**Abstract:**

This project focuses on the analysis and clustering of the **Iris dataset** using machine learning techniques.

The Iris flower data set is a specific set of information compiled by **Ronald Fisher**, a biologist, in the 1930s. It describes particular biological characteristics of various types of Iris flowers, specifically, contains measurements of length & width of Sepal & Petal for three species of iris flowers.

The analysis begins with **Exploratory data analysis (EDA)**, where we visualize the distribution of features, examine pairwise relationships, and investigate potential correlations.

Following EDA, we apply unsupervised learning algorithms, specifically **K-means clustering and hierarchical clustering**, to partition the dataset into distinct clusters based on feature similarity.

For K-means clustering, we employ the elbow method to determine the optimal number of clusters and visualize the resulting clusters using scatter plots and 3D scatter plots. Additionally, hierarchical clustering is utilized to construct a dendrogram, revealing the hierarchical structure of the data and allowing for an intuitive understanding of cluster relationships.

Through this analysis, we aim to gain insights into the natural groupings present in the Iris dataset and provide a foundational understanding of clustering techniques in machine learning.

The results of this project can inform future studies in clustering analysis and serve as a basis for further exploration of similar datasets.

**Aim:**

The aim of this project is to explore the inherent structure within the Iris dataset using unsupervised learning techniques and to identify natural groupings or clusters of data points based on their feature similarities.

Additionally, we seek to gain insights into the characteristics of each cluster and understand how they correspond to the different species of iris flowers present in the dataset. Through this analysis, we aim to provide a comprehensive understanding of clustering techniques in machine learning and their applicability to real-world datasets.

**Dataset Info:**

The Iris flower data set or Fisher's Iris data set is a multivariate data set introduced by the British statistician, eugenicist, and biologist Ronald Fisher in his 1936 paper. The use of multiple measurements in taxonomic problems as an example of linear discriminant analysis. It is sometimes called Anderson's Iris data set because Edgar Anderson collected the data to quantify the morphologic variation of Iris flowers of three related species. Two of the three species were collected in the Gaspé Peninsula "all from the same pasture, and picked on the same day and measured at the same time by the same person with the same apparatus".

The data set consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). Four features were measured from each sample: the length and the width of the sepals and petals, in centimeters.

1. sepal length in cm

2. sepal width in cm

3. petal length in cm

4. petal width in cm

5. class: -- Iris Setosa -- Iris Versicolour -- Iris Virginica

**Tools Used:**

Python: The primary programming language used for data analysis, machine learning, and visualization tasks.

Jupyter Notebook: An interactive development environment used for running Python code in a notebook format, facilitating code execution, visualization, and documentation.

**Libraries Used:**

Pandas: A powerful data analysis library used for data manipulation, exploration, and transformation.

NumPy: A fundamental package for scientific computing in Python, used for numerical operations and array manipulation.

Matplotlib: A versatile plotting library for creating static, interactive, and publication-quality visualizations.

Seaborn: A statistical data visualization library based on Matplotlib, used for creating attractive and informative statistical graphics.

Scikit-learn: A machine learning library in Python, providing simple and efficient tools for data mining and data analysis tasks, including clustering algorithms.

Plotly: An interactive visualization library used for creating interactive, web-based plots and dashboards.

SciPy: A library used for scientific and technical computing, including modules for hierarchical clustering and dendrogram visualization.

**Flowchart:**

Start

|

|--> 1. Load Data (Iris Dataset)

| --> Check data integrity and format

| --> Preprocess data if necessary (e.g., handling missing values, scaling)

|

|--> 2. Exploratory Data Analysis (EDA)

| --> Visualize data distributions (histograms, boxplots)

| --> Explore pairwise relationships (scatter plots, pairplots)

| --> Investigate correlations between features (heatmaps)

|

|--> 3. Clustering Analysis

| --> 3.1. K-means Clustering

| | --> Determine optimal number of clusters (elbow method)

| | --> Perform K-means clustering

| | --> Visualize clusters (scatter plots, 3D scatter plots)

| |

| --> 3.2. Hierarchical Clustering

| --> Generate dendrogram

| --> Interpret cluster hierarchy

|

|--> 4. Interpretation and Insights

| --> Analyze cluster characteristics and differences

| --> Relate clusters to species of iris flowers

|

End

**Code:**

#import libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette\_score

from sklearn.preprocessing import MinMaxScaler .

from sklearn.preprocessing import StandardScaler

import plotly.graph\_objects as go

import plotly.express as px

from scipy.cluster.hierarchy import linkage, dendrogram

import warnings # To ignore warnings.

warnings.simplefilter(action='ignore')

#To import the Iris dataset:

iris = pd.read\_csv("D:\\IRIS dataset.csv") # the iris dataset is now a Pandas DataFrame

