task-1-1

July 14, 2025

0.1 Task 1 : Iris Flower Classification

0.2 Problem Statement:

The Iris flower dataset contains measurements of three species of iris flowers: Setosa, Versicolor, and Virginica. Each species differs in physical characteristics such as sepal length, sepal width, petal length, and petal width.The objective of this project is to develop a machine learning model that can classify a given iris flower into its correct species based on these four features.

0.3 1 Import libraries

[79]:
```
# Import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, accuracy_score,␣
↪confusion_matrix
```

0.4 2 Load Iris dataset

[80]:
```
# Load data
df = pd.read_csv('Iris (1).csv')
```

[81]:
```
df.dtypes
```

[81]:
```
Id                 int64
SepalLengthCm    float64
SepalWidthCm     float64
PetalLengthCm    float64
PetalWidthCm     float64
Species           object
dtype: object
```

[82]:
```
df.shape
```

[82]:
```
(150, 6)
```

[83]:
```
df.ndim
```

[83]:
```
2
```

[84]:
```
df.isnull()
```

[84]:
```
        Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  Species
0    False          False         False          False         False    False
1    False          False         False          False         False    False
2    False          False         False          False         False    False
3    False          False         False          False         False    False
4    False          False         False          False         False    False
..     …              …             …              …             …        …
145  False          False         False          False         False    False
146  False          False         False          False         False    False
147  False          False         False          False         False    False
148  False          False         False          False         False    False
149  False          False         False          False         False    False

[150 rows x 6 columns]
```

[85]:
```
df.describe()
```

[85]:
```
               Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
count  150.000000     150.000000    150.000000     150.000000    150.000000
mean    75.500000       5.843333      3.054000       3.758667      1.198667
std     43.445368       0.828066      0.433594       1.764420      0.763161
min      1.000000       4.300000      2.000000       1.000000      0.100000
25%     38.250000       5.100000      2.800000       1.600000      0.300000
50%     75.500000       5.800000      3.000000       4.350000      1.300000
75%    112.750000       6.400000      3.300000       5.100000      1.800000
max    150.000000       7.900000      4.400000       6.900000      2.500000
```

[86]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object
```

```
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

0.5 3 Data preprocessing

1.Handling missing values (if any)

2Label encoding the target

3.Feature scaling (normalization or standardization)

