Lab 6: Clustering

Name: **Krish Agarwal**
Reg No: **21112016**
Class: **4BSc DS A**

Objective :

1. Download the "Car Evaluation" Dataset from UCI Repository (https://archive.ics.uci.edu/ml/datasets/Car+Evaluation (https://archive.ics.uci.edu/ml/datasets/Car+Evaluation)). Remove the target 'Class Values' from the dataset while applying clustering algorithms.
2. Find the optimal number of clusters using Elbow and Silhouette Method.
3. Compare KMeans and Agglomerative Clustering methods for clustering the instances in the above dataset. Validate the optimal number of clusters found out in the previous question. **Hint**: Even if the algorithm does not require labels, for cross-checking of clustering values, you may use the labels.
4. Find what hyperparameters were suitable in KMeans (n_clusters, max_iter, init, algorithm)
5. Find what hyperparameters were suitable in Agglomerative Clustering (n_clusters, metric, linkage)
6. Plot Hierarchical Clustering (Dendrogram).
7. Compare the better clustering algorithm with any classification algorithm, and write your notes on the same.

Probelm Definition :

Train 2 clustering models - **KMeans** and **Agglomerative** - for the given Car Evaluation dataset and compare both the models. After the the comparison, validate the optimal number of clusters.

Observations :

- Via the elbow method, the optimal value of K is 4 as the graph bends at 4.
- Via the silhouette method, the optimal value of K is 4 as the maxima of the graph is at 4.
- The Silhouette Score of the KMeans model is 0.15756037323919975.
- The Silhouette Score of the Agglomerative model is 0.11147083427605417.
- It is noticed that the silhouette_scores are highest (0.172048) when there are 8 clusters, init = 'k-means++' and max iteration is either 100 or 200.
- It is observed that the silhouette score is the highest when the number of clusters are 2 and the linkage is average.
- The KMeans algorithm resulted in an optimal number of 8 clusters, while AGNES clustering found 4 clusters. KMeans partitions the data, while AGNES merges clusters hierarchically. The centroid distance metric used in KMeans is easier to comprehend, whereas reading dendrograms in AGNES requires experience and expertise. With KMeans, we had to specify the number of clusters, but in AGNES, there were no pre-determined cluster numbers.

References :

1. StackOverflow
2. GeekforGeeks
3. TutorialsPoint
4. Medium
5. W3School

Completion Status:

Question Number	Status
1	Completed
2	Completed
3	Completed
4	Completed
5	Completed
6	Completed
7	Completed

Code :

Q1) Download the "Car Evaluation" Dataset from UCI Repository. Remove the target 'Class Values' from the dataset while applying clustering algorithms.

In [34]:

```
# importing all the necessary libraries / modules
import pandas as pd
import numpy as np
import random
import warnings
warnings.filterwarnings("ignore")
import matplotlib.pyplot as plt
import seaborn as sns
from krishKiLibrary import countUnique
from sklearn.preprocessing import OrdinalEncoder
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster import hierarchy
from sklearn.decomposition import PCA
from scipy.cluster.hierarchy import linkage, dendrogram
from sklearn.metrics import silhouette_samples, silhouette_score
import matplotlib.cm as cm
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
```

In [2]:

```
# --- Imorting the dataframe ---
df = pd.read_csv("D:/Z/Downloads/car.data", names = ["Buying", "Maint", "Doors", "Pe

# --- Removing the target variable from the training dataset ---
df_training = df.drop('Class_Val', axis = 1)
```

In [3]:

```
df
```

Out[3]:

	Buying	Maint	Doors	Persons	Lug_boot	Safety	Class_Val
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc
3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc
...	...	...	...	...	...	...	...
1723	low	low	5more	more	med	med	good
1724	low	low	5more	more	med	high	vgood
1725	low	low	5more	more	big	low	unacc
1726	low	low	5more	more	big	med	good
1727	low	low	5more	more	big	high	vgood

1728 rows × 7 columns

In [4]:

```
# --- Counting the number of unique values in each column ---
for i in (df_training.columns):
    print(i, ": ", countUnique(df_training, df_training[i].unique(), i))
```

```
Buying :  {'vhigh': 432, 'high': 432, 'med': 432, 'low': 432}
Maint :  {'vhigh': 432, 'high': 432, 'med': 432, 'low': 432}
Doors :  {'2': 432, '3': 432, '4': 432, '5more': 432}
Persons :  {'2': 576, '4': 576, 'more': 576}
Lug_boot :  {'small': 576, 'med': 576, 'big': 576}
Safety :  {'low': 576, 'med': 576, 'high': 576}
```

In [5]:

```
# --- Ordinally Encoding the values of the columns ---
buying_mapper = {'vhigh': 4, 'high': 3, 'med': 2, 'low': 1}
maint_mapper = {'vhigh': 4, 'high': 3, 'med': 2, 'low': 1}
doors_mapper = {2: 2, 3: 3, 4: 4, '5more': 5}
persons_mapper = {2: 1, 4: 2, 'more': 3}
lug_boot_mapper = {'big': 3, 'med': 2, 'small': 1}
safety_mapper = {'high': 3, 'med': 2, 'low': 1}
```

In [6]:

```
df_training["Buying"] = df_training["Buying"].replace(buying_mapper)
df_training["Maint"] = df_training["Maint"].replace(maint_mapper)
df_training["Doors"] = df_training["Doors"].replace(doors_mapper)
df_training["Persons"] = df_training["Persons"].replace(persons_mapper)
df_training["Lug_boot"] = df_training["Lug_boot"].replace(lug_boot_mapper)
df_training["Safety"] = df_training["Safety"].replace(safety_mapper)
```

In [7]:

```
df_training
```

Out[7]:

	Buying	Maint	Doors	Persons	Lug_boot	Safety
0	4	4	2	2	1	1
1	4	4	2	2	1	2
2	4	4	2	2	1	3
3	4	4	2	2	2	1
4	4	4	2	2	2	2
...	...	...	...	...	...	...
1723	1	1	5	3	2	2
1724	1	1	5	3	2	3
1725	1	1	5	3	3	1
1726	1	1	5	3	3	2
1727	1	1	5	3	3	3

