**Iris flower Classification App**



KENYATTA UNIVERSITY

DEPARTMENT OF COMPUTING AND INFORMATION TECHNOLOGY

SCHOOL OF PURE AND APPLIED SCIENCE

SMA 390 SCIENTIFIC PROGRAMMING

CAT 2 ASSIGNMENT

|  |  |  |
| --- | --- | --- |
| S.NO | NAME | REG.NO |
| 1 | LEVY NATILI | I163/5855/2021 |
| 2 | SHELDON BICHANGA | I163/5858/2021 |
| 3 | COLLINS KIMUTAI | I163/6502/2021 |
| 4 | EMMANUEL KAMOING | I163/7337/2021 |
| 5 | SALOME NDERI | I163/5821/2021 |
| 6 | STANELY MUKUNYA | I163/5850/2021 |
| 7 | JAMES MULI | I163/4935/2021 |
| 8 | ROBINA MEBO | I163/6787/2021 |
| 9 | FAITH CHEBET | I163/1319/2021 |
| 10 | KARIMI MARTIN | I163/5791/2021 |
| 11 | GIDEON WARWIMBO | I163s/13229/2020 |

Contents

[Introduction 2](#_Toc161410017)

[1.Iris\_classification.ipynb 3](#_Toc161410018)

[Exploring the data. 4](#_Toc161410019)

[Modelling 5](#_Toc161410020)

[Hierarchical Clustering 7](#_Toc161410021)

[2.App.py 10](#_Toc161410022)

[Reference 13](#_Toc161410023)

# Introduction

We created a simple python app application which classifies the iris flower to its subsequent species groups (e.g., Setosa, Versicolor, or Virginica) based on their sepal length, sepal width, petal length and petal width. The app utilizes the model we train to predict the species of the iris flower.

To solve the problem, we created two files:

1. Iris\_classification.ipynb
2. App.py

We also use iris dataset to train the model for prediction.

# 1.Iris\_classification.ipynb

The objective of this .ipynb file is to create a optimum model to classify iris flower into its subsequent species group. It enabled us to clean, manipulate, visualize and model iris data. We first imported the necessary python libraries

# Data Manipulation and Visualisation libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

# Modeling libraries

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.datasets import load\_iris

from sklearn.cluster import KMeans

from sklearn.mixture import GaussianMixture

from scipy.cluster.hierarchy import linkage, dendrogram, fcluster

from sklearn import metrics

# Model for saving the model

import joblib

We then load the data.

# load data

iris\_data = load\_iris(as\_frame=True)['frame']

# Loads entire dataset including both the feature data and the target data.

iris\_unsupervised=load\_iris(as\_frame=True)['data']

# Loads only the featured data suitable for unsupervised learning.

#view the data

iris\_data.head()

|  | **sepal length (cm)** | **sepal width (cm)** | **petal length (cm)** | **petal width (cm)** | **target** |
| --- | --- | --- | --- | --- | --- |
| 0 | 5.1 | 3.5 | 1.4 | 0.2 | 0 |
| 1 | 4.9 | 3.0 | 1.4 | 0.2 | 0 |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 | 0 |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 | 0 |
| 4 | 5.0 | 3.6 | 1.4 | 0.2 | 0 |

## Exploring the data.

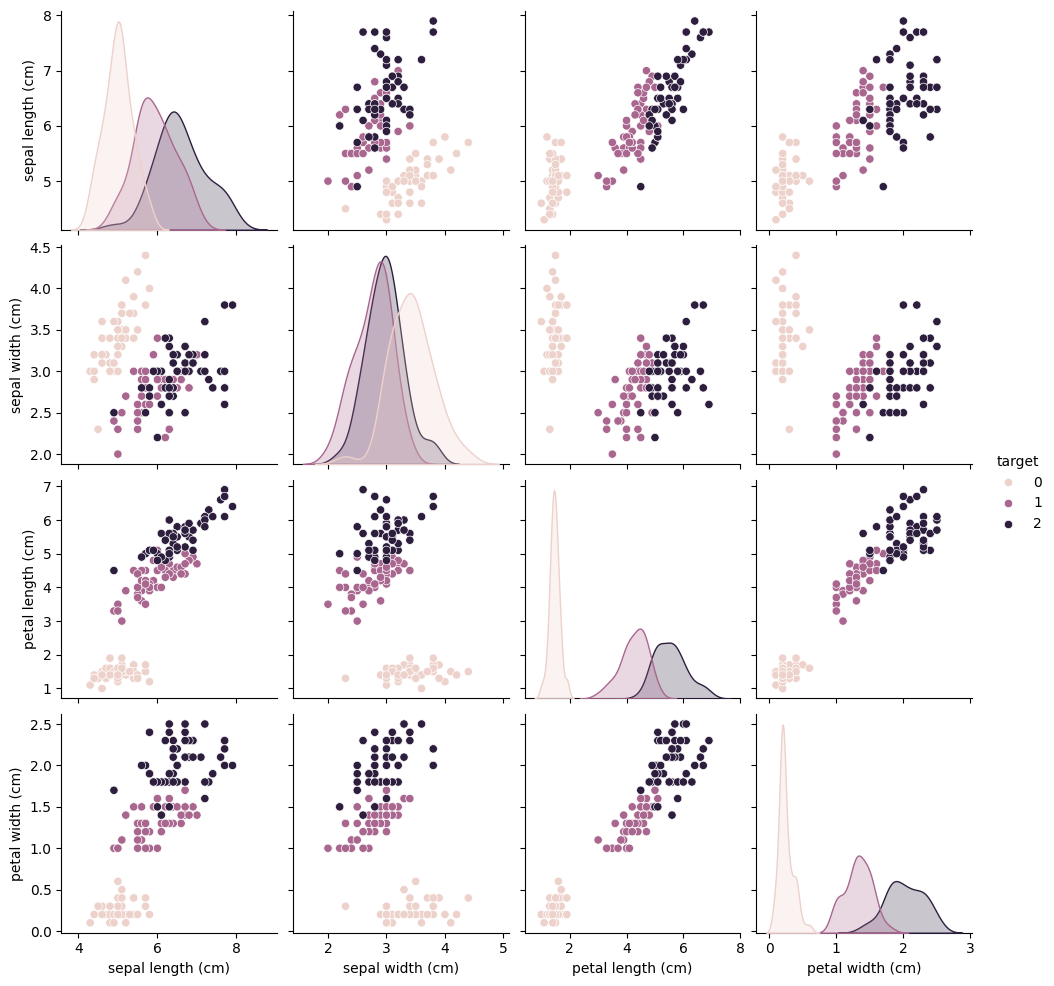
# check info

iris\_unsupervised.info()

iris\_unsupervised.describe()

# Exploring the data by visualizing with a pairplot.

sns.pairplot(iris\_data, hue='target')



## Modelling

We use SVC, Decision tree classifier and Simple Random forest classifier to model our dataset. It involves splitting the data using sklearn.train\_test\_split.

# Split data using sklearn.train\_test\_split

x=iris\_data.drop('target', axis=1)

y=iris\_data['target']

x\_train, x\_test, y\_train, y\_test=train\_test\_split(x, y, train\_size=0.7, random\_state=42)

train\_size=0.7 implies that 70% of data are used for training and 30% is used for testing. Random\_state=42 is for maintaining the consistency of the model.

# Checking the number of sample rows and columns of x-test

x\_test.shape

(45, 4)

# Instantiate the classification models

svc= SVC()

decision\_tree=DecisionTreeClassifier()

random\_forest=RandomForestClassifier()

models=[svc, decision\_tree, random\_forest]

iris\_data['target'].value\_counts()

target

0 50

1 50

2 50

Name: count, dtype: int64

# Get optimum cluster by using KMeans and plotting the graph.

inertias=[]

clusters=list(range(1,11))

for cluster in clusters:

    kmeans=KMeans(n\_clusters=cluster)

    kmeans.fit(iris\_unsupervised)

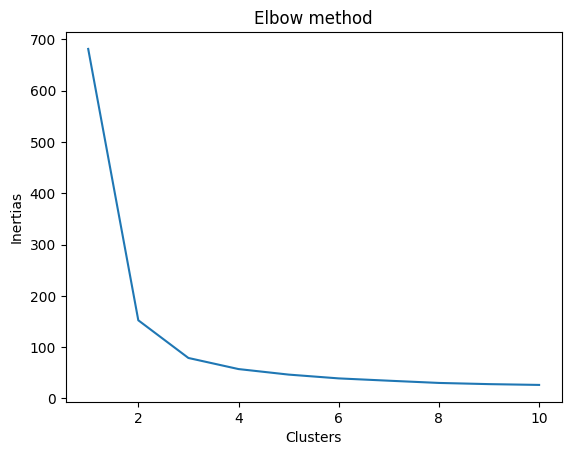
    inertias.append(kmeans.inertia\_)

plt.plot(clusters, inertias)

plt.xlabel("Clusters")

plt.ylabel("Inertias")

plt.title('Elbow method')



From the diagram it is clearly seen that the optimum elbow is at cluster 3.

# Instantiate the model

kmeans=KMeans(n\_clusters=3)

#Fit and predict

labels=kmeans.fit\_predict(iris\_unsupervised)

'''

Evaluate the algorithm

High silhouette score(close to 1) implies dense, well-separated clusters

with good clustering structure implies the inverse'''

metrics.silhouette\_score(iris\_unsupervised, labels)

0.5528190123564102

# plot the clusters using matplotlib

plt.figure(figsize=(12,8))# defining size of the figure.

plt.scatter(iris\_unsupervised['petal length (cm)'],

            iris\_unsupervised['petal width (cm)'],

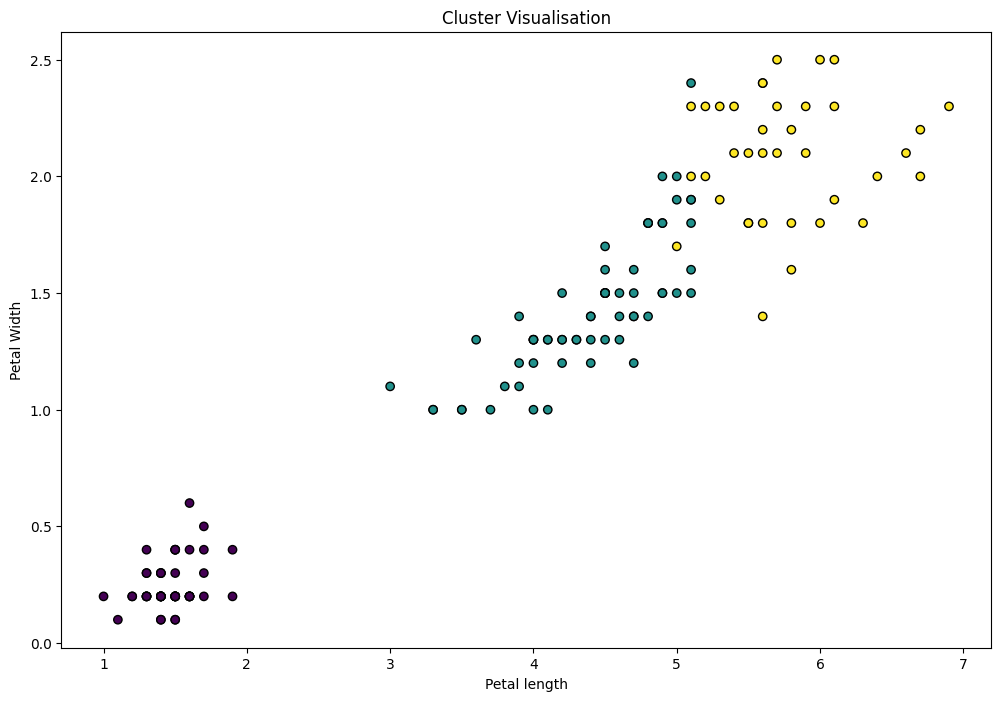
             c = labels, edgecolors="k")

plt.xlabel("Petal length")

plt.ylabel("Petal Width")

plt.title("Cluster Visualisation")

plt.show()



## Hierarchical Clustering

# create a cluster matrix

linkage\_matrix=linkage(iris\_unsupervised, method='ward', metric='euclidean')

# plot dendrogan

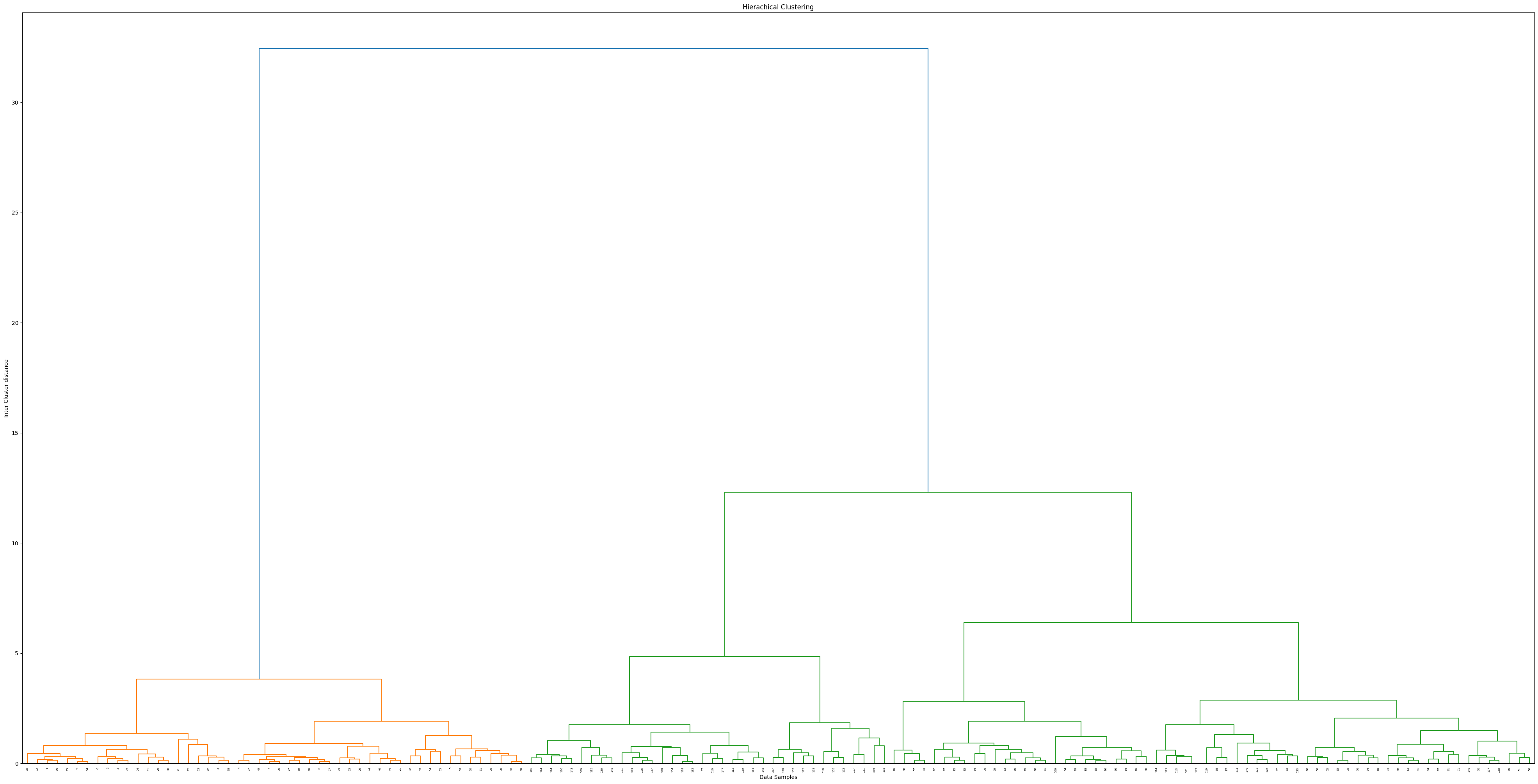
plt.figure(figsize=(50,25))

dendrogram(linkage\_matrix)

plt.xlabel("Data Samples")

plt.ylabel("Inter Cluster distance")

plt.title("Hierachical Clustering")



# Obtain the labels

labels\_hierarchical=fcluster(linkage\_matrix, 3,criterion='maxclust')

metrics.silhouette\_score(iris\_unsupervised, labels\_hierarchical)

#metrics.accuracy\_score(iris\_unsupervised, labels\_hierarchical)

