Objective

The purpose of this notebook is to employ three unsupervised machine learning techniques to improve our comprehension of the morphological variations present within the Iris species.

This information can be valuable for various stakeholders, including botanists, researchers, and horticulturists, as it enhances our ability to categorize and interpret the diversity present in Iris flowers.

Import libraries and dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.cluster import KMeans, AgglomerativeClustering
import scipy.cluster.hierarchy as sch
from sklearn.mixture import GaussianMixture
from sklearn.metrics import silhouette_score
```

```
df = pd.read_csv('./Datasets/IRIS.csv')
```

```
df.head()
```

```
   sepal_length  sepal_width  petal_length  petal_width      species
0           5.1          3.5           1.4          0.2  Iris-setosa
1           4.9          3.0           1.4          0.2  Iris-setosa
2           4.7          3.2           1.3          0.2  Iris-setosa
3           4.6          3.1           1.5          0.2  Iris-setosa
4           5.0          3.6           1.4          0.2  Iris-setosa
```

```
df.dtypes
```

```
sepal_length    float64
sepal_width     float64
petal_length    float64
petal_width     float64
species          object
dtype: object
```

```
class_counts = df['species'].value_counts()

print("Class Counts:")
print(class_counts)
```

```
Class Counts:
species
```

```
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: count, dtype: int64
```

```
df.shape
```

```
(150, 5)
```

There are 150 observations, with features including sepal length, sepal width, petal length, and petal width. The dataset comprises three types of Iris flowers: Iris-setosa, Iris-versicolor, and Iris-virginica, each with 50 instances.

Check for missing values

```
df.isnull().sum()
```

```
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
species         0
dtype: int64
```

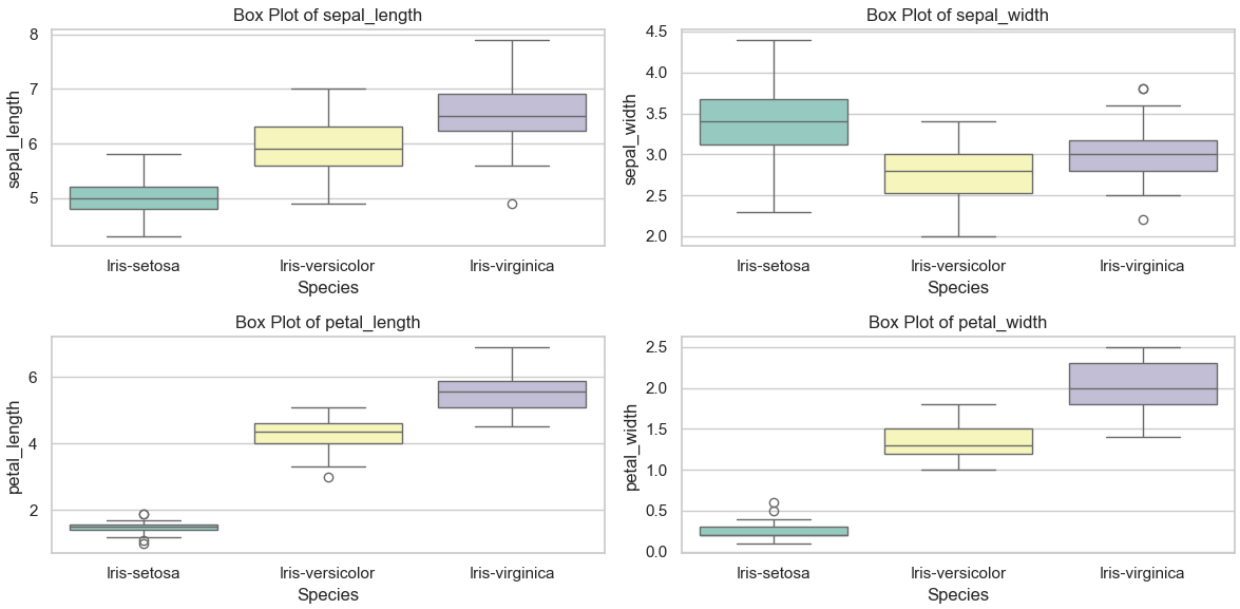
There are no missing values.

Exploratory data analysis

Check for outliers

```
# Box plots
sns.set(style="whitegrid")
plt.figure(figsize=(12, 6))

attributes = ['sepal_length', 'sepal_width', 'petal_length',
'petal_width']

for i, column in enumerate(attributes):
    plt.subplot(2, 2, i + 1)
    sns.boxplot(x='species', y=column, data=df, palette='Set3')

    plt.title(f'Box Plot of {column}')
    plt.xlabel('Species')
    plt.ylabel(column)

