Sri Lanka Institute Of Information Technology

Analysis-of-Iris-flower-dataset Using Decision Tree Algorithm

Project Report

SE4060 - Machine Learning

IT16001480

H.K.D.C.Jayalath

Table of Contents

List of Figures

1 Introduction

The Iris flower data set or Fisher's Iris data set is a multivariate data set introduced by Sir Ronald Fisher in the 1936 as an example of discriminant analysis. [1]

The data set consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor), so 150 total samples. And then four numeric properties about those classes: Sepal Length, Sepal Width, Petal Length, and Petal Width.

One species, Iris Setosa, is "linearly separable" from the other two.

I have performed various classification techniques to classify into three species using Dimensionality Reduction techniques such as PCA and LDA.

Figure 1: Iris versicolor

Figure 2: Iris setosa

Figure 3: Iris virginica

2 Proposed Solution

In this assignment I have used a dataset which is contains some of the features and their diagnosis to identify species of Iris flowers. For that create a machine learning classifier model using random forest classification algorithm to predict a given flower is fallen under which category.

3 Dataset

3.1 Description of a Dataset

Before starting to predict whether the given flower is fallen under which category, We need to find out a dataset which is related to this context. For prediction purposes I have used a dataset in a Kaggle learning repository. Following link contains Iris dataset.

Link: https://www.kaggle.com/arshid/iris-flower-dataset

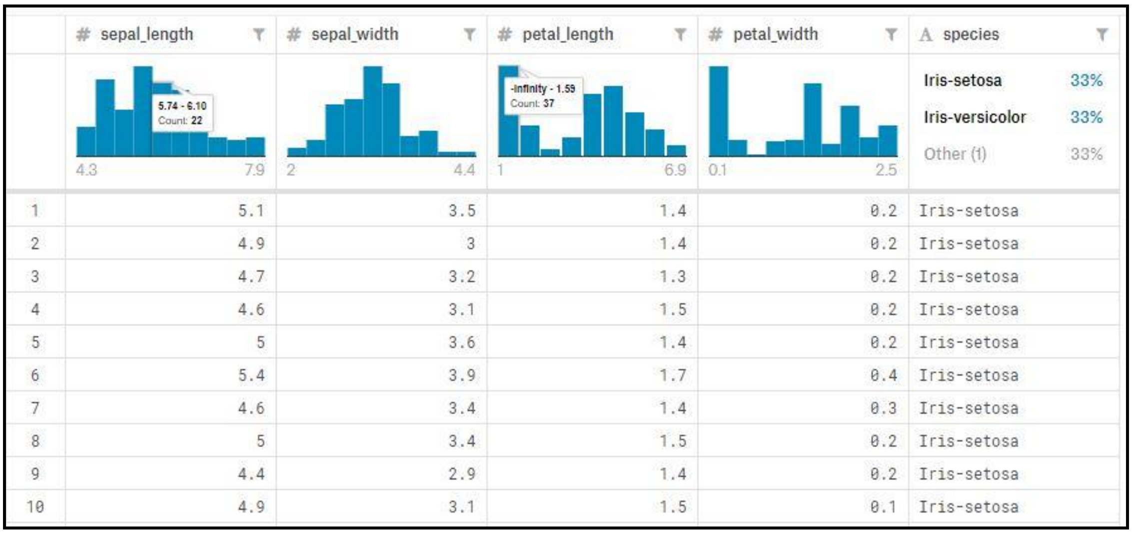
The iris dataset contains measurements for 150 iris flowers from three different species. The three classes in the Iris dataset:

Iris-setosa (n=50), Iris-versicolor (n=50), Iris-virginica (n=50)
The four features of the Iris dataset:

- Sepal length in cm
- Sepal width in cm
- Petal length in cm
- Petal width in cm

3.2 Description of Features

Iris dataset contains categorical attributes. Attributes contain some important features of an Iris flowers that are more useful to categorize the given Iris. And also there is a class attribute which contains information about a given Iris Flowers



	# sepal_length	# sepal_width	# petal_length	# petal_width	A species
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5	3.6	1.4	0.2	Iris-setosa
6	5.4	3.9	1.7	0.4	Iris-setosa
7	4.6	3.4	1.4	0.3	Iris-setosa
8	5	3.4	1.5	0.2	Iris-setosa
9	4.4	2.9	1.4	0.2	Iris-setosa
10	4.9	3.1	1.5	0.1	Iris-setosa

Figure 4: Dataset Features

4 Methodology

4.1 Algorithm

4.1.1Decision Tree Classifier Algorithm

Decision Tree is a type of supervised learning algorithm that is mostly used for classification problems. Surprisingly, it works for both categorical and continuous dependent variables. Decision tree is also very flexible, easy to understand and also easy to debug. The basic procedure of Decision trees is, it split data into different branches based on a certain condition and so on until there is no more splitting to done. The most of the cases it works very well with both classification problems and regression problems.

Basically decision trees are trees which produce outputs based on certain decisions. It used for classification and prediction. There are two types of decision trees based on splitting attributes. It can be univariate as well as multivariate. Mainly decision trees used a top-down approach. Decision trees are trees which produce outputs based on certain decisions. It used for classification and prediction. There are two types of decision trees based on splitting attributes. These are.

- univariate
- multivariate

Mainly decision trees used a top-down approach. To find out the best attribute for split decision trees use multiple methods available.

- Information Gain
- Gain Ratio
- Gini Index

In decision trees, basically information gain is calculated using the entropy. The attribute with the highest information gain is selected as a splitting attribute. In this scenario "petal width (cm)" consider as a splitting attribute. The main drawback of decision tree is overfitting and highly depend on the tanning data.

Decision tree use pruning techniques to improve the accuracy of an algorithm. And after pruning tree become less complex. It helps to avoid the problem like overfitting. Also decision tree have some advantages. They are High accuracy and can handle high dimensional data

However, considering all the facts, I decided to use Decision Tree Algorithm for Iris classification problem.

