Question

In [ ]:

In [1]:

```
import pandas as pd
import numpy as np
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D  # noqa: F401 unused import
from matplotlib import cm
import seaborn as sns
from sklearn.model_selection import train_test_split
import pdb
```

# #

1. Dataset: Consider a dataset that has both numerical and categorical features. You may use the dataset used by you in the second assignment. It will be good to have 50 or more features and at least 1000 patterns. The number of classes can be 2 or more.
2. Your Tasks: There are three subtasks. For all the subtasks, split the dataset into train and test parts. Do this splitting randomly 10 times and report the average accuracy. You may vary the test and train dataset sizes. The subtasks are: (a) Subtask1: Build a decision tree using the training data. Tune the parameters corresponding to pruning the decision tree. Use the best decision tree to classify the test dataset and obtain the accuracy. (b) Subtask 2: Build a Random Forest Classifier using the training dataset. Vary the size of the random forest by using different number of decision trees. Obtain the classification accuracy on the test data. (c) Subtask3: Use XGBoost classifier to classify the test dataset. Tune any associ- ated parameters. Get the accuracy on the test dataset.
3. Report your results. Provide details on which platform/package is used for each sub- task. Analysis of results is important.

In [2]:

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn import tree
import random

class data_modeling():
```

```python
    def __init__(self):
        pass
    def process_data(self,train_max, test_max):

        from sklearn import preprocessing
        le = preprocessing.LabelEncoder()

        DM = pd.read_csv("s.csv")
        #DM.head()
        #n = np.randint(10)
        n = random.randint(1, 10)
        for i in range(n):
            DM=DM.sample(frac=1)
            DM = DM.reset_index(drop=True)
        #DM.drop(['gender','race/ethnicity','parental level of education', 'lunch','test preparation course'],


        DM.Result = DM.Result.astype(int)

        g = preprocessing.LabelEncoder()
        e = preprocessing.LabelEncoder()
        p = preprocessing.LabelEncoder()
        l = preprocessing.LabelEncoder()
        t = preprocessing.LabelEncoder()
        DM['gender_n'] = g.fit_transform( DM['gender'])
        DM['gender_n'] =  DM['gender_n'] * 10


        DM['ethnicity_n'] = 10 * e.fit_transform( DM['race/ethnicity'])

        DM['parental level of education_n'] = 10 *  p.fit_transform( DM['parental level of education'])
        DM['lunch_n'] = 10 * l.fit_transform( DM['lunch'])
        DM['test preparation course_n']= 10 * t.fit_transform(DM['test preparation course'])
        DM.drop(['gender','race/ethnicity','parental level of education', 'lunch','test preparation course'], a

        DM['Result'] = DM['Result2']
        DM['Result'] = DM.mean(axis=1)/10
        DM['Result'] = DM['Result'].astype(int)
        p = DM['Result']

        DM.reset_index(drop=True)
        Y = DM['Result']
        DM.drop(['Result' , 'Result2'],axis=1, inplace=True)
        X = DM
        #Y = DM['Result']
        print('f FFFF ----->>  {Y}')
```

```
        import sklearn.model_selection as model_selection
        X_train, X_test, y_train, y_test = model_selection.train_test_split(X, Y, train_size=0.75,test_size=0.2

        #return (DM,,DM, )


        return (X_train,y_train,X_test,y_test)
```

In [ ]:

In [3]:
```
print("Hello World!")
dm = data_modeling()


x_train,y_train,x_test,y_test = dm.process_data(5,5)
```

```
Hello World!
f FFFF ----->>  {Y}
```

In [4]:
```
global_accuracy = 0
acc_list = []
global_clf = None
for index in range(1,10):
    dm = data_modeling()

    x_train,y_train,x_test,y_test = dm.process_data(5,5)

    print(__doc__)

    import numpy as np
    import matplotlib.pyplot as plt


    from sklearn.tree import DecisionTreeClassifier, plot_tree

    # Parameters
    n_classes = 3
    plot_colors = "ryb"
    plot_step = 0.05

    for pairidx, pair in enumerate([[0, 1], [0, 2], [0, 3],
                                    [1, 2], [1, 3], [2, 3]]):
```

```
        # We only take the two corresponding features

        X = x_train.iloc[:,pair]
        clf = DecisionTreeClassifier().fit(X, y_train)


        # Plot the decision boundary
        #plt.subplot(2, 3, pairidx + 1)

        x_min, x_max = 37,100 #X[:, 0].min() - 1, X[:, 0].max() + 1
        y_min, y_max =  0,3 # X[:, 1].min() - 1, X[:, 1].max() + 1
        xx, yy = np.meshgrid(np.arange(x_min, x_max, plot_step),
                             np.arange(y_min, y_max, plot_step))
        plt.tight_layout(h_pad=1, w_pad=1, pad=2.5)

        Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
        Z = Z.reshape(xx.shape)
        cs = plt.contourf(xx, yy, Z, cmap=plt.cm.RdYlBu)

    X = x_train.iloc[:,:]
    # Train
    #clf = DecisionTreeClassifier().fit( x_train.iloc[:,:], y_train)
    import graphviz
    from sklearn import tree
    import pydotplus


    clf = DecisionTreeClassifier().fit( X, y_train)

    path = clf.cost_complexity_pruning_path(X, y_train)

    path

    dot_data = tree.export_graphviz(clf, node_ids = True,
                                    proportion = True,
                                    feature_names = list(X.columns.values.tolist()) ,
                                    class_names = ['6','7'],
                                    filled = True,
                                    rounded = True)

    import graphviz
    gvz_graph = graphviz.Source(dot_data)
    gvz_graph
    gvz_graph.render('dtree_render_' + str(index), view=True)

    path = clf.cost_complexity_pruning_path(X, y_train)
```

```
path
ccp_alphas, impurities = path.ccp_alphas, path.impurities

plt.figure(figsize=(10, 6))
plt.plot(ccp_alphas, impurities)
plt.xlabel("effective alpha")
plt.ylabel("total impurity of leaves")

ccp_alphas, impurities = path.ccp_alphas, path.impurities

plt.figure(figsize=(10, 6))
plt.plot(ccp_alphas, impurities)
plt.xlabel("effective alpha")
plt.ylabel("total impurity of leaves")


clfs = []

for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(random_state=0, ccp_alpha=ccp_alpha)
    clf.fit(X, y_train)
    clfs.append(clf)
    #print(f'clfs {clfs}')

tree_depths = [clf.tree_.max_depth for clf in clfs]
plt.figure(figsize=(10,  6))
plt.plot(ccp_alphas[:-1], tree_depths[:-1])
plt.title("Decision Tree Model")
plt.xlabel("effective alpha")
plt.ylabel("total depth")

from sklearn.metrics import accuracy_score

print(f' x len {len(x_test)}')
print(f' y len {len(y_test)}')


acc_scores = [accuracy_score(y_test, clf.predict(x_test)) for clf in clfs]

clf_ = None
acc = 0
for clf in clfs:
    a = accuracy_score(y_test, clf.predict(x_test))
    print(f'accuracy score per decision tree-------> {a}')

    if (a > acc) :
```

```python
            acc = a
            clf_ = clf

    #acc = accuracy_score(y_test, clf.predict(x_test))
    #acc_list.append(acc)
    #print(f'accuracy score per decision tree-------> {acc}')
    if acc > global_accuracy:
        global_accuracy = acc
        global_clf = clf_


    #print(f'acc score {acc_scores}')
    #tree_depths = [clf.tree_.max_depth for clf in clfs]
#acc_scores = acc_list

for scored in acc_scores:
    print(f'accuracy score per decision tree-------> {acc}')

plt.figure(figsize=(10,  6))
plt.grid()
plt.plot(acc_scores[:-1], acc_scores[:-1])
plt.title("Decision Tree Model")
plt.xlabel("Accuracy scores")
plt.ylabel("Accuracy scores")
print(f" Decision Tree accuracy = {accuracy_score(y_test, global_clf.predict(x_test))}")
```

```
f FFFF ----->>  {Y}
Automatically created module for IPython interactive environment
 x len 250
 y len 250
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.596
accuracy score per decision tree-------> 0.596
accuracy score per decision tree-------> 0.592
accuracy score per decision tree-------> 0.592
accuracy score per decision tree-------> 0.592
accuracy score per decision tree-------> 0.592
accuracy score per decision tree-------> 0.592
accuracy score per decision tree-------> 0.592
accuracy score per decision tree-------> 0.592
accuracy score per decision tree-------> 0.596
accuracy score per decision tree-------> 0.596
accuracy score per decision tree-------> 0.596
accuracy score per decision tree-------> 0.596
accuracy score per decision tree-------> 0.596
accuracy score per decision tree-------> 0.596
```

```
accuracy score per decision tree-------> 0.596
accuracy score per decision tree-------> 0.596
accuracy score per decision tree-------> 0.596
accuracy score per decision tree-------> 0.592
accuracy score per decision tree-------> 0.592
accuracy score per decision tree-------> 0.592
accuracy score per decision tree-------> 0.588
accuracy score per decision tree-------> 0.592
accuracy score per decision tree-------> 0.604
accuracy score per decision tree-------> 0.604
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.604
accuracy score per decision tree-------> 0.604
accuracy score per decision tree-------> 0.604
accuracy score per decision tree-------> 0.604
accuracy score per decision tree-------> 0.604
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.604
accuracy score per decision tree-------> 0.604
accuracy score per decision tree-------> 0.604
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.596
accuracy score per decision tree-------> 0.592
accuracy score per decision tree-------> 0.596
accuracy score per decision tree-------> 0.596
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.608
accuracy score per decision tree-------> 0.616
accuracy score per decision tree-------> 0.628
accuracy score per decision tree-------> 0.64
accuracy score per decision tree-------> 0.644
accuracy score per decision tree-------> 0.64
accuracy score per decision tree-------> 0.636
accuracy score per decision tree-------> 0.636
accuracy score per decision tree-------> 0.648
accuracy score per decision tree-------> 0.644
accuracy score per decision tree-------> 0.656
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.652
accuracy score per decision tree-------> 0.656
accuracy score per decision tree-------> 0.664
accuracy score per decision tree-------> 0.664
accuracy score per decision tree-------> 0.64
accuracy score per decision tree-------> 0.64
accuracy score per decision tree-------> 0.64
accuracy score per decision tree-------> 0.624
```

```
f FFFF ----->>  {Y}
Automatically created module for IPython interactive environment
 x len 250
 y len 250
accuracy score per decision tree-------> 0.604
accuracy score per decision tree-------> 0.604
accuracy score per decision tree-------> 0.604
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.588
accuracy score per decision tree-------> 0.588
accuracy score per decision tree-------> 0.588
accuracy score per decision tree-------> 0.588
accuracy score per decision tree-------> 0.584
accuracy score per decision tree-------> 0.584
accuracy score per decision tree-------> 0.584
accuracy score per decision tree-------> 0.584
accuracy score per decision tree-------> 0.58
accuracy score per decision tree-------> 0.58
accuracy score per decision tree-------> 0.58
accuracy score per decision tree-------> 0.58
accuracy score per decision tree-------> 0.584
accuracy score per decision tree-------> 0.584
accuracy score per decision tree-------> 0.584
accuracy score per decision tree-------> 0.584
accuracy score per decision tree-------> 0.584
accuracy score per decision tree-------> 0.58
accuracy score per decision tree-------> 0.584
accuracy score per decision tree-------> 0.58
accuracy score per decision tree-------> 0.576
accuracy score per decision tree-------> 0.576
accuracy score per decision tree-------> 0.576
accuracy score per decision tree-------> 0.576
accuracy score per decision tree-------> 0.58
accuracy score per decision tree-------> 0.58
accuracy score per decision tree-------> 0.588
accuracy score per decision tree-------> 0.588
accuracy score per decision tree-------> 0.584
accuracy score per decision tree-------> 0.584
accuracy score per decision tree-------> 0.604
accuracy score per decision tree-------> 0.604
accuracy score per decision tree-------> 0.604
accuracy score per decision tree-------> 0.604
accuracy score per decision tree-------> 0.604
accuracy score per decision tree-------> 0.608
accuracy score per decision tree-------> 0.608
```

```
accuracy score per decision tree-------> 0.612
accuracy score per decision tree-------> 0.612
accuracy score per decision tree-------> 0.612
accuracy score per decision tree-------> 0.616
accuracy score per decision tree-------> 0.632
accuracy score per decision tree-------> 0.636
accuracy score per decision tree-------> 0.644
accuracy score per decision tree-------> 0.652
accuracy score per decision tree-------> 0.652
accuracy score per decision tree-------> 0.652
accuracy score per decision tree-------> 0.652
accuracy score per decision tree-------> 0.644
accuracy score per decision tree-------> 0.652
accuracy score per decision tree-------> 0.652
accuracy score per decision tree-------> 0.636
accuracy score per decision tree-------> 0.588
accuracy score per decision tree-------> 0.576
accuracy score per decision tree-------> 0.576
accuracy score per decision tree-------> 0.592
accuracy score per decision tree-------> 0.612
accuracy score per decision tree-------> 0.636
accuracy score per decision tree-------> 0.704
accuracy score per decision tree-------> 0.672
accuracy score per decision tree-------> 0.672
accuracy score per decision tree-------> 0.648
f FFFF ----->>  {Y}
Automatically created module for IPython interactive environment
 x len 250
 y len 250
accuracy score per decision tree-------> 0.592
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.608
accuracy score per decision tree-------> 0.608
accuracy score per decision tree-------> 0.608
accuracy score per decision tree-------> 0.608
accuracy score per decision tree-------> 0.608
accuracy score per decision tree-------> 0.608
accuracy score per decision tree-------> 0.612
accuracy score per decision tree-------> 0.612
accuracy score per decision tree-------> 0.612
accuracy score per decision tree-------> 0.612
accuracy score per decision tree-------> 0.612
accuracy score per decision tree-------> 0.612
accuracy score per decision tree-------> 0.612
accuracy score per decision tree-------> 0.612
```

```
accuracy score per decision tree-------> 0.612
accuracy score per decision tree-------> 0.612
accuracy score per decision tree-------> 0.616
accuracy score per decision tree-------> 0.616
accuracy score per decision tree-------> 0.616
accuracy score per decision tree-------> 0.616
accuracy score per decision tree-------> 0.616
accuracy score per decision tree-------> 0.62
accuracy score per decision tree-------> 0.616
accuracy score per decision tree-------> 0.612
accuracy score per decision tree-------> 0.612
accuracy score per decision tree-------> 0.612
accuracy score per decision tree-------> 0.608
accuracy score per decision tree-------> 0.608
accuracy score per decision tree-------> 0.608
accuracy score per decision tree-------> 0.608
accuracy score per decision tree-------> 0.612
accuracy score per decision tree-------> 0.612
accuracy score per decision tree-------> 0.616
accuracy score per decision tree-------> 0.612
accuracy score per decision tree-------> 0.604
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.592
accuracy score per decision tree-------> 0.596
accuracy score per decision tree-------> 0.596
accuracy score per decision tree-------> 0.596
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.604
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.592
accuracy score per decision tree-------> 0.584
accuracy score per decision tree-------> 0.588
accuracy score per decision tree-------> 0.596
accuracy score per decision tree-------> 0.596
accuracy score per decision tree-------> 0.596
accuracy score per decision tree-------> 0.604
accuracy score per decision tree-------> 0.596
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.624
accuracy score per decision tree-------> 0.624
accuracy score per decision tree-------> 0.624
accuracy score per decision tree-------> 0.624
accuracy score per decision tree-------> 0.62
accuracy score per decision tree-------> 0.632
accuracy score per decision tree-------> 0.632
accuracy score per decision tree-------> 0.62
accuracy score per decision tree-------> 0.616
accuracy score per decision tree-------> 0.66
```

```
accuracy score per decision tree-------> 0.66
f FFFF ----->>  {Y}
Automatically created module for IPython interactive environment
 x len 250
 y len 250
accuracy score per decision tree-------> 0.536
accuracy score per decision tree-------> 0.536
accuracy score per decision tree-------> 0.536
accuracy score per decision tree-------> 0.54
accuracy score per decision tree-------> 0.544
accuracy score per decision tree-------> 0.544
accuracy score per decision tree-------> 0.544
accuracy score per decision tree-------> 0.544
accuracy score per decision tree-------> 0.564
accuracy score per decision tree-------> 0.564
accuracy score per decision tree-------> 0.564
accuracy score per decision tree-------> 0.564
accuracy score per decision tree-------> 0.564
accuracy score per decision tree-------> 0.564
accuracy score per decision tree-------> 0.564
accuracy score per decision tree-------> 0.564
accuracy score per decision tree-------> 0.564
accuracy score per decision tree-------> 0.56
accuracy score per decision tree-------> 0.56
accuracy score per decision tree-------> 0.564
accuracy score per decision tree-------> 0.564
accuracy score per decision tree-------> 0.568
accuracy score per decision tree-------> 0.576
accuracy score per decision tree-------> 0.588
accuracy score per decision tree-------> 0.592
accuracy score per decision tree-------> 0.592
accuracy score per decision tree-------> 0.592
accuracy score per decision tree-------> 0.592
accuracy score per decision tree-------> 0.592
accuracy score per decision tree-------> 0.592
accuracy score per decision tree-------> 0.592
accuracy score per decision tree-------> 0.596
accuracy score per decision tree-------> 0.596
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.596
accuracy score per decision tree-------> 0.596
accuracy score per decision tree-------> 0.588
accuracy score per decision tree-------> 0.588
accuracy score per decision tree-------> 0.584
accuracy score per decision tree-------> 0.592
accuracy score per decision tree-------> 0.592
accuracy score per decision tree-------> 0.584
accuracy score per decision tree-------> 0.584
```

```
accuracy score per decision tree-------> 0.584
accuracy score per decision tree-------> 0.588
accuracy score per decision tree-------> 0.588
accuracy score per decision tree-------> 0.584
accuracy score per decision tree-------> 0.584
accuracy score per decision tree-------> 0.588
accuracy score per decision tree-------> 0.576
accuracy score per decision tree-------> 0.572
accuracy score per decision tree-------> 0.572
accuracy score per decision tree-------> 0.608
accuracy score per decision tree-------> 0.608
accuracy score per decision tree-------> 0.616
accuracy score per decision tree-------> 0.616
accuracy score per decision tree-------> 0.632
accuracy score per decision tree-------> 0.632
accuracy score per decision tree-------> 0.636
accuracy score per decision tree-------> 0.628
f FFFF ----->>  {Y}
Automatically created module for IPython interactive environment
 x len 250
 y len 250
accuracy score per decision tree-------> 0.572
accuracy score per decision tree-------> 0.572
accuracy score per decision tree-------> 0.572
accuracy score per decision tree-------> 0.572
accuracy score per decision tree-------> 0.572
accuracy score per decision tree-------> 0.572
accuracy score per decision tree-------> 0.572
accuracy score per decision tree-------> 0.572
accuracy score per decision tree-------> 0.568
accuracy score per decision tree-------> 0.568
accuracy score per decision tree-------> 0.568
accuracy score per decision tree-------> 0.568
accuracy score per decision tree-------> 0.572
accuracy score per decision tree-------> 0.572
accuracy score per decision tree-------> 0.568
accuracy score per decision tree-------> 0.568
accuracy score per decision tree-------> 0.568
accuracy score per decision tree-------> 0.576
accuracy score per decision tree-------> 0.576
accuracy score per decision tree-------> 0.576
accuracy score per decision tree-------> 0.576
accuracy score per decision tree-------> 0.572
accuracy score per decision tree-------> 0.568
accuracy score per decision tree-------> 0.572
accuracy score per decision tree-------> 0.576
accuracy score per decision tree-------> 0.576
accuracy score per decision tree-------> 0.58
accuracy score per decision tree-------> 0.576
```

```
accuracy score per decision tree-------> 0.564
accuracy score per decision tree-------> 0.564
accuracy score per decision tree-------> 0.568
accuracy score per decision tree-------> 0.572
accuracy score per decision tree-------> 0.576
accuracy score per decision tree-------> 0.568
accuracy score per decision tree-------> 0.56
accuracy score per decision tree-------> 0.564
accuracy score per decision tree-------> 0.572
accuracy score per decision tree-------> 0.572
accuracy score per decision tree-------> 0.572
accuracy score per decision tree-------> 0.584
accuracy score per decision tree-------> 0.592
accuracy score per decision tree-------> 0.588
accuracy score per decision tree-------> 0.592
accuracy score per decision tree-------> 0.592
accuracy score per decision tree-------> 0.592
accuracy score per decision tree-------> 0.592
accuracy score per decision tree-------> 0.584
accuracy score per decision tree-------> 0.592
accuracy score per decision tree-------> 0.596
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.612
accuracy score per decision tree-------> 0.604
accuracy score per decision tree-------> 0.604
accuracy score per decision tree-------> 0.596
accuracy score per decision tree-------> 0.596
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.608
accuracy score per decision tree-------> 0.604
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.612
accuracy score per decision tree-------> 0.608
accuracy score per decision tree-------> 0.604
accuracy score per decision tree-------> 0.588
accuracy score per decision tree-------> 0.596
accuracy score per decision tree-------> 0.592
accuracy score per decision tree-------> 0.62
accuracy score per decision tree-------> 0.632
accuracy score per decision tree-------> 0.632
accuracy score per decision tree-------> 0.632
accuracy score per decision tree-------> 0.664
accuracy score per decision tree-------> 0.664
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.64
f FFFF ----->>  {Y}
Automatically created module for IPython interactive environment
```

```
 x len 250
 y len 250
accuracy score per decision tree-------> 0.588
accuracy score per decision tree-------> 0.588
accuracy score per decision tree-------> 0.584
accuracy score per decision tree-------> 0.584
accuracy score per decision tree-------> 0.584
accuracy score per decision tree-------> 0.596
accuracy score per decision tree-------> 0.596
accuracy score per decision tree-------> 0.596
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.596
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.608
accuracy score per decision tree-------> 0.612
accuracy score per decision tree-------> 0.616
accuracy score per decision tree-------> 0.616
accuracy score per decision tree-------> 0.616
accuracy score per decision tree-------> 0.616
accuracy score per decision tree-------> 0.612
accuracy score per decision tree-------> 0.612
accuracy score per decision tree-------> 0.616
accuracy score per decision tree-------> 0.616
accuracy score per decision tree-------> 0.62
accuracy score per decision tree-------> 0.62
accuracy score per decision tree-------> 0.624
accuracy score per decision tree-------> 0.62
accuracy score per decision tree-------> 0.62
accuracy score per decision tree-------> 0.62
accuracy score per decision tree-------> 0.612
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.604
accuracy score per decision tree-------> 0.604
accuracy score per decision tree-------> 0.608
accuracy score per decision tree-------> 0.608
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.604
accuracy score per decision tree-------> 0.604
accuracy score per decision tree-------> 0.616
accuracy score per decision tree-------> 0.624
accuracy score per decision tree-------> 0.62
accuracy score per decision tree-------> 0.608
```

