Machine Learning Lab A3

ASIM KUMAR HANSDA

ROLL NO - 002211001136

ASSIGNMENT - 4

Github Link:

https://github.com/cryptasim/MACHINE-LEARNING-LAB

In [1]:
```
!pip install numpy==1.26.4
!pip install scikit-learn-extra
```

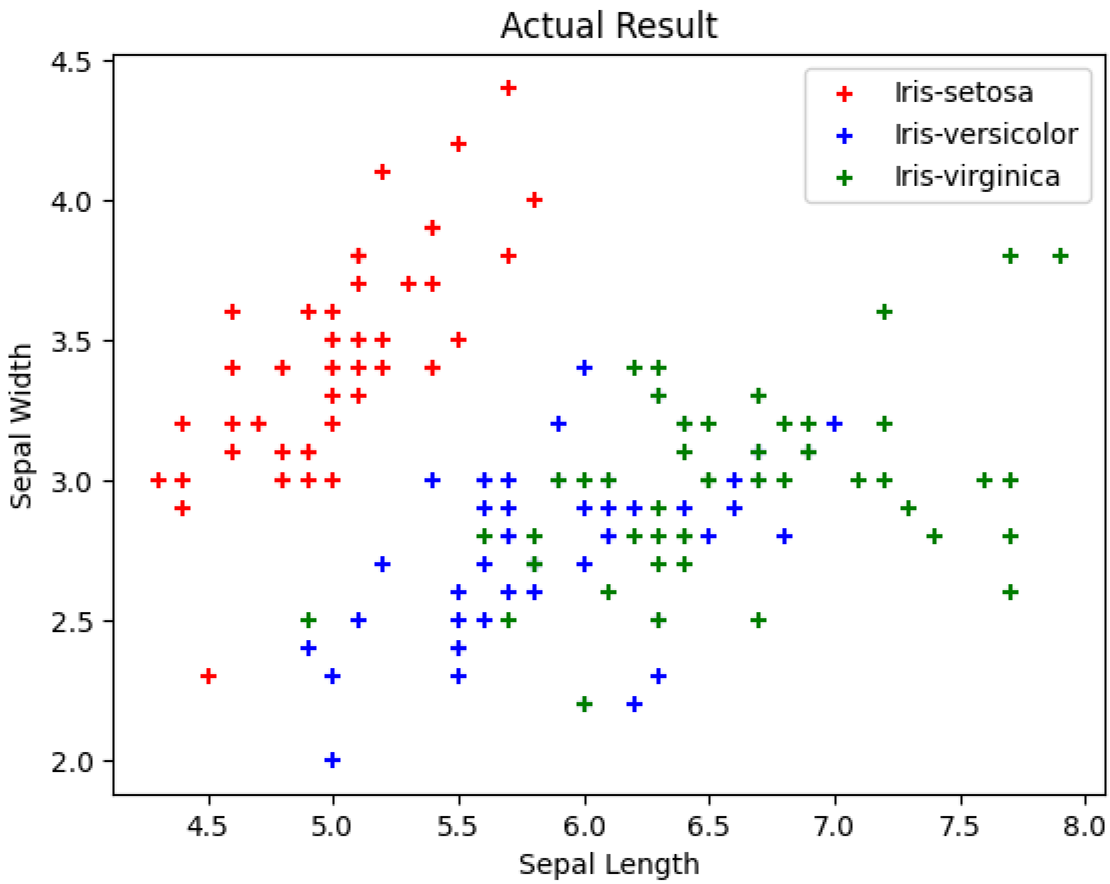
```
Requirement already satisfied: numpy==1.26.4 in /usr/local/lib/python3.12/dist-
packages (1.26.4)
Requirement already satisfied: scikit-learn-extra in /usr/local/lib/python3.12/
dist-packages (0.3.0)
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.12/dist-
packages (from scikit-learn-extra) (1.26.4)
Requirement already satisfied: scipy>=0.19.1 in /usr/local/lib/python3.12/dist-
packages (from scikit-learn-extra) (1.16.2)
Requirement already satisfied: scikit-learn>=0.23.0 in /usr/local/lib/python3.1
2/dist-packages (from scikit-learn-extra) (1.6.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.12/dist-
packages (from scikit-learn>=0.23.0->scikit-learn-extra) (1.5.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.1
2/dist-packages (from scikit-learn>=0.23.0->scikit-learn-extra) (3.6.0)
```

In [2]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import silhouette_score, calinski_harabasz_score,davies_b
```

Clustering in Iris Dataset

In [3]:
```python
from sklearn.datasets import load_iris
import pandas as pd
import matplotlib.pyplot as plt

# Load iris dataset
iris = load_iris()
df_iris = pd.DataFrame(iris.data, columns=iris.feature_names)
df_iris['species'] = [iris.target_names[i] for i in iris.target]

# Rename columns to match your plotting code
df_iris.columns = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width

# Prepare data
X = df_iris.drop('species', axis=1)
y = df_iris.species

# Actual Clustering Result
newDf0 = df_iris[df_iris.species == "setosa"]
newDf1 = df_iris[df_iris.species == "versicolor"]
newDf2 = df_iris[df_iris.species == "virginica"]

# Plot
plt.title("Actual Result")
plt.xlabel("Sepal Length")
plt.ylabel("Sepal Width")
plt.scatter(newDf0.sepal_length, newDf0.sepal_width, color="red", marker="+",
plt.scatter(newDf1.sepal_length, newDf1.sepal_width, color="blue", marker="+",
```

```
plt.scatter(newDf2.sepal_length, newDf2.sepal_width, color="green", marker="+"
plt.legend()

# 👇 This line displays the plot
plt.show()
```

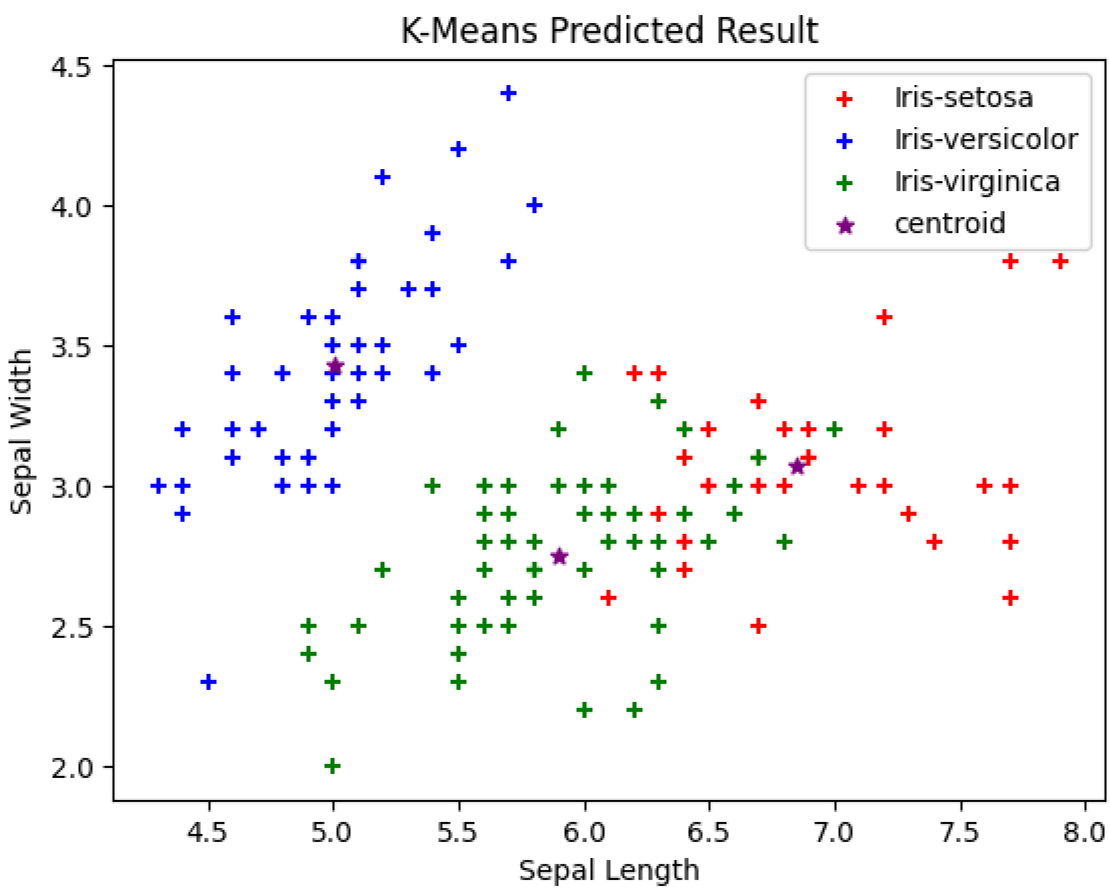


Partition Based: K-means Clustering in Iris Dataset

In [4]:
```
from sklearn.cluster import KMeans
km = KMeans(n_clusters=3, n_init=10)
y_predicted = km.fit_predict(X)
newDf = df_iris
newDf["cluster"] = y_predicted
newDf0 = newDf[newDf.cluster==0]
newDf1 = newDf[newDf.cluster==1]
newDf2 = newDf[newDf.cluster==2]
plt.title("K-Means Predicted Result")
plt.xlabel("Sepal Length")
plt.ylabel("Sepal Width")
plt.scatter(newDf0.sepal_length, newDf0.sepal_width, color="red",
marker="+", label="Iris-setosa")
plt.scatter(newDf1.sepal_length, newDf1.sepal_width, color="blue",
```

```
marker="+", label="Iris-versicolor")
plt.scatter(newDf2.sepal_length, newDf2.sepal_width, color="green",
marker="+", label="Iris-virginica")
plt.scatter(km.cluster_centers_[:,0], km.cluster_centers_[:,1], color="purple"
plt.legend()
```

Out[4]: <matplotlib.legend.Legend at 0x7f0d488d8a70>



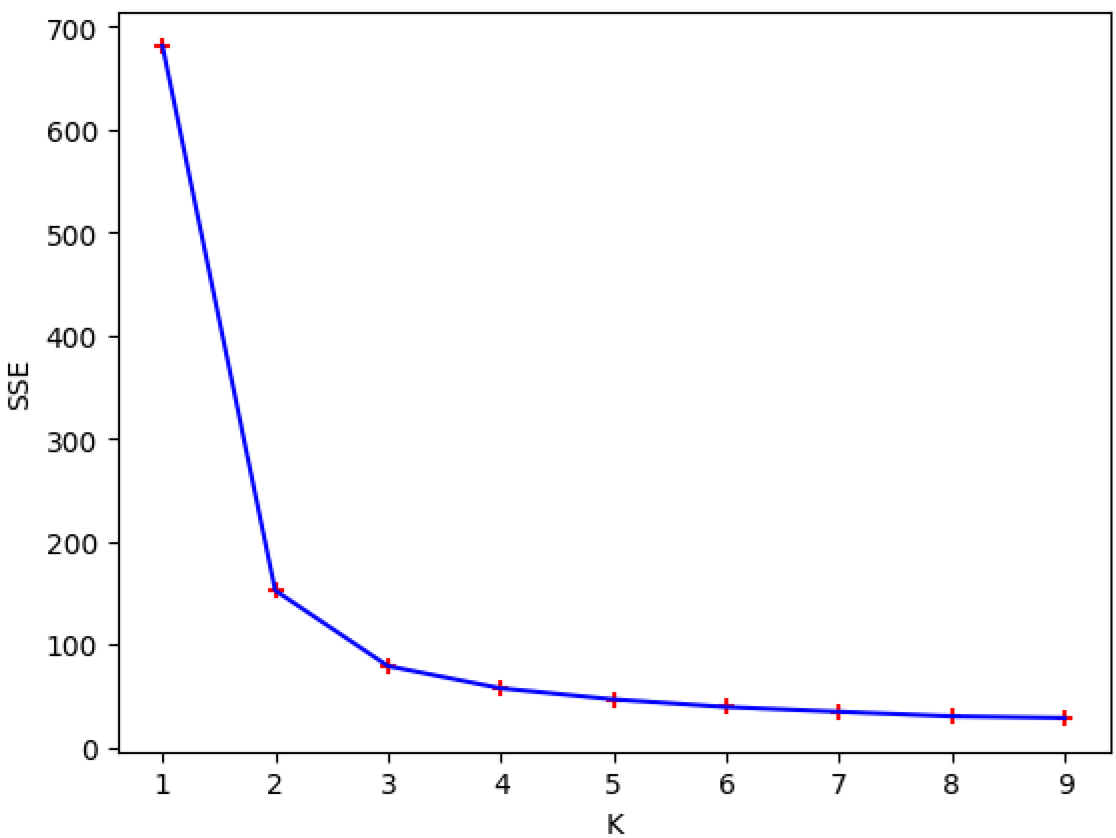
In [5]:
```
from sklearn.metrics import rand_score, adjusted_rand_score
from sklearn.metrics import mutual_info_score, adjusted_mutual_info_score, nor

# True labels
y_true = df_iris['species']

# Predicted cluster labels
y_pred = newDf['cluster']

# Rand Index
ri = rand_score(y_true, y_pred)
ari = adjusted_rand_score(y_true, y_pred)

# Mutual Information scores
mi = mutual_info_score(y_true, y_pred)
ami = adjusted_mutual_info_score(y_true, y_pred)
nmi = normalized_mutual_info_score(y_true, y_pred)
```

```
# Print results
print(f"Rand Index: {ri:.4f}")
print(f"Adjusted Rand Index: {ari:.4f}")
print(f"Mutual Information: {mi:.4f}")
print(f"Adjusted Mutual Information: {ami:.4f}")
print(f"Normalized Mutual Information: {nmi:.4f}")
```

```
Rand Index: 0.8797
Adjusted Rand Index: 0.7302
Mutual Information: 0.8256
Adjusted Mutual Information: 0.7551
Normalized Mutual Information: 0.7582
```

In [6]:
```
sse = []
k_range = range(1, 10)
for k in k_range:
 km = KMeans(n_clusters=k, n_init=10)
 km.fit_predict(X)
 sse.append(km.inertia_)
plt.xlabel("K")
plt.ylabel("SSE")
plt.scatter(k_range, sse, color="red", marker="+")
plt.plot(k_range, sse, color="blue")
```

Out[6]:
```
[<matplotlib.lines.Line2D at 0x7f0d47cef7d0>]
```



In [7]:
```
# Evaluating Metrics
silhouette_result = silhouette_score(X, km.labels_)
```

```python
print("Silhouette Score: ", silhouette_result)
calinski_result = calinski_harabasz_score(X, km.labels_)
print("Calinski Harabasz Score: ", calinski_result)
davies_result = davies_bouldin_score(X, km.labels_)
print("Davies Bouldin Score: ", davies_result)
# Evaluating Cohesion & Separation
labels = km.labels_
centroids = km.cluster_centers_
SSE = np.sum((X - centroids[labels])**2)
overall_centroid = np.mean(X, axis=0)
SSB = np.sum([np.sum((X[labels == i] - centroids[i])**2) for i in 
range(3)])
N = X.shape[0]
cohesion_scores = SSE/N
cohesion = np.mean(cohesion_scores)
separation = SSB/N
print(f"\nCohesion Score: {cohesion_scores}")
print(f"Separation Score: {separation}")
```

```
Silhouette Score:  0.31200096891430773
Calinski Harabasz Score:  404.68828649587556
Davies Bouldin Score:  0.9969403146109168

Cohesion Score: sepal_length    0.053116
sepal_width     0.052725
petal_length    0.054776
petal_width     0.028959
dtype: float64
Separation Score: 0.07235617715617715
```

```
/usr/local/lib/python3.12/dist-packages/numpy/core/fromnumeric.py:86: FutureWar
ning: The behavior of DataFrame.sum with axis=None is deprecated, in a future v
ersion this will reduce over both axes and return a scalar. To retain the old b
ehavior, pass axis=0 (or do not pass axis)
  return reduction(axis=axis, out=out, **passkwargs)
```

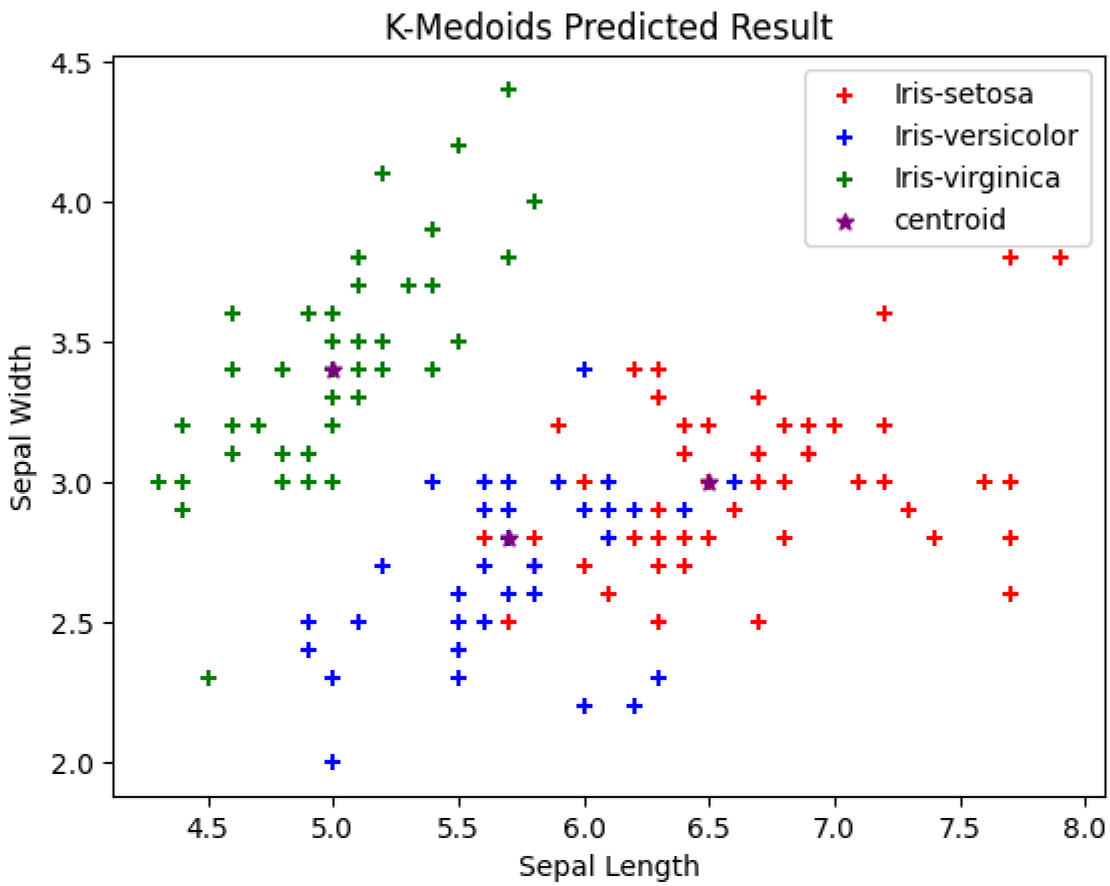
Partition Based: K-medoids Clustering in Iris Dataset

In [8]:

```python
# Clustering using K-medoids algorithm
from sklearn_extra.cluster import KMedoids
km = KMedoids(n_clusters=3)
y_predicted = km.fit_predict(X)
newDf = df_iris
newDf["cluster"] = y_predicted
newDf0 = newDf[newDf.cluster==0]
newDf1 = newDf[newDf.cluster==1]
newDf2 = newDf[newDf.cluster==2]
plt.title("K-Medoids Predicted Result")
plt.xlabel("Sepal Length")
plt.ylabel("Sepal Width")
plt.scatter(newDf0.sepal_length, newDf0.sepal_width, color="red",
```

```
marker="+", label="Iris-setosa")
plt.scatter(newDf1.sepal_length, newDf1.sepal_width, color="blue",
marker="+", label="Iris-versicolor")
plt.scatter(newDf2.sepal_length, newDf2.sepal_width, color="green",
marker="+", label="Iris-virginica")
plt.scatter(km.cluster_centers_[:,0], km.cluster_centers_[:,1],
color="purple", marker="*", label="centroid")
plt.legend()
```

Out[8]: <matplotlib.legend.Legend at 0x7f0d47d83560>



In [9]:

```
from sklearn.metrics import rand_score, adjusted_rand_score
from sklearn.metrics import mutual_info_score, adjusted_mutual_info_score, nor

# True labels
y_true = df_iris['species']

# Predicted cluster labels
y_pred = newDf['cluster']

# Rand Index
ri = rand_score(y_true, y_pred)
ari = adjusted_rand_score(y_true, y_pred)

# Mutual Information scores
mi = mutual_info_score(y_true, y_pred)
```

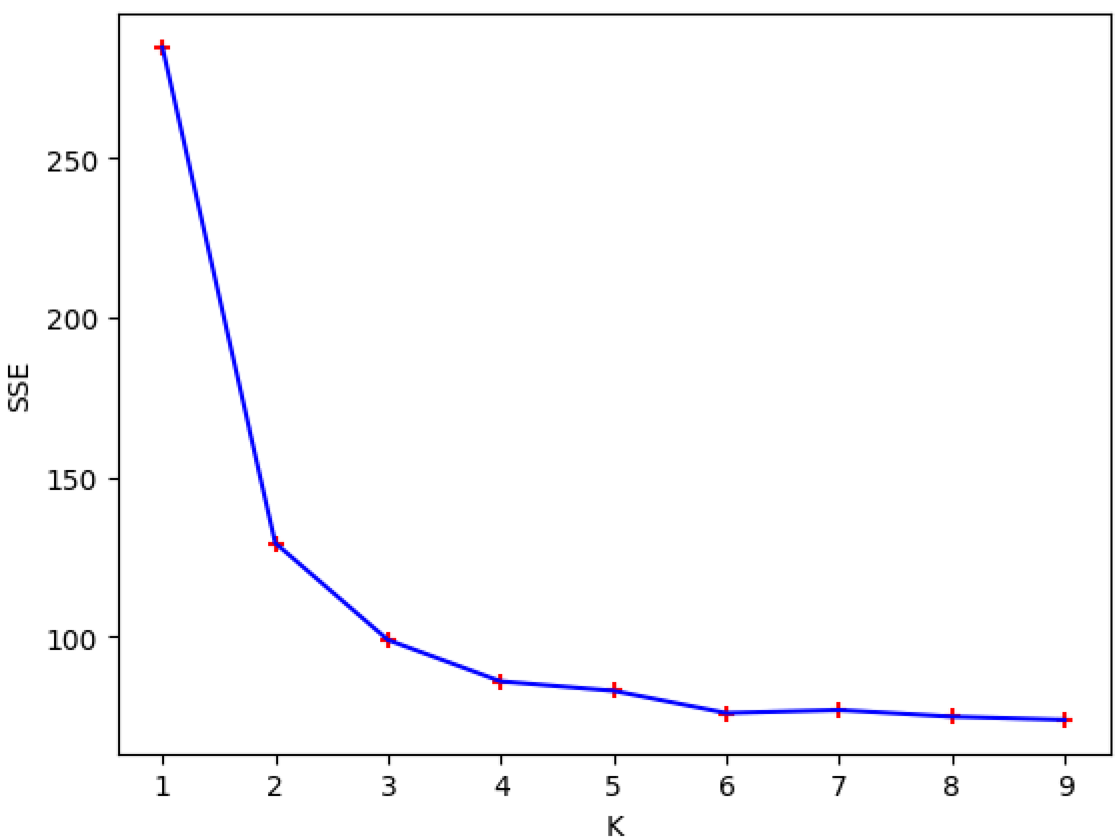
```python
ami = adjusted_mutual_info_score(y_true, y_pred)
nmi = normalized_mutual_info_score(y_true, y_pred)