plt.tight_layout()
plt.show()
```

```
C:\Users\yashk\AppData\Local\Temp\ipykernel_12592\104556284.py:9:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.boxplot(x='species', y=column, data=df, palette='Set3')
C:\Users\yashk\AppData\Local\Temp\ipykernel_12592\104556284.py:9:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.boxplot(x='species', y=column, data=df, palette='Set3')
C:\Users\yashk\AppData\Local\Temp\ipykernel_12592\104556284.py:9:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.boxplot(x='species', y=column, data=df, palette='Set3')
C:\Users\yashk\AppData\Local\Temp\ipykernel_12592\104556284.py:9:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.boxplot(x='species', y=column, data=df, palette='Set3')
```



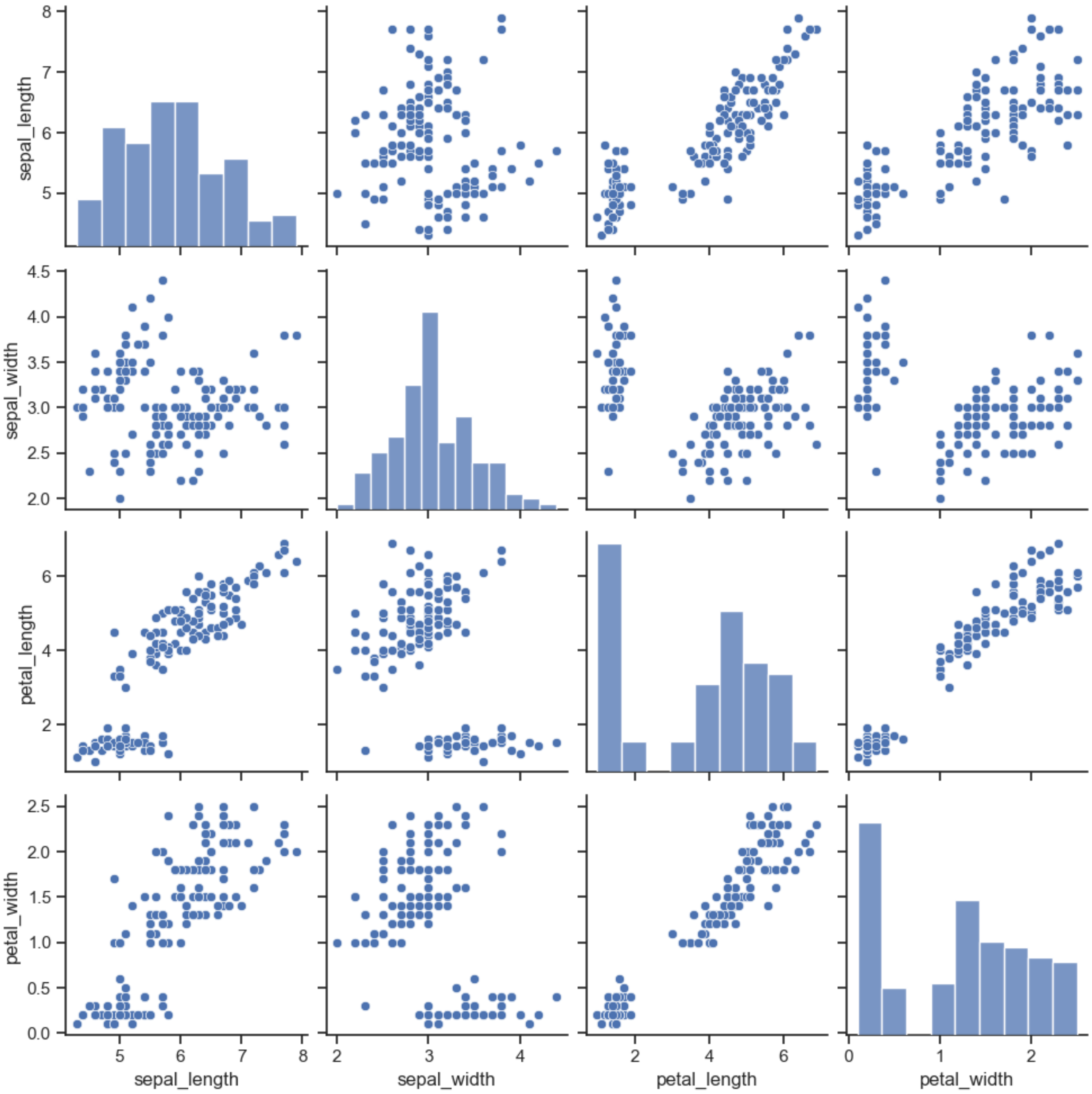
Outliers are detected in the sepal length and sepal width of Iris-virginica, as well as in the petal length of Iris-setosa and Iris-versicolor, and the petal width of Iris-setosa.

Please note that the box plot for Iris-setosa, particularly in the petal length attribute, exhibits congestion due to the presence of outliers. Before proceeding with the replacement of outliers with the median, let's examine the quantiles.

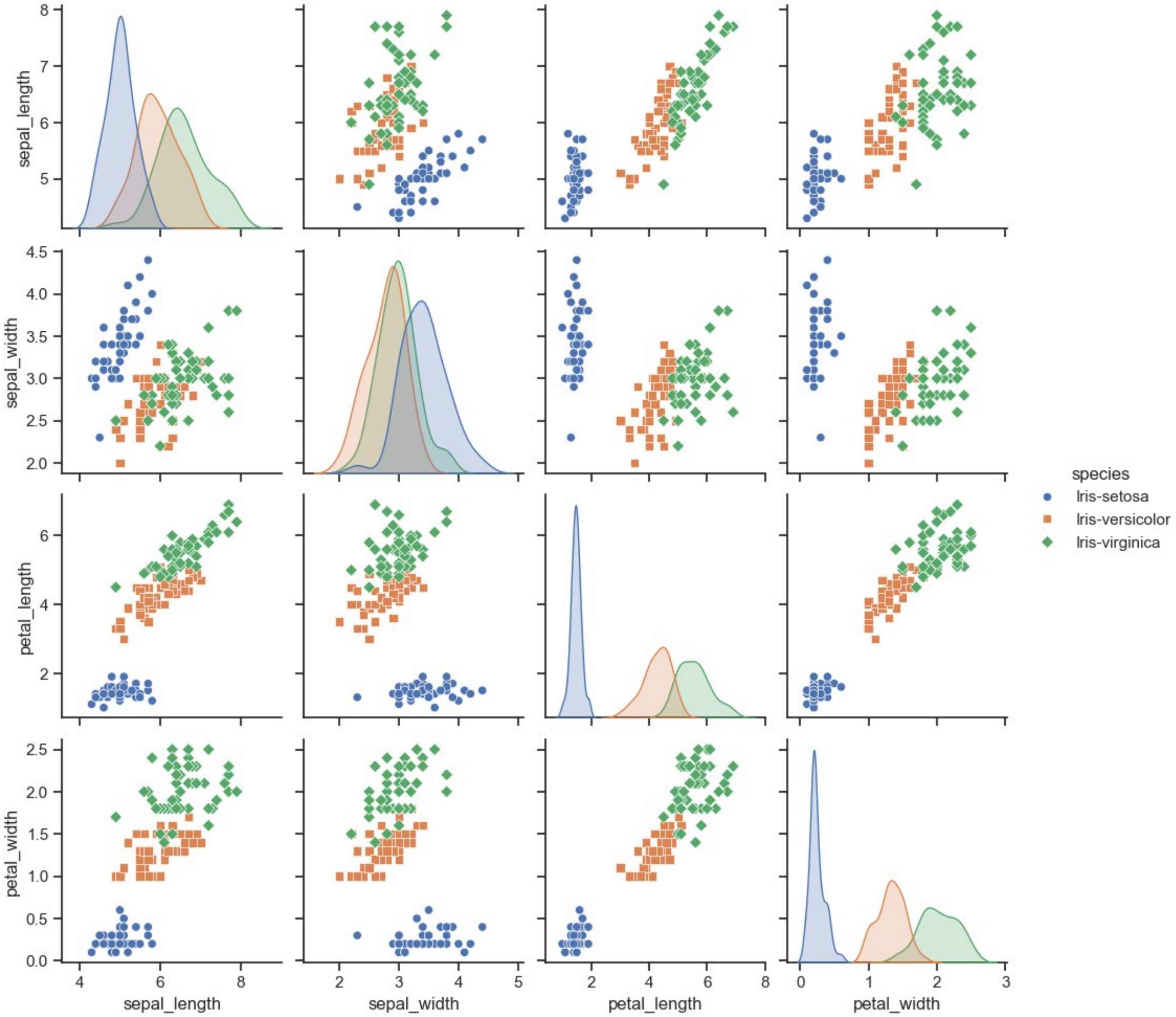
```
#Display the quantiles specifically for the 'Iris-setosa' species and
the 'petal_length' attribute.
setosa_quantiles = df[df['species'] == 'Iris-setosa']
['petal_length'].quantile([0.05, 0.95])
print(setosa_quantiles)
```

```
0.05    1.2
0.95    1.7
Name: petal_length, dtype: float64
```

```
sns.set(style="ticks")
sns.pairplot(df,markers=["o", "s", "D"])
plt.show()
```



```
sns.set(style="ticks")
sns.pairplot(df,hue="species", markers=["o", "s", "D"])
plt.show()
```



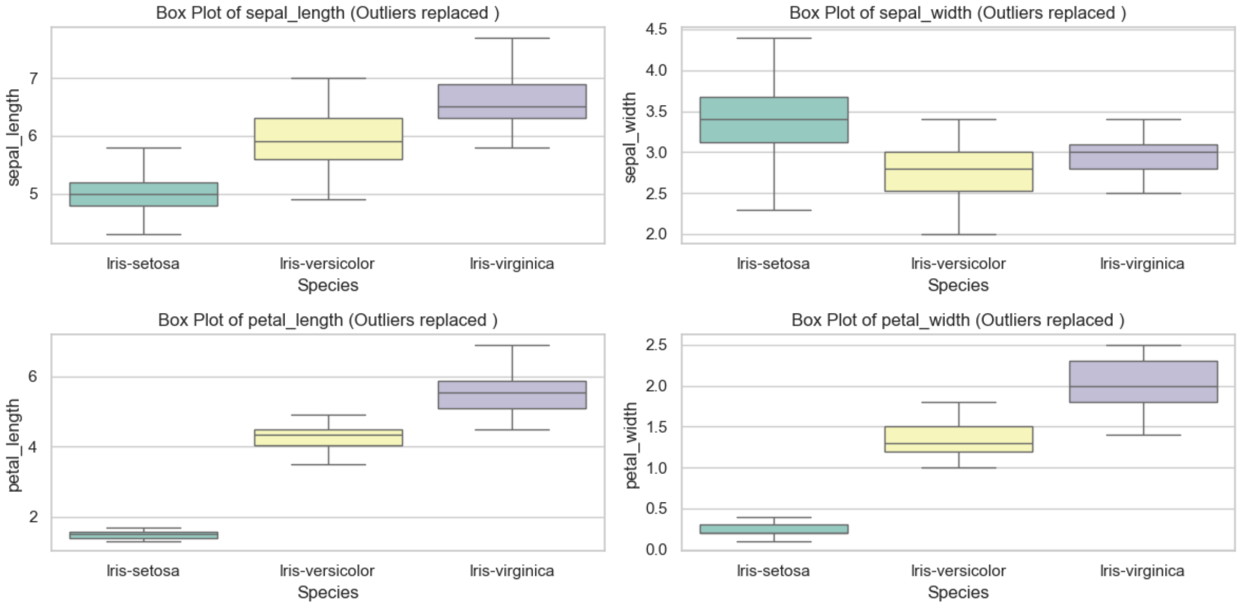
Use a combination of Winsorization and replacing outliers with median values.