```
accuracy score per decision tree-------> 0.608
accuracy score per decision tree-------> 0.608
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.588
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.608
accuracy score per decision tree-------> 0.616
accuracy score per decision tree-------> 0.616
accuracy score per decision tree-------> 0.624
accuracy score per decision tree-------> 0.624
accuracy score per decision tree-------> 0.652
accuracy score per decision tree-------> 0.656
accuracy score per decision tree-------> 0.616
accuracy score per decision tree-------> 0.588
accuracy score per decision tree-------> 0.62
accuracy score per decision tree-------> 0.64
f FFFF ----->>  {Y}
Automatically created module for IPython interactive environment
<ipython-input-4-d0fbdd5cf8f1>:80: RuntimeWarning: More than 20 figures have been opened. Figures created throu
gh the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly closed and may consume too m
uch memory. (To control this warning, see the rcParam `figure.max_open_warning`).
  plt.figure(figsize=(10, 6))
 x len 250
 y len 250
accuracy score per decision tree-------> 0.568
accuracy score per decision tree-------> 0.564
accuracy score per decision tree-------> 0.564
accuracy score per decision tree-------> 0.564
accuracy score per decision tree-------> 0.56
accuracy score per decision tree-------> 0.56
accuracy score per decision tree-------> 0.56
accuracy score per decision tree-------> 0.56
accuracy score per decision tree-------> 0.56
accuracy score per decision tree-------> 0.556
accuracy score per decision tree-------> 0.556
accuracy score per decision tree-------> 0.556
accuracy score per decision tree-------> 0.556
accuracy score per decision tree-------> 0.556
accuracy score per decision tree-------> 0.56
accuracy score per decision tree-------> 0.56
accuracy score per decision tree-------> 0.56
accuracy score per decision tree-------> 0.556
accuracy score per decision tree-------> 0.556
accuracy score per decision tree-------> 0.556
accuracy score per decision tree-------> 0.556
accuracy score per decision tree-------> 0.56
accuracy score per decision tree-------> 0.564
```

```
accuracy score per decision tree-------> 0.564
accuracy score per decision tree-------> 0.564
accuracy score per decision tree-------> 0.568
accuracy score per decision tree-------> 0.568
accuracy score per decision tree-------> 0.568
accuracy score per decision tree-------> 0.568
accuracy score per decision tree-------> 0.568
accuracy score per decision tree-------> 0.572
accuracy score per decision tree-------> 0.572
accuracy score per decision tree-------> 0.58
accuracy score per decision tree-------> 0.58
accuracy score per decision tree-------> 0.58
accuracy score per decision tree-------> 0.588
accuracy score per decision tree-------> 0.588
accuracy score per decision tree-------> 0.592
accuracy score per decision tree-------> 0.592
accuracy score per decision tree-------> 0.596
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.592
accuracy score per decision tree-------> 0.6
accuracy score per decision tree-------> 0.604
accuracy score per decision tree-------> 0.608
accuracy score per decision tree-------> 0.608
accuracy score per decision tree-------> 0.62
accuracy score per decision tree-------> 0.628
accuracy score per decision tree-------> 0.624
accuracy score per decision tree-------> 0.624
accuracy score per decision tree-------> 0.628
accuracy score per decision tree-------> 0.64
accuracy score per decision tree-------> 0.644
accuracy score per decision tree-------> 0.656
accuracy score per decision tree-------> 0.664
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.656
accuracy score per decision tree-------> 0.648
accuracy score per decision tree-------> 0.648
accuracy score per decision tree-------> 0.652
accuracy score per decision tree-------> 0.652
accuracy score per decision tree-------> 0.644
accuracy score per decision tree-------> 0.64
accuracy score per decision tree-------> 0.652
accuracy score per decision tree-------> 0.652
accuracy score per decision tree-------> 0.676
accuracy score per decision tree-------> 0.676
accuracy score per decision tree-------> 0.676
accuracy score per decision tree-------> 0.688
accuracy score per decision tree-------> 0.624
```

```
accuracy score per decision tree-------> 0.624
accuracy score per decision tree-------> 0.652
f FFFF ----->>  {Y}
Automatically created module for IPython interactive environment
 x len 250
 y len 250
accuracy score per decision tree-------> 0.636
accuracy score per decision tree-------> 0.636
accuracy score per decision tree-------> 0.636
accuracy score per decision tree-------> 0.636
accuracy score per decision tree-------> 0.632
accuracy score per decision tree-------> 0.632
accuracy score per decision tree-------> 0.632
accuracy score per decision tree-------> 0.628
accuracy score per decision tree-------> 0.628
accuracy score per decision tree-------> 0.628
accuracy score per decision tree-------> 0.628
accuracy score per decision tree-------> 0.628
accuracy score per decision tree-------> 0.628
accuracy score per decision tree-------> 0.624
accuracy score per decision tree-------> 0.624
accuracy score per decision tree-------> 0.624
accuracy score per decision tree-------> 0.624
accuracy score per decision tree-------> 0.624
accuracy score per decision tree-------> 0.624
accuracy score per decision tree-------> 0.62
accuracy score per decision tree-------> 0.62
accuracy score per decision tree-------> 0.628
accuracy score per decision tree-------> 0.632
accuracy score per decision tree-------> 0.64
accuracy score per decision tree-------> 0.64
accuracy score per decision tree-------> 0.64
accuracy score per decision tree-------> 0.64
accuracy score per decision tree-------> 0.64
accuracy score per decision tree-------> 0.636
accuracy score per decision tree-------> 0.636
accuracy score per decision tree-------> 0.636
accuracy score per decision tree-------> 0.632
accuracy score per decision tree-------> 0.64
accuracy score per decision tree-------> 0.644
accuracy score per decision tree-------> 0.648
accuracy score per decision tree-------> 0.652
accuracy score per decision tree-------> 0.652
accuracy score per decision tree-------> 0.652
accuracy score per decision tree-------> 0.652
accuracy score per decision tree-------> 0.652
accuracy score per decision tree-------> 0.652
accuracy score per decision tree-------> 0.656
accuracy score per decision tree-------> 0.656
```

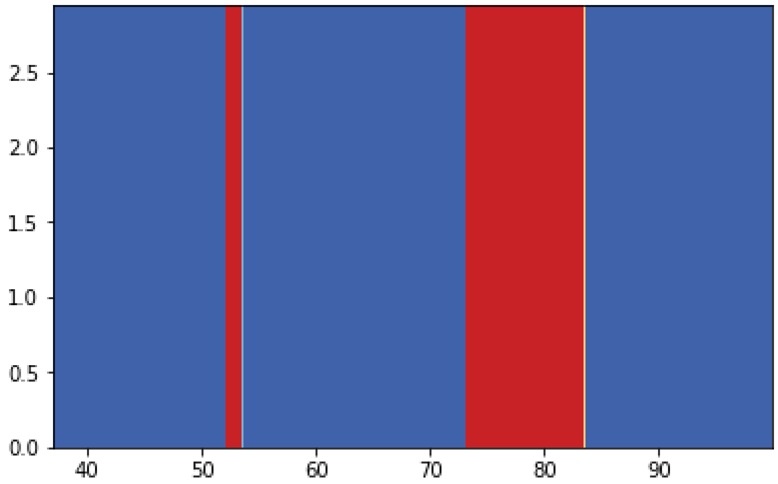
```
accuracy score per decision tree-------> 0.656
accuracy score per decision tree-------> 0.652
accuracy score per decision tree-------> 0.648
accuracy score per decision tree-------> 0.648
accuracy score per decision tree-------> 0.652
accuracy score per decision tree-------> 0.652
accuracy score per decision tree-------> 0.648
accuracy score per decision tree-------> 0.632
accuracy score per decision tree-------> 0.632
accuracy score per decision tree-------> 0.636
accuracy score per decision tree-------> 0.636
accuracy score per decision tree-------> 0.632
accuracy score per decision tree-------> 0.636
accuracy score per decision tree-------> 0.644
accuracy score per decision tree-------> 0.648
accuracy score per decision tree-------> 0.648
accuracy score per decision tree-------> 0.68
accuracy score per decision tree-------> 0.684
accuracy score per decision tree-------> 0.684
accuracy score per decision tree-------> 0.688
accuracy score per decision tree-------> 0.688
accuracy score per decision tree-------> 0.688
accuracy score per decision tree-------> 0.7
accuracy score per decision tree-------> 0.704
accuracy score per decision tree-------> 0.708
accuracy score per decision tree-------> 0.704
accuracy score per decision tree-------> 0.7
accuracy score per decision tree-------> 0.7
accuracy score per decision tree-------> 0.712
accuracy score per decision tree-------> 0.72
accuracy score per decision tree-------> 0.728
accuracy score per decision tree-------> 0.628
accuracy score per decision tree-------> 0.628
accuracy score per decision tree-------> 0.624
accuracy score per decision tree-------> 0.712
accuracy score per decision tree-------> 0.712
accuracy score per decision tree-------> 0.692
accuracy score per decision tree-------> 0.692
f FFFF ----->>  {Y}
Automatically created module for IPython interactive environment
 x len 250
 y len 250
accuracy score per decision tree-------> 0.632
accuracy score per decision tree-------> 0.632
accuracy score per decision tree-------> 0.632
accuracy score per decision tree-------> 0.632
accuracy score per decision tree-------> 0.636
accuracy score per decision tree-------> 0.636
accuracy score per decision tree-------> 0.636
```

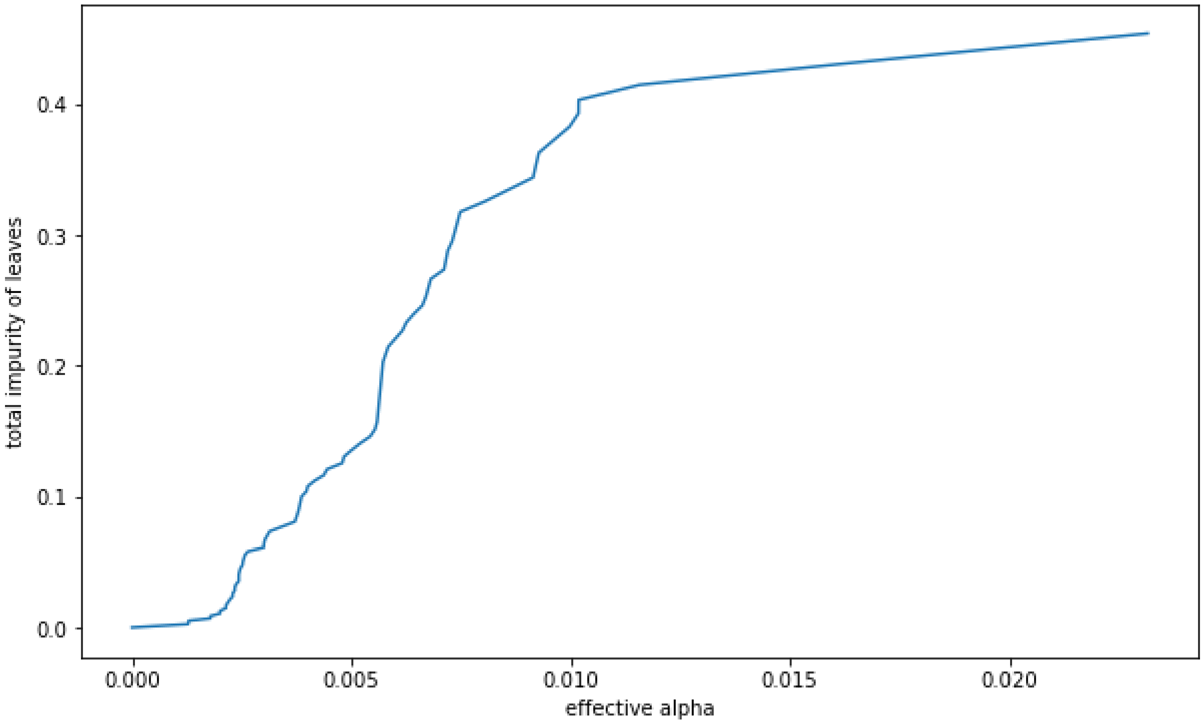
```
accuracy score per decision tree-------> 0.644
accuracy score per decision tree-------> 0.644
accuracy score per decision tree-------> 0.644
accuracy score per decision tree-------> 0.644
accuracy score per decision tree-------> 0.64
accuracy score per decision tree-------> 0.64
accuracy score per decision tree-------> 0.64
accuracy score per decision tree-------> 0.64
accuracy score per decision tree-------> 0.64
accuracy score per decision tree-------> 0.64
accuracy score per decision tree-------> 0.636
accuracy score per decision tree-------> 0.632
accuracy score per decision tree-------> 0.632
accuracy score per decision tree-------> 0.636
accuracy score per decision tree-------> 0.648
accuracy score per decision tree-------> 0.656
accuracy score per decision tree-------> 0.656
accuracy score per decision tree-------> 0.652
accuracy score per decision tree-------> 0.648
accuracy score per decision tree-------> 0.652
accuracy score per decision tree-------> 0.64
accuracy score per decision tree-------> 0.64
accuracy score per decision tree-------> 0.64
accuracy score per decision tree-------> 0.64
accuracy score per decision tree-------> 0.64
accuracy score per decision tree-------> 0.632
accuracy score per decision tree-------> 0.624
accuracy score per decision tree-------> 0.628
accuracy score per decision tree-------> 0.636
accuracy score per decision tree-------> 0.636
accuracy score per decision tree-------> 0.62
accuracy score per decision tree-------> 0.62
accuracy score per decision tree-------> 0.62
accuracy score per decision tree-------> 0.62
accuracy score per decision tree-------> 0.62
accuracy score per decision tree-------> 0.636
accuracy score per decision tree-------> 0.64
accuracy score per decision tree-------> 0.64
accuracy score per decision tree-------> 0.636
accuracy score per decision tree-------> 0.632
accuracy score per decision tree-------> 0.632
accuracy score per decision tree-------> 0.636
accuracy score per decision tree-------> 0.632
accuracy score per decision tree-------> 0.636
accuracy score per decision tree-------> 0.644
accuracy score per decision tree-------> 0.64
accuracy score per decision tree-------> 0.64
accuracy score per decision tree-------> 0.644
accuracy score per decision tree-------> 0.652
```

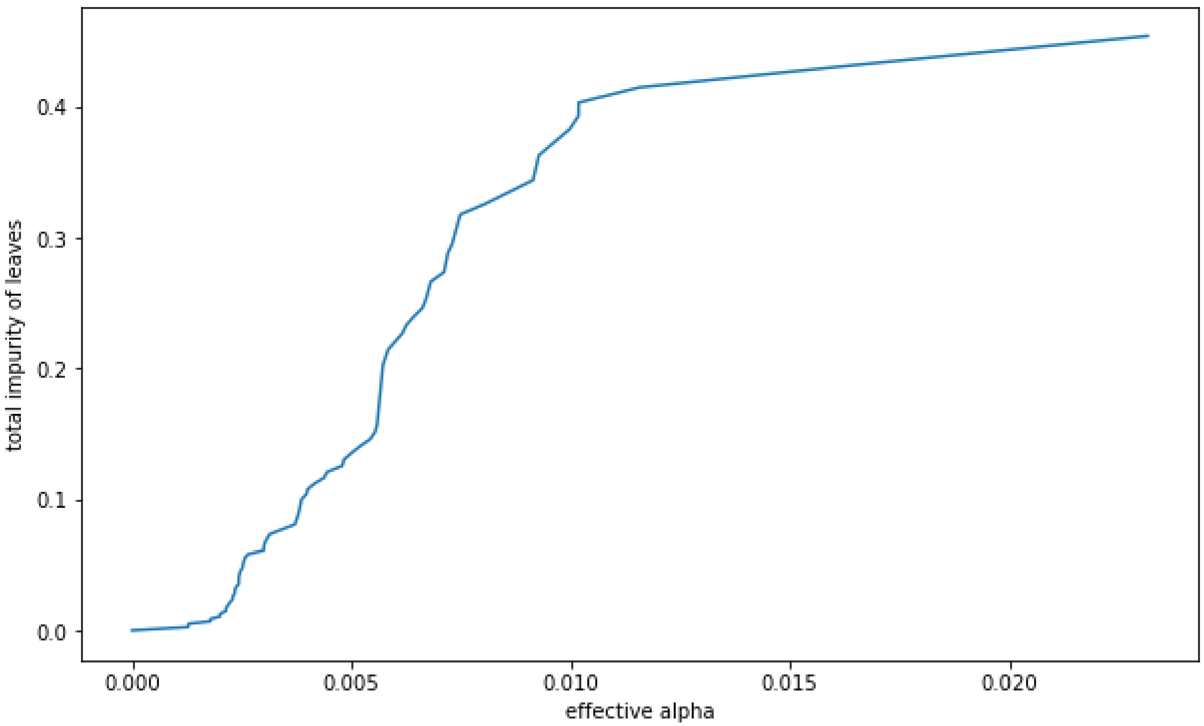
```
accuracy score per decision tree-------> 0.636
accuracy score per decision tree-------> 0.64
accuracy score per decision tree-------> 0.636
accuracy score per decision tree-------> 0.632
accuracy score per decision tree-------> 0.64
accuracy score per decision tree-------> 0.636
accuracy score per decision tree-------> 0.636
accuracy score per decision tree-------> 0.628
accuracy score per decision tree-------> 0.636
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.656
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.648
accuracy score per decision tree-------> 0.644
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.636
accuracy score per decision tree-------> 0.636
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
```