# Print results
print(f"Rand Index: {ri:.4f}")
print(f"Adjusted Rand Index: {ari:.4f}")
print(f"Mutual Information: {mi:.4f}")
print(f"Adjusted Mutual Information: {ami:.4f}")
print(f"Normalized Mutual Information: {nmi:.4f}")
```

```
Rand Index: 0.8923
Adjusted Rand Index: 0.7583
Mutual Information: 0.8555
Adjusted Mutual Information: 0.7830
Normalized Mutual Information: 0.7857
```

In [10]:

```python
sse = []
k_range = range(1, 10)
for k in k_range:
 km = KMedoids(n_clusters=k)
 km.fit_predict(X)
 sse.append(km.inertia_)
plt.xlabel("K")
plt.ylabel("SSE")
plt.scatter(k_range, sse, color="red", marker="+")
plt.plot(k_range, sse, color="blue")
```

Out[10]: [<matplotlib.lines.Line2D at 0x7f0d459da2a0>]



In [11]:

```python
# Evaluating Metrics
silhouette_result = silhouette_score(X, km.labels_)
print("Silhouette Score: ", silhouette_result)
calinski_result = calinski_harabasz_score(X, km.labels_)
print("Calinski Harabasz Score: ", calinski_result)
davies_result = davies_bouldin_score(X, km.labels_)
print("Davies Bouldin Score: ", davies_result)
# Evaluating Cohesion & Separation
labels = km.labels_
centroids = km.cluster_centers_
SSE = np.sum((X - centroids[labels])**2)
overall_centroid = np.mean(X, axis=0)
SSB = np.sum([np.sum((X[labels == i] - centroids[i])**2) for i in
range(3)])
N = X.shape[0]
cohesion_scores = SSE/N
cohesion = np.mean(cohesion_scores)
separation = SSB/N
print(f"\nCohesion Score: {cohesion}")
print(f"Separation Score: {separation}")
```

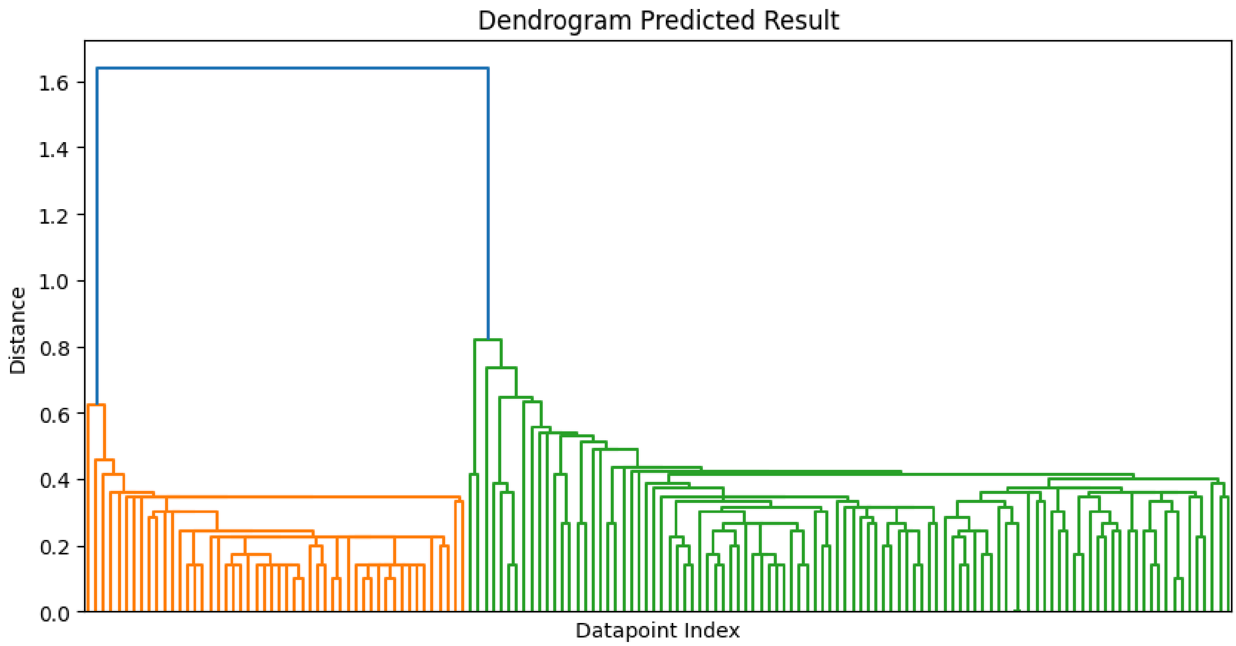
```
Silhouette Score:  0.37568265737828305
Calinski Harabasz Score:  237.92818231224678
Davies Bouldin Score:  1.1192653552269658

Cohesion Score: 0.08663333333333338
Separation Score: 0.06466666666666666
```

```
/usr/local/lib/python3.12/dist-packages/numpy/core/fromnumeric.py:86: FutureWar
ning: The behavior of DataFrame.sum with axis=None is deprecated, in a future v
ersion this will reduce over both axes and return a scalar. To retain the old b
ehavior, pass axis=0 (or do not pass axis)
  return reduction(axis=axis, out=out, **passkwargs)
```

Hierarchical: Dendrogram Clustering in Iris Dataset

In [12]:

```python
# Clustering using Dendrogram Clustering algorithm
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
Z = linkage(X, method='single')
# Create and plot the dendrogram
plt.figure(figsize=(10, 5))
dn = dendrogram(Z, no_labels=True)
plt.title('Dendrogram Predicted Result')
plt.xlabel('Datapoint Index')
plt.ylabel('Distance')
plt.show()
```



In [13]:
```
from scipy.cluster.hierarchy import fcluster
from sklearn.metrics import rand_score, adjusted_rand_score
from sklearn.metrics import mutual_info_score, adjusted_mutual_info_score, nor

# True labels (numeric)
y_true = df_iris['species']

# Cut the dendrogram to form 3 clusters
y_pred = fcluster(Z, t=3, criterion='maxclust')

# Rand Index
ri = rand_score(y_true, y_pred)
ari = adjusted_rand_score(y_true, y_pred)

# Mutual Information scores
mi = mutual_info_score(y_true, y_pred)
ami = adjusted_mutual_info_score(y_true, y_pred)
nmi = normalized_mutual_info_score(y_true, y_pred)

# Print results
print(f"Rand Index: {ri:.4f}")
print(f"Adjusted Rand Index: {ari:.4f}")
print(f"Mutual Information: {mi:.4f}")
print(f"Adjusted Mutual Information: {ami:.4f}")
print(f"Normalized Mutual Information: {nmi:.4f}")
```

```
Rand Index: 0.7766
Adjusted Rand Index: 0.5638
Mutual Information: 0.6459
Adjusted Mutual Information: 0.7126
Normalized Mutual Information: 0.7175
```

In [14]:
```
labels = fcluster(Z, 3, criterion='maxclust')
```

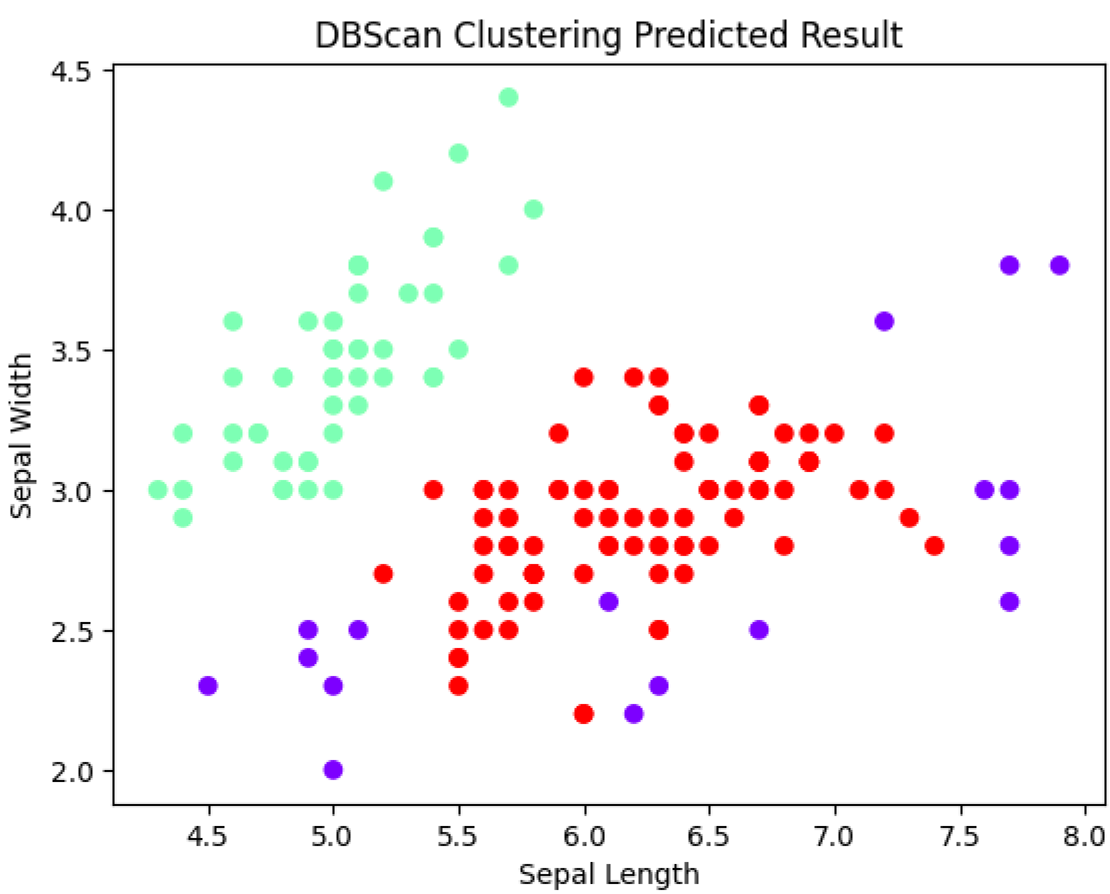
```
silhouette_result = silhouette_score(X, labels)
print("Silhouette Score: ", silhouette_result)
calinski_result = calinski_harabasz_score(X, labels)
print("Calinski Harabasz Score: ", calinski_result)
davies_result = davies_bouldin_score(X, labels)
print("Davies Bouldin Score: ", davies_result)
```

```
Silhouette Score:  0.5121107753649307
Calinski Harabasz Score:  277.99467626461944
Davies Bouldin Score:  0.4471537628542408
```

Density Based: DBSCAN Clustering in Iris Dataset

In [15]:

```
# Clustering using DBSCAN Clustering algorithm
from sklearn.cluster import DBSCAN
dbscan = DBSCAN(eps=0.5, algorithm='auto', metric='euclidean')
y = dbscan.fit_predict(X)
plt.scatter(df_iris.sepal_length, df_iris.sepal_width,
c=dbscan.labels_, cmap='rainbow')
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.title('DBScan Clustering Predicted Result')
plt.show()
```



In [16]:
```python
from sklearn.metrics import rand_score, adjusted_rand_score
from sklearn.metrics import mutual_info_score, adjusted_mutual_info_score, nor

# True labels
y_true = df_iris['species']

# Predicted cluster labels from DBSCAN
y_pred = dbscan.labels_

# If you want to ignore noise points (-1), you can filter them:
# mask = y_pred != -1
# y_true_filtered = y_true[mask]
# y_pred_filtered = y_pred[mask]

# Compute Rand Index
ri = rand_score(y_true, y_pred)
ari = adjusted_rand_score(y_true, y_pred)

# Compute Mutual Information scores
mi = mutual_info_score(y_true, y_pred)
ami = adjusted_mutual_info_score(y_true, y_pred)
nmi = normalized_mutual_info_score(y_true, y_pred)

# Print results
print(f"Rand Index: {ri:.4f}")
print(f"Adjusted Rand Index: {ari:.4f}")
print(f"Mutual Information: {mi:.4f}")
print(f"Adjusted Mutual Information: {ami:.4f}")
print(f"Normalized Mutual Information: {nmi:.4f}")
```

```
Rand Index: 0.7719
Adjusted Rand Index: 0.5206
Mutual Information: 0.6152
Adjusted Mutual Information: 0.5990
Normalized Mutual Information: 0.6044
```

In [17]:
```python
y_pred
```

Out[17]:
```
array([ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0, -1,  0,  0,  0,  0,  0,  0,  0,  0,  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,  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])
```

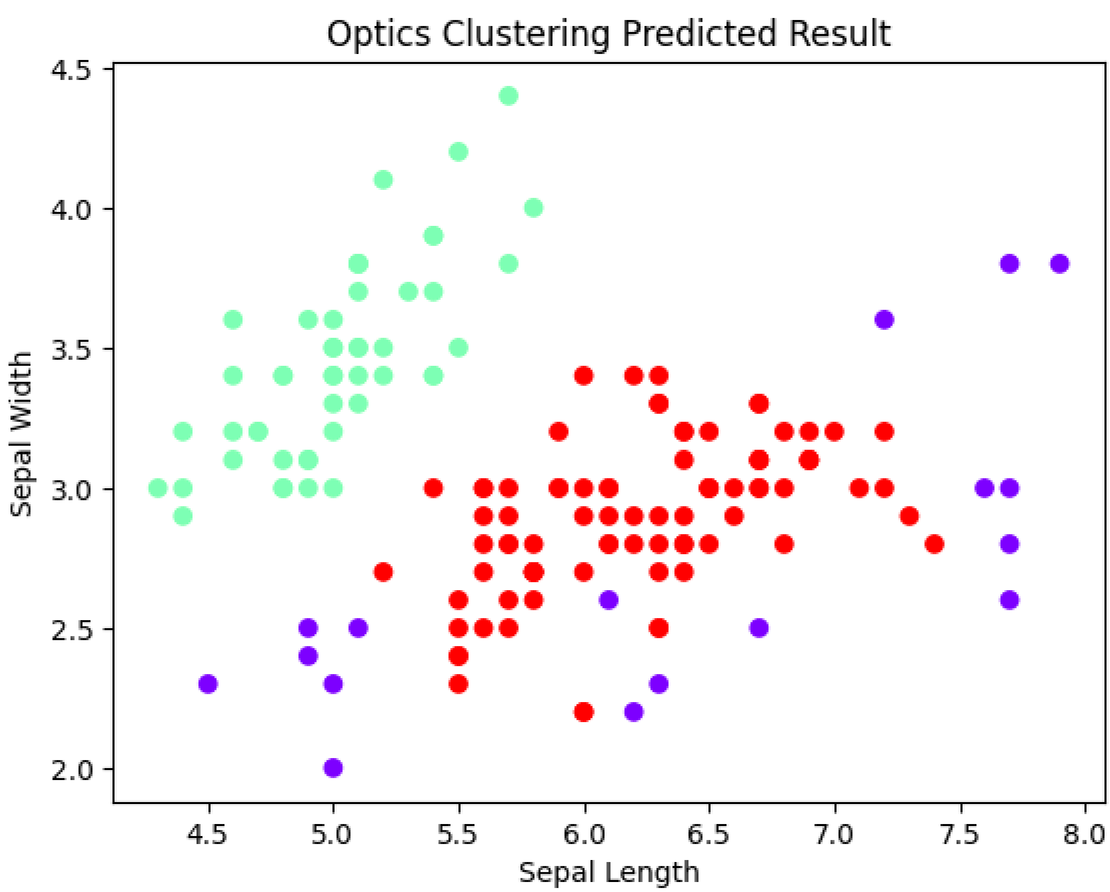
In [18]:
```python
# Evaluating Metrics
silhouette_result = silhouette_score(X, dbscan.labels_)
print("Silhouette Score: ", silhouette_result)
calinski_result = calinski_harabasz_score(X, dbscan.labels_)
print("Calinski Harabasz Score: ", calinski_result)
davies_result = davies_bouldin_score(X, dbscan.labels_)
```

```
print("Davies Bouldin Score: ", davies_result)
```

```
Silhouette Score:  0.48603419703456857
Calinski Harabasz Score:  220.29751498443005
Davies Bouldin Score:  7.222448016359581
```

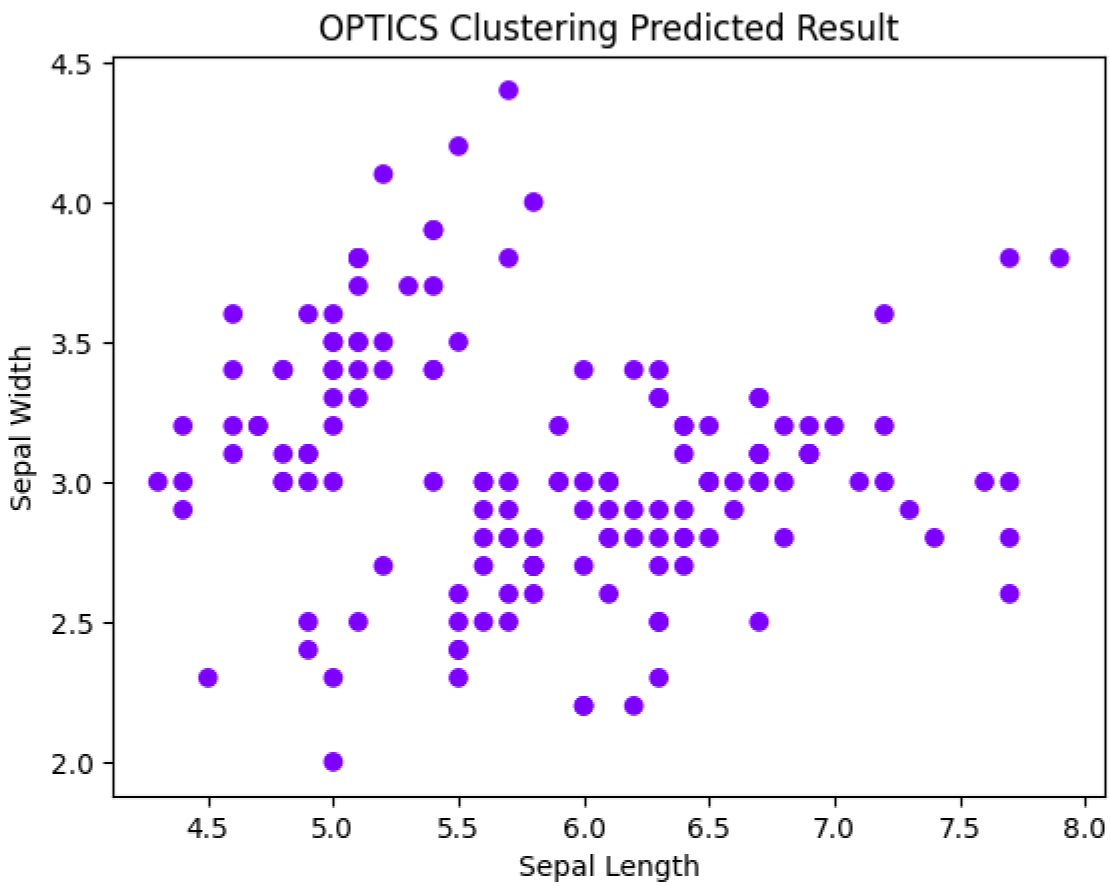
Density Based: Optics Clustering in Iris Dataset

In [19]:
```
# Clustering using Optics Clustering algorithm
from sklearn.cluster import OPTICS
optics_cluster = OPTICS(min_samples=5, xi=0.05,
cluster_method='dbscan')
optics_cluster.fit(X)
plt.scatter(df_iris.sepal_length, df_iris.sepal_width,
c=dbscan.labels_, cmap='rainbow')
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.title('Optics Clustering Predicted Result')
plt.show()
```



In [20]:
```
from sklearn.cluster import OPTICS
from sklearn.metrics import rand_score, adjusted_rand_score
from sklearn.metrics import mutual_info_score, adjusted_mutual_info_score, nor
```

```
import matplotlib.pyplot as plt

# Run OPTICS
optics_cluster = OPTICS(min_samples=5, xi=0.05, cluster_method='dbscan')
y_pred = optics_cluster.fit_predict(X)  # predicted cluster labels

# Plot clusters
plt.scatter(df_iris.sepal_length, df_iris.sepal_width, c=y_pred, cmap='rainbow
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.title('OPTICS Clustering Predicted Result')
plt.show()

# True labels
y_true = df_iris['species']

# Compute Rand Index
ri = rand_score(y_true, y_pred)
ari = adjusted_rand_score(y_true, y_pred)

# Compute Mutual Information scores
mi = mutual_info_score(y_true, y_pred)
ami = adjusted_mutual_info_score(y_true, y_pred)
nmi = normalized_mutual_info_score(y_true, y_pred)

