# iris-classification-task1-1

May 7, 2024

PROJECT NAME-Iris Flower Classification

Industry-OASIS INFOBYTE

Contribution-Individual

Member Name-Assala Iddoub

PROJECT OVERVIEW Goal: The goal of this project is to train a machine learning model to classify iris flowers into one of three species (setosa, versicolor, or virginica) based on their measurements, namely sepal length, sepal width, petal length, and petal width.

Technologies Used:

Scikit-learn: Scikit-learn library is used for machine learning tasks, including dataset loading, model training, and evaluation Python: The project is implemented using the Python. NumPy: NumPy library is used for numerical computations and data manipulation. Matplotlib: Matplotlib library is used for data visualization, such as plotting the iris flowers and their measurements.

## 1 IMPORT LIBRARIES

[22]:
```
#for numerical operations
import pandas as pd
# for data manipulation
import numpy as np
#importing tools for visualizations
import matplotlib.pyplot as plt
import seaborn as sns
```

## 2 LOAD DATA

[9]:
```
iris_data= pd.read_csv('C:\\Users\\Assala\\Dropbox\\PC\\Downloads\\archive␣
↪(1)\\iris.csv')
```

[10]:
```
#dataset First Look
print(iris_data)
```

```
     Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  \
0     1            5.1           3.5            1.4           0.2
1     2            4.9           3.0            1.4           0.2
```

```
2      3            4.7           3.2           1.3           0.2
3      4            4.6           3.1           1.5           0.2
4      5            5.0           3.6           1.4           0.2
..   …            …           …           …           …
145  146            6.7           3.0           5.2           2.3
146  147            6.3           2.5           5.0           1.9
147  148            6.5           3.0           5.2           2.0
148  149            6.2           3.4           5.4           2.3
149  150            5.9           3.0           5.1           1.8

            Species
0       Iris-setosa
1       Iris-setosa
2       Iris-setosa
3       Iris-setosa
4       Iris-setosa
..              …
145  Iris-virginica
146  Iris-virginica
147  Iris-virginica
148  Iris-virginica
149  Iris-virginica

[150 rows x 6 columns]
```

## 3 Analyse and visualize

[14]:
```python
#visualize data columns
print(iris_data.columns)
```

```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
```

[25]:
```python
#describe the data
iris_data.describe()
```

[25]:

| | 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 |

[24]:
```
print(iris_data.shape)
```

```
(150, 6)
```

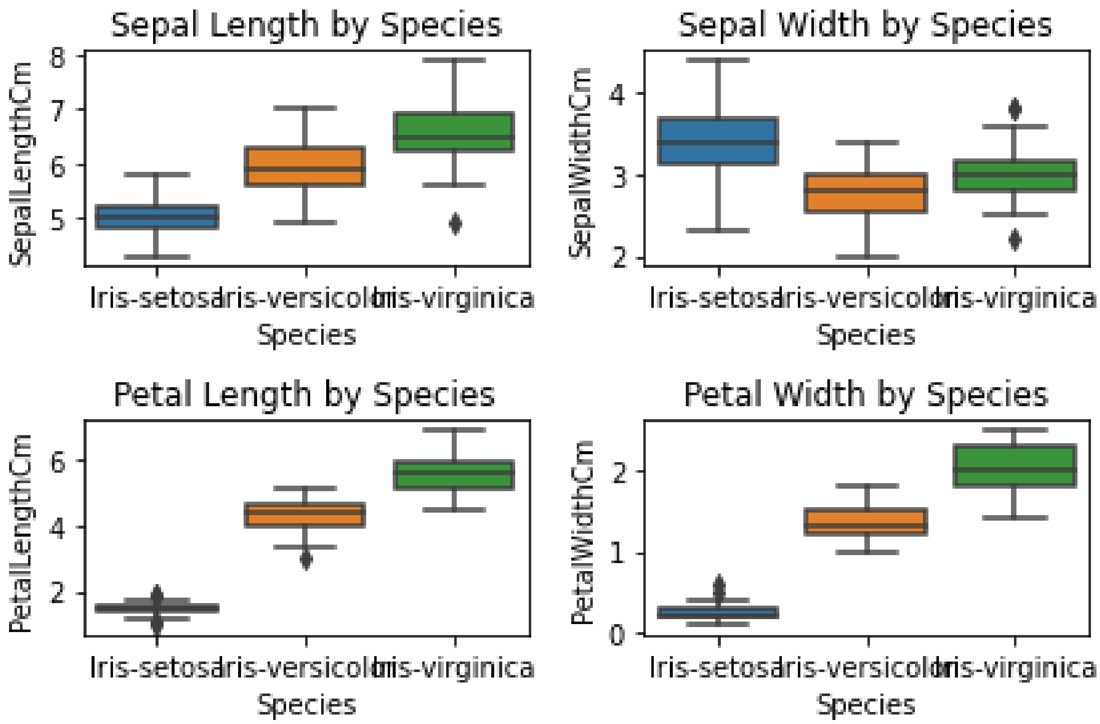
[23]:
```
print(iris_data.head())
```

```
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
0   1            5.1           3.5            1.4           0.2  Iris-setosa
1   2            4.9           3.0            1.4           0.2  Iris-setosa
2   3            4.7           3.2            1.3           0.2  Iris-setosa
3   4            4.6           3.1            1.5           0.2  Iris-setosa
4   5            5.0           3.6            1.4           0.2  Iris-setosa
```