4.2 Implementation

4.2.1 Data Preprocessing

As a first stage of data preprocessing we need to import our dataset to jupyter notebook. Mainly that dataset is read from the Excel (csv) file. Following figure shows the first five instances contains in the Iris dataset.

```
In [2]: # Importing the dataset
        dataset = pd.read_csv('Iris.csv')

In [17]: dataset.head()

Out[17]:
```

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

Figure 5 – Code to Load Dataset & get first five instances

4.2.2 Split labels and features

Next step is to identify features and labels. Basically, label is a value that we want to Predict. In here 'Prediction value' is consider as a label. All other columns are considered
as features. That means features are attributes which helps to predict outcome.

So that before start applying any learning algorithm we need to remove label from Features.

```
In [9]:  # Remove Id Column
         dataset.drop("Id", axis=1, inplace = True)

In [10]: dataset["Species"].value_counts()

Out[10]: Iris-versicolor    50
         Iris-setosa        50
         Iris-virginica     50
         Name: Species, dtype: int64
```

```
In [66]: iris.feature_names

Out[66]: ['sepal length (cm)',
          'sepal width (cm)',
          'petal length (cm)',
          'petal width (cm)']
```

Figure 6 – Code to Drop the id Value

Figure 7-Features List and labels

4.2.3 Split training set and test set

```
In [67]:
         iris.target_names

Out[67]: array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

Next, we need to divide data into two categories. Training set and Test set. Basically, Training set is used to build a machine learning model and Test set is used to test Performance in the machine learning model. In here I use scikit-learn library to divide Training set and Test set

```
In [28]: # Splitting the dataset into the Training set and Test set (Petal Lenth vs Petal Width)
         from sklearn.model_selection import train_test_split
         X_trainp, X_testp, y_trainp, y_testp = train_test_split(X_p, y, test_size = 0.3, random_state = 0)

In [29]: # Splitting the dataset into the Training set and Test set (Sepal Length vs Sepal Width)
         from sklearn.model_selection import train_test_split
         X_trains, X_tests, y_trains, y_tests = train_test_split(X_s, y, test_size = 0.3, random_state = 0)
```

Figure 8-Split data in to tanning set and test data

Mainly specify 0.3 as a test size. So that 30% of the data is consider as a test set. And 70% of the data is consider as a training set.

4.2.4 Dimension Reduction

- To speed up the machine learning algorithm

High dimension of the dataset will slow down the performance of the algorithm. Reducing the dimension would help to solve the problem.

Our dataset has four numeric features. I have performed Decision tree classification techniques to classify into three species using Dimensionality Reduction techniques such as PCA and LDA. I also classified them by selecting two features to give better results.

Out[22]:

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

```
In [23]: X_p = dataset.iloc[:,[2, 3]].values
         y = dataset.iloc[:, 4].values
         X_s = dataset.iloc[:,[0, 1]].values

In [24]: X_p.shape
Out[24]: (150, 2)

In [25]: X_s.shape
Out[25]: (150, 2)

In [26]: y.shape
Out[26]: (150,)
```

Figure 9-Dimention reduction

4.2.5 Visualizing the data

- Decision Tree

```
In [11]: #load data
         iris=datasets.load_iris()
         x=iris.data
         y=iris.target

In [12]: iris.feature_names
         iris.target_names

Out[12]: array(['setosa', 'versicolor', 'virginica'], dtype='<U10')

In [13]: #create decision tree classifier
         clf=DecisionTreeClassifier(random_state=0)

         #train the modle
         model=clf.fit(x,y)

In [14]: dot_data = tree.export_graphviz(model,out_file=None,
                                         feature_names=iris.feature_names,
                                         class_names=iris.target_names)
         #draw graph
         graph=pydotplus.graph_from_dot_data(dot_data)

         #show graph
         Image(graph.create_png())
```

Figure 10 – Code to Create the Decision tree

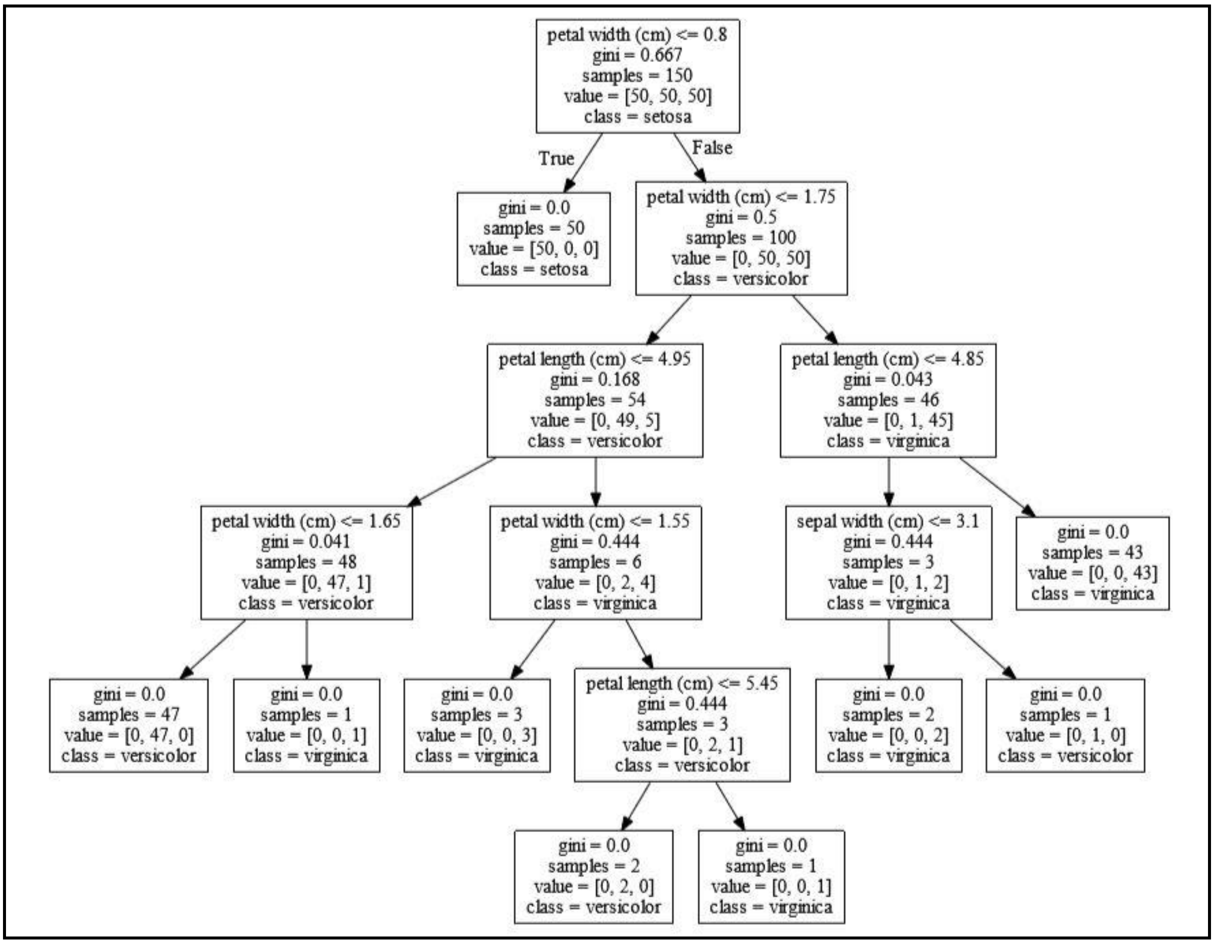


Figure 11 – Graph of Decision tree

- Scatter Plot