# Print results
print(f"Rand Index: {ri:.4f}")
print(f"Adjusted Rand Index: {ari:.4f}")
print(f"Mutual Information: {mi:.4f}")
print(f"Adjusted Mutual Information: {ami:.4f}")
print(f"Normalized Mutual Information: {nmi:.4f}")
```

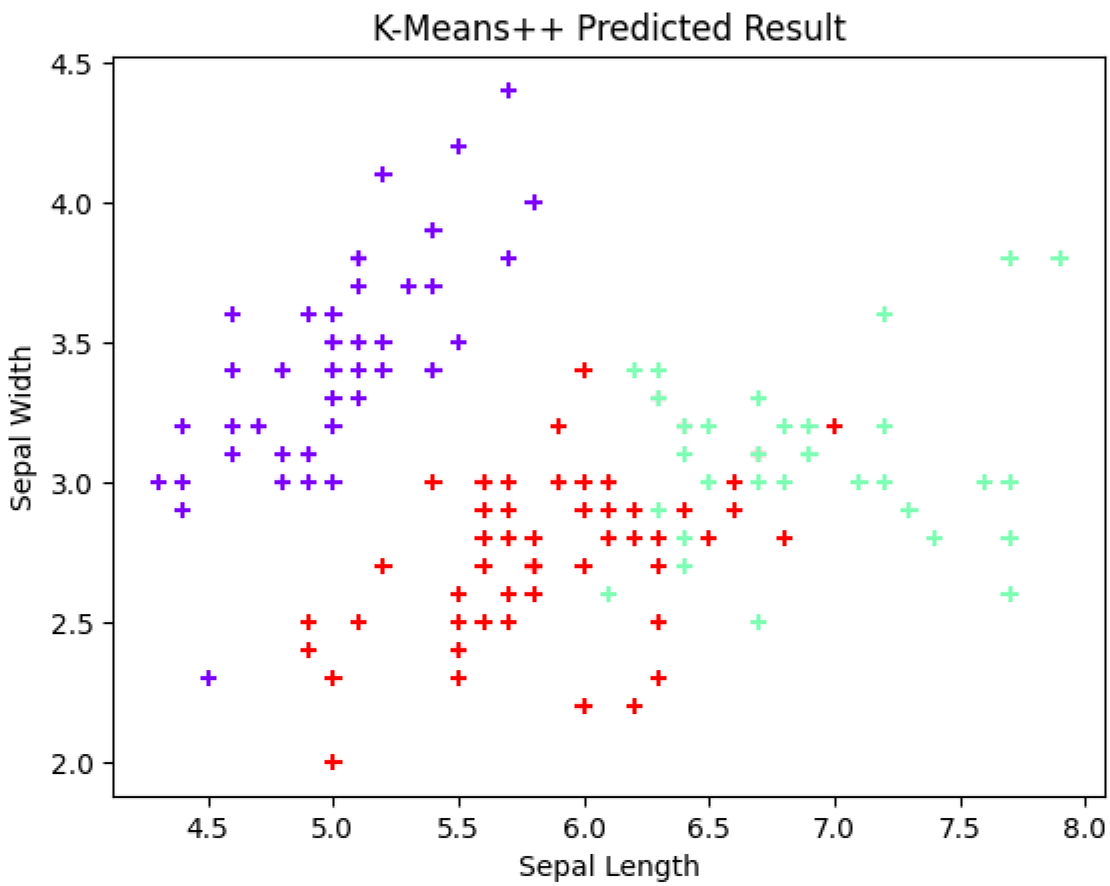



```
Rand Index: 0.3289
Adjusted Rand Index: 0.0000
Mutual Information: 0.0000
Adjusted Mutual Information: 0.0000
Normalized Mutual Information: 0.0000
```

K-means++ Clustering in Iris Dataset

In [21]:

```
# Clustering using K-means++ algorithm
from sklearn.cluster import KMeans
km = KMeans(init='k-means++', n_clusters=3, n_init=10, max_iter=300,
random_state=42)
km = KMeans(n_clusters=3, n_init=10)
y_predicted = km.fit_predict(X)
plt.title("K-Means++ Predicted Result")
plt.xlabel("Sepal Length")
plt.ylabel("Sepal Width")
plt.scatter(df_iris.sepal_length, df_iris.sepal_width, c=km.labels_,
cmap='rainbow', marker="+")
plt.show()
```

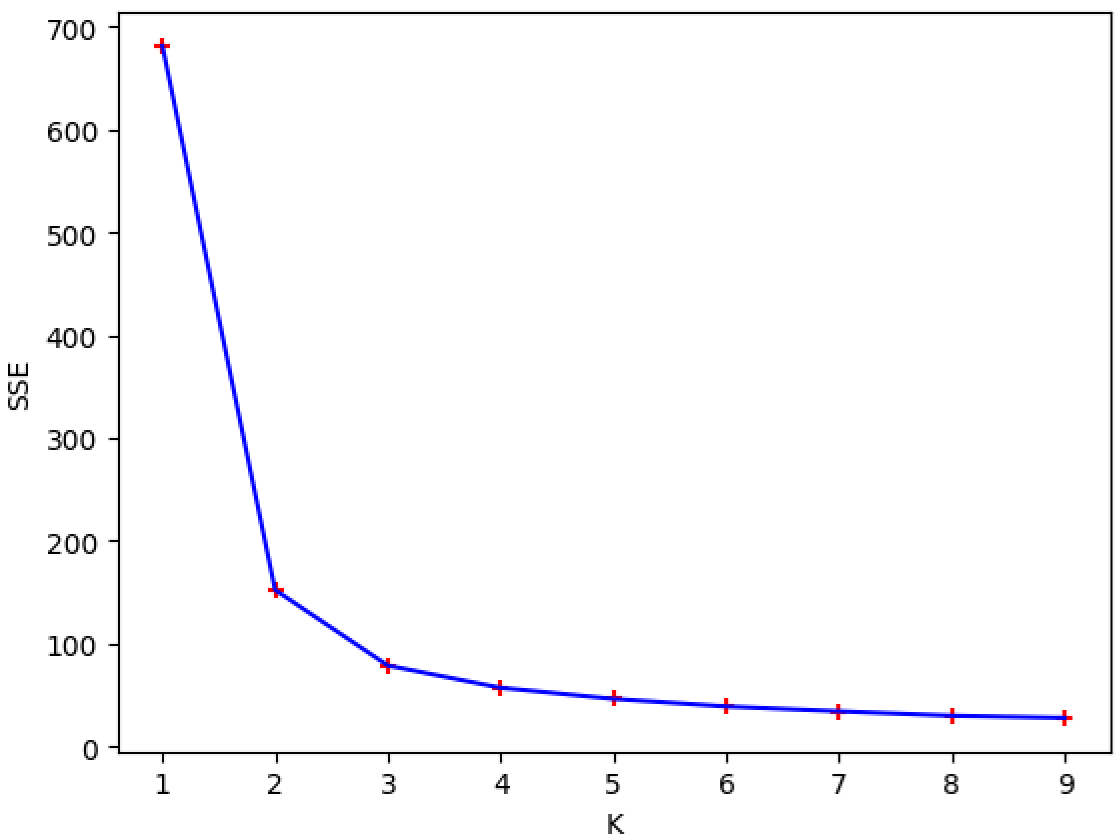


In [22]:
```
from sklearn.metrics import rand_score, adjusted_rand_score
from sklearn.metrics import mutual_info_score, adjusted_mutual_info_score, nor

# True labels
y_true = df_iris['species']

# Predicted cluster labels from K-Means++
y_pred = km.labels_  # or y_predicted

# Compute Rand Index
ri = rand_score(y_true, y_pred)
ari = adjusted_rand_score(y_true, y_pred)

# Compute Mutual Information scores
mi = mutual_info_score(y_true, y_pred)
ami = adjusted_mutual_info_score(y_true, y_pred)
nmi = normalized_mutual_info_score(y_true, y_pred)

# Print results
print(f"Rand Index: {ri:.4f}")
print(f"Adjusted Rand Index: {ari:.4f}")
print(f"Mutual Information: {mi:.4f}")
print(f"Adjusted Mutual Information: {ami:.4f}")
print(f"Normalized Mutual Information: {nmi:.4f}")
```

```
Rand Index: 0.8797
Adjusted Rand Index: 0.7302
Mutual Information: 0.8256
Adjusted Mutual Information: 0.7551
Normalized Mutual Information: 0.7582
```

In [23]:
```python
# Visualisation of SSE (Sum of Squared Errors) & Elbow Graph:
sse = []
k_range = range(1, 10)
for k in k_range:
 km = KMeans(n_clusters=k, n_init=10)
 km.fit_predict(X)
 sse.append(km.inertia_)
plt.xlabel("K")
plt.ylabel("SSE")
plt.scatter(k_range, sse, color="red", marker="+")
plt.plot(k_range, sse, color="blue")
# We can see here, our elbow is at K=3
```

Out[23]: `[<matplotlib.lines.Line2D at 0x7f0d4580af60>]`



In [24]:
```python
# Evaluating Metrics
silhouette_result = silhouette_score(X, km.labels_)
print("Silhouette Score: ", silhouette_result)
calinski_result = calinski_harabasz_score(X, km.labels_)
print("Calinski Harabasz Score: ", calinski_result)
davies_result = davies_bouldin_score(X, km.labels_)
print("Davies Bouldin Score: ", davies_result)
```

```
# Evaluating Cohesion & Separation
labels = km.labels_
centroids = km.cluster_centers_
SSE = np.sum((X - centroids[labels])**2)
overall_centroid = np.mean(X, axis=0)
SSB = np.sum([np.sum((X[labels == i] - centroids[i])**2) for i in 
range(3)])
N = X.shape[0]
cohesion_scores = SSE/N
cohesion = np.mean(cohesion_scores)
separation = SSB/N
print(f"\nCohesion Score: {cohesion}")
print(f"Separation Score: {separation}")
```

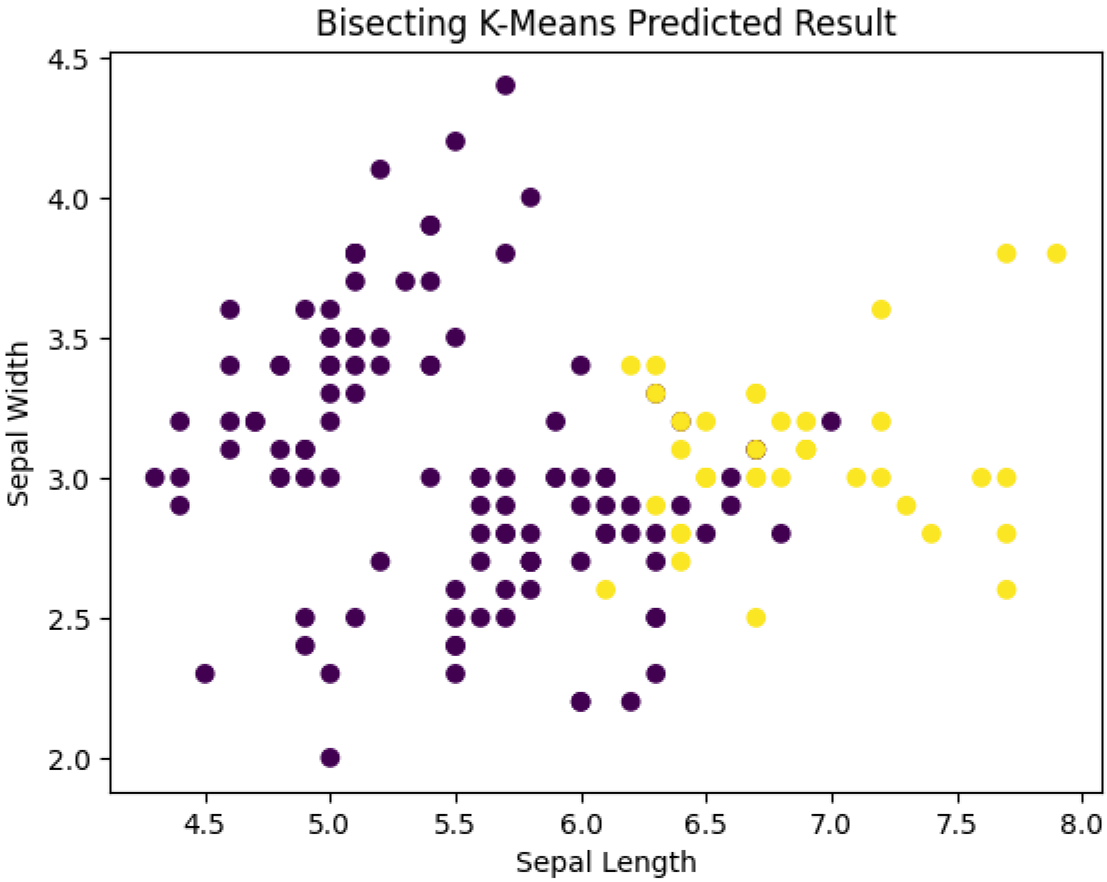
```
Silhouette Score:  0.3416185449488845
Calinski Harabasz Score:  411.50528902921917
Davies Bouldin Score:  0.933140542284917

Cohesion Score: 0.046641455722639925
Separation Score: 0.06232722222222222
```

```
/usr/local/lib/python3.12/dist-packages/numpy/core/fromnumeric.py:86: FutureWar
ning: The behavior of DataFrame.sum with axis=None is deprecated, in a future v
ersion this will reduce over both axes and return a scalar. To retain the old b
ehavior, pass axis=0 (or do not pass axis)
  return reduction(axis=axis, out=out, **passkwargs)
```

Bisecting K-means Clustering in Iris Dataset

In [25]:

```
# Clustering using Bisecting K-means algorithm
from sklearn.cluster import KMeans
km = KMeans(n_clusters=1, n_init=10, random_state=0).fit(X)
K=3
for i in range(K-1):
 largest_cluster = np.argmax(np.bincount(km.labels_))
 largest_cluster_mask = (km.labels_ == largest_cluster)
 X_split = X[largest_cluster_mask]
 km.labels_[largest_cluster_mask] = KMeans(n_clusters=2, n_init=10, 
random_state=0).fit(X_split).labels_
plt.title("Bisecting K-Means Predicted Result")
plt.xlabel("Sepal Length")
plt.ylabel("Sepal Width")
plt.scatter(df_iris.sepal_length, df_iris.sepal_width, c=km.labels_, 
cmap='viridis')
plt.show()
```



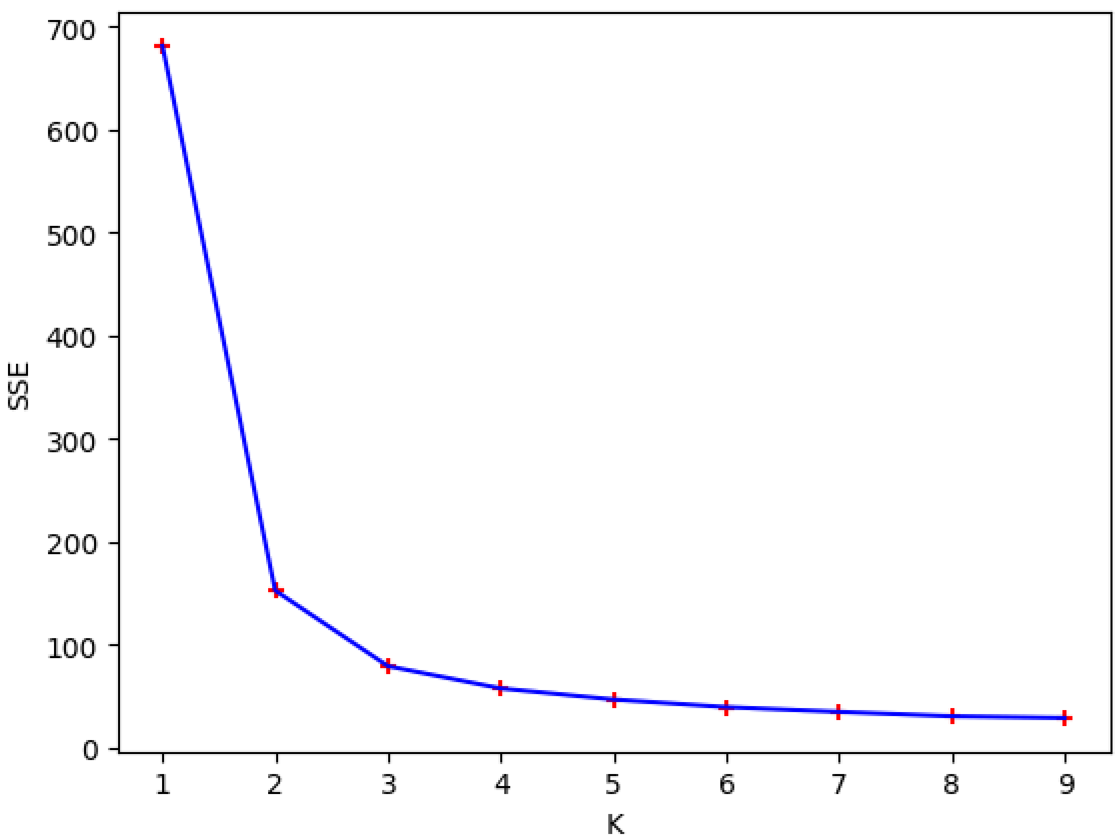
In [26]:

```
from sklearn.metrics import rand_score, adjusted_rand_score
from sklearn.metrics import mutual_info_score, adjusted_mutual_info_score, nor

# True labels
y_true = df_iris['species']

# Predicted cluster labels from Bisecting K-Means
y_pred = km.labels_

# Compute Rand Index
ri = rand_score(y_true, y_pred)
ari = adjusted_rand_score(y_true, y_pred)

# Compute Mutual Information scores
mi = mutual_info_score(y_true, y_pred)
ami = adjusted_mutual_info_score(y_true, y_pred)
nmi = normalized_mutual_info_score(y_true, y_pred)

# Print results
print(f"Rand Index: {ri:.4f}")
print(f"Adjusted Rand Index: {ari:.4f}")
print(f"Mutual Information: {mi:.4f}")
print(f"Adjusted Mutual Information: {ami:.4f}")
print(f"Normalized Mutual Information: {nmi:.4f}")
```

```
Rand Index: 0.6023
Adjusted Rand Index: 0.2646
Mutual Information: 0.3123
Adjusted Mutual Information: 0.3701
Normalized Mutual Information: 0.3753
```

In [27]:
```python
# Visualisation of SSE (Sum of Squared Errors) & Elbow Graph:
sse = []
k_range = range(1, 10)
for k in k_range:
 km = KMeans(n_clusters=k, n_init=10)
 km.fit_predict(X)
 sse.append(km.inertia_)
plt.xlabel("K")
plt.ylabel("SSE")
plt.scatter(k_range, sse, color="red", marker="+")
plt.plot(k_range, sse, color="blue")
# We can see here, our elbow is at K=3
```

Out[27]:
```
[<matplotlib.lines.Line2D at 0x7f0d45641700>]
```



In [28]:
```python
# Evaluating Metrics
silhouette_result = silhouette_score(X, km.labels_)
print("Silhouette Score: ", silhouette_result)
calinski_result = calinski_harabasz_score(X, km.labels_)
print("Calinski Harabasz Score: ", calinski_result)
davies_result = davies_bouldin_score(X, km.labels_)
print("Davies Bouldin Score: ", davies_result)
```

```python
# Evaluating Cohesion & Separation
labels = km.labels_
centroids = km.cluster_centers_
SSE = np.sum((X - centroids[labels])**2)
overall_centroid = np.mean(X, axis=0)
SSB = np.sum([np.sum((X[labels == i] - centroids[i])**2) for i in
range(3)])
N = X.shape[0]
cohesion_scores = SSE/N
cohesion = np.mean(cohesion_scores)
separation = SSB/N
print(f"\nCohesion Score: {cohesion}")
print(f"Separation Score: {separation}")
```

```
Silhouette Score:  0.3383490904961073
Calinski Harabasz Score:  403.26070549187233
Davies Bouldin Score:  0.9782372259014865

Cohesion Score: 0.04755509895877542
Separation Score: 0.08940774410774412
```

```
/usr/local/lib/python3.12/dist-packages/numpy/core/fromnumeric.py:86: FutureWar
ning: The behavior of DataFrame.sum with axis=None is deprecated, in a future v
ersion this will reduce over both axes and return a scalar. To retain the old b
ehavior, pass axis=0 (or do not pass axis)
  return reduction(axis=axis, out=out, **passkwargs)
```

WINE DATASET

In [29]:

```
pip install ucimlrepo
```

```
Collecting ucimlrepo
  Downloading ucimlrepo-0.0.7-py3-none-any.whl.metadata (5.5 kB)
Requirement already satisfied: pandas>=1.0.0 in /usr/local/lib/python3.12/dist-
packages (from ucimlrepo) (2.2.2)
Requirement already satisfied: certifi>=2020.12.5 in /usr/local/lib/python3.12/
dist-packages (from ucimlrepo) (2025.10.5)
Requirement already satisfied: numpy>=1.26.0 in /usr/local/lib/python3.12/dist-
packages (from pandas>=1.0.0->ucimlrepo) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python
3.12/dist-packages (from pandas>=1.0.0->ucimlrepo) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-p
ackages (from pandas>=1.0.0->ucimlrepo) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dis
t-packages (from pandas>=1.0.0->ucimlrepo) (2025.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packa
ges (from python-dateutil>=2.8.2->pandas>=1.0.0->ucimlrepo) (1.17.0)
Downloading ucimlrepo-0.0.7-py3-none-any.whl (8.0 kB)
Installing collected packages: ucimlrepo
Successfully installed ucimlrepo-0.0.7
```

In [30]:

```python
from ucimlrepo import fetch_ucirepo
```

```python
# fetch dataset
wine = fetch_ucirepo(id=109)