```
attributes = ['sepal_length', 'sepal_width', 'petal_length',
'petal_width']
species_with_outliers = {'Iris-setosa': ['petal_length',
'petal_width'],
                         'Iris-versicolor': ['petal_length'],
                         'Iris-virginica': ['sepal_length',
'sepal_width']}

# Winsorize values for 'Iris-setosa' and 'petal_length'
df.loc[(df['species'] == 'Iris-setosa') & (df['petal_length'] < 1.3),
'petal_length'] = 1.3
df.loc[(df['species'] == 'Iris-setosa') & (df['petal_length'] > 1.7),
'petal_length'] = 1.7

# Find and replace outliers with median for specific species and
```

```
attributes
for species, outlier_columns in species_with_outliers.items():
    for column in outlier_columns:
        q_low = df[df['species'] == species][column].quantile(0.05)
        q_high = df[df['species'] == species][column].quantile(0.95)

        # Replace outliers with median
        median_value = df[df['species'] == species][column].median()
        df.loc[(df['species'] == species) & ((df[column] < q_low) |
(df[column] > q_high)), column] = median_value

# Box plots for each attribute with 'species' as hue after handling
outliers
sns.set(style="whitegrid")
plt.figure(figsize=(12, 6))

for i, column in enumerate(attributes):
    plt.subplot(2, 2, i + 1)
    sns.boxplot(x='species', y=column, data=df, palette='Set3')

    plt.title(f'Box Plot of {column} (Outliers replaced )')
    plt.xlabel('Species')
    plt.ylabel(column)

plt.tight_layout()
plt.show()
```

```
C:\Users\yashk\AppData\Local\Temp\ipykernel_12592\2944058266.py:26:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.boxplot(x='species', y=column, data=df, palette='Set3')
C:\Users\yashk\AppData\Local\Temp\ipykernel_12592\2944058266.py:26:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.boxplot(x='species', y=column, data=df, palette='Set3')
C:\Users\yashk\AppData\Local\Temp\ipykernel_12592\2944058266.py:26:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.boxplot(x='species', y=column, data=df, palette='Set3')
```

```
C:\Users\yashk\AppData\Local\Temp\ipykernel_12592\2944058266.py:26:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.boxplot(x='species', y=column, data=df, palette='Set3')
```



Outliers have been removed.

Define X (features)

```
x = df.drop(columns=df.columns[-1])
x.head(2)

   sepal_length  sepal_width  petal_length  petal_width
0           5.1          3.5           1.4          0.2
1           4.9          3.0           1.4          0.2
```

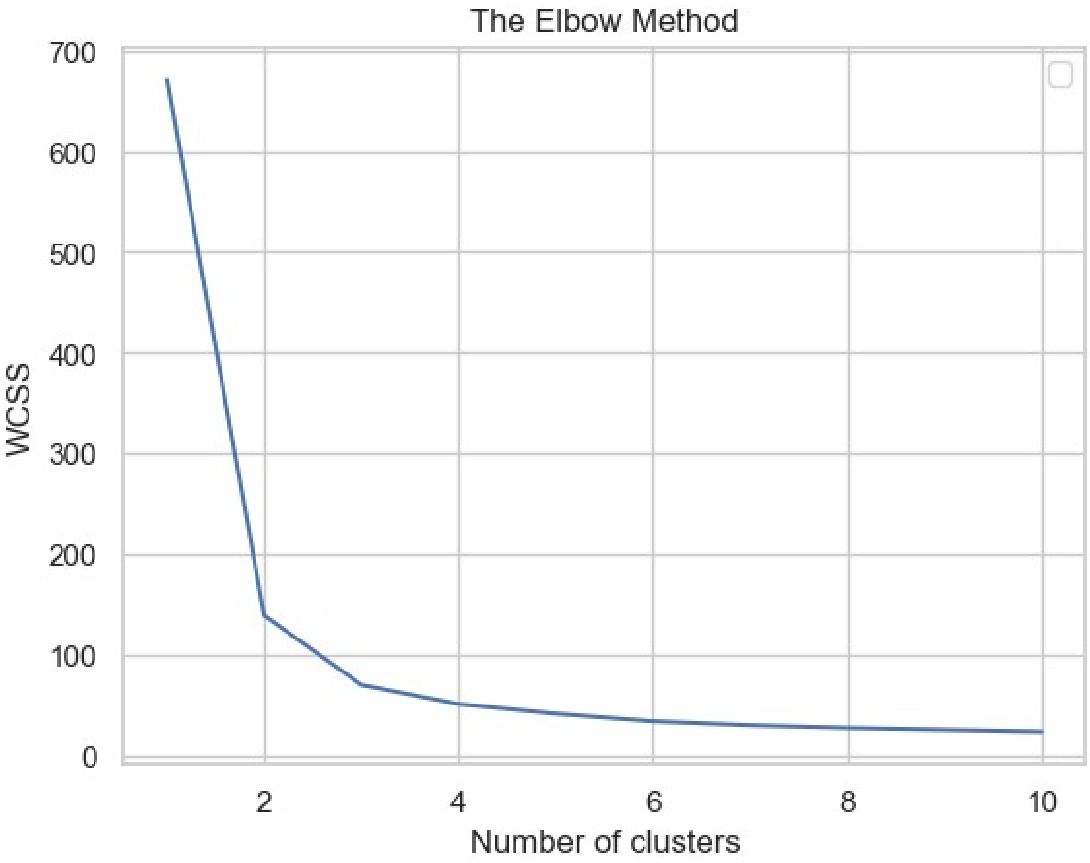
In this notebook, I'll use three distinct clustering methods: K-means, hierarchical, and Gaussian Mixture Models (GMM).

1. K-Means clustering

Define the number of clusters, K

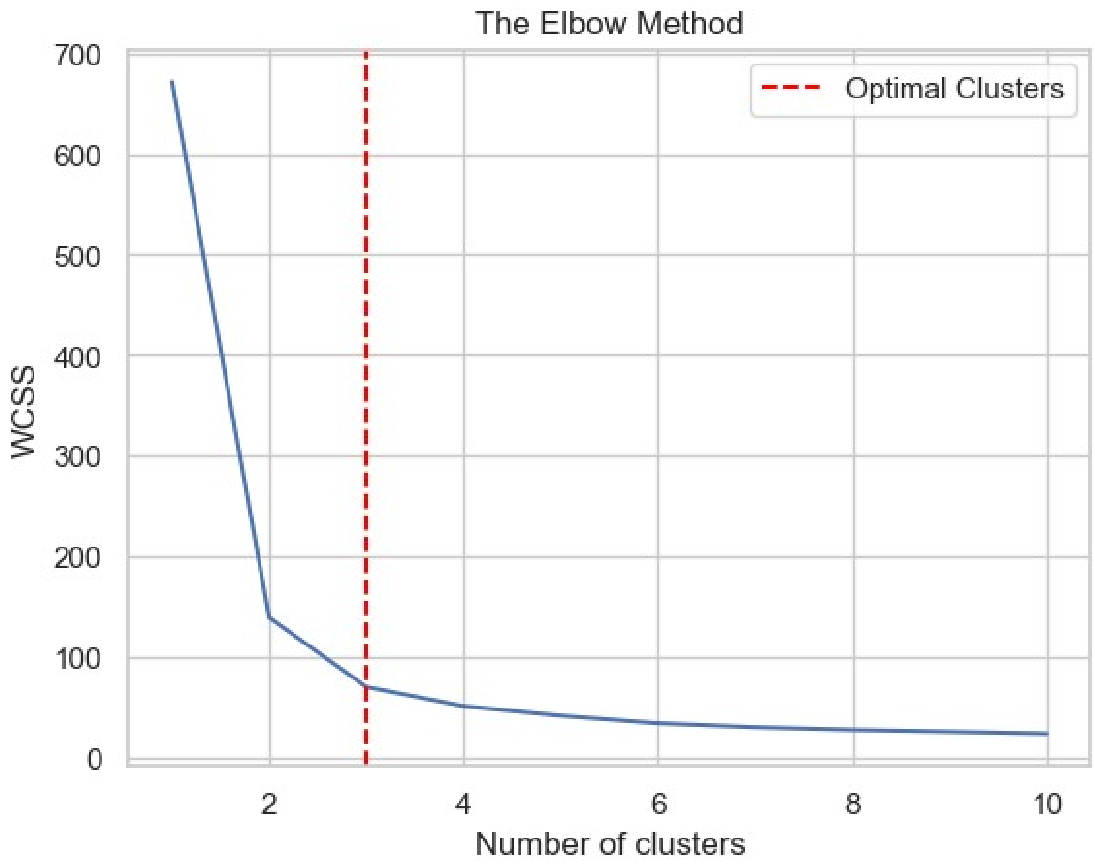
```python
# Elbow method
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42,
n_init=10)
    kmeans.fit(x)
    wcss.append(kmeans.inertia_)

plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.legend()
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists
whose label start with an underscore are ignored when legend() is
called with no argument.
```



The elbow bend is evident at the 3rd cluster (as shown below), indicating the presence of three clusters in the dataset.