# Extracting features from the dataset.

x = iris.iloc[:, [0, 1, 2, 3]].values # Selecting columns containing feature values.

iris.info() #To view Iris data below

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 150 entries, 0 to 149

Data columns (total 5 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 sepal\_length 150 non-null float64

1 sepal\_width 150 non-null float64

2 petal\_length 150 non-null float64

3 petal\_width 150 non-null float64

4 species 150 non-null object

dtypes: float64(4), object(1)

memory usage: 6.0+ KB

# dimension

print("the dimension:", iris.shape) # number of rows and columns.

the dimension: (150, 5)

iris.nunique() #To check number of uniqe elements in each columns.

sepal\_length 35

sepal\_width 23

petal\_length 43

petal\_width 22

species 3

dtype: int64

iris.describe(include='all') # checking the simple statistical parameters

| **sepal\_length** | **sepal\_width** | **petal\_length** | **petal\_width** | **species** |
| --- | --- | --- | --- | --- |
| **scount** | 150.000000 | 150.000000 | 150.000000 | 150.000000 | 150 |
| **unique** | NaN | NaN | NaN | NaN | 3 |
| **top** | NaN | NaN | NaN | NaN | Iris-setosa |
| **freq** | NaN | NaN | NaN | NaN | 50 |
| **mean** | 5.843333 | 3.054000 | 3.758667 | 1.198667 | NaN |
| **std** | 0.828066 | 0.433594 | 1.764420 | 0.763161 | NaN |
| **min** | 4.300000 | 2.000000 | 1.000000 | 0.100000 | NaN |
| **25%** | 5.100000 | 2.800000 | 1.600000 | 0.300000 | NaN |
| **50%** | 5.800000 | 3.000000 | 4.350000 | 1.300000 | NaN |
| **75%** | 6.400000 | 3.300000 | 5.100000 | 1.800000 | NaN |
| **max** | 7.900000 | 4.400000 | 6.900000 | 2.500000 | NaN |

# Samples for each species by frequency distribution

iris.groupby('species').size()

species

Iris-setosa 50

Iris-versicolor 50

Iris-virginica 50

dtype: int64

# Calculating the mean for each feature by species

iris.groupby('species').mean()

species

Iris-setosa 5.006 3.418 1.464 0.244

Iris-versicolor 5.936 2.770 4.260 1.326

Iris-virginica 6.588 2.974 5.552 2.026

# Setting up training data

X = iris.iloc[:, :-1].values # Extract features (all columns except the last one) and store them in X.

y = iris.iloc[:, -1].values # Extract target variable (the last column) and store it in y.

# Creating separate datasets for each species

iris\_setosa=iris.loc[iris["species"]=="Iris-setosa"]

iris\_virginica=iris.loc[iris["species"]=="Iris-virginica"]

iris\_versicolor=iris.loc[iris["species"]=="Iris-versicolor"]

