SPARKS FOUNDATION

TSF DATA SCIENCE TASK 6

PREDICTION USING DECISION TREE ALGORITHM

MARGARET OLUWADARE

Data Science and Business Analytics intern at The Sparks Foundation.

Flow of Analysis

1. Introduction
2. Import the required libraries
3. Exploratory Data Analysis
4. Building a prediction model

1. INTRODUCTION

Using sklearn datasets Iris data set to train our model using Decision tree classifier. We will figure out accuracy of your model and use that to predict different samples in our test dataset. The data contain the following features:

1. Sepal Length
2. Sepal Width
3. Petal Length
4. Petal Width

Using above 4 features you will clasify a flower in one of the three categories,

1. Setosa
2. Versicolour
3. Virginica

2. IMPORT LIBRARIES

In [12]:
```
#import libraries for the workflow
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import datasets

##LIBRARY FOR VISUALIZATION
from autoviz.AutoViz_Class import AutoViz_Class
AV = AutoViz_Class()
import seaborn as sns
```

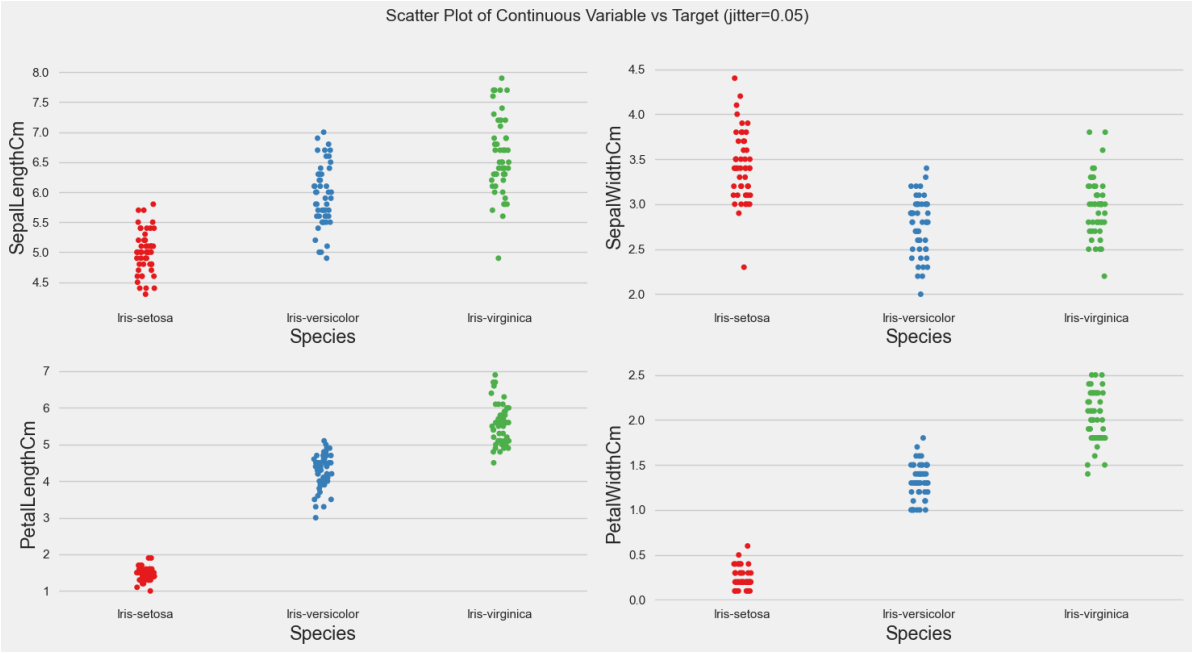
```
%matplotlib inline

## LIBRARY FOR MODELLING
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn import tree
import graphviz
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import multilabel_confusion_matrix, accuracy_score
from sklearn.metrics import multilabel_confusion_matrix, accuracy_score
```