```
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
accuracy score per decision tree-------> 0.66
```

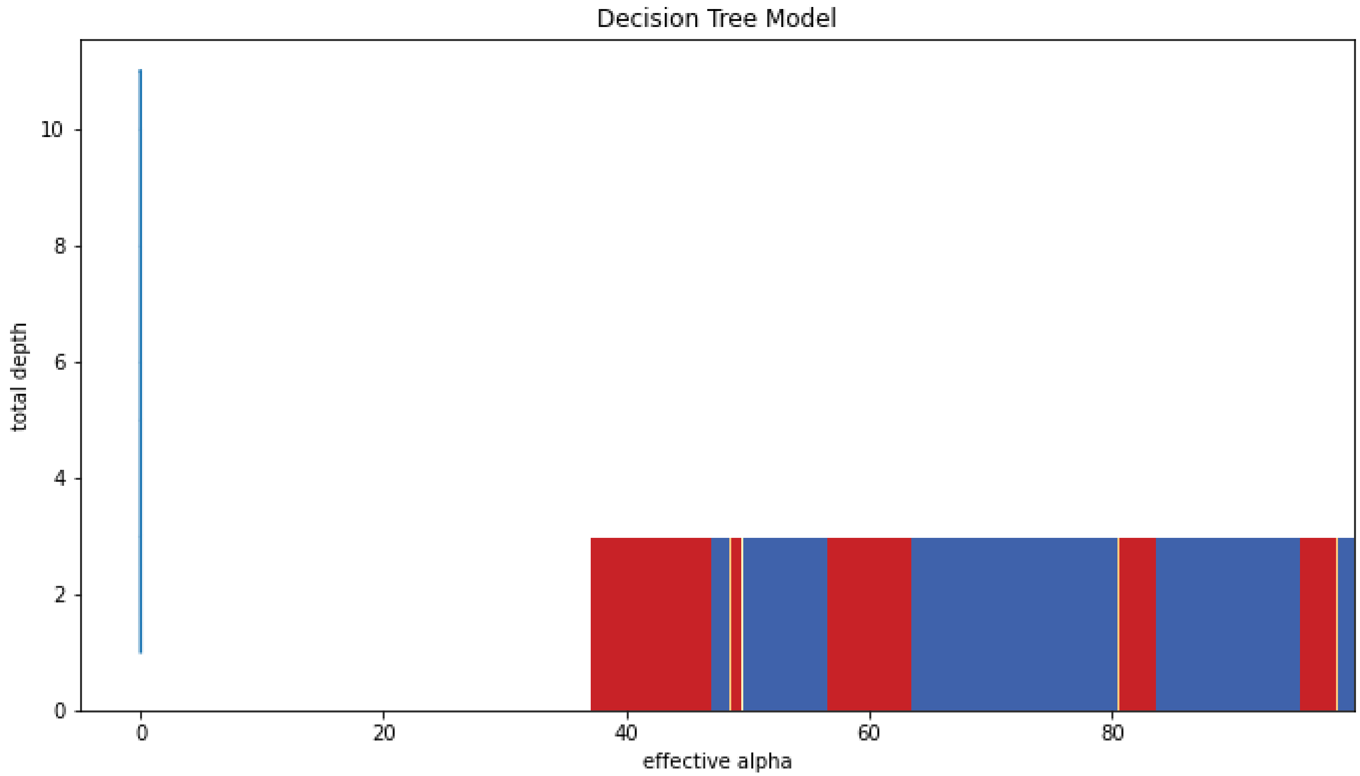
```
accuracy score per decision tree-------> 0.66
 Decision Tree accuracy = 0.716
```

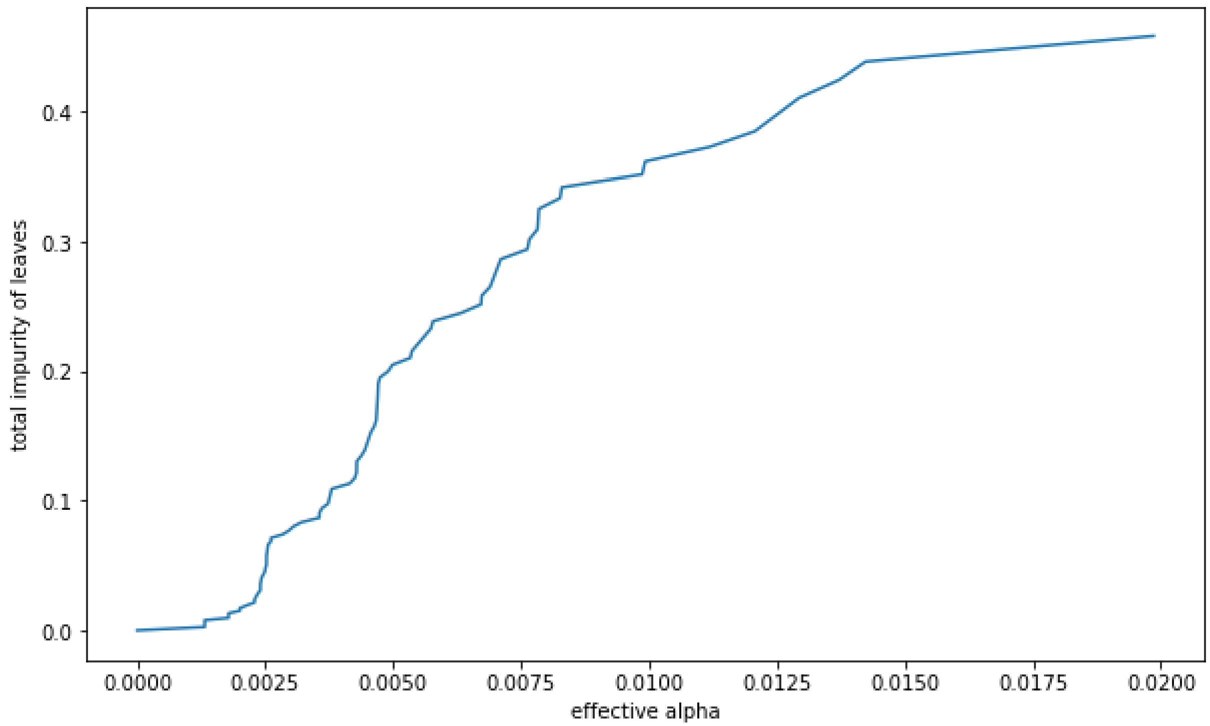




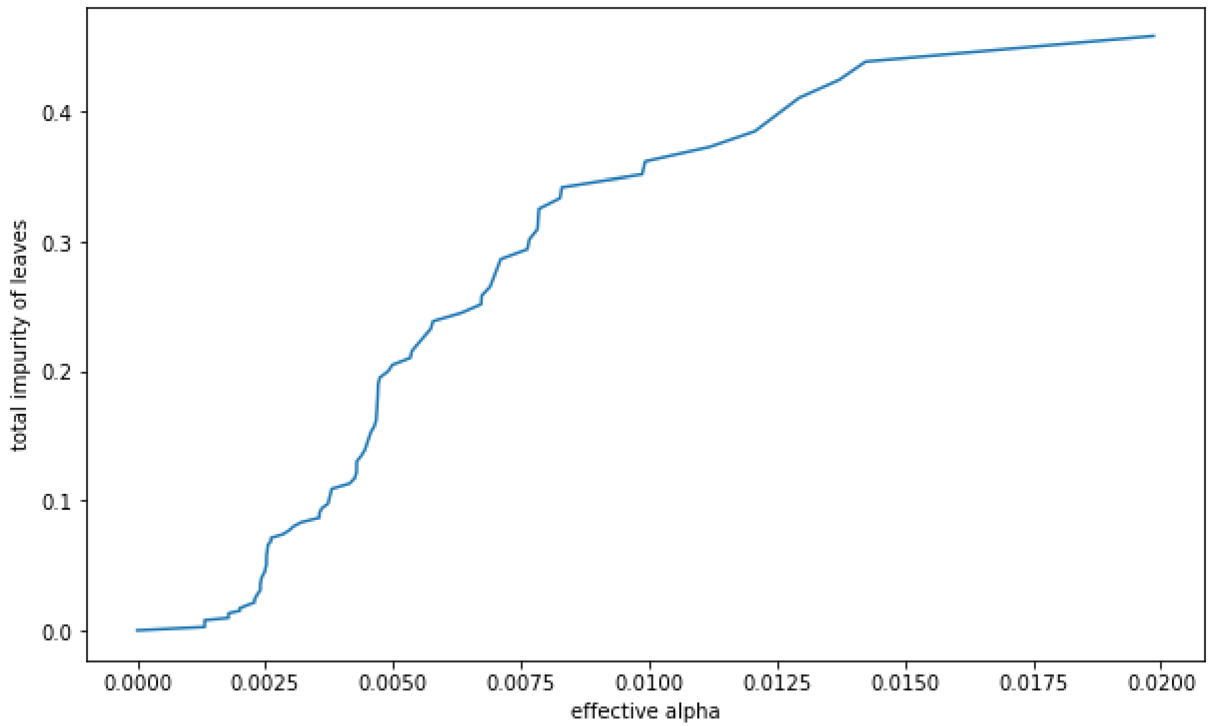


total impurity of leaves
0.0
0.1
0.2
0.3
0.4
0.000
0.005
0.010
0.015
0.020
effective alpha

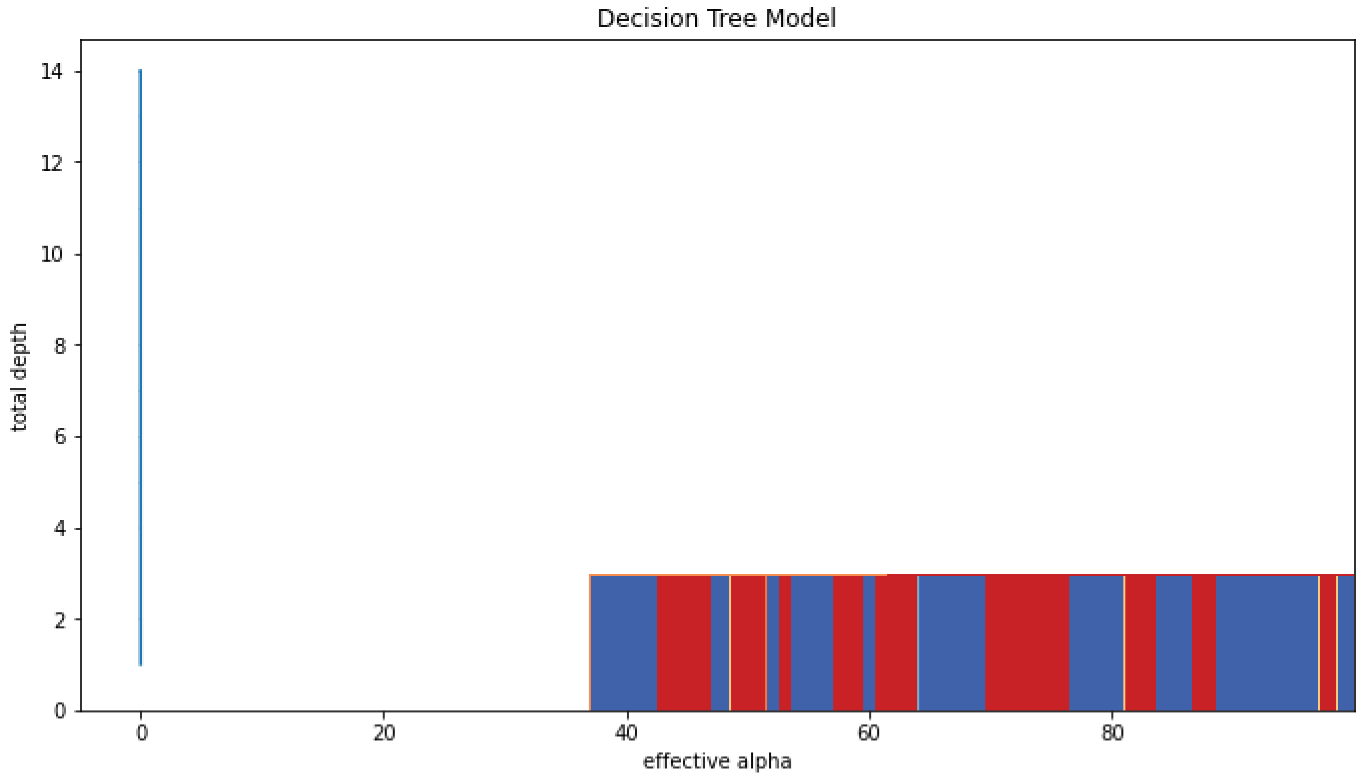

Decision Tree Model
total depth
effective alpha
0
2
4
6
8
10
0
20
40
60
80

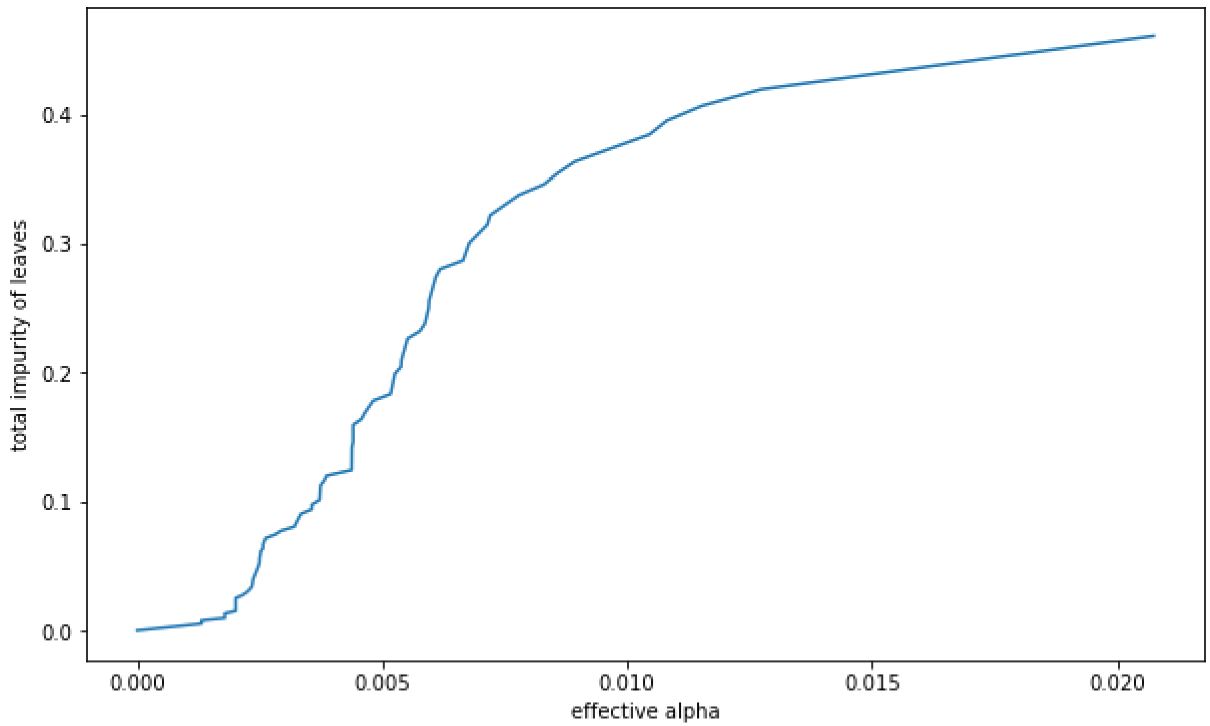


total impurity of leaves
0.0
0.1
0.2
0.3
0.4
0.0000
0.0025
0.0050
0.0075
0.0100
0.0125
0.0150
0.0175
0.0200
effective alpha

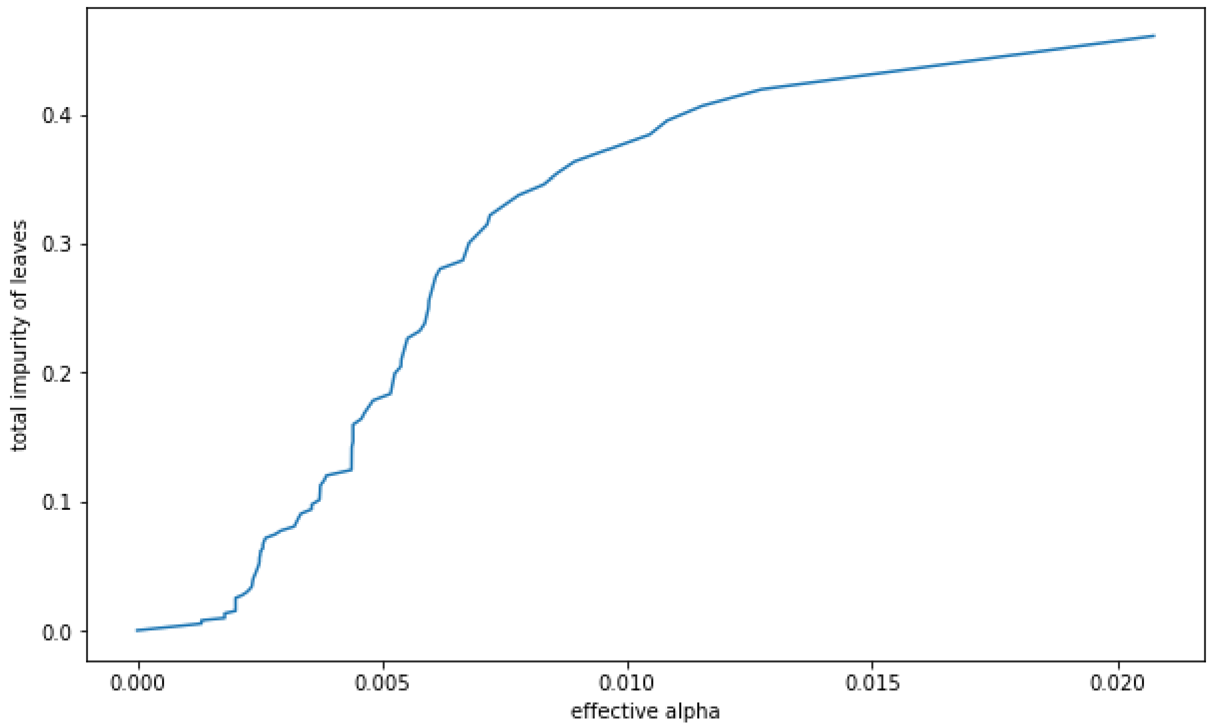


0.4
0.3
0.2
0.1
0.0
total impurity of leaves
0.0000
0.0025
0.0050
0.0075
0.0100
0.0125
0.0150
0.0175
0.0200
effective alpha

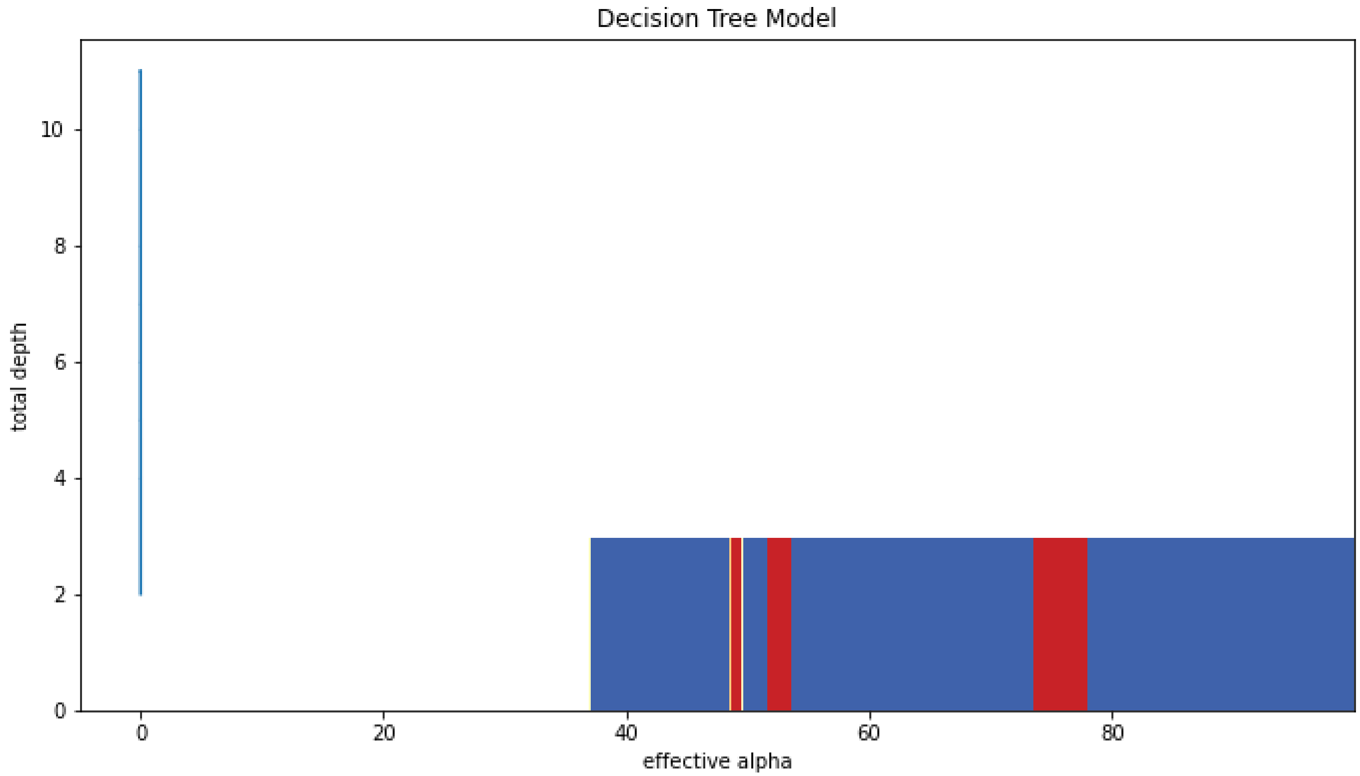

Decision Tree Model
total depth
effective alpha
0
2
4
6
8
10
12
14
0
20
40
60
80

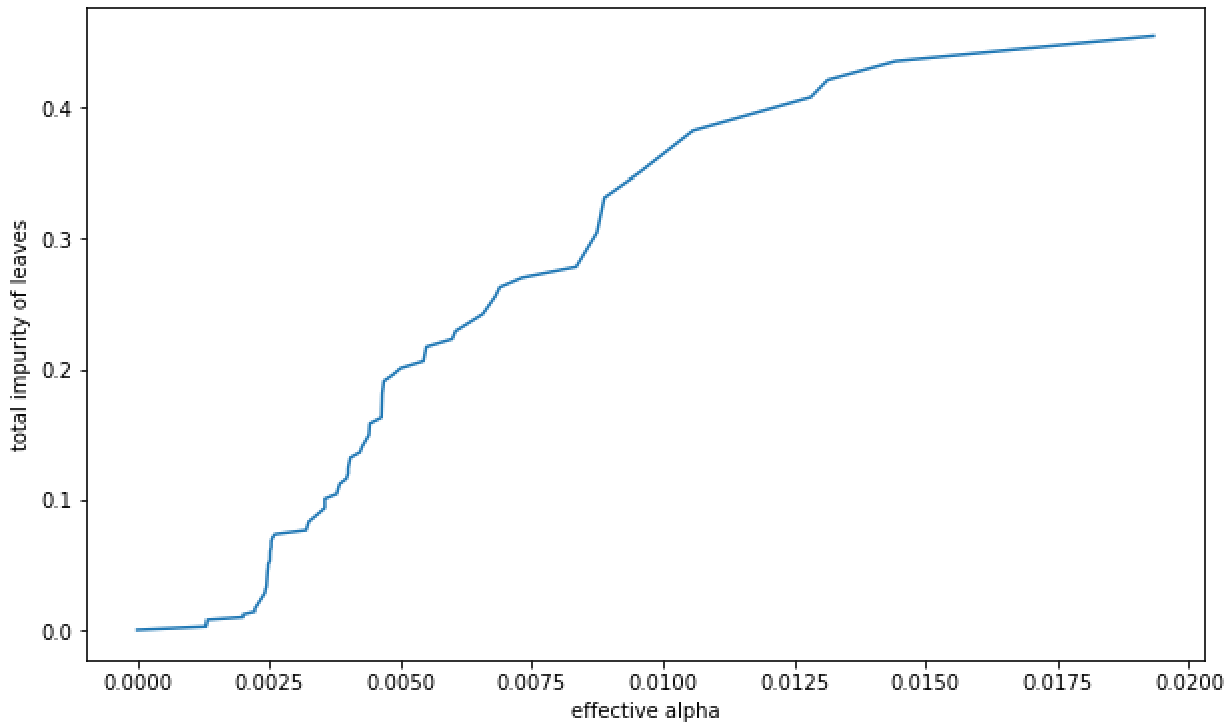


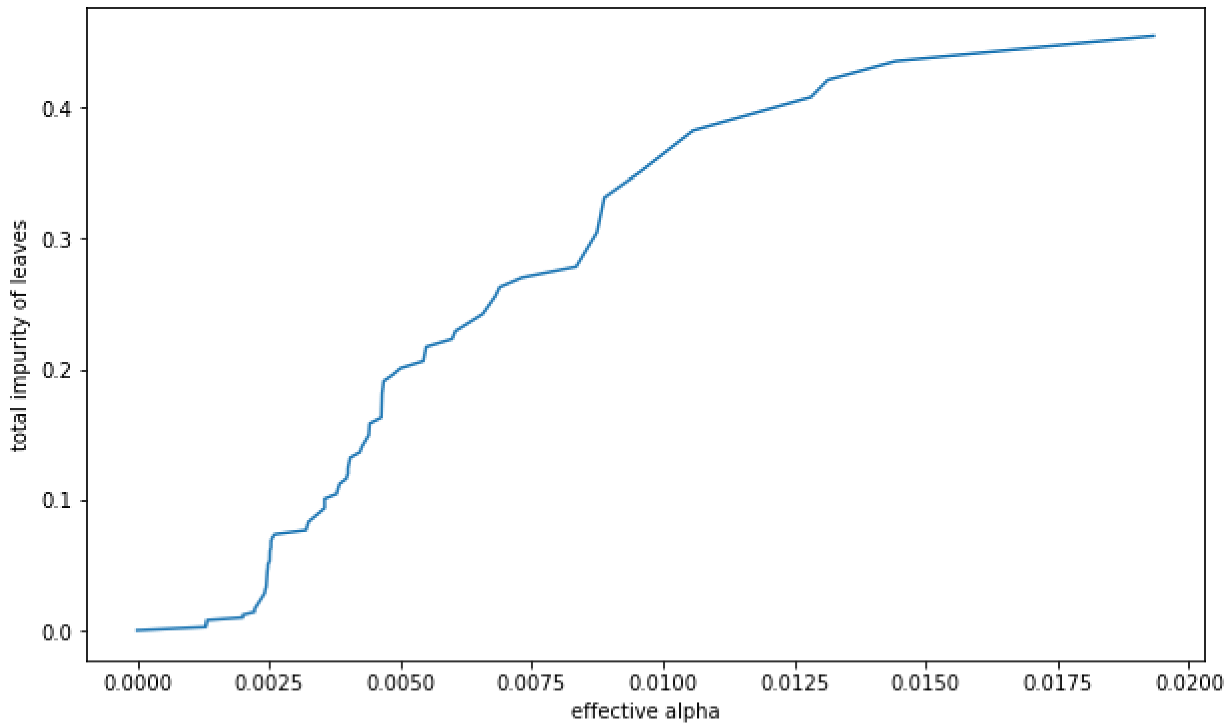
total impurity of leaves
0.0
0.1
0.2
0.3
0.4
0.000
0.005
0.010
0.015
0.020
effective alpha



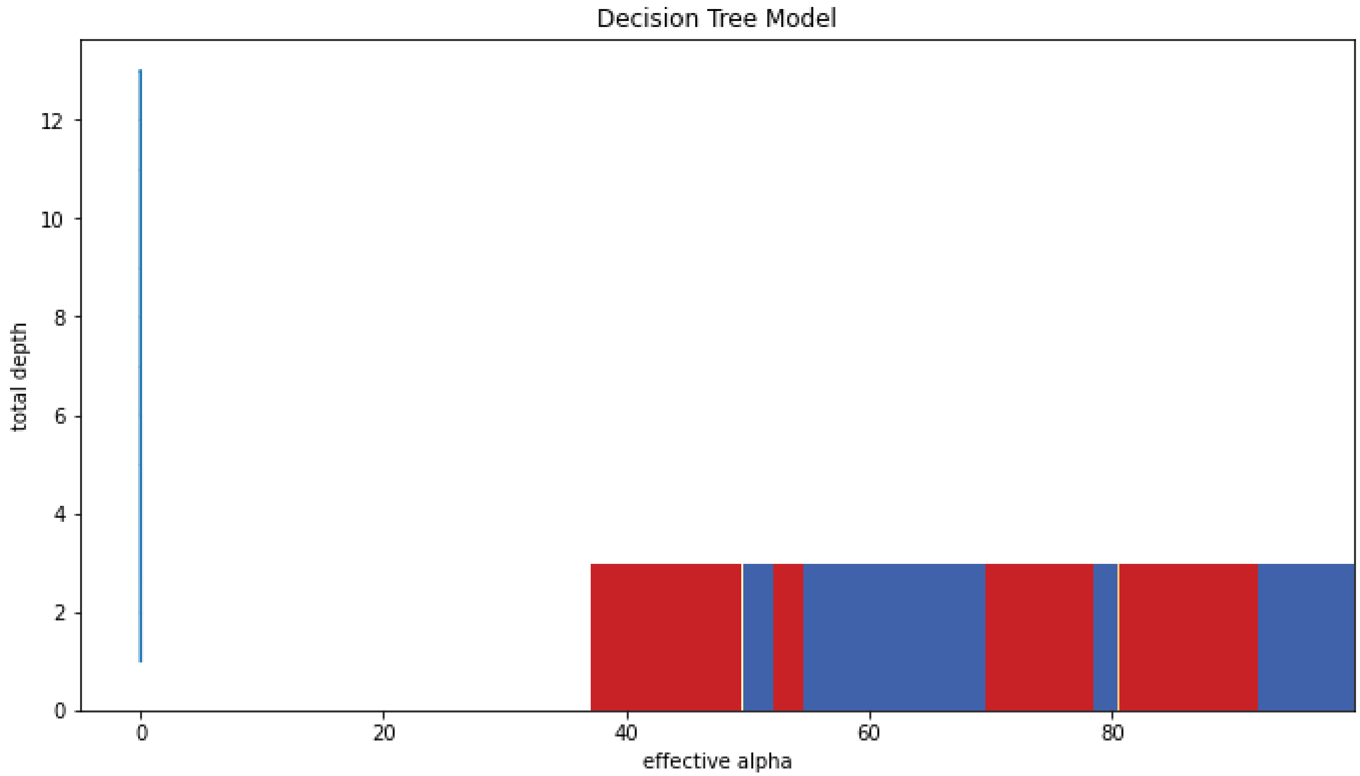
0.4
0.3
0.2
0.1
0.0
total impurity of leaves
0.000
0.005
0.010
0.015
0.020