[28]:
```
#count the number for each specie
print('\ncount of each Species:')
print(iris_data['Species'].value_counts())
```

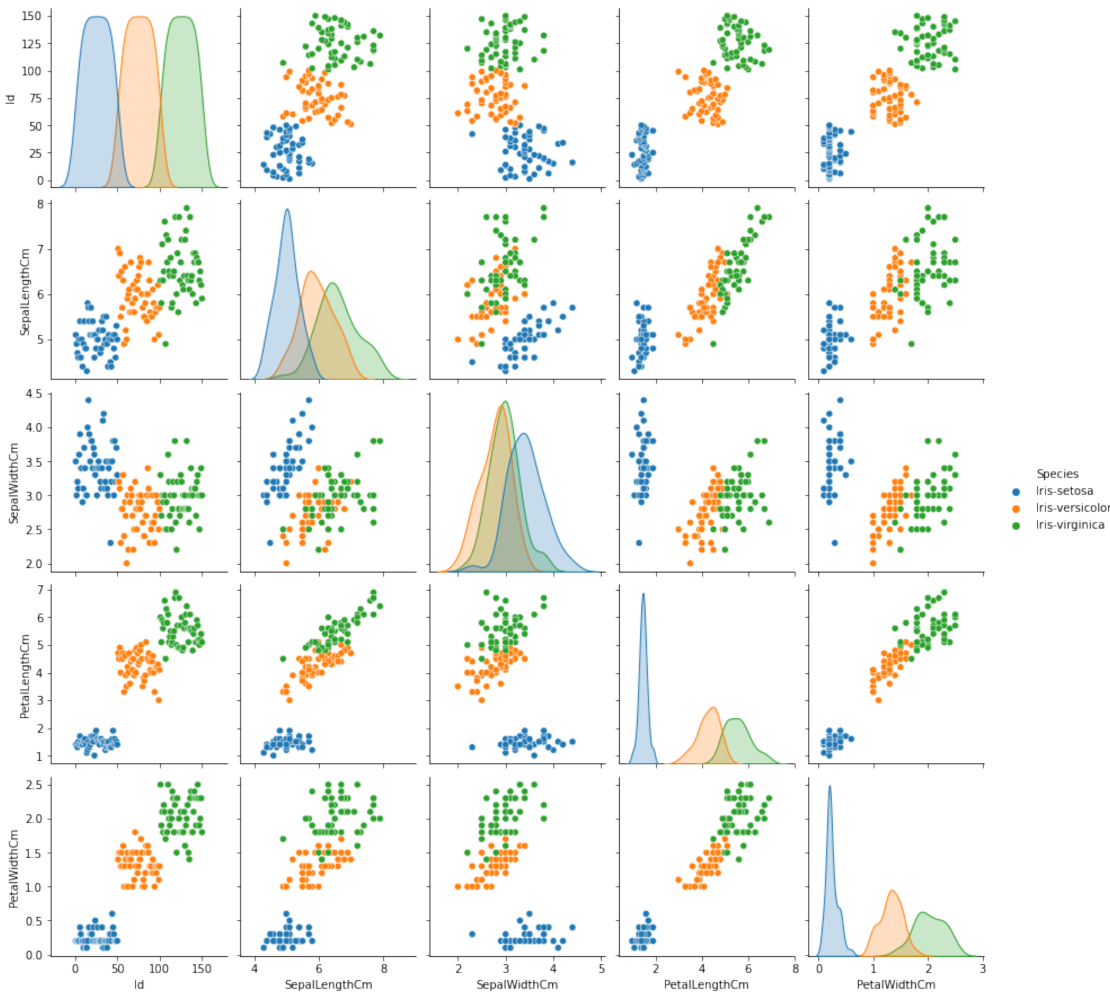
```

count of each Species:
Iris-virginica     50
Iris-versicolor    50
Iris-setosa        50
Name: Species, dtype: int64
```

[33]:
```
plt.subplot(2, 2, 1)
sns.boxplot(data=iris_data, x='Species', y='SepalLengthCm')
plt.title('Sepal Length by Species')

# Sepal Width
plt.subplot(2, 2, 2)
sns.boxplot(data=iris_data, x='Species', y='SepalWidthCm')
plt.title('Sepal Width by Species')

# Petal Length
plt.subplot(2, 2, 3)
sns.boxplot(data=iris_data, x='Species', y='PetalLengthCm')
plt.title('Petal Length by Species')

# Petal Width
plt.subplot(2, 2, 4)
sns.boxplot(data=iris_data, x='Species', y='PetalWidthCm')
plt.title('Petal Width by Species')

plt.tight_layout()
plt.show()
```




[35]:
```
sns.pairplot(iris_data ,hue= 'Species')
```

[35]: <seaborn.axisgrid.PairGrid at 0x1ef62605040>




[36]:
```
iris_data.dtypes
```

[36]:
```
Id                int64
SepalLengthCm    float64
SepalWidthCm     float64
PetalLengthCm    float64
PetalWidthCm     float64
Species           object
dtype: object
```

[38]:
```
iris_data.head()
```

[38]:

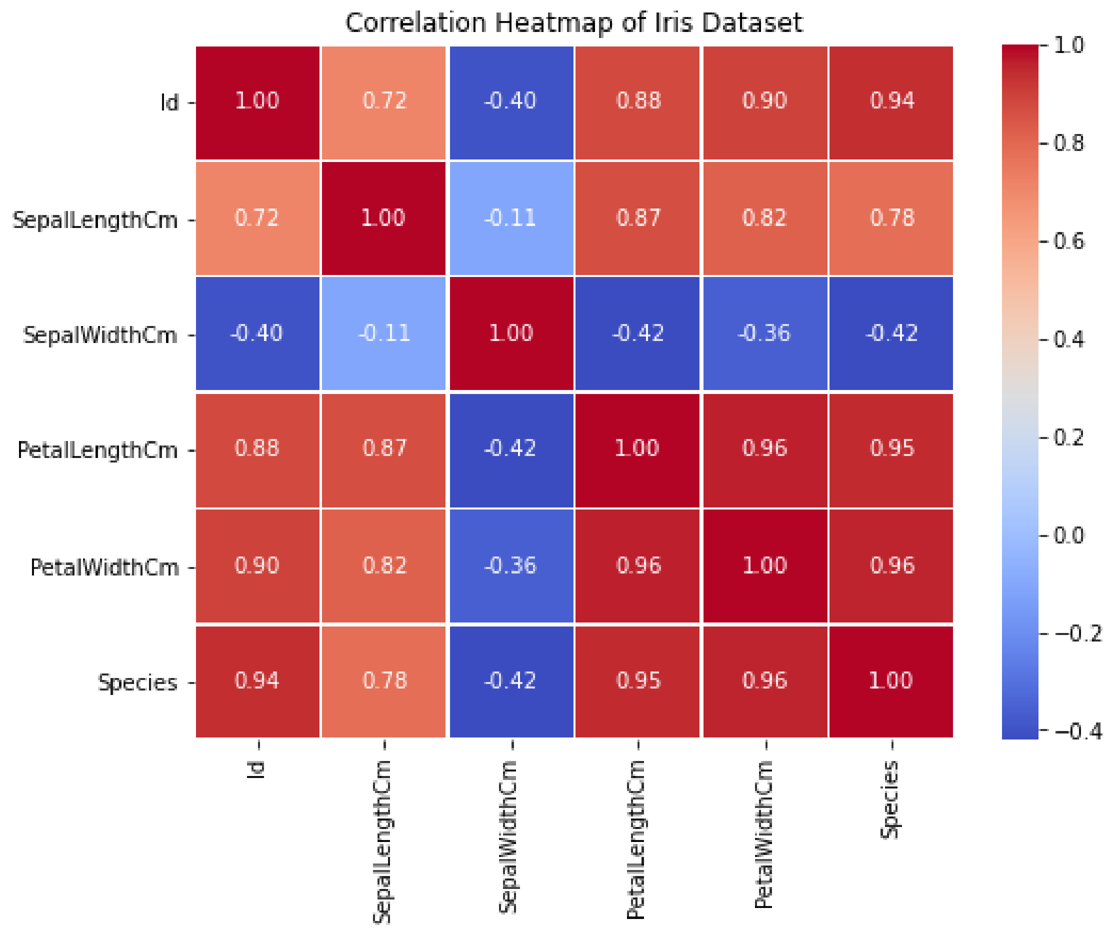
| | Id | SepalLengthCm | SepalWidthCm | PetalLengthCm | PetalWidthCm | Species |
|---|---|---|---|---|---|---|
| 0 | 1 | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| 1 | 2 | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| 2 | 3 | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |

```
3   4              4.6            3.1            1.5            0.2  Iris-setosa
4   5              5.0            3.6            1.4            0.2  Iris-setosa
```

[41]:
```
from sklearn.preprocessing import LabelEncoder
# Load the dataset into a DataFrame
iris_data = pd.read_csv('C:\\Users\\Assala\\Dropbox\\PC\\Downloads\\archive␣
↪(1)\\iris.csv')

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Convert the 'Species' column to numerical values
iris_data['Species'] = label_encoder.fit_transform(iris_data['Species'])

# Display the first few rows of the modified DataFrame
print(iris_data.head())
```

```
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  Species
0   1            5.1           3.5            1.4           0.2        0
1   2            4.9           3.0            1.4           0.2        0
2   3            4.7           3.2            1.3           0.2        0
3   4            4.6           3.1            1.5           0.2        0
4   5            5.0           3.6            1.4           0.2        0
```

[42]:
```
correlation_matrix = iris_data.corr()

# Set up the matplotlib figure
plt.figure(figsize=(8, 6))

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f",␣
↪linewidths=0.5)
plt.title('Correlation Heatmap of Iris Dataset')
plt.show()
```

