pd.read_csv('fruits.csv')
y = df['fruit_label'].values
X = df[['mass', 'width', 'height', 'color_score']].values

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=45)

# Using scikit-learn with Euclidean distance
clf_euclidean = KNeighborsClassifier(n_neighbors=5, metric='euclidean')  # Fixed here
clf_euclidean.fit(X_train, y_train)
predictions_euclidean = clf_euclidean.predict(X_test)

accuracy_euclidean = accuracy_score(y_test, predictions_euclidean)
print("Accuracy with Euclidean distance (using sklearn):", accuracy_euclidean)

# Using scikit-learn with Manhattan distance
clf_manhattan = KNeighborsClassifier(n_neighbors=5, metric='manhattan')  # Fixed here
clf_manhattan.fit(X_train, y_train)
predictions_manhattan = clf_manhattan.predict(X_test)

accuracy_manhattan = accuracy_score(y_test, predictions_manhattan)
print("Accuracy with Manhattan distance (using sklearn):", accuracy_manhattan)
```

```
Accuracy with Euclidean distance (using sklearn): 0.5833333333333334
Accuracy with Manhattan distance (using sklearn): 0.5833333333333334
```

8.Write a program to perform unsupervised K-means clustering techniques

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans

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

# Number of clusters
K = 3

# K-means using scikit-learn
kmeans = KMeans(n_clusters=K, random_state=0)
labels = kmeans.fit_predict(X)
centroids = kmeans.cluster_centers_

# Print results
print("K-means Labels:", labels)
print("K-means Centroids:", centroids)

# Plotting K-means results
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
plt.scatter(centroids[:, 0], centroids[:, 1], marker='x', color='red', s=200)
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.title('K-means Clustering of Iris Dataset')
plt.show()
```

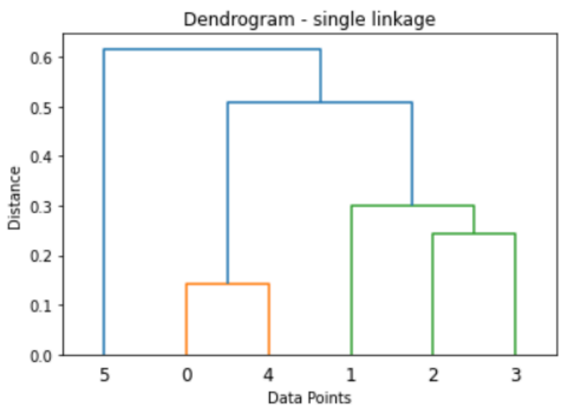
```
K-means Labels: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 2 0 0 0 0 2 0 0 0 0
 0 0 2 2 0 0 0 0 2 0 2 0 2 0 0 2 2 0 0 0 0 0 2 0 0 0 0 2 0 0 0 2 0 0 0 2 0
 0 2]
K-means Centroids: [[6.85       3.07368421 5.74210526 2.07105263]
 [5.006      3.428      1.462      0.246     ]
 [5.9016129  2.7483871  4.39354839 1.43387097]]
```

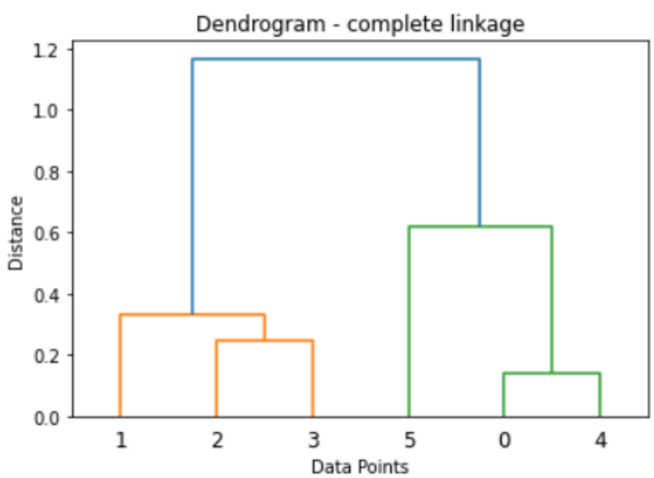


9.Write a program to perform agglomerative clustering based on single-linkage ,complete-linkage criteria