effective alpha

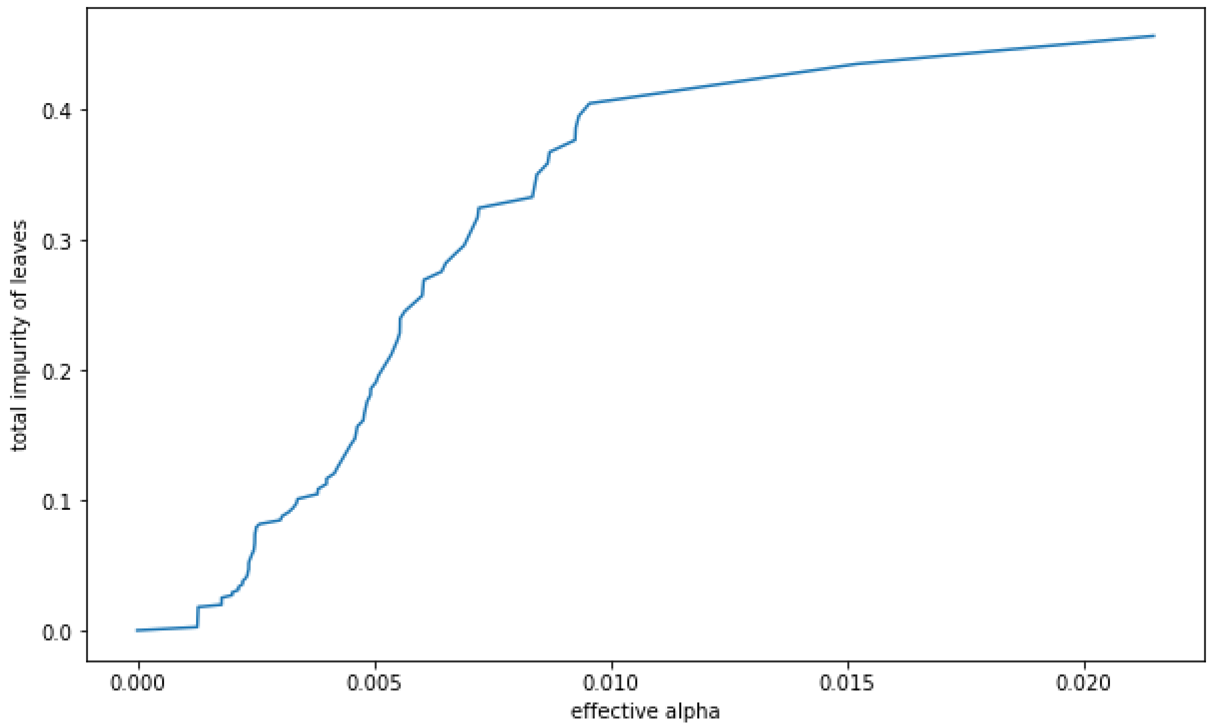

Decision Tree Model
total depth
effective alpha
0
2
4
6
8
10
0
20
40
60
80


total impurity of leaves
0.0
0.1
0.2
0.3
0.4
0.0000
0.0025
0.0050
0.0075
0.0100
0.0125
0.0150
0.0175
0.0200
effective alpha

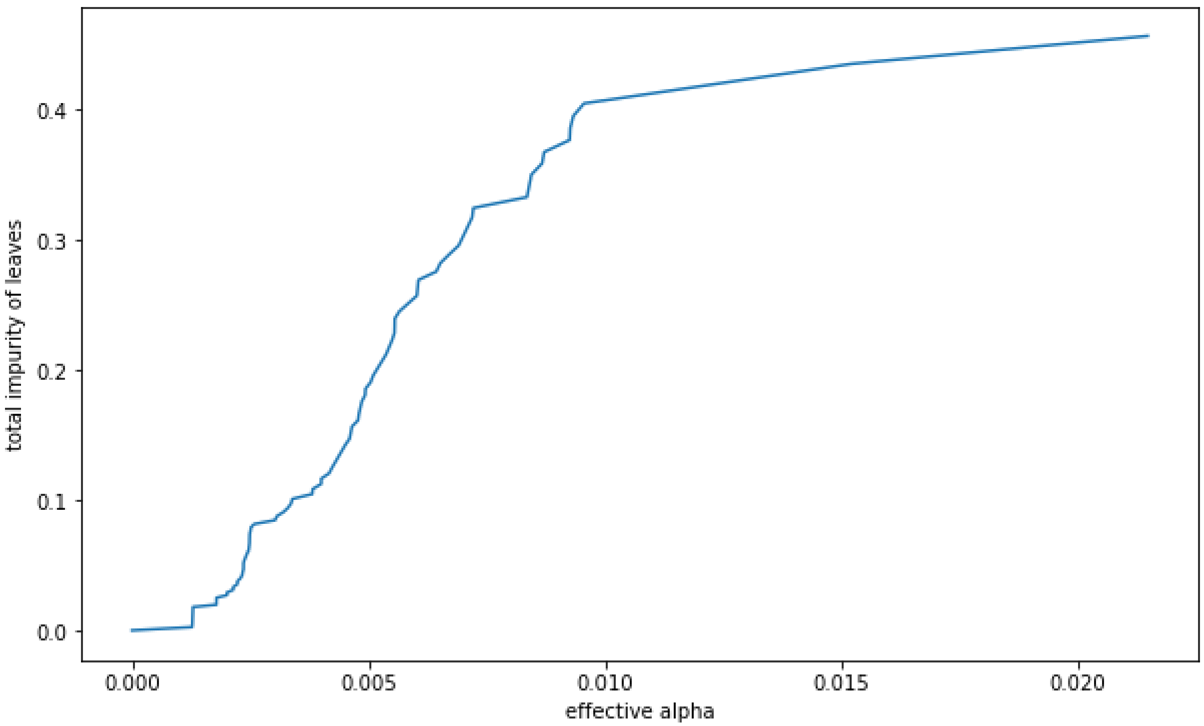


total impurity of leaves
0.0
0.1
0.2
0.3
0.4
0.0000
0.0025
0.0050
0.0075
0.0100
0.0125
0.0150
0.0175
0.0200
effective alpha

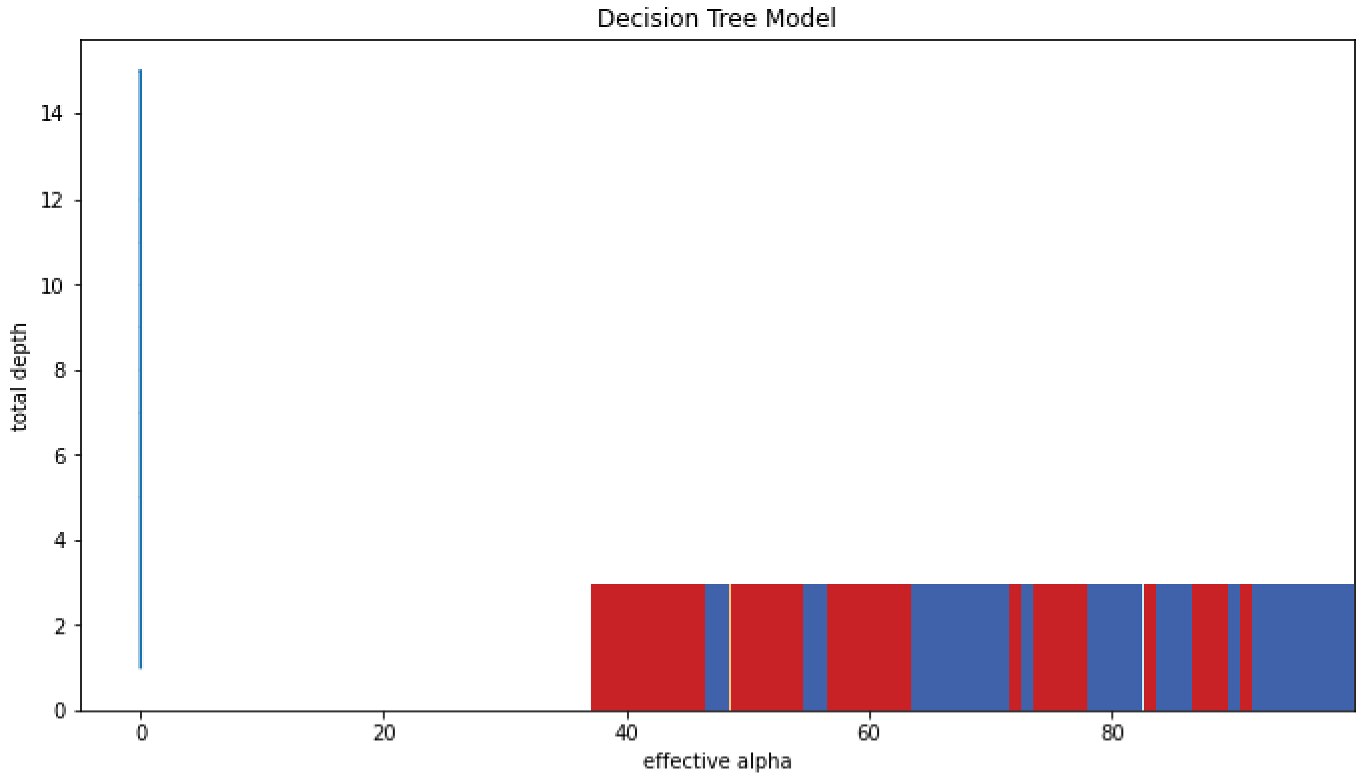

Decision Tree Model
total depth
effective alpha
0
2
4
6
8
10
12
0
20
40
60
80

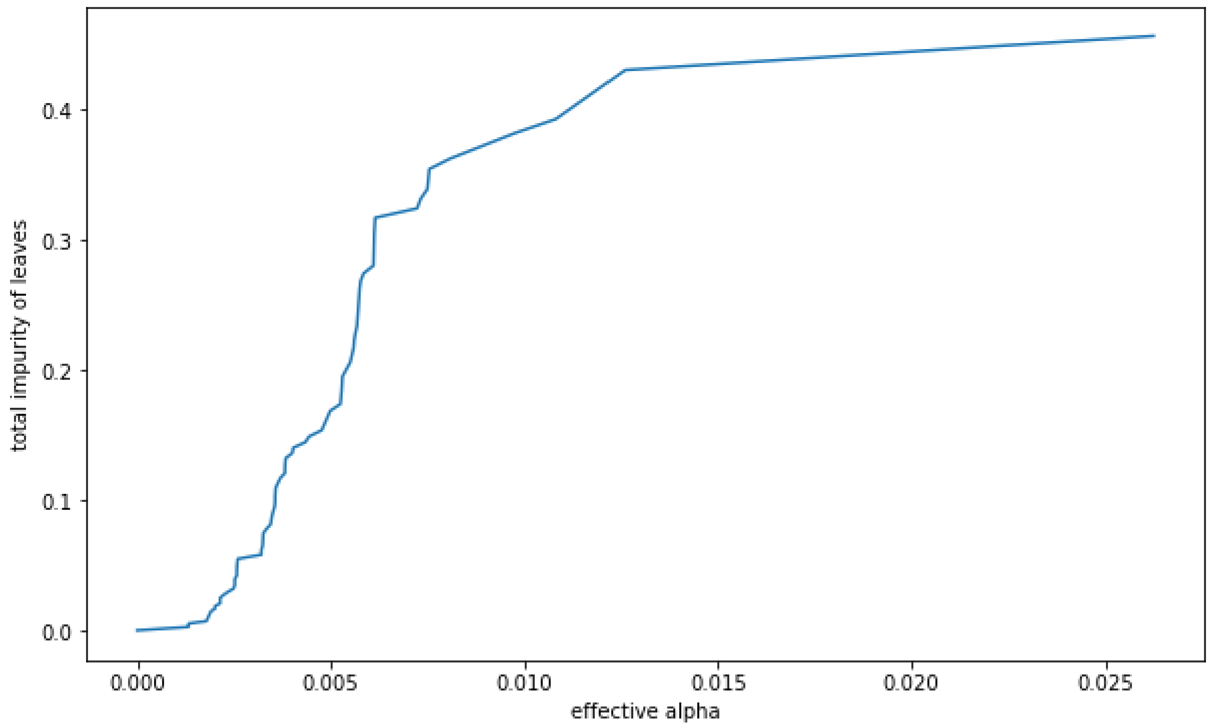


0.4
0.3
0.2
0.1
0.0
total impurity of leaves
0.000
0.005
0.010
0.015
0.020
effective alpha

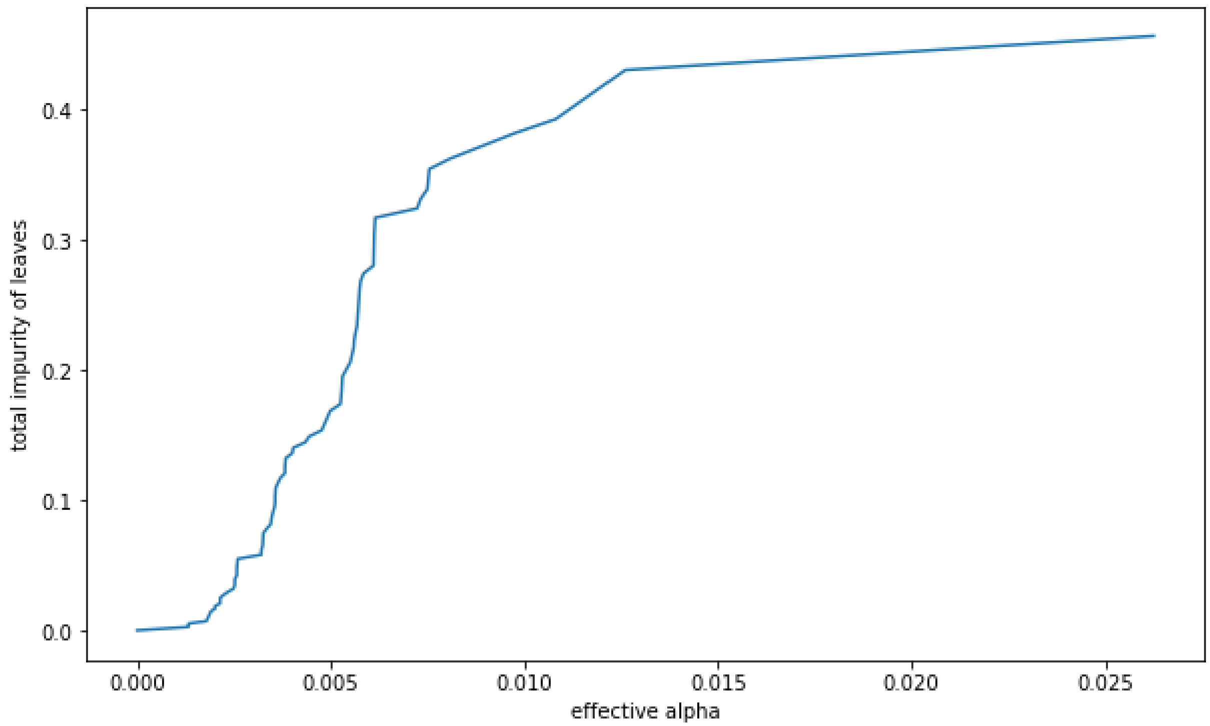


0.4
0.3
0.2
0.1
0.0
total impurity of leaves
0.000
0.005
0.010
0.015
0.020
effective alpha

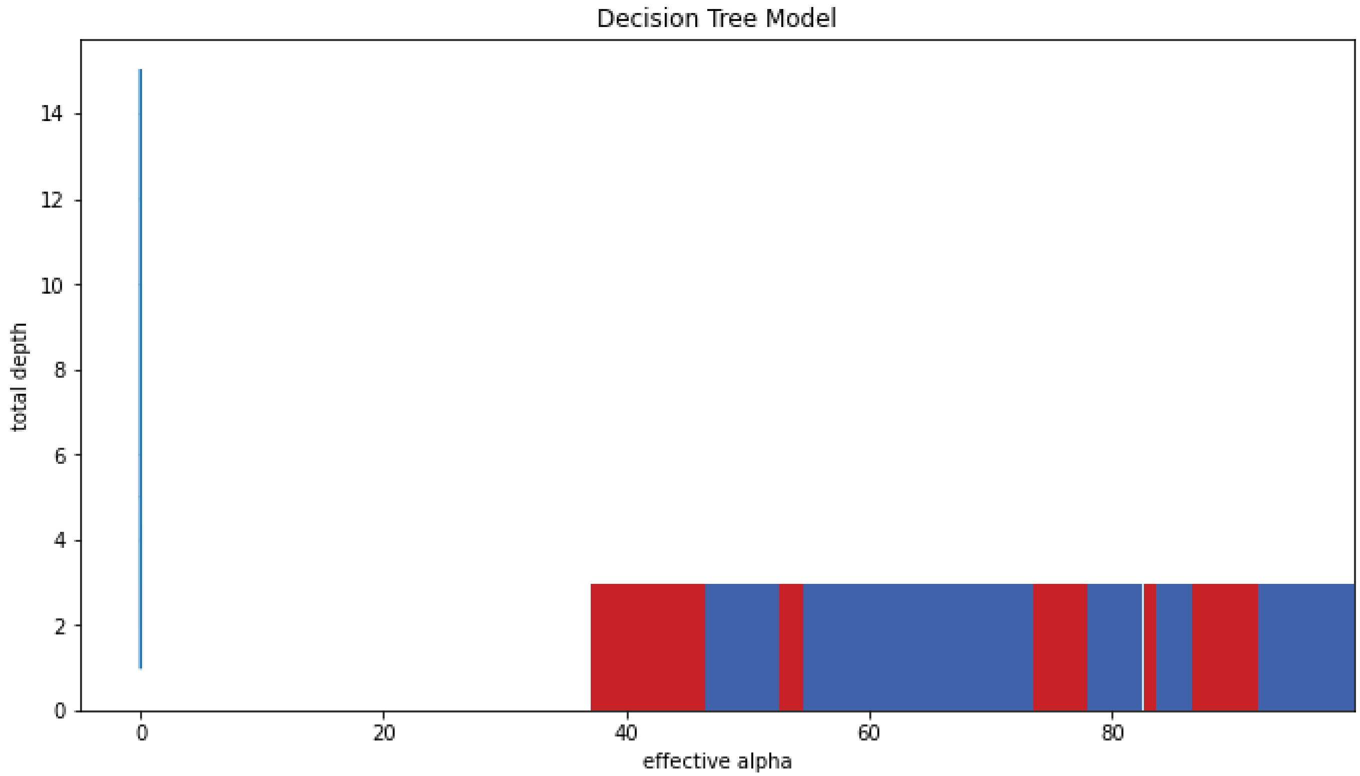

Decision Tree Model
total depth
effective alpha
0
20
40
60
80
0
2
4
6
8
10
12
14



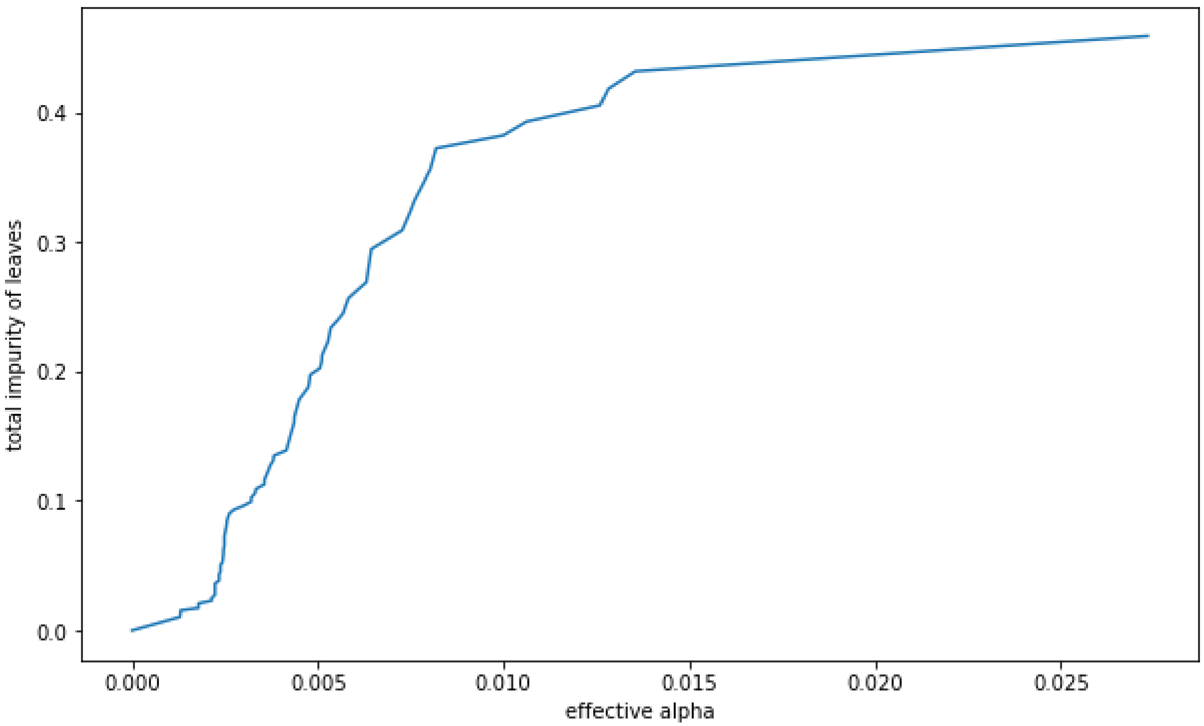
total impurity of leaves
0.0
0.1
0.2
0.3
0.4
0.000
0.005
0.010
0.015
0.020
0.025
effective alpha



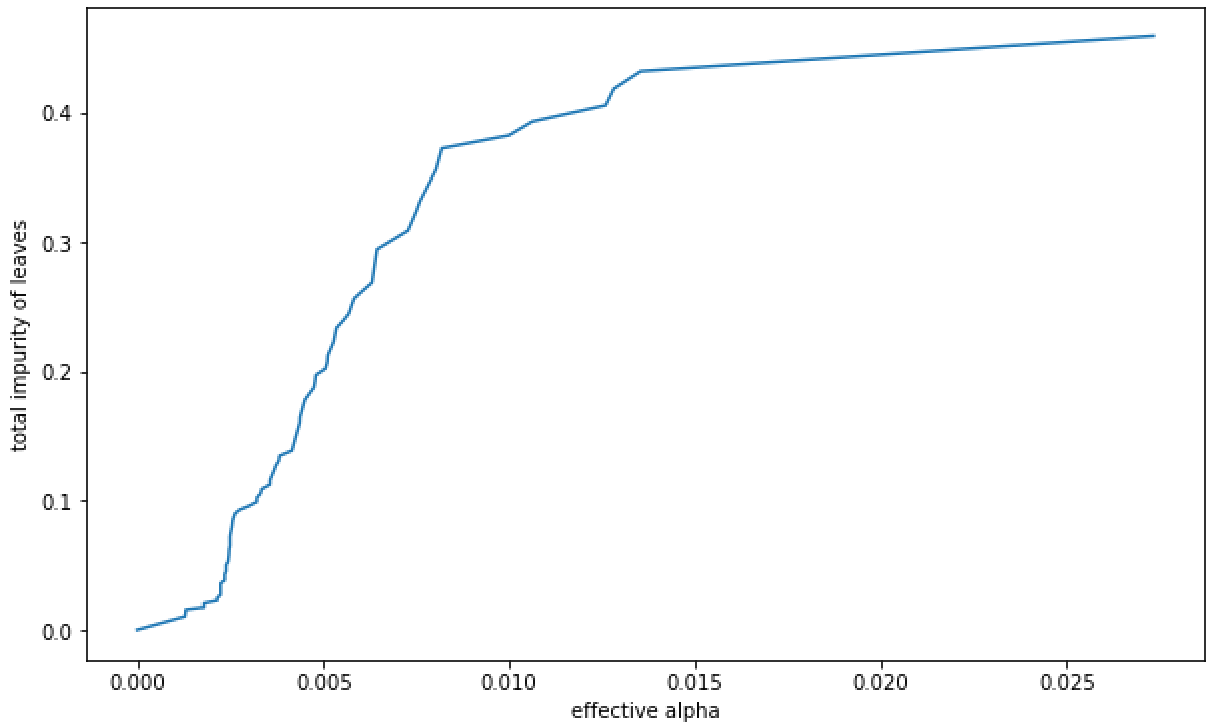
total impurity of leaves
0.0
0.1
0.2
0.3
0.4
0.000
0.005
0.010
0.015
0.020
0.025
effective alpha

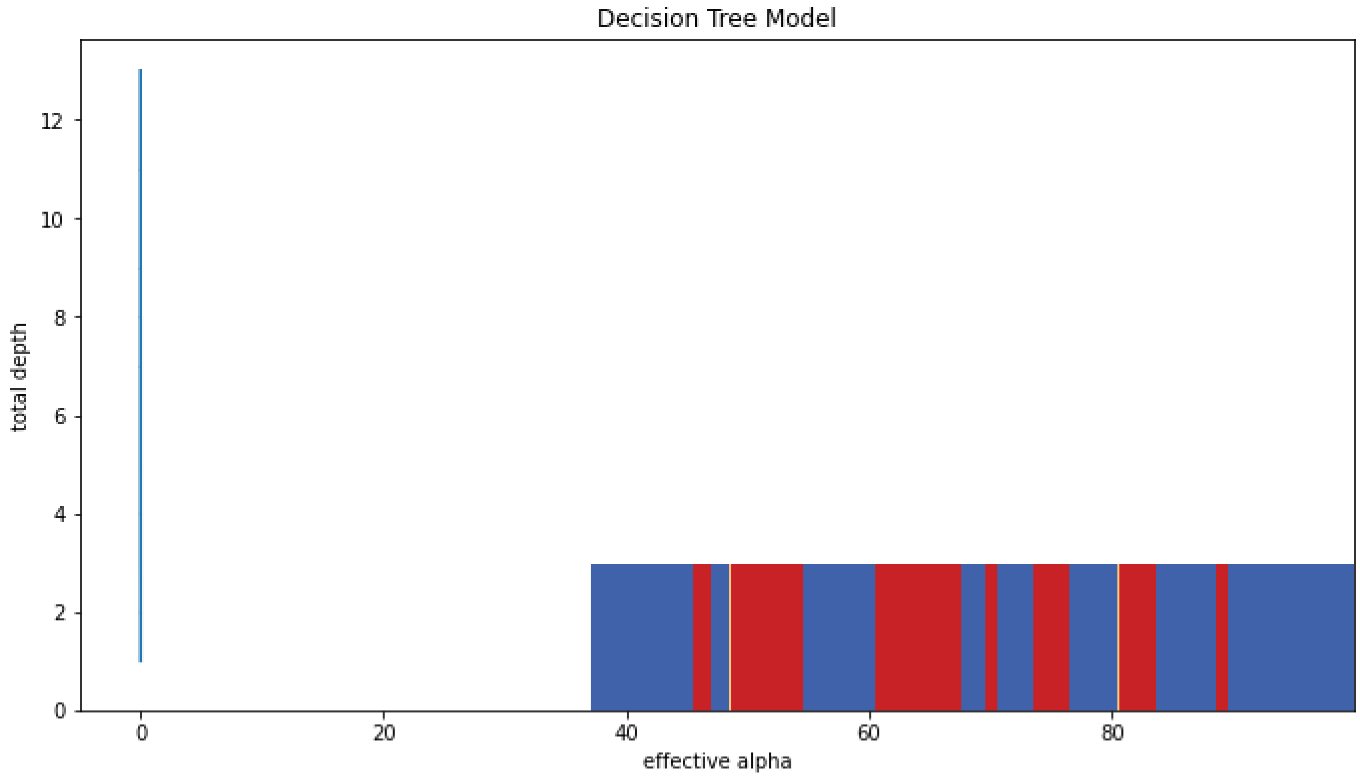


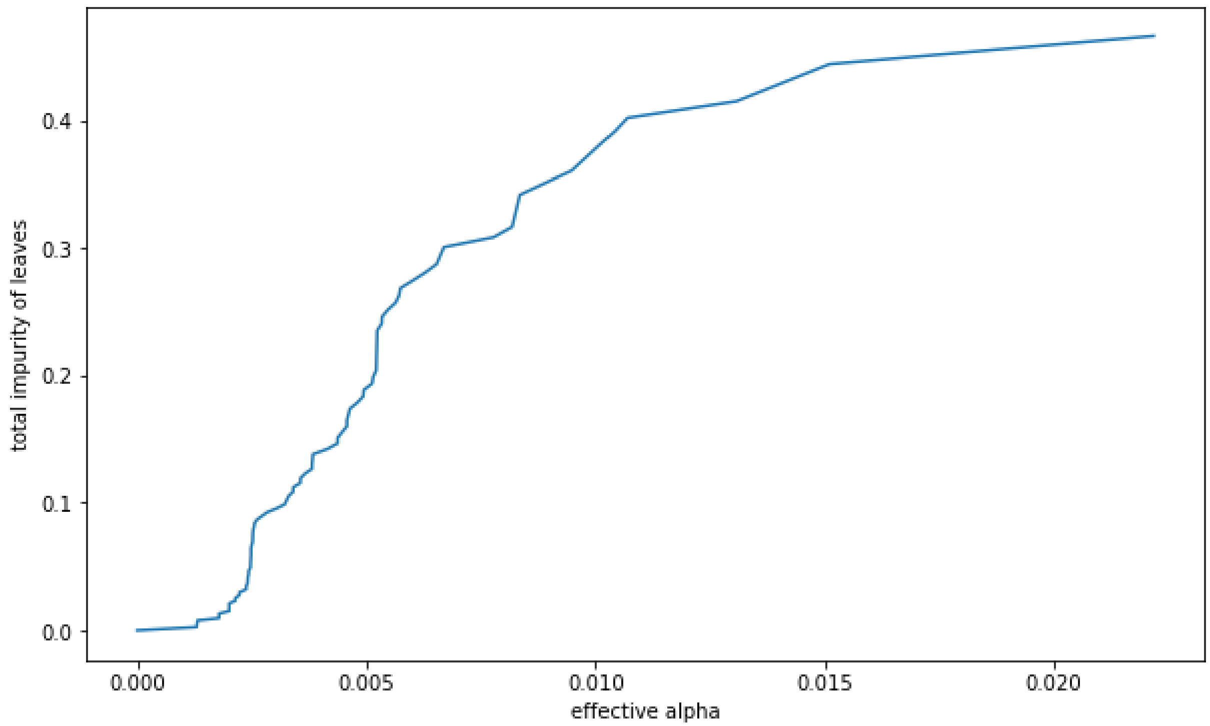