3. Exploratory Data Analysis

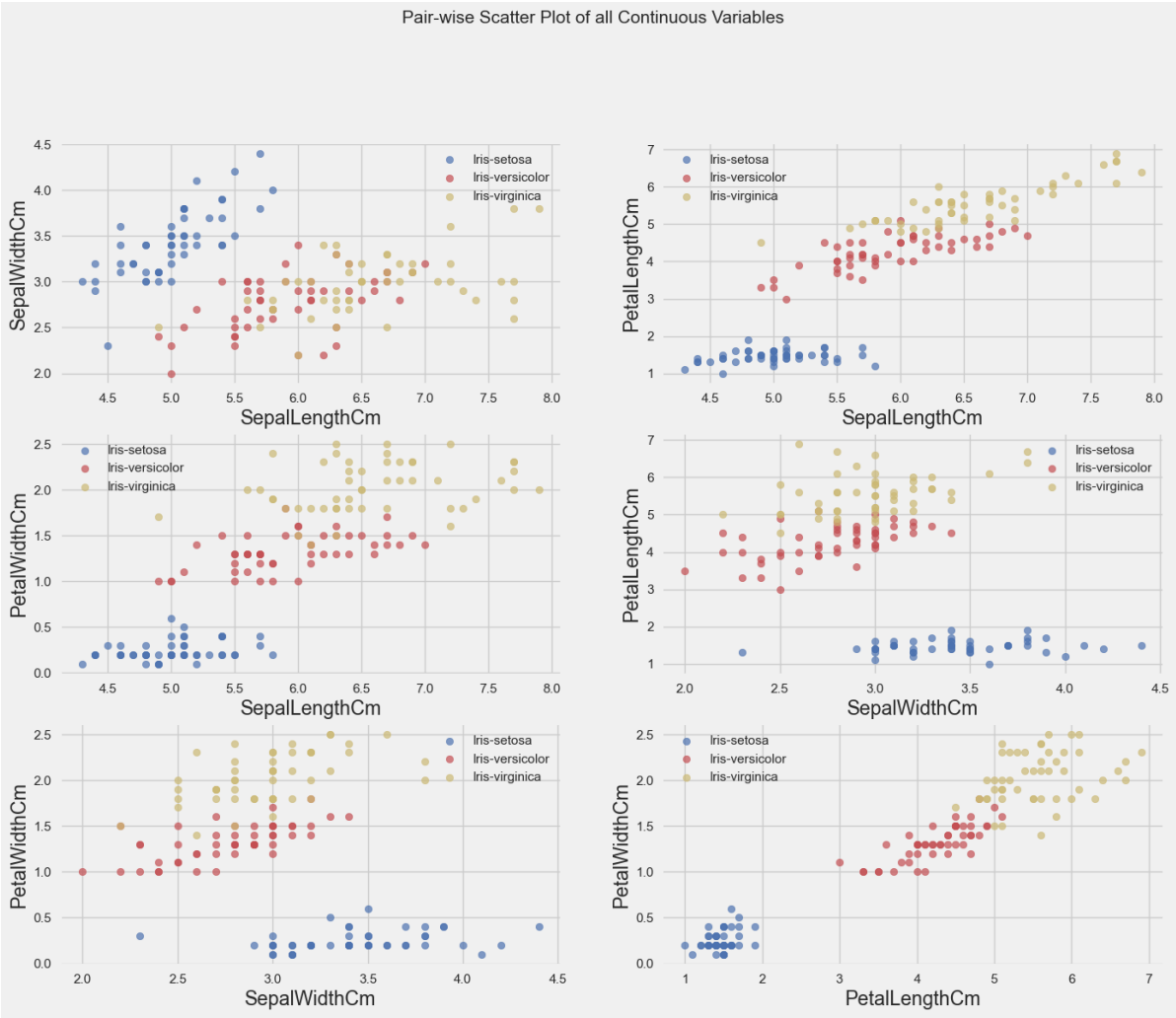
We will employ the Autoviz for a total visualization of the dataset.

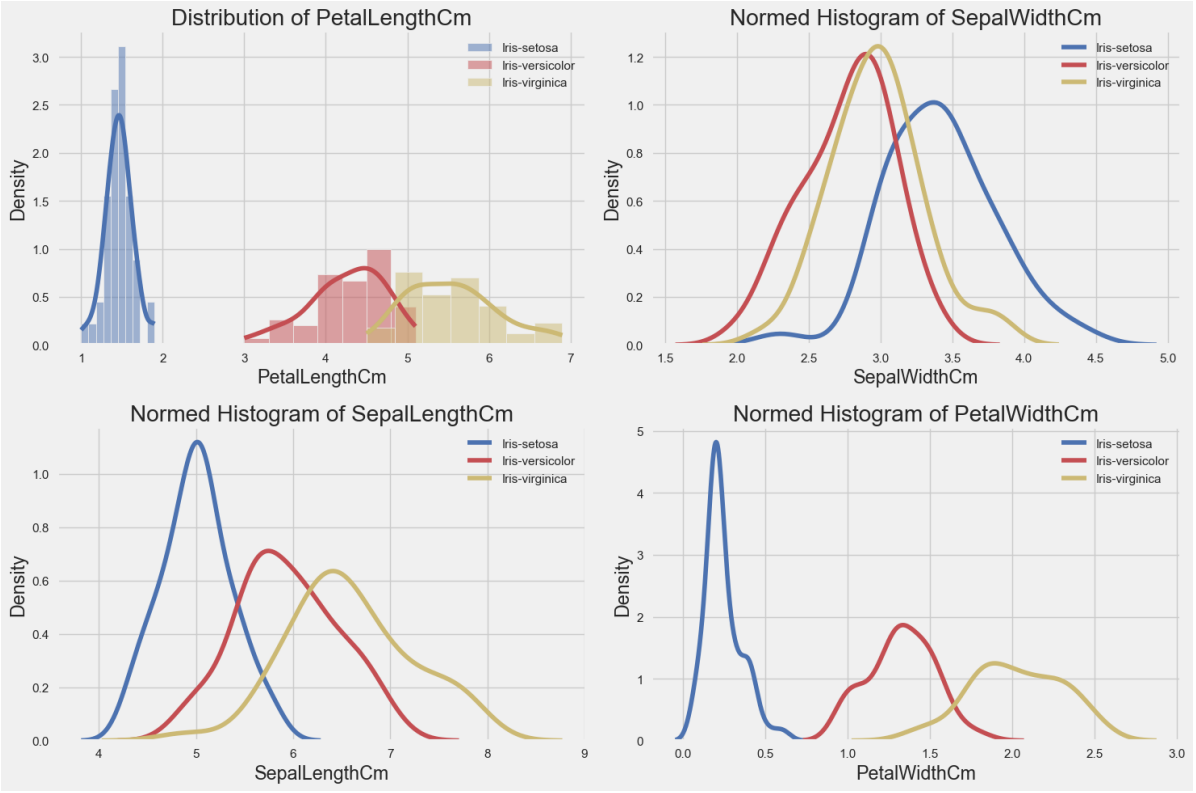
In [2]:
```
filename = 'iriss.csv'
sep = ","
dft = AV.AutoViz(
filename,
sep=",",
depVar="Species",
dfte=None,
header=0,
verbose=0,
lowess=False,
chart_format="svg",
max_rows_analyzed=150000,
max_cols_analyzed=50,
)
```

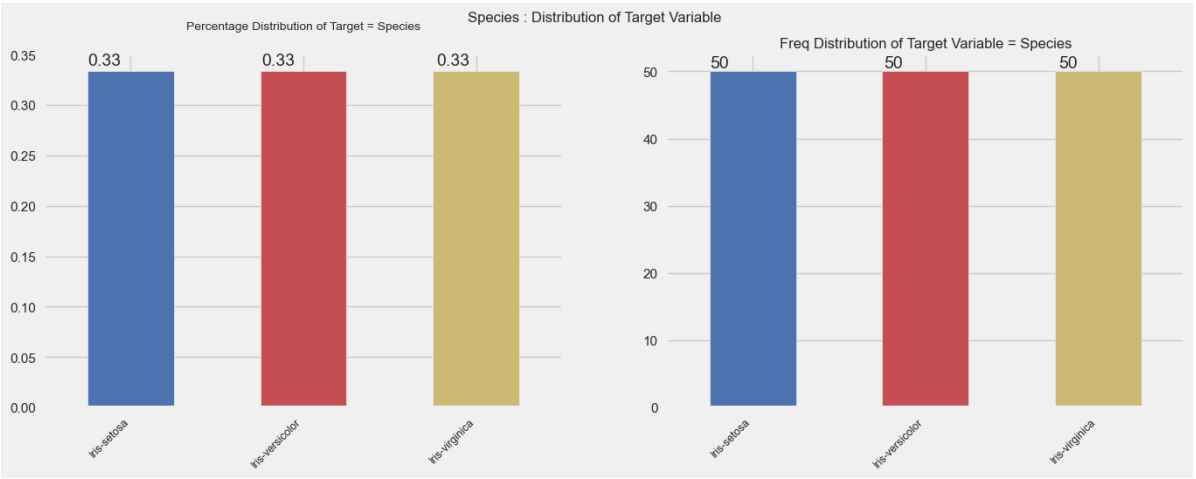
```
Shape of your Data Set loaded: (150, 6)
################################################################################################
#####
######################### C L A S S I F Y I N G  V A R I A B L E S  ###############
#####
################################################################################################
#####
Classifying variables in data set...
    Number of Numeric Columns =  4
    Number of Integer-Categorical Columns =  0
    Number of String-Categorical Columns =  0
    Number of Factor-Categorical Columns =  0
    Number of String-Boolean Columns =  0
    Number of Numeric-Boolean Columns =  0
    Number of Discrete String Columns =  0
    Number of NLP String Columns =  0
    Number of Date Time Columns =  0
    Number of ID Columns =  1
    Number of Columns to Delete =  0
    5 Predictors classified...
        1 variables removed since they were ID or low-information variables

################ Multi_Classification problem ####################
```