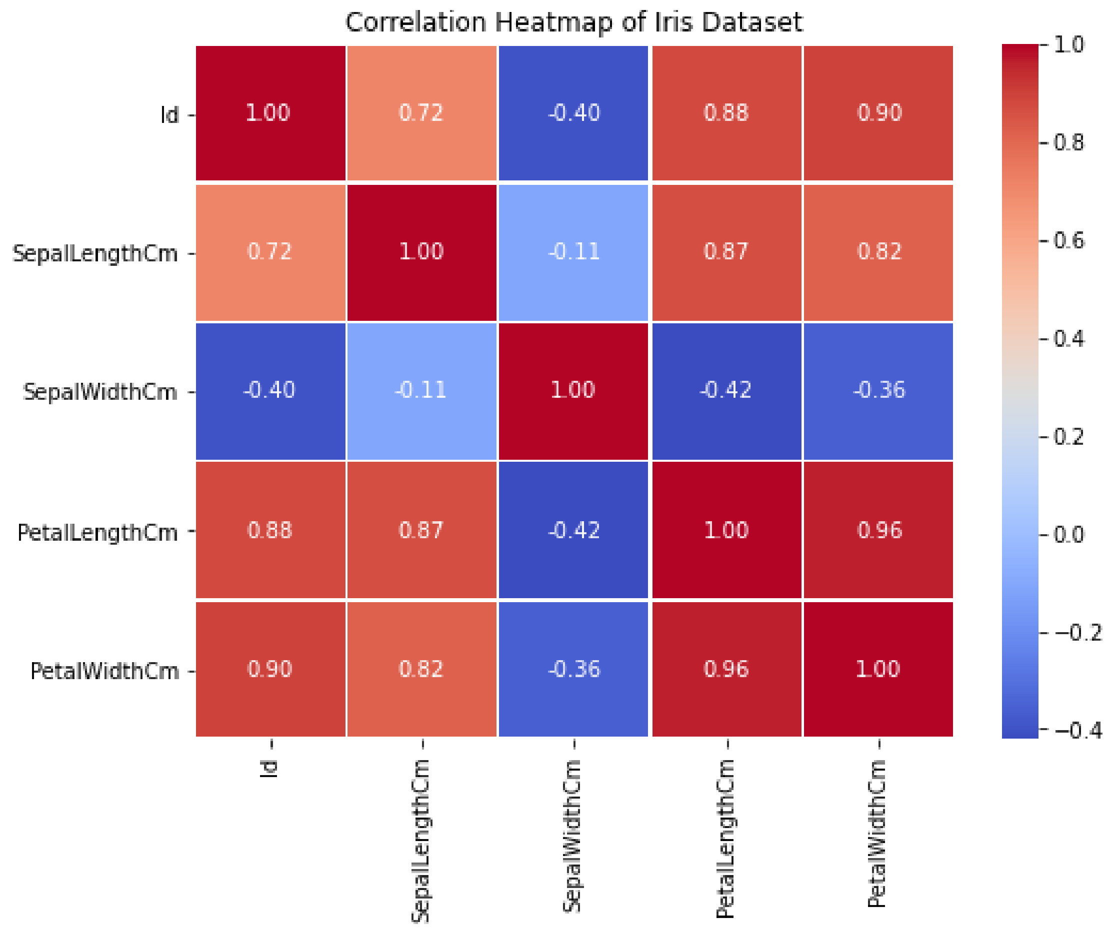



[43]:
```
iris_data.drop(columns=['Species'], inplace=True)

# Display the first few rows of the modified DataFrame
print(iris_data.head())
```

```
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
0   1            5.1           3.5            1.4           0.2
1   2            4.9           3.0            1.4           0.2
2   3            4.7           3.2            1.3           0.2
3   4            4.6           3.1            1.5           0.2
4   5            5.0           3.6            1.4           0.2
```

[44]:
```
correlation_matrix = iris_data.corr()

# Set up the matplotlib figure
plt.figure(figsize=(8, 6))

# Draw the heatmap with the mask and correct aspect ratio
```

```
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f",␣
    ↪linewidths=0.5)
plt.title('Correlation Heatmap of Iris Dataset')
plt.show()
```




[45]:
```
correlation_matrix = iris_data.corr()

# Display the correlation matrix
print("Correlation Matrix:")
print(correlation_matrix)
```

```
Correlation Matrix:
                     Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  \
Id             1.000000       0.716676     -0.397729       0.882747
SepalLengthCm  0.716676       1.000000     -0.109369       0.871754
SepalWidthCm  -0.397729      -0.109369      1.000000      -0.420516
PetalLengthCm  0.882747       0.871754     -0.420516       1.000000
PetalWidthCm   0.899759       0.817954     -0.356544       0.962757
```

```
               PetalWidthCm
Id                 0.899759
SepalLengthCm      0.817954
SepalWidthCm      -0.356544
PetalLengthCm      0.962757
PetalWidthCm       1.000000
```

# 4 MODELING TRAINING AND EVALUATION

# 5 LOGISTIC REGRESSION

[187]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Initialize the Logistic Regression classifier
model = LogisticRegression(max_iter=1000, random_state=42)

# Train the classifier on the training data
model.fit(X_train, y_train)

# Predict the classes for testing data
y_pred = model.predict(X_test)

# Evaluate the model
print("Logistic Regression:")
print("Classification Report:")
print(classification_report(y_test, y_pred))
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Logistic Regression:
Classification Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      1.00      1.00         9
 Iris-virginica       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
```

```
Accuracy: 1.0
```

## 6 SUPPORT MACHINE VECTOR

[166]:
```python
from sklearn.svm import SVC
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=50)
# Initialize the SVM classifier
model = SVC(kernel='rbf', random_state=50)
# Train the classifier on the training data
model.fit(X_train, y_train)
# Predict the classes for testing data
y_pred = model.predict(X_test)
# Evaluate the model
print("Support Vector Machine (SVM):")
print("Classification Report:")
print(classification_report(y_test, y_pred))
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Support Vector Machine (SVM):
Classification Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         9
Iris-versicolor       1.00      1.00      1.00        12
 Iris-virginica       1.00      1.00      1.00         9

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

Accuracy: 1.0
```

## 7 K NEAREST NEIGHBORS

[173]:
```python
from sklearn.neighbors import KNeighborsClassifier
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Initialize the KNN classifier
model = KNeighborsClassifier(n_neighbors=5)

# Train the classifier on the training data
```

```python
model.fit(X_train, y_train)

# Predict the classes for testing data
y_pred = model.predict(X_test)

# Evaluate the model
print("K-Nearest Neighbors (KNN):")
print("Classification Report:")
print(classification_report(y_test, y_pred))
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
K-Nearest Neighbors (KNN):
Classification Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      1.00      1.00         9
 Iris-virginica       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

Accuracy: 1.0
```

[153]:

```python
from sklearn.tree import DecisionTreeClassifier

# Initialize the Decision Tree classifier
model = DecisionTreeClassifier(random_state=42)

# Train the classifier on the training data
model.fit(X_train, y_train)

# Predict the classes for testing data
y_pred = model.predict(X_test)

# Evaluate the model
print("Decision Tree Classifier:")
print("Classification Report:")
print(classification_report(y_test, y_pred))
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Decision Tree Classifier:
Classification Report:
                 precision    recall  f1-score   support
```

```
    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      1.00      1.00         9
 Iris-virginica       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

Accuracy: 1.0
```

[171]:
```
# Initialize classifiers
classifiers = {
    'Logistic Regression': LogisticRegression(max_iter=1000, random_state=42),
    'K-Nearest Neighbors': KNeighborsClassifier(),
    'Support Vector Machine': SVC(random_state=50)
}

# Train and evaluate each classifier
results = []
for name, clf in classifiers.items():
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    results.append({'Model': name, 'Accuracy': accuracy})

# Assemble results into a DataFrame
results_df = pd.DataFrame(results)

# Display the results
print(results_df)
```

```
                    Model  Accuracy
0     Logistic Regression       1.0
1     K-Nearest Neighbors       1.0
2  Support Vector Machine       1.0
```

# 8 Conclusion

Based on the provided accuracy scores, it seems that all three models Logistic Regression, K-Nearest Neighbors, and Support Vector Machine—achieve perfect accuracy on the test data. While achieving such high accuracy might be desirable, especially in classification tasks, it's essential to interpret these results cautiously and consider potential reasons for such performance due to Data Equality,Model Tunning and Overfitting.This project was a very important and basic for every data scientist student.

[ ]:
```
```