Decision Tree Model
total depth
effective alpha
0
2
4
6
8
10
12
14
0
20
40
60
80

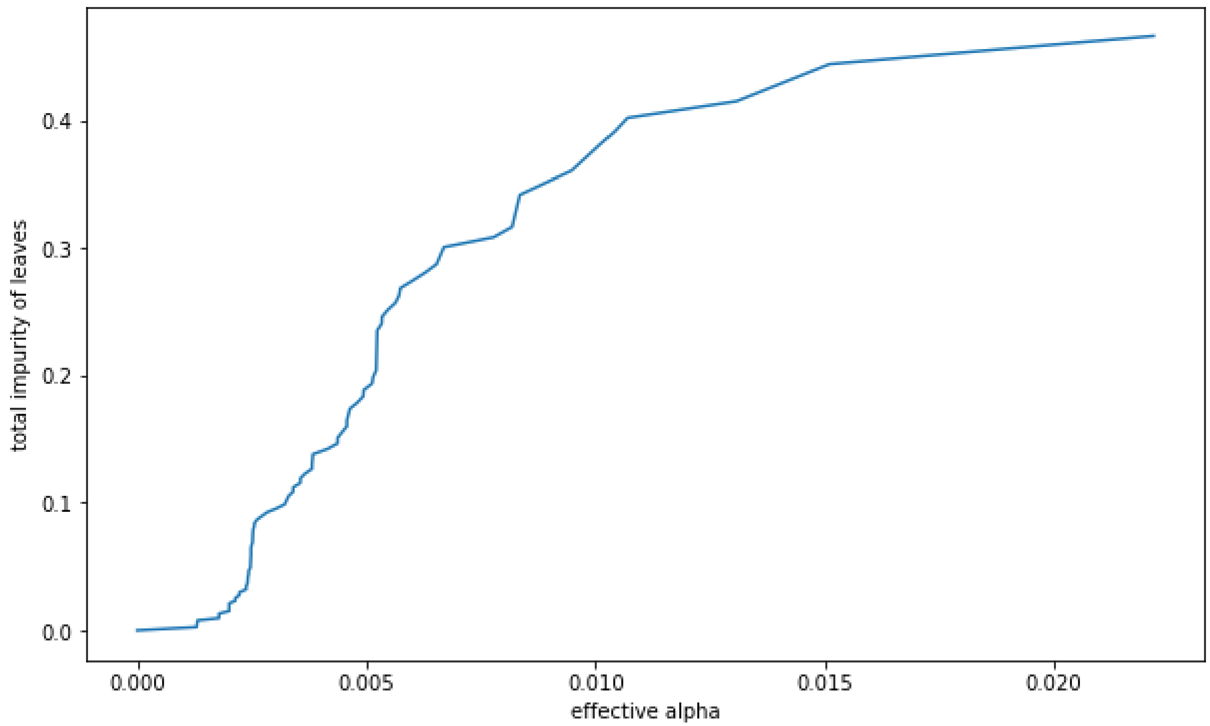


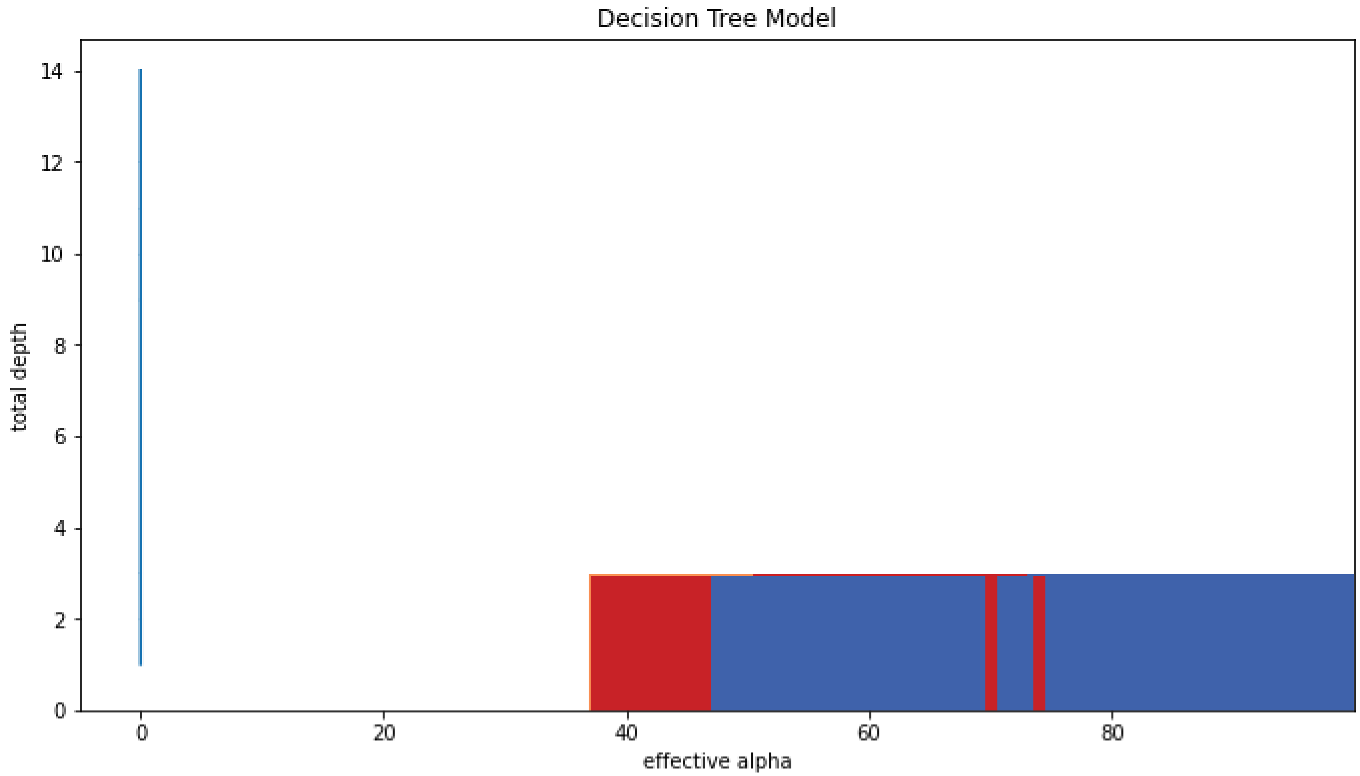
total impurity of leaves
effective alpha
0.0
0.1
0.2
0.3
0.4
0.000
0.005
0.010
0.015
0.020
0.025


0.0
0.1
0.2
0.3
0.4
total impurity of leaves
0.000
0.005
0.010
0.015
0.020
0.025
effective alpha

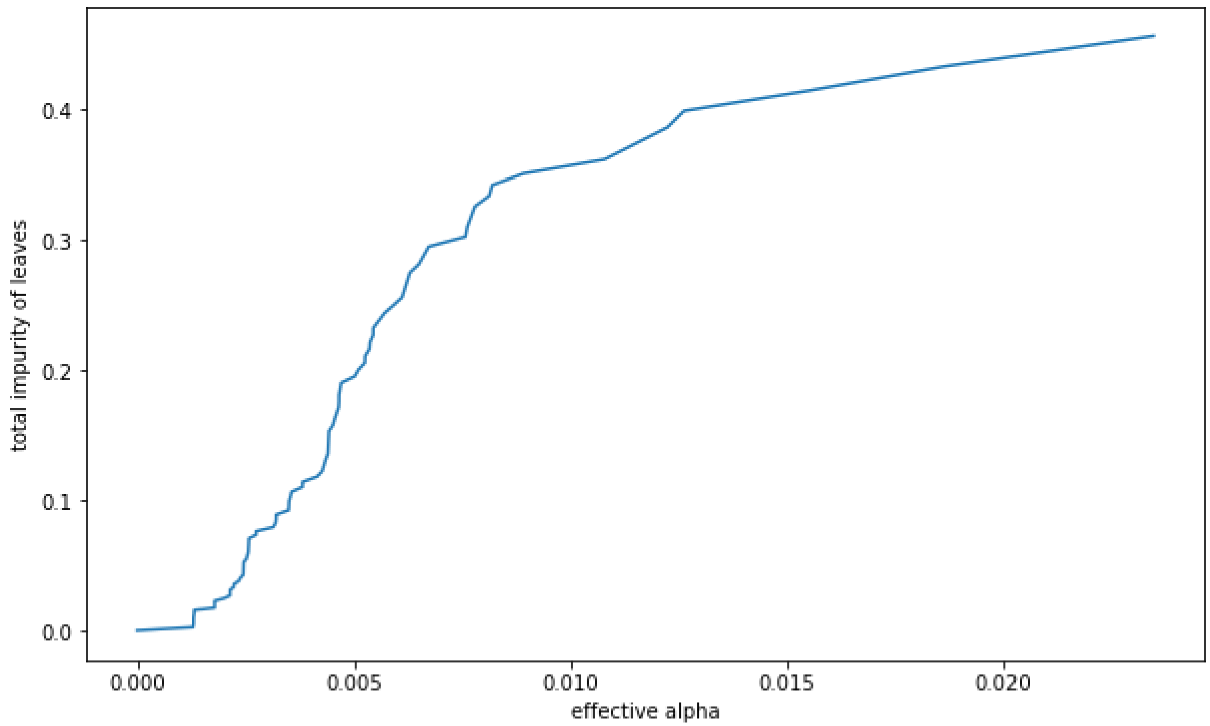

Decision Tree Model
total depth
effective alpha
0
2
4
6
8
10
12
0
20
40
60
80

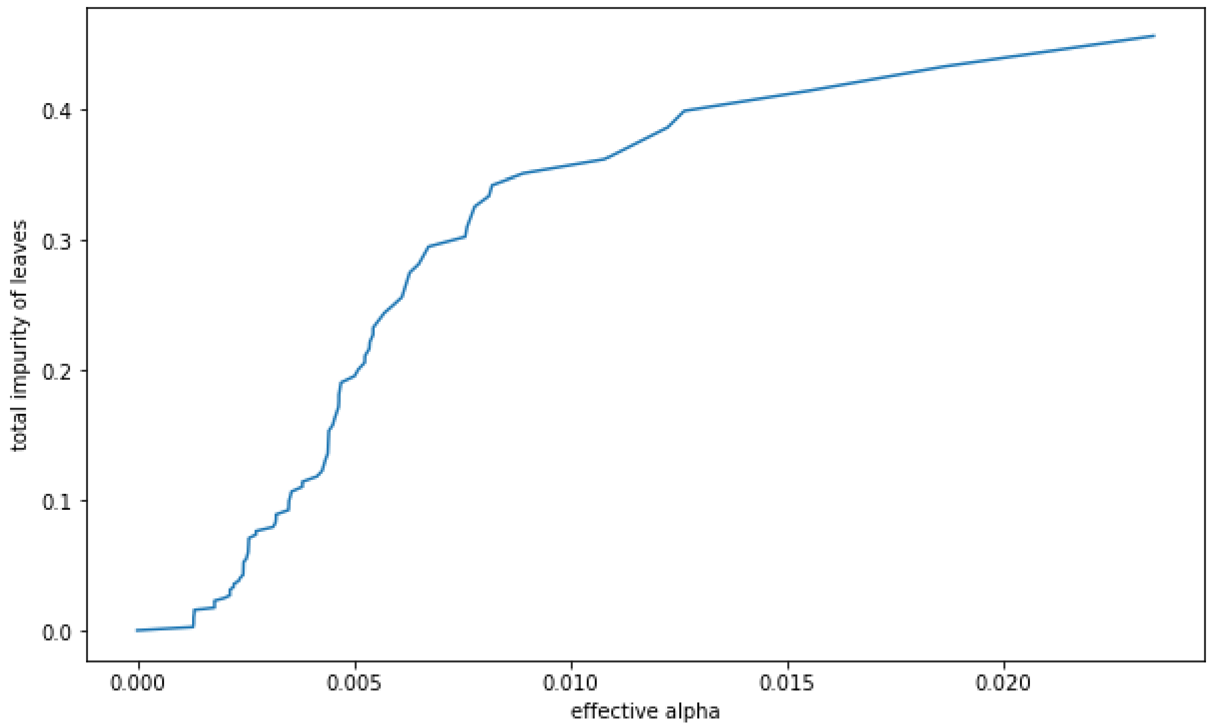

total impurity of leaves
effective alpha
0.0
0.1
0.2
0.3
0.4
0.000
0.005
0.010
0.015
0.020

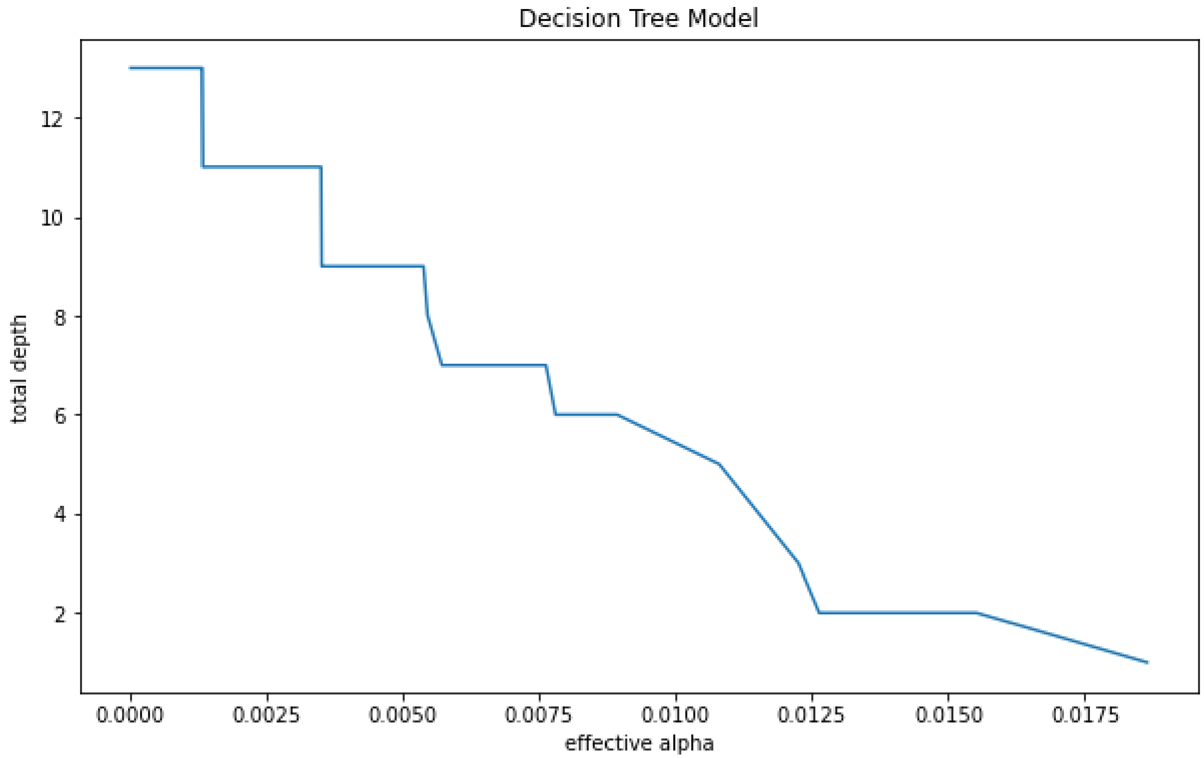

total impurity of leaves
effective alpha
0.0
0.1
0.2
0.3
0.4
0.000
0.005
0.010
0.015
0.020

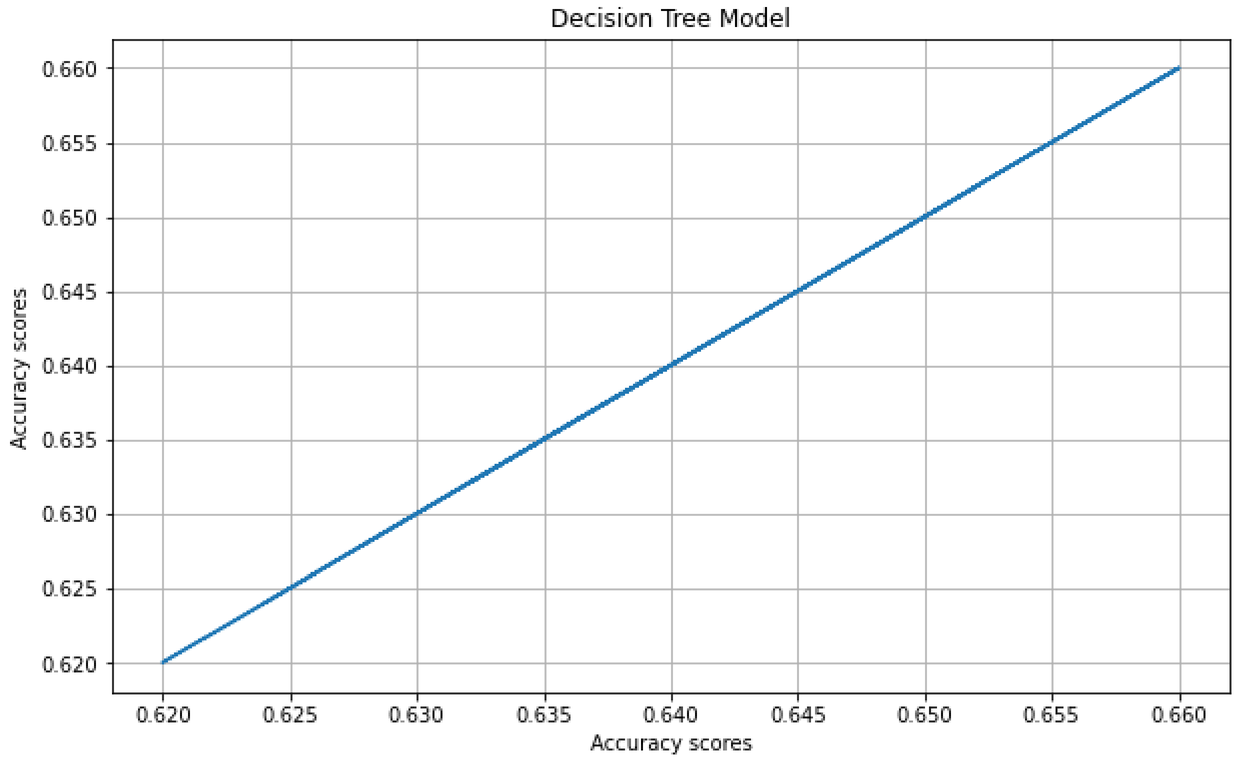


Decision Tree Model
total depth
effective alpha
0
2
4
6
8
10
12
14
0
20
40
60
80


total impurity of leaves
effective alpha
0.0
0.1
0.2
0.3
0.4
0.000
0.005
0.010
0.015
0.020


total impurity of leaves
effective alpha
0.0
0.1
0.2
0.3
0.4
0.000
0.005
0.010
0.015
0.020


Decision Tree Model
total depth
effective alpha
12
10
8
6
4
2
0.0000
0.0025
0.0050
0.0075
0.0100
0.0125
0.0150
0.0175



In [5]:
```
print(f" accuracy = {accuracy_score(y_test, global_clf.predict(x_test))}")
```

