**✅ 1. Choose a classification dataset and normalize features**

We’ll use the **Iris dataset** for simplicity and apply **StandardScaler** for normalization.

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

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

from sklearn.preprocessing import StandardScaler

# Load dataset

iris = load\_iris()

X = iris.data[:, :2] # Use first two features for visualization

y = iris.target

# Split dataset

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

# Normalize features

scaler = StandardScaler()

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

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

**✅ 2. Use KNeighborsClassifier from sklearn**

from sklearn.neighbors import KNeighborsClassifier

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

# Initialize model with k=5

knn = KNeighborsClassifier(n\_neighbors=5)

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

# Predictions

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

**✅ 3. Experiment with different values of K**

k\_range = range(1, 21)

accuracies = []

for k in k\_range:

knn = KNeighborsClassifier(n\_neighbors=k)

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

acc = accuracy\_score(y\_test, knn.predict(X\_test))

accuracies.append(acc)

# Plot accuracy vs K

plt.figure()

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

plt.title("K vs Accuracy")

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

plt.ylabel("Accuracy")

plt.show()

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

**✅ 4. Evaluate model using accuracy and confusion matrix**

print("Accuracy for k=5:", accuracy\_score(y\_test, y\_pred))

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

**✅ 5. Visualize Decision Boundaries**

For decision boundaries, we’ll use the **first two features** for 2D visualization.

from matplotlib.colors import ListedColormap

def plot\_decision\_boundaries(X, y, model, title):

x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1

y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, 0.01),

np.arange(y\_min, y\_max, 0.01))

Z = model.predict(np.c\_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, alpha=0.4, cmap=ListedColormap(('red', 'green', 'blue')))

plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', cmap=ListedColormap(('red', 'green', 'blue')))

plt.title(title)

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.show()

# Plot for k=5

plot\_decision\_boundaries(np.vstack((X\_train, X\_test)), np.hstack((y\_train, y\_test)), knn, "KNN Decision Boundary (k=5)")

**✅ Summary of Results**

* Accuracy changes with **k** (larger k = smoother decision boundary).
* Best accuracy is found by plotting accuracy vs **k**.
* Decision boundaries show how KNN classifies based on nearest neighbors.

👉 Do you want me to **combine all steps into one ready-to-run Python script**, or **create a Jupyter Notebook with explanations, plots, and outputs**, or **generate a full PDF report with code, results, and visualizations**?