**✅ Step 1: Install Required Libraries**

Make sure you have the libraries:

pip install scikit-learn pandas matplotlib

**✅ Step 2: Import Libraries**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

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

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

**✅ Step 3: Load Dataset**

We’ll use the **Iris dataset** for simplicity.

iris = load\_iris()

X = iris.data

y = iris.target

# Convert to DataFrame for better visualization

df = pd.DataFrame(X, columns=iris.feature\_names)

df['target'] = y

print(df.head())

**✅ Step 4: Split Data**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

**✅ Step 5: Standardize Features**

KNN is distance-based, so scaling is important.

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

**✅ Step 6: Train KNN Model**

Start with **k=5**.

knn = KNeighborsClassifier(n\_neighbors=5)

knn.fit(X\_train, y\_train)

**✅ Step 7: Predictions & Evaluation**

y\_pred = knn.predict(X\_test)

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))

print("Classification Report:\n", classification\_report(y\_test, y\_pred))

**✅ Step 8: Tune ‘k’ (Number of Neighbors)**

Find the best **k** using an accuracy plot:

k\_range = range(1, 21)

scores = []

for k in k\_range:

knn = KNeighborsClassifier(n\_neighbors=k)

knn.fit(X\_train, y\_train)

scores.append(accuracy\_score(y\_test, knn.predict(X\_test)))

plt.plot(k\_range, scores, marker='o')

plt.xlabel('Number of Neighbors (k)')

plt.ylabel('Accuracy')

plt.title('KNN Accuracy vs k')

plt.show()

print("Best Accuracy:", max(scores), "at k =", scores.index(max(scores)) + 1)

**✅ Key Notes**

* **KNN** = Non-parametric, lazy learning algorithm.
* Works by finding the **k nearest points** (based on distance) and classifying by **majority vote**.
* **Distance metric**: Default = Euclidean.
* **Scaling** is important.
* Larger **k** → smoother decision boundary, less variance but more bias.
* Smaller **k** → more variance, can overfit.

👉 Do you want me to:  
✔ **Combine everything into one complete Python script**,  
✔ **Create a Jupyter Notebook with step-by-step explanations and visualizations**, or  
✔ **Generate a PDF report with explanations, code, outputs, and plots**?