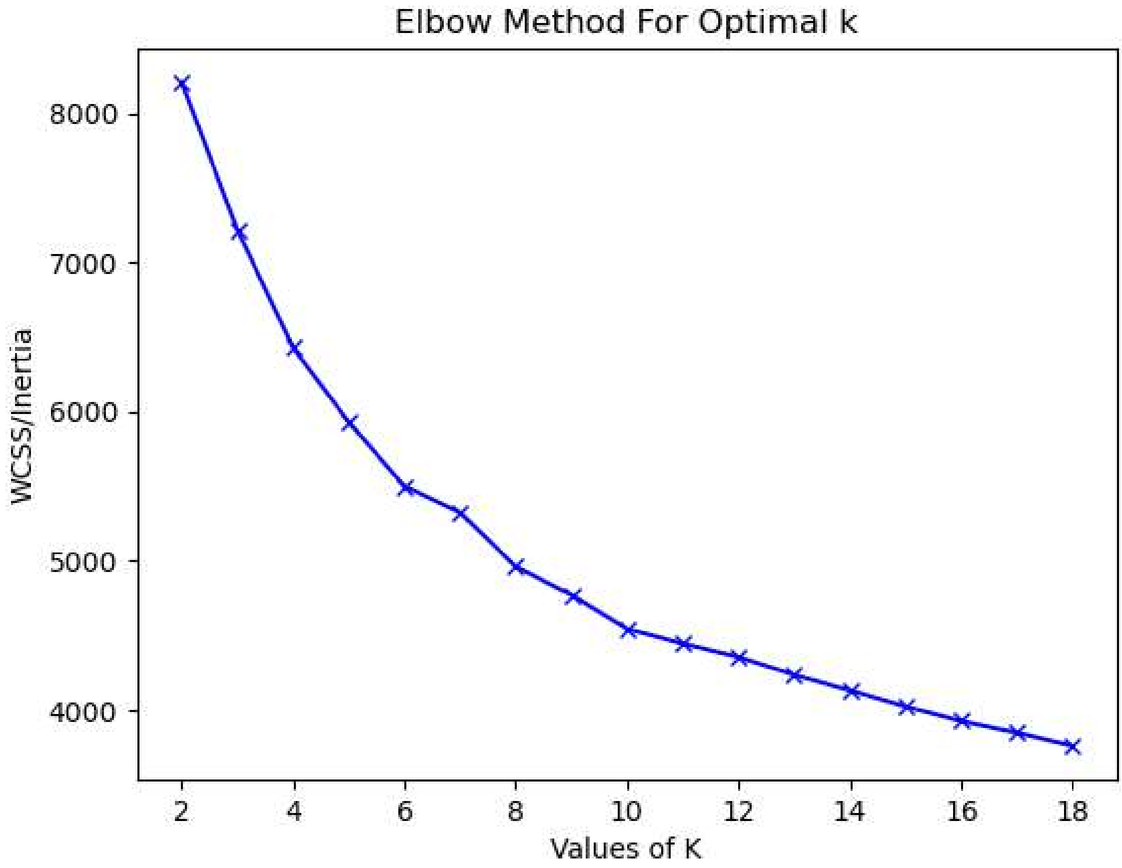
1728 rows × 6 columns

Q2) Find the optimal number of clusters using Elbow and Silhouette Method.

Elbow Method

In [8]:

```python
# --- Plotting the elbow curve ---
wcss = []
K = range(2,19)

for num_clusters in K :
    kmeans = KMeans(n_clusters = num_clusters, random_state = 42, n_init =1)
    kmeans.fit(df_training)
    wcss.append(kmeans.inertia_) # appending the wcss values into the list

plt.plot(K, wcss, 'bx-') # plotting the wcss values w.r.t. K
plt.xlabel('Values of K')
plt.ylabel('WCSS/Inertia')
plt.title('Elbow Method For Optimal k')
plt.show()
```

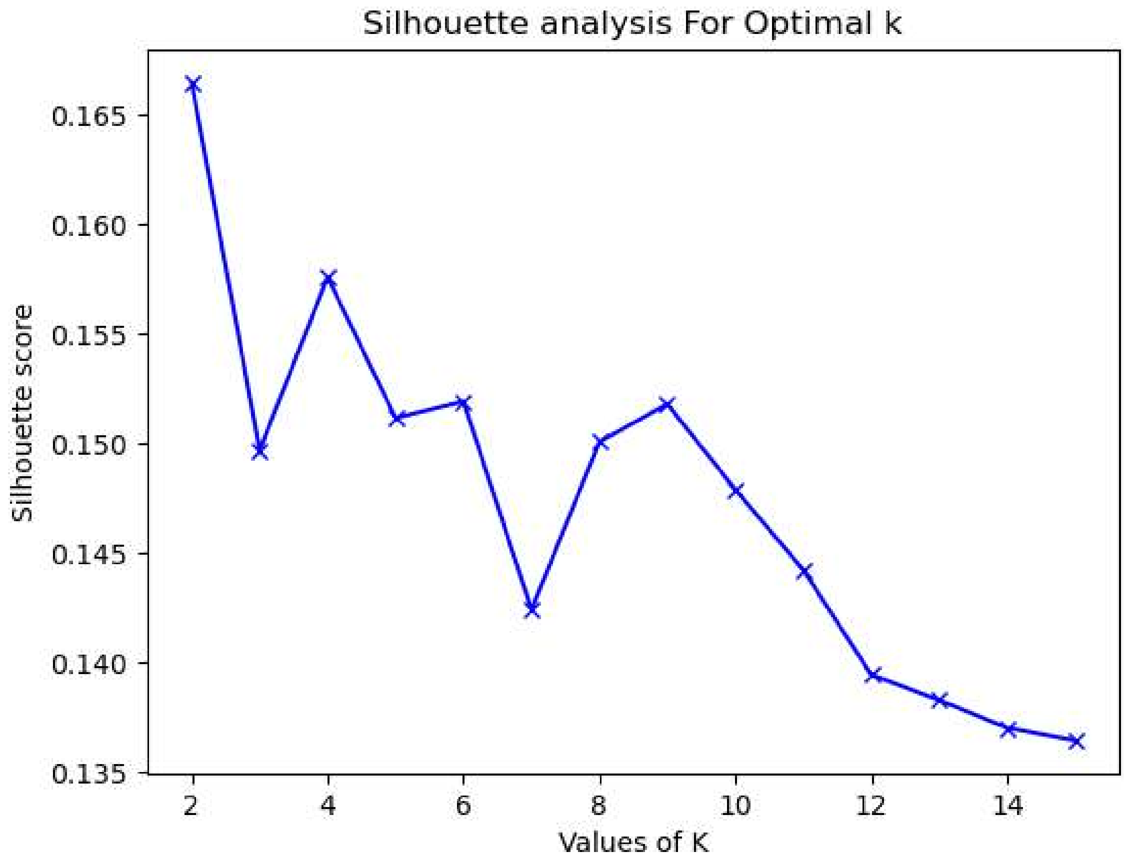


Inference: The optimal value of K is 4 as the graph bends there.

Silhouette Method

In [9]:

```
range_n_clusters = range(2, 16)
silhouette_avg = []

for num_clusters in range_n_clusters:
    kmeans = KMeans(n_clusters = num_clusters, random_state = 42, n_init = 1) # init
    kmeans.fit(df_training)
    cluster_labels = kmeans.labels_
    silhouette_avg.append(silhouette_score(df_training, cluster_labels))

plt.plot(range_n_clusters, silhouette_avg, 'bx-') # silhouette score
plt.xlabel('Values of K')
plt.ylabel('Silhouette score')
plt.title('Silhouette analysis For Optimal k')
plt.show()
```



Inference: The optimal value of K is 4 as the maxima of the graph is 4.