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.datasets import load_iris

iris = load_iris()
data = iris.data[:6]

def proximity_matrix(data):
  n = data.shape[0]
  proximity_matrix = np.zeros((n, n))
  for i in range(n):
    for j in range(i+1, n):
        proximity_matrix[i, j] = np.linalg.norm(data[i] - data[j])
        proximity_matrix[j, i] = proximity_matrix[i, j]
  return proximity_matrix

def plot_dendrogram(data, method):
  linkage_matrix = linkage(data, method=method)
  dendrogram(linkage_matrix)
  plt.title(f'Dendrogram - {method} linkage')
  plt.xlabel('Data Points')
  plt.ylabel('Distance')
  plt.show()

print("Proximity matrix:")
print(proximity_matrix(data))
plot_dendrogram(data, 'single')
plot_dendrogram(data, 'complete')
```

```
Proximity matrix:
[[0.         0.53851648 0.50990195 0.64807407 0.14142136 0.6164414 ]
 [0.53851648 0.         0.3        0.33166248 0.60827625 1.09087121]
 [0.50990195 0.3        0.         0.24494897 0.50990195 1.08627805]
 [0.64807407 0.33166248 0.24494897 0.         0.64807407 1.16619038]
 [0.14142136 0.60827625 0.50990195 0.64807407 0.         0.6164414 ]
 [0.6164414  1.09087121 1.08627805 1.16619038 0.6164414  0.        ]]
```



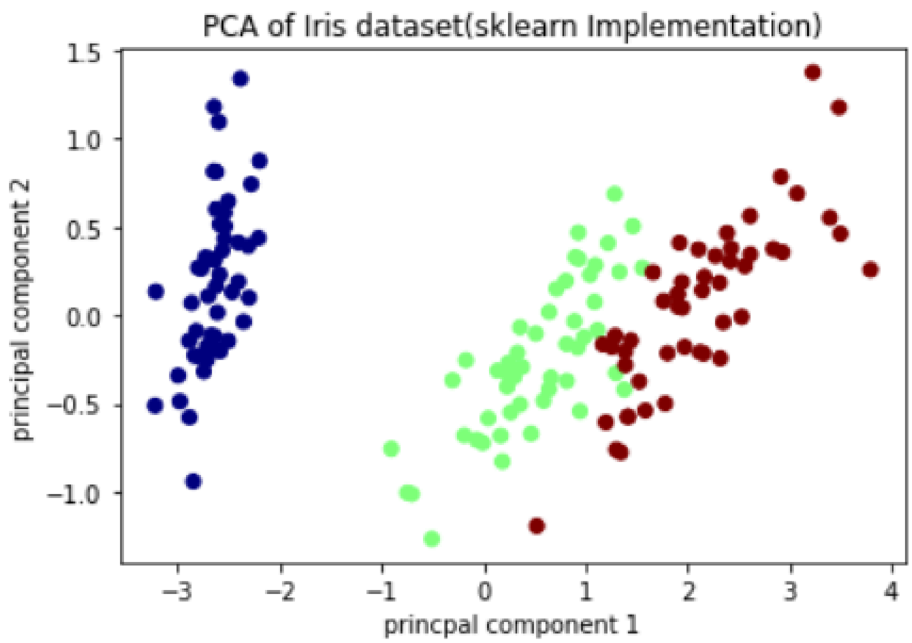


10. Write a program to develop the principal component Analysis(PCA) algorithm.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA as SklearnPCA
x=load_iris().data
y=load_iris().target
PCA=SklearnPCA(n_components=2)
X_projected=pca.fit_transform(x)
print("Shape of data:",x.shape)
print("shape of transformed data:",X_projected.shape)
```

```
pc1=X_projected[:,0]
pc2=X_projected[:,1]
plt.scatter(pc1,pc2,c=y,cmap="jet")
plt.xlabel("princpal component 1")
plt.ylabel("principal component 2")
plt.title("PCA of Iris dataset(sklearn Implementation)")
plt.show()
```

```
Shape of data: (150, 4)
shape of transformed data: (150, 2)
```