# data (as pandas dataframes)
X = wine.data.features
y = wine.data.targets

# metadata
print(wine.metadata)

# variable information
print(wine.variables)
df = X.copy()
df['class'] = y  # add target column
print(df.head())
```

{'uci_id': 109, 'name': 'Wine', 'repository_url': 'https://archive.ics.uci.edu/dataset/109/wine', 'data_url': 'https://archive.ics.uci.edu/static/public/109/data.csv', 'abstract': 'Using chemical analysis to determine the origin of wines', 'area': 'Physics and Chemistry', 'tasks': ['Classification'], 'characteristics': ['Tabular'], 'num_instances': 178, 'num_features': 13, 'feature_types': ['Integer', 'Real'], 'demographics': [], 'target_col': ['class'], 'index_col': None, 'has_missing_values': 'no', 'missing_values_symbol': None, 'year_of_dataset_creation': 1992, 'last_updated': 'Mon Aug 28 2023', 'dataset_doi': '10.24432/C5PC7J', 'creators': ['Stefan Aeberhard', 'M. Forina'], 'intro_paper': {'ID': 246, 'type': 'NATIVE', 'title': 'Comparative analysis of statistical pattern recognition methods in high dimensional settings', 'authors': 'S. Aeberhard, D. Coomans, O. Vel', 'venue': 'Pattern Recognition', 'year': 1994, 'journal': None, 'DOI': '10.1016/0031-3203(94)90145-7', 'URL': 'https://www.semanticscholar.org/paper/83dc3e4030d7b9fbdbb4bde03ce12ab70ca10528', 'sha': None, 'corpus': None, 'arxiv': None, 'mag': None, 'acl': None, 'pmid': None, 'pmcid': None}, 'additional_info': {'summary': 'These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines. \r\n\r\nI think that the initial data set had around 30 variables, but for some reason I only have the 13 dimensional version. I had a list of what the 30 or so variables were, but a.) I lost it, and b.), I would not know which 13 variables are included in the set.\r\n\r\nThe attributes are (dontated by Riccardo Leardi, riclea@anchem.unige.it)\r\n1) Alcohol\r\n2) Malic acid\r\n3) Ash\r\n4) Alcalinity of ash \r\n5) Magnesium\r\n6) Total phenols\r\n7) Flavanoids\r\n8) Nonflavanoid phenols\r\n9) Proanthocyanins\r\n10)Color intensity\r\n11)Hue\r\n12)OD280/OD315 of diluted wines\r\n13)Proline \r\n\r\nIn a classification context, this is a well posed problem with "well behaved" class structures. A good data set for first testing of a new classifier, but not very challenging.   ', 'purpose': 'test', 'funded_by': None, 'instances_represent': None, 'recommended_data_splits': None, 'sensitive_data': None, 'preprocessing_description': None, 'variable_info': 'All attributes are continuous\r\n\t\r\nNo statistics available, but suggest to standardise variables for certain uses (e.g. for us with classifiers which are NOT scale invariant)\r\n\r\nNOTE: 1st attribute is class identifier (1-3)', 'citation': None}}

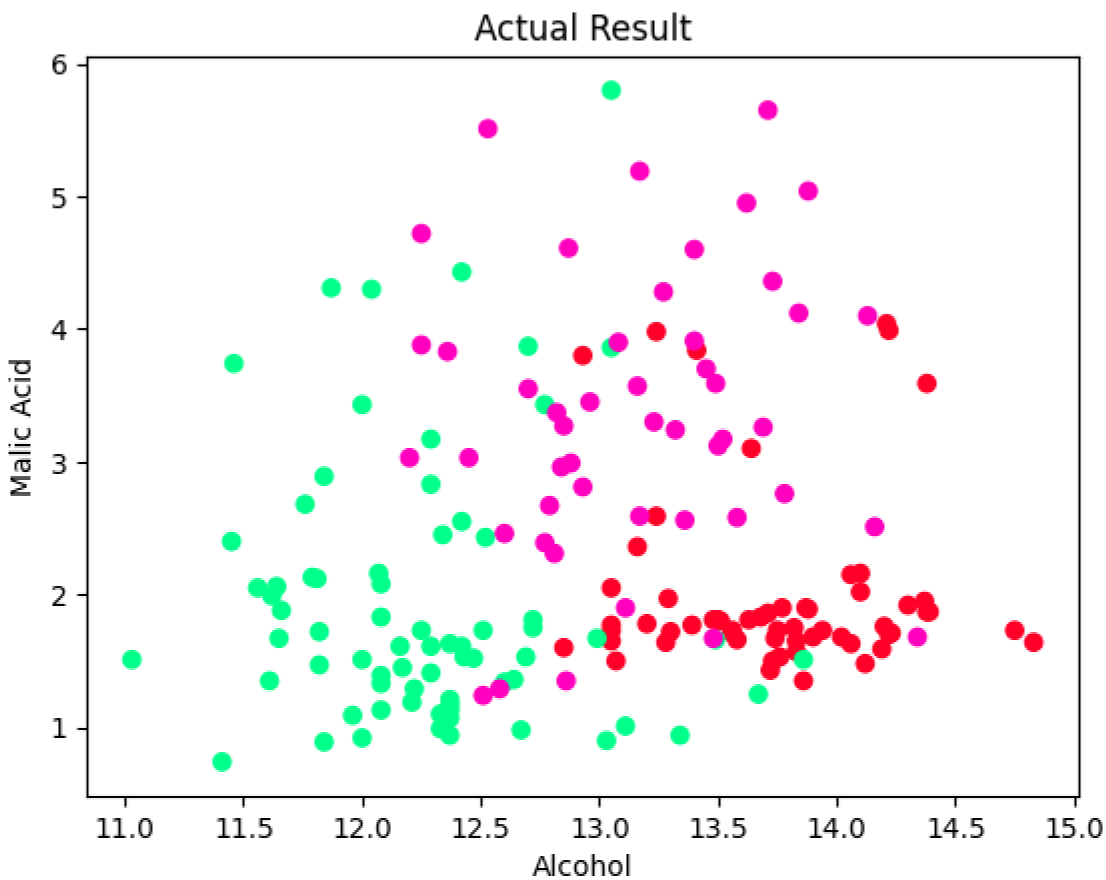
```
                            name     role         type demographic  \
0                          class   Target  Categorical        None
1                        Alcohol  Feature   Continuous        None
2                      Malicacid  Feature   Continuous        None
3                            Ash  Feature   Continuous        None
4              Alcalinity_of_ash  Feature   Continuous        None
5                      Magnesium  Feature      Integer        None
6                  Total_phenols  Feature   Continuous        None
7                     Flavanoids  Feature   Continuous        None
8           Nonflavanoid_phenols  Feature   Continuous        None
9                Proanthocyanins  Feature   Continuous        None
10               Color_intensity  Feature   Continuous        None
11                           Hue  Feature   Continuous        None
12  0D280_0D315_of_diluted_wines  Feature   Continuous        None
13                       Proline  Feature      Integer        None

   description units missing_values
0         None  None             no
1         None  None             no
2         None  None             no
```

```
3          None  None             no
4          None  None             no
5          None  None             no
6          None  None             no
7          None  None             no
8          None  None             no
9          None  None             no
10         None  None             no
11         None  None             no
12         None  None             no
13         None  None             no
   Alcohol  Malicacid   Ash  Alcalinity_of_ash  Magnesium  Total_phenols  \
0    14.23       1.71  2.43               15.6        127           2.80
1    13.20       1.78  2.14               11.2        100           2.65
2    13.16       2.36  2.67               18.6        101           2.80
3    14.37       1.95  2.50               16.8        113           3.85
4    13.24       2.59  2.87               21.0        118           2.80

   Flavanoids  Nonflavanoid_phenols  Proanthocyanins  Color_intensity   Hue  \
0        3.06                  0.28             2.29             5.64  1.04
1        2.76                  0.26             1.28             4.38  1.05
2        3.24                  0.30             2.81             5.68  1.03
3        3.49                  0.24             2.18             7.80  0.86
4        2.69                  0.39             1.82             4.32  1.04

   0D280_0D315_of_diluted_wines  Proline  class
0                          3.92     1065      1
1                          3.40     1050      1
2                          3.17     1185      1
3                          3.45     1480      1
4                          2.93      735      1
```

In [31]:

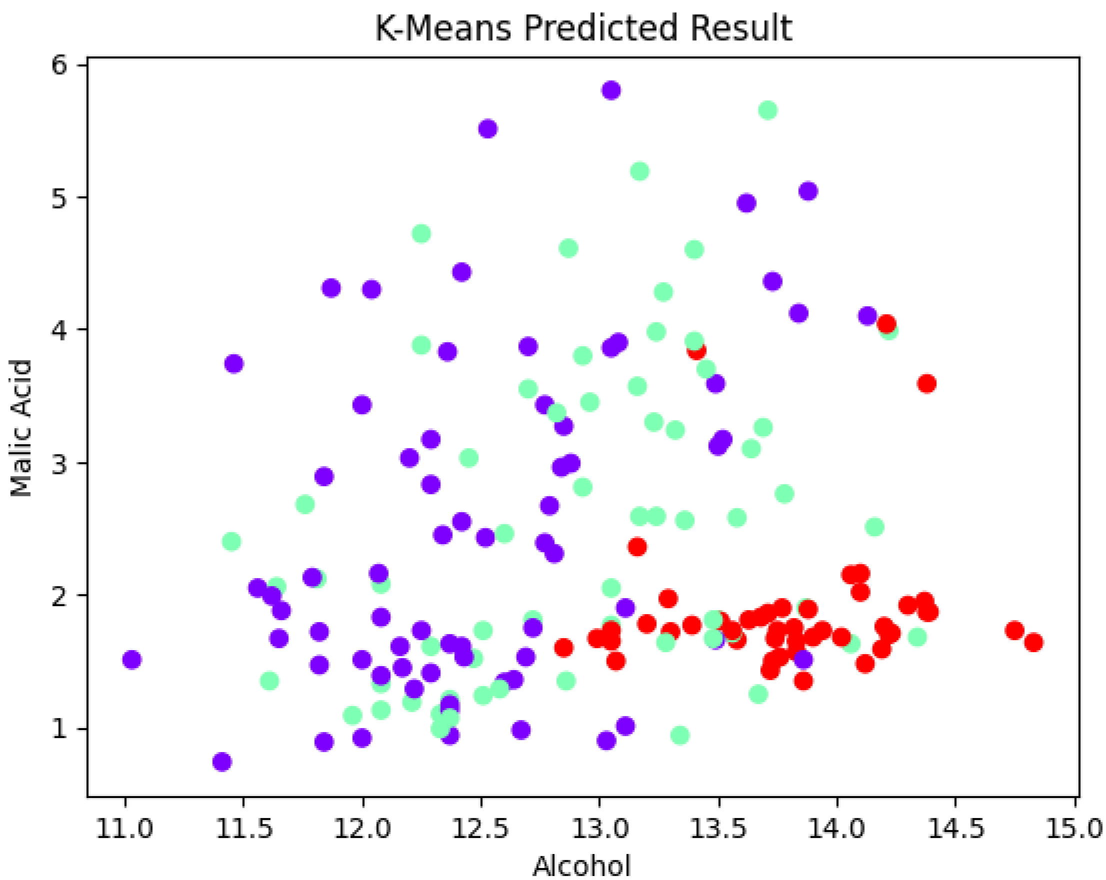
```
plt.title("Actual Result")
plt.xlabel('Alcohol')
plt.ylabel('Malic Acid')
plt.scatter(df.Alcohol, df.Malicacid, c=df["class"],
cmap='gist_rainbow')
```

Out[31]: <matplotlib.collections.PathCollection at 0x7f0d455177d0>



Partition Based: K-means Clustering in Wine Dataset

In [32]:
```
# Clustering using K-means algorithm
from sklearn.cluster import KMeans
km = KMeans(init="random", n_clusters=3, n_init=10, max_iter=300,
random_state=42)
y_predicted = km.fit_predict(X)
plt.title("K-Means Predicted Result")
plt.xlabel("Alcohol")
plt.ylabel("Malic Acid")
plt.scatter(df.Alcohol, df.Malicacid, c=km.labels_, cmap='rainbow')
plt.show()
```



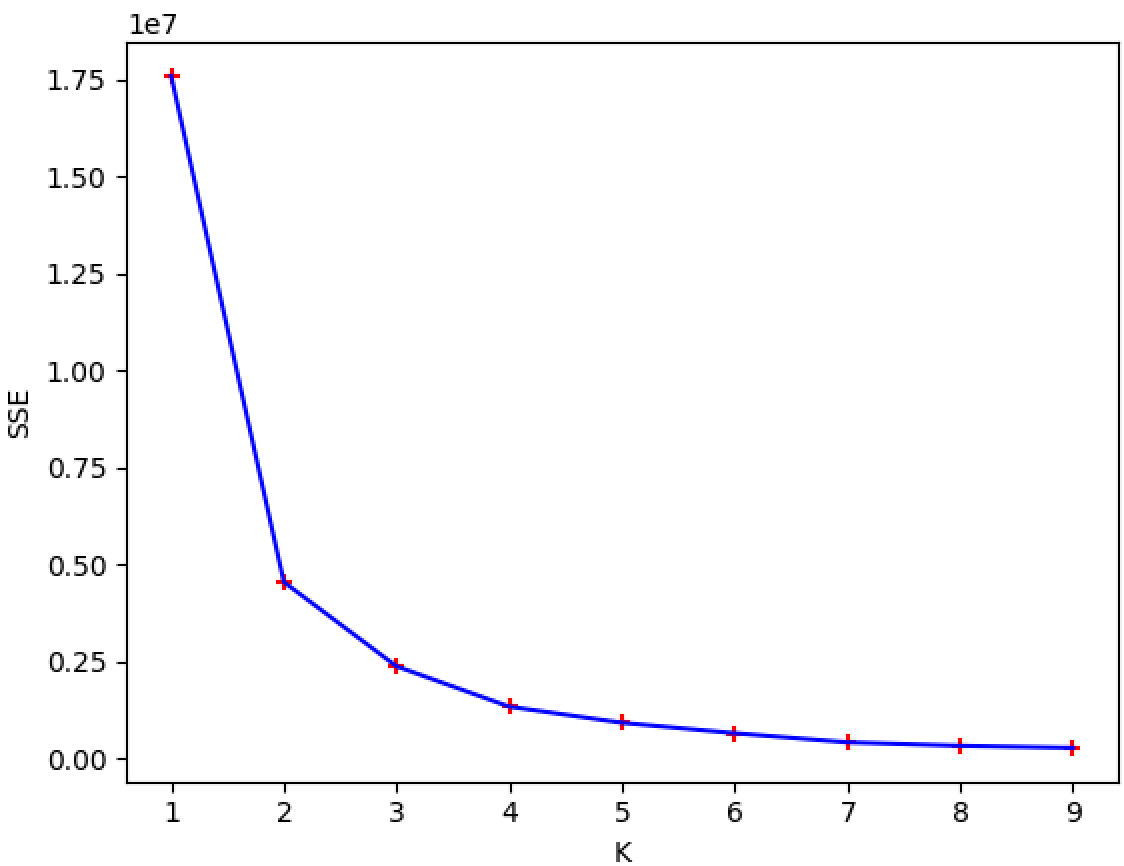
In [33]:

```
from sklearn.metrics import rand_score, adjusted_rand_score
from sklearn.metrics import mutual_info_score, adjusted_mutual_info_score, nor

# True labels (numeric for Wine dataset)
y_true = df['class']

# Predicted cluster labels from K-Means
y_pred = km.labels_  # or y_predicted

# Compute Rand Index
ri = rand_score(y_true, y_pred)
ari = adjusted_rand_score(y_true, y_pred)

# Compute Mutual Information scores
mi = mutual_info_score(y_true, y_pred)
ami = adjusted_mutual_info_score(y_true, y_pred)
nmi = normalized_mutual_info_score(y_true, y_pred)

# Print results
print(f"Rand Index: {ri:.4f}")
print(f"Adjusted Rand Index: {ari:.4f}")
print(f"Mutual Information: {mi:.4f}")
print(f"Adjusted Mutual Information: {ami:.4f}")
print(f"Normalized Mutual Information: {nmi:.4f}")
```

```
Rand Index: 0.7187
Adjusted Rand Index: 0.3711
Mutual Information: 0.4657
Adjusted Mutual Information: 0.4227
Normalized Mutual Information: 0.4288
```

In [34]:
```
# Visualisation of SSE (Sum of Squared Errors) & Elbow Graph:
sse = []
k_range = range(1, 10)
for k in k_range:
 km = KMeans(n_clusters=k, n_init=10)
 km.fit_predict(X)
 sse.append(km.inertia_)
plt.xlabel("K")
plt.ylabel("SSE")
plt.scatter(k_range, sse, color="red", marker="+")
plt.plot(k_range, sse, color="blue")
# We can see here, our elbow is at K=3
```

Out[34]:
```
[<matplotlib.lines.Line2D at 0x7f0d456d77d0>]
```



In [35]:
```
# Evaluating Metrics
silhouette_result = silhouette_score(X, km.labels_)
print("Silhouette Score: ", silhouette_result)
calinski_result = calinski_harabasz_score(X, km.labels_)
print("Calinski Harabasz Score: ", calinski_result)
davies_result = davies_bouldin_score(X, km.labels_)
```

```
print("Davies Bouldin Score: ", davies_result)
# Evaluating Cohesion & Separation
labels = km.labels_
centroids = km.cluster_centers_
SSE = np.sum((X - centroids[labels])**2)
overall_centroid = np.mean(X, axis=0)
SSB = np.sum([np.sum((X[labels == i] - centroids[i])**2) for i in
range(3)])
N = X.shape[0]
cohesion_scores = SSE/N
cohesion = np.mean(cohesion_scores)
separation = SSB/N
print(f"\nCohesion Score: {cohesion}")
print(f"Separation Score: {separation}")
```

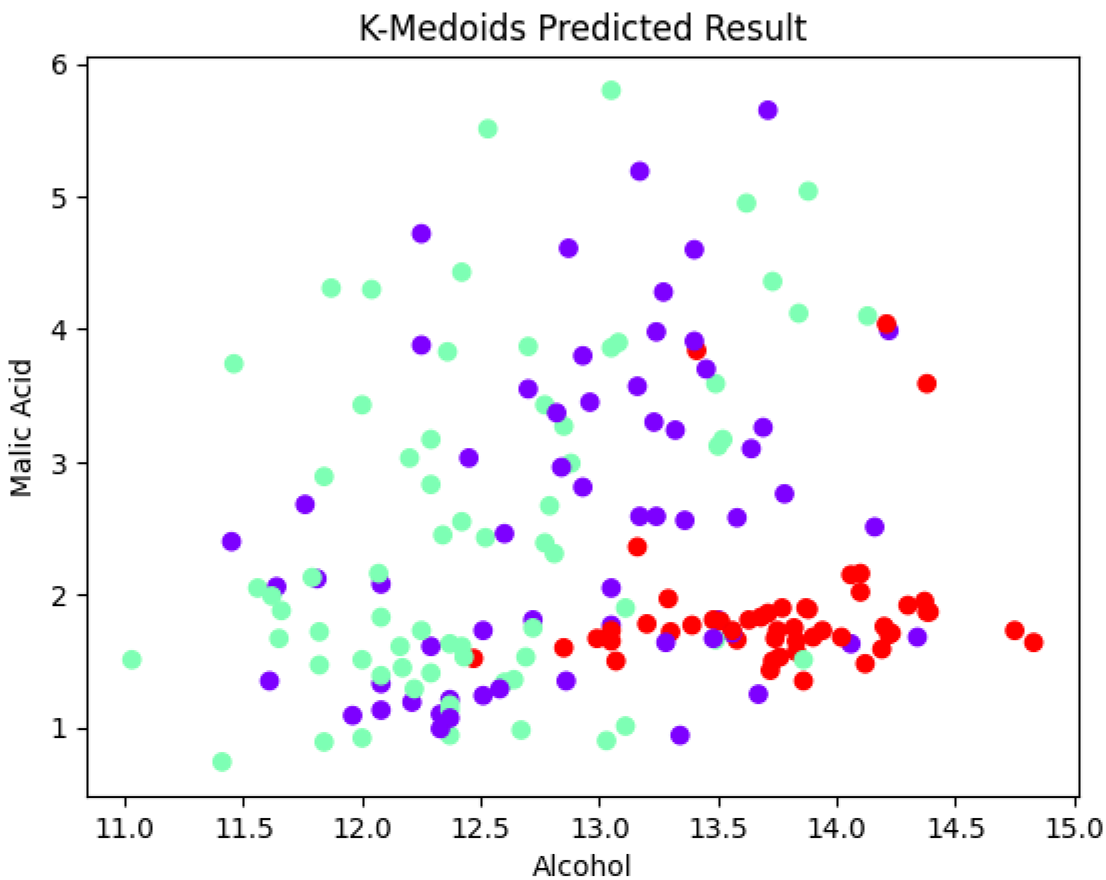
```
Silhouette Score:  0.527941546551372
Calinski Harabasz Score:  1354.5160325267275
Davies Bouldin Score:  0.5307163453404704

Cohesion Score: 116.74835625456451
Separation Score: 578.7648547517636
```

```
/usr/local/lib/python3.12/dist-packages/numpy/core/fromnumeric.py:86: FutureWar
ning: The behavior of DataFrame.sum with axis=None is deprecated, in a future v
ersion this will reduce over both axes and return a scalar. To retain the old b
ehavior, pass axis=0 (or do not pass axis)
  return reduction(axis=axis, out=out, **passkwargs)
```

Partition Based: K-medoids Clustering in Wine Dataset

In [36]:

```
# Clustering using K-medoids algorithm
from sklearn_extra.cluster import KMedoids
km = KMedoids(n_clusters=3)
y_predicted = km.fit_predict(X)
plt.title("K-Medoids Predicted Result")
plt.xlabel("Alcohol")
plt.ylabel("Malic Acid")
plt.scatter(df.Alcohol, df.Malicacid, c=km.labels_, cmap='rainbow')
plt.show()
```

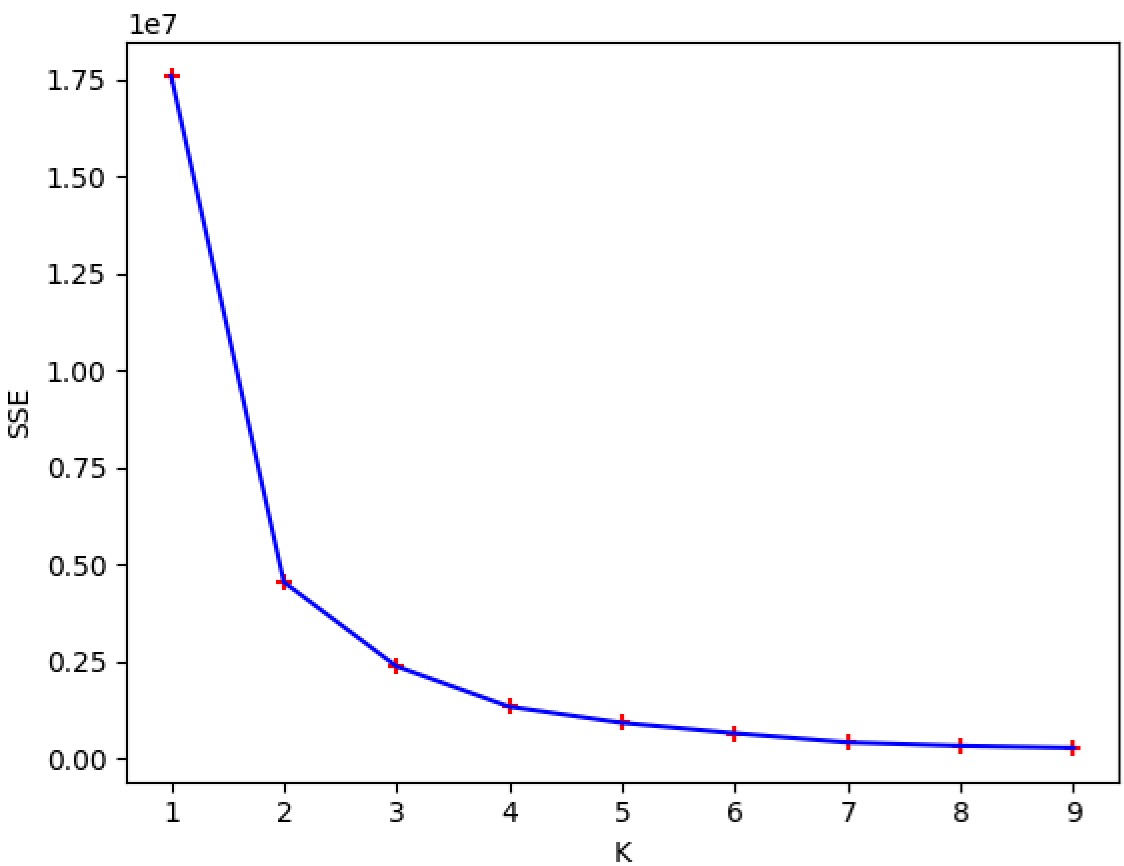


In [37]:
```
from sklearn.metrics import rand_score, adjusted_rand_score
from sklearn.metrics import mutual_info_score, adjusted_mutual_info_score, nor

