In [ ]:
```
import sklearn
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn import tree
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
import sys

#Input: The Iris data file
#Processing: Replaces Iris labels with 0, 1, 2; Randomizes each record entry
#Output: Returns a Feature file and a label file
def Process_DataSet():
    with open ("iris.data") as f:
        lines = [i[:-1] for i in f.readlines()]
        n = ["Iris-setosa", "Iris-versicolor", "Iris-virginica"]
        x = [n.index(i.split(",")[-1]) for i in lines if i !=""]
        x = np.array(x, dtype = np.int32)
        y = [[float(j) for j in i.split(",")[:-1]] for i in lines if i != ""]
        y = np.array(y)
        i = np.argsort(np.random.random(x.shape[0]))
        x = x[i]
        y = y[i]
        np.save("iris_features.npy", y)
        np.save("iris_labels.npy", x)

#Input: Selected number of trees in forest
#Processing: Create training, testing sets, and Random Forest Object (based upon
selected trees)
#Output: Training set, testing set, and Random Forest Object
def Create_Forest_Object(forest_size):
    x = np.load("iris_features.npy")
    y = np.load("iris_labels.npy")
    N = 120
    x_train = x[:N]; x_test = x[N:] #Features
    y_train = y[:N]; y_test = y[N:] #Label
    return x_train, x_test, y_train, y_test, RandomForestClassifier(n_estimators =
forest_size, criterion='gini',
                                        max_depth=None, min_samples_split=2,
min_samples_leaf=1,
                                        min_weight_fraction_leaf=0.0,
max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True,
                                        oob_score=False, n_jobs=None,
random_state=None, verbose=0, warm_start=False, class_weight=None,
                                        ccp_alpha=0.0, max_samples=None)

#Input: Feature Training Set, Label Training Set, Random Forest Object, Number of trees
in Random Forest Object
#Processing: Plots Random Forest Object, displays, and stores into .png file
#Output: Displayed Random Forest Object, and .png Forest file
def Print_Forest(x_train, y_train, clf, forest_size):
    fn=x_train
    cn=y_train
```