4.Train-test split

[87]:
```python
df.isnull().sum()
```

[87]:
```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

[88]:
```python
# Drop 'Id' column if it exists
if 'Id' in df.columns:
    df.drop(columns=['Id'], inplace=True)
```

[89]:
```python
# Encode target labels
le = LabelEncoder()
df['Species'] = le.fit_transform(df['Species']) # setosa=0, versicolor=1,␣
 ↪virginica=2
```

[90]:
```python
# Split features and labels
X = df.drop('Species', axis=1)
y = df['Species']
```

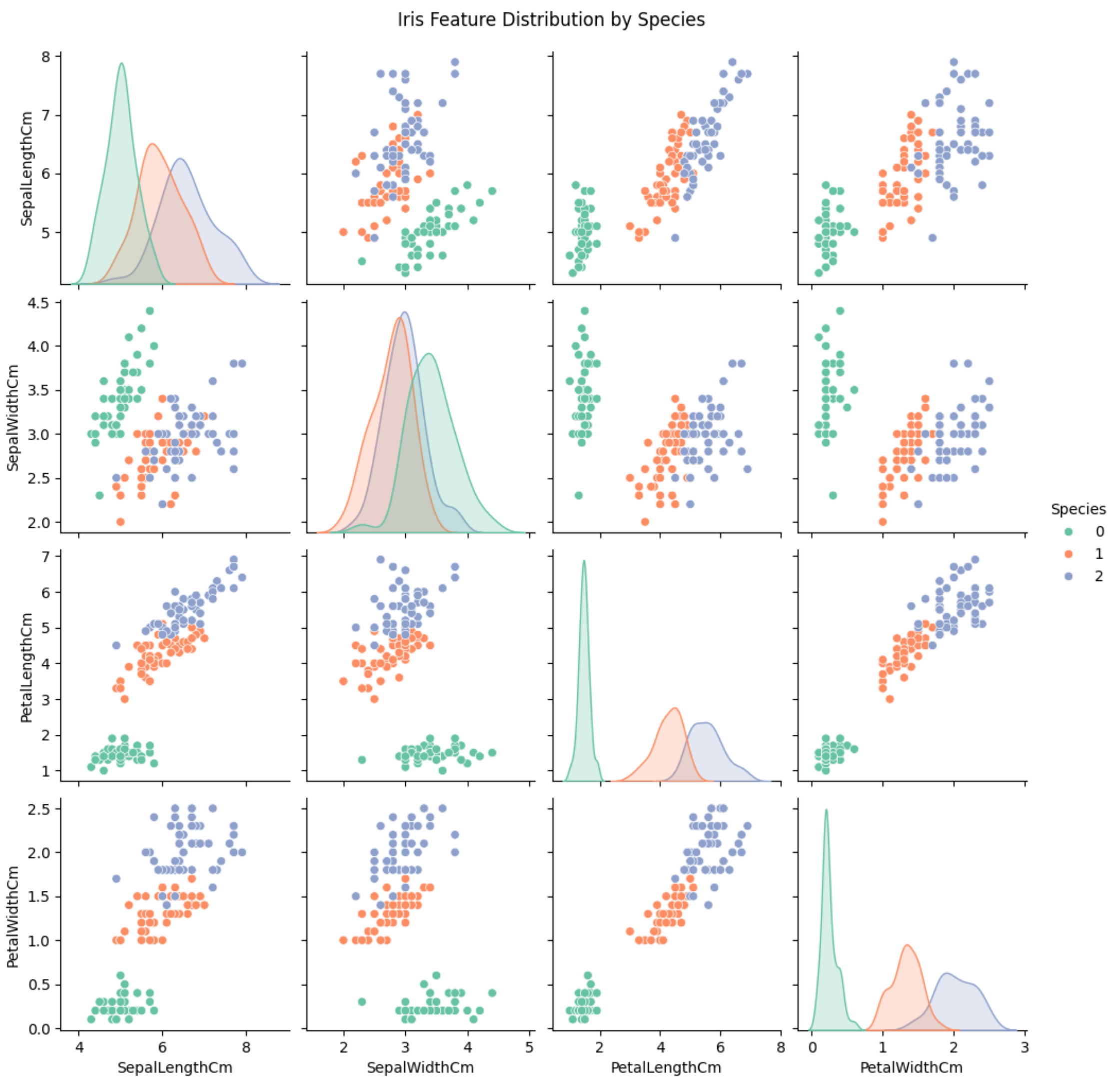
[91]:
```python
#Split the dataset into training and testing sets (80/20)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[92]:
```python
# Standardize features (important for models like Logistic Regression, SVM, KNN)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

0.6 4 Visualize Dataset Distribution

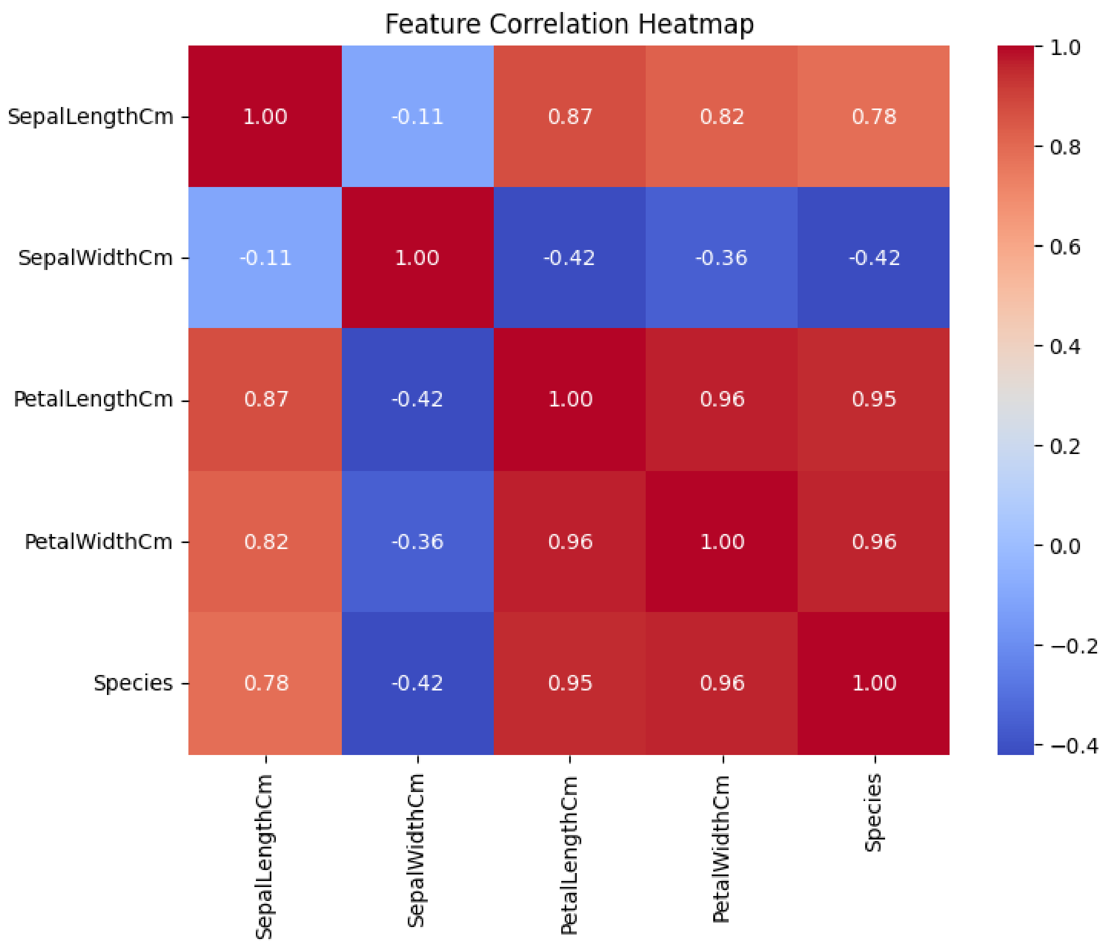
[93]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.pairplot(df, hue='Species', palette='Set2')
plt.suptitle('Iris Feature Distribution by Species', y=1.02)
plt.show()
```