VISUALIZATION

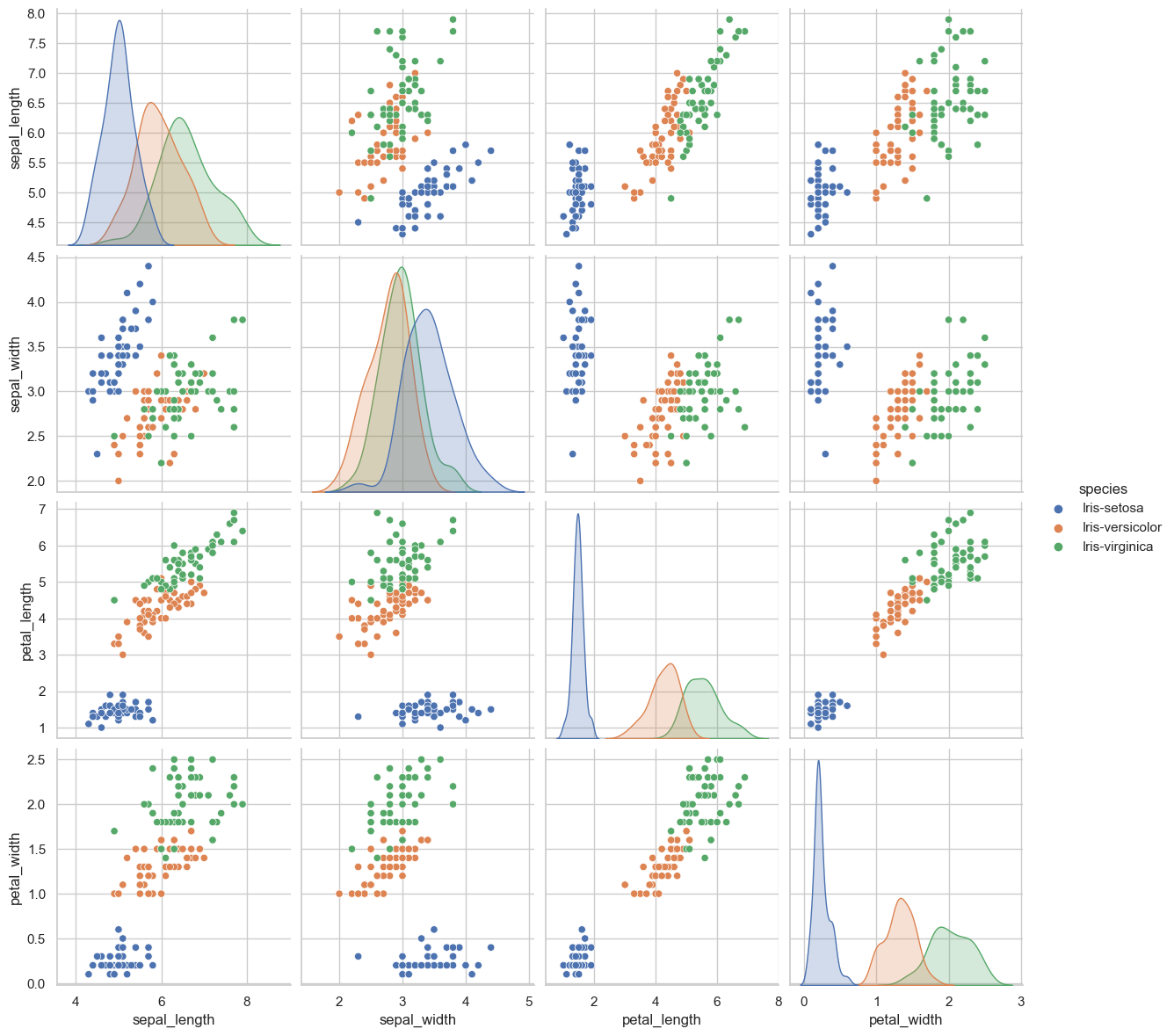
#Use pairplot to analyze the relationship between species for all characteristic combinations.

sns.set(style="whitegrid")

sns.pairplot(iris,hue="species", size=3)

plt.show()

# An observable trend shows a close relationship between two of the species



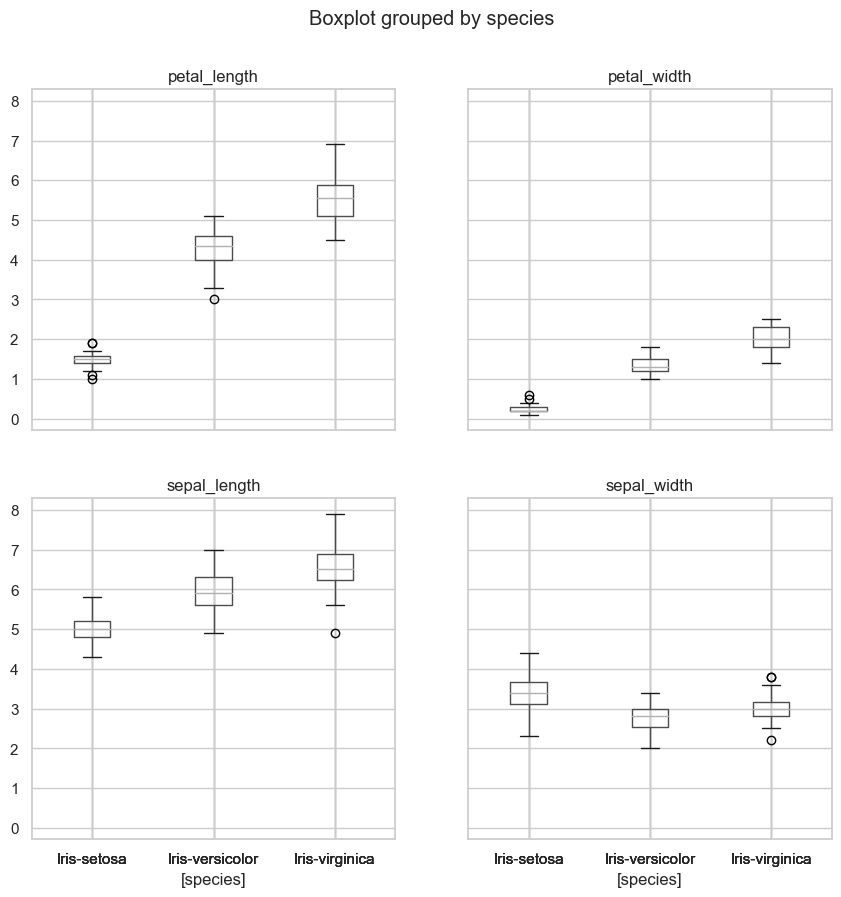
# In the above scatter plot, we can see PetalWidthCm and PetalLengthCm has the strongest pari-wise relationship for classification.

