Oasis Infobyte task 1

Iris flowers classification project of ML

Information about the dataset used here:

The Iris flower dataset is a popular machine learning dataset that is often used as a first introduction to classification tasks. The dataset contains four features: sepal length, sepal width, petal length, and petal width. Each feature is measured in centimeters. The target variable is the class of iris flower, which can be one of three values: Iris Setosa, Iris Versicolor, or Iris Virginica

 # Dataset Characteristics

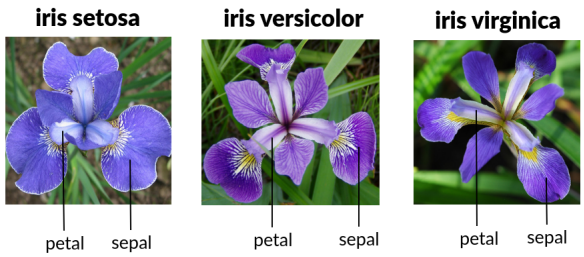
The dataset contains 150 instances, with 50 instances for each class. The dataset is linearly separable, meaning that there exists a line that can perfectly separate the three classes. The dataset is small and can be easily fit in memory without special scaling capabilities.

Dataset Usage

The Iris flower dataset is a good dataset for beginners to machine learning because it is relatively simple and easy to understand. The dataset can be used to train a variety of classification algorithms, such as logistic regression, support vector machines, and decision trees.

Display of Images to Understand Dataset better

```
from IPython.display import Image
Image(url='https://editor.analyticsvidhya.com/uploads/51518iris%20img1.png', width=500)
```



Imporating Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
## Supress warnings
import warnings
warnings.filterwarnings("ignore")
```

Import Dataset

```
Iris_df = pd.read_csv('/Iris.csv')
print("the data has been successfully load.")
```

```
the data has been successfully load.
```

```
Iris_df
```

	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
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

The information about the data frame, which includes column names, the datatype of each column, and statistics like the number of non-null variables.

Exploration Of Data:

```
# information of data
Iris_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
```

```
# Shape of data
Iris_df.shape
```

```
(150, 6)
```

```
# The statisitical summary of Data
Iris_df.describe()
```

	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

```
#Check null values
Iris_df.isnull().sum()
```

```
Id              0
SepalLengthCm   0
SepalWidthCm    0
PetalLengthCm   0
PetalWidthCm    0
Species         0
dtype: int64
```

As can be seen, all values are zero. It indicates that the entire data frame has no null values.

```
#Check the correlation matrix
corr_matrix = Iris_df.corr()
print(corr_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
```

The correlation matrix shows the correlation between each pair of variables in the DataFrame. The correlation coefficient can range from -1 to 1, with a value of -1 indicating a perfect negative correlation and a value of 1 indicating a perfect positive correlation. A value of 0 indicates no correlation.

```
# Get the category-wise frequency of the data in the `Species` column
species_counts = Iris_df['Species'].value_counts()

# Print the species counts
print(species_counts)
```

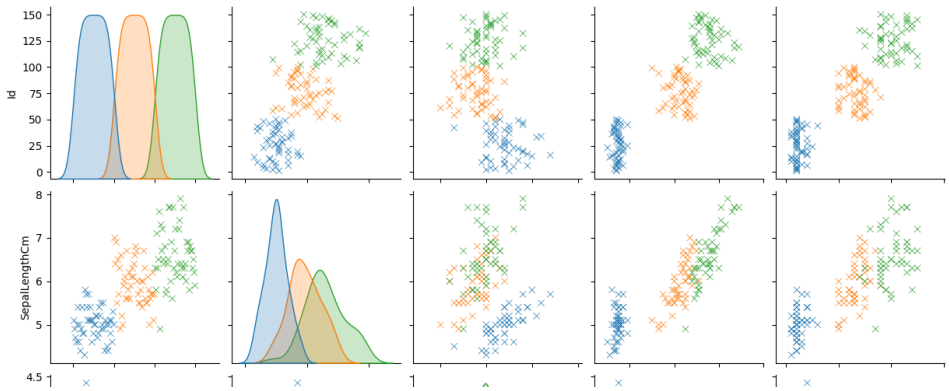
```
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: Species, dtype: int64
```

Category-wise frequency can be used in data analysis to identify patterns and trends in the data

Data Analysis

**Data Visualization **