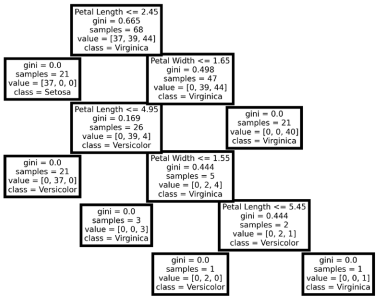
```
    fig, axes = plt.subplots(nrows = 1,ncols = forest_size,figsize = (10,2), dpi=1000)
    if forest_size == 1:
        tree.plot_tree(clf.estimators_[0], max_depth=None, feature_names=["Sepal
Length", "Sepal Width", "Petal Length", "Petal Width"], class_names=["Setosa",
"Versicolor", "Virginica"], label='all',
                       filled=False, impurity=True, node_ids=False, proportion=False,
rounded=False,
                       precision=3, ax=None, fontsize=None);
        axes.set_title('Estimator: ' + str(0), fontsize = 11)
    else:
        for index in range(0, forest_size):
            tree.plot_tree(clf.estimators_[index], max_depth=None,
feature_names=["Sepal Length", "Sepal Width", "Petal Length", "Petal Width"],
class_names=["Setosa", "Versicolor", "Virginica"], label='all',
                       filled=False, impurity=True, node_ids=False, proportion=False,
rounded=False,
                       precision=3, ax=axes[index], fontsize=None);
            axes[index].set_title('Estimator: ' + str(index), fontsize = 11)
    fig.savefig('rf_trees.png')

#Input: Random Forest Object,Label Test Set,Feature Test Set
#Processing: Creates labeled Confusion Matrix for Random Forest Object
#Output: Confusion Matrix for Random Forest Object
def Print_Confusion_Matrix(clf, x_test, y_test):
    length = len(x_test)
    y_pred = []
    for index in range(0, length):
        y_pred.append(clf.predict(x_test[[index]]))
    cm = confusion_matrix(y_test, y_pred, labels=clf.classes_, normalize = 'all')
    disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=["Setosa",
"Versicolor", "Virginica"])
    disp.plot(cmap=plt.cm.Blues)
    plt.show()

#Input: x_train as feature training set, y_train as label training set as ground truth,
clf as Random Forest Object
#Processing: Creates an array ("y_pred") of predicted labels from x_train, provides
y_pred and y_train to F1 function
#Output: Outputs F1 score for all three classes of Iris' ("Setosa", "Versicolor",
"Virginica") as array
def Print_F1(x_train, y_train, clf):
    length = len(x_train)
    y_pred = []
    for index in range(0, length):
        y_pred.append(clf.predict(x_train[[index]]))
    f1 = f1_score(y_train, y_pred, labels= [0, 1, 2], average=None)
    print("F1 scores generally")
    print(f1)
    print("F1 scores for each class:")
    print("Setosa")
    print(f1[0])
    print("Versicolor")
    print(f1[1])
    print("Virginica")
    print(f1[2])

#Input: x_train as feature training set, y_train as label training set as ground truth,
```

```
    clf as Random Forest Object
    #Processing:Creates an array ("y_pred") of predicted labels from x_train, provides
    y_pred and y_train to Accuracy function
    #Output: Outputs total Accuracy score for all three classes of Iris' ("Setosa",
    "Versicolor", "Virginica") as array
    def Print_Accuracy(x_train, y_train, clf):
        length = len(x_train)
        y_pred = []
        for index in range(0, length):
            y_pred.append(clf.predict(x_train[[index]]))
        accuracy = accuracy_score(y_train, y_pred, normalize=True)
        print("Accuracy is:")
        print(accuracy)

    #Input: x_train as feature training set, y_train as label training set as ground truth,
    clf as Random Forest Object
    #Processing: Creates an array ("y_pred") of predicted labels from x_train, provides
    y_pred and y_train to Accuracy function
    #Output: Outputs a classification report (e.g., F1 Score, Recall, Accuracy) for all
    three classes of Iris' ("Setosa", "Versicolor", "Virginica") as arra
    def Print_Report(x_train, y_train, clf):
        length = len(x_train)
        y_pred = []
        for index in range(0, length):
            y_pred.append(clf.predict(x_train[[index]]))
        target_names = ["Setosa", "Versicolor", "Virginica"]
        print(classification_report(y_train, y_pred, target_names=target_names))

    #Called only initially to prepare data
    Process_DataSet()

    #Processes data, creates and trains Random Forest
    forest_size = int(input("Select number of estimators (trees) (enter 0 to end)"))
    if forest_size == 0:
        sys.exit("Bye Bye!!")

    else:
        x_train, x_test, y_train, y_test, clf = Create_Forest_Object(forest_size)
        clf.fit(x_train, y_train)

    #Print_Forest(x_train, y_train, clf, forest_size)
    #Print_Confusion_Matrix(clf, x_test, y_test)
    #Print_F1(x_train, y_train, clf)
    while(True):
        print ("Select analysis type:")
        print ("1. Enter '1' to print Forest")
        print ("2. Enter '2' to print Confusion Matrix")
        print ("3. Enter '3' to print F1 score")
        print ("4. Enter '4' to print Accuracy")
        print ("5. Enter '5' to print Classification Report")
        print ("6. Enter '6' to End Program")
        menu_selection = int(input("Select Input Options: "))

        if menu_selection == 1:
            Print_Forest(x_train, y_train, clf, forest_size)

        elif menu_selection == 2:
```