# To make a Pandas boxplot grouped by species, use .boxplot

# Modify the figsize, by placing a value in the X and Y cordinates

iris.boxplot(by="species", figsize=(10, 10))

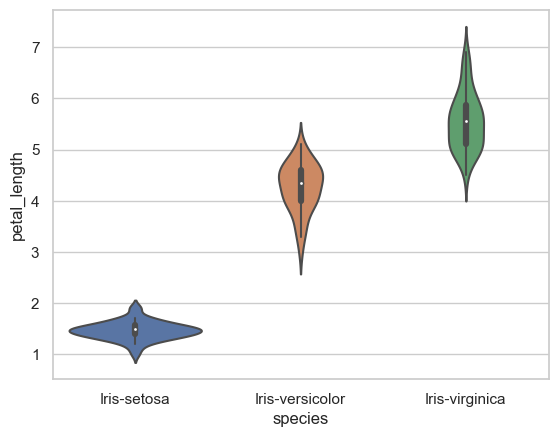
plt.show()



# A violin plot shows the density of the data, simularly to a scatter plot, and presents catagorical data like a box plot. Denser regions of the data are fatter.

sns.violinplot(x="species", y="petal\_length", data=iris, size=6)

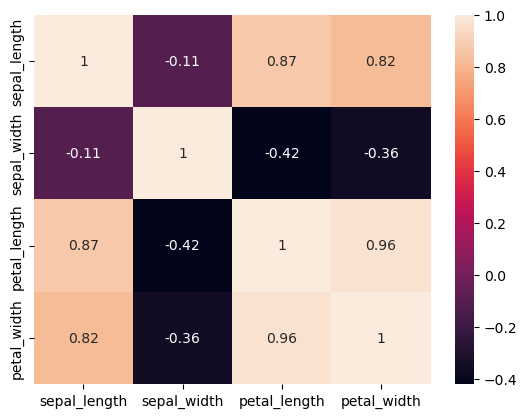
plt.show()



# Creating a heatmap to visualize the correlation matrix

sns.heatmap(iris.corr(), annot=True) # annot = annotations

plt.show()



# Creating FacetGrids to plot distributions of features by species

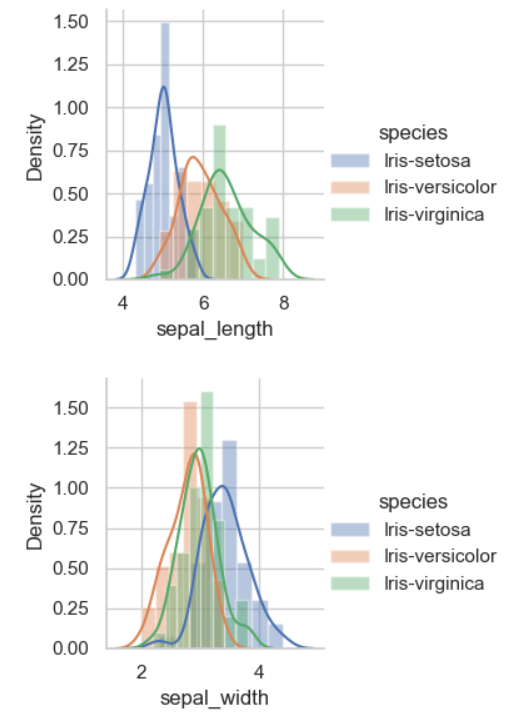
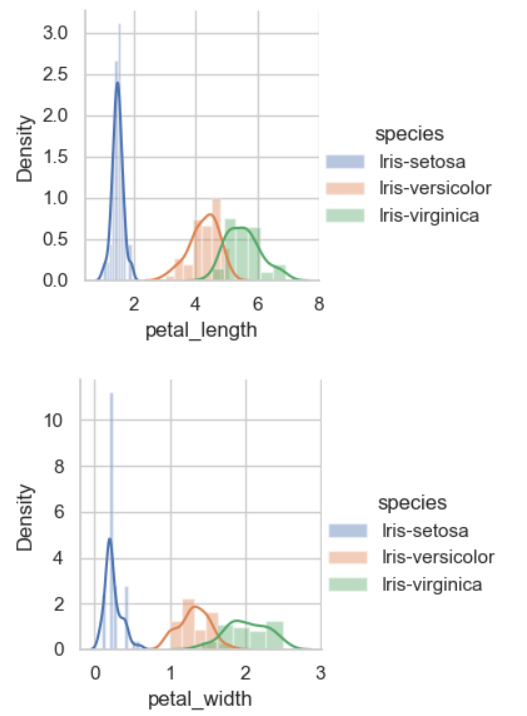
sns.FacetGrid(iris,hue="species",height=3).map(sns.distplot,"petal\_length").add\_legend()