0.5543236611296426

# plot the clusters

plt.figure(figsize=(12,8))

plt.scatter(iris\_unsupervised['petal length (cm)'],

            iris\_unsupervised['petal width (cm)'],

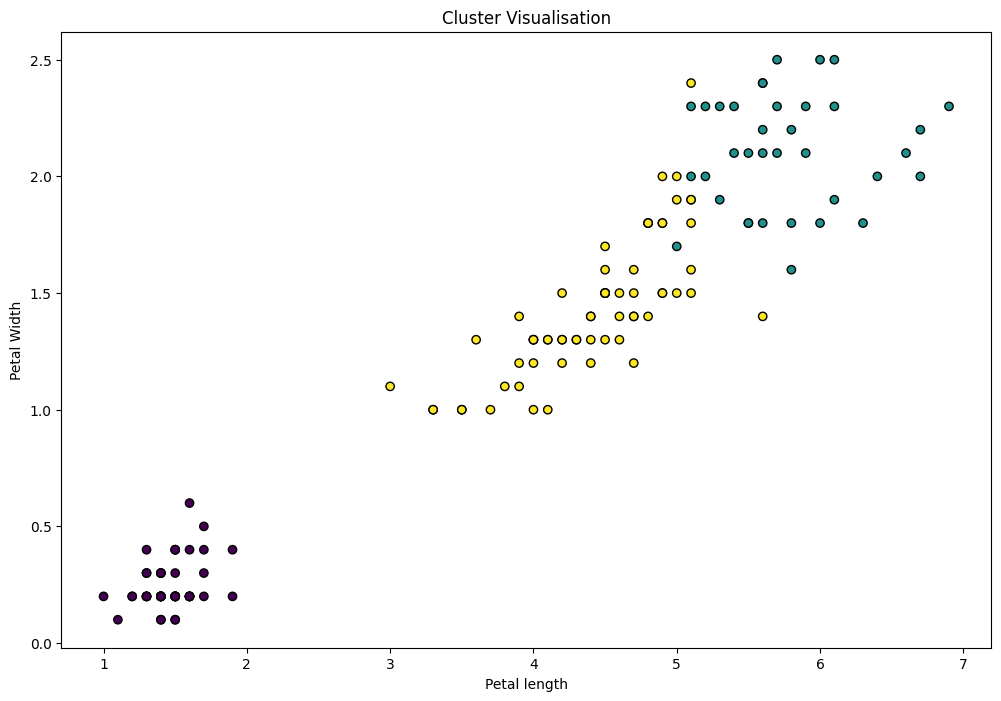
             c = labels\_hierarchical, edgecolors="k")

plt.xlabel("Petal length")

plt.ylabel("Petal Width")

plt.title("Cluster Visualisation")

plt.show()



#Training the models

for model in models:

    model.fit(x\_train, y\_train)

    predictions=model.predict(x\_test)

    accuracy=metrics.accuracy\_score(y\_test, predictions)

    print(f'This {model} has accuracy of {accuracy}')

This SVC() has accuracy of 1.0

This DecisionTreeClassifier() has accuracy of 1.0

This RandomForestClassifier() has accuracy of 1.o

All models have the same accuracy score hence any can be used for training.

Saving the models

# Save the models

joblib.dump(random\_forest, 'random\_forest\_model.pkl')

joblib.dump(svc, 'svc\_model.pk1')

joblib.dump(decision\_tree, "decision\_tree\_classifier.joblib")

Models are saved in the same folder as the app.py and jupyter notebook file.

# 2.App.py

This application provides a straightforward and user-friendly interface for predicting the species of iris flowers based on their structural measurement. It utilizes streamlit library of python.

# import libraries

import joblib

import streamlit as st

#Define the functions and variables

with open('svc\_model.pk1', 'rb') as model:

    classifier=joblib.load(model)

def predictor(sepal\_Length, sepal\_width, petal\_Length,petal\_width):

    global classifier

    prediction= classifier.predict([[sepal\_Length, sepal\_width, petal\_Length,petal\_width ]])

    if prediction == 0:

        return 'satosa'

    elif prediction ==1:

        return 'versicolor'

    else:

        return 'virginica'

def main():

    st.title('Iris Prediction app')

    # Body

    sepal\_length=st.number\_input('Sepal Length')

    sepal\_width=st.number\_input('Sepal Width')

    petal\_length=st.number\_input('Petal Length')

    petal\_Width=st.number\_input('petal Width')

# Predict

    if st.button('Predict'):

        prediction=predictor(sepal\_length, sepal\_width, petal\_length, petal\_Width)

        st.success(f'The flower is an iris {prediction}')

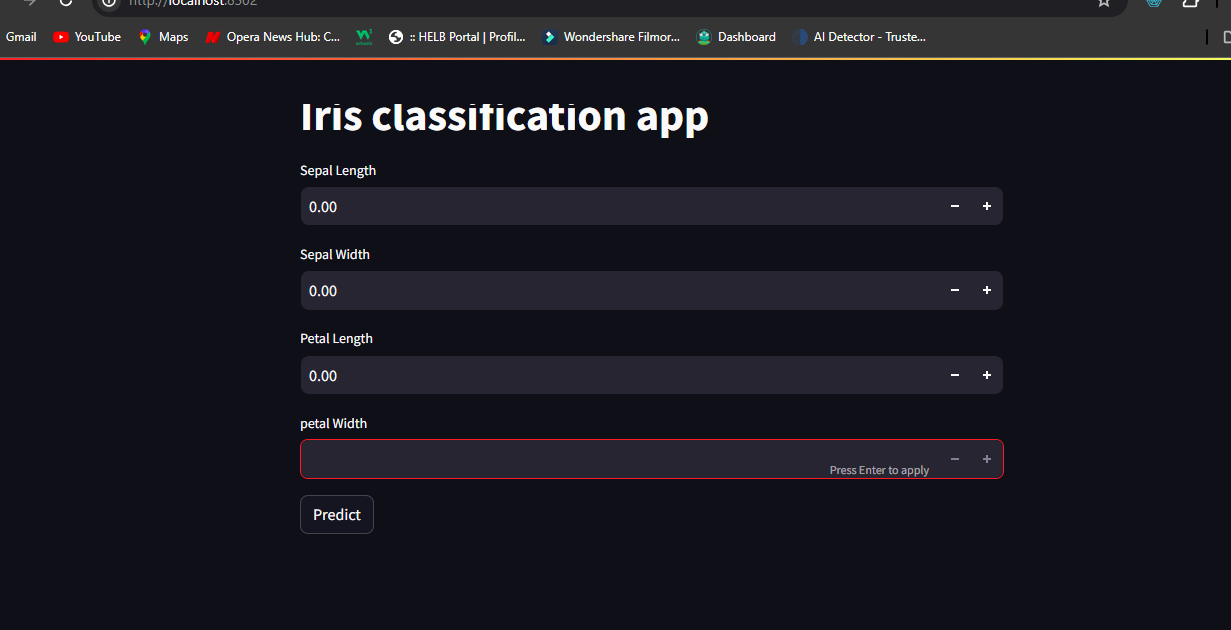
if \_\_name\_\_=="\_\_main\_\_":

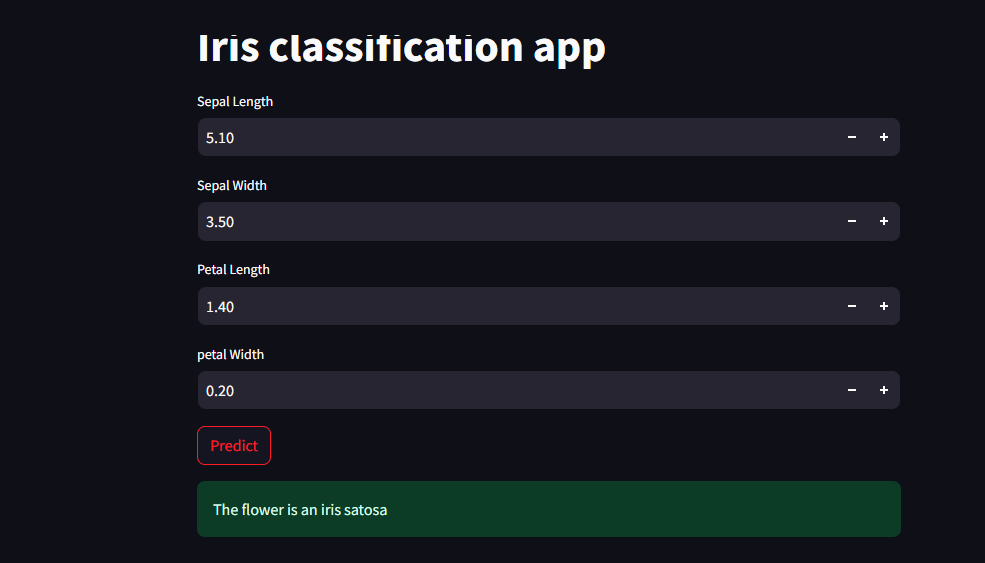
    main()

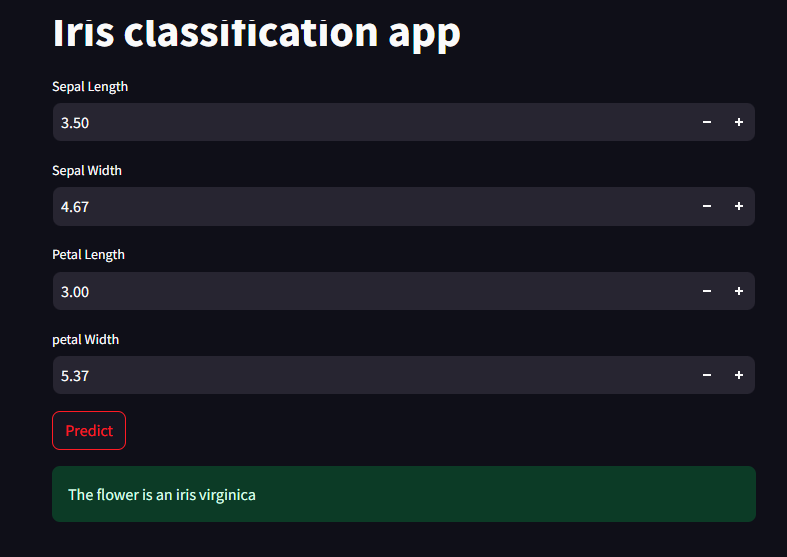
It loads a pre-trained model from a file named using the joblib library. For our case, it loads Support Vector Classifier (SVC) from a file named 'svc\_model.pk1'.

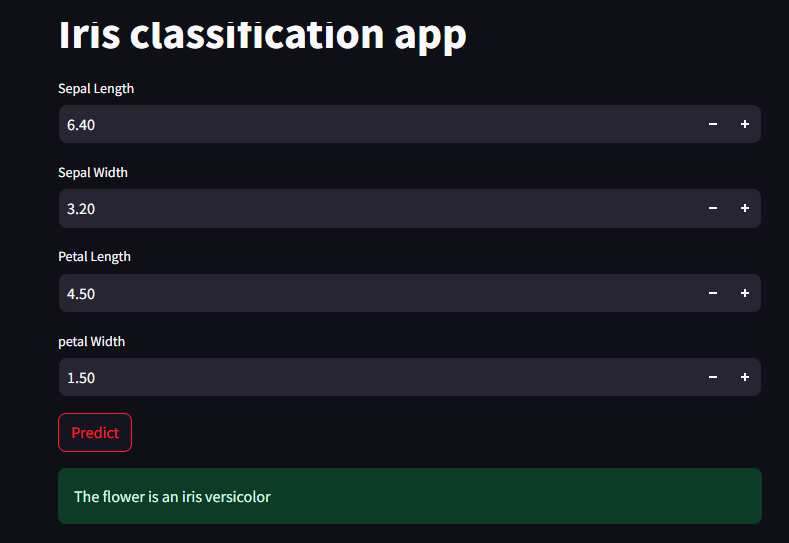
To use the application, users input the sepal length, sepal width, petal length, and petal width of the iris flower through numerical input fields provided by the Streamlit interface. Upon clicking the "Predict" button, the application uses the input values to make a prediction using the loaded SVC model. The predicted species of the iris flower (Setosa, Versicolor, or Virginica) is then displayed to the user.

App outlook when runned.









# Reference

https://www.kaggle.com/datasets?search=iris+flower

https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html