Iris Flower Classification Project Report

# Project Title

Iris Flower Classification Using Machine Learning Algorithms

# Objective

The primary goal of this project is to build and evaluate machine learning models that can accurately classify Iris flowers into three species—Setosa, Versicolor, and Virginica—based on their sepal and petal dimensions.

# Dataset Description

Source: The Iris dataset is a well-known dataset available in seaborn and sklearn libraries.

Features:

* • Sepal Length (cm)  
  • Sepal Width (cm)  
  • Petal Length (cm)  
  • Petal Width (cm)

Target Variable: species (Setosa, Versicolor, Virginica)

# Exploratory Data Analysis

• The dataset contains 150 samples (50 per species).  
• No missing values were found.  
• A pairplot was used to visualize feature relationships.  
• A correlation heatmap highlighted strong correlation between petal length and petal width.

# Data Preprocessing

• The target variable was label-encoded (Setosa=0, Versicolor=1, Virginica=2).  
• Features and target were separated into X and y.  
• Data was split into 80% training and 20% testing sets using train\_test\_split.

# Machine Learning Models Used

|  |  |
| --- | --- |
| Model | Description |
| K-Nearest Neighbors | Classifies based on closest data points |
| Logistic Regression | Probabilistic linear classifier |
| Decision Tree | Tree-based flowchart decision structure |
| Support Vector Machine | Optimal hyperplane-based classification |

# Model Evaluation

Each model was evaluated using accuracy score and a classification report.

|  |  |
| --- | --- |
| Model | Accuracy |
| K-Nearest Neighbors | ~96.7% |
| Logistic Regression | ~100% |
| Decision Tree | ~100% |
| Support Vector Machine | ~100% |

Note: Performance may vary slightly due to train-test split randomness.

# Conclusion

All models performed well on the Iris dataset, with Logistic Regression, Decision Tree, and SVM achieving perfect accuracy. KNN also performed well and is easy to implement. Decision Trees offer better interpretability, while SVM works well for small datasets with clear margin separation.

# Future Work

• Test models on larger or noisier datasets.  
• Perform hyperparameter tuning using GridSearchCV.  
• Deploy the best model using Flask or Streamlit.

# Code Snippets

## Importing Libraries

import pandas as pd  
import seaborn as sns  
import matplotlib.pyplot as plt  
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import LabelEncoder  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.linear\_model import LogisticRegression  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.svm import SVC  
from sklearn.metrics import accuracy\_score, classification\_report

## Loading Dataset

df = sns.load\_dataset("iris")

## Preprocessing

le = LabelEncoder()  
df['species'] = le.fit\_transform(df['species'])  
  
X = df.drop('species', axis=1)  
y = df['species']  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

## Model Training and Evaluation

models = {  
 "K-Nearest Neighbors": KNeighborsClassifier(n\_neighbors=3),  
 "Logistic Regression": LogisticRegression(max\_iter=200),  
 "Decision Tree": DecisionTreeClassifier(),  
 "Support Vector Machine": SVC()  
}  
  
for name, model in models.items():  
 print(f"\n--- {name} ---")  
 model.fit(X\_train, y\_train)  
 y\_pred = model.predict(X\_test)  
 print("Accuracy:", accuracy\_score(y\_test, y\_pred))  
 print("Classification Report:")  
 print(classification\_report(y\_test, y\_pred))