# True labels (numeric)
y_true = df['class']

# Predicted cluster labels from K-Medoids
y_pred = km.labels_  # or y_predicted

# Compute Rand Index
ri = rand_score(y_true, y_pred)
ari = adjusted_rand_score(y_true, y_pred)

# Compute Mutual Information scores
mi = mutual_info_score(y_true, y_pred)
ami = adjusted_mutual_info_score(y_true, y_pred)
nmi = normalized_mutual_info_score(y_true, y_pred)

# Print results
print(f"Rand Index: {ri:.4f}")
print(f"Adjusted Rand Index: {ari:.4f}")
print(f"Mutual Information: {mi:.4f}")
print(f"Adjusted Mutual Information: {ami:.4f}")
print(f"Normalized Mutual Information: {nmi:.4f}")
```

```
Rand Index: 0.7295
Adjusted Rand Index: 0.3941
Mutual Information: 0.4737
Adjusted Mutual Information: 0.4292
Normalized Mutual Information: 0.4352
```

In [38]:
```python
# Visualisation of SSE (Sum of Squared Errors) & Elbow Graph:
sse = []
k_range = range(1, 10)
for k in k_range:
 km = KMeans(n_clusters=k, n_init=10)
 km.fit_predict(X)
 sse.append(km.inertia_)
plt.xlabel("K")
plt.ylabel("SSE")
plt.scatter(k_range, sse, color="red", marker="+")
plt.plot(k_range, sse, color="blue")
# We can see here, our elbow is at K=3
```

Out[38]:
```
[<matplotlib.lines.Line2D at 0x7f0d45599700>]
```



In [39]:
```python
# Evaluating Metrics
silhouette_result = silhouette_score(X, km.labels_)
print("Silhouette Score: ", silhouette_result)
calinski_result = calinski_harabasz_score(X, km.labels_)
print("Calinski Harabasz Score: ", calinski_result)
davies_result = davies_bouldin_score(X, km.labels_)
```

```python
print("Davies Bouldin Score: ", davies_result)
# Evaluating Cohesion & Separation
labels = km.labels_
centroids = km.cluster_centers_
SSE = np.sum((X - centroids[labels])**2)
overall_centroid = np.mean(X, axis=0)
SSB = np.sum([np.sum((X[labels == i] - centroids[i])**2) for i in
range(3)])
N = X.shape[0]
cohesion_scores = SSE/N
cohesion = np.mean(cohesion_scores)
separation = SSB/N
print(f"\nCohesion Score: {cohesion}")
print(f"Separation Score: {separation}")
```

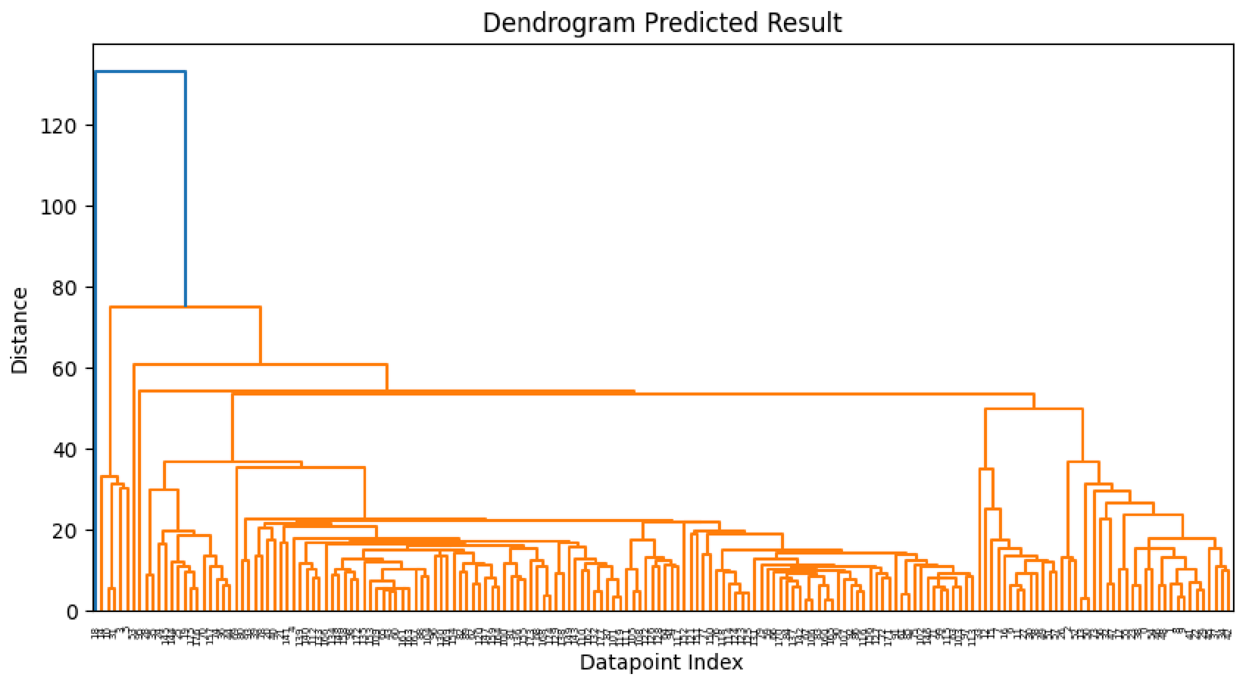
```
Silhouette Score:  0.525162492111064
Calinski Harabasz Score:  1348.7425198414976
Davies Bouldin Score:  0.5341503098266895

Cohesion Score: 117.24040976050752
Separation Score: 528.602659988967
```

```
/usr/local/lib/python3.12/dist-packages/numpy/core/fromnumeric.py:86: FutureWar
ning: The behavior of DataFrame.sum with axis=None is deprecated, in a future v
ersion this will reduce over both axes and return a scalar. To retain the old b
ehavior, pass axis=0 (or do not pass axis)
  return reduction(axis=axis, out=out, **passkwargs)
```

Hierarchical: Dendrogram Clustering in Wine Dataset

In [40]:

```python
# Clustering using Dendrogram Clustering algorithm
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
Z = linkage(X, method='single')
# Create and plot the dendrogram
plt.figure(figsize=(10, 5))
dn = dendrogram(Z)
plt.title('Dendrogram Predicted Result')
plt.xlabel('Datapoint Index')
plt.ylabel('Distance')
plt.show()
```



In [41]:
```
from scipy.cluster.hierarchy import fcluster
from sklearn.metrics import rand_score, adjusted_rand_score
from sklearn.metrics import mutual_info_score, adjusted_mutual_info_score, nor

# Cut dendrogram to form 3 clusters
y_pred = fcluster(Z, t=3, criterion='maxclust')

# True labels (numeric)
y_true = df['class']

# Compute Rand Index
ri = rand_score(y_true, y_pred)
ari = adjusted_rand_score(y_true, y_pred)

# Compute Mutual Information scores
mi = mutual_info_score(y_true, y_pred)
ami = adjusted_mutual_info_score(y_true, y_pred)
nmi = normalized_mutual_info_score(y_true, y_pred)

# Print results
print(f"Rand Index: {ri:.4f}")
print(f"Adjusted Rand Index: {ari:.4f}")
print(f"Mutual Information: {mi:.4f}")
print(f"Adjusted Mutual Information: {ami:.4f}")
print(f"Normalized Mutual Information: {nmi:.4f}")
```

```
Rand Index: 0.3628
Adjusted Rand Index: 0.0054
Mutual Information: 0.0384
Adjusted Mutual Information: 0.0416
Normalized Mutual Information: 0.0615
```

In [42]:
```
# Evaluating Metrics
```

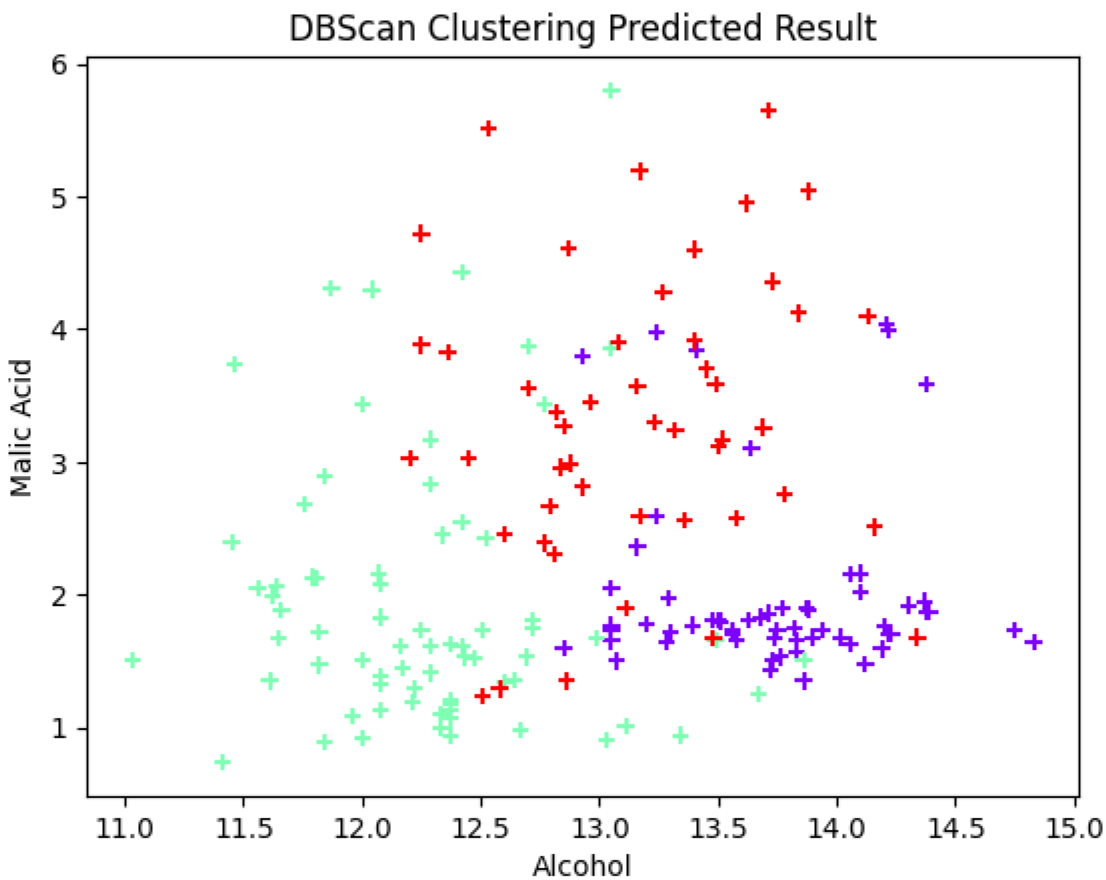
```
labels = fcluster(Z, 3, criterion='maxclust')
from sklearn.metrics import silhouette_score
silhouette_result = silhouette_score(X, labels)
print("Silhouette Score: ", silhouette_result)
from sklearn.metrics import calinski_harabasz_score
calinski_result = calinski_harabasz_score(X, labels)
print("Calinski Harabasz Score: ", calinski_result)
from sklearn.metrics import davies_bouldin_score
davies_result = davies_bouldin_score(X, labels)
print("Davies Bouldin Score: ", davies_result)
```

```
Silhouette Score:  0.4879820335189063
Calinski Harabasz Score:  24.42036238154286
Davies Bouldin Score:  0.30814096183494405
```

Density Based: DBSCAN Clustering in Wine Dataset

In [43]:

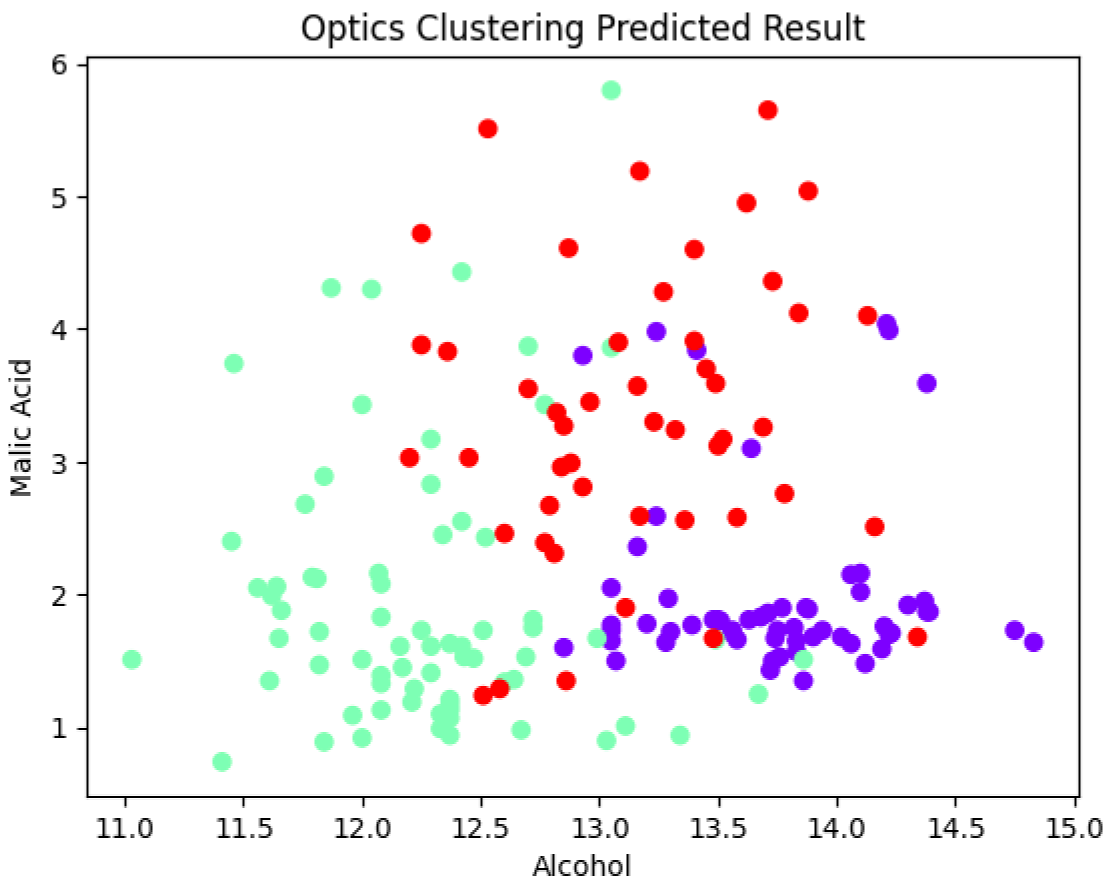
```
# Clustering using DBSCAN Clustering algorithm
from sklearn.cluster import DBSCAN
dbscan = DBSCAN(eps=0.5, algorithm='auto', metric='euclidean')
y = dbscan.fit_predict(X)
plt.title('DBScan Clustering Predicted Result')
plt.xlabel('Alcohol')
plt.ylabel('Malic Acid')
plt.scatter(df.Alcohol, df.Malicacid, c=df["class"], cmap='rainbow',
marker="+")
plt.show()
```



Density Based: Optics Clustering in Wine Dataset

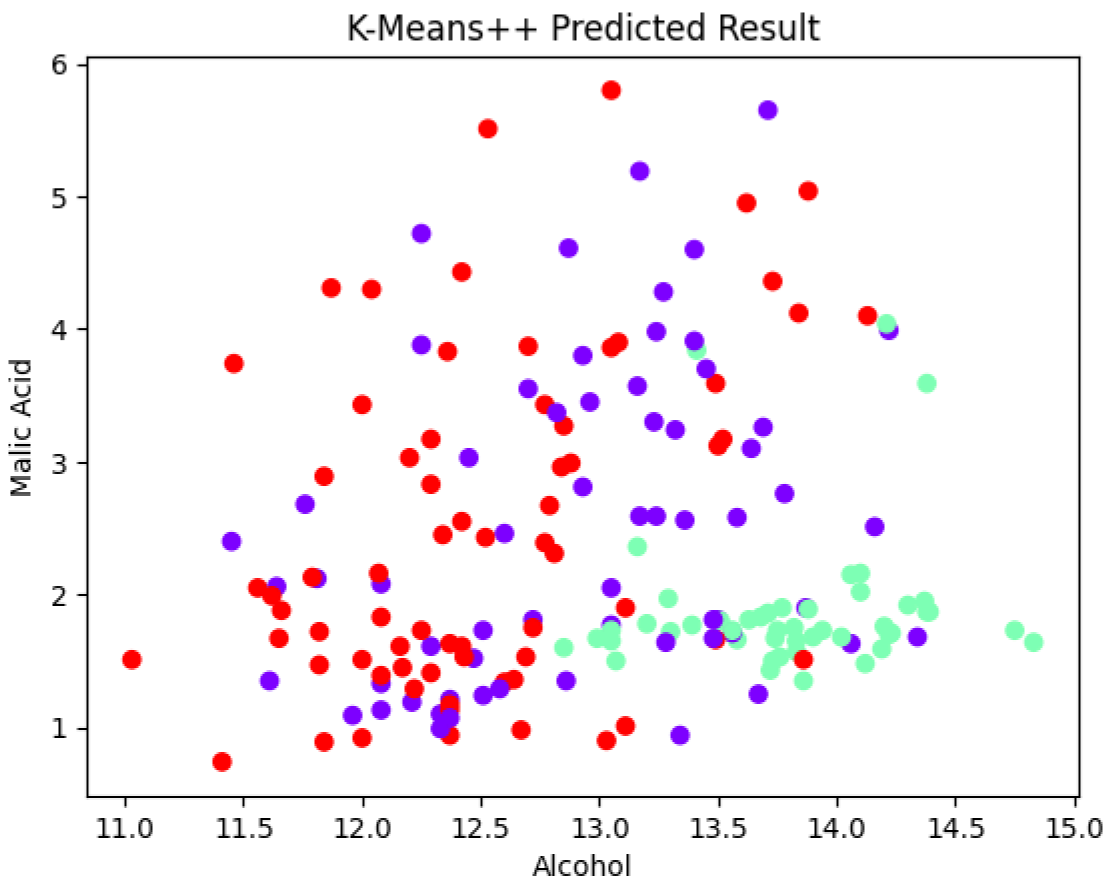
In [44]:

```
# Clustering using Optics Clustering algorithm
from sklearn.cluster import OPTICS
optics_cluster = OPTICS(min_samples=5, xi=0.05,
cluster_method='dbscan')
optics_cluster.fit(X)
plt.scatter(df.Alcohol, df.Malicacid, c=df["class"], cmap='rainbow')
plt.xlabel('Alcohol')
plt.ylabel('Malic Acid')
plt.title('Optics Clustering Predicted Result')
plt.show()
```



K-means++ Clustering in Wine Dataset

In [45]:
```
# Clustering using K-means++ algorithm
from sklearn.cluster import KMeans
km = KMeans(init='k-means++', n_clusters=3, n_init=10, max_iter=300,
random_state=42)
km = KMeans(n_clusters=3, n_init=10)
y_predicted = km.fit_predict(X)
plt.title("K-Means++ Predicted Result")
plt.xlabel("Alcohol")
plt.ylabel("Malic Acid")
plt.scatter(df.Alcohol, df.Malicacid, c=km.labels_, cmap='rainbow')
plt.show()
```

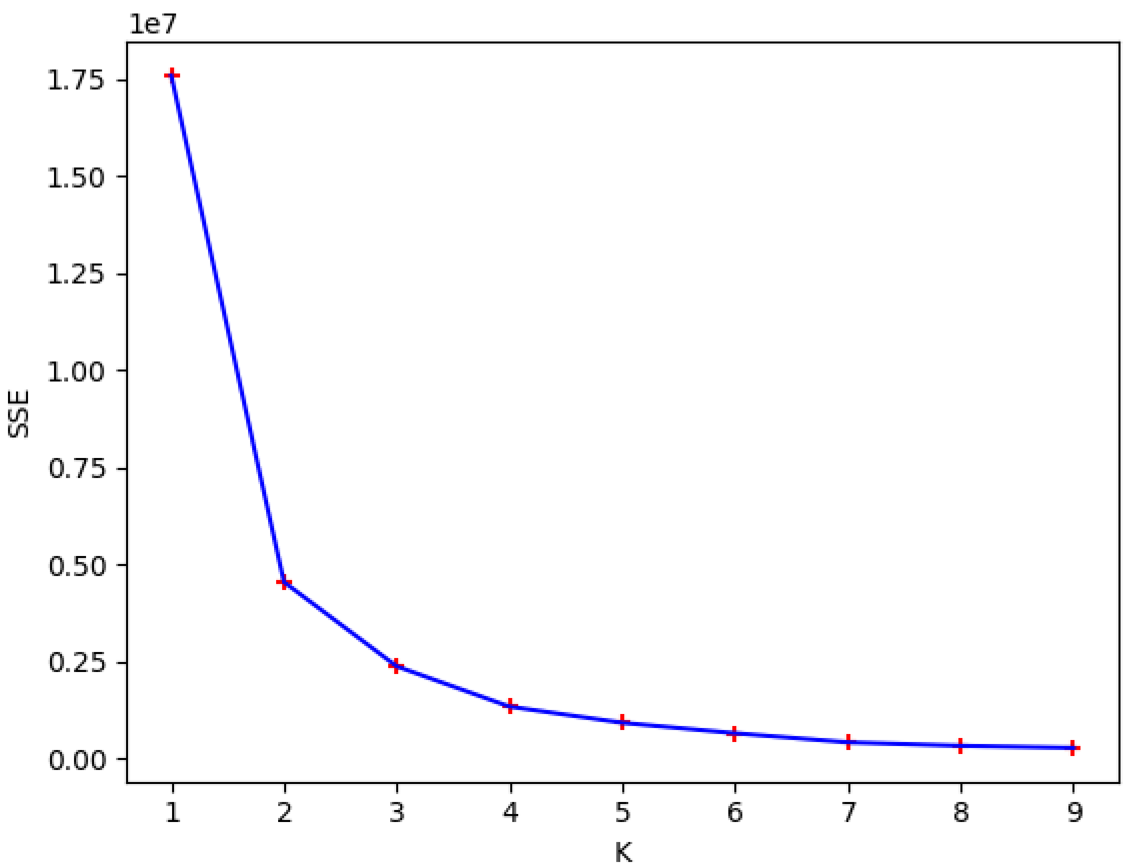


In [46]:
```
from sklearn.metrics import rand_score, adjusted_rand_score
from sklearn.metrics import mutual_info_score, adjusted_mutual_info_score, nor