```
# Elbow method
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42,
n_init=10)
    kmeans.fit(x)
    wcss.append(kmeans.inertia_)

plt.plot(range(1, 11), wcss)
plt.axvline(x=3, color='red', linestyle='--', label='Optimal
Clusters')
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.legend()
plt.show()
```

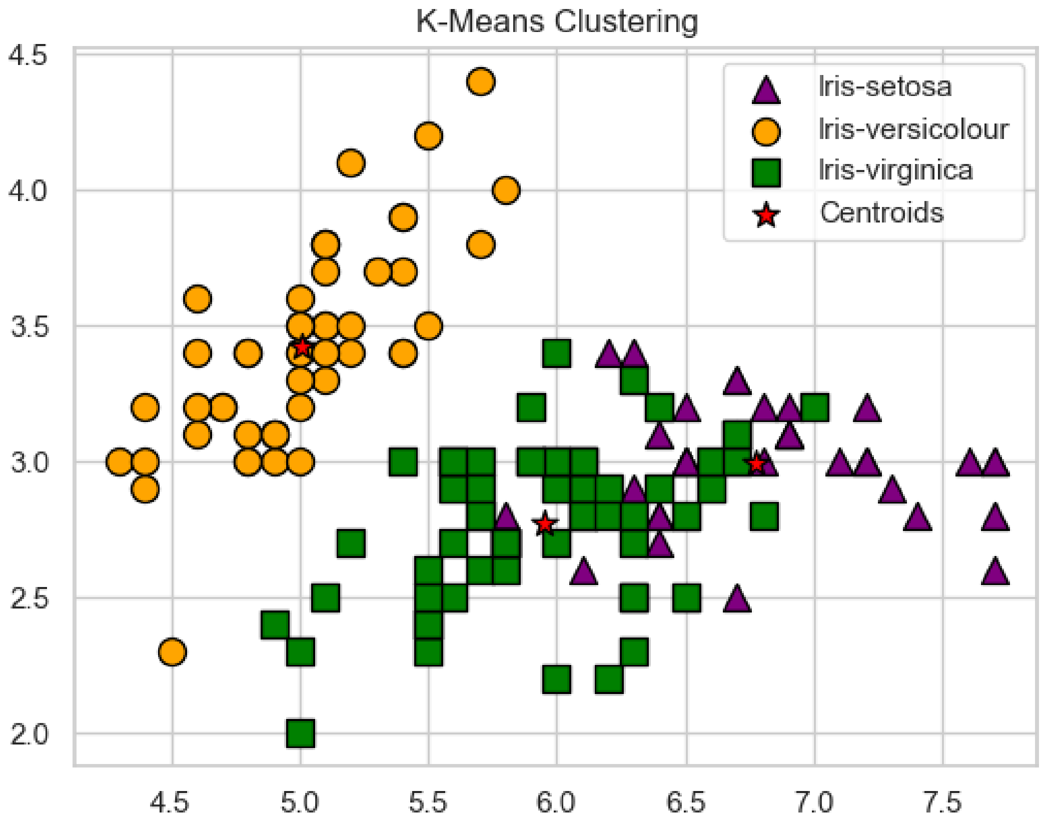


Training the K-Means model on the dataset

```
kmeans = KMeans(n_clusters = 3, init = 'k-means++', max_iter = 300,
n_init = 10, random_state = 42)
y_kmeans = kmeans.fit_predict(x)
```

Visualising the clusters

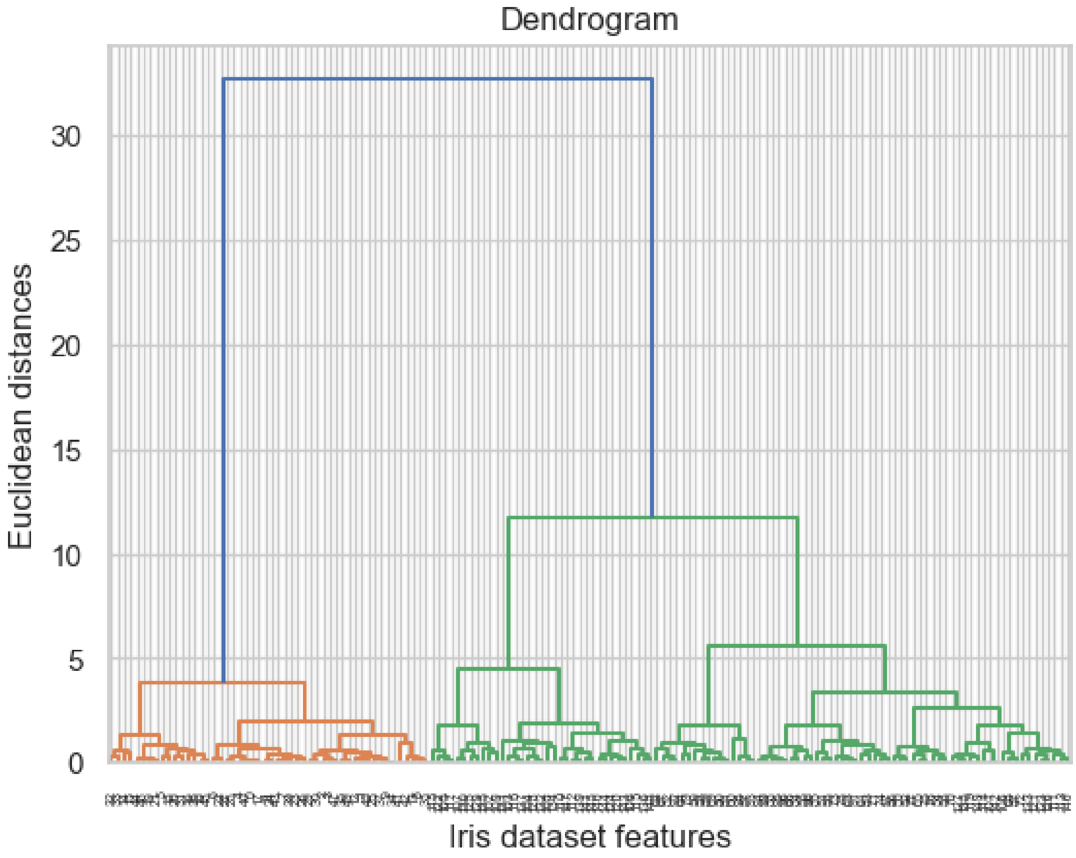
```
plt.scatter(x.iloc[y_kmeans == 0, 0], x.iloc[y_kmeans == 0, 1], s=100,
c='purple', marker='^', label='Iris-setosa', edgecolors='black')
plt.scatter(x.iloc[y_kmeans == 1, 0], x.iloc[y_kmeans == 1, 1], s=100,
c='orange', marker='o', label='Iris-versicolour', edgecolors='black')
plt.scatter(x.iloc[y_kmeans == 2, 0], x.iloc[y_kmeans == 2, 1], s=100,
c='green', marker='s', label='Iris-virginica', edgecolors='black')
# Plotting the centroids of the clusters
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,
1], s=100, c='red', marker='*', label='Centroids', edgecolors='black')
plt.title('K-Means Clustering')
plt.legend()
plt.show()
```



2. Hierarchical Clustering

Using the dendrogram to find the optimal number of clusters