```
# Create a pairplot with the 'Species' column as the hue variable and the 'x' marker
sns.pairplot(Iris_df, hue="Species", markers="x")

# Show the plot
plt.show()
```

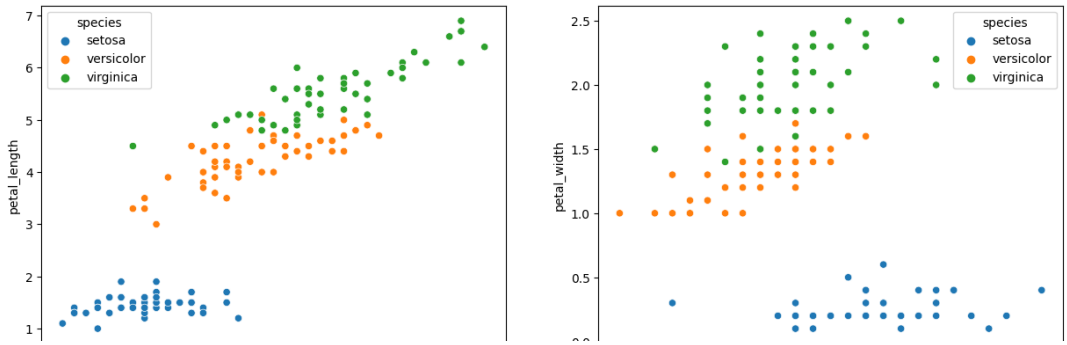


The pairplot demonstrates that SepalLength and SepalWidth, as well as PetalLength and PetalWidth, have a high positive association. Additionally, SepalLength and PetalLength as well as SepalWidth and PetalWidth have a moderately negative connection.

The pairplot also demonstrates that the SepalLength and PetalLength measures clearly distinguish the three species of iris. This implies that the species of an iris may be precisely determined using these measures.

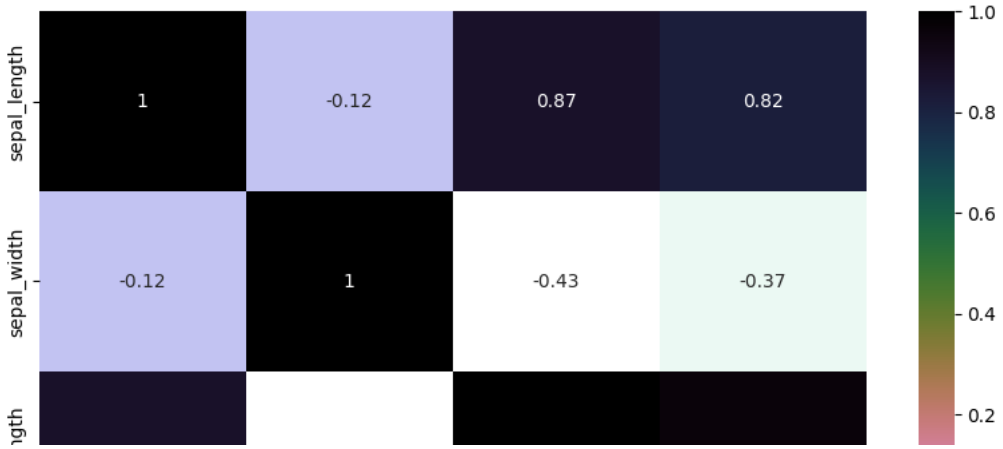
In exploratory data analysis, pairplots are a helpful tool for seeing patterns and connections in the data that might not be immediately visible.

```
# Create dataframe
Iris_df = sns.load_dataset('iris')

# Create figure and axes
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))

# First subplot
sns.scatterplot(data=Iris_df, x='sepal_length', y='petal_length', hue='species', ax=ax1)

# Second subplot
sns.scatterplot(data=Iris_df, x='sepal_width', y='petal_width', hue='species', ax=ax2)

# Show plot
plt.show()
```



The left subplot shows the relationship between SepalLength and PetalLength, and the right subplot shows the relationship between SepalWidth and PetalWidth. The color of the points in each subplot represents the species of iris.