sns.FacetGrid(iris,hue="species",height=3).map(sns.distplot,"petal\_width").add\_legend()

sns.FacetGrid(iris,hue="species",height=3).map(sns.distplot,"sepal\_length").add\_legend()

sns.FacetGrid(iris,hue="species",height=3).map(sns.distplot,"sepal\_width").add\_legend()

plt.show()



K-MEANS  is a centroid-based algorithm, or a distance-based algorithm, where we calculate the distances to assign a point to a cluster. In K-Means, each cluster is associated with a centroid.

**STEPS TO FOLLOW**

* Choose the number of clusters k
* Select k random points from the data as centroids
* Assign all the points to the closest cluster centroid
* Recompute the centroids of newly formed clusters
* Repeat steps 3 and 4

#Finding the optimum number of clusters for k-means classification

from sklearn.cluster import KMeans

wcss = [] # Initializing a list to store within-cluster sum of squares (WCSS) values

for i in range(1, 11):

# Create KMeans model for each number of clusters from 1 to 10

kmeans =KMeans(n\_clusters=i, init = 'k-means++', max\_iter = 300, n\_init = 10, random\_state = 0)

kmeans.fit(x) # Fit the model to the data

wcss.append(kmeans.inertia\_) # Append the WCSS value to the list

**Using the elbow method to determine the optimal number of clusters for k-means clustering**

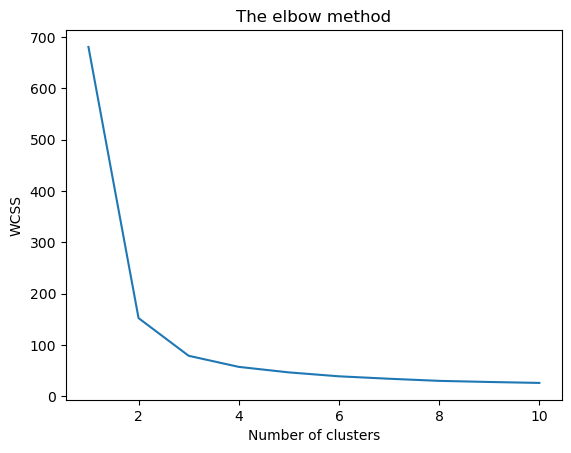
plt.plot(range(1, 11), wcss) # Plotting the number of clusters against the WCSS values

plt.title('The elbow method')

plt.xlabel('Number of clusters')

plt.ylabel('WCSS') #within cluster sum of squares

plt.show()

****

# Creating KMeans model with 3 clusters and fitting it to the data

kmeans = KMeans(n\_clusters = 3, init = 'k-means++', max\_iter = 300, n\_init = 10, random\_state = 0)

y\_kmeans = kmeans.fit\_predict(x) # Predicting the cluster labels for the data

# Retrieving the centroids of the clusters

centroids= kmeans.cluster\_centers\_ # Get the coordinates of the centroids for each cluster

print(centroids)

[[5.9016129 2.7483871 4.39354839 1.43387097]

[5.006 3.418 1.464 0.244 ]

[6.85 3.07368421 5.74210526 2.07105263]]

#Visualising the clusters

plt.scatter(x[y\_kmeans == 0, 0], x[y\_kmeans == 0, 1], s = 100, c = 'purple', label = 'Iris-setosa')

plt.scatter(x[y\_kmeans == 1, 0], x[y\_kmeans == 1, 1], s = 100, c = 'orange', label = 'Iris-versicolour')

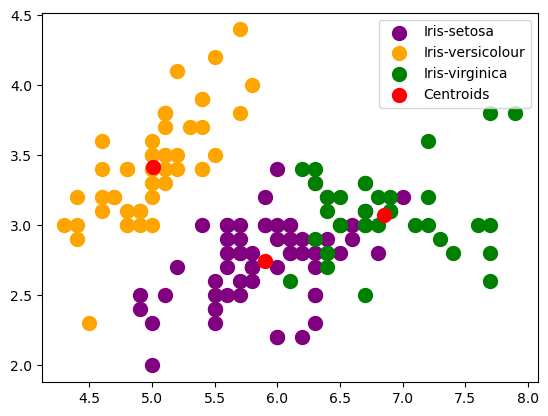
plt.scatter(x[y\_kmeans == 2, 0], x[y\_kmeans == 2, 1], s = 100, c = 'green', label = 'Iris-virginica')

#Plotting the centroids of the clusters

plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:,1], s = 100, c = 'red', label = 'Centroids')

plt.legend()

plt.show()



# Creating a 3D scatter plot

fig = plt.figure(figsize = (15,15))

ax = fig.add\_subplot(111, projection='3d')

# Scatter plot for each cluster

plt.scatter(x[y\_kmeans == 0, 0], x[y\_kmeans == 0, 1], s = 100, c = 'purple', label = 'Iris-setosa')

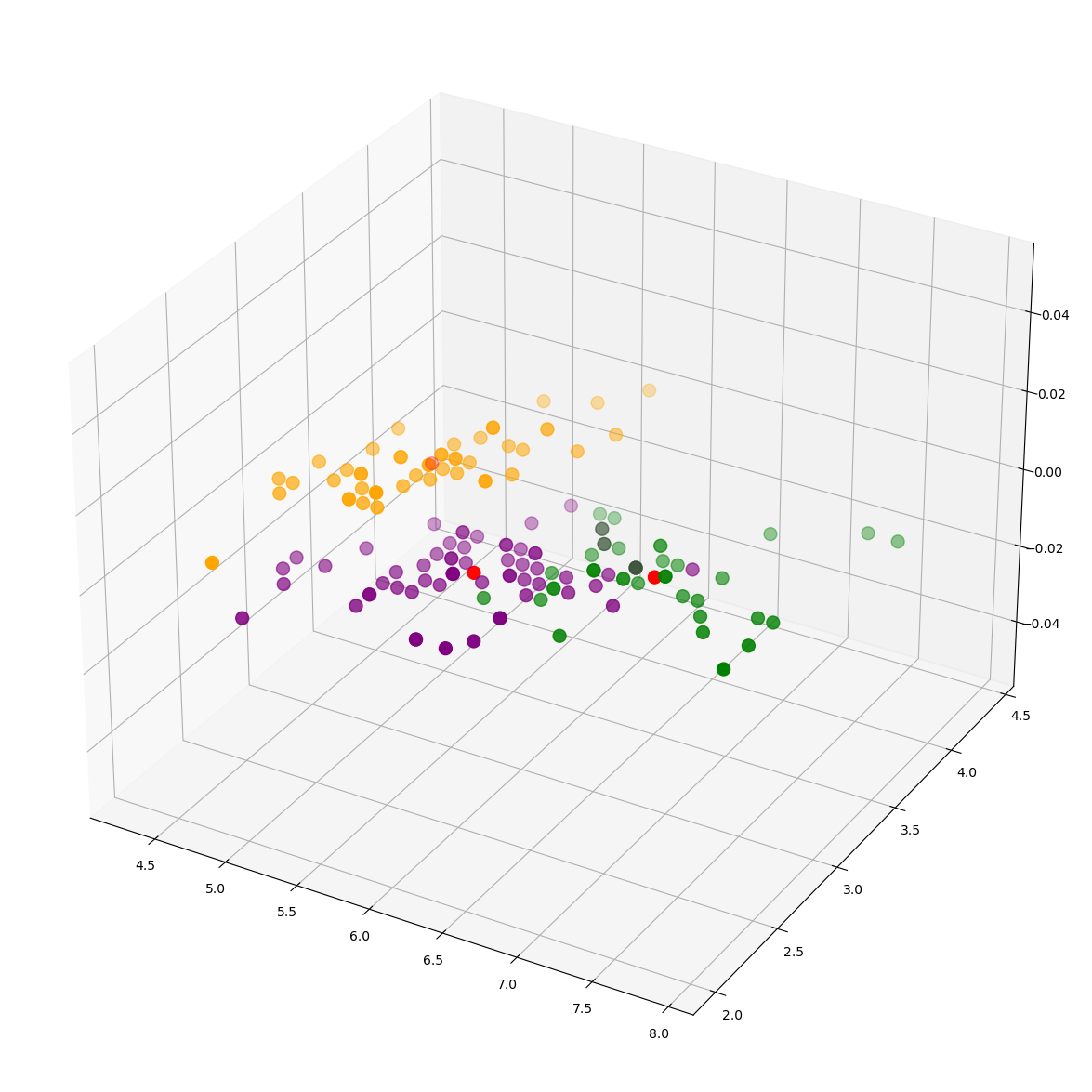
plt.scatter(x[y\_kmeans == 1, 0], x[y\_kmeans == 1, 1], s = 100, c = 'orange', label = 'Iris-versicolour')

plt.scatter(x[y\_kmeans == 2, 0], x[y\_kmeans == 2, 1], s = 100, c = 'green', label = 'Iris-virginica')

#Plotting the centroids of the clusters

plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:,1], s = 100, c = 'red', label = 'Centroids')

plt.show()



cluster=kmeans.labels\_ # Assigning cluster labels to a variable

cluster #our tag values are between 0-2

array([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, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 2, 2, 2, 2, 0, 2, 2, 2,

2, 2, 2, 0, 0, 2, 2, 2, 2, 0, 2, 0, 2, 0, 2, 2, 0, 0, 2, 2, 2, 2,

2, 0, 2, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 0])

# Adding cluster labels to the Iris dataset

iris["cluster\_no"] = cluster

# Creating a Plotly figure

fig = go.Figure()

# Adding scatter plot traces for each cluster

fig.add\_trace(go.Scatter(

x=X[y\_kmeans == 0, 0], y=X[y\_kmeans == 0, 1],

mode='markers',marker\_color='#DB4CB2',name='Iris-setosa'

))

fig.add\_trace(go.Scatter(

x=X[y\_kmeans == 1, 0], y=X[y\_kmeans == 1, 1],

mode='markers',marker\_color='#c9e9f6',name='Iris-versicolour'

))

fig.add\_trace(go.Scatter(

x=X[y\_kmeans == 2, 0], y=X[y\_kmeans == 2, 1],

mode='markers',marker\_color='#7D3AC1',name='Iris-virginica'

))

fig.add\_trace(go.Scatter(

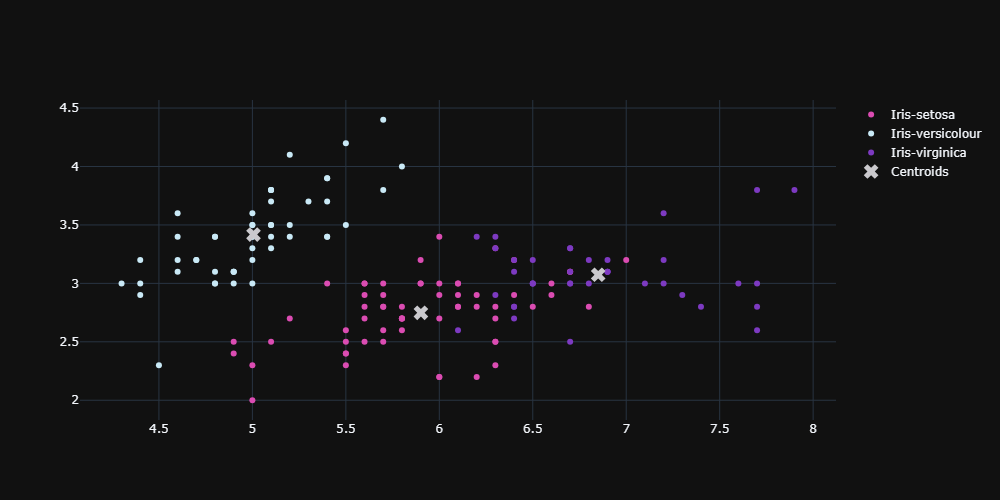
x=centroids[:, 0], y=centroids[:,1],

mode='markers',marker\_color='#CAC9CD',marker\_symbol=4,marker\_size=13,name='Centroids'

))

# Updating layout

fig.update\_layout(template='plotly\_dark',width=1000, height=500,)

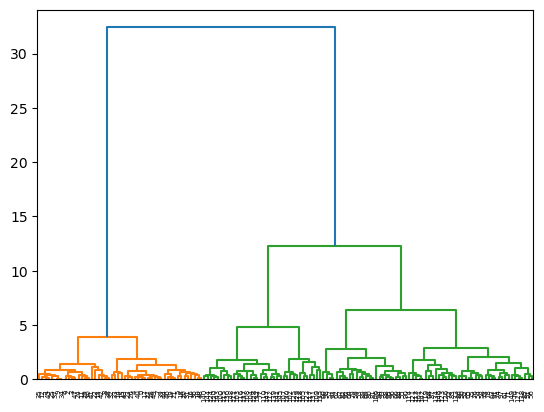


**HIERARCHICAL CLUSTER**

import scipy.cluster.hierarchy as sch

dendogram=sch.dendrogram(sch.linkage(x,method="ward")) # Generating the dendrogram

plt.show()



# We looked at the hierarchy of clusters with the dendogram, we see that we can choose the optimum number of clusters ideally 4.

plt.figure(figsize=(10,10))

plt.title("Hierarchical Clustering Dendogram")

plt.xlabel("Observation Units")

plt.ylabel("distances")

dendrogram(linkage(x, "complete"), # Generating the dendrogram with complete linkage method