```
dendrogram = sch.dendrogram(sch.linkage(x, method = 'ward'))
plt.title('Dendrogram')
plt.xlabel('Iris dataset features')
plt.ylabel('Euclidean distances')
plt.show()
```

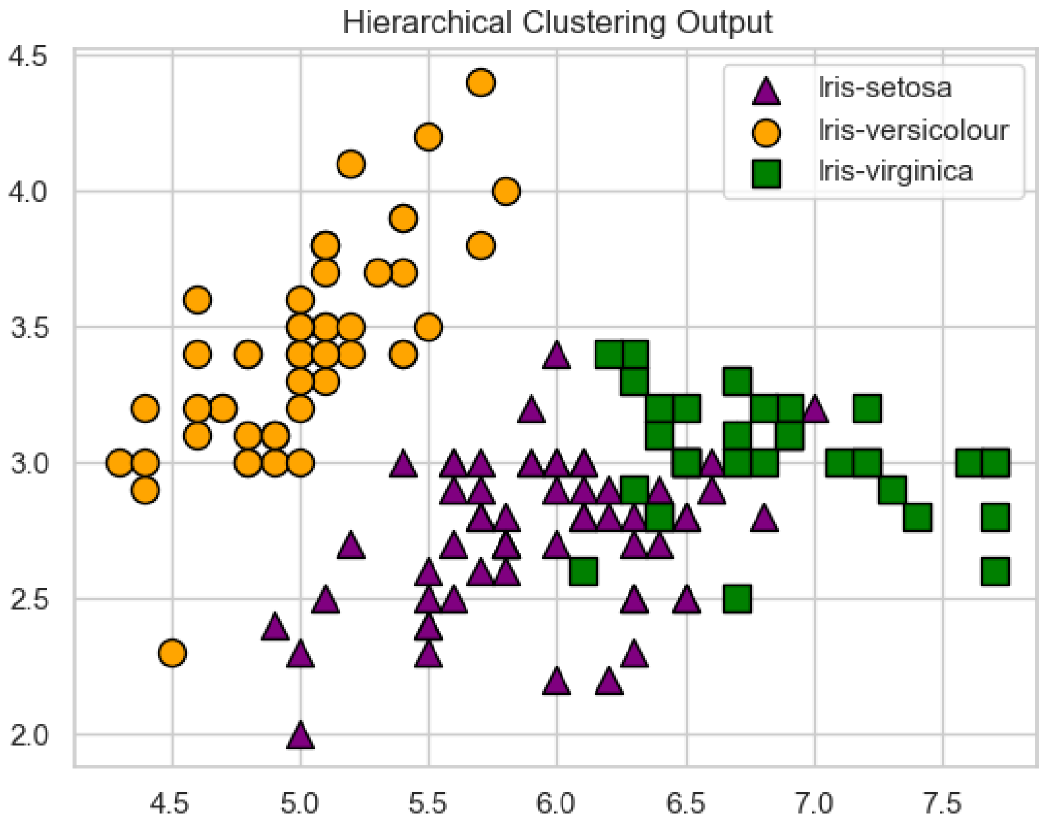


Training the Hierarchical Clustering model on the dataset

```
hc = AgglomerativeClustering(n_clusters=3, linkage='ward',
metric='euclidean')
y_hc = hc.fit_predict(x)
```

Visualising the clusters

```
plt.scatter(x.iloc[y_hc == 0, 0], x.iloc[y_hc == 0, 1], s=100,
c='purple', marker='^', label='Iris-setosa', edgecolors='black')
plt.scatter(x.iloc[y_hc == 1, 0], x.iloc[y_hc == 1, 1], s=100,
c='orange', marker='o', label='Iris-versicolour', edgecolors='black')
plt.scatter(x.iloc[y_hc == 2, 0], x.iloc[y_hc == 2, 1], s=100,
c='green', marker='s', label='Iris-virginica', edgecolors='black')
plt.title('Hierarchical Clustering Output')
plt.legend()
plt.show()
```

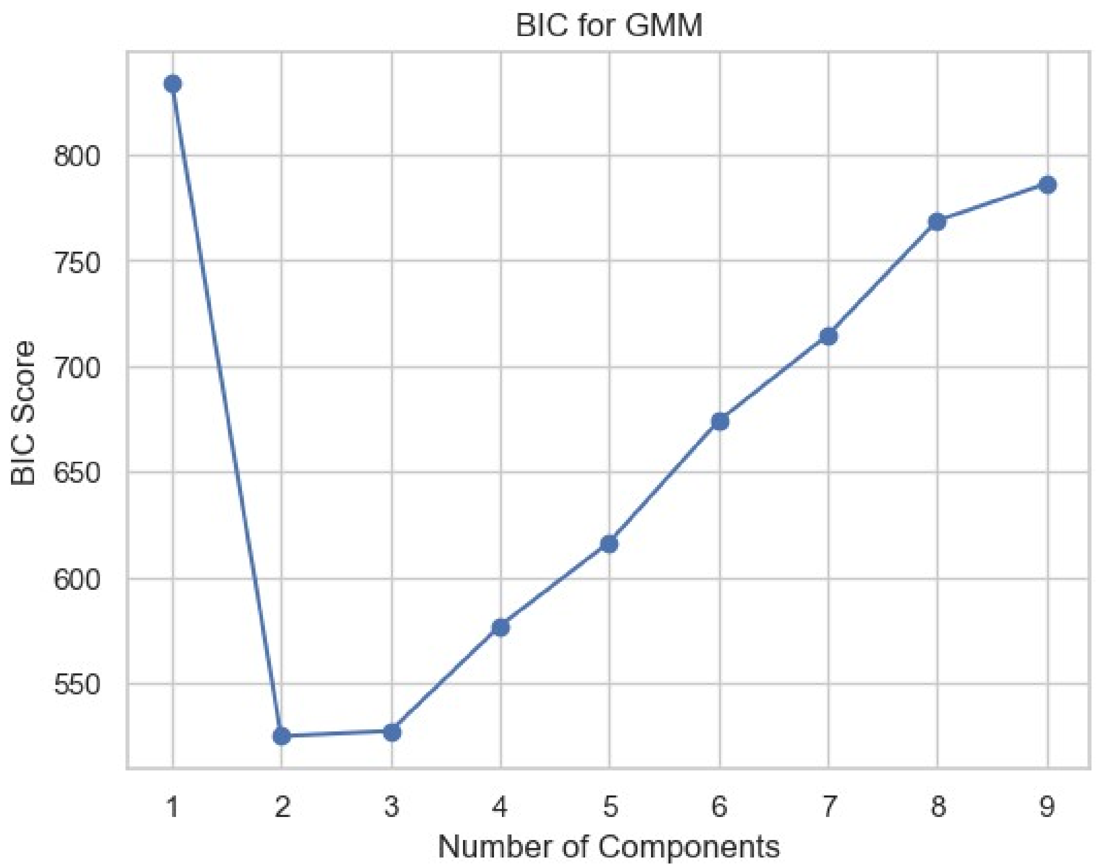


3. Gaussian Mixture Model (GMM)

In the context of GMMs, Bayesian Information Criterion (BIC) is often employed to choose the optimal number of components (clusters) for the GMM.

Fit GMM models for different numbers of components and plot BIC

```
n_components = range(1, 10)

bic = []
for n in n_components:
    gmm = GaussianMixture(n_components=n, random_state=42)
    gmm.fit(x)
    bic.append(gmm.bic(x))

# Plot the BIC values
plt.plot(n_components, bic, marker='o')
plt.xlabel('Number of Components')
plt.ylabel('BIC Score')
plt.title('BIC for GMM')
plt.show()
```



Choose the number of clusters based on the lowest BIC value. While in this example it suggests 2 clusters, remember our dataset has three clusters, so I'm selecting 3.

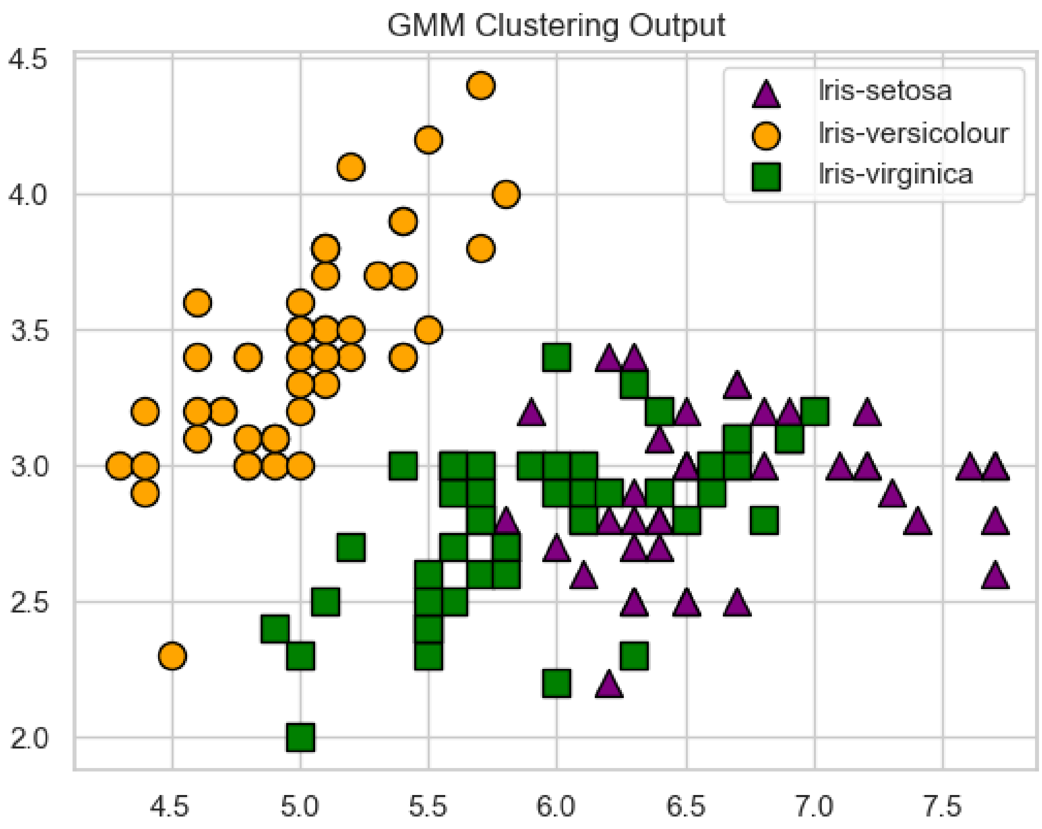
As a side note, for other datasets, follow the same logic and go with the lowest BIC for optimal clustering.

Training the GMM model on the dataset

```
gmm = GaussianMixture(n_components=3, random_state=42)
y_gmm = gmm.fit_predict(x)
```

Visualising the clusters

```
plt.scatter(x.iloc[y_gmm == 0, 0], x.iloc[y_gmm == 0, 1], s=100,
c='purple', marker='^', label='Iris-setosa', edgecolors='black')
plt.scatter(x.iloc[y_gmm == 1, 0], x.iloc[y_gmm == 1, 1], s=100,
c='orange', marker='o', label='Iris-versicolour', edgecolors='black')
plt.scatter(x.iloc[y_gmm == 2, 0], x.iloc[y_gmm == 2, 1], s=100,
c='green', marker='s', label='Iris-virginica', edgecolors='black')
plt.title('GMM Clustering Output')
plt.legend()
plt.show()
```



Summary

In the Iris dataset with features like Sepal length, Sepal width, Petal length, and Petal width, we used three clustering algorithms: K-Means, Hierarchical Clustering, and GMM, to reveal distinct clusters.

```python
labels = list(df["species"].unique())
labels
```

```
['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
```

```python
X = df.drop("species", axis=1)
y = df["species"]
```

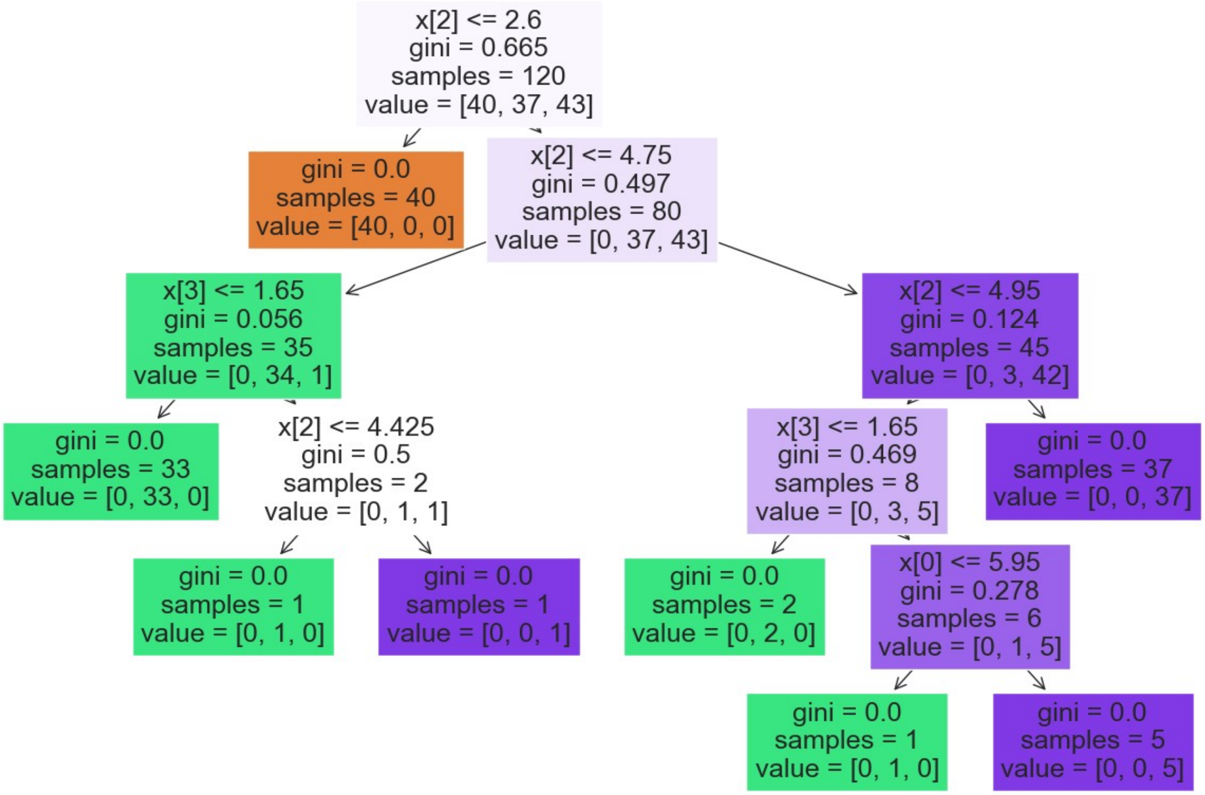
```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)
```

Post Pruning Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)
```

```
DecisionTreeClassifier()
```

