****

**AI MSE REPORT**

**Title Page**

**Problem Statement:** Iris Flower Classification

**Personal Details:**

* Name: Aastha Srivastava
* Roll Number: 04
* Date: 11/03/2025
* Course: Introduction To AI
* Instructor: MR. BIKKI GUPTA

**Introduction**

The Iris Classification project uses the **K-Nearest Neighbors (KNN)** algorithm to classify iris flowers into three species: *setosa*, *versicolor*, and *virginica*. The goal is to train a KNN model using an optimized value of k to maximize classification accuracy.

**Methodology**

1. **Data Loading: Loaded the Iris dataset using pandas.**
2. **Data Cleaning: Stripped whitespace and checked for missing values.**
3. **Encoding: Mapped species names to numeric values using factorize.**
4. **Splitting: Split data into training (80%) and test (20%) sets.**
5. **Hyperparameter Tuning: Used GridSearchCV to find the best k value.**
6. **Model Training: Trained the KNN model using the best k.**
7. **Prediction: Predicted species on test data and a new sample.**
8. **Evaluation: Assessed model accuracy and performance using a classification report and confusion matrix.**

**CODE**

**# Import necessary libraries**

**import pandas as pd**

**import numpy as np**

**import matplotlib.pyplot as plt**

**import seaborn as sns**

**from sklearn.model\_selection import train\_test\_split, GridSearchCV**

**from sklearn.neighbors import KNeighborsClassifier**

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

**from sklearn.exceptions import NotFittedError**

**# Step 1: Load the Iris dataset using pandas**

**df = pd.read\_csv('/content/iris\_data.csv')**

**# Step 2: Clean the data (strip any whitespace from column names)**

**df.columns = df.columns.str.strip()**

**# Step 3: Ensure correct column names and data types**

**print("\nColumn Names:", df.columns.tolist())**

**print("\nData Types:\n", df.dtypes)**

**# Step 4: Check for missing values**

**print("\nMissing Values:\n", df.isnull().sum())**

**# Step 5: Map species names to numerical values using factorize (automatic)**

**df['Species'], species\_map = pd.factorize(df['Species'])**

**print("\nSpecies Mapping:", dict(enumerate(species\_map)))**

**# Step 6: Drop rows with missing values (if any)**

**if df.isnull().sum().sum() > 0:**

**df.dropna(inplace=True)**

**# Step 7: Split the data into features (X) and target (y)**

**X = df.drop('Species', axis=1)**

**y = df['Species']**

**# Step 8: Split into training and testing sets (80% train, 20% test)**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(**

**X, y, test\_size=0.2, random\_state=42)**

**# Step 9: Hyperparameter tuning for best 'k' value**

**param\_grid = {'n\_neighbors': range(1, 11)}**

**grid = GridSearchCV(KNeighborsClassifier(), param\_grid, cv=5)**

**grid.fit(X\_train, y\_train)**

**best\_k = grid.best\_params\_['n\_neighbors']**

**print(f"\nBest k value: {best\_k}")**

**# Step 10: Create and train the KNN model with best 'k'**

**knn = KNeighborsClassifier(n\_neighbors=best\_k)**

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

**# Step 11: Predict on the test set**

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

**# Step 12: Evaluate the model (accuracy)**

**accuracy = accuracy\_score(y\_test, y\_pred)**

**print(f"\nTest Accuracy: {accuracy:.2f}")**

**# Step 13: Classification Report (Test Data)**

**target\_names = list(species\_map)**

**print("\nTest Set Classification Report:\n")**

**print(classification\_report(y\_test, y\_pred, target\_names=target\_names))**

**# Step 14: Classification Report (Training Data) — Check for overfitting**

**y\_train\_pred = knn.predict(X\_train)**

**print("\nTraining Set Classification Report:\n")**

**print(classification\_report(y\_train, y\_train\_pred, target\_names=target\_names))**

**# Step 15: Confusion Matrix**

**cm = confusion\_matrix(y\_test, y\_pred)**

**plt.figure(figsize=(6, 4))**

**sns.heatmap(cm, annot=True, cmap='Blues', fmt='d', xticklabels=target\_names, yticklabels=target\_names)**

**plt.xlabel('Predicted')**

**plt.ylabel('Actual')**

**plt.title('Confusion Matrix')**

**plt.show()**

**# Step 16: Predict a new sample**

**new\_sample = pd.DataFrame([[5.1, 3.5, 1.4, 0.2]], columns=X.columns)**

**# Step 17: Ensure model is fitted before predicting**

**try:**

**new\_sample = new\_sample.astype(float)**

**predicted\_class = knn.predict(new\_sample)[0]**

**predicted\_species = species\_map[predicted\_class]**

**print(f"\nPredicted class for sample {new\_sample.iloc[0].values}: {predicted\_species}")**

**except NotFittedError as e:**

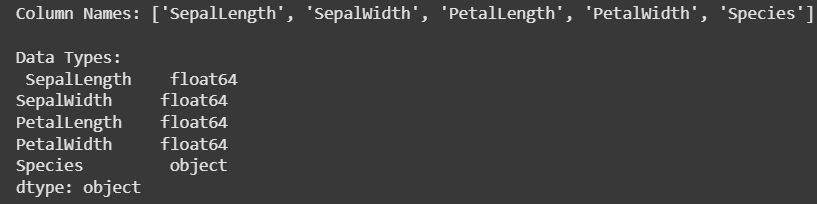
**print(f"Model Error: {e}")**

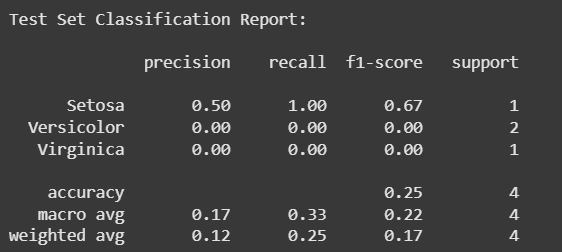
**except ValueError as e:**

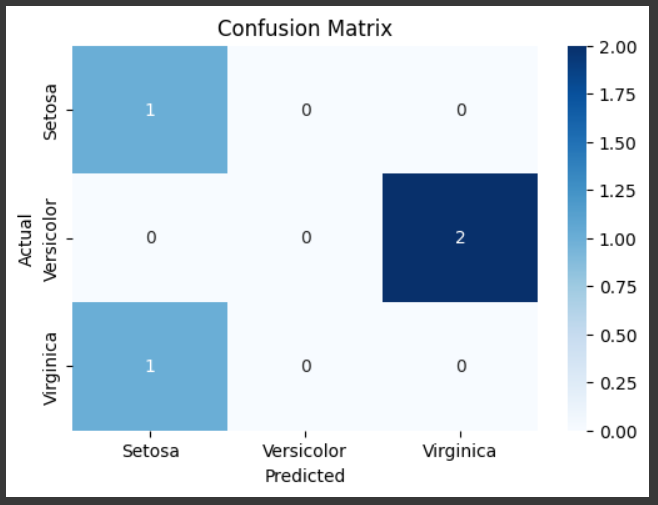
**print(f"Value Error: {e}")**

**Output/Result**

The following are the results obtained from the analysis:







**References/Credits**

* Scikit-learn Documentation – <https://scikit-learn.org>
* Python Documentation – <https://docs.python.org>