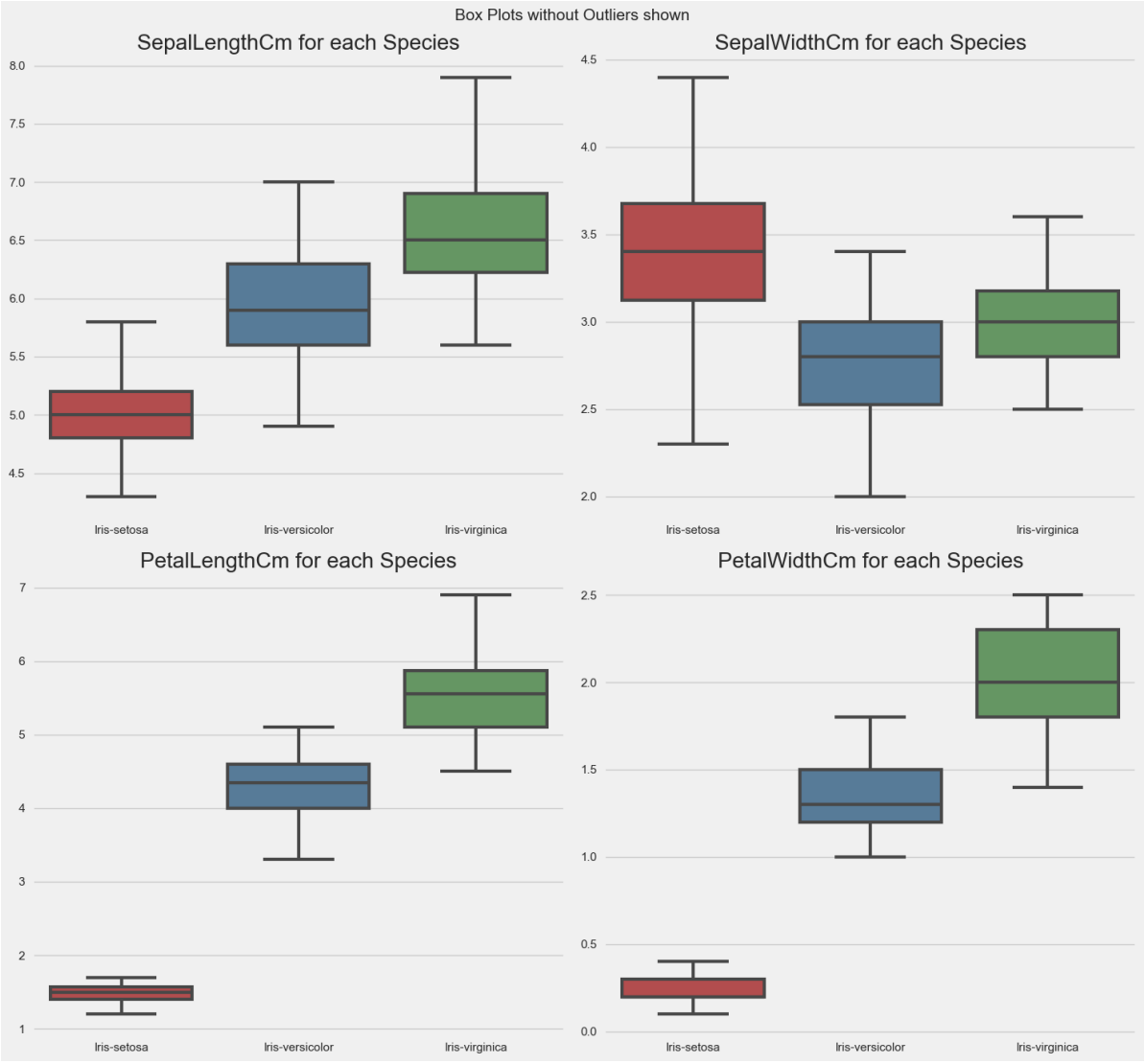


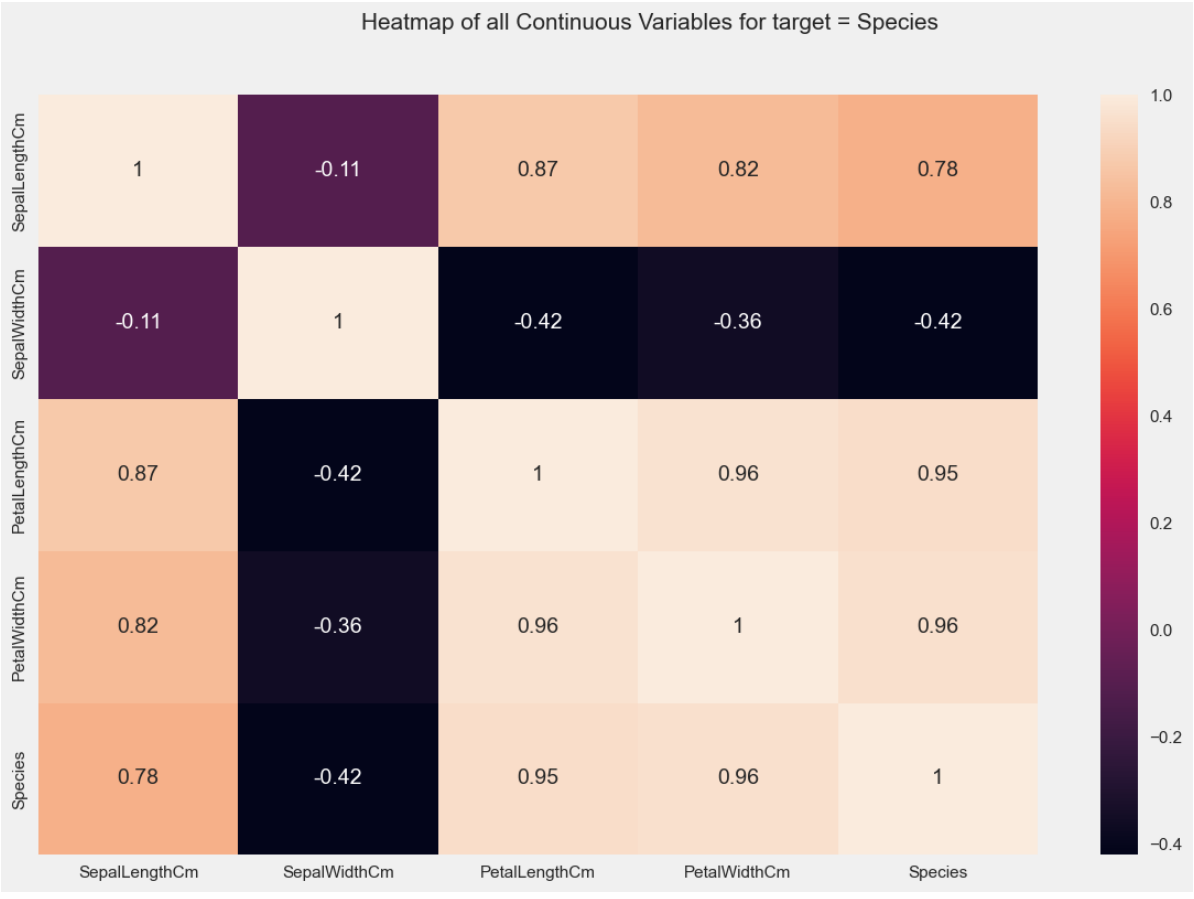

```
Total Number of Scatter Plots = 10
```




Distribution of PetalLengthCm
Density
PetalLengthCm
Iris-setosa
Iris-versicolor
Iris-virginica
Normed Histogram of SepalWidthCm
Density
SepalWidthCm
Iris-setosa
Iris-versicolor
Iris-virginica
Normed Histogram of SepalLengthCm
Density
SepalLengthCm
Iris-setosa
Iris-versicolor
Iris-virginica
Normed Histogram of PetalWidthCm
Density
PetalWidthCm
Iris-setosa
Iris-versicolor
Iris-virginica


Percentage Distribution of Target = Species
0.33
0.33
0.33
Iris-setosa
Iris-versicolor
Iris-virginica
Species : Distribution of Target Variable
Freq Distribution of Target Variable = Species
50
50
50
Iris-setosa
Iris-versicolor
Iris-virginica


Box Plots without Outliers shown
SepalLengthCm for each Species
SepalWidthCm for each Species
PetalLengthCm for each Species
PetalWidthCm for each Species
Iris-setosa
Iris-versicolor
Iris-virginica


Heatmap of all Continuous Variables for target = Species
SepalLengthCm
SepalWidthCm
PetalLengthCm
PetalWidthCm
Species
1
-0.11
0.87
0.82
0.78
-0.11
1
-0.42
-0.36
-0.42
0.87
-0.42
1
0.96
0.95
0.82
-0.36
0.96
1
0.96
0.78
-0.42
0.95
0.96
1

```
No categorical or boolean vars in data set. Hence no pivot plots...
No categorical or numeric vars in data set. Hence no bar charts.
All Plots done
Time to run AutoViz = 10 seconds

 ###################### AUTO VISUALIZATION Completed ########################
```

A corresponding correlation matrix to quantitatively examine the relationship between variables is also produced. The following are the takebacks form the correlation matrix:

1. Petal measurements have highly positive correlation

2. Sepal measurements are uncorrelated.

3. Petal features also have relatively high correlation with sepal_length, but not with sepal_width.

4. DATA MODELLING AND ANALYSIS

We have gained some insight into the data, so we are looking at modelling the data using suitable machine learning algorithm specifically Decision tree.

In [3]:
```
data = pd.read_csv('iriss.csv')
```

Train-Test Split

Now, we can split the dataset into a training set and a test set. The test set will be use for the purpose of reporting and validation of our model. In addition, I used either a stratified hold-out approach or cross validation to estimate model accuracy if the need arise.

Generally, the rule of thumb is have 20–30% of dataset as the test set but due to the size of our dataset, we will take 30% to ensure there are enough data points to test the model performance.

In [4]:
```
data.columns
```

Out[4]:
```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
```

In [5]:
```
X = data[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']]
y = data['Species']
```

In [ ]:
```
#Rename "class" column to avoid conflict with inbuilt function
#data.rename(columns = {'sepal_length':'sepal_length', 'sepal_width':'sepal_width',
```

In [ ]:
```
#Check the distribution of the data set by class
#data.groupby('target').size()
```

In [6]:
```
Xtrain, Xtest, Ytrain, Ytest = train_test_split(X, y, test_size=0.30, random_state
```

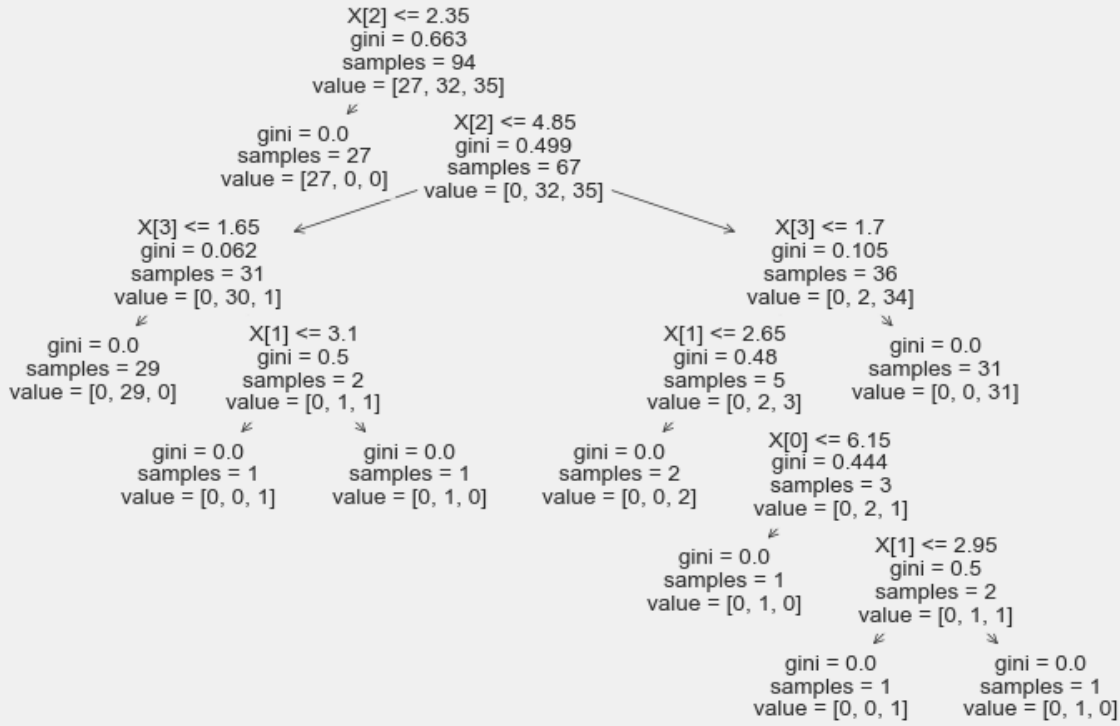
In [7]:
```
#spliting training data into validation train and validation test
Xt, Xcv, Yt, Ycv = train_test_split(Xtrain, Ytrain, test_size=0.10, random_state=3
```

In [8]:
```
'''Now we have create a Decision tree classifier and trained it with training datas
data_clf = DecisionTreeClassifier(criterion='gini',min_samples_split=2)
data_clf.fit(Xt, Yt)
```