```
accuracy = 0.716
```

In [6]:
```
    #done
    dm = data_modeling()


    x_train,y_train,x_test,y_test = dm.process_data(5,5)

    X = x_train.iloc[:,:]
    # Train
    #clf = DecisionTreeClassifier().fit( x_train.iloc[:,:], y_train)
    import graphviz
    from sklearn import tree
    import pydotplus

    #X = x_train.iloc[:,2:50]

    clf = DecisionTreeClassifier().fit( X, y_train)
```

```
    path = clf.cost_complexity_pruning_path(X, y_train)
    path

    dot_data = tree.export_graphviz(clf, node_ids = True,
                                    proportion = True,
                                    feature_names = list(X.columns.values.tolist()) ,
                                    class_names = ['5','6','7'],
                                    filled = True,
                                    rounded = True)

    import graphviz
    gvz_graph = graphviz.Source(dot_data)
    gvz_graph
    gvz_graph.render('dtree_render_', view=True)
```

```
f FFFF ----->>  {Y}
```

Out[6]:
```
'dtree_render_.pdf'
```

In [ ]:

In [7]:
```
#done
path = clf.cost_complexity_pruning_path(X, y_train)
path
```

Out[7]:
```
{'ccp_alphas': array([0.        , 0.0012973 , 0.00130233, 0.00133333, 0.00177778,
        0.00177778, 0.00177778, 0.00177778, 0.00177778, 0.00177778,
        0.002     , 0.002     , 0.002     , 0.002     , 0.002     ,
        0.00213333, 0.00213333, 0.00213333, 0.00213333, 0.00216021,
        0.00228571, 0.00233333, 0.00237037, 0.00237037, 0.00243101,
        0.00244444, 0.00247619, 0.00249462, 0.0026383 , 0.00279365,
        0.00296296, 0.00296296, 0.00302763, 0.00304762, 0.00318774,
        0.0032    , 0.00320703, 0.00332549, 0.0036129 , 0.00374077,
        0.00380503, 0.00381818, 0.004     , 0.00415385, 0.00434603,
        0.00436364, 0.00443932, 0.00457143, 0.00457143, 0.00458738,
        0.00465455, 0.00474074, 0.00482655, 0.00484034, 0.0048949 ,
        0.00494779, 0.00525128, 0.00534896, 0.00541799, 0.0055543 ,
        0.00557618, 0.0056779 , 0.00581114, 0.00605257, 0.00631304,
        0.00666109, 0.00677653, 0.00691234, 0.00713607, 0.00780828,
        0.00971319, 0.00992648, 0.01068889, 0.0110296 , 0.01113014,
        0.01310189, 0.01567061]),
 'impurities': array([0.        , 0.00259459, 0.00519925, 0.00653258, 0.00831036,
        0.01008813, 0.01186591, 0.01364369, 0.01542147, 0.01719925,
        0.01919925, 0.02119925, 0.02319925, 0.02519925, 0.02719925,
        0.02933258, 0.03146591, 0.03359925, 0.03573258, 0.0422132 ,
        0.04678463, 0.04911796, 0.05148833, 0.0538587 , 0.05872072,
```

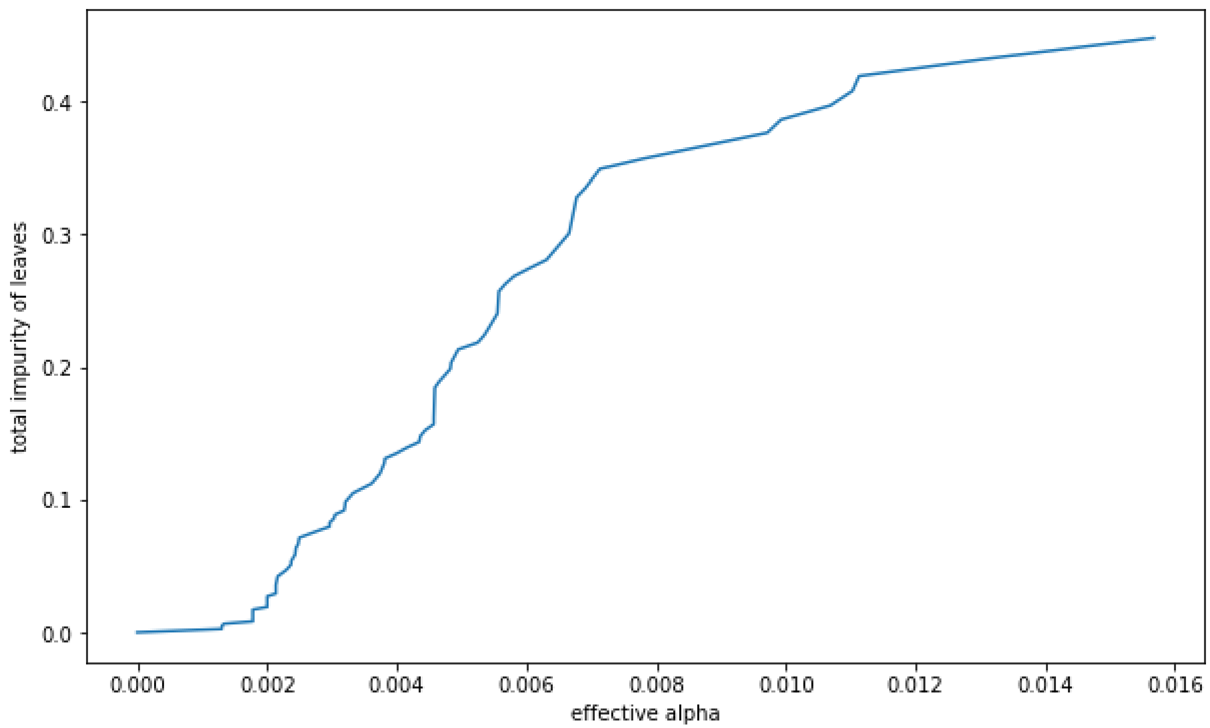
```
0.06360961, 0.0660858 , 0.07107504, 0.07371334, 0.07650699,
0.07946996, 0.08243292, 0.08546055, 0.08850817, 0.0916959 ,
0.0948959 , 0.09810294, 0.10475392, 0.11197972, 0.11946127,
0.12707134, 0.13088952, 0.13488952, 0.13904337, 0.1433894 ,
0.14775303, 0.15219236, 0.15676379, 0.16133521, 0.1842721 ,
0.18892664, 0.19366738, 0.19849393, 0.20333426, 0.20822917,
0.21317696, 0.21842824, 0.2237772 , 0.22919519, 0.24030378,
0.25703232, 0.26271022, 0.26852136, 0.27457393, 0.28088698,
0.30087024, 0.32797635, 0.33488869, 0.34916084, 0.35696912,
0.3763955 , 0.38632198, 0.39701087, 0.40804048, 0.41917061,
0.4322725 , 0.44794311])}
```

In [8]:

```
ccp_alphas, impurities = path.ccp_alphas, path.impurities

plt.figure(figsize=(10, 6))
plt.plot(ccp_alphas, impurities)
plt.xlabel("effective alpha")
plt.ylabel("total impurity of leaves")
```

Out[8]:

```
Text(0, 0.5, 'total impurity of leaves')
```



In [9]:

```
#done
clfs = []
```

```python
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(random_state=0, ccp_alpha=ccp_alpha)
    clf.fit(X, y_train)
    clfs.append(clf)
```

In [10]:
```python
#done
print(f'clfs {clfs}')
```

```
clfs [DecisionTreeClassifier(random_state=0), DecisionTreeClassifier(ccp_alpha=0.0012972972972972985, random_st
ate=0), DecisionTreeClassifier(ccp_alpha=0.0013023255813953535, random_state=0), DecisionTreeClassifier(ccp_alpha
=0.0013333333333333333, random_state=0), DecisionTreeClassifier(ccp_alpha=0.0017777777777777776, random_state=
0), DecisionTreeClassifier(ccp_alpha=0.0017777777777777776, random_state=0), DecisionTreeClassifier(ccp_alpha=
0.0017777777777777776, random_state=0), DecisionTreeClassifier(ccp_alpha=0.0017777777777777776, random_state=
0), DecisionTreeClassifier(ccp_alpha=0.0017777777777777776, random_state=0), DecisionTreeClassifier(ccp_alpha=
0.0017777777777777776, random_state=0), DecisionTreeClassifier(ccp_alpha=0.002, random_state=0), DecisionTreeCl
assifier(ccp_alpha=0.002, random_state=0), DecisionTreeClassifier(ccp_alpha=0.002, random_state=0), DecisionTre
eClassifier(ccp_alpha=0.002, random_state=0), DecisionTreeClassifier(ccp_alpha=0.002, random_state=0), Decision
TreeClassifier(ccp_alpha=0.0021333333333333333, random_state=0), DecisionTreeClassifier(ccp_alpha=0.002133333333
333333, random_state=0), DecisionTreeClassifier(ccp_alpha=0.0021333333333333333, random_state=0), DecisionTreeCl
assifier(ccp_alpha=0.0021333333333333333, random_state=0), DecisionTreeClassifier(ccp_alpha=0.00216020671834625
8, random_state=0), DecisionTreeClassifier(ccp_alpha=0.0022857142857142863, random_state=0), DecisionTreeClassi
fier(ccp_alpha=0.0023333333333333335, random_state=0), DecisionTreeClassifier(ccp_alpha=0.0023703703703703703,
random_state=0), DecisionTreeClassifier(ccp_alpha=0.0023703703703703703, random_state=0), DecisionTreeClassifie
r(ccp_alpha=0.002431007751937984, random_state=0), DecisionTreeClassifier(ccp_alpha=0.0024444444444444445, rando
m_state=0), DecisionTreeClassifier(ccp_alpha=0.002476190476190477, random_state=0), DecisionTreeClassifier(ccp_
alpha=0.002494623655913977, random_state=0), DecisionTreeClassifier(ccp_alpha=0.0026382978723404286, random_st
ate=0), DecisionTreeClassifier(ccp_alpha=0.002793650793650795, random_state=0), DecisionTreeClassifier(ccp_alph
a=0.002962962962962963 2, random_state=0), DecisionTreeClassifier(ccp_alpha=0.0029629629629629632, random_state=
0), DecisionTreeClassifier(ccp_alpha=0.0030276276276276276, random_state=0), DecisionTreeClassifier(ccp_alpha=
0.003047619047619048, random_state=0), DecisionTreeClassifier(ccp_alpha=0.003187739463601306, random_state=0),
DecisionTreeClassifier(ccp_alpha=0.0031999999999999997, random_state=0), DecisionTreeClassifier(ccp_alpha=0.003
2070317782285328, random_state=0), DecisionTreeClassifier(ccp_alpha=0.003325490196078429, random_state=0), Deci
sionTreeClassifier(ccp_alpha=0.003612903225806452, random_state=0), DecisionTreeClassifier(ccp_alpha=0.00374077
27665771558, random_state=0), DecisionTreeClassifier(ccp_alpha=0.0038050340703657346, random_state=0), Decision
TreeClassifier(ccp_alpha=0.0038181818181818181 8, random_state=0), DecisionTreeClassifier(ccp_alpha=0.004, random_
state=0), DecisionTreeClassifier(ccp_alpha=0.004153846153846155, random_state=0), DecisionTreeClassifier(ccp_al
pha=0.004346031746031747, random_state=0), DecisionTreeClassifier(ccp_alpha=0.0043636363636363636 4, random_state=
0), DecisionTreeClassifier(ccp_alpha=0.0044393241167434 75, random_state=0), DecisionTreeClassifier(ccp_alpha=0.
004571428571428571, random_state=0), DecisionTreeClassifier(ccp_alpha=0.004571428571428571, random_state=0), De
cisionTreeClassifier(ccp_alpha=0.004587376386170 91, random_state=0), DecisionTreeClassifier(ccp_alpha=0.0046545
45454545455, random_state=0), DecisionTreeClassifier(ccp_alpha=0.0047407407407407 4, random_state=0), DecisionTr
eeClassifier(ccp_alpha=0.004826546003016589, random_state=0), DecisionTreeClassifier(ccp_alpha=0.00484033613445
3781, random_state=0), DecisionTreeClassifier(ccp_alpha=0.004894901960784 31, random_state=0), DecisionTreeClass
ifier(ccp_alpha=0.004947789513358863, random_state=0), DecisionTreeClassifier(ccp_alpha=0.005251282051282051, r
andom_state=0), DecisionTreeClassifier(ccp_alpha=0.00534896214896215, random_state=0), DecisionTreeClassifier(c
cp_alpha=0.005417989417989418, random_state=0), DecisionTreeClassifier(ccp_alpha=0.005554295152195 51, random_st
ate=0), DecisionTreeClassifier(ccp_alpha=0.005576181508384896, random_state=0), DecisionTreeClassifier(ccp_alph
```

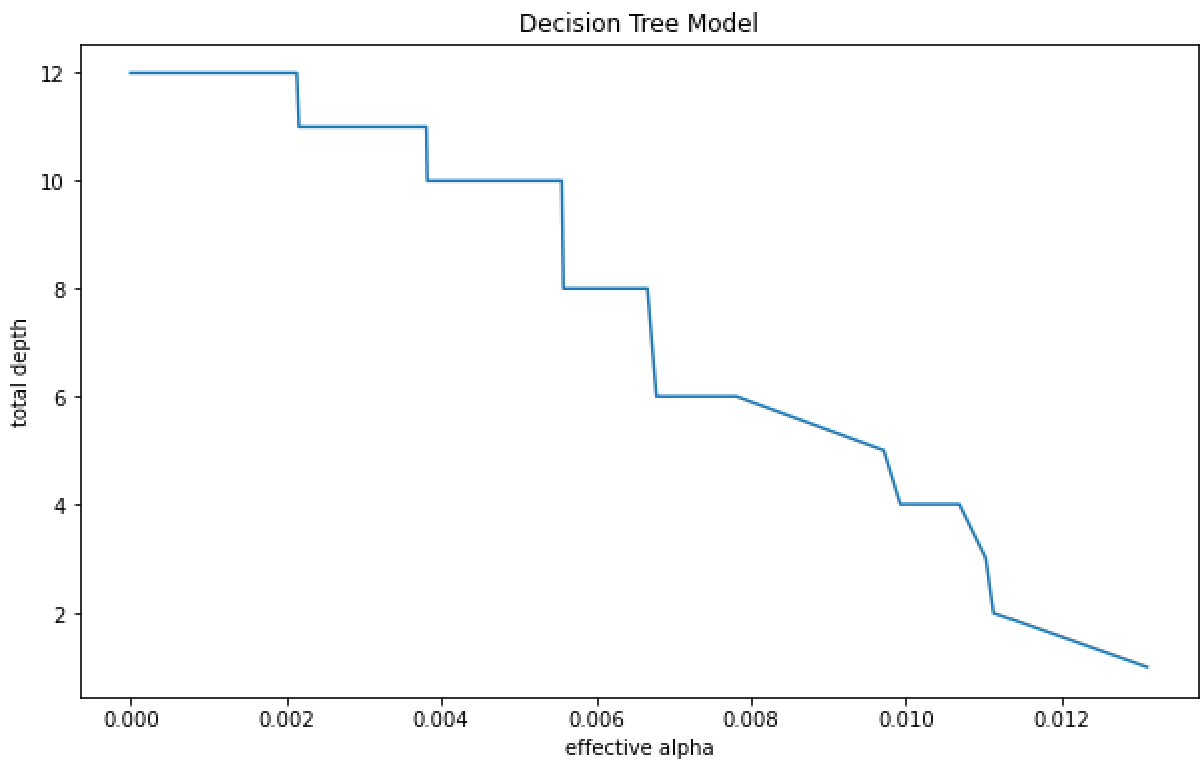
```
a=0.005677897252090796, random_state=0), DecisionTreeClassifier(ccp_alpha=0.005811380145278516, random_state=
0), DecisionTreeClassifier(ccp_alpha=0.006052574876104288, random_state=0), DecisionTreeClassifier(ccp_alpha=0.
006313041455898605 5, random_state=0), DecisionTreeClassifier(ccp_alpha=0.006661089198866969, random_state=0), D
ecisionTreeClassifier(ccp_alpha=0.006776527154871 37, random_state=0), DecisionTreeClassifier(ccp_alpha=0.006912
341808893534, random_state=0), DecisionTreeClassifier(ccp_alpha=0.007136071136071137, random_state=0), Decision
TreeClassifier(ccp_alpha=0.007808278867102399, random_state=0), DecisionTreeClassifier(ccp_alpha=0.009713190624
145343, random_state=0), DecisionTreeClassifier(ccp_alpha=0.009926481203842527, random_state=0), DecisionTreeCl
assifier(ccp_alpha=0.010688893014020345, random_state=0), DecisionTreeClassifier(ccp_alpha=0.01102960485569182
1, random_state=0), DecisionTreeClassifier(ccp_alpha=0.011130135184009 57, random_state=0), DecisionTreeClassifi
er(ccp_alpha=0.013101886121700546, random_state=0), DecisionTreeClassifier(ccp_alpha=0.015670614095186275, rand
om_state=0)]
```

In [11]:

```python
#done
tree_depths = [clf.tree_.max_depth for clf in clfs]
plt.figure(figsize=(10,  6))
plt.plot(ccp_alphas[:-1], tree_depths[:-1])
plt.title("Decision Tree Model")
plt.xlabel("effective alpha")
plt.ylabel("total depth")
```

Out[11]:

```
Text(0, 0.5, 'total depth')
```



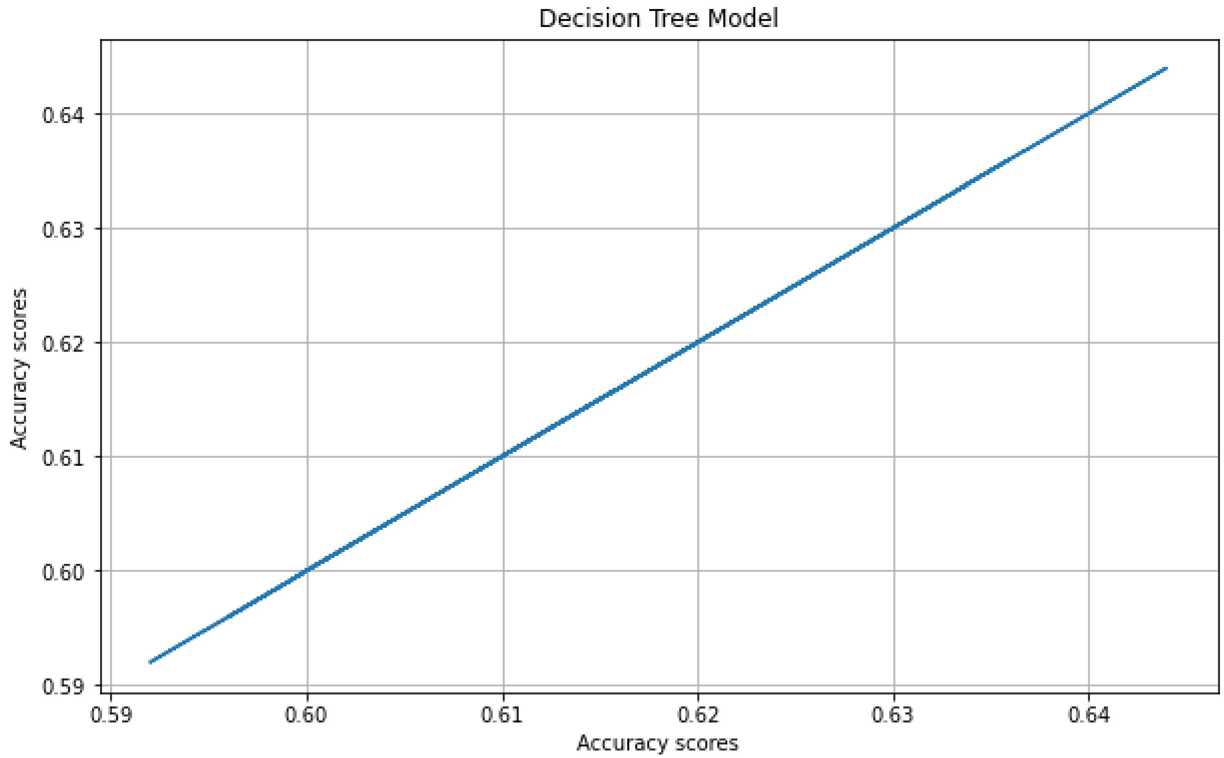
In [12]:

```python
#done
from sklearn.metrics import accuracy_score

print(f' x len {len(x_test)}')
print(f' y len {len(y_test)}')

acc_scores = [accuracy_score(y_test, clf.predict(x_test)) for clf in clfs]

print(f'acc score {acc_scores}')
tree_depths = [clf.tree_.max_depth for clf in clfs]
plt.figure(figsize=(10,  6))
plt.grid()
plt.plot(acc_scores[:-1], acc_scores[:-1])
plt.title("Decision Tree Model")
plt.xlabel("Accuracy scores")
plt.ylabel("Accuracy scores")
```

```
 x len 250
 y len 250
acc score [0.616, 0.616, 0.616, 0.616, 0.612, 0.612, 0.612, 0.612, 0.612, 0.612, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6,
0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.596, 0.592, 0.604, 0.608, 0.608, 0.616, 0.616, 0.616, 0.616, 0.6
16, 0.616, 0.616, 0.616, 0.616, 0.608, 0.612, 0.62, 0.624, 0.624, 0.624, 0.624, 0.628, 0.628, 0.628, 0.636, 0.6
36, 0.632, 0.624, 0.624, 0.624, 0.62, 0.604, 0.6, 0.596, 0.604, 0.596, 0.604, 0.604, 0.616, 0.612, 0.616, 0.63
6, 0.632, 0.628, 0.624, 0.616, 0.62, 0.644, 0.6, 0.624, 0.624, 0.596]
```

Out[12]:

```
Text(0, 0.5, 'Accuracy scores')
```



In [ ]:

In [ ]:

In [13]:
```
global_accuracy = 0
acc_list = []
global_clf = None
for index in range(1,10):
    from sklearn import tree

    from sklearn.tree import export_graphviz
    from IPython import display
    from sklearn.ensemble import RandomForestRegressor

    import matplotlib.pyplot as plt
    from sklearn import tree
        #import pandas as pd
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.model_selection import train_test_split
    dm = data_modeling()
```

```
x_train,y_train,x_test,y_test = dm.process_data(5,5)

print(__doc__)

import numpy as np
import matplotlib.pyplot as plt

from sklearn.tree import DecisionTreeClassifier, plot_tree

# Parameters
n_classes = 3
plot_colors = "ryb"
plot_step = 0.05

'''
for pairidx, pair in enumerate([[0, 1], [0, 2], [0, 3],
                                [1, 2], [1, 3], [2, 3]]):


    X = x_train.iloc[:,pair]


    clf = RandomForestRegressor().fit(X, y_train.to_numpy().ravel())

    estimator = clf.estimators_[5]

    print(f'clf ', clf)

    # Plot the decision boundary
    plt.subplot(2, 3, pairidx + 1)

    x_min, x_max = 37,100 #X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max =  0,3 # X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, plot_step),
                         np.arange(y_min, y_max, plot_step))
    #plt.tight_layout(h_pad=1, w_pad=1, pad=2.5)

    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    cs = plt.contourf(xx, yy, Z, cmap=plt.cm.RdYlBu)


    # Plot the training points
    for i, color in zip(range(n_classes), plot_colors):
```

```
            plt.scatter(X.iloc[i, 0], X.iloc[i, 1], c=color,
                  cmap=plt.cm.RdYlBu, edgecolor='black', s=15)
        #plt.scatter(X.iloc[i,0],X.iloc[i,1],s=15)

        plt.suptitle("Decision surface of a decision tree using paired features")
        plt.legend(loc='lower right', borderpad=0, handletextpad=0)
        plt.axis("tight")

        fig, axes = plt.subplots(nrows = 1,ncols = 5,figsize = (10,2), dpi=900)
        for index in range(0, 5):
            tree.plot_tree(clf.estimators_[index],
                   filled = True,
                   ax = axes[index]);

        axes[index].set_title('Estimator: ' + str(index), fontsize = 11)
        s = 'rf' + str(index) +'.png'
        fig.savefig(s)

    import graphviz
    from sklearn import tree
    import pydotplus
    from six import StringIO
#X = x_train.iloc[:,2:50]

    #dm = data_modeling()

    '''
    from sklearn.tree import export_graphviz
        #from sklearn.externals.six import StringIO
    from six import StringIO
    from IPython.display import Image
    import pydotplus
    import os


    i_tree = 0
    dotfile = StringIO()
    l = [ 1,10]

    '''
    for tree_in_forest in clf.estimators_:
        if i_tree in l:
            export_graphviz(tree_in_forest, out_file=dotfile)
            s = 'dot -Tpng tree.dot -o ' + 'tree' + str(i_tree) + '.png'
            os.system(s)
```

```
        i_tree = i_tree + 1
    '''



#x_train,y_train,x_test,y_test = dm.process_data(5,5)

    X = x_train
    acc= 0
    for estimator in range (2,100):

    #clf = RandomForestRegressor(n_estimators=estimator).fit(X, y_train.to_numpy().ravel())
        clf = RandomForestClassifier(n_estimators=estimator).fit(X, y_train.to_numpy().ravel())

        acc_scores = accuracy_score(y_test, clf.predict(x_test))
        if acc_scores  > acc:
            acc =acc_scores
            estimator_tree = estimator
            clf_ = clf
            print(f"estimator {estimator_tree}")
            print(f'accuracy scores = {acc_scores}')

        if acc > global_accuracy:
            global_accuracy = acc
        global_clf = clf_

    print(" done")

    print(f'accuracy scores = {acc_scores}')


acc_scores = accuracy_score(y_test, global_clf.predict(x_test))
print(f' Accuracy of Random Forest {acc_scores}')
```

```
f FFFF ----->>  {Y}
Automatically created module for IPython interactive environment
estimator 2
accuracy scores = 0.528
estimator 3
accuracy scores = 0.66
estimator 9
accuracy scores = 0.676
estimator 11
accuracy scores = 0.712
estimator 13
accuracy scores = 0.724
estimator 15
```

```
accuracy scores = 0.74
estimator 30
accuracy scores = 0.76
 done
accuracy scores = 0.688
f FFFF ----->>  {Y}
Automatically created module for IPython interactive environment
estimator 2
accuracy scores = 0.588
estimator 3
accuracy scores = 0.644
estimator 6
accuracy scores = 0.688
estimator 10
accuracy scores = 0.72
estimator 22
accuracy scores = 0.724
estimator 23
accuracy scores = 0.728
estimator 39
accuracy scores = 0.732
estimator 68
accuracy scores = 0.736
 done
accuracy scores = 0.68
f FFFF ----->>  {Y}
Automatically created module for IPython interactive environment
estimator 2
accuracy scores = 0.564
estimator 3
accuracy scores = 0.68
estimator 8
accuracy scores = 0.7
estimator 9
accuracy scores = 0.716
estimator 10
accuracy scores = 0.728
estimator 13
accuracy scores = 0.732
estimator 21
accuracy scores = 0.748
estimator 23
accuracy scores = 0.764
estimator 34
accuracy scores = 0.772
estimator 35
accuracy scores = 0.776
estimator 38
accuracy scores = 0.788
```

```
 done
accuracy scores = 0.772
f FFFF ----->>  {Y}
Automatically created module for IPython interactive environment
estimator 2
accuracy scores = 0.504
estimator 3
accuracy scores = 0.624
estimator 4
accuracy scores = 0.656
estimator 5
accuracy scores = 0.66
estimator 8
accuracy scores = 0.7
estimator 10
accuracy scores = 0.716
estimator 13
accuracy scores = 0.728
estimator 15
accuracy scores = 0.744
estimator 18
accuracy scores = 0.756
estimator 33
accuracy scores = 0.764
estimator 48
accuracy scores = 0.776
 done
accuracy scores = 0.756
f FFFF ----->>  {Y}
Automatically created module for IPython interactive environment
estimator 2
accuracy scores = 0.6
estimator 3
accuracy scores = 0.608
estimator 6
accuracy scores = 0.632
estimator 7
accuracy scores = 0.696
estimator 9
accuracy scores = 0.704
estimator 11
accuracy scores = 0.716
estimator 14
accuracy scores = 0.74
estimator 18
accuracy scores = 0.744
estimator 36
accuracy scores = 0.748
estimator 59
```

```
accuracy scores = 0.76
 done
accuracy scores = 0.732
f FFFF ----->>  {Y}
Automatically created module for IPython interactive environment
estimator 2
accuracy scores = 0.508
estimator 3
accuracy scores = 0.608
estimator 4
accuracy scores = 0.616
estimator 5
accuracy scores = 0.628
estimator 7
accuracy scores = 0.664
estimator 9
accuracy scores = 0.668
estimator 10
accuracy scores = 0.676
estimator 11
accuracy scores = 0.688
estimator 13
accuracy scores = 0.728
estimator 16
accuracy scores = 0.748
 done
accuracy scores = 0.732
f FFFF ----->>  {Y}
Automatically created module for IPython interactive environment
estimator 2
accuracy scores = 0.548
estimator 3
accuracy scores = 0.572
estimator 4
accuracy scores = 0.644
estimator 5
accuracy scores = 0.684
estimator 10
accuracy scores = 0.712
 done
accuracy scores = 0.664
f FFFF ----->>  {Y}
Automatically created module for IPython interactive environment
estimator 2
accuracy scores = 0.596
estimator 3
accuracy scores = 0.616
estimator 5
accuracy scores = 0.7
```

```
estimator 8
accuracy scores = 0.704
estimator 15
accuracy scores = 0.728
estimator 23
accuracy scores = 0.74
 done
accuracy scores = 0.68
f FFFF ----->>  {Y}
Automatically created module for IPython interactive environment
estimator 2
accuracy scores = 0.52
estimator 3
accuracy scores = 0.664
estimator 6
accuracy scores = 0.672
estimator 7
accuracy scores = 0.684
estimator 12
accuracy scores = 0.7
estimator 13
accuracy scores = 0.704
estimator 15
accuracy scores = 0.712
estimator 18
accuracy scores = 0.716
estimator 21
accuracy scores = 0.74
estimator 26
accuracy scores = 0.744
estimator 37
accuracy scores = 0.752
estimator 49
accuracy scores = 0.76
estimator 59
accuracy scores = 0.78
 done
accuracy scores = 0.748
 Accuracy of Random Forest 0.78
```

In [15]:

```
acc_scores = accuracy_score(y_test, global_clf.predict(x_test))
print(f' Accuracy of Random Forest {acc_scores}')
```

```
 Accuracy of Random Forest 0.78
```

In [18]:

```
#done
from sklearn.tree import export_graphviz
        #from sklearn.externals.six import StringIO
from six import StringIO
```

```python
from IPython.display import Image
import pydotplus
import os


i_tree = 0
dotfile = StringIO()
l = [ 1,10,100]


for tree_in_forest in clf.estimators_:
    if i_tree in l:
        export_graphviz(tree_in_forest, out_file=dotfile)
        s = 'dot -Tpng tree.dot -o ' + 'tree' + str(i_tree) + '.png'
        os.system(s)

    i_tree = i_tree + 1
    print(" For loop done")
print(" done")

dotfile = StringIO()
export_graphviz(tree_in_forest, out_file=dotfile)
s = 'dot -Tpng tree.dot -o ' + 'tree' + str(i_tree) + '.png'
os.system(s)
import graphviz
gvz_graph = graphviz.Source(dot_data)
gvz_graph
gvz_graph.render('rf_render_'+ str(i_tree), view=True)
```

```
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
```

```
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
```

```
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
For loop done
done
```

Out[18]: 'rf_render_99.pdf'

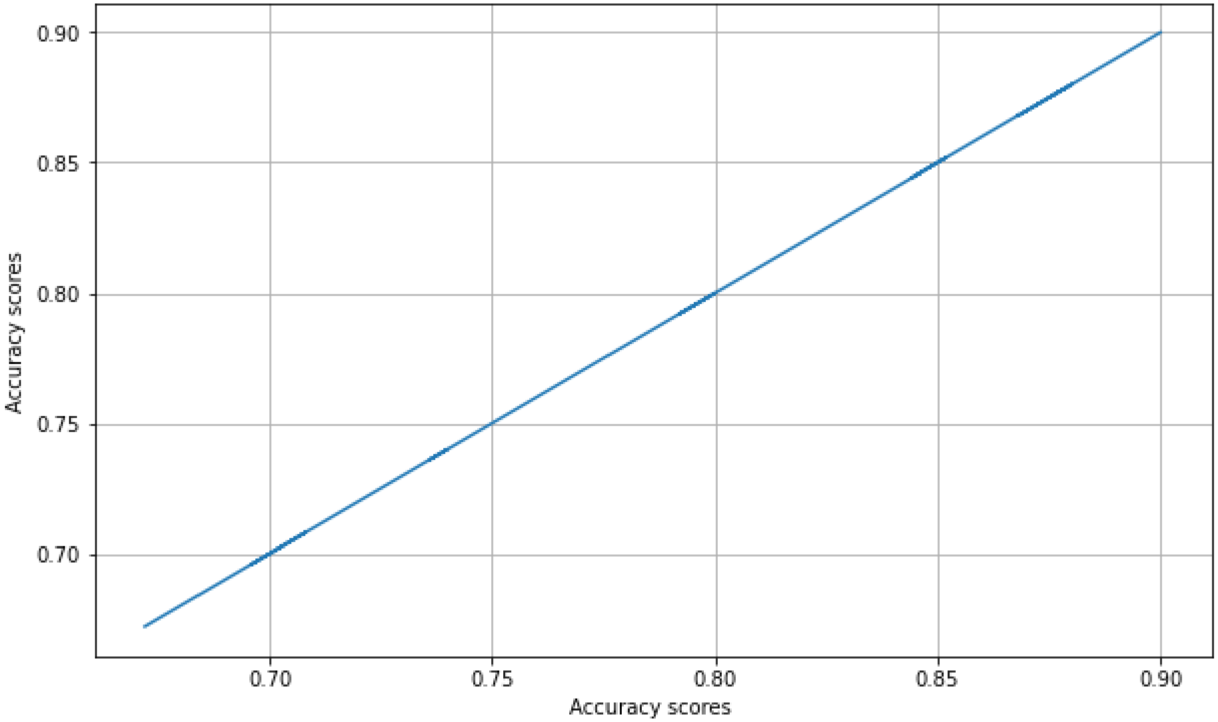
In [16]:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import accuracy_score

print(f' x len {len(x_test)}')
print(f' y len {len(y_test)}')


acc_scores = [accuracy_score(y_test, clf.predict(x_test)) for clf in clfs]
print(f' accuracy = {acc_scores}')
#tree_depths = [clf.tree_.max_depth for clf in clfs]
plt.figure(figsize=(10,  6))
```

```
plt.grid()
plt.plot(acc_scores[:-1], acc_scores[:-1])
plt.xlabel("Accuracy scores ")
plt.ylabel("Accuracy scores")
```

```
x len 250
y len 250
accuracy = [0.9, 0.9, 0.9, 0.9, 0.896, 0.896, 0.896, 0.896, 0.896, 0.896, 0.884, 0.884, 0.884, 0.884, 0.884,
0.88, 0.88, 0.88, 0.88, 0.876, 0.876, 0.876, 0.872, 0.872, 0.872, 0.872, 0.872, 0.876, 0.88, 0.876, 0.88, 0.88,
0.88, 0.876, 0.876, 0.868, 0.872, 0.872, 0.872, 0.86, 0.852, 0.844, 0.844, 0.852, 0.852, 0.848, 0.848, 0.844,
0.844, 0.844, 0.84, 0.832, 0.828, 0.828, 0.828, 0.82, 0.8, 0.8, 0.792, 0.8, 0.8, 0.796, 0.792, 0.776, 0.764, 0.
76, 0.736, 0.74, 0.72, 0.716, 0.7, 0.696, 0.7, 0.708, 0.672, 0.672, 0.648]
```

Out[16]: Text(0, 0.5, 'Accuracy scores')



In [19]:
```
import xgboost as xgb
from xgboost.sklearn import XGBClassifier
from xgboost.sklearn import XGBRegressor
from xgboost import plot_tree

print("Hello World!")

x_train,y_train,x_test,y_test = dm.process_data(5,5)
```