0.7 Heatmap of Feature Correlation

[94]:
```
plt.figure(figsize=(8,6))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Feature Correlation Heatmap')
plt.show()
```



0.8 Count Plot of Each Species

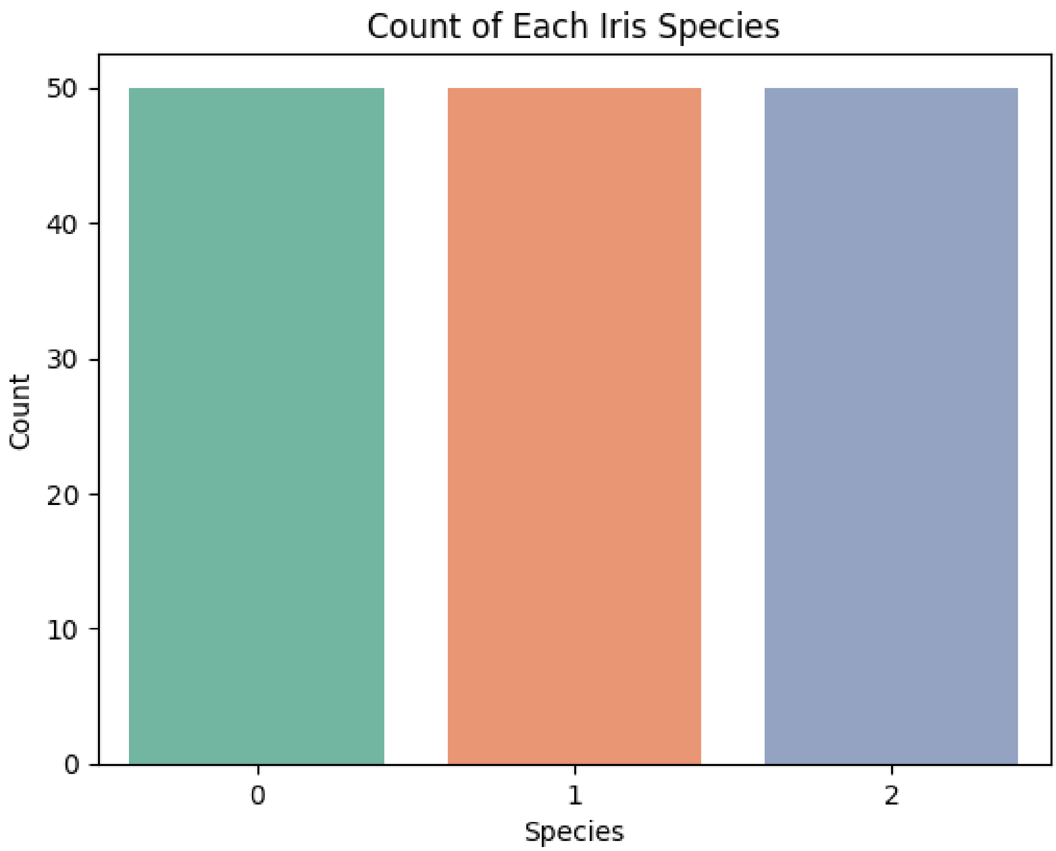
[96]:
```
sns.countplot(x='Species', data=df, palette='Set2')
plt.title('Count of Each Iris Species')
plt.xlabel('Species')
plt.ylabel('Count')
plt.show()
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_14136\217132003.py:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
```