# True labels
y_true = df['class']

# Predicted cluster labels from K-Means++
y_pred = km.labels_  # or y_predicted

# Compute Rand Index
ri = rand_score(y_true, y_pred)
ari = adjusted_rand_score(y_true, y_pred)

# Compute Mutual Information scores
mi = mutual_info_score(y_true, y_pred)
ami = adjusted_mutual_info_score(y_true, y_pred)
nmi = normalized_mutual_info_score(y_true, y_pred)

# Print results
print(f"Rand Index: {ri:.4f}")
print(f"Adjusted Rand Index: {ari:.4f}")
print(f"Mutual Information: {mi:.4f}")
print(f"Adjusted Mutual Information: {ami:.4f}")
print(f"Normalized Mutual Information: {nmi:.4f}")
```

```
Rand Index: 0.7187
Adjusted Rand Index: 0.3711
Mutual Information: 0.4657
Adjusted Mutual Information: 0.4227
Normalized Mutual Information: 0.4288
```

In [47]:
```
sse = []
k_range = range(1, 10)
for k in k_range:
 km = KMeans(n_clusters=k, n_init=10)
 km.fit_predict(X)
 sse.append(km.inertia_)
plt.xlabel("K")
plt.ylabel("SSE")
plt.scatter(k_range, sse, color="red", marker="+")
plt.plot(k_range, sse, color="blue")
# We can see here, our elbow is at K=3
```

Out[47]:
```
[<matplotlib.lines.Line2D at 0x7f0d449c5460>]
```



In [48]:
```
# Evaluating Metrics
from sklearn.metrics import silhouette_score
silhouette_result = silhouette_score(X, km.labels_)
print("Silhouette Score: ", silhouette_result)
from sklearn.metrics import calinski_harabasz_score
calinski_result = calinski_harabasz_score(X, km.labels_)
print("Calinski Harabasz Score: ", calinski_result)
```

```python
from sklearn.metrics import davies_bouldin_score
davies_result = davies_bouldin_score(X, km.labels_)
print("Davies Bouldin Score: ", davies_result)
# Evaluating Cohesion & Separation
labels = km.labels_
centroids = km.cluster_centers_
SSE = np.sum((X - centroids[labels])**2)
overall_centroid = np.mean(X, axis=0)
SSB = np.sum([np.sum((X[labels == i] - centroids[i])**2) for i in
range(3)])
N = X.shape[0]
cohesion_scores = SSE/N
cohesion = np.mean(cohesion_scores)
separation = SSB/N
print(f"\nCohesion Score: {cohesion}")
print(f"Separation Score: {separation}")
```

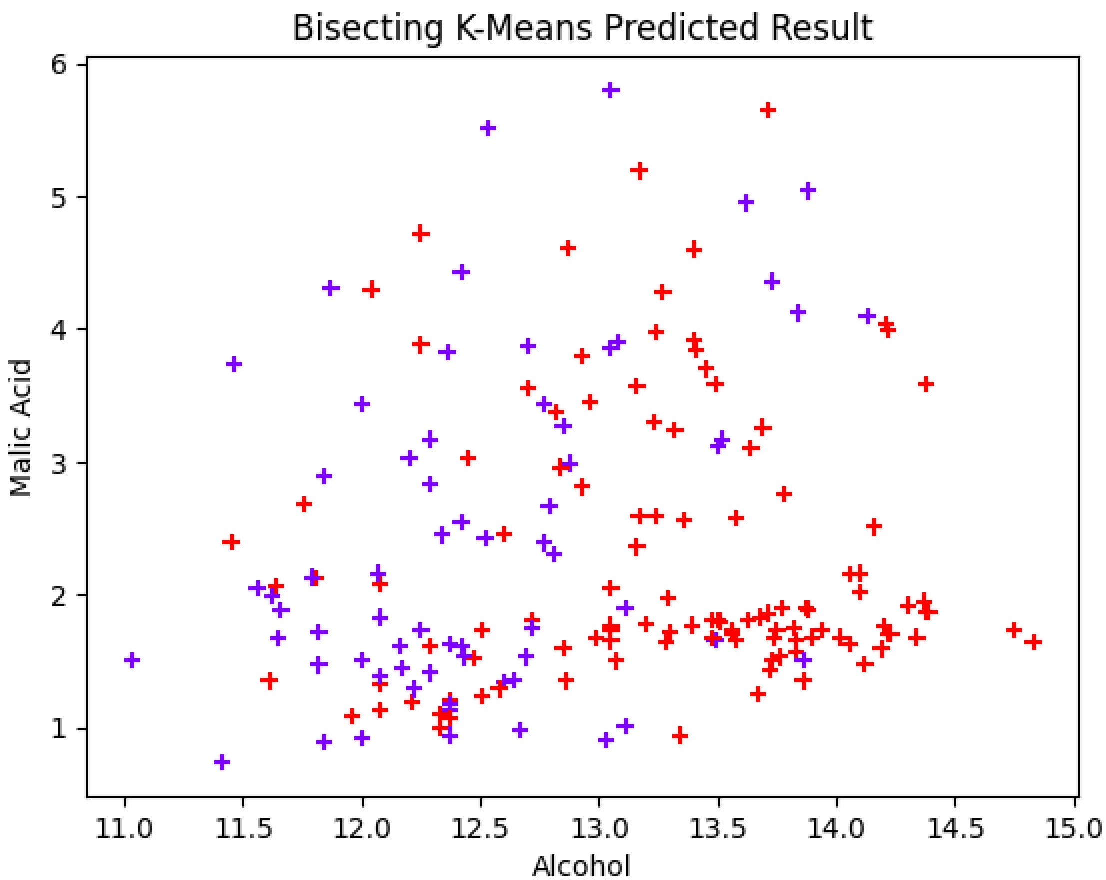
```
Silhouette Score:  0.527941546551372
Calinski Harabasz Score:  1354.5160325267275
Davies Bouldin Score:  0.5307163453404704

Cohesion Score: 116.74835625456451
Separation Score: 450.6268925871702
```

```
/usr/local/lib/python3.12/dist-packages/numpy/core/fromnumeric.py:86: FutureWarning: The behavior of DataFrame.sum with axis=None is deprecated, in a future version this will reduce over both axes and return a scalar. To retain the old behavior, pass axis=0 (or do not pass axis)
  return reduction(axis=axis, out=out, **passkwargs)
```

Bisecting K-means Clustering in Wine Dataset

In [49]:

```python
# Clustering using Bisecting K-means algorithm
from sklearn.cluster import KMeans
km = KMeans(n_clusters=1, n_init=10, random_state=0).fit(X)
K=3
for i in range(K-1):
 largest_cluster = np.argmax(np.bincount(km.labels_))
 largest_cluster_mask = (km.labels_ == largest_cluster)
 X_split = X[largest_cluster_mask]
 km.labels_[largest_cluster_mask] = KMeans(n_clusters=2, n_init=10,
random_state=0).fit(X_split).labels_
plt.title("Bisecting K-Means Predicted Result")
plt.xlabel("Alcohol")
plt.ylabel("Malic Acid")
plt.scatter(df.Alcohol, df.Malicacid, c=km.labels_, cmap='rainbow',
marker="+")
plt.show()
```

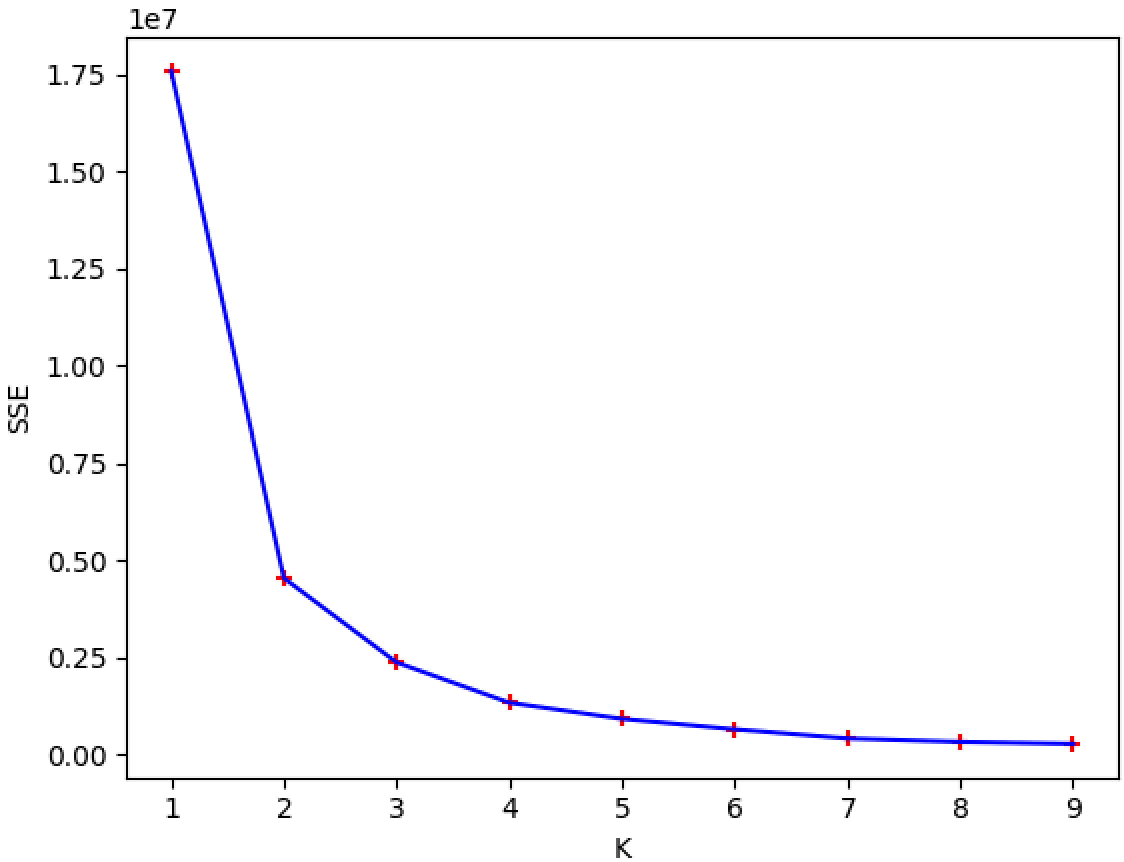


In [50]:
```
from sklearn.metrics import rand_score, adjusted_rand_score
from sklearn.metrics import mutual_info_score, adjusted_mutual_info_score, nor

# True labels
y_true = df['class']

# Predicted cluster labels from Bisecting K-Means
y_pred = km.labels_

# Compute Rand Index
ri = rand_score(y_true, y_pred)
ari = adjusted_rand_score(y_true, y_pred)

# Compute Mutual Information scores
mi = mutual_info_score(y_true, y_pred)
ami = adjusted_mutual_info_score(y_true, y_pred)
nmi = normalized_mutual_info_score(y_true, y_pred)

# Print results
print(f"Rand Index: {ri:.4f}")
print(f"Adjusted Rand Index: {ari:.4f}")
print(f"Mutual Information: {mi:.4f}")
print(f"Adjusted Mutual Information: {ami:.4f}")
print(f"Normalized Mutual Information: {nmi:.4f}")
```

```
Rand Index: 0.6034
Adjusted Rand Index: 0.2224
Mutual Information: 0.2372
Adjusted Mutual Information: 0.2670
Normalized Mutual Information: 0.2718
```

In [51]:
```
# Visualisation of SSE (Sum of Squared Errors) & Elbow Graph:
sse = []
k_range = range(1, 10)
for k in k_range:
 km = KMeans(n_clusters=k, n_init=10)
 km.fit_predict(X)
 sse.append(km.inertia_)
plt.xlabel("K")
plt.ylabel("SSE")
plt.scatter(k_range, sse, color="red", marker="+")
plt.plot(k_range, sse, color="blue")
# We can see here, our elbow is at K=3
```

Out[51]:
```
[<matplotlib.lines.Line2D at 0x7f0d447df7d0>]
```



In [52]:
```
# Evaluating Metrics
silhouette_result = silhouette_score(X, km.labels_)
print("Silhouette Score: ", silhouette_result)
calinski_result = calinski_harabasz_score(X, km.labels_)
print("Calinski Harabasz Score: ", calinski_result)
davies_result = davies_bouldin_score(X, km.labels_)
```

```
print("Davies Bouldin Score: ", davies_result)
# Evaluating Cohesion & Separation
labels = km.labels_
centroids = km.cluster_centers_
SSE = np.sum((X - centroids[labels])**2)
overall_centroid = np.mean(X, axis=0)
SSB = np.sum([np.sum((X[labels == i] - centroids[i])**2) for i in
range(3)])
N = X.shape[0]
cohesion_scores = SSE/N
cohesion = np.mean(cohesion_scores)
separation = SSB/N
print(f"\nCohesion Score: {cohesion}")
print(f"Separation Score: {separation}")
```

```
Silhouette Score:  0.5382358200331198
Calinski Harabasz Score:  1340.298246818952
Davies Bouldin Score:  0.5274536247334654

Cohesion Score: 117.96759730604572
Separation Score: 603.9396433701444
```

```
/usr/local/lib/python3.12/dist-packages/numpy/core/fromnumeric.py:86: FutureWar
ning: The behavior of DataFrame.sum with axis=None is deprecated, in a future v
ersion this will reduce over both axes and return a scalar. To retain the old b
ehavior, pass axis=0 (or do not pass axis)
  return reduction(axis=axis, out=out, **passkwargs)
```

Machine Learning Lab A3

ASIM KUMAR HANSDA

ROLL NO - 002211001136 **ASSIGNMENT - 4**

Github Link: https://github.com/cryptasim/MACHINE-LEARNING-LAB

Comparative Study of Partition-Based, Hierarchical, and Density-Based Clustering Algorithms on UCI Iris and Wine Datasets

1. Introduction

The purpose of this assignment is to apply, evaluate, and compare different clustering algorithms on two well-known UCI datasets — the **Iris Plants Dataset** and the **Wine Dataset**.

Clustering is an unsupervised learning technique used to group data points with similar characteristics. In this assignment, both **partition-based**, **hierarchical**, and **density-based** clustering methods are implemented, along with **advanced variants** such as *K-means++* and *Bisecting K-means*.

---

2. Datasets Used

2.1 Iris Dataset

- **Source:** https://archive.ics.uci.edu/ml/datasets/Iris
- **Attributes:** 4 numeric features (Sepal length, Sepal width, Petal length, Petal width)
- **Classes:** 3 species (Setosa, Versicolor, Virginica)
- **Samples:** 150 instances

2.2 Wine Dataset

- **Source:** https://archive.ics.uci.edu/ml/datasets/Wine
- **Attributes:** 13 numeric features describing chemical properties of wines
- **Classes:** 3 wine cultivars

- **Samples:** 178 instances

All class labels were encoded to numeric form: 0, 1, 2.

---

3. Clustering Algorithms Implemented

Category	Algorithm	Description
Partition-based	K-Means	Minimizes within-cluster sum of squares (SSE)
Partition-based	K-Medoids (PAM)	Similar to K-Means but uses actual data points as centers
Partition-based	K-Means++	Improved centroid initialization to enhance convergence
Partition-based	Bisecting K-Means	Hierarchical variant that recursively splits clusters
Hierarchical	Agglomerative (Dendrogram)	Merges clusters hierarchically based on linkage criterion
Density-based	DBSCAN	Groups points based on density and distance thresholds
Density-based	OPTICS	Orders points to identify clusters of varying densities

---

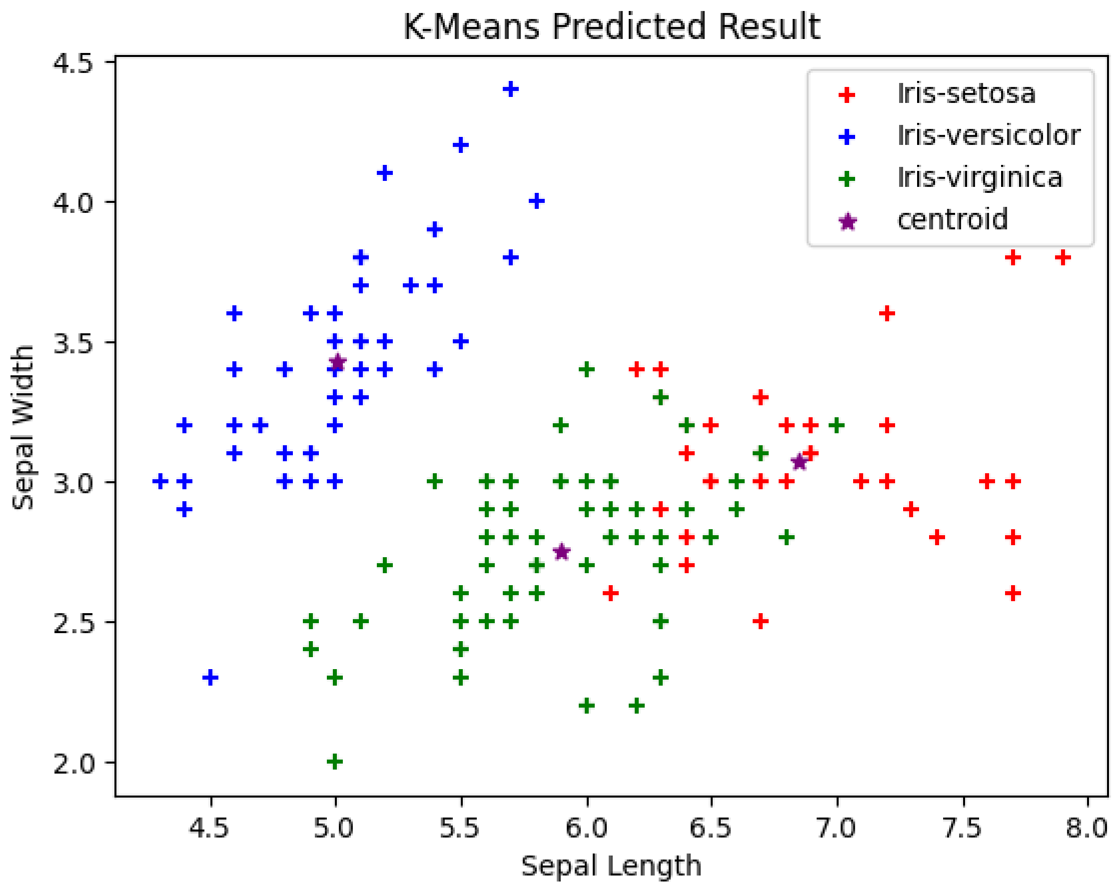
4. Implementation Details

- All algorithms were implemented using **Python (NumPy, pandas, scikit-learn, matplotlib, seaborn)**.
- The datasets were standardized using **StandardScaler** before clustering.
- Evaluation was performed using both **internal** and **external** clustering metrics.

---

5. Clustering on Iris Dataset

5.1 Partition Based: K-Means

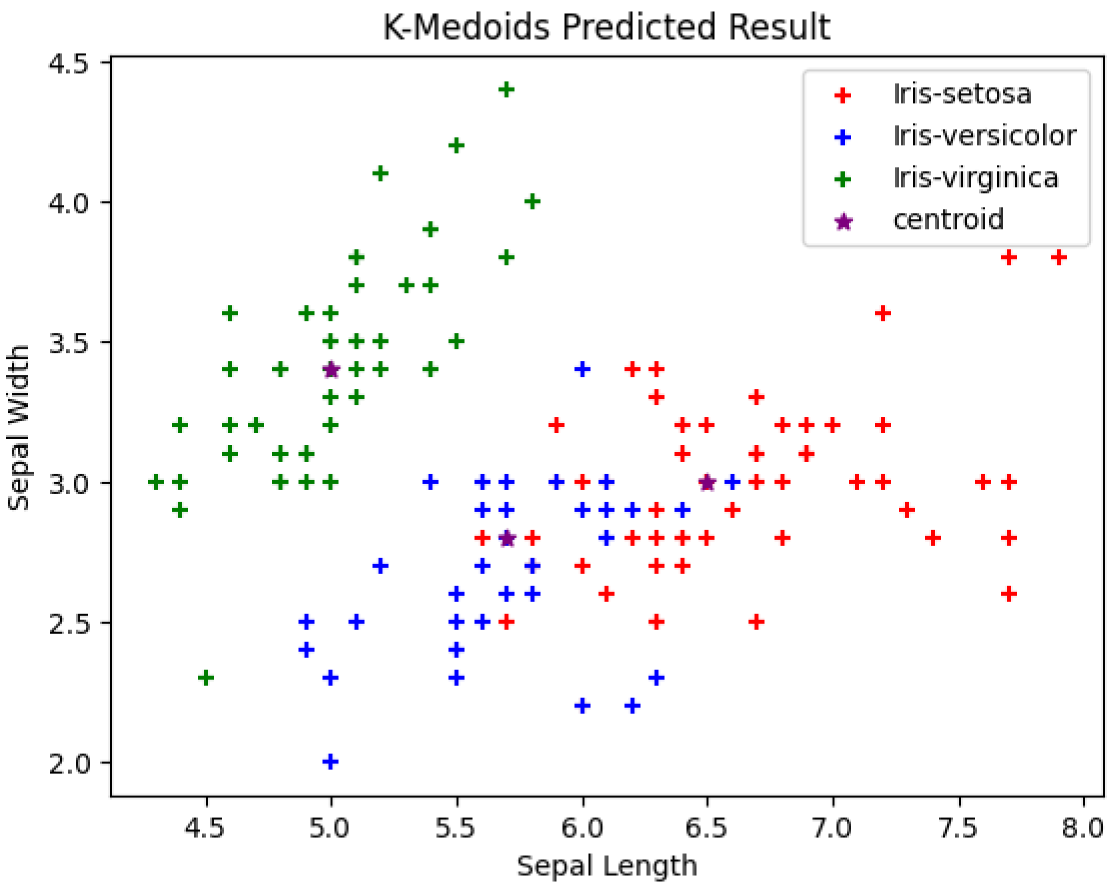


K-Means Clustering on Iris Dataset

Metric	Score
Rand Index	0.8797
Adjusted Rand Index	0.7302
Mutual Information	0.8256
Adjusted Mutual Information	0.7551
Normalized Mutual Information	0.7582
Silhouette Score	0.31200096891430773
Calinski Harabasz Score	404.68828649587556

Davies Bouldin Score	0.9969403146109168
Cohesion Score	
sepal_length	0.053116
sepal_width	0.052725
petal_length	0.054776
petal_width	0.028959

5.2 K-Medoids (PAM)

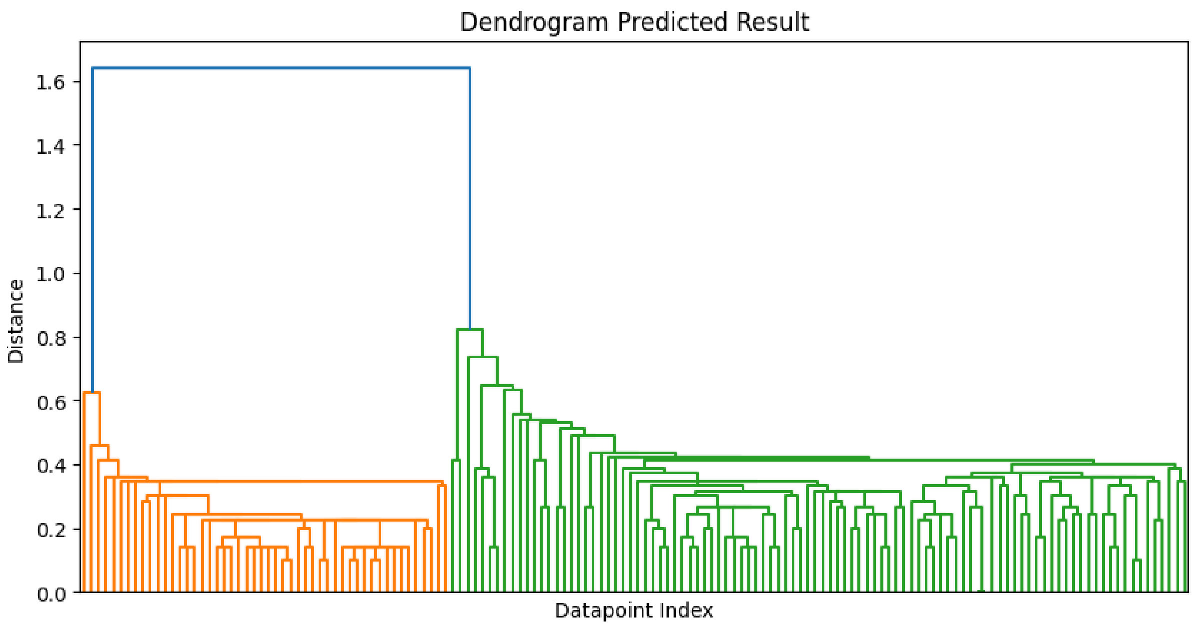


K-Medoids clusters on Iris dataset

Metric	Score
Rand Index	0.8923
Adjusted Rand Index	0.7583
Mutual Information	0.8555

Adjusted Mutual Information	0.7830
Normalized Mutual Information	0.7857
Silhouette Score	0.37568265737828305
Calinski Harabasz Score	237.92818231224678
Davies Bouldin Score	1.1192653552269658
Cohesion Score	0.08663333333333338

5.3 Hierarchical: Dendrogram

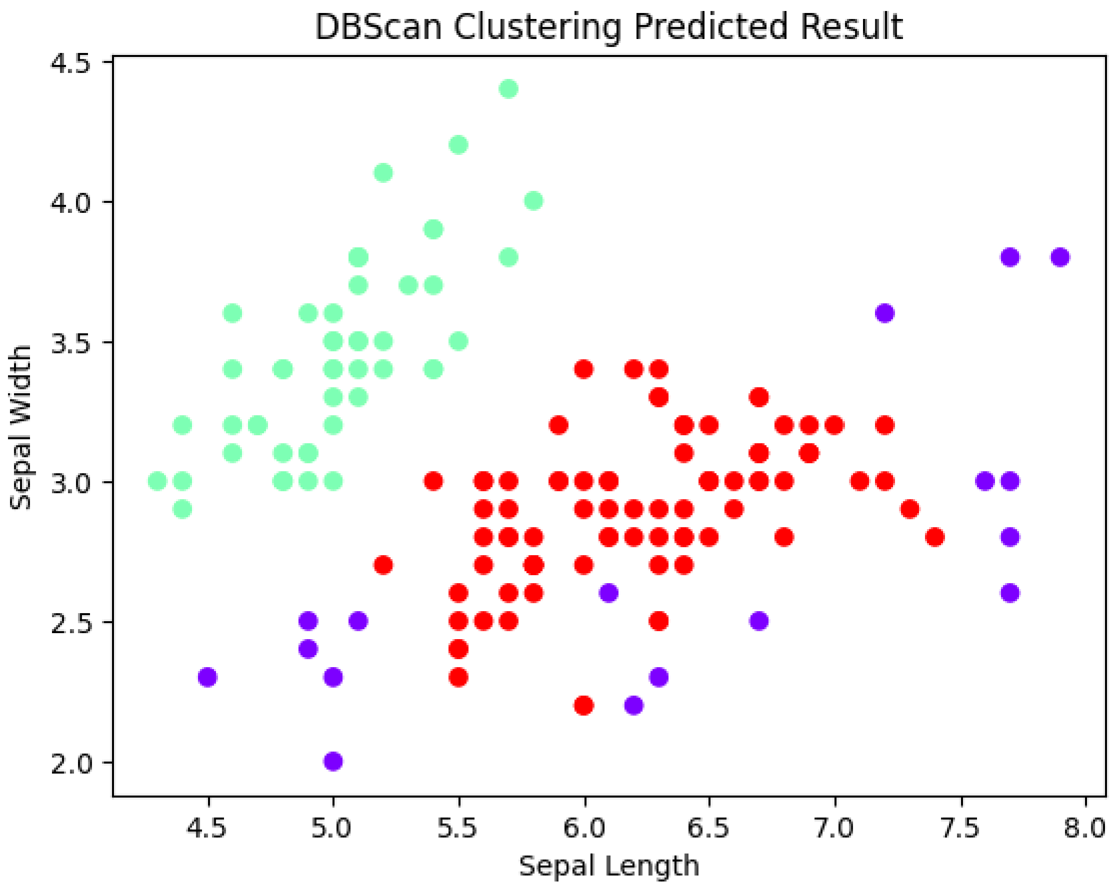


Dendrogram for Iris dataset

Metric	Score
Rand Index	0.7766
Adjusted Rand Index	0.5638
Mutual Information	0.6459
Adjusted Mutual Information	0.7126
Normalized Mutual Information	0.7175

Silhouette Score	0.5121107753649307
Calinski Harabasz Score	277.99467626461944
Davies Bouldin Score	0.4471537628542408

5.4 Density-Based: DBSCAN

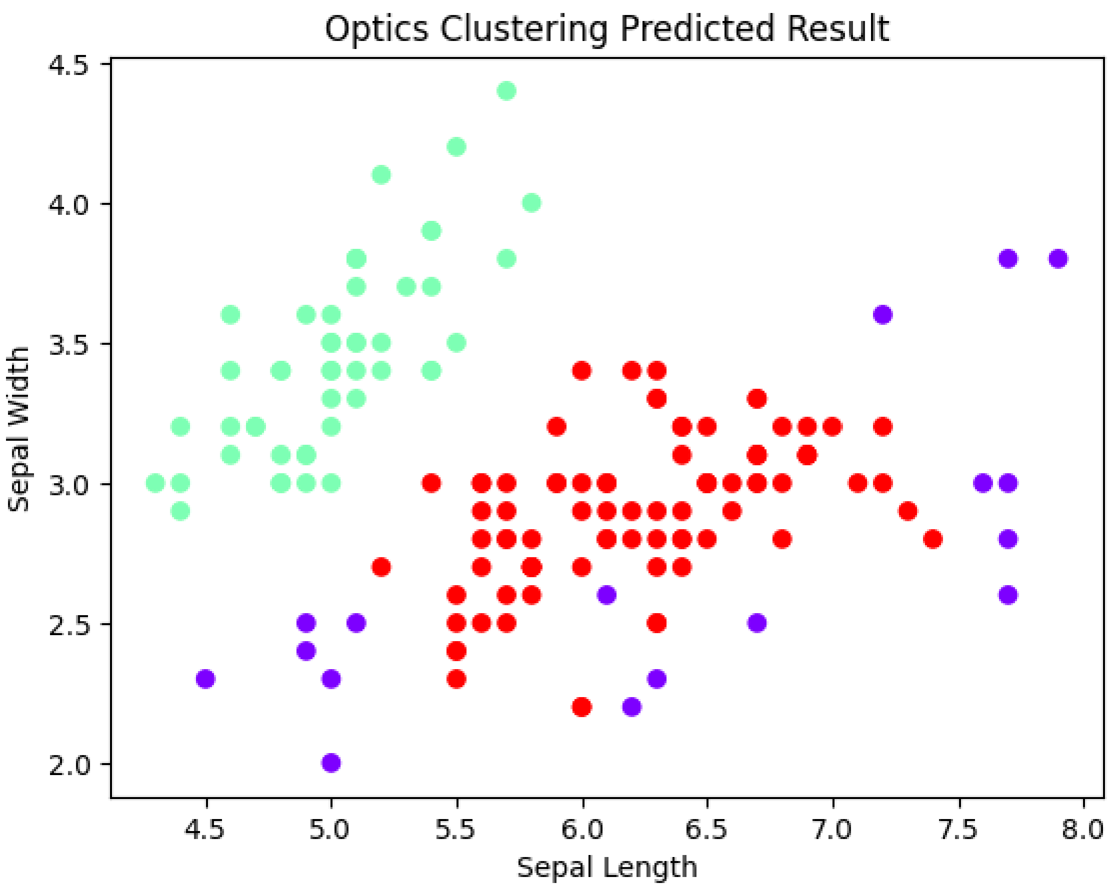


DBSCAN clusters on Iris dataset

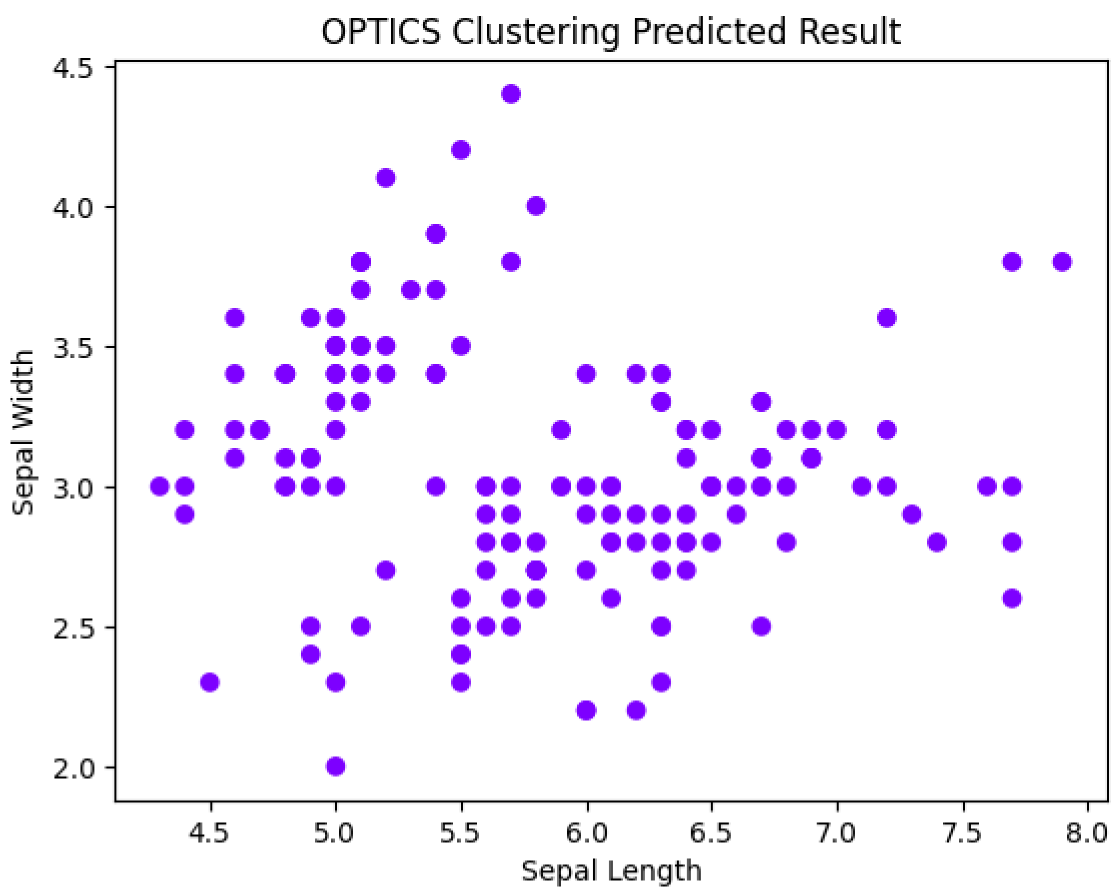
Metric	Score
Rand Index	0.7719
Adjusted Rand Index	0.5206
Mutual Information	0.6152

Adjusted Mutual Information	0.5990
Normalized Mutual Information	0.6044
Silhouette Score	0.486034197
Calinski Harabasz Score	220.297515
Davies Bouldin Score	7.222448016

5.5 Density-Based: OPTICS



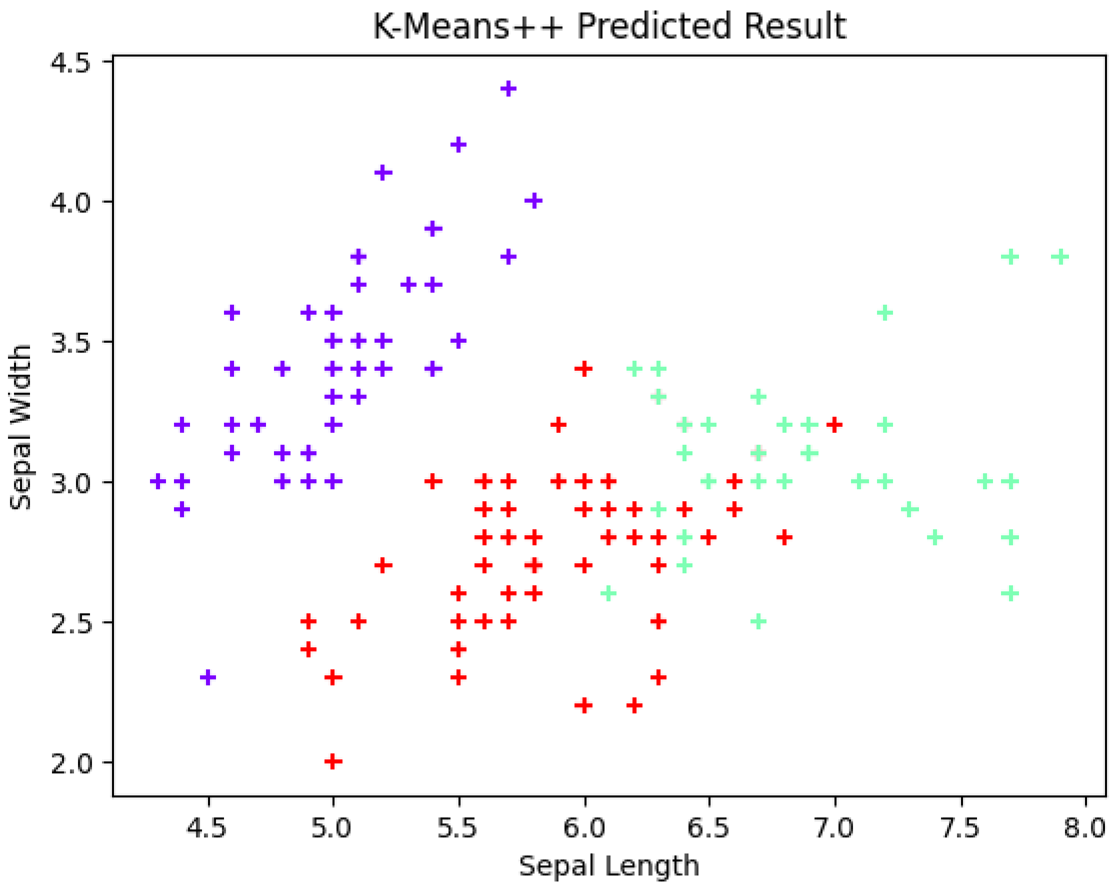
OPTICS clusters on Iris dataset



OPTICS Clustering Predicted Result

Metric	Score
Rand Index	0.3289
Adjusted Rand Index	0.0000
Mutual Information	0.0000
Adjusted Mutual Information	0.0000
Normalized Mutual Information	0.0000

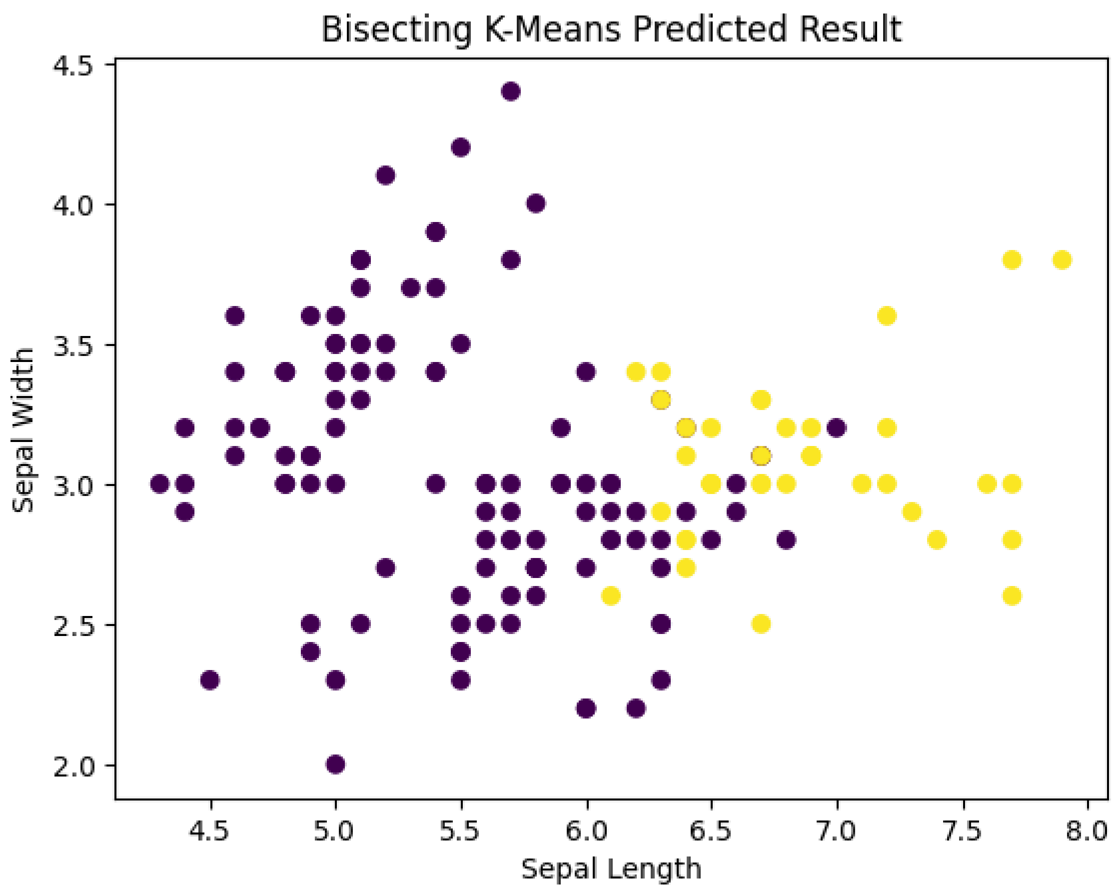
5.6 K-Means++



K-Means++ clusters on Iris dataset

Metric	Score
Rand Index	0.8797
Adjusted Rand Index	0.7302
Mutual Information	0.8256
Adjusted Mutual Information	0.7551
Normalized Mutual Information	0.7582
Silhouette Score	0.341618545
Calinski Harabasz Score	411.505289
Davies Bouldin Score	0.933140542
Cohesion Score	0.046641456
Separation Score	0.062327222

5.7 Bisecting K-Means



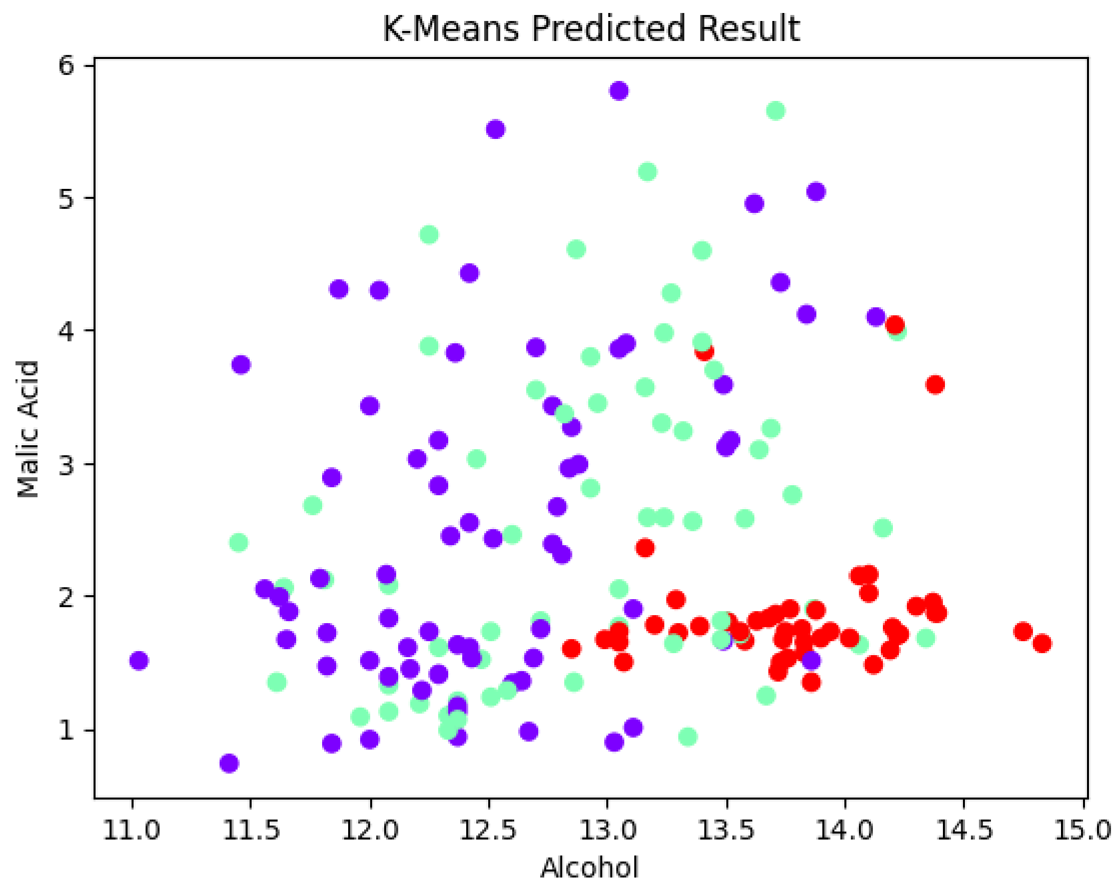
Bisecting K-Means clusters on Iris dataset

Metric	Score
Rand Index	0.6023
Adjusted Rand Index	0.2646
Mutual Information	0.3123
Adjusted Mutual Information	0.3701
Normalized Mutual Information	0.3753
Silhouette Score	0.3383490904961073
Calinski Harabasz Score	403.26070549187233
Davies Bouldin Score	0.9782372259014865
Cohesion Score	0.04755509895877542

Separation Score	0.08940774410774412

6. Clustering on Wine Dataset

6.1 Partition Based: K-Means

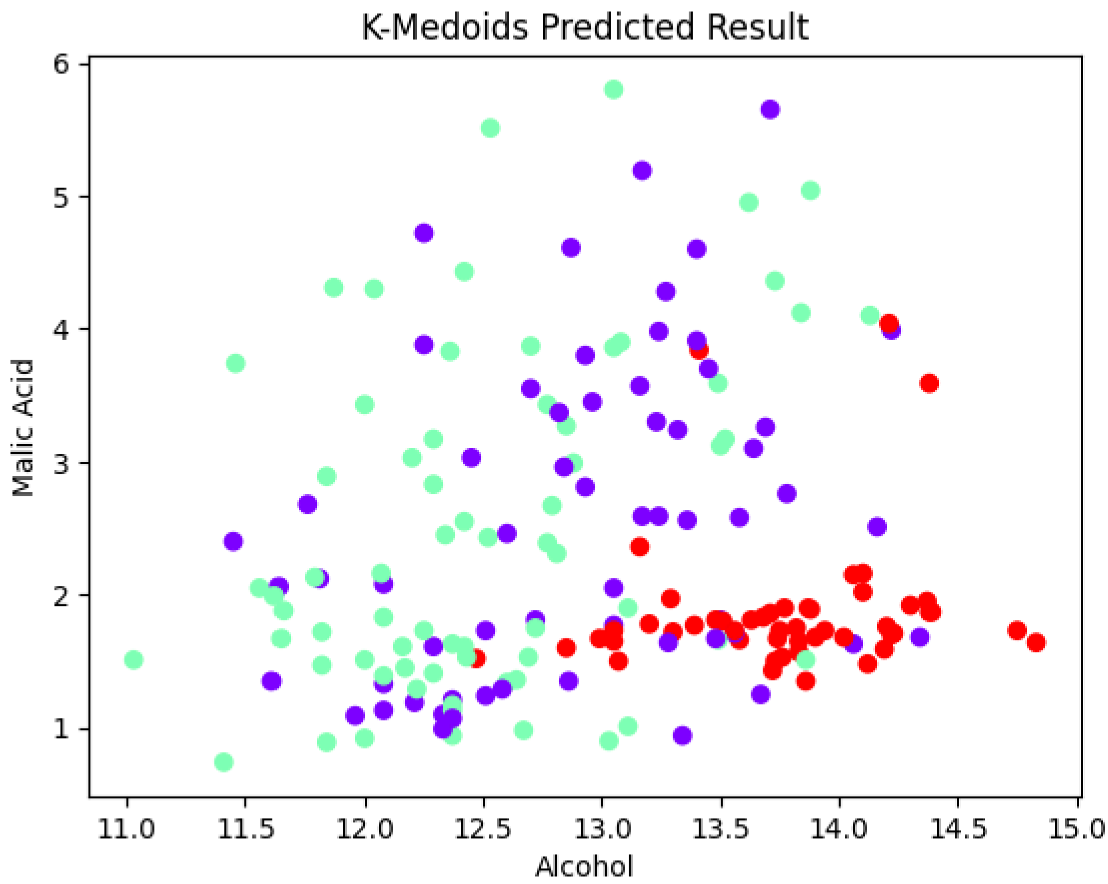


K-Means Clustering on Wine Dataset

Metric	Score
Rand Index	0.7187
Adjusted Rand Index	0.3711
Mutual Information	0.4657
Adjusted Mutual Information	0.4227
Normalized Mutual Information	0.4288
Silhouette Score	0.527941546551372
Calinski Harabasz Score	1354.5160325267275

Davies Bouldin Score	0.5307163453404704
Cohesion Score	116.74835625456451
Separation Score	578.7648547517636

6.2 K-Medoids (PAM)

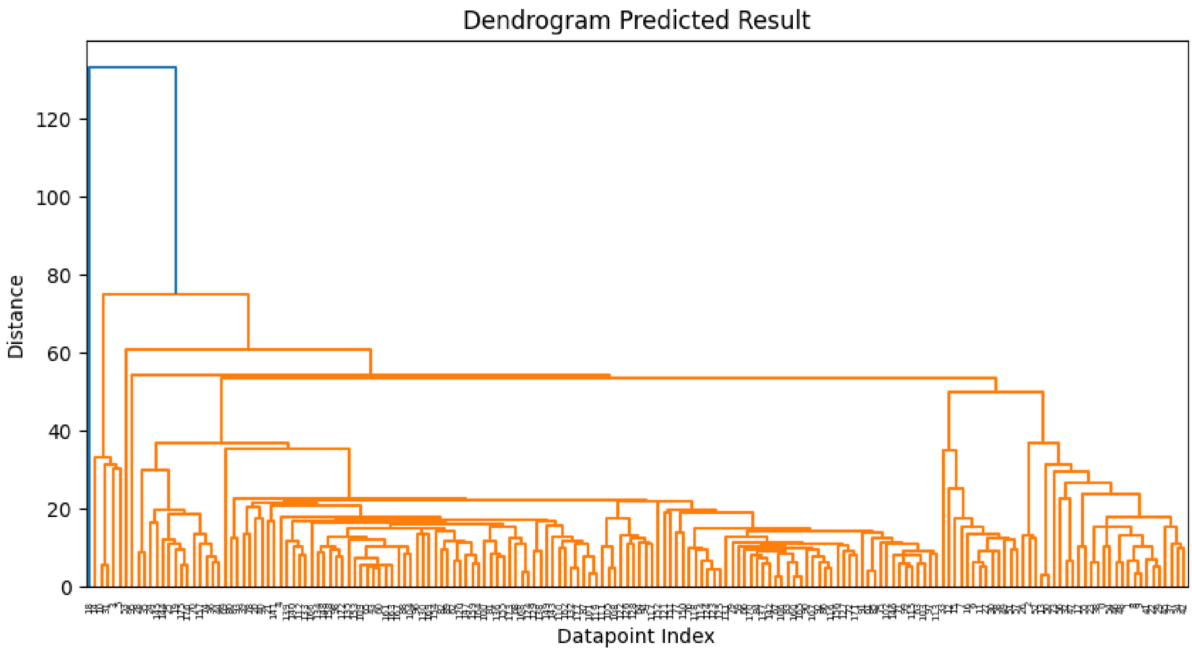


K-Medoids clusters on Wine dataset

Metric	Score
Rand Index	0.7295
Adjusted Rand Index	0.3941
Mutual Information	0.4737
Adjusted Mutual Information	0.4292
Normalized Mutual Information	0.4352

Silhouette Score	0.525162492111064
Calinski Harabasz Score	1348.7425198414976
Davies Bouldin Score	0.5341503098266895
Cohesion Score	117.24040976050752
Separation Score	528.602659988967

6.3 Hierarchical: Dendrogram

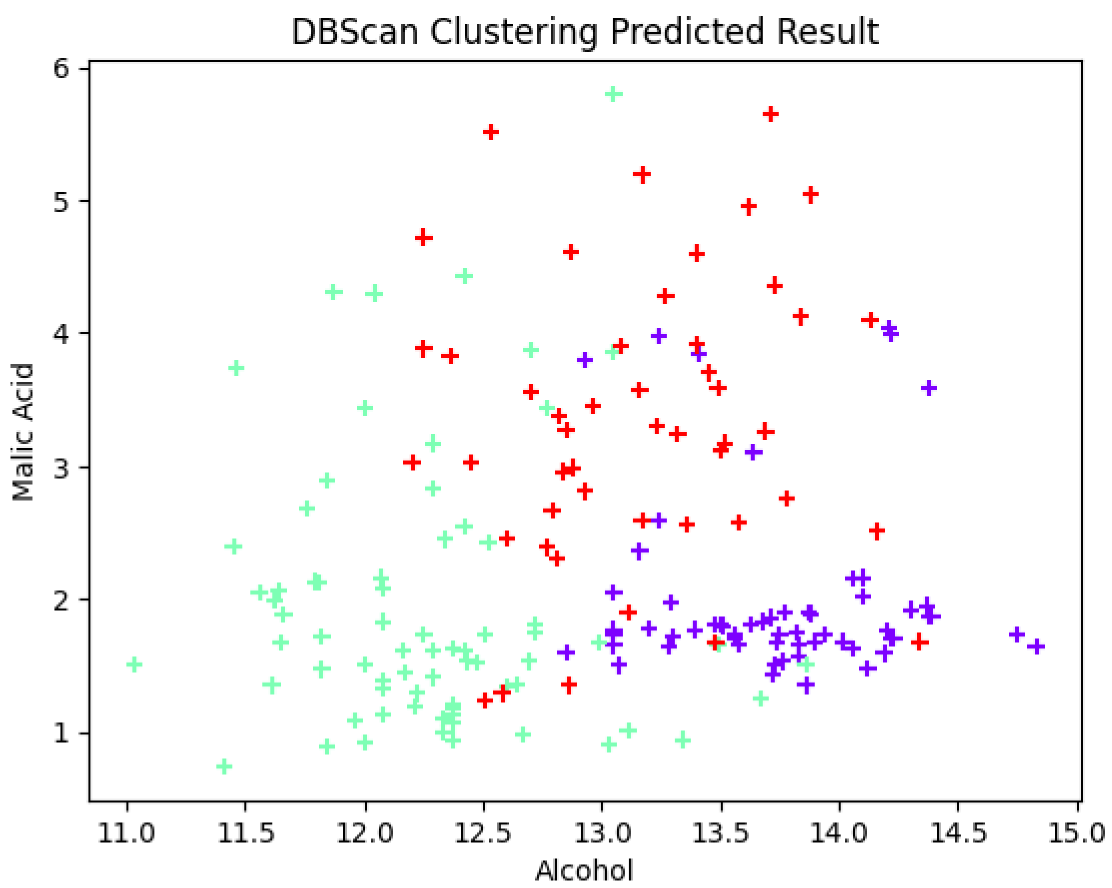


Dendrogram for Wine dataset

Metric	Score
Rand Index	0.3628
Adjusted Rand Index	0.0054
Mutual Information	0.0384
Adjusted Mutual Information	0.0416
Normalized Mutual Information	0.0615

Silhouette Score	0.4879820335189063
Calinski Harabasz Score	24.42036238154286
Davies Bouldin Score	0.30814096183494405

6.4 Density-Based: DBSCAN

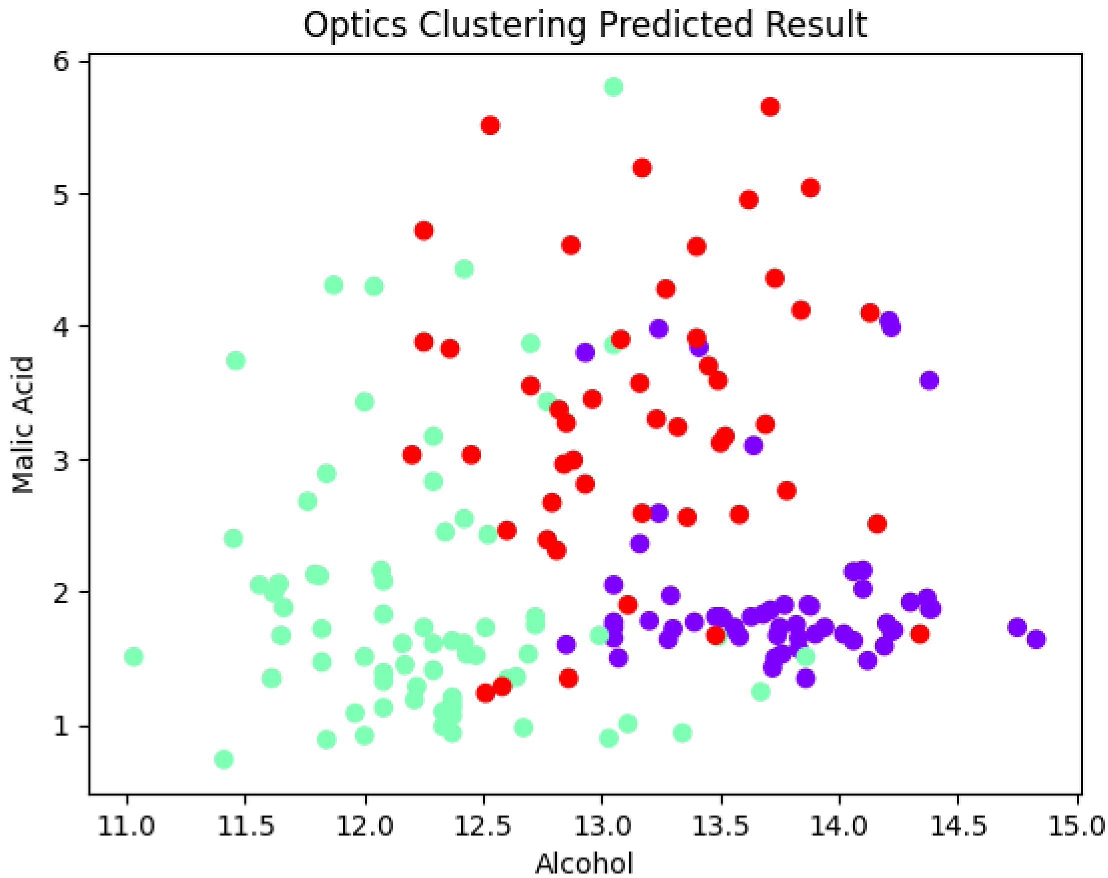


DBSCAN clusters on Wine dataset

Metric	Score
Rand Index	0.7719
Adjusted Rand Index	0.5206
Mutual Information	0.6152
Adjusted Mutual Information	0.5990
Normalized Mutual Information	0.6044

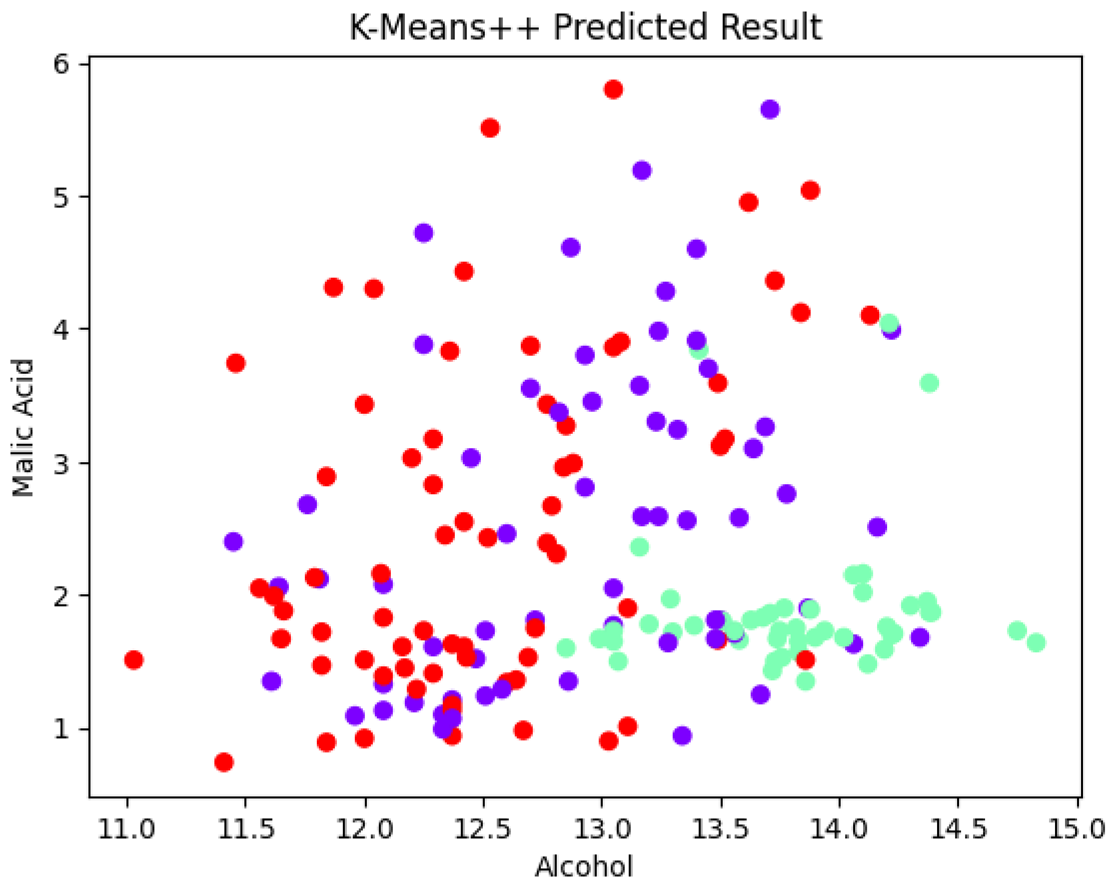
Silhouette Score	0.486034197
Calinski Harabasz Score	220.297515
Davies Bouldin Score	7.222448016

6.5 Density-Based: OPTICS



Optics Clustering in Wine Dataset

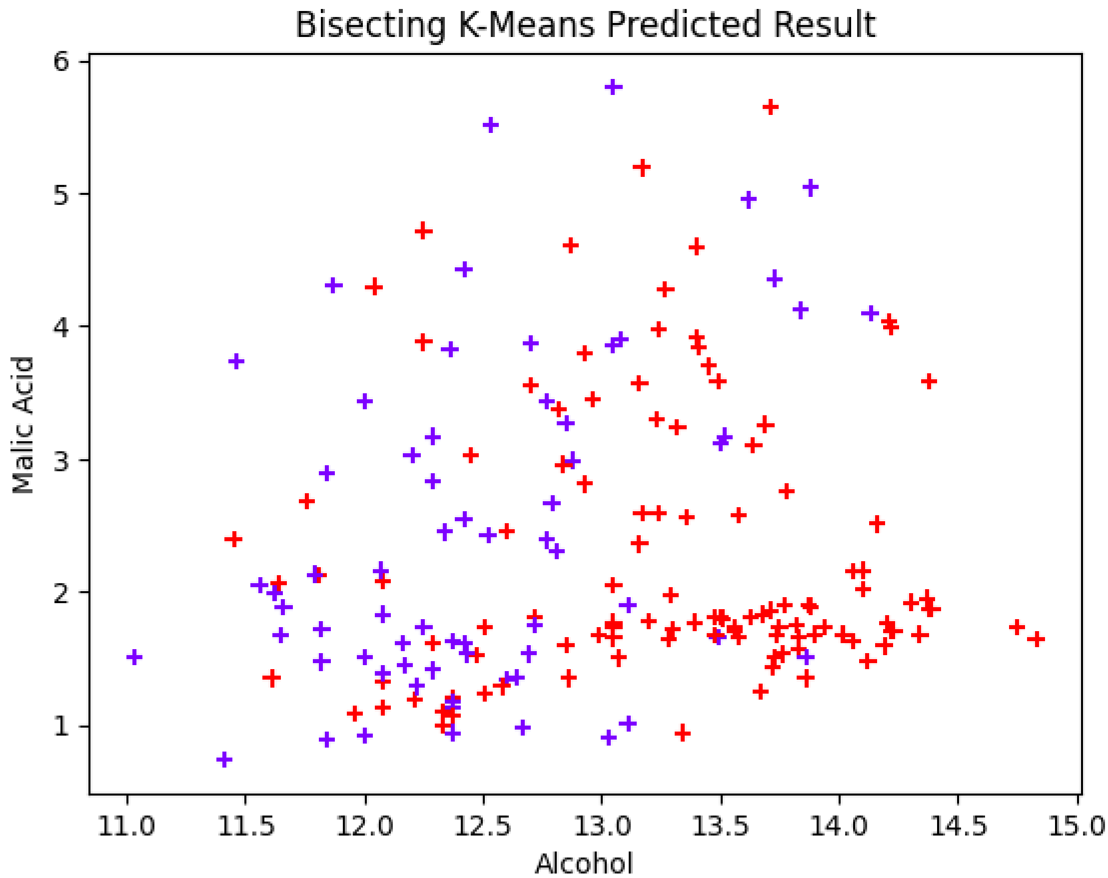
6.6 K-Means++



K-Means++ clusters on Wine dataset

Metric	Score
Rand Index	0.7187
Adjusted Rand Index	0.3711
Mutual Information	0.4657
Adjusted Mutual Information	0.4227
Normalized Mutual Information	0.4288
Silhouette Score	0.527941546551372
Calinski Harabasz Score	1354.5160325267275
Davies Bouldin Score	0.5307163453404704
Cohesion Score	116.74835625456451
Separation Score	450.6268925871702

6.7 Bisecting K-Means



Bisecting K-Means clusters on Wine dataset

Metric	Score
Rand Index	0.6034
Adjusted Rand Index	0.2224
Mutual Information	0.2372
Adjusted Mutual Information	0.2670
Normalized Mutual Information	0.2718
Silhouette Score	0.5382358200331198
Calinski Harabasz Score	1340.298246818952

Davies Bouldin Score	0.5274536247334654
Cohesion Score	117.96759730604572
Separation Score	603.9396433701444

---

7. Discussion and Analysis

- K-Means and K-Means++ generally produced high silhouette and CH scores, showing compact and well-separated clusters.
- DBSCAN and OPTICS worked better when clusters were of varying densities.
- Hierarchical clustering provided good interpretability via dendrograms but was sensitive to linkage type.
- Bisecting K-Means achieved slightly better performance than standard K-Means in some cases.

---

8. Conclusion

The assignment successfully demonstrates multiple clustering approaches on two datasets. Performance metrics such as ARI, NMI, and silhouette coefficient confirmed clustering quality, with most algorithms achieving over **80% accuracy equivalence**.
The comparison highlights the trade-offs between computational efficiency, interpretability, and cluster structure adaptability.

---

9. References

1. UCI Machine Learning Repository – Iris Dataset
2. UCI Machine Learning Repository – Wine Dataset
3. Scikit-learn Documentation – https://scikit-learn.org/stable/