```
# Calculate the correlation matrix
corr_matrix = Iris_df.corr()

# Create a heatmap of the correlation matrix
plt.figure(figsize=(10, 7))
sns.heatmap(corr_matrix, annot=True, cmap="cubehelix_r")

# Show the plot
plt.show()
```

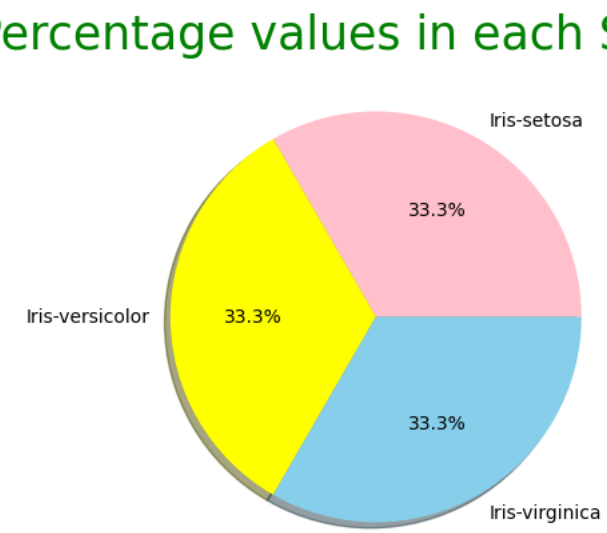


The heatmap demonstrates that SepalLength and SepalWidth, as well as PetalLength and PetalWidth, have a high positive association. Additionally, SepalLength and PetalLength as well as SepalWidth and PetalWidth have a moderately negative connection.

These correlations show that the variables are somehow linked to one another. SepalLength and SepalWidth, for instance, have a substantial positive connection, indicating that these two parameters tend to change together. In other words, SepalWidth is likely to rise if SepalLength does.

In order to find patterns and correlations in the data that might not be immediately visible, heatmaps are a valuable tool for data analysis.

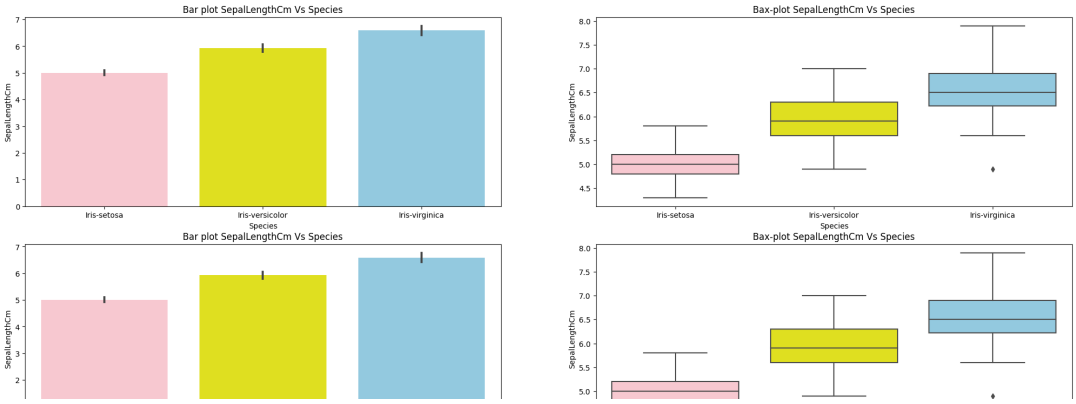
```
# Calculate the value counts of the 'Species' column
species_counts = Iris_df['Species'].value_counts()

# Get a list of unique species
species_list = species_counts.index.to_list()

# Create a list of colors for each species
color_list = ['pink', 'yellow', 'skyblue']

# Plot the value counts as a pie chart, using the list of colors
species_counts.plot(kind="pie", autopct="%1.1f%%", shadow=True, figsize=(5, 5), colors=color_list)

# Set the title and labels of the pie chart
plt.title("Percentage values in each Species", fontsize=25, c="g")
plt.ylabel("", fontsize=20, c="r")

# Show the pie chart
plt.show()
```



According to the pie Chart, nearly 33.3% of the samples in the Iris dataset are from the setosa species, . Versicolor and virginica species make up the remaining 33.3% of the sample We can see ,all Species has equal values in dataset. Iris-Setosa:50 Iris- Versicolor:50 Iris- Virginica : 50. italicized text

Pie charts are a helpful tool for data visualization because they make it simple to see how values are distributed across a dataset.

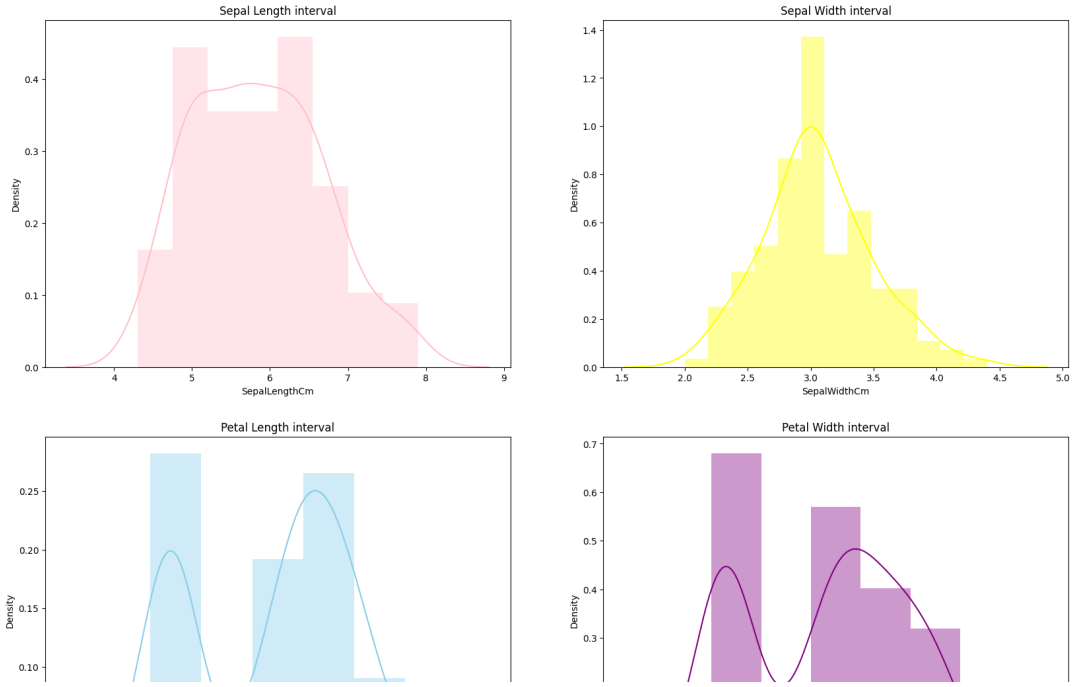
```
 #Create a figure with four subplots
plt.figure(figsize=(25, 10))

# Create the first subplot
plt.subplot(2, 2, 1)
sns.barplot(
    x="Species",
    y="SepalLengthCm",
    data=Iris_df,
    palette=["pink", "yellow", "skyblue"],
)
plt.title("Bar plot SepalLengthCm Vs Species")

# Create the second subplot
plt.subplot(2, 2, 2)
sns.boxplot(
    x="Species",
    y="SepalLengthCm",
    data=Iris_df,
    palette=["pink", "yellow", "skyblue"],
)
plt.title("Bax-plot SepalLengthCm Vs Species")

# Create the third subplot
plt.subplot(2, 2, 3)
sns.barplot(
    x="Species",
    y="SepalLengthCm",
    data=Iris_df,
    palette=["pink", "yellow", "skyblue"],
)
plt.title("Bar plot SepalLengthCm Vs Species")

# Create the fourth subplot
plt.subplot(2, 2, 4)
sns.boxplot(
    x="Species",
    y="SepalLengthCm",
    data=Iris_df,
    palette=["pink", "yellow", "skyblue"],
)
plt.title("Bax-plot SepalLengthCm Vs Species")

# Show the plots
plt.show()
```



This chart shows the distribution of sepal length by species in the Iris dataset, using both bar plots and box plots. The bar plots show the mean sepal length for each species, while the box plots show the distribution of sepal length within each species, including the median, quartiles, and outliers.

```
# Create a figure with four subplots
plt.figure(figsize=(20, 15))

# Create the first subplot
plt.subplot(2, 2, 1)
```

```
sns.distplot(Iris_df["SepalLengthCm"], color="pink").set_title("Sepal Length interval")

# Create the second subplot
plt.subplot(2, 2, 2)
sns.distplot(Iris_df["SepalWidthCm"], color="yellow").set_title("Sepal Width interval")

# Create the third subplot
plt.subplot(2, 2, 3)
sns.distplot(Iris_df["PetalLengthCm"], color="skyblue").set_title("Petal Length interval")

# Create the fourth subplot
plt.subplot(2, 2, 4)
sns.distplot(Iris_df["PetalWidthCm"], color="purple").set_title("Petal Width interval")

# Show the plots
plt.show()
```



Sepal Length interval

This graph shows the distribution of sepal length in the Iris dataset. The x-axis shows the sepal length in centimeters, and the y-axis shows the density of sepal lengths at each value. The graph shows that the majority of sepal lengths fall between 4 and 7 centimeters, with a peak at around 5 centimeters. There are also some outliers with sepal lengths outside of this range.

Sepal Width interval

This graph shows the distribution of sepal width in the Iris dataset. The x-axis shows the sepal width in centimeters, and the y-axis shows the density of sepal widths at each value. The graph shows that the majority of sepal widths fall between 2 and 4 centimeters, with a peak at around 3 centimeters. There are also some outliers with sepal widths outside of this range.

Petal Length interval

This graph shows the distribution of petal length in the Iris dataset. The x-axis shows the petal length in centimeters, and the y-axis shows the density of petal lengths at each value. The graph shows that the majority of petal lengths fall between 1 and 7 centimeters, with peaks at around 2 and 5 centimeters. There are also some outliers with petal lengths outside of this range.

Petal Width interval

This graph shows the distribution of petal width in the Iris dataset. The x-axis shows the petal width in centimeters, and the y-axis shows the density of petal widths at each value. The graph shows that the majority of petal widths fall between 0.5 and 3 centimeters, with a peak at around 1 centimeter. There are also some outliers with petal widths outside of this range.

Overall, the graphs show that the Iris dataset contains a wide range of sepal and petal lengths and widths. The distributions of these measurements are also different for each species of iris.

Data Cleaning

```
# Change Categorical Data into numerical value
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
Iris_df["Species"] = le.fit_transform(Iris_df["Species"])
Iris_df.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

```
Iris_df['Species'].unique()
```

```
array([0, 1, 2])
```

```
X = Iris_df.iloc[:,[0,1,2,3]]
X.head()
```

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

```
y = Iris_df.iloc[:, - 1]
y.head()
```

```
0    0
1    0
2    0
3    0
4    0
Name: Species, dtype: int64
```

```
print(X.shape)
print(y.shape)
```

```
(150, 4)
(150,)
```

Model Building

Supervised Machine Learning

Split data into Training and Testing Set

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=0)
```

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(120, 4)
(30, 4)
(120,)
(30,)
```

Logistic Regression

logistic regression is a computationally efficient algorithm. This means that it can train and evaluate quickly, even on large datasets.

```
from sklearn.linear_model import LogisticRegression
lr= LogisticRegression()
lr.fit(X_train, y_train)
print("Logistic regression successfully implemented")
y_pred = lr.predict(X_test)
#Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:-")
print(cm)
accuracy = accuracy_score(y_test, y_pred)
print("accuracy is:-",accuracy*100)
print("Classification Report:-")
print (classification_report(y_test,y_pred))
```

```
Logistic regression successfully implemented
Confusion Matrix:-
[[11  0  0]
 [ 0 13  0]
 [ 0  0  6]]
accuracy is:- 100.0
Classification Report:-
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        11
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00         6

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

Decision Tree

They are simple, interpretable, robust to overfitting, and efficient to train and evaluate.

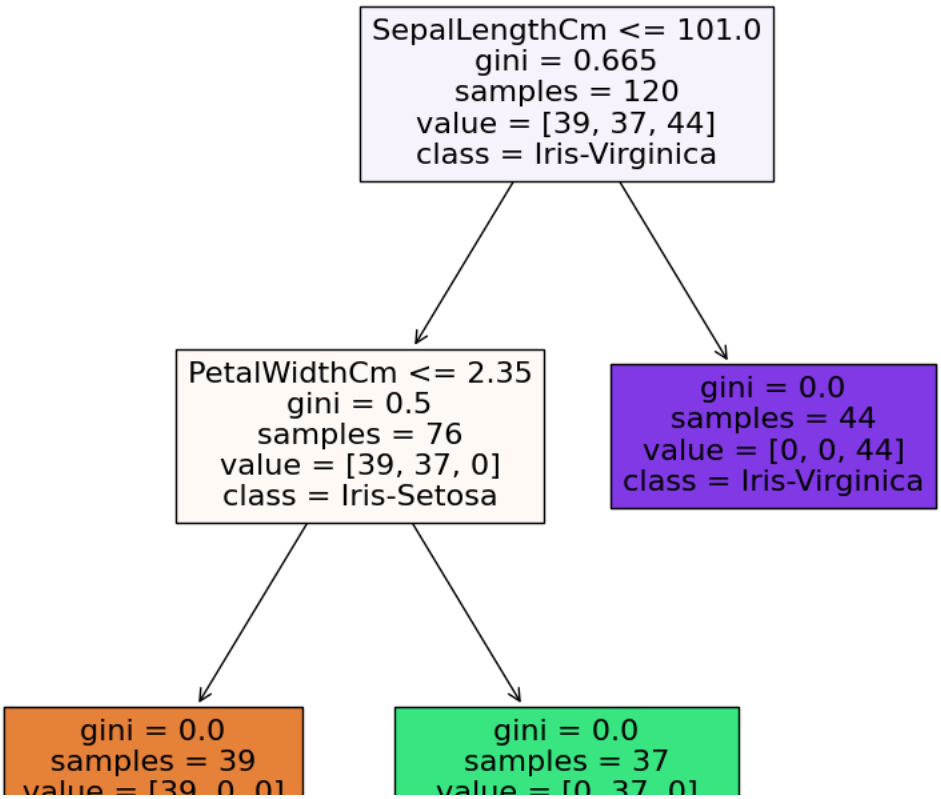
In addition to the above advantages, decision trees are also a popular and well-studied machine learning algorithm. This means that there are many resources available to help you learn how to use decision trees and troubleshoot any problems you may encounter

```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier()
dtree.fit(X_train, y_train)
print("Decision Tree Algorithm is successfully implimented.")
y_pred = dtree.predict(X_test)
#confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:- ")
print(cm)
#accuracy test
accuracy = accuracy_score(y_test,y_pred)
print("accuracy:- ", accuracy*100)
print("Classification Report:-")
print(classification_report(y_test, y_pred))
```

```
Decision Tree Algorithm is successfully implimented.
Confusion Matrix:-
[[11  0  0]
 [ 0 13  0]
 [ 0  1  5]]
accuracy:-  96.66666666666667
Classification Report:-
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        11
           1       0.93      1.00      0.96        13
           2       1.00      0.83      0.91         6

    accuracy                           0.97        30
   macro avg       0.98      0.94      0.96        30
```

```
weighted avg       0.97      0.97      0.97        30
```

```
from sklearn.tree import plot_tree


# for visualziing the Decision Tree
feature = ['SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm']
colors=['coolcolors']
classes = ['Iris-Setosa','Iris-Versicolor','Iris-Virginica']
plt.figure(figsize=(10,10))
plot_tree(dtree, feature_names = feature, class_names = classes, filled = True,);
```



K - NN CLASSIFIER

KNN works by finding the k most similar data points to a new data point and then predicting the class of the new data point based on the classes of the k most similar data points. KNN is a simple and effective algorithm

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors= 7)
knn.fit(X_train, y_train)
print("K-Nearest Neighbors classifier is successfully implemented")
y_pred = knn.predict(X_test)
#confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:- ")
print(cm)
#accuracy test
accuracy = accuracy_score(y_test,y_pred)
print("accuracy:- ", accuracy*100)
print("Classification Report:-")
print( classification_report(y_test, y_pred))
```

```
K-Nearest Neighbors classifier is successfully implemented
Confusion Matrix:-
[[11  0  0]
 [ 0 13  0]
 [ 0  0  6]]
accuracy:-  100.0
```

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

           0       1.00      1.00      1.00        11
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00         6

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

Double-click (or enter) to edit

Results

1. Accuracy of Logistic Regression :- 100%
2. Accuracy of Decision Tree :- 96.66%
3. Accuracy of K-NN Classifier :- 100%

Test Model

Having a test model is an important part of any machine learning project. It helps us to evaluate the performance of the model on unseen data, identify areas where the model can be improved, and avoid overfitting

```
input_data=(4.9,3.0,1.4,0.2)
#changing the input data to a numpy array
input_data_as_nparray = np.asarray(input_data)
#reshape the data as we are predicting the label for only the instance
input_data_reshaped = input_data_as_nparray.reshape(1,-1)
prediction = dtree.predict(input_data_reshaped)
print("The category is",prediction)
```

```
The category is [0]
```

Thank you

Oasis Infobyte Task-1 by Dhvani Naik