v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
sns.countplot(x='Species', data=df, palette='Set2')
```



[ ]:

```

```

0.9 1. Logistic Regression

[97]:
```
from sklearn.linear_model import LogisticRegression

log_model = LogisticRegression()
log_model.fit(X_train_scaled, y_train)
y_pred_log = log_model.predict(X_test_scaled)

print("  Logistic Regression")
print("Accuracy:", accuracy_score(y_test, y_pred_log))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_log))
print("Classification Report:\n", classification_report(y_test, y_pred_log))
```

```
 Logistic Regression
Accuracy: 1.0
Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

0.10 2. Decision Tree

[98]:
```python
from sklearn.tree import DecisionTreeClassifier

dt_model = DecisionTreeClassifier()
dt_model.fit(X_train, y_train)
y_pred_dt = dt_model.predict(X_test)

print("  Decision Tree")
print("Accuracy:", accuracy_score(y_test, y_pred_dt))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_dt))
print("Classification Report:\n", classification_report(y_test, y_pred_dt))
```

```
 Decision Tree
Accuracy: 1.0
Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

0.11 3. K-Nearest Neighbors (KNN)

[99]:
```
from sklearn.neighbors import KNeighborsClassifier

knn_model = KNeighborsClassifier()
knn_model.fit(X_train_scaled, y_train)
y_pred_knn = knn_model.predict(X_test_scaled)

print("  K-Nearest Neighbors")
print("Accuracy:", accuracy_score(y_test, y_pred_knn))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_knn))
print("Classification Report:\n", classification_report(y_test, y_pred_knn))
```

```
  K-Nearest Neighbors
Accuracy: 1.0
Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

0.12 4. Support Vector Machine (SVM)

[100]:
```
from sklearn.svm import SVC

svm_model = SVC()
svm_model.fit(X_train_scaled, y_train)
y_pred_svm = svm_model.predict(X_test_scaled)

print("  Support Vector Machine")
print("Accuracy:", accuracy_score(y_test, y_pred_svm))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_svm))
print("Classification Report:\n", classification_report(y_test, y_pred_svm))
```

```
  Support Vector Machine
Accuracy: 1.0
Confusion Matrix:
 [[10  0  0]
```

```
 [ 0  9  0]
 [ 0  0 11]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

0.13 5. Random Forest

[101]:
```python
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)

print("  Random Forest")
print("Accuracy:", accuracy_score(y_test, y_pred_rf))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_rf))
print("Classification Report:\n", classification_report(y_test, y_pred_rf))
```

```
  Random Forest
Accuracy: 1.0
Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

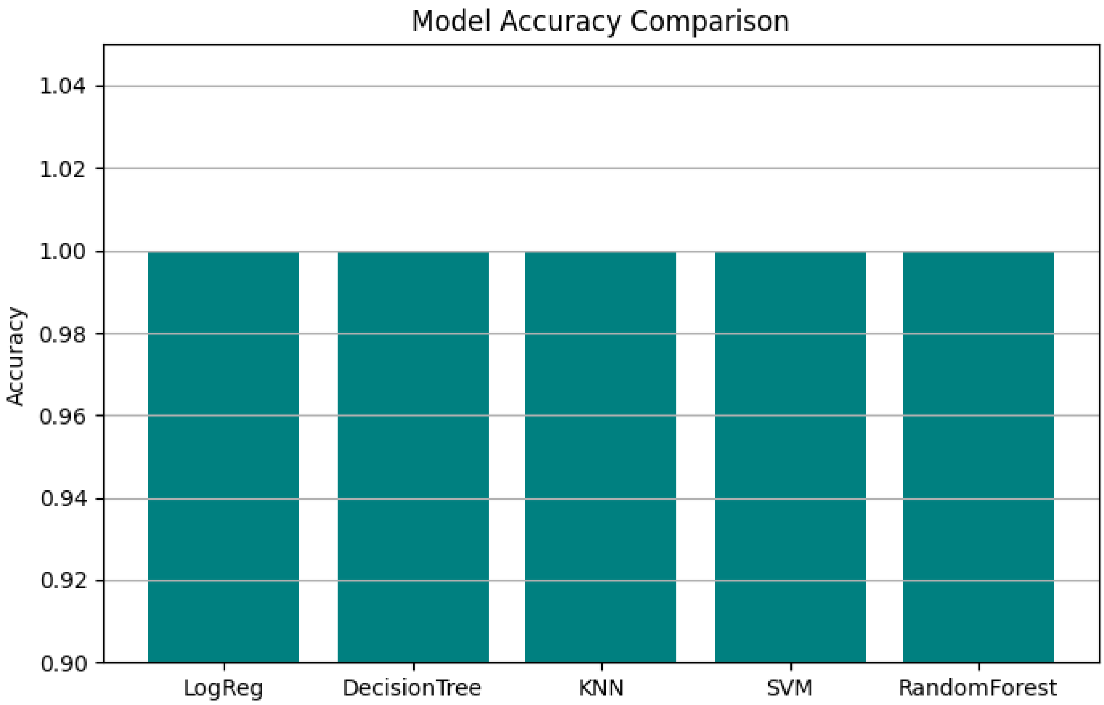
[ ]:

[102]:

```
import matplotlib.pyplot as plt

models = ['LogReg', 'DecisionTree', 'KNN', 'SVM', 'RandomForest']
accuracies = [
    accuracy_score(y_test, y_pred_log),
    accuracy_score(y_test, y_pred_dt),
    accuracy_score(y_test, y_pred_knn),
    accuracy_score(y_test, y_pred_svm),
    accuracy_score(y_test, y_pred_rf),
]

plt.figure(figsize=(8, 5))
plt.bar(models, accuracies, color='teal')
plt.title("Model Accuracy Comparison")
plt.ylim(0.9, 1.05)
plt.ylabel("Accuracy")
plt.grid(axis='y')
plt.show()
```



[ ]:

```

```

[16]:

```
#  Predict a sample
sample = pd.DataFrame([[6.0, 2.7, 4.5, 1.5]], columns=X.columns)
y_pred = model.predict(sample)
```

```
print("Predicted Species:", le.inverse_transform(y_pred))
```

```
Predicted Species: ['Iris-versicolor']
```

[21]:
```
sample = pd.DataFrame([[5.1, 3.5, 1.4, 0.2]], columns=X.columns)
y_pred = model.predict(sample)
print("Predicted Species:", le.inverse_transform(y_pred))
```

```
Predicted Species: ['Iris-setosa']
```

[23]:
```
sample = pd.DataFrame([[6.3, 3.3, 6.0, 2.5]], columns=X.columns)
y_pred = model.predict(sample)
print("Predicted Species:", le.inverse_transform(y_pred))
```

```
Predicted Species: ['Iris-virginica']
```

0.14 Conclusion

In this project, we built and evaluated multiple machine learning models to classify the species of Iris flowers using the classic Iris dataset. The models included:

Logistic Regression

Decision Tree

K-Nearest Neighbors (KNN)

Support Vector Machine (SVM)

Random Forest

We compared the models based on metrics like accuracy, confusion matrix, and classification report. Among all the models, most achieved high accuracy (close to 97–100%) due to the simplicity and separability of the Iris dataset.

[ ]:
```

```