Out[8]:
```
DecisionTreeClassifier()
```

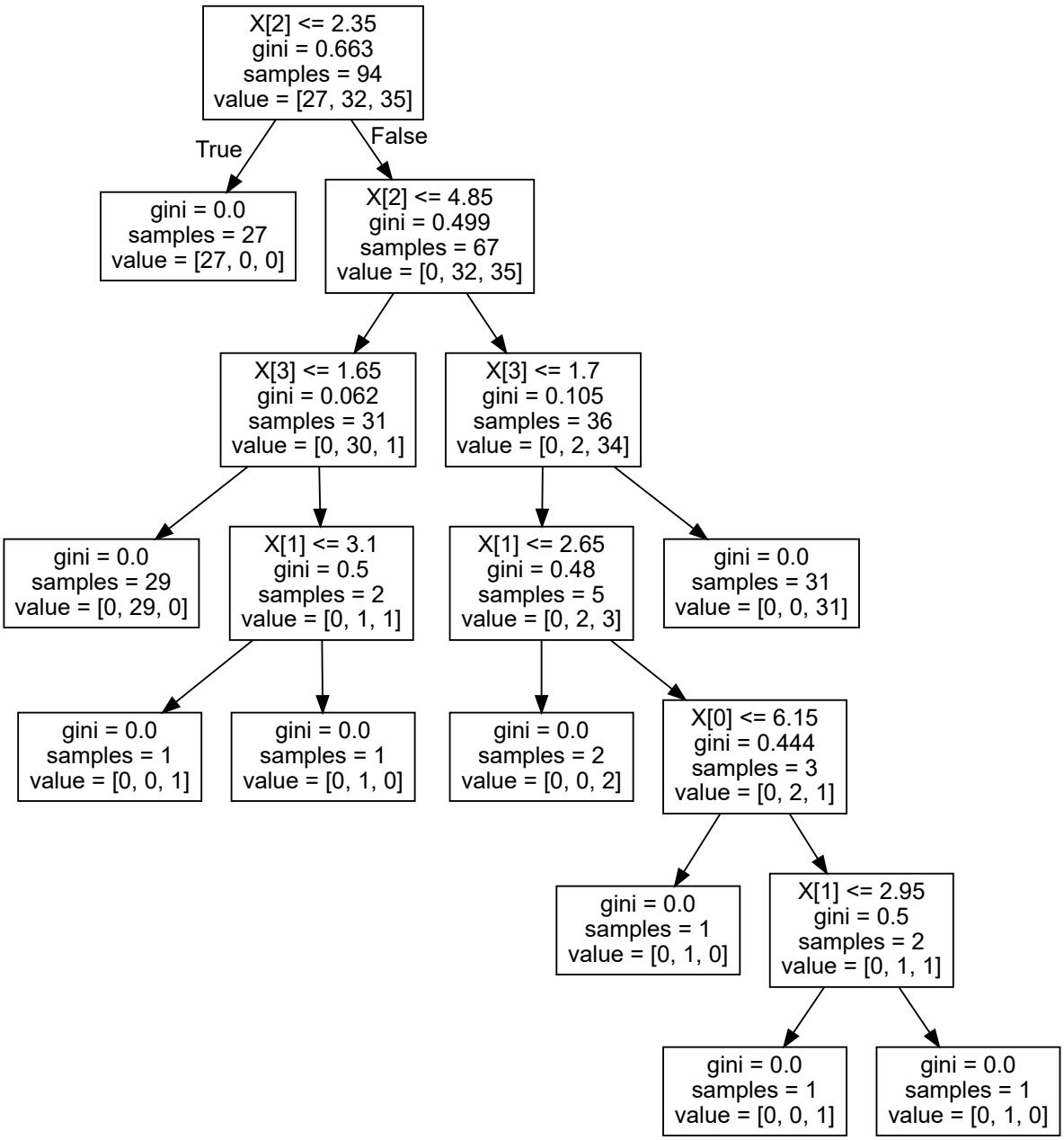
In [9]:
```
'''Visualized the Tree which is formed on train dataset'''

tree.plot_tree(data_clf)
```

Out[9]:
```
[Text(0.36363636363636365, 0.9285714285714286, 'X[2] <= 2.35\ngini = 0.663\nsample
s = 94\nvalue = [27, 32, 35]'),
 Text(0.2727272727272727, 0.7857142857142857, 'gini = 0.0\nsamples = 27\nvalue =
[27, 0, 0]'),
 Text(0.45454545454545453, 0.7857142857142857, 'X[2] <= 4.85\ngini = 0.499\nsample
s = 67\nvalue = [0, 32, 35]'),
 Text(0.18181818181818182, 0.6428571428571429, 'X[3] <= 1.65\ngini = 0.062\nsample
s = 31\nvalue = [0, 30, 1]'),
 Text(0.09090909090909091, 0.5, 'gini = 0.0\nsamples = 29\nvalue = [0, 29, 0]'),
 Text(0.2727272727272727, 0.5, 'X[1] <= 3.1\ngini = 0.5\nsamples = 2\nvalue = [0,
1, 1]'),
 Text(0.18181818181818182, 0.35714285714285715, 'gini = 0.0\nsamples = 1\nvalue =
[0, 0, 1]'),
 Text(0.36363636363636365, 0.35714285714285715, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1, 0]'),
 Text(0.7272727272727273, 0.6428571428571429, 'X[3] <= 1.7\ngini = 0.105\nsamples
= 36\nvalue = [0, 2, 34]'),
 Text(0.6363636363636364, 0.5, 'X[1] <= 2.65\ngini = 0.48\nsamples = 5\nvalue =
[0, 2, 3]'),
 Text(0.5454545454545454, 0.35714285714285715, 'gini = 0.0\nsamples = 2\nvalue =
[0, 0, 2]'),
 Text(0.7272727272727273, 0.35714285714285715, 'X[0] <= 6.15\ngini = 0.444\nsample
s = 3\nvalue = [0, 2, 1]'),
 Text(0.6363636363636364, 0.21428571428571427, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1, 0]'),
 Text(0.8181818181818182, 0.21428571428571427, 'X[1] <= 2.95\ngini = 0.5\nsamples
= 2\nvalue = [0, 1, 1]'),
 Text(0.7272727272727273, 0.07142857142857142, 'gini = 0.0\nsamples = 1\nvalue =
[0, 0, 1]'),
 Text(0.9090909090909091, 0.07142857142857142, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1, 0]'),
 Text(0.8181818181818182, 0.5, 'gini = 0.0\nsamples = 31\nvalue = [0, 0, 31]')]
```



In [10]:
```
'''Visualizing Decision Tree using graphviz library'''

grpdata = tree.export_graphviz(data_clf, out_file=None)

graph = graphviz.Source(grpdata)
graph
```

Out[10]:



In [11]:
```
''' As our model has been trained we can validate our Decision tree using cross val

print('Accuracy score is:',cross_val_score(data_clf, Xt, Yt, cv=3, scoring='accura
```

```
Accuracy score is: 0.9361559139784946
```

