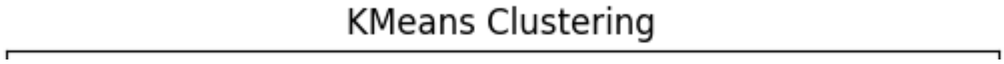
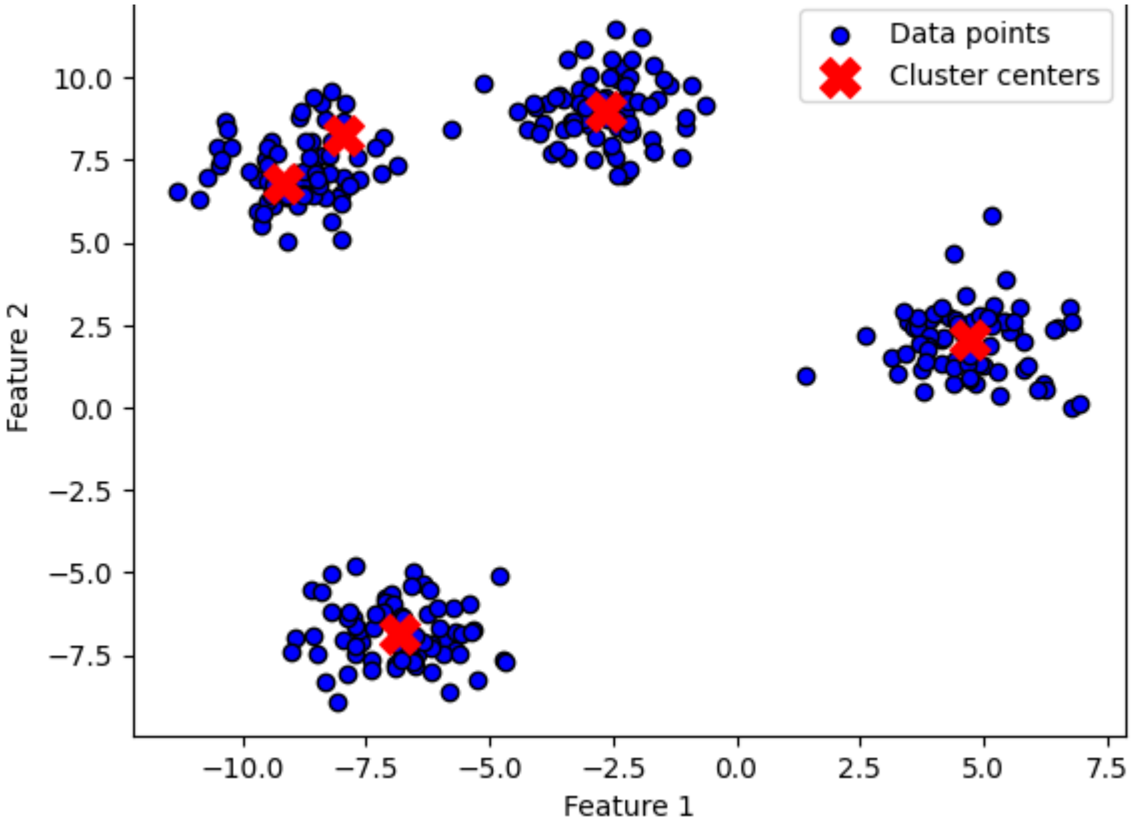
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

# Function to generate a sample dataset
def generate_dataset():
    # Generating a random dataset with 4 clusters
    data, labels = make_blobs(n_samples=300, centers=4, random_state=42)
    return data

# Function to plot the dataset and cluster centers
def plot_clusters(data, centroids):
    plt.scatter(data[:, 0], data[:, 1], c='blue', marker='o', edgecolors='k', lab
    plt.scatter(centroids[:, 0], centroids[:, 1], c='red', marker='X', s=200, lab
    plt.title('KMeans Clustering')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.legend()
    plt.show()

# Function to perform KMeans clustering
def kmeans_clustering(data, num_clusters):
    kmeans = KMeans(n_clusters=num_clusters, random_state=42)
    kmeans.fit(data)
    centroids = kmeans.cluster_centers_
    labels = kmeans.labels_

    return centroids, labels

# Main function
def main():
    # Generate a sample dataset
    data = generate_dataset()

    # Input the number of clusters
    num_clusters = int(input("Enter the number of clusters: "))

    # Perform KMeans clustering
    centroids, labels = kmeans_clustering(data, num_clusters)

    # Plot the results
    plot_clusters(data, centroids)

if __name__ == "__main__":
    main()
```

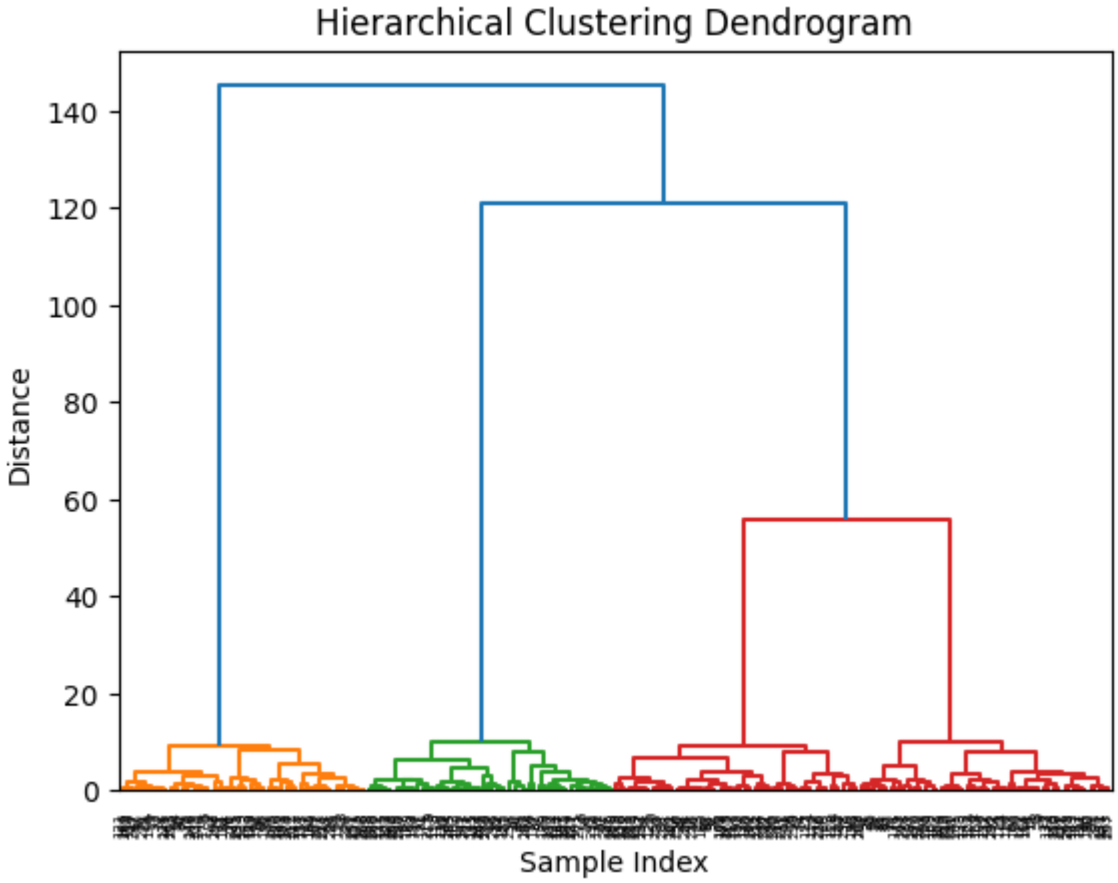
```
Enter the number of clusters: 5
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Fut
  warnings.warn(
```





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

# Function to generate a sample dataset
def generate_dataset():
    # Generating a random dataset with 4 clusters
    data, labels = make_blobs(n_samples=300, centers=4, random_state=42)
    return data

# Function to plot the dendrogram
def plot_dendrogram(data):
    # Create linkage matrix
    linkage_matrix = linkage(data, 'ward')

    # Plot the dendrogram
    dendrogram(linkage_matrix)
    plt.title('Hierarchical Clustering Dendrogram')
    plt.xlabel('Sample Index')
    plt.ylabel('Distance')
    plt.show()

# Function to perform Hierarchical clustering
def hierarchical_clustering(data, num_clusters):
    hierarchical = AgglomerativeClustering(n_clusters=num_clusters)
    labels = hierarchical.fit_predict(data)

    return labels
```

```
# Main function
def main():
    # Generate a sample dataset
    data = generate_dataset()

    # Plot the dendrogram
    plot_dendrogram(data)

    # Input the number of clusters
    num_clusters = int(input("Enter the number of clusters: "))

    # Perform Hierarchical clustering
    labels = hierarchical_clustering(data, num_clusters)

    # Plot the results
    plt.scatter(data[:, 0], data[:, 1], c=labels, cmap='viridis', marker='o', e
    plt.title('Hierarchical Clustering')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.show()

if __name__ == "__main__":
    main()
```



```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-3-b33ccda36116> in <cell line: 53>()
     52
     53 if __name__ == "__main__":
---> 54     main()
```

```
---> 34      main()

        2 frames
/usr/local/lib/python3.10/dist-packages/ipykernel/kernelbase.py in
_input_request(self, prompt, ident, parent, password)
    893             except KeyboardInterrupt:
    894                 # re-raise KeyboardInterrupt, to truncate traceback
--> 895                 raise KeyboardInterrupt("Interrupted by user") from
None
    896             except Exception as e:
    897                 self.log.warning("Invalid Message:", exc_info=True)

KeyboardInterrupt: Interrupted by user
```

SEARCH STACK OVERFLOW

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, export_text
from sklearn.metrics import accuracy_score, classification_report, confusion_ma
from sklearn.datasets import make_classification

# Function to generate a sample dataset
def generate_dataset():
    # Generating a random dataset with 2 classes
    data, labels = make_classification(n_samples=300, n_features=4, n_informati
                                       n_redundant=0, n_clusters_per_class=1, r
    return data, labels

# Function to split the dataset into training and testing sets
def split_dataset(data, labels):
    return train_test_split(data, labels, test_size=0.2, random_state=42)

# Function to train a Decision Tree model
def train_decision_tree(train_data, train_labels):
    dt_classifier = DecisionTreeClassifier(random_state=42)
    dt_classifier.fit(train_data, train_labels)
    return dt_classifier

# Function to evaluate the Decision Tree model
def evaluate_decision_tree(model, test_data, test_labels):
    predictions = model.predict(test_data)
    accuracy = accuracy_score(test_labels, predictions)

    print("Accuracy:", accuracy)
    print("\nConfusion Matrix:\n", confusion_matrix(test_labels, predictions))
    print("\nClassification Report:\n", classification_report(test_labels, pred

# Main function
def main():
    # Generate a sample dataset
    data, labels = generate_dataset()

    # Split the dataset
    train_data, test_data, train_labels, test_labels = split_dataset(data, labe
    train_data, test_data, train_labels, test_labels = split_dataset(data, labe

    # Train a Decision Tree model
    dt_model = train_decision_tree(train_data, train_labels)

    # Evaluate the model
    evaluate_decision_tree(dt_model, test_data, test_labels)

    # Display the Decision Tree rules
    tree_rules = export_text(dt_model, feature_names=list(range(train_data.shap
    print("\nDecision Tree Rules:\n", tree_rules)

if __name__ == "__main__":
    main()
```

```
Accuracy: 0.9166666666666666

Confusion Matrix:
 [[35  3]
 [ 2 20]]

Classification Report:
               precision    recall  f1-score   support

           0       0.95      0.92      0.93        38
           1       0.87      0.91      0.89        22

    accuracy                           0.92        60
   macro avg       0.91      0.92      0.91        60
weighted avg       0.92      0.92      0.92        60


Decision Tree Rules:
 |--- 3 <= 0.20
|   |--- 0 <= 1.73
|   |   |--- 1 <= -2.14
|   |   |   |--- class: 1
|   |   |--- 1 >  -2.14
|   |   |   |--- 3 <= -0.40
|   |   |   |   |--- 2 <= 0.38
|   |   |   |   |   |--- class: 0
|   |   |   |   |--- 2 >  0.38
|   |   |   |   |   |--- 2 <= 0.40
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- 2 >  0.40
|   |   |   |   |   |   |--- class: 0
|   |   |   |--- 3 >  -0.40
|   |   |   |   |--- 0 <= 1.21
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- 0 >  1.21
|   |   |   |   |   |--- 2 <= -1.35
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- 2 >  -1.35
|   |   |   |   |   |   |--- class: 0
|   |--- 0 >  1.73
|   |   |--- class: 1
|--- 3 >  0.20
|   |--- 3 <= 0.34
|   |--- 3 <= 0.34
|   |   |--- 3 <= 0.30
|   |   |   |--- class: 1
|   |   |--- 3 >  0.30
|   |   |   |--- class: 0
|   |--- 3 >  0.34
|   |   |--- 1 <= -1.47
|   |   |   |--- 1 <= -1.53
|   |   |   |   |--- class: 1
|   |   |   |--- 1 >  -1.53
|   |   |   |   |--- class: 0
|   |   |--- 1 >  -1.47
|   |   |   |--- class: 1
```