Q3) Agglomerative Clustering vs KMeans Clustering

KMeans Clustering

In [10]:

```
#Initialize the class object
kmeans = KMeans(n_clusters = 4).fit(df_training)
label = kmeans.predict(df_training)

# Calculating the aggregzzated silhouette score
silhouette_score_average = silhouette_score(df_training, label)
print('Silhouette Score (Accuracy): {}'.format(silhouette_score_average))
```

```
Silhouette Score (Accuracy): 0.1564952907335478
```

An Attempt to Plotting

In [11]:

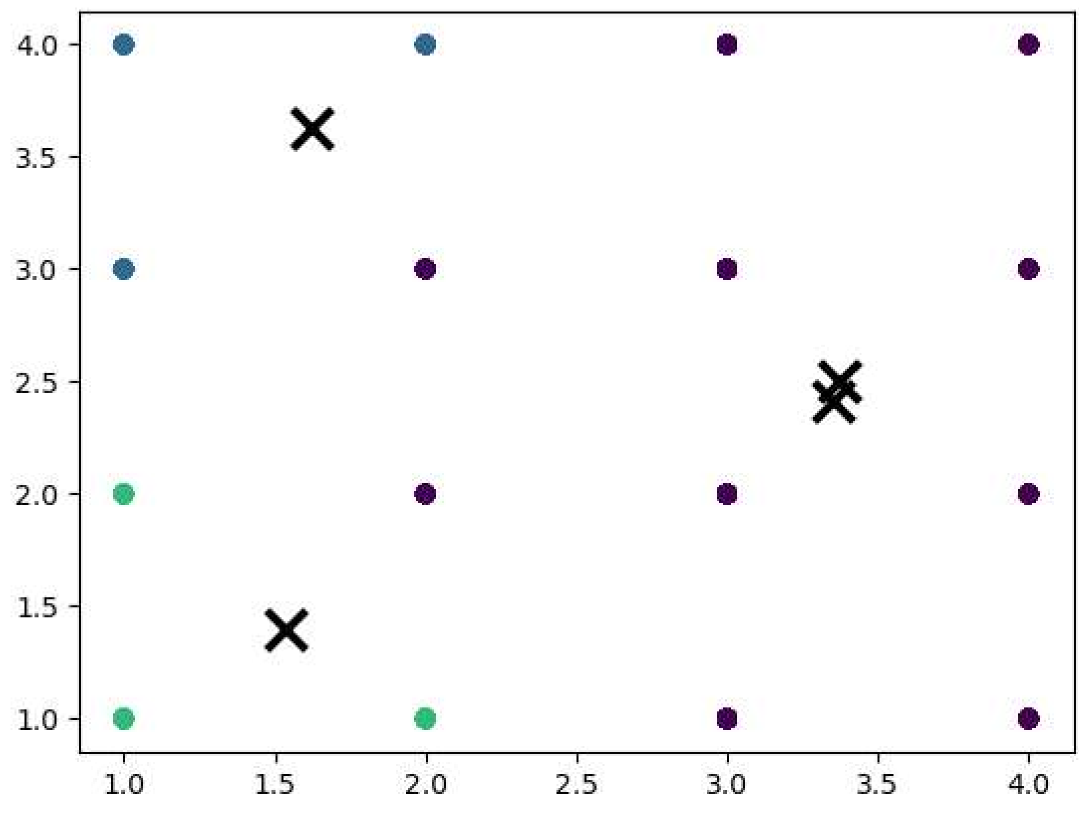
```
# Storing all the clsuters in seperate variables
label_0 = df_training[label == 0]
label_1 = df_training[label == 1]
label_2 = df_training[label == 2]
label_3 = df_training[label == 3]
```

In [12]:

```
# Plotting the clusters
plt.scatter(df_training.iloc[:, 0], df_training.iloc[:, 1], c = label, cmap = 'viric

# Plot the cluster centers as black dots
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], marker = '
```

Out[12]:

```
<matplotlib.collections.PathCollection at 0x2f228455fd0>
```

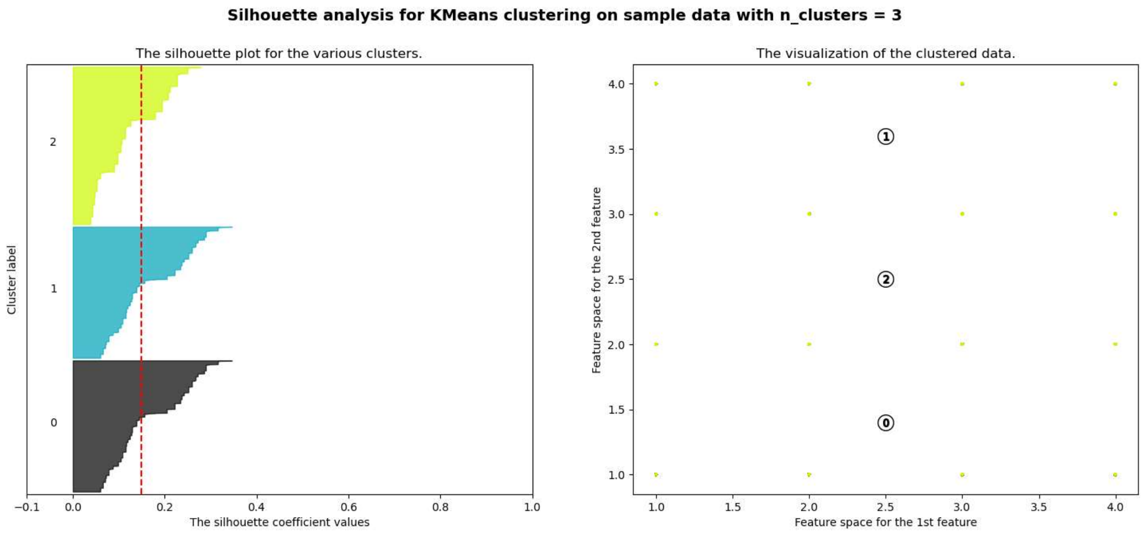


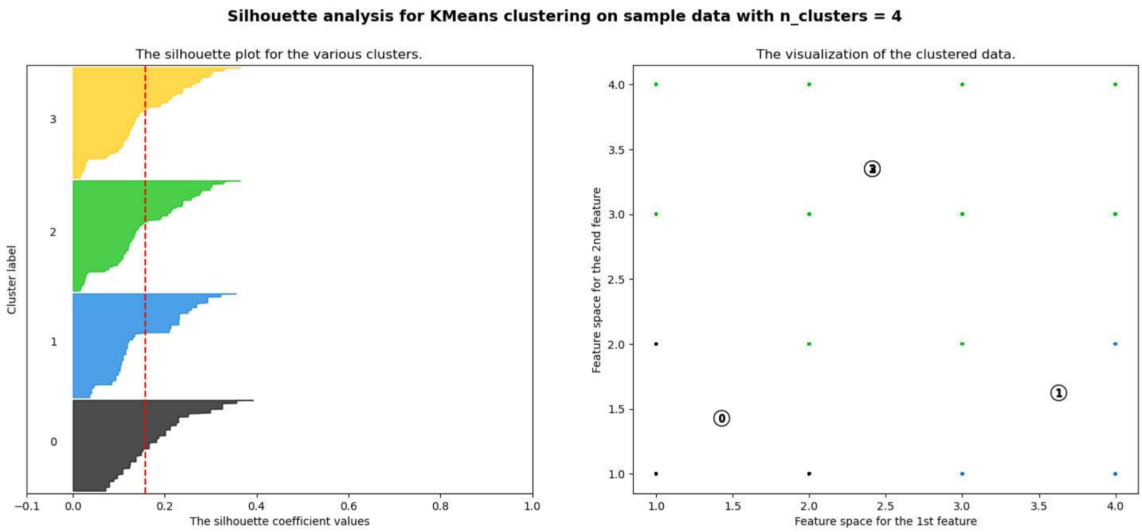
In [13]:

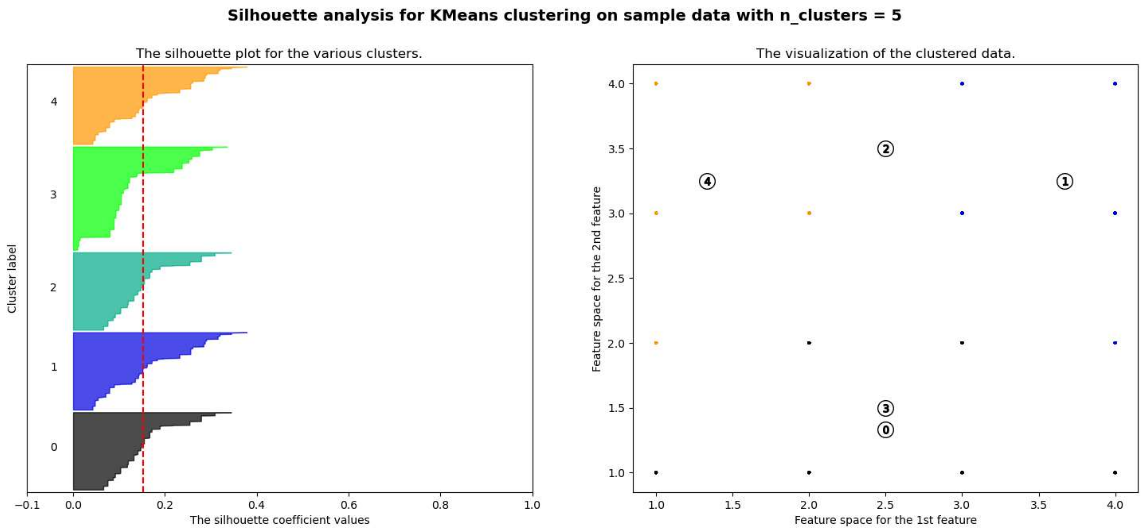
```
# Plotting KMEANS for various values of K
range_n_clusters = range(3,13)

for n_clusters in range_n_clusters:

    # Create a subplot with 1 row and 2 columns
    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(18, 7)

    # The 1st subplot is the silhouette plot
    # The silhouette coefficient can range from -1, 1 but in this example all
    # lie within [-0.1, 1]
    ax1.set_xlim([-0.1, 1])

    # The (n_clusters+1)*10 is for inserting blank space between silhouette
    # plots of individual clusters, to demarcate them clearly.
    ax1.set_ylim([0, len(df_training) + (n_clusters + 1) * 10])

    # Initialize the clusterer with n_clusters value and a random generator
    # seed of 10 for reproducibility.
    clusterer = KMeans(n_clusters=n_clusters, random_state=10)
    cluster_labels = clusterer.fit_predict(df_training)

    # The silhouette_score gives the average value for all the samples.
    # This gives a perspective into the density and separation of the formed
    # clusters
    silhouette_avg = silhouette_score(df_training, cluster_labels)
    print("For n_clusters =", n_clusters, "The average silhouette_score is :", silhc

    # Compute the silhouette scores for each sample
    sample_silhouette_values = silhouette_samples(df_training, cluster_labels)

    y_lower = 10

    for i in range(n_clusters):

        # Aggregate the silhouette scores for samples belonging to
        # cluster i, and sort them
        ith_cluster_silhouette_values = \
        sample_silhouette_values[cluster_labels == i]
        ith_cluster_silhouette_values.sort()
        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i
        color = cm.nipy_spectral(float(i) / n_clusters)
        ax1.fill_betweenx(np.arange(y_lower, y_upper),
        0, ith_cluster_silhouette_values,
        facecolor=color, edgecolor=color, alpha=0.7)

        # Label the silhouette plots with their cluster numbers at the middle
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

        # Compute the new y_lower for next plot
        y_lower = y_upper + 10 # 10 for the 0 samples

    ax1.set_title("The silhouette plot for the various clusters.")
    ax1.set_xlabel("The silhouette coefficient values")
    ax1.set_ylabel("Cluster label")

    # The vertical line for average silhouette score of all the values
```

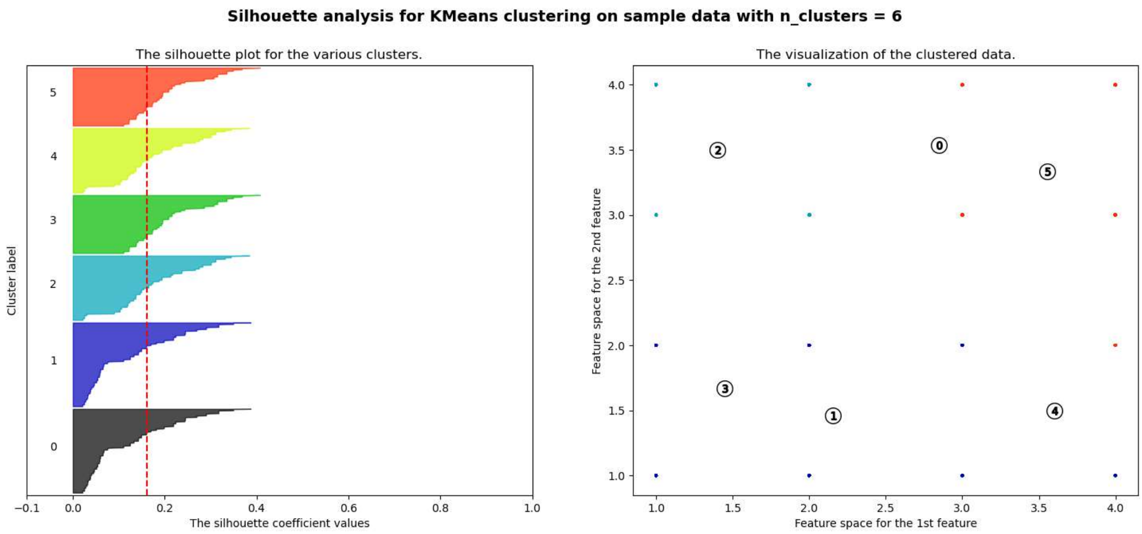
```
    ax1.axvline(x=silhouette_avg, color="red", linestyle="--")
    ax1.set_yticks([]) # Clear the yaxis labels / ticks
    ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])

    # 2nd Plot showing the actual clusters formed
    colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
    ax2.scatter(df_training.iloc[:, 0], df_training.iloc[:, 1], marker='.', s=30, lw
    c=colors, edgecolor='k')

    # Labeling the clusters
    centers = clusterer.cluster_centers_

    # Draw white circles at cluster centers
    ax2.scatter(centers[:, 0], centers[:, 1], marker='o',
    c="white", alpha=1, s=200, edgecolor='k')

    for i, c in enumerate(centers):
        ax2.scatter(c[0], c[1], marker='$%d$' % i, alpha=1,
        s=50, edgecolor='k')
        ax2.set_title("The visualization of the clustered data.")
        ax2.set_xlabel("Feature space for the 1st feature")
        ax2.set_ylabel("Feature space for the 2nd feature")
        plt.suptitle(("Silhouette analysis for KMeans clustering on sample data "
        "with n_clusters = %d" % n_clusters),
        fontsize=14, fontweight='bold')
plt.show()
```

```
For n_clusters = 3 The average silhouette_score is : 0.1496903095639953
For n_clusters = 4 The average silhouette_score is : 0.15756037323919983
For n_clusters = 5 The average silhouette_score is : 0.15168620164271934
For n_clusters = 6 The average silhouette_score is : 0.16154197589141794
For n_clusters = 7 The average silhouette_score is : 0.15929390277317174
For n_clusters = 8 The average silhouette_score is : 0.17204794782267452
For n_clusters = 9 The average silhouette_score is : 0.1582127116740343
For n_clusters = 10 The average silhouette_score is : 0.14964182627756134
For n_clusters = 11 The average silhouette_score is : 0.14260146059974588
For n_clusters = 12 The average silhouette_score is : 0.14131637907315775
```