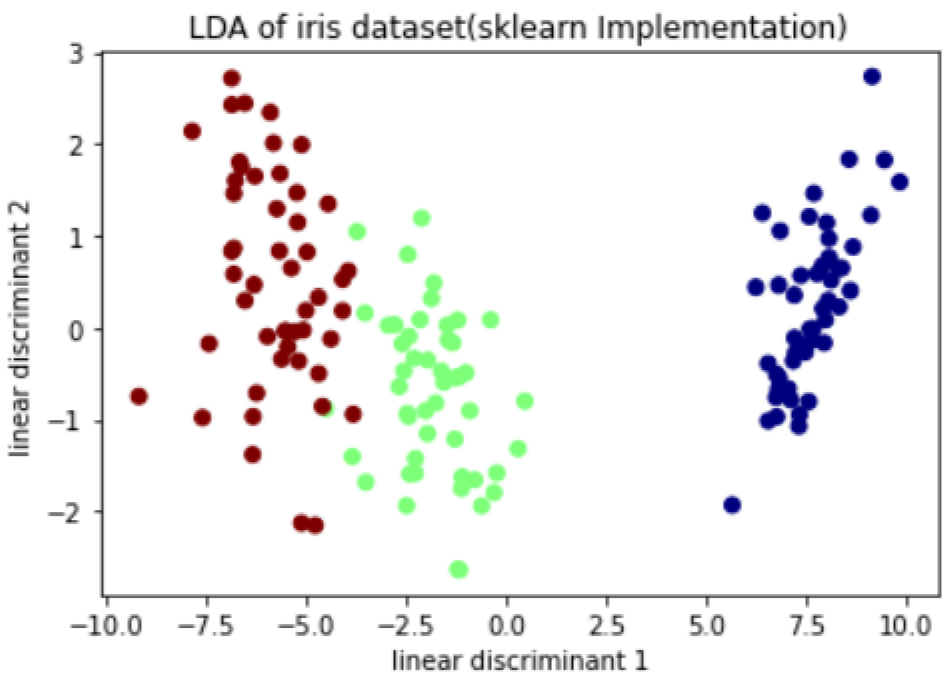


11. Write a program to develop the Linear Discriminant Analysis(LDA) algorithm.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
X=load_iris().data
y=load_iris().target
lda=LinearDiscriminantAnalysis(n_components=2)
X_projected=lda.fit_transform(x,y)
print("shape of data:",X.shape)
print("shape of transformed data:",X_projected.shape)
```

```
ld1=X_projected[:,0]
ld2=X_projected[:,1]
plt.scatter(ld1,ld2,c=y,cmap="jet")
plt.xlabel("linear discriminant 1")
plt.ylabel("linear discriminant 2")
plt.title("LDA of iris dataset(sklearn Implementation)")
plt.show()
```

```
shape of data: (150, 4)
shape of transformed data: (150, 2)
```



12.Write a program to develop simple single layer perceptron to implement AND,OR Boolean functions.

```
import numpy as np
def step_function(X):
    return np.where(X>=0,1,0)
X_and=np.array([[0,0],[0,1],[1,0],[1,1]])
Y_and=np.array([[0],[0],[0],[1]])
X_or=np.array([[0,0],[0,1],[1,0],[1,1]])
Y_or=np.array([[0],[1],[1],[1]])
class perceptron:
    def __init__(self,input_size,learning_rate=0.1,epochs=1000):
```

```python
        self.weights=np.zeros((input_size,1))
        self.bias=0
        self.learning_rate=learning_rate
        self.epochs=epochs
    def train(self,X,Y):
        for _ in range(self.epochs):
            for inputs,label in zip(X,Y):
                inputs=inputs.reshape(-1,1)
                linear_output=np.dot(inputs.T,self.weights)+self.bias
                prediction=step_function(linear_output)
                error=label-prediction
                self.weights+=self.learning_rate*error*inputs
                self.bias+=self.learning_rate*error
    def predict(self,X):
        linear_output=np.dot(X,self.weights)+self.bias
        return step_function(linear_output)
perceptron_and=perceptron(input_size=2)
perceptron_and.train(X_and,Y_and)
perceptron_or=perceptron(input_size=2)
perceptron_or.train(X_or,Y_or)
print("AND function predictions:")
print(perceptron_and.predict(X_and))
print("OR function predictions:")
print(perceptron_or.predict(X_or))
and_test_input=np.array([[1,0]])
print("\n AND function Predition for input [1,0]")
print(perceptron_and.predict(and_test_input))
or_test_input=np.array([[1,0]])
print("\n or function Predition for input [1,0]")
print(perceptron_or.predict(or_test_input))
```

```
AND function predictions:
[[0]
 [0]
 [0]
 [1]]
OR function predictions:
[[0]
 [1]
 [1]
 [1]]

 AND function Predition for input [1,0]
[[0]]

 or function Predition for input [1,0]
[[1]]
```