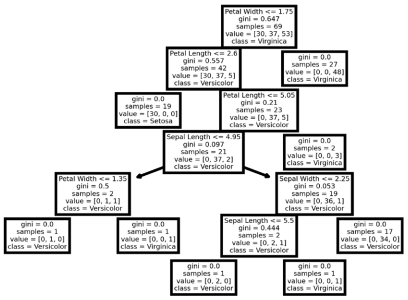
```
            Print_Confusion_Matrix(clf, x_test, y_test)

        elif menu_selection == 3:
            Print_F1(x_train, y_train, clf)

        elif menu_selection == 4:
            Print_Accuracy(x_train, y_train, clf)

        elif menu_selection == 5:
            Print_Report(x_train, y_train, clf)

        elif menu_selection == 6:
            sys.exit("Bye Bye!!")

        else:
            print ("Entered wrong selection, try again")
            print (" ")
```

```
Select number of estimators (trees) (enter 0 to end)3
Select analysis type:
1. Enter '1' to print Forest
2. Enter '2' to print Confusion Matrix
3. Enter '3' to print F1 score
4. Enter '4' to print Accuracy
5. Enter '5' to print Classification Report
6. Enter '6' to End Program
Select Input Options: 1
Select analysis type:
1. Enter '1' to print Forest
2. Enter '2' to print Confusion Matrix
3. Enter '3' to print F1 score
4. Enter '4' to print Accuracy
5. Enter '5' to print Classification Report
6. Enter '6' to End Program
Select Input Options: 2
```

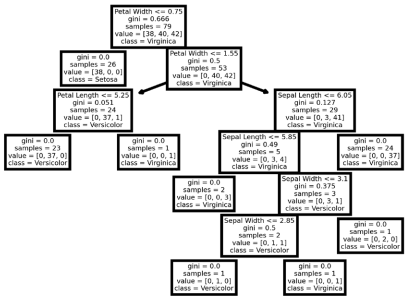
Estimator: 0

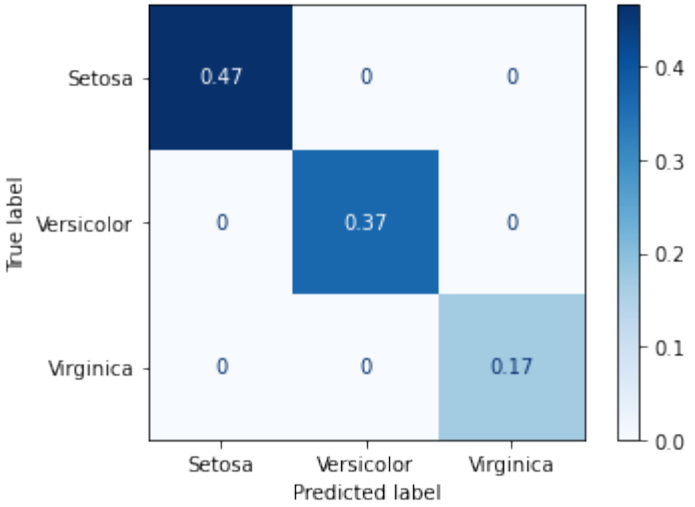



Estimator: 1




Estimator: 2






```
Select analysis type:
1. Enter '1' to print Forest
2. Enter '2' to print Confusion Matrix
3. Enter '3' to print F1 score
4. Enter '4' to print Accuracy
5. Enter '5' to print Classification Report
6. Enter '6' to End Program
Select Input Options: 3
F1 scores generally
[1.         0.97368421 0.97826087]
F1 scores for each class:
Setosa
1.0
Versicolor
0.9736842105263158
Virginica
0.9782608695652174
Select analysis type:
1. Enter '1' to print Forest
2. Enter '2' to print Confusion Matrix
3. Enter '3' to print F1 score
4. Enter '4' to print Accuracy
5. Enter '5' to print Classification Report
6. Enter '6' to End Program
Select Input Options: 4
Accuracy is:
0.9833333333333333
Select analysis type:
1. Enter '1' to print Forest
2. Enter '2' to print Confusion Matrix
3. Enter '3' to print F1 score
4. Enter '4' to print Accuracy
5. Enter '5' to print Classification Report
6. Enter '6' to End Program
Select Input Options: 5
              precision    recall  f1-score   support

      Setosa       1.00      1.00      1.00        36
  Versicolor       1.00      0.95      0.97        39
   Virginica       0.96      1.00      0.98        45
```

```
    accuracy                           0.98       120
   macro avg       0.99      0.98      0.98       120
weighted avg       0.98      0.98      0.98       120

Select analysis type:
1. Enter '1' to print Forest
2. Enter '2' to print Confusion Matrix
3. Enter '3' to print F1 score
4. Enter '4' to print Accuracy
5. Enter '5' to print Classification Report
6. Enter '6' to End Program
```

In [ ]:

In [ ]: