**Project Report: Iris Flower Classification**

**1. Project Overview**

**Objective**:  
The goal of this project is to build a machine learning model to classify Iris flowers into three species (*Setosa*, *Versicolor*, *Virginica*) based on their petal and sepal dimensions.

**Dataset**:  
The Iris dataset is a small and clean dataset with:

* 150 samples
* 4 numerical features:
  + Sepal Length
  + Sepal Width
  + Petal Length
  + Petal Width
* 3 target classes:
  + *Setosa*
  + *Versicolor*
  + *Virginica*

**2. Data Exploration**

**Key Insights**:

* The dataset contains no missing values or outliers, making it clean and ready for analysis.
* The distribution of target classes is balanced:
  + 50 samples per class (*Setosa*, *Versicolor*, *Virginica*).
* Petal dimensions are more effective in distinguishing between species than sepal dimensions.

**Visual Analysis**:

* A pair plot reveals that *Setosa* is linearly separable from the other two species based on petal dimensions.
* *Versicolor* and *Virginica* show slight overlap, indicating they may require more advanced classification techniques.

**3. Data Preprocessing and Feature Engineering**

* **Preprocessing Steps**:
  + The target variable (species) was encoded as integers (0, 1, 2) corresponding to the three species.
  + The dataset was split into features (X) and the target (y).
  + An 80-20 train-test split was applied to create training and testing datasets.
* **Feature Engineering**:
  + No additional features were created as the dataset already contains meaningful and sufficient features for classification.

**4. Model Selection and Training**

**Algorithm Used**:

* Random Forest Classifier:
  + Chosen for its robustness and ability to handle multi-class classification.
  + Parameters:
    - Number of trees (n\_estimators): 100
    - Random state: 42 (for reproducibility)

**Training**:

* The model was trained on 80% of the data using scikit-learn's RandomForestClassifier.

**5. Model Evaluation**

* **Metrics**:
  + Accuracy: 100% (on the test set).
  + Precision, Recall, and F1-Score were perfect for all three classes.
* **Confusion Matrix**:
  + The model correctly classified all 30 test samples (20% of the dataset).

**Discussion of Model Performance:**

* **Strengths**:
  + **High Accuracy**: Achieved 100% accuracy on the test data due to the simplicity and linear separability of the dataset.
  + **Balanced Dataset**: Each class had an equal number of samples (50 per class), which helped avoid bias.
  + **Interpretable Results**: The Random Forest model provided feature importances, making it easy to understand which features contributed the most to the predictions.
* **Limitations**:
  + **Simple Dataset**: The Iris dataset is relatively small (150 samples) and lacks noise or real-world complexities.
  + **Limited Generalizability**: While the model performs perfectly on this dataset, it may not generalize as well to more complex datasets with overlapping classes or noise.
* **Insights**:
  + Petal dimensions (length and width) are the most significant features for distinguishing between species.
  + *Setosa* is distinctly separable from the other two species, while *Versicolor* and *Virginica* have overlapping regions that require more complex decision boundaries.

**6. Conclusion**

* **Project Outcome**:
  + Successfully built and evaluated a machine learning model to classify Iris flowers with 100% accuracy on the test set.
  + Demonstrated effective use of a Random Forest Classifier for multi-class classification.
* **Key Insights**:
  + Petal dimensions (length and width) are the most important features for distinguishing between species.
  + Simple algorithms like Random Forest can achieve excellent results when the dataset is well-prepared and clean.
* **Limitations**:
  + The dataset is small, which may not generalize well to larger, more complex datasets.
  + The problem is relatively simple, and advanced techniques may not yield significant improvement.
* **Future Work**:
  + Test other classification algorithms, such as Support Vector Machines (SVM) or Neural Networks.
  + Deploy the model using a web application (e.g., Flask or Fast API) for real-time predictions.
  + Evaluate the model on a larger dataset with more complex and noisy data.

**SOURCE CODE:**

**# Import necessary libraries**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.ensemble import RandomForestClassifier

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

**# Load the dataset**

def load\_data():

iris = load\_iris()

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

data['species'] = iris.target

target\_mapping = {0: "setosa", 1: "versicolor", 2: "virginica"}

data['species'] = data['species'].map(target\_mapping)

print("Dataset Loaded Successfully!")

return data

data = load\_data()

**# Data Exploration**

def explore\_data(data):

print("\nDataset Overview:")

print(data.head())

print("\nDataset Summary:")

print(data.info())

print("\nDataset Statistics:")

print(data.describe())

print("\nVisualizing pairplot:")

sns.pairplot(data, hue='species', palette='Set1')

plt.show()

explore\_data(data)

**# Preprocessing**

def preprocess\_data(data):

**# Split data into features (X) and target (y)**

X = data.drop(columns=['species'])

y = data['species']

**# Train-test split**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

return X\_train, X\_test, y\_train, y\_test

X\_train, X\_test, y\_train, y\_test = preprocess\_data(data)

**# Model Selection and Training**

def train\_model(X\_train, y\_train):

model = RandomForestClassifier(random\_state=42, n\_estimators=100)

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

print("\nModel Trained Successfully!")

return model

model = train\_model(X\_train, y\_train)

**# Model Evaluation**

def evaluate\_model(model, X\_test, y\_test):

# Predictions

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

**# Classification Report**

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

**# Accuracy Score**

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

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

**# Confusion Matrix**

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

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=model.classes\_, yticklabels=model.classes\_)

plt.title("Confusion Matrix")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.show()

return accuracy

accuracy = evaluate\_model(model, X\_test, y\_test)

**# Save the Model**

def save\_model(model, file\_name="iris\_classifier.pkl"):

import joblib

joblib.dump(model, file\_name)

print(f"Model saved as {file\_name}")

save\_model(model)