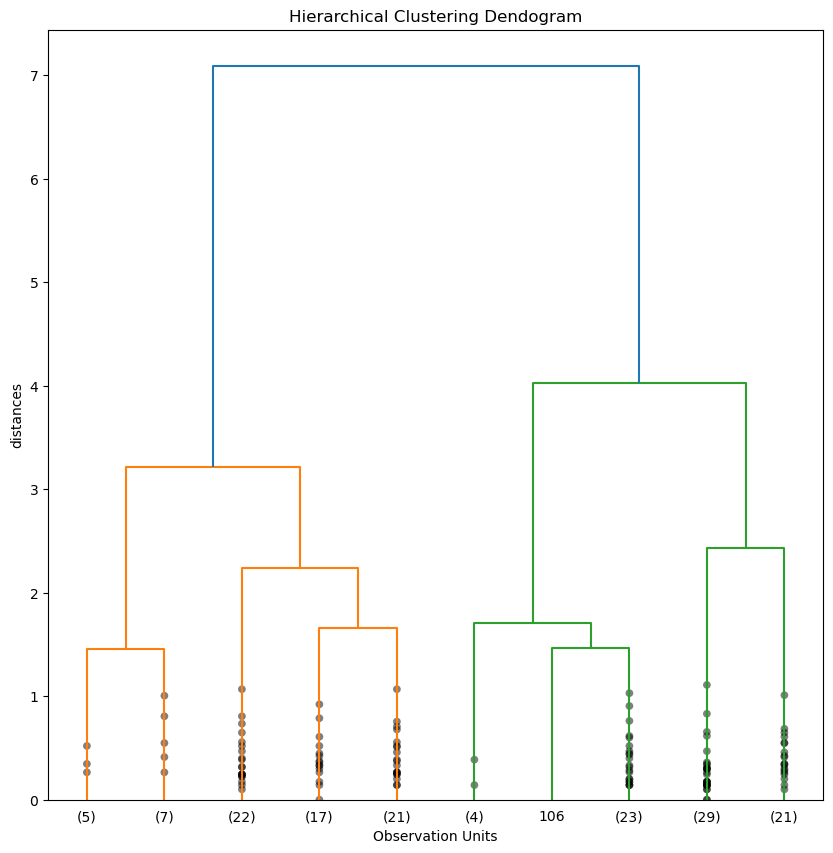
leaf\_font\_size=10, # Setting font size for leaf labels

truncate\_mode="lastp", # Truncating the dendrogram to show last p merged clusters

p=10, # Number of clusters to show when truncate\_mode=”lastp”

show\_contracted=True) # Showing contracted dendrogram branches

plt.show()



**INTERPRETATION AND INSIGHTS**

Analysis of Cluster Characteristics

# Compute mean feature values for each cluster

cluster\_means = iris.groupby('cluster\_no').mean()

# Print cluster centroids and mean feature values

print("Cluster Centroids:")

print(centroids)

print("\nMean Feature Values for Each Cluster:")

print(cluster\_means)

Cluster Centroids:

[[5.9016129 2.7483871 4.39354839 1.43387097]

[5.006 3.418 1.464 0.244 ]

[6.85 3.07368421 5.74210526 2.07105263]]

Mean Feature Values for Each Cluster:

sepal\_length sepal\_width petal\_length petal\_width

cluster\_no

0 5.901613 2.748387 4.393548 1.433871

1 5.006000 3.418000 1.464000 0.244000

2 6.850000 3.073684 5.742105 2.071053

Relating Clusters to Species of Iris Flowers

# Compare cluster labels to species labels

cluster\_species\_comparison = pd.crosstab(iris['species'], iris['cluster\_no'])

# Print the comparison table

print("Cluster vs. Species Comparison:")

print(cluster\_species\_comparison)

Cluster vs. Species Comparison:

cluster\_no 0 1 2

species

Iris-setosa 0 50 0

Iris-versicolor 48 0 2

Iris-virginica 14 0 36

**Conclusion:**

In this project, we conducted an analysis of the Iris dataset, focusing on classification and clustering of iris flowers based on their features. Through exploratory data analysis (EDA), we gained insights into the distribution of features, explored pairwise relationships, and identified correlations between variables.

Using unsupervised learning algorithms such as K-means and hierarchical clustering, we identified natural groupings or clusters within the data. By determining the optimal number of clusters and interpreting the cluster characteristics, we gained insights into the inherent structure of the dataset. We related the clusters to the species of iris flowers and assessed the agreement between the clusters and the actual species labels.

Through our analysis, we observed that the clustering results aligned well with the known species categories, particularly for the Iris-setosa species. However, there were some overlaps and ambiguities observed, especially between the Iris-versicolor and Iris-virginica species.

In conclusion, by combining data analysis, machine learning algorithms, and biological insights, we gained a deeper understanding of the characteristics of iris flowers and their relationships based on their features.