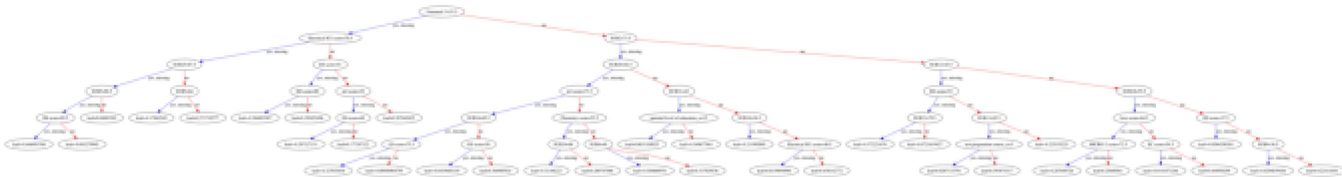
```
x_data =x_train
y_data = y_train

final_train = X_train =  X_test = x_data
y_train = y_data

#optimized_GBM.fit(final_train, y_train)

#X = X_train.iloc[:,2:52]

X = X_train.iloc[:,:]

    # fit model no training data
model = XGBClassifier()
model.fit(X, y_train.to_numpy().ravel())

image = xgb.to_graphviz(model)

fig, ax = plt.subplots(figsize=(12, 12))

export_graphviz(tree_in_forest, out_file=dotfile)
plot_tree(model,ax=ax, fontsize=50,num_trees=1)
format = 'png'
image.render('xgboost_tree', format = format,  view=True)

print(f'\nEnd XGBoost \n {model}')

from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV

cv_params = {'max_depth': [3,5,7], 'min_child_weight': [1,3,5]}
ind_params = {'learning_rate': 0.1, 'n_estimators': 1000, 'seed':0, 'subsample': 0.8, 'colsample_bytree': 0.8,
             'objective': 'binary:logistic'}

c = XGBClassifier(**ind_params)
#c = XGBClassifier()
optimized_ = GridSearchCV(c, cv_params, scoring = 'accuracy', cv = 5, n_jobs = -1)
#optimized_ = GridSearchCV(c, scoring = 'accuracy')

print(f'\noptimized_GBM \n {optimized_}')
optimized_.fit(final_train, y_train)
```

```
Hello World!
```

```
f FFFF ----->>  {Y}

End XGBoost
 XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)

optimized_GBM
 GridSearchCV(cv=5,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=0.8, gamma=None,
                                     gpu_id=None, importance_type='gain',
                                     interaction_constraints=None,
                                     learning_rate=0.1, max_delta_step=None,
                                     max_depth=None, min_child_weight=None,
                                     missing=nan, monotone_constraints=None,
                                     n_estimators=1000, n_jobs=None,
                                     num_parallel_tree=None, random_state=None,
                                     reg_alpha=None, reg_lambda=None,
                                     scale_pos_weight=None, seed=0,
                                     subsample=0.8, tree_method=None,
                                     validate_parameters=None, verbosity=None),
             n_jobs=-1,
             param_grid={'max_depth': [3, 5, 7], 'min_child_weight': [1, 3, 5]},
             scoring='accuracy')
```

```
Out[19]: GridSearchCV(cv=5,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=0.8, gamma=None,
                                     gpu_id=None, importance_type='gain',
                                     interaction_constraints=None,
                                     learning_rate=0.1, max_delta_step=None,
                                     max_depth=None, min_child_weight=None,
                                     missing=nan, monotone_constraints=None,
                                     n_estimators=1000, n_jobs=None,
                                     num_parallel_tree=None, random_state=None,
                                     reg_alpha=None, reg_lambda=None,
                                     scale_pos_weight=None, seed=0,
                                     subsample=0.8, tree_method=None,
                                     validate_parameters=None, verbosity=None),
             n_jobs=-1,
```

```
             param_grid={'max_depth': [3, 5, 7], 'min_child_weight': [1, 3, 5]},
             scoring='accuracy')
```



In [20]:

```
print(f' keys {optimized_.cv_results_.keys()}')
print(f'optimized_ , {optimized_}')
for i in ['mean_test_score', 'std_test_score']:
        print(i," : ",optimized_.cv_results_[i])
```

```
 keys dict_keys(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_time', 'param_max_depth', 'para
m_min_child_weight', 'params', 'split0_test_score', 'split1_test_score', 'split2_test_score', 'split3_test_scor
e', 'split4_test_score', 'mean_test_score', 'std_test_score', 'rank_test_score'])
optimized_ , GridSearchCV(cv=5,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=0.8, gamma=None,
                                     gpu_id=None, importance_type='gain',
                                     interaction_constraints=None,
                                     learning_rate=0.1, max_delta_step=None,
                                     max_depth=None, min_child_weight=None,
                                     missing=nan, monotone_constraints=None,
                                     n_estimators=1000, n_jobs=None,
                                     num_parallel_tree=None, random_state=None,
                                     reg_alpha=None, reg_lambda=None,
                                     scale_pos_weight=None, seed=0,
                                     subsample=0.8, tree_method=None,
                                     validate_parameters=None, verbosity=None),
             n_jobs=-1,
             param_grid={'max_depth': [3, 5, 7], 'min_child_weight': [1, 3, 5]},
             scoring='accuracy')
mean_test_score  :  [0.84533333 0.852      0.85733333 0.85866667 0.83866667 0.84933333
 0.844      0.844      0.84666667]
std_test_score  :  [0.03512517 0.03330666 0.03492214 0.02578544 0.04328715 0.03542755
 0.04057366 0.04100948 0.04195235]
```

In [21]:

```
xgdmat = xgb.DMatrix(final_train, y_train)

params = {'eta': 0.1, 'seed':0, 'subsample': 0.8, 'colsample_bytree': 0.8
              }
m = ['error']
```

```
cv_results = xgb.cv(
    params,
    dtrain=xgdmat,
    num_boost_round=3000,
    seed=42,
    nfold=5,
    #metrics={'mae'},
    early_stopping_rounds=100
)
```

In [22]:
```
cv_results.tail(5)
```

Out[22]:

	train-rmse-mean	train-rmse-std	test-rmse-mean	test-rmse-std
484	0.0004	0.000004	0.410042	0.008643
485	0.0004	0.000004	0.410042	0.008643
486	0.0004	0.000004	0.410042	0.008643
487	0.0004	0.000004	0.410041	0.008643
488	0.0004	0.000004	0.410041	0.008643

In [23]:
```
#import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn import tree
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
```

In [24]:
```
our_params = {'eta': 0.1, 'seed':0, 'subsample': 0.8, 'colsample_bytree': 0.8,
              }

final_gb = xgb.train(our_params, xgdmat,num_boost_round = 432)
```

In [25]:
```
importances = final_gb.get_fscore()
importances
```

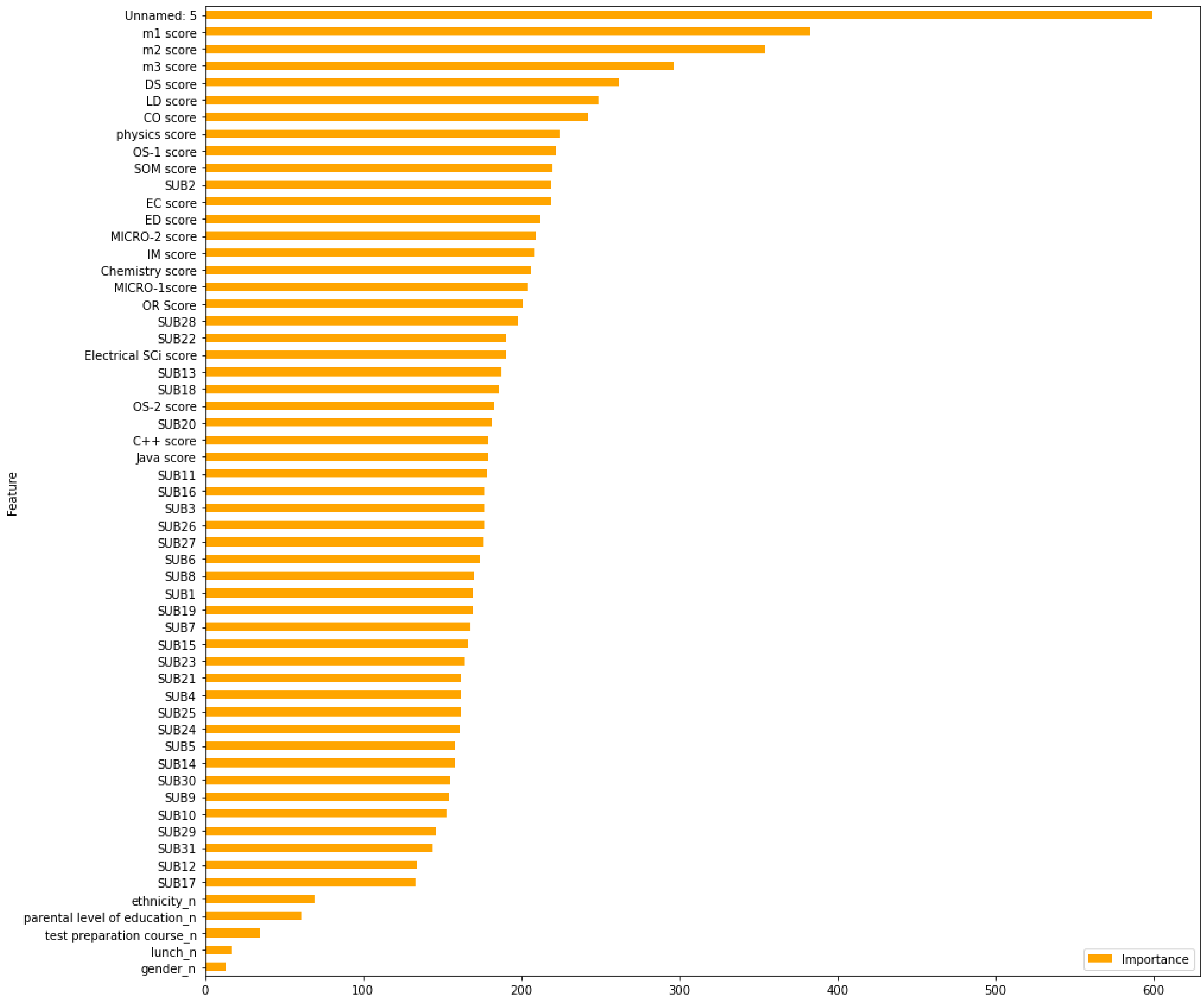
Out[25]:
```
{'Unnamed: 5': 599,
 'physics score': 224,
 'SUB2': 219,
 'm3 score': 296,
 'SUB25': 162,
```

'SUB11': 178,
'EC score': 219,
'OS-2 score': 183,
'MICRO-1score': 204,
'Electrical SCi score': 190,
'SUB28': 198,
'MICRO-2 score': 209,
'OR Score': 201,
'SUB20': 181,
'SUB12': 134,
'SUB8': 170,
'Java score': 179,
'SUB18': 186,
'SOM score': 220,
'SUB27': 176,
'm1 score': 383,
'SUB10': 153,
'SUB29': 146,
'SUB14': 158,
'SUB4': 162,
'SUB16': 177,
'SUB24': 161,
'IM score': 208,
'SUB5': 158,
'SUB7': 168,
'DS score': 262,
'CO score': 242,
'C++ score': 179,
'SUB31': 144,
'SUB13': 187,
'SUB23': 164,
'SUB3': 177,
'SUB9': 154,
'ED score': 212,
'ethnicity_n': 69,
'SUB26': 177,
'SUB19': 169,
'SUB1': 169,
'SUB15': 166,
'Chemistry score': 206,
'LD score': 249,
'SUB30': 155,
'm2 score': 354,
'SUB17': 133,
'OS-1 score': 222,
'SUB21': 162,
'SUB22': 190,
'SUB6': 174,
'parental level of education_n': 61,

```
 'lunch_n': 17,
 'test preparation course_n': 35,
 'gender_n': 13}
```

In [26]:
```
importance_frame = pd.DataFrame({'Importance': list(importances.values()), 'Feature': list(importances.keys())}
importance_frame.sort_values(by = 'Importance', inplace = True)
importance_frame.plot(kind = 'barh', x = 'Feature', figsize = (15,15), color = 'orange')
```

Out[26]:
```
<AxesSubplot:ylabel='Feature'>
```



Unnamed: 5
m1 score
m2 score
m3 score
DS score
LD score
CO score
physics score
OS-1 score
SOM score
SUB2
EC score
ED score
MICRO-2 score
IM score
Chemistry score
MICRO-1score
OR Score
SUB28
SUB22
Electrical SCi score
SUB13
SUB18
OS-2 score
SUB20
C++ score
Java score
SUB11
SUB16
SUB3
SUB26
SUB27
SUB6
SUB8
SUB1
SUB19
SUB7
SUB15
SUB23
SUB21
SUB4
SUB25
SUB24
SUB5
SUB14
SUB30
SUB9
SUB10
SUB29
SUB31
SUB12
SUB17
ethnicity_n
parental level of education_n
test preparation course_n
lunch_n
gender_n
Feature
0
100
200
300
400
500
600
Importance

In [27]:
```python
testdmat = xgb.DMatrix(x_test)
```

In [28]:
```python
from sklearn.metrics import accuracy_score
y_pred = final_gb.predict(testdmat) # Predict using our testdmat
y_pred
```

Out[28]:
```
array([6.0361977, 5.5431232, 6.120784 , 5.5515933, 5.4079924, 5.7313156,
       5.7845755, 5.79256  , 5.694661 , 5.782032 , 5.524447 , 5.465162 ,
       5.7722216, 5.5813537, 5.366006 , 5.961733 , 5.336498 , 5.678376 ,
       5.5900364, 5.7437744, 5.78428  , 5.5374813, 5.810441 , 4.953341 ,
       5.7345047, 5.7439084, 5.8183336, 5.93387  , 5.715259 , 5.6215615,
       5.913161 , 5.717261 , 5.5359087, 5.1328497, 5.583662 , 6.0126076,
       5.839967 , 5.791254 , 5.6944566, 5.9962683, 5.9046216, 5.423353 ,
       5.5698943, 6.026517 , 5.4443874, 5.4859867, 5.5613346, 5.5821004,
       5.6111255, 5.7355537, 5.823526 , 5.6096168, 5.349395 , 5.513648 ,
       5.632035 , 5.8189607, 5.6096253, 5.6206665, 5.390196 , 5.7618647,
       5.66842  , 4.876798 , 5.88809  , 5.7590084, 5.9405437, 5.8320627,
       5.334043 , 5.609115 , 5.2388678, 5.740442 , 5.725162 , 5.6859293,
       5.5355945, 5.969744 , 5.7810674, 5.800886 , 5.525785 , 5.6315303,
       5.705573 , 5.568319 , 5.570446 , 5.631691 , 5.86961  , 5.874181 ,
       5.8046823, 5.9184337, 5.632352 , 5.7437587, 5.8982286, 5.657721 ,
       5.8751273, 5.5418234, 5.8046165, 5.5523486, 5.23184  , 5.5101624,
       5.3561687, 5.860641 , 5.520564 , 5.4655395, 5.557718 , 5.464664 ,
       5.3276005, 5.4601536, 5.8336883, 5.5672398, 5.7929835, 5.6077223,
       5.553751 , 5.40875  , 4.9859605, 5.8352976, 5.5255713, 5.5716667,
       5.391779 , 5.674884 , 5.7447815, 5.5522423, 5.8579283, 5.6939864,
       5.8991547, 5.3354316, 5.406425 , 5.9482713, 5.487004 , 5.4649262,
       5.235033 , 5.4720335, 5.7636876, 5.400336 , 5.8142686, 5.5588336,
       5.746298 , 5.4464006, 5.669022 , 5.209456 , 5.852705 , 6.1171675,
       5.609097 , 5.5552096, 5.523874 , 6.0224185, 5.348506 , 5.2650223,
       5.5878563, 5.758736 , 5.7598853, 5.4583435, 5.5876317, 5.791147 ,
       5.864552 , 5.9404497, 5.5316663, 5.494836 , 5.252971 , 5.7293024,
       5.3750777, 5.768155 , 5.8137608, 5.360561 , 5.5011444, 5.5287933,
       5.7198133, 5.718968 , 5.5640883, 5.27181  , 5.6463046, 5.4663944,
       5.509905 , 5.9119134, 5.9014964, 5.303211 , 5.2357354, 5.6201744,
       5.532865 , 5.594006 , 5.7504535, 5.556195 , 5.607877 , 5.6298547,
       5.8858852, 5.483853 , 5.688022 , 5.5541677, 5.4357615, 5.586385 ,
       5.6437254, 5.3771186, 5.395736 , 5.2556667, 5.9230623, 5.7702575,
       5.3826756, 5.890423 , 5.378423 , 5.737162 , 5.2800655, 5.8292923,
       5.9418383, 5.8950844, 5.5568066, 5.860063 , 5.6972895, 5.5491233,
       5.411632 , 5.176983 , 5.633167 , 5.7129674, 5.743939 , 5.697816 ,
       5.344012 , 5.2524695, 5.2904143, 5.775973 , 5.633863 , 5.7308345,
       5.3967924, 5.7516875, 5.9448757, 5.4374285, 5.8928475, 5.541425 ,
       5.7082705, 5.711905 , 5.7733135, 5.4745584, 5.430108 , 5.934937 ,
       6.026533 , 5.744135 , 6.0653706, 5.8844104, 5.503555 , 5.269984 ,
       5.627707 , 5.509114 , 5.663281 , 5.824488 , 5.483779 , 5.6952305,
       5.354967 , 5.3659616, 5.7931933, 5.7845306, 5.776753 , 5.8938184,
       5.869105 , 5.582813 , 5.652052 , 5.744979 ], dtype=float32)
```

In [32]:

```
#accuracy_score(y1, y_test), 1-accuracy_score(y1, y_test)
y_pred = np.rint(y_pred)
print (f'y test {y_test}')
print (f'y pred {y_pred}')



print(f'\n\n Accuracy of XGBoost  = {accuracy_score(y_pred, y_test )}\n\n')
# 1-accuracy_score(y_pred, y_test)
```

```
y test 91     6
968    5
755    6
796    6
848    5
      ..
119    6
985    6
278    5
624    5
958    6
Name: Result, Length: 250, dtype: int64
y pred [6. 6. 6. 6. 5. 6. 6. 6. 6. 6. 6. 5. 6. 6. 5. 6. 5. 6. 6. 6. 6. 6. 6. 5.
 6. 6. 6. 6. 6. 6. 6. 6. 6. 5. 6. 6. 6. 6. 6. 6. 6. 5. 6. 6. 5. 5. 6. 6.
 6. 6. 6. 6. 5. 6. 6. 6. 6. 6. 5. 6. 6. 5. 6. 6. 6. 6. 5. 6. 5. 6. 6. 6.
 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 5. 6.
 5. 6. 6. 5. 6. 5. 5. 5. 6. 6. 6. 6. 6. 5. 5. 6. 6. 6. 5. 6. 6. 6. 6. 6.
 6. 5. 5. 6. 5. 5. 5. 5. 6. 5. 6. 6. 6. 5. 6. 5. 6. 6. 6. 6. 6. 6. 5. 5.
 6. 6. 6. 5. 6. 6. 6. 6. 6. 5. 5. 6. 5. 6. 6. 5. 6. 6. 6. 6. 6. 5. 6. 5.
 6. 6. 6. 5. 5. 6. 6. 6. 6. 6. 6. 6. 6. 5. 6. 6. 5. 6. 6. 5. 5. 5. 6. 6.
 5. 6. 5. 6. 5. 6. 6. 6. 6. 6. 6. 6. 5. 5. 6. 6. 6. 6. 5. 5. 5. 6. 6. 6.
 5. 6. 6. 5. 6. 6. 6. 6. 6. 5. 5. 6. 6. 6. 6. 6. 6. 5. 6. 6. 6. 6. 5. 6.
 5. 5. 6. 6. 6. 6. 6. 6. 6. 6.]


 Accuracy of XGBoost  = 0.748
```

In [33]:

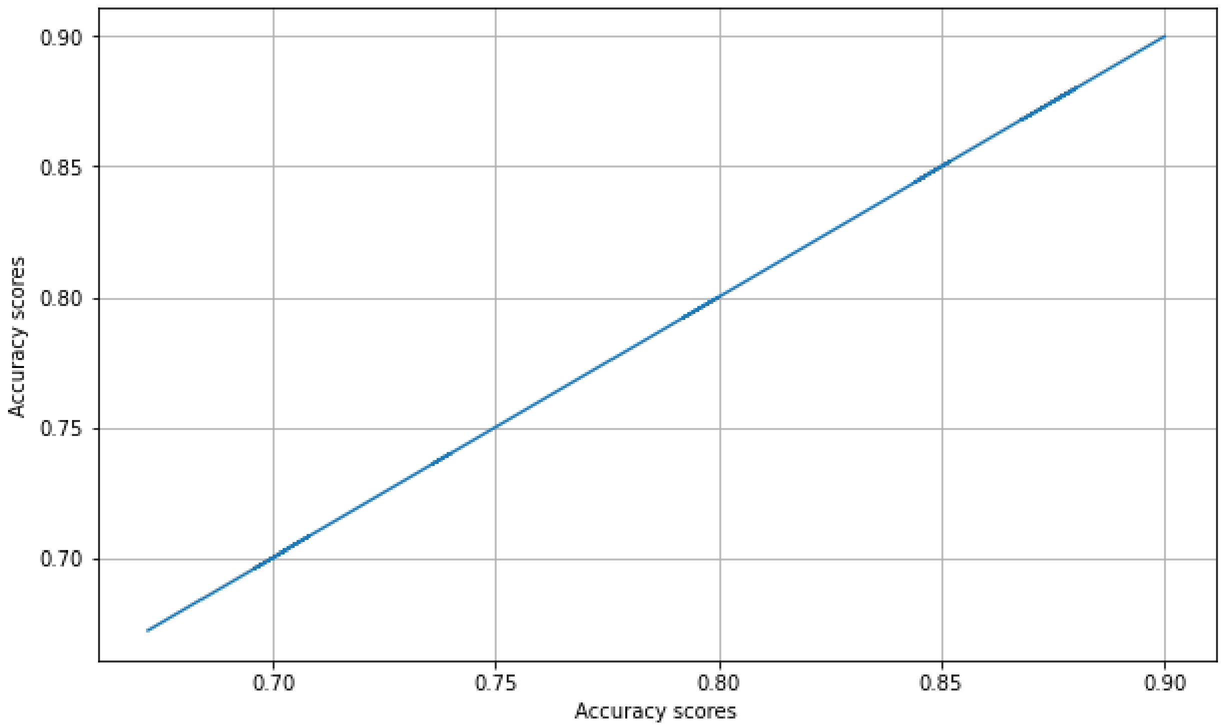
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import accuracy_score

print(f' x len {len(x_test)}')
print(f' y len {len(y_test)}')

#acc_scores = [ a for a in accuracy_score(y_test,y_pred)]
```

```
#acc_scores = [accuracy_score(y_train, clf.predict(X)) for clf in clfs]

print(f' accuracy = {acc_scores}')
plt.figure(figsize=(10,  6))
plt.grid()
#plt.plot(ccp_alphas[:-1], acc_scores[:-1])
plt.plot( acc_scores[:-1], acc_scores[:-1])
plt.xlabel("Accuracy scores")
plt.ylabel("Accuracy scores")
```

```
 x len 250
 y len 250
 accuracy = [0.9, 0.9, 0.9, 0.9, 0.896, 0.896, 0.896, 0.896, 0.896, 0.896, 0.884, 0.884, 0.884, 0.884, 0.884,
0.88, 0.88, 0.88, 0.88, 0.876, 0.876, 0.876, 0.872, 0.872, 0.872, 0.872, 0.872, 0.876, 0.88, 0.876, 0.88, 0.88,
0.88, 0.876, 0.876, 0.868, 0.872, 0.872, 0.872, 0.86, 0.852, 0.844, 0.844, 0.852, 0.852, 0.848, 0.848, 0.844,
0.844, 0.844, 0.84, 0.832, 0.828, 0.828, 0.828, 0.82, 0.8, 0.8, 0.792, 0.8, 0.8, 0.796, 0.792, 0.776, 0.764, 0.
76, 0.736, 0.74, 0.72, 0.716, 0.7, 0.696, 0.7, 0.708, 0.672, 0.672, 0.648]
```

Out[33]:

```
Text(0, 0.5, 'Accuracy scores')
```



In [ ]:

In [ ]:

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```