```
In [16]: # By selecting two features SepalLengthCm and SepelWidthCm
         sns.FacetGrid(dataset, hue="Species", size=6) \
            .map(plt.scatter, "SepalLengthCm", "SepalWidthCm") \
            .add_legend()
         plt.title('Sepal Length Vs Sepel Width')
         plt.show()

         C:\Users\dilun\Anaconda3\lib\site-packages\seaborn\axisgrid.py:230: UserWarning: The `size` paramter has been renamed to `heigh
         t`; please update your code.
           warnings.warn(msg, UserWarning)
```

Figure 12 – Code to Draw Scatter Plot

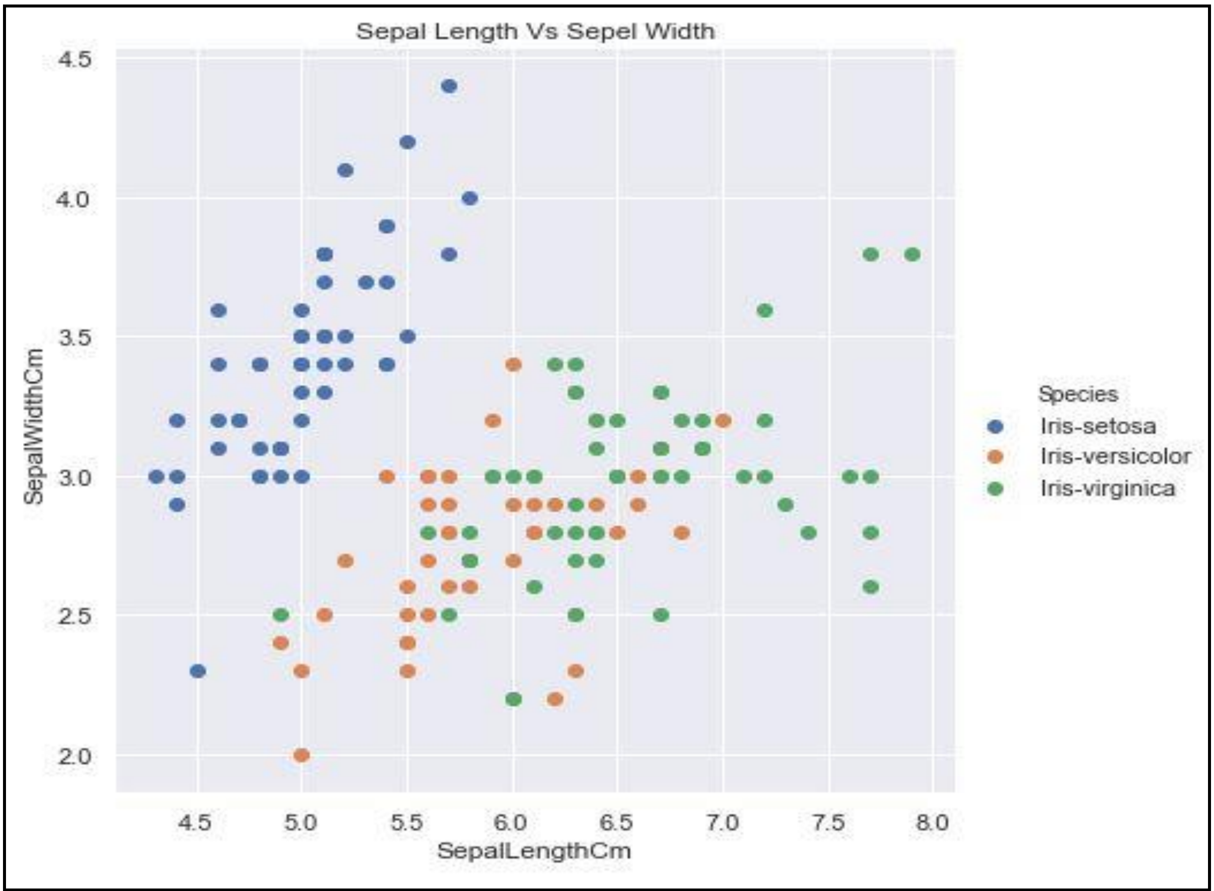


Figure 13 –Scatter Plot

- We can identify Iris-setosa flowers from others Using SepalLengthCm and SepalWidthCm features
- Seperating Iris-versicolor from Iris-viginica is little bit harder than other one (Iris-setosa).because these two have considerable overlap.

- Pair Plot

```
In [18]: sns.pairplot(dataset, hue="Species", size=3);
         plt.show()

C:\Users\dilun\Anaconda3\lib\site-packages\seaborn\axisgrid.py:2065: UserWarning: The `size` parameter has been renamed to `hei
ght`; pleaes update your code.
  warnings.warn(msg, UserWarning)
```

Figure 14 –Coding to Draw Pair Plot

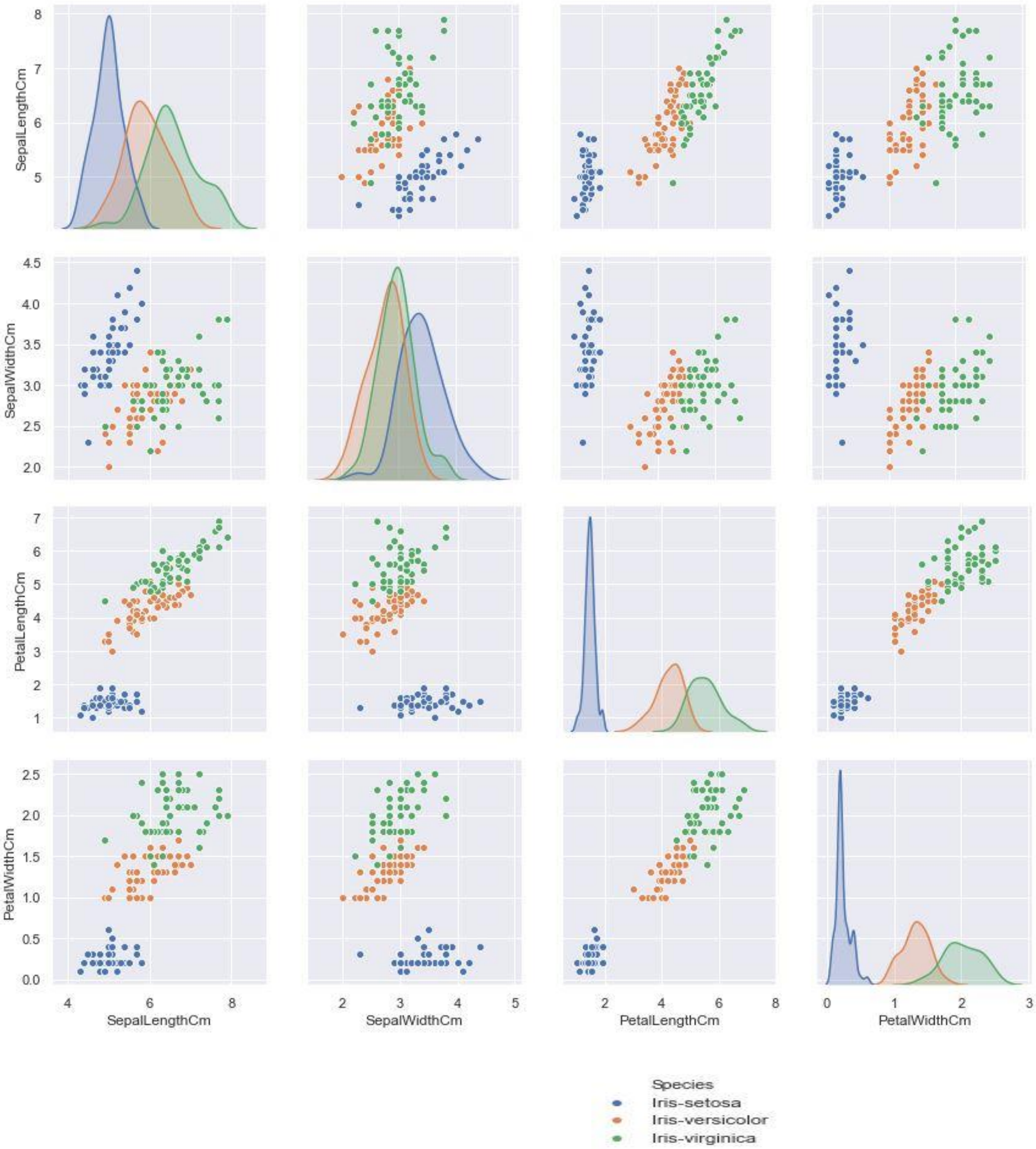


Figure 15 –Pair Plot

- The best features to identify various flower types are PetalLengthCm and PetalWidthCm.
- Iris-setosa can be easily identified, Iris-virginica and Iris-versicolor have some overlap.

- Box Plot

```
In [59]: plt.figure(figsize=(14,12))
         plt.subplot(2,2,1)
         sns.boxplot(x='Species', y = 'SepalLengthCm', data=dataset)
         plt.subplot(2,2,2)
         sns.boxplot(x='Species', y = 'SepalWidthCm', data=dataset)

         plt.subplot(2,2,3)
         sns.boxplot(x='Species', y = 'PetalLengthCm', data=dataset)
         plt.subplot(2,2,4)
         sns.boxplot(x='Species', y = 'PetalWidthCm', data=dataset)

Out[59]: <matplotlib.axes._subplots.AxesSubplot at 0x18bb4260a90>
```

Figure 16 –Coding for Draw Box Plot

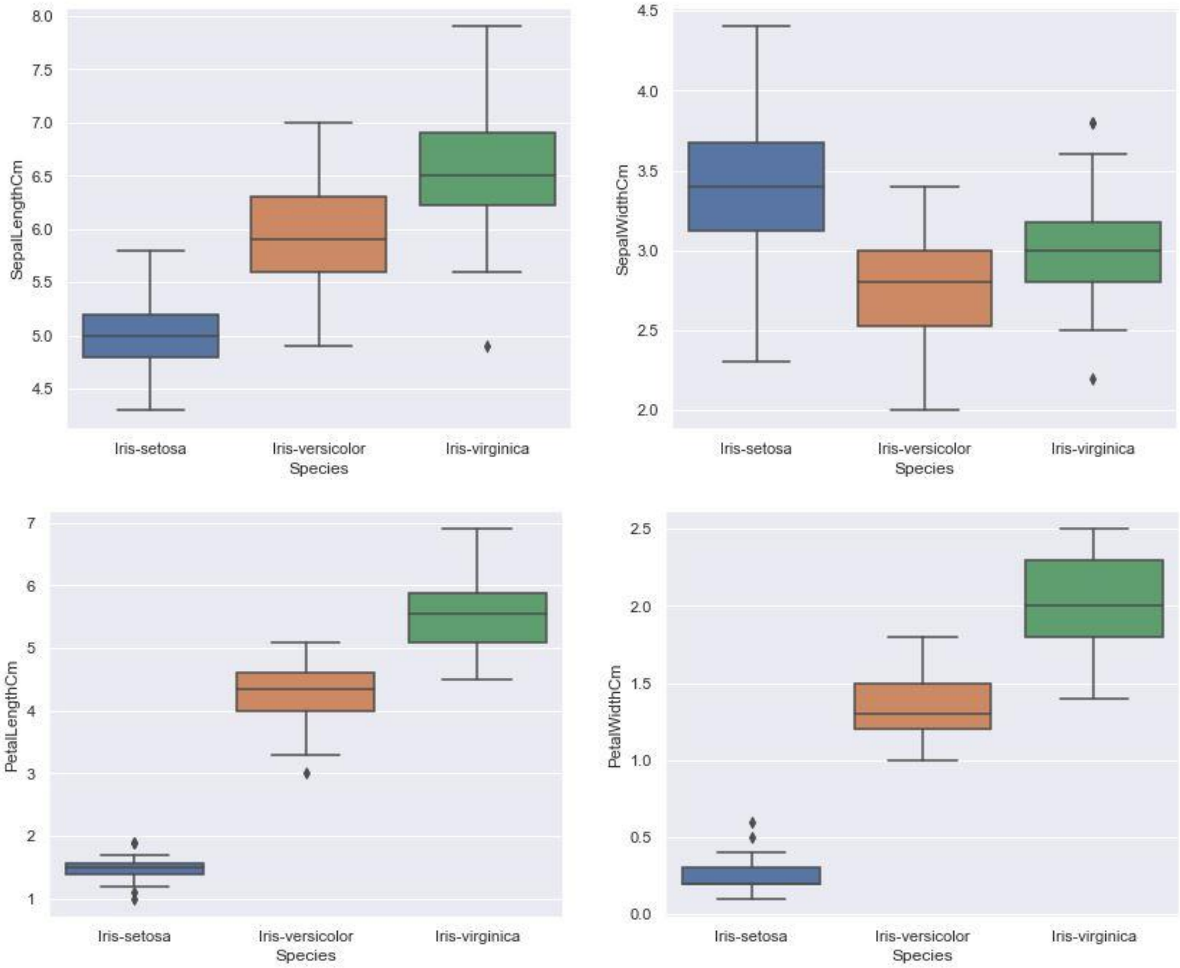


Figure 17 –Box Plot

4.2.6 Classification

```
In [21]: dataset.shape
Out[21]: (150, 5)

In [22]: dataset.head()
Out[22]:
```

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

```
In [23]: X_p = dataset.iloc[:,[2, 3]].values
         y = dataset.iloc[:, 4].values
         X_s = dataset.iloc[:,[0, 1]].values

In [24]: X_p.shape
Out[24]: (150, 2)

In [26]: y.shape
Out[26]: (150,)

In [27]: # Encoding categorical data (IT16001480)
         from sklearn.preprocessing import LabelEncoder, OneHotEncoder
         # Encoding the Dependent Variable
         labelencoder_y = LabelEncoder()
         y = labelencoder_y.fit_transform(y)
         y
         # Iris-setosa == 0
         # Iris-versicolor == 1
         # Iris-virginica == 2
Out[27]: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])

In [60]: # Splitting the dataset into the Training set and Test set (Petal Lenth vs Petal Width)
         from sklearn.model_selection import train_test_split
         X_trainp, X_testp, y_trainp, y_testp = train_test_split(X_p, y, test_size = 0.3, random_state = 0)

In [61]: # Splitting the dataset into the Training set and Test set (Sepal Length vs Sepal Width)
         from sklearn.model_selection import train_test_split
         X_trains, X_tests, y_trains, y_tests = train_test_split(X_s, y, test_size = 0.3, random_state = 0)
```

Figure 18 – Classification

4.2.7 Implementing Decision Tree Classification Algorithm

Following codes show the implementation of Decision tree classification algorithm

Step 1:

- First I set Decision Tree Classification to the Training set
- Then fit the classifier to tanning set using two properties. (Petal Length vs Petal Width)
- Finally predict the test data set results

Decision Tree Classifier

```
In [44]: # Fitting Decision Tree Classification to the Training set
         from sklearn.tree import DecisionTreeClassifier
         classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
```

Petal Length vs Petal Width`

```
In [45]: classifier.fit(X_trainp, y_trainp)

Out[45]: DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,
                     max_features=None, max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, presort=False, random_state=0,
                     splitter='best')

In [46]: # Predicting the Test set results
         y_predp = classifier.predict(X_testp)
         y_predp

Out[46]: array([2, 1, 0, 2, 0, 2, 0, 1, 1, 1, 2, 1, 1, 1, 1, 0, 1, 1, 0, 0, 2, 1,
                0, 0, 1, 0, 0, 1, 1, 0, 2, 1, 0, 2, 2, 1, 0, 2, 1, 1, 2, 0, 2, 0,
                0])
```

Figure 19 – step 1 process

Step 2:

- Then Measuring the accuracy
- Making confusion matrix

```
In [47]: # Measuring Accuracy
         from sklearn import metrics
         print('The accuracy of Decision Tree Classifier is: ', metrics.accuracy_score(y_predp, y_testp))

         The accuracy of Decision Tree Classifier is:  0.9555555555555556

In [48]: # Making confusion matrix
         from sklearn.metrics import confusion_matrix
         print(confusion_matrix(y_testp, y_predp))

         [[16  0  0]
          [ 0 17  1]
          [ 0  1 10]]
```

Figure 20 – step 2 process

Step 3:

- Visualizing the Training set results (Petal Length vs Petal Width)

```
In [49]: # Visualising the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_trainp, y_trainp
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('pink', 'lightgreen', 'lightblue')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green', 'blue'))(i), label = j)
plt.title('Decision Tree (Training set)')
plt.xlabel('PetalLengthCm')
plt.ylabel('PetalWidthCm')
plt.legend()
plt.show()

'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or
RGBA value for all points.
```

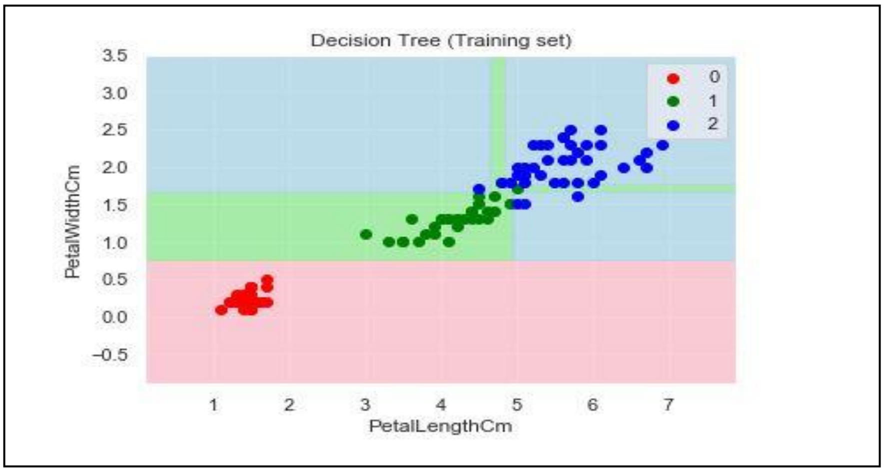


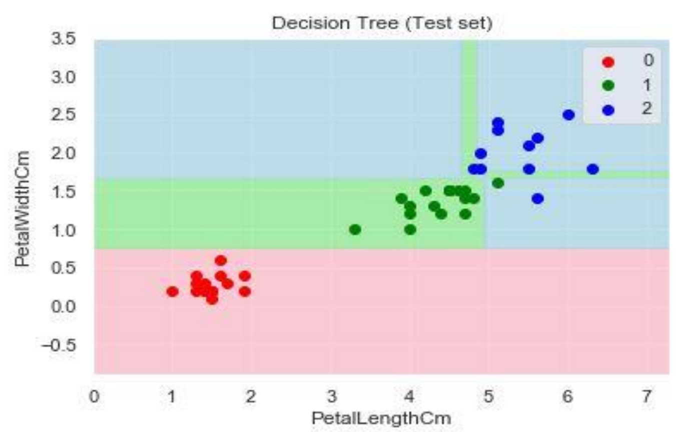
Figure 21 – step 3 process

Step 4:

- Visualizing the Test set results(Petal Length vs Petal Width)

```
In [50]: # Visualising the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_testp, y_testp
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('pink', 'lightgreen', 'lightblue')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green', 'blue'))(i), label = j)
plt.title('Decision Tree (Test set)')
plt.xlabel('PetalLengthCm')
plt.ylabel('PetalWidthCm')
plt.legend()
plt.show()

'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or
RGBA value for all points.
```



** Sepal Length vs Sepal Width**

Figure 22 – step 4 process

Step 5:

- Fitting Decision Tree classifier to the Training set
- Predicting the Test set results
- Measuring Accuracy
- Making confusion matrix

```
In [51]: # Fitting Decision Tree classifier to the Training set
         classifier.fit(X_trains, y_trains)

Out[51]: DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,
                     max_features=None, max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, presort=False, random_state=0,
                     splitter='best')

In [52]: # Predicting the Test set results
         y_preds = classifier.predict(X_tests)
         y_preds

Out[52]: array([1, 2, 0, 2, 0, 1, 0, 1, 1, 1, 1, 2, 1, 2, 1, 0, 2, 1, 0, 0, 1, 1,
                0, 0, 2, 0, 0, 2, 1, 0, 1, 0, 0, 1, 2, 1, 0, 2, 2, 1, 1, 0, 1, 0,
                0])

In [53]: # Measuring Accuracy
         from sklearn import metrics
         print('The accuracy of the Decision Tree Classifier using Sepals is:', metrics.accuracy_score(y_preds, y_tests))

         The accuracy of the Decision Tree Classifier using Sepals is: 0.6444444444444445

In [54]: # Making confusion matrix
         from sklearn.metrics import confusion_matrix
         print(confusion_matrix(y_tests, y_preds))

         [[16  0  0]
          [ 1 10  7]
          [ 0  8  3]]
```

Figure 23 – step 5 process

Step 6:

- Visualizing the Training set results (Sepal Length vs Sepal Width)

```
In [55]: # Visualising the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_trains, y_trains
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('pink', 'lightgreen', 'lightblue')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green', 'blue'))(i), label = j)
plt.title('Decision Tree(Training set)')
plt.xlabel('SepalLengthCm')
plt.ylabel('SepalWidthCm')
plt.legend()
plt.show()

'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or
RGBA value for all points.
```

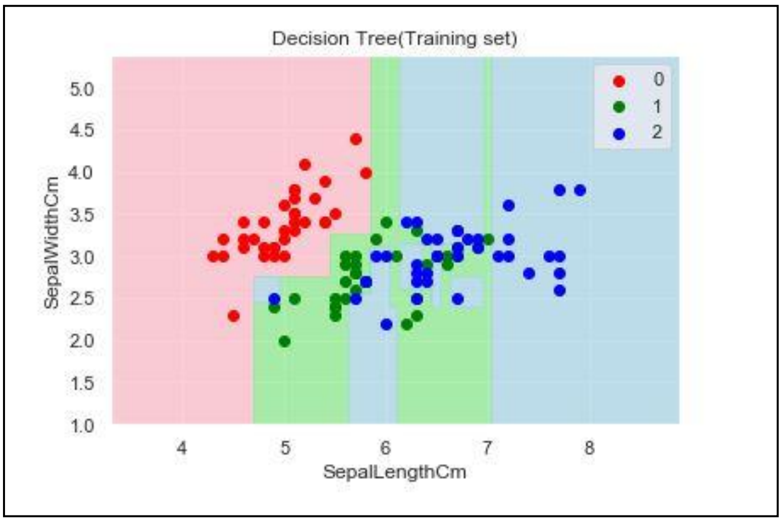


Figure 24 – step 6 process

Step 7:

- Visualizing the Test set results (Sepal Length vs Sepal Width)

```
In [56]: # Visualising the Test set results
         from matplotlib.colors import ListedColormap
         X_set, y_set = X_tests, y_tests
         X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                              np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
         plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
                      alpha = 0.75, cmap = ListedColormap(('pink', 'lightgreen', 'lightblue')))
         plt.xlim(X1.min(), X1.max())
         plt.ylim(X2.min(), X2.max())
         for i, j in enumerate(np.unique(y_set)):
             plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                         c = ListedColormap(('red', 'green', 'blue'))(i), label = j)
         plt.title('Decision Tree(Test set)')
         plt.xlabel('SepalLengthCm')
         plt.ylabel('SepalWidthCm')
         plt.legend()
         plt.show()

'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in
case its length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or
RGBA value for all points.
```

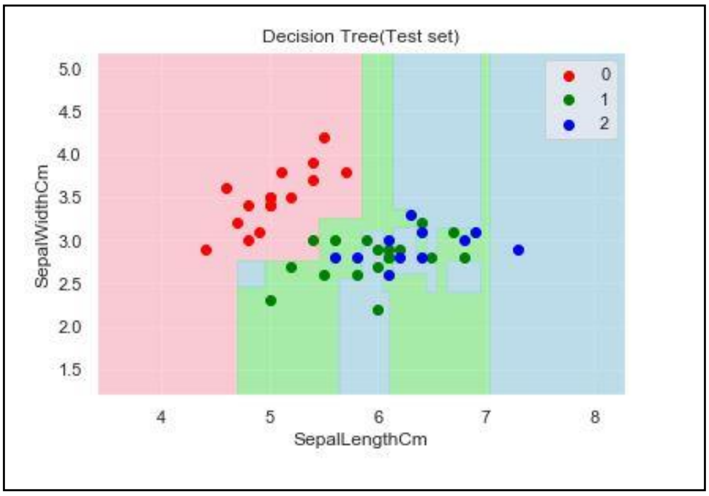


Figure 25 – step 7 process

Observations

- The accuracy of the Decision Tree Classifier using Petals is: 95 %
- The accuracy of the Decision Tree Classifier using Sepals is: 64 %

Overall Observations

- Using Petals over Sepal for training the data gives a much better accuracy.

5.Evaluation

5.1 Future Works

*

- Training this dataset again with different training set and test set.
- In future we can train this dataset with some other different machine learning Algorithms. After that we can improve accuracy of our algorithm.
- Try different dataset with different features to do prediction.
- Remove some unnecessary features by looking at the feature importance and do Classification again. After that we can improve accuracy of our algorithm.
- Use other ensemble methods available to improve accuracy. Such as Bagging and Boosting method

6.Discussion

Under discussion we are go0ing to speak about two main ensemble techniques. Bagging and Boosting are similar in that they are both ensemble techniques, where a set of weak learners are combined to create a strong learner that obtains better performance than a single one

9.1 Bagging

Bootstrap Aggregation (or Bagging for short), is a simple and very powerful ensemble method.
Bagging is the application of the Bootstrap procedure to a high-variance machine learning algorithm, typically decision trees.

9.2 Boosting

Boosting refers to a group of algorithms that utilize weighted averages to make weak learners into stronger learners.
Unlike bagging that had each model run independently and then aggregate the outputs at the end without preference to any model.
Boosting is all about “teamwork”. Each model that runs, dictates what features the next model will focus on. [2]

7. Appendix

7.1 Code

```
#H.K.D.C.Jayalath-IT16001480

# importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn import datasets
from IPython.display import Image
import pydotplus
from sklearn.datasets import load_svmlight_file
import seaborn as sns; sns.set()
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.tree import export_graphviz
import seaborn as sns
from IPython.display import Image

#Picture of the three different Iris Flower types:

# The Iris Setosa
from IPython.display import Image
url =
'http://upload.wikimedia.org/wikipedia/commons/5/56/Kosaciec_szczecinkowaty_Iris_setosa.jpg'
Image(url,width=300, height=300)

# The Iris Versicolor
from IPython.display import Image
url = 'http://upload.wikimedia.org/wikipedia/commons/4/41/Iris_versicolor_3.jpg'
Image(url,width=300, height=300)

# The Iris Virginica
from IPython.display import Image
url = 'http://upload.wikimedia.org/wikipedia/commons/9/9f/Iris_virginica.jpg'
Image(url,width=300, height=300)

# importing the dataset
dataset = pd.read_csv('Iris.csv')
dataset.head()
dataset.info()

# Remove Id Column
dataset.drop("Id", axis=1, inplace = True)
dataset["Species"].value_counts()

#load data
iris=datasets.load_iris()
x=iris.data
y=iris.target
iris.feature_names
iris.target_names

#create decision tree classifier
clf=DecisionTreeClassifier(random_state=0)

#train the modle
model=clf.fit(x,y)
```

```
dot_data = tree.export_graphviz(model,out_file=None,
                                feature_names=iris.feature_names,
                                class_names=iris.target_names)
```

#**draw graph**
```
graph=pydotplus.graph_from_dot_data(dot_data)
```

#**show graph**
```
Image(graph.create_png())
#create pdf
graph.write_pdf("iris.pdf")
```

Create scatter Plot

#**By selecting two features SepalLengthCm and SepelWidthCm**
```
sns.FacetGrid(dataset, hue="Species", size=6) \
   .map(plt.scatter, "SepalLengthCm", "SepalWidthCm") \
   .add_legend()
plt.title('Sepal Length Vs Sepel Width')
plt.show()
```

 # By selecting two features PetalLengthCm and PetalWidthCm
```
sns.FacetGrid(dataset, hue="Species", size=6) \
   .map(plt.scatter, "PetalLengthCm", "PetalWidthCm") \
   .add_legend()
plt.title('Petal Length Vs Petal Width')
plt.show()
```

#**create pair plot**
```
sns.pairplot(dataset, hue="Species", size=3);
plt.show()
```

#create box plot
```
plt.figure(figsize=(14,12))
plt.subplot(2,2,1)
sns.boxplot(x='Species', y = 'SepalLengthCm', data=dataset)
plt.subplot(2,2,2)
sns.boxplot(x='Species', y = 'SepalWidthCm', data=dataset)
plt.subplot(2,2,3)
sns.boxplot(x='Species', y = 'PetalLengthCm', data=dataset)
plt.subplot(2,2,4)
sns.boxplot(x='Species', y = 'PetalWidthCm', data=dataset)
```

#create violin plot
```
plt.figure(figsize=(15,10))
plt.subplot(2,2,1)
sns.violinplot(x='Species', y = 'SepalLengthCm', data=dataset)
plt.subplot(2,2,2)
sns.violinplot(x='Species', y = 'SepalWidthCm', data=dataset)
plt.subplot(2,2,3)
sns.violinplot(x='Species', y = 'PetalLengthCm', data=dataset)
plt.subplot(2,2,4)
sns.violinplot(x='Species', y = 'PetalWidthCm', data=dataset
```

#**classification**
```
dataset.shape
X_p = dataset.iloc[:,[2, 3]].values
y = dataset.iloc[:, 4].values
X_s = dataset.iloc[:,[0, 1]].values
X_p.shape
X_s.shape
y.shape
```

Encoding categorical data
```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
# Encoding the Dependent Variable
labelencoder_y = LabelEncoder()
y = labelencoder_y.fit_transform(y)
y
# Iris-setosa == 0
# Iris-versicolor == 1
# Iris-virginica == 2
```

 # Splitting the dataset into the Training set and Test set (Petal Lenth vs Petal Width)

```
from sklearn.model_selection import train_test_split
X_trainp, X_testp, y_trainp, y_testp = train_test_split(X_p, y, test_size = 0.3, random_state =
0)
```

 # Splitting the dataset into the Training set and Test set (Sepal Length vs Sepal Width)

```
from sklearn.model_selection import train_test_split
X_trains, X_tests, y_trains, y_tests = train_test_split(X_s, y, test_size = 0.3, random_state =
0)
```

#**Decision Tree**

 # Fitting Decision Tree Classification to the Training set

```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)

classifier.fit(X_trainp, y_trainp)
```

 # Predicting the Test set results

```
y_predp = classifier.predict(X_testp)
y_predp
```

 # Measuring Accuracy

```
from sklearn import metrics
print('The accuracy of Decision Tree Classifier is: ', metrics.accuracy_score(y_predp,
y_testp))
```

 # Making confusion matrix

```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_testp, y_predp))
```

 # Visualising the Training set results

```
from matplotlib.colors import ListedColormap
X_set, y_set = X_trainp, y_trainp
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1,
step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step =
0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('pink', 'lightgreen', 'lightblue')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green', 'blue'))(i), label = j)
plt.title('Decision Tree (Training set)')
plt.xlabel('PetalLengthCm')
plt.ylabel('PetalWidthCm')
plt.legend()
plt.show()
```

 # Visualising the Test set results

```
from matplotlib.colors import ListedColormap
X_set, y_set = X_testp, y_testp
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1,
step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step =
0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('pink', 'lightgreen', 'lightblue')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green', 'blue'))(i), label = j)
plt.title('Decision Tree (Test set)')
plt.xlabel('PetalLengthCm')
plt.ylabel('PetalWidthCm')
plt.legend()
plt.show()
```

#Sepal Length vs Sepal Width

# **Fitting Decision Tree classifier to the Training set**
classifier.fit(X_trains, y_trains)

# Predicting the Test set results
y_preds = classifier.predict(X_tests)
y_preds

# **Measuring Accuracy**
from sklearn import metrics
print('The accuracy of the Decision Tree Classifier using Sepals is:', metrics.accuracy_score(y_preds, y_tests))

# **Making confusion matrix**
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_tests, y_preds))

Visualising the Training set results
```
from matplotlib.colors import ListedColormap
X_set, y_set = X_trains, y_trains
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('pink', 'lightgreen', 'lightblue')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green', 'blue'))(i), label = j)
plt.title('Decision Tree(Training set)')
plt.xlabel('SepalLengthCm')
plt.ylabel('SepalWidthCm')
plt.legend()
plt.show()
```

Visualising the Test set results
```
from matplotlib.colors import ListedColormap
X_set, y_set = X_tests, y_tests
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('pink', 'lightgreen', 'lightblue')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green', 'blue'))(i), label = j)
plt.title('Decision Tree(Test set)')
plt.xlabel('SepalLengthCm')
plt.ylabel('SepalWidthCm')
plt.legend()
plt.show()
```

References

[1] - https://en.wikipedia.org/wiki/Iris_flower_data_set

[2] - https://becominghuman.ai/ensemble-learning-bagging-and-boosting-d20f38be9b1e