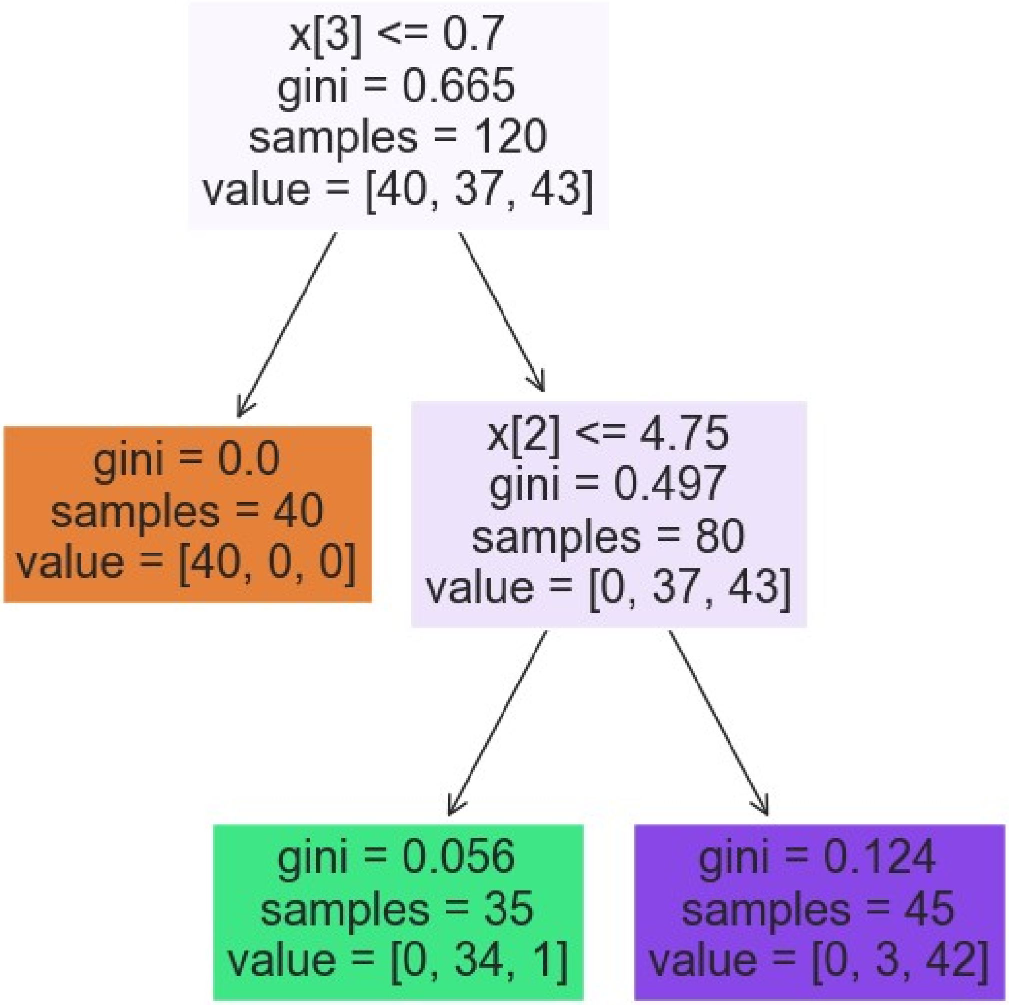
```
from sklearn import tree
plt.figure(figsize=(15,10))
tree.plot_tree(dt, filled=True)
plt.show()
```



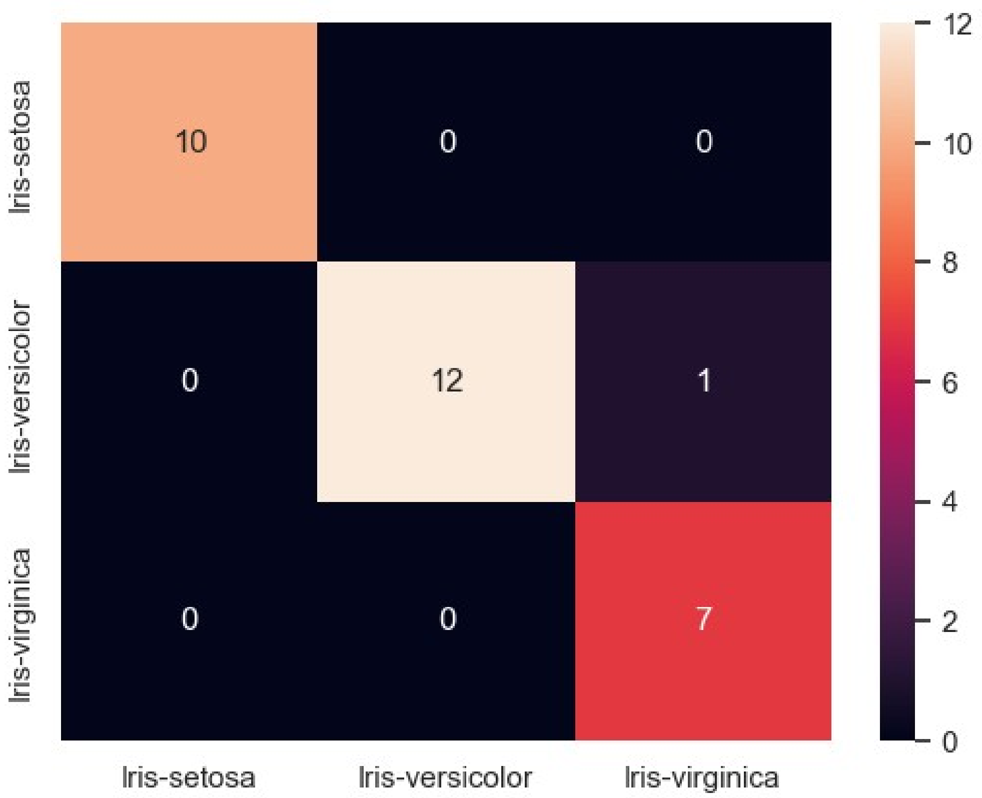
```
dt = DecisionTreeClassifier(max_depth=2)
dt.fit(X_train, y_train)
```

```
DecisionTreeClassifier(max_depth=2)
```

```
plt.figure(figsize=(7, 8))
tree.plot_tree(dt, filled=True)
plt.show()
```



```
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
y_pred = dt.predict(X_test)
score = accuracy_score(y_test, y_pred)
print("Post Pruning Score: ", score)
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt="d", xticklabels=labels,
yticklabels=labels)
plt.show()

Post Pruning Score:  0.9666666666666667
```



Pre Pruning

```
from sklearn.model_selection import GridSearchCV

dt = DecisionTreeClassifier()
param_grid = {
    "criterion": ["gini", "entropy", "log_loss"],
    "splitter": ["best", "random"],
    "max_depth": [2,3,4,5,6,7,8,9,10],
    "min_samples_split": [2,3,4,5,6,7,7,8,9,10],
    "max_features": ["auto", "sqrt", "log2"],
}

gscv = GridSearchCV(dt, param_grid, cv=5, scoring="accuracy")
gscv.fit(X_train, y_train)
```

```
c:\Users\yashk\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\model_selection\_validation.py:425: FitFailedWarning:

2700 fits failed out of a total of 8100.
The score on these train-test partitions for these parameters will be
set to nan.
If these failures are not expected, you can try to debug them by
setting error_score='raise'.
```