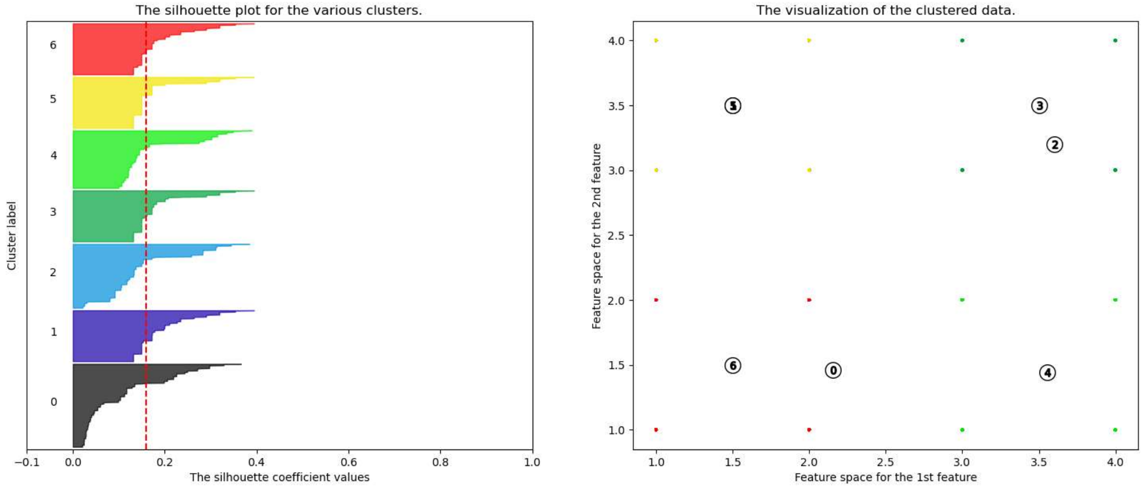



Silhouette analysis for KMeans clustering on sample data with n_clusters = 4
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
The visualization of the clustered data.
Feature space for the 1st feature
Feature space for the 2nd feature

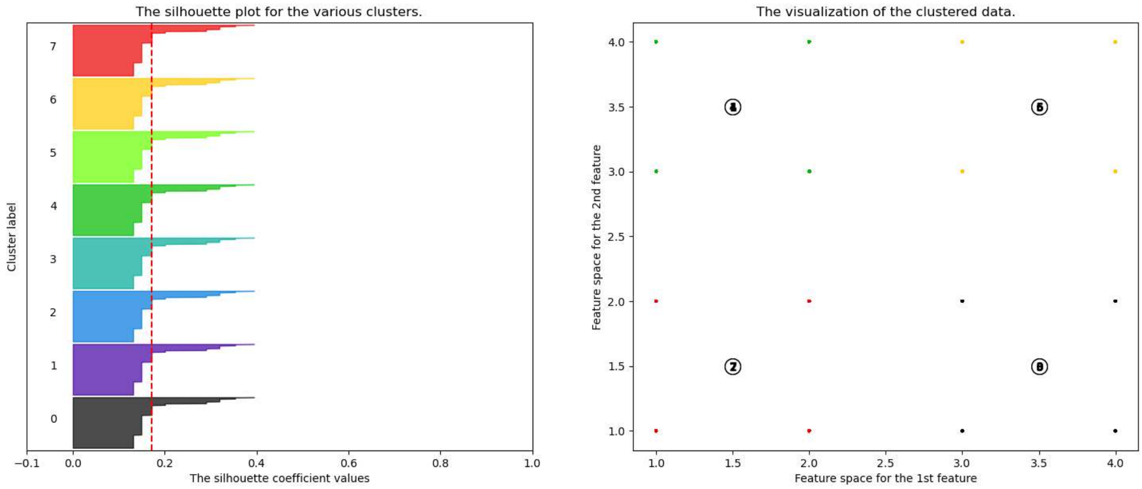

Silhouette analysis for KMeans clustering on sample data with n_clusters = 5
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
The visualization of the clustered data.
Feature space for the 1st feature
Feature space for the 2nd feature


Silhouette analysis for KMeans clustering on sample data with n_clusters = 6
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
The visualization of the clustered data.
Feature space for the 1st feature
Feature space for the 2nd feature

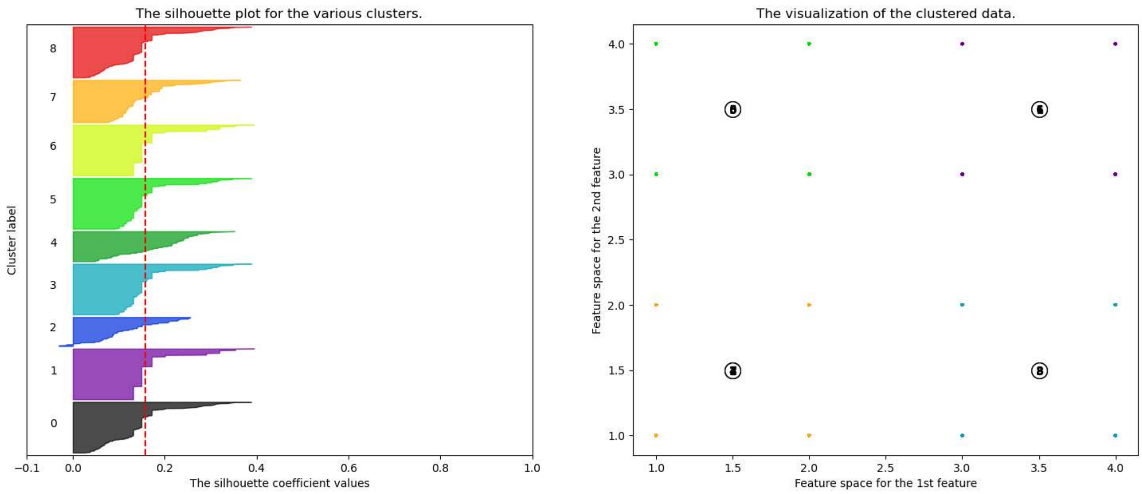
Silhouette analysis for KMeans clustering on sample data with n_clusters = 7

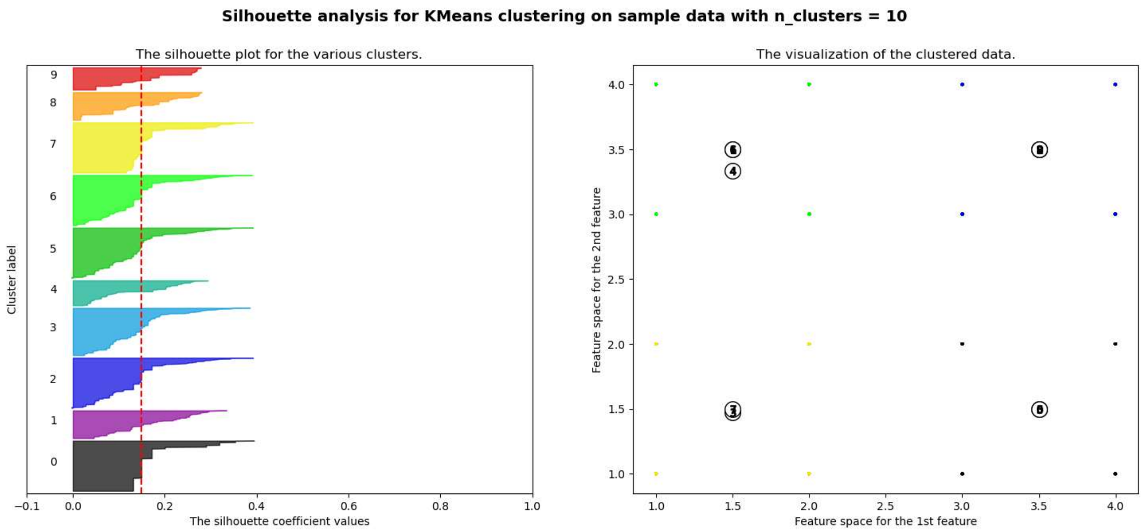


Silhouette analysis for KMeans clustering on sample data with n_clusters = 8

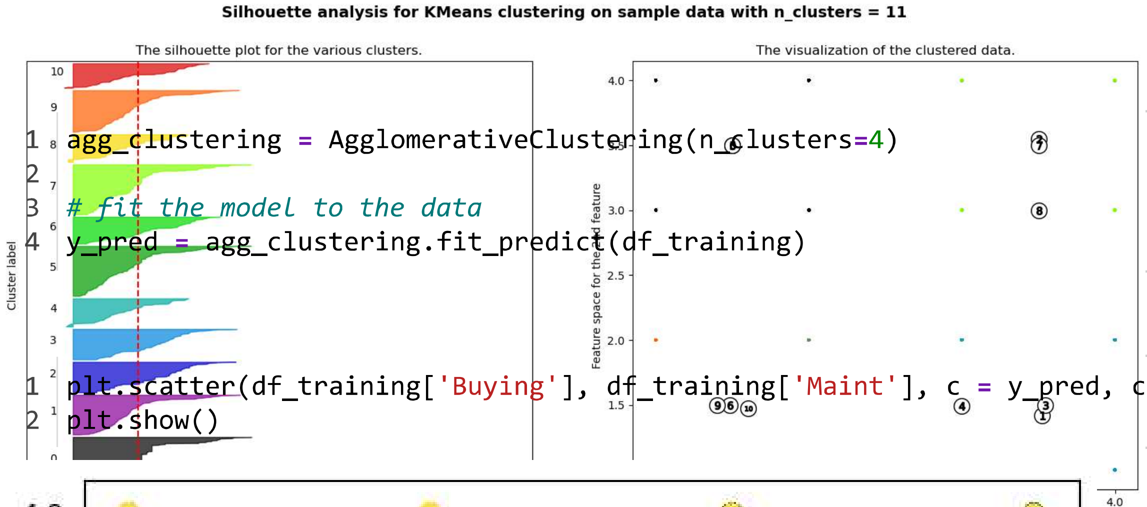


Silhouette analysis for KMeans clustering on sample data with n_clusters = 9



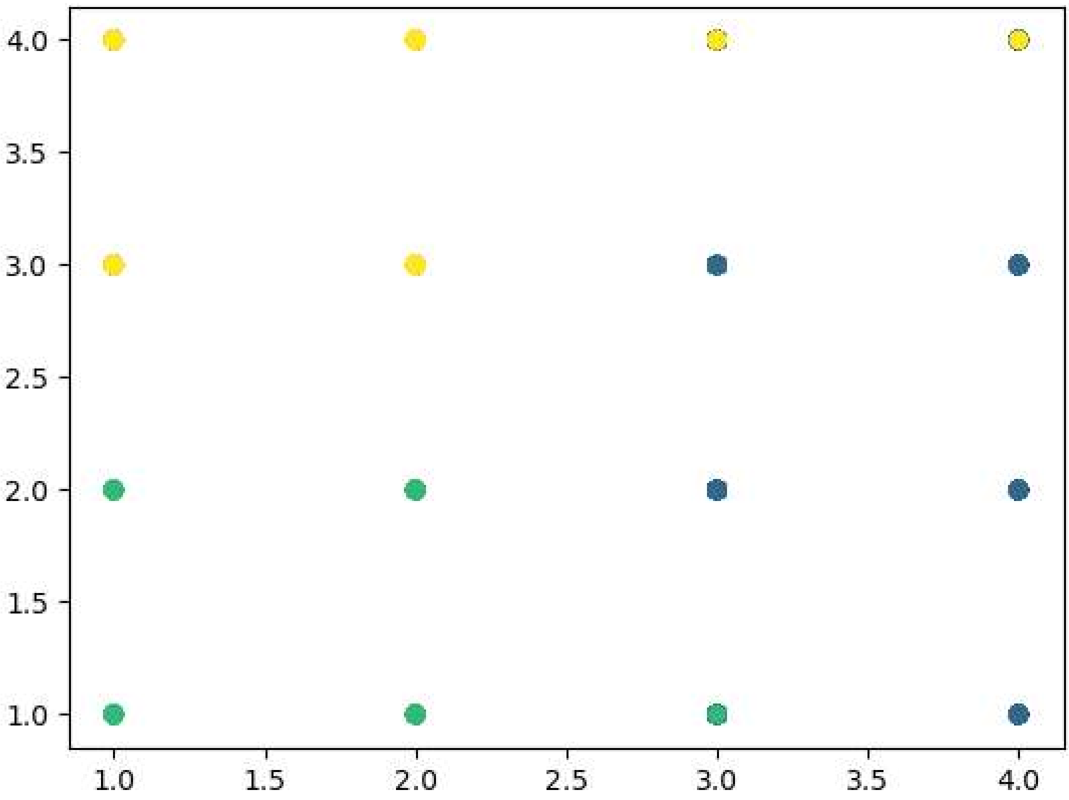


Agglomerative Clustering




```python
agg_clustering = AgglomerativeClustering(n_clusters=4)

# fit the model to the data
y_pred = agg_clustering.fit_predict(df_training)
```

```python
plt.scatter(df_training['Buying'], df_training['Maint'], c = y_pred, cmap='viridis')
plt.show()
```



In [16]:

```
print("Silhouette Score (Accuracy): ", silhouette_score(df_training, y_pred))
```

```
Silhouette Score (Accuracy):  0.111470834276054l7
```

Q4) Suitability of Hyperparameters in KMeans Clustering

In [17]:

```
# Setting up hyperparameters
n_clusters_ = range(2, 16)
init_ = ['k-means++', 'random']
max_iter_ = [100, 200, 300, 400, 500]
random_state_ = [234, 325, 98, 235, 324, 3]
```

In [18]:

```
# Creating a dataframe to store all the possible combinations
df_ = pd.DataFrame(columns = ['Algorithm', 'init', 'n_clusters', 'max_iter', 'random

# Calculating all the possible silhoutte scores for the above set hyper parameters
for i in init_:
    for j in max_iter_:
        for k in random_state_:
            for l in n_clusters_:
                kmeans = KMeans(init = i, max_iter = j, random_state = k, n_clusters
                pred = kmeans.predict(df_training)
                ss = silhouette_score(df_training, pred)

                dict_ = {}
                dict_['Algorithm'] = "KMeans"
                dict_['n_clusters'] = l
                dict_['init'] = i
                dict_['random_state'] = k
                dict_['max_iter'] = j
                dict_['silhouette_score'] = ss

                df_ = df_.append(dict_, ignore_index = True)
```

In [19]:

```
df_
```

Out[19]:

	Algorithm	init	n_clusters	max_iter	random_state	silhouette_score
0	KMeans	k-means++	2	100	234	0.166422
1	KMeans	k-means++	3	100	234	0.149690
2	KMeans	k-means++	4	100	234	0.161509
3	KMeans	k-means++	5	100	234	0.151145
4	KMeans	k-means++	6	100	234	0.161870
...	...	...	...	...	...	...
835	KMeans	random	11	500	3	0.141286
836	KMeans	random	12	500	3	0.136150
837	KMeans	random	13	500	3	0.138239
838	KMeans	random	14	500	3	0.133412
839	KMeans	random	15	500	3	0.137375

840 rows × 6 columns

In [20]:

```
# Following are 10 instances with the largest silhouette scores
df_.iloc[df_['silhouette_score'].nlargest(10).index]
```

Out[20]:

	Algorithm	init	n_clusters	max_iter	random_state	silhouette_score
6	KMeans	k-means++	8	100	234	0.172048
34	KMeans	k-means++	8	100	98	0.172048
48	KMeans	k-means++	8	100	235	0.172048
62	KMeans	k-means++	8	100	324	0.172048
76	KMeans	k-means++	8	100	3	0.172048
90	KMeans	k-means++	8	200	234	0.172048
118	KMeans	k-means++	8	200	98	0.172048
132	KMeans	k-means++	8	200	235	0.172048
146	KMeans	k-means++	8	200	324	0.172048
160	KMeans	k-means++	8	200	3	0.172048

Inference: It is noticed that the silhouette_scores are highest (0.172048) when there are 8 clusters, init = 'k-means++' and max iteration is either 100 or 200.

Q5) Suitability of Hyperparameters in Agglomerative Clustering

In [21]:

```
# Setting up hyperparameters
n_clusters_ = range(2, 16)
linkage_ = ['ward', 'ward', 'complete', 'average', 'single']
random_state_ = [234, 325, 98, 235, 324, 3]
```

In [22]:

```
# Creating a dataframe to store all the possible combinations
df__ = pd.DataFrame(columns = ['Algorithm', 'linkage', 'n_clusters', 'silhouette_sco

# Calculating all the possible silhoutte scores for the above set hyper parameters
for i in linkage_:
    for l in n_clusters_:
        agg_clustering = AgglomerativeClustering(n_clusters = l, linkage = i)
        pred = agg_clustering.fit_predict(df_training)
        ss = silhouette_score(df_training, pred)

        dict_ = {}
        dict_['Algorithm'] = "Agglomerative Clustering"
        dict_['n_clusters'] = l
        dict_['linkage'] = i
        dict_['silhouette_score'] = ss

        df__ = df__.append(dict_, ignore_index = True)
```

In [23]:

```
df__
```

Out[23]:

	Algorithm	linkage	n_clusters	silhouette_score
0	Agglomerative Clustering	ward	2	0.116495
1	Agglomerative Clustering	ward	3	0.115315
2	Agglomerative Clustering	ward	4	0.111471
3	Agglomerative Clustering	ward	5	0.110009
4	Agglomerative Clustering	ward	6	0.104988
...	...	...	...	...
65	Agglomerative Clustering	single	11	-0.043878
66	Agglomerative Clustering	single	12	-0.045161
67	Agglomerative Clustering	single	13	-0.046390
68	Agglomerative Clustering	single	14	-0.047671
69	Agglomerative Clustering	single	15	-0.049013

70 rows × 4 columns

In [24]:

```python
# Following are 10 instances with the largest silhouette scores
df__.iloc[df__['silhouette_score'].nlargest(10).index]
```

Out[24]:

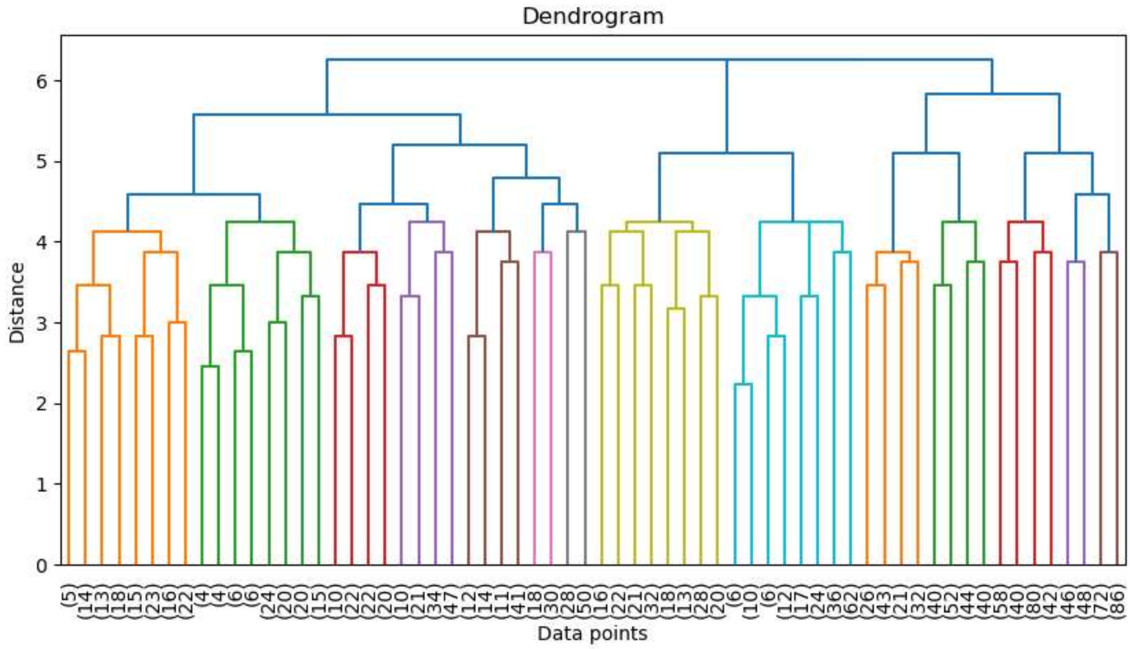
	Algorithm	linkage	n_clusters	silhouette_score
42	Agglomerative Clustering	average	2	0.150735
0	Agglomerative Clustering	ward	2	0.116495
14	Agglomerative Clustering	ward	2	0.116495
1	Agglomerative Clustering	ward	3	0.115315
15	Agglomerative Clustering	ward	3	0.115315
2	Agglomerative Clustering	ward	4	0.111471
16	Agglomerative Clustering	ward	4	0.111471
3	Agglomerative Clustering	ward	5	0.110009
17	Agglomerative Clustering	ward	5	0.110009
43	Agglomerative Clustering	average	3	0.109328

Inference: It is observed that the silhouette score is the highest when the number of clusters are 2 and the linkage is average.

Q6) Hierarchical Clustering (Dendrogram)

In [25]:

```
link = linkage(df_training, method = "complete")

# Plot dendrogram
plt.figure(figsize = (10, 5))
dendrogram(link, leaf_font_size=10, truncate_mode = "level", p = 5)
plt.xlabel("Data points")
plt.ylabel("Distance")
plt.title("Dendrogram")
plt.show()
```



Q7) Compare the better clustering algorithm with any classification algorithm, and write your notes on the same.

DecisionTreeClassifier

In [28]:

```
X = df_training.copy()
y = df['Class_Val']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_st
```

In [29]:

```python
# Build the decision tree
tree = DecisionTreeClassifier(max_depth = 5, min_samples_split = 5)

# Train the decision tree
tree.fit(X_train, y_train)

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

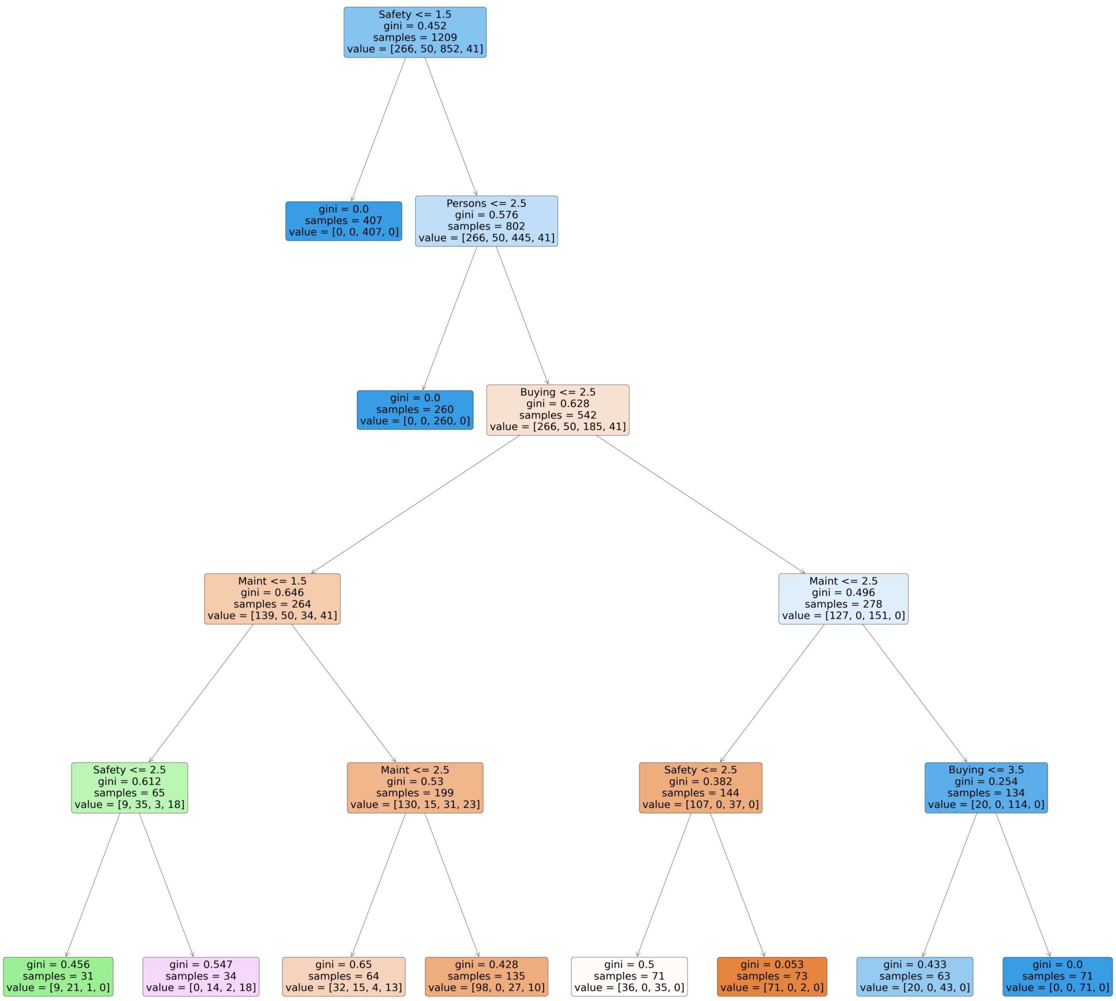
In [31]:

```python
# Evaluate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy of Decision Tree:", accuracy)
```

```
Accuracy of Decision Tree: 0.8458574181117534
```

In [33]:

```python
# Plotting the decision tree
fig, ax = plt.subplots(figsize=(80, 80))
plot_tree(tree, filled=True, rounded=True, ax=ax, feature_names=X.columns)
plt.show()
```



KNN Classifier

In [35]:

```
# Train the KNN classifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

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

# Evaluate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy of KNN:", accuracy)
```

```
Accuracy of KNN: 0.9402697495183044
```

Inference: The KMeans algorithm resulted in an optimal number of 8 clusters, while AGNES clustering found 4 clusters. KMeans partitions the data, while AGNES merges clusters hierarchically. The centroid distance metric used in KMeans is easier to comprehend, whereas reading dendrograms in AGNES requires experience and expertise. With KMeans, we had to specify the number of clusters, but in AGNES, there were no pre-determined cluster numbers.