In [13]:
```
'''Checking validation test data on our trained model and getting performance metri

Y_hat = data_clf.predict(Xcv)
print('Accuracy score for validation test data is:',accuracy_score(Ycv, Y_hat))
multilabel_confusion_matrix(Ycv , Y_hat)
```

```
Accuracy score for validation test data is: 0.9090909090909091
```

Out[13]:
```
array([[[5, 0],
        [0, 6]],

       [[8, 0],
        [1, 2]],

       [[8, 1],
        [0, 2]]], dtype=int64)
```

In [14]:
```
'''Checking our model performance on actual unseen test data'''
YT_hat = data_clf.predict(Xtest)
```

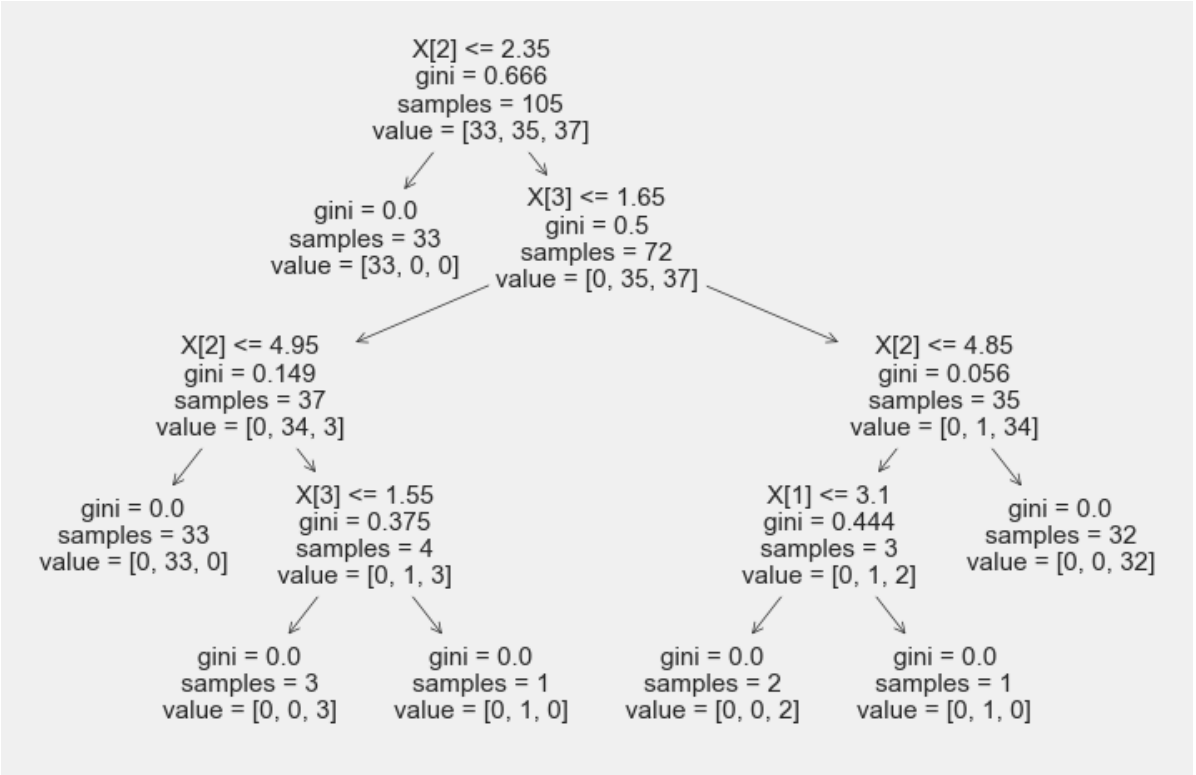
```
YT_hat

print('Model Accuracy Score on totally unseen data(Xtest) is:',accuracy_score(Ytes
multilabel_confusion_matrix(Ytest , YT_hat)
```

Out[14]:
```
Model Accuracy Score on totally unseen data(Xtest) is: 95.55555555555556 %
array([[[28,  0],
        [ 0, 17]],

       [[29,  1],
        [ 1, 14]],

       [[31,  1],
        [ 1, 12]]], dtype=int64)
```

As we know our selected feature are working well and model gives very good accuracy score on validate or actual test data. So Now we can trained our model on Actual train dataset with selected features for evaluating/ deploying our model in real world cases.

In [15]:
```
'''Training model on Actual train data... '''
dataFclf = DecisionTreeClassifier(criterion='gini',min_samples_split=2)
dataFclf.fit(Xtrain, Ytrain)

#Visualize tree structure..
tree.plot_tree(dataFclf)
```