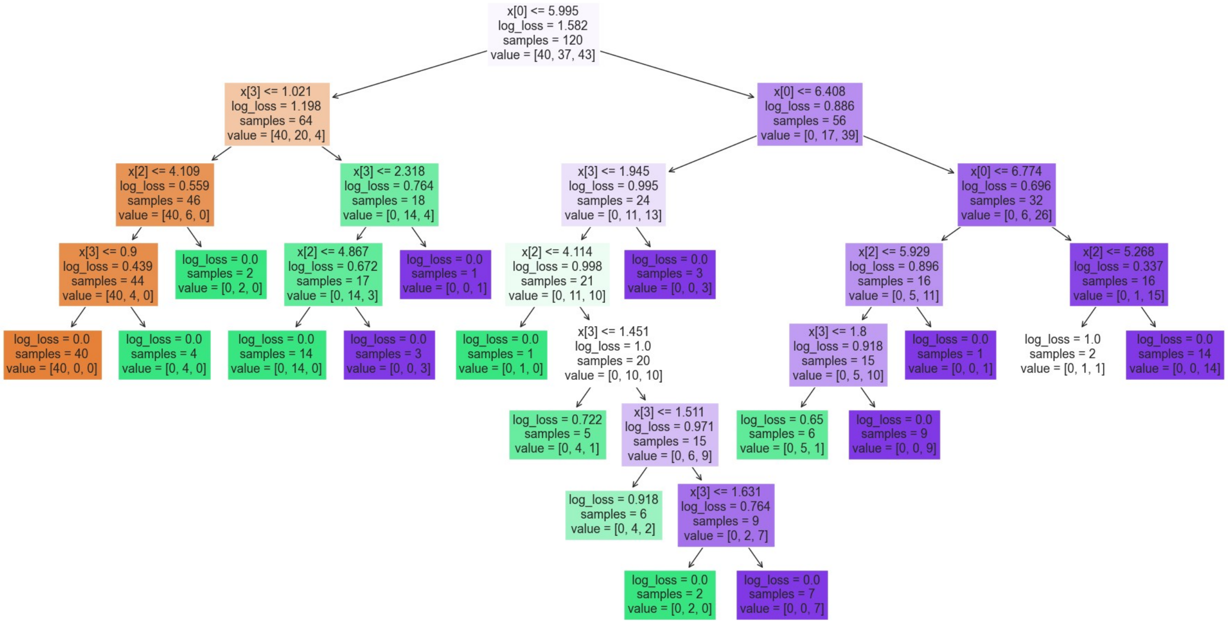
```
Below are more details about the failures:
----------------------------------------------------------------------
----------
2700 fits failed with the following error:
Traceback (most recent call last):
  File "c:\Users\yashk\AppData\Local\Programs\Python\Python311\Lib\
site-packages\sklearn\model_selection\_validation.py", line 729, in
_fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "c:\Users\yashk\AppData\Local\Programs\Python\Python311\Lib\
site-packages\sklearn\base.py", line 1145, in wrapper
    estimator._validate_params()
  File "c:\Users\yashk\AppData\Local\Programs\Python\Python311\Lib\
site-packages\sklearn\base.py", line 638, in _validate_params
    validate_parameter_constraints(
  File "c:\Users\yashk\AppData\Local\Programs\Python\Python311\Lib\
site-packages\sklearn\utils\_param_validation.py", line 96, in
validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The
'max_features' parameter of DecisionTreeClassifier must be an int in
the range [1, inf), a float in the range (0.0, 1.0], a str among
{'sqrt', 'log2'} or None. Got 'auto' instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
c:\Users\yashk\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\model_selection\_search.py:979: UserWarning: One or
more of the test scores are non-finite: [       nan        nan
nan ... 0.94166667 0.94166667 0.95      ]
  warnings.warn(

GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['gini', 'entropy', 'log_loss'],
                         'max_depth': [2, 3, 4, 5, 6, 7, 8, 9, 10],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'min_samples_split': [2, 3, 4, 5, 6, 7, 7, 8,
9, 10],
                         'splitter': ['best', 'random']},
             scoring='accuracy')
```

```python
print("Best params: ", gscv.best_params_)
print("Best score: ", gscv.best_score_)
```

```
Best params:  {'criterion': 'log_loss', 'max_depth': 8,
'max_features': 'log2', 'min_samples_split': 9, 'splitter': 'random'}
Best score:  0.9833333333333334
```

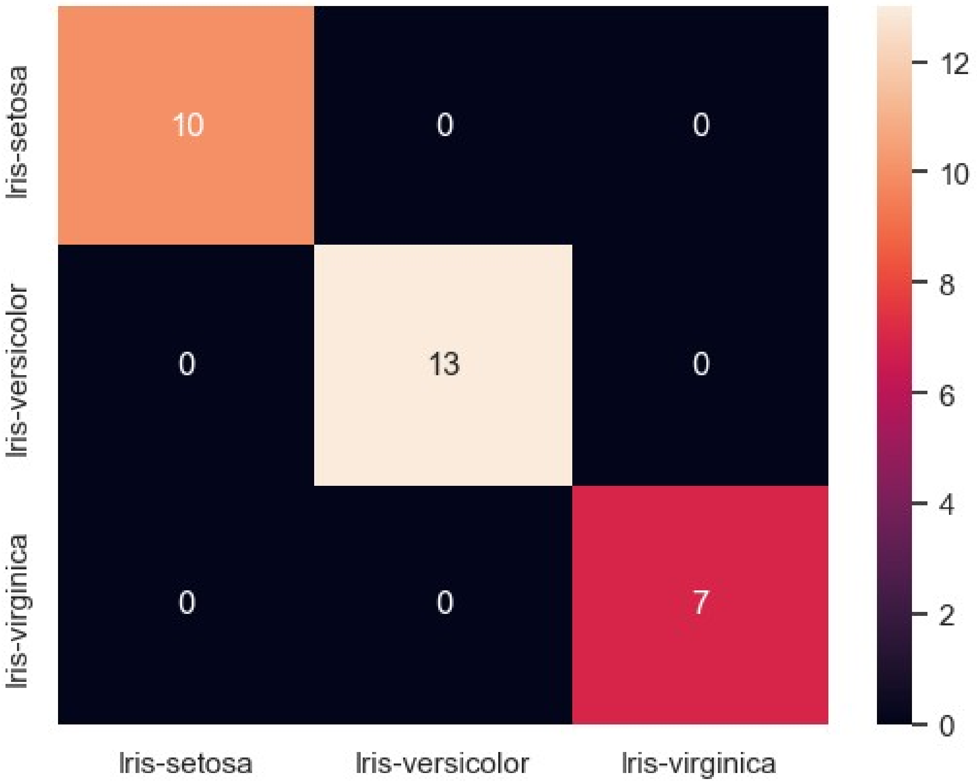
```python
from sklearn import tree
dt = gscv.best_estimator_
```

```
plt.figure(figsize=(23,12))
tree.plot_tree(dt, filled=True)
plt.show()
```


x[0] <= 5.995 log_loss = 1.582 samples = 120 value = [40, 37, 43]
x[3] <= 1.021 log_loss = 1.198 samples = 64 value = [40, 20, 4]
x[0] <= 6.408 log_loss = 0.886 samples = 56 value = [0, 17, 39]
x[2] <= 4.109 log_loss = 0.559 samples = 46 value = [40, 6, 0]
x[3] <= 2.318 log_loss = 0.764 samples = 18 value = [0, 14, 4]
x[3] <= 1.945 log_loss = 0.995 samples = 24 value = [0, 11, 13]
x[0] <= 6.774 log_loss = 0.696 samples = 32 value = [0, 6, 26]
x[3] <= 0.9 log_loss = 0.439 samples = 44 value = [40, 4, 0]
log_loss = 0.0 samples = 2 value = [0, 2, 0]
x[2] <= 4.867 log_loss = 0.672 samples = 17 value = [0, 14, 3]
log_loss = 0.0 samples = 1 value = [0, 0, 1]
x[2] <= 4.114 log_loss = 0.998 samples = 21 value = [0, 11, 10]
log_loss = 0.0 samples = 3 value = [0, 0, 3]
x[2] <= 5.929 log_loss = 0.896 samples = 16 value = [0, 5, 11]
x[2] <= 5.268 log_loss = 0.337 samples = 16 value = [0, 1, 15]
log_loss = 0.0 samples = 40 value = [40, 0, 0]
log_loss = 0.0 samples = 4 value = [0, 4, 0]
log_loss = 0.0 samples = 14 value = [0, 14, 0]
log_loss = 0.0 samples = 3 value = [0, 0, 3]
log_loss = 0.0 samples = 1 value = [0, 1, 0]
x[3] <= 1.451 log_loss = 1.0 samples = 20 value = [0, 10, 10]
x[3] <= 1.8 log_loss = 0.918 samples = 15 value = [0, 5, 10]
log_loss = 0.0 samples = 1 value = [0, 0, 1]
log_loss = 1.0 samples = 2 value = [0, 1, 1]
log_loss = 0.0 samples = 14 value = [0, 0, 14]
log_loss = 0.722 samples = 5 value = [0, 4, 1]
x[3] <= 1.511 log_loss = 0.971 samples = 15 value = [0, 6, 9]
log_loss = 0.65 samples = 6 value = [0, 5, 1]
log_loss = 0.0 samples = 9 value = [0, 0, 9]
log_loss = 0.918 samples = 6 value = [0, 4, 2]
x[3] <= 1.631 log_loss = 0.764 samples = 9 value = [0, 2, 7]
log_loss = 0.0 samples = 2 value = [0, 2, 0]
log_loss = 0.0 samples = 7 value = [0, 0, 7]

```
y_pred = dt.predict(X_test)
score = accuracy_score(y_test, y_pred)
print("Pre-Pruning Score: ", score)
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt="d", xticklabels=labels,
yticklabels=labels)
plt.show()
```

```
Pre-Pruning Score:  1.0
```


Iris-setosa
Iris-versicolor
Iris-virginica
10
0
0
0
13
0
0
0
7
Iris-setosa
Iris-versicolor
Iris-virginica
12
10
8
6
4
2
0