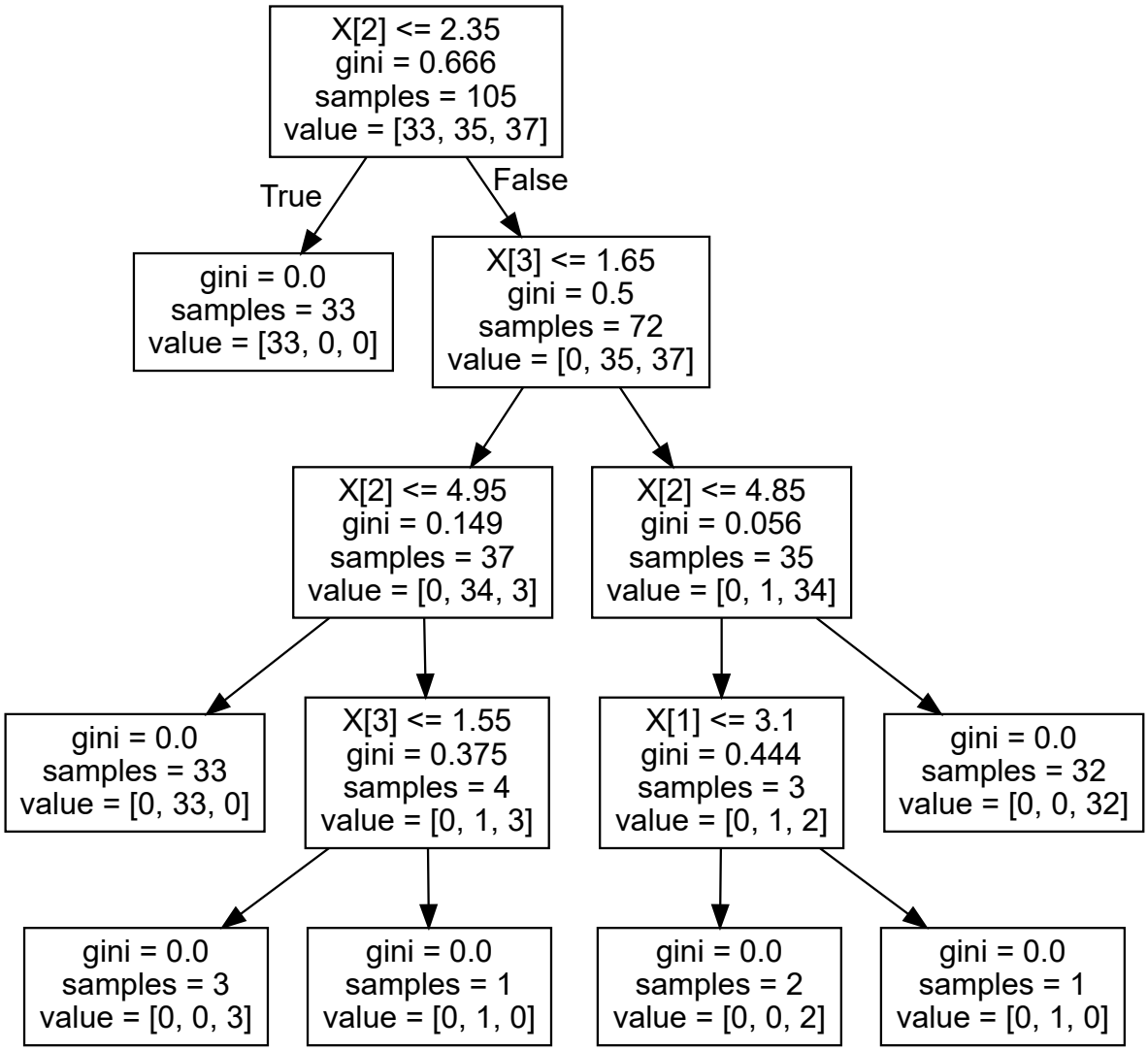
Out[15]:
```
[Text(0.4, 0.9, 'X[2] <= 2.35\ngini = 0.666\nsamples = 105\nvalue = [33, 35, 3
7]'),
 Text(0.3, 0.7, 'gini = 0.0\nsamples = 33\nvalue = [33, 0, 0]'),
 Text(0.5, 0.7, 'X[3] <= 1.65\ngini = 0.5\nsamples = 72\nvalue = [0, 35, 37]'),
 Text(0.2, 0.5, 'X[2] <= 4.95\ngini = 0.149\nsamples = 37\nvalue = [0, 34, 3]'),
 Text(0.1, 0.3, 'gini = 0.0\nsamples = 33\nvalue = [0, 33, 0]'),
 Text(0.3, 0.3, 'X[3] <= 1.55\ngini = 0.375\nsamples = 4\nvalue = [0, 1, 3]'),
 Text(0.2, 0.1, 'gini = 0.0\nsamples = 3\nvalue = [0, 0, 3]'),
 Text(0.4, 0.1, 'gini = 0.0\nsamples = 1\nvalue = [0, 1, 0]'),
 Text(0.8, 0.5, 'X[2] <= 4.85\ngini = 0.056\nsamples = 35\nvalue = [0, 1, 34]'),
 Text(0.7, 0.3, 'X[1] <= 3.1\ngini = 0.444\nsamples = 3\nvalue = [0, 1, 2]'),
 Text(0.6, 0.1, 'gini = 0.0\nsamples = 2\nvalue = [0, 0, 2]'),
 Text(0.8, 0.1, 'gini = 0.0\nsamples = 1\nvalue = [0, 1, 0]'),
 Text(0.9, 0.3, 'gini = 0.0\nsamples = 32\nvalue = [0, 0, 32]')]
```



In [16]:

```
'''Visualizing Decision Tree using graphviz library'''

grpfdata = tree.export_graphviz(dataFclf, out_file=None)

graph1 = graphviz.Source(grpfdata)
graph1
```

Out[16]:



In [17]:
```
'''Checking the performance of model on Actual Test data'''

YT_Fhat = dataFclf.predict(Xtest)
YT_Fhat

print('Model Accuracy Score on totally unseen data(Xtest) is:',accuracy_score(Ytes
multilabel_confusion_matrix(Ytest , YT_Fhat)
```

Out[17]:
```
Model Accuracy Score on totally unseen data(Xtest) is: 95.55555555555556 %
array([[[28,  0],
        [ 0, 17]],

       [[29,  1],
        [ 1, 14]],

       [[31,  1],
        [ 1, 12]]], dtype=int64)
```

Conclusion

Our trained model was able to capture all the iris species classes of the dataset at about 90% accuracy for validation set, and 95% for the trained set.

The test model score was about 95% accuracy, which is also a very good one. Showing that our tain model is actually not overfiting the model.

We explored the Iris dataset, and then built a decision tree or real world deployment. We saw that the petal measurements are more helpful at classifying instances of flowers than the sepal ones. Furthermore, ourmodels achieved a test accuracy of 95%.

In [ ]:

In [ ]: