CART

In [1]:
```
#IMPORT PYTHON MODULES

from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
#This module needs to be separately invoked from tree library
from sklearn.model_selection import train_test_split,StratifiedKFold
#Stratified K-fold is a variation of KFold that returns stratified folds. The folds
from sklearn.metrics import confusion_matrix
#from sklearn.tree import export_graphviz
import pandas as pd
import numpy as np
#While pandas adopts many coding idioms from NumPy, the biggest difference is that p
import graphviz
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification #for bootstrapping
from sklearn.model_selection import KFold
#import category_encoders as ce
```

In [2]:
```
#READ AND EXPLORE DATA
#Use pandas read_csv function to upload Bikebuyer data from the computer folder (Mac
#Alternatively upload data file to jupyter homepage and use pd.read_csv("Bikebuyer.c
dtf=pd.read_csv("car_data.csv")
```

In [3]:
```
dtf['buying'],_ = pd.factorize(dtf['buying'])
dtf['maintenance'],_ = pd.factorize(dtf['maintenance'])
dtf['doors'],_ = pd.factorize(dtf['doors'])
dtf['persons'],_ = pd.factorize(dtf['persons'])
dtf['lug_boot'],_ = pd.factorize(dtf['lug_boot'])
dtf['safety'],_ = pd.factorize(dtf['safety'])
dtf
```

Out[3]:

	buying	maintenance	doors	persons	lug_boot	safety	car_evaluation
0	0	0	0	0	0	0	unacc
1	0	0	0	0	0	1	unacc
2	0	0	0	0	0	2	unacc
3	0	0	0	0	1	0	unacc
4	0	0	0	0	1	1	unacc
...	...	...	...	...	...	...	...
1723	3	3	3	2	1	1	good
1724	3	3	3	2	1	2	vgood
1725	3	3	3	2	2	0	unacc
1726	3	3	3	2	2	1	good
1727	3	3	3	2	2	2	vgood

1728 rows × 7 columns

In [4]:
```
#Extract relevant features from data
X=dtf[['buying','maintenance','doors','persons','lug_boot','safety']]
#'MaritalStatus' not included as scikit learn CART algorithm doesnt support categori
#type(X)
#np.shape(X) # dimension of dataframe
#X.dtypes
X.head()
```

Out[4]:

	buying	maintenance	doors	persons	lug_boot	safety
0	0	0	0	0	0	0
1	0	0	0	0	0	1
2	0	0	0	0	0	2
3	0	0	0	0	1	0
4	0	0	0	0	1	1

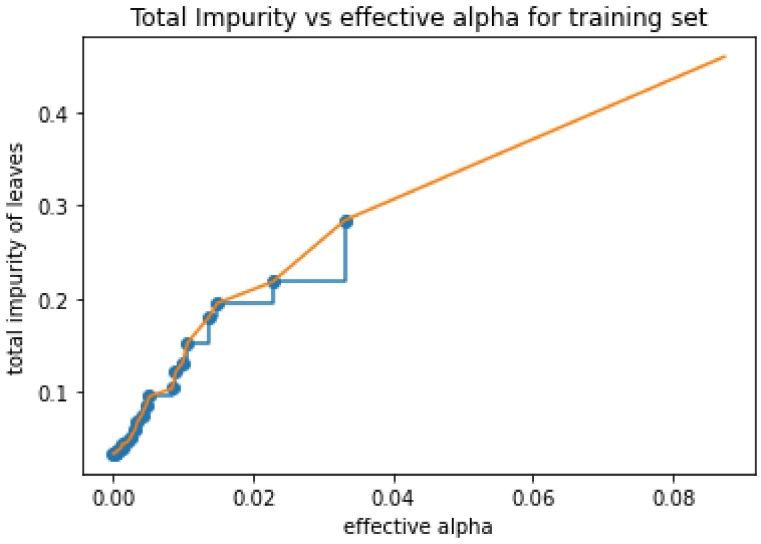
In [5]:
```
#DATA PREPARATION
#scikit-learn uses an optimised version of the CART algorithm; however, scikit-learn
#Assign Bikebuyer as target variable y
y=dtf['car_evaluation']
#Split X into train and test data: https://scikit-learn.org/stable/modules/generated
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_sta
```

In [6]:
```
#MODEL BUILDING
#Specify decision tree model as dt using scikit learn "DecisionTreeClassifier" modul
dt=DecisionTreeClassifier(criterion='gini',
    min_samples_leaf=5 ,max_depth=10,
    ccp_alpha=0.0)
#Fit the model using fit() method
dt.fit(X_train, y_train)
```

Out[6]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=10, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=5, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

In [7]:
```
#TREE PRUNING USING CCP ALPHA
#Determining Cost Complexity Parameter (ccp_alpha) for post pruning the tree: https:
path = dt.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities
#Using matplotlib.pyplot to plot the effect of varying ccp_alpha on error
fig, ax = plt.subplots()
ax.plot(ccp_alphas[:-1],impurities[:-1],marker='o', drawstyle="steps-post")
ax.set_xlabel("effective alpha")
ax.set_ylabel("total impurity of leaves")
ax.set_title("Total Impurity vs effective alpha for training set")
plt.plot(ccp_alphas, impurities)

dts = [] #tree building
for ccp_alpha in ccp_alphas:
    dt = DecisionTreeClassifier(random_state=0, ccp_alpha=ccp_alpha)
    dt.fit(X_train, y_train)
    dts.append(dt)
```



In [8]:
```
dts
```

Out[8]:
```
[DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                        max_depth=None, max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_split=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort='deprecated',
                        random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.00012004225487371664, class_weight=None,
                        criterion='gini', max_depth=None, max_features=None,
                        max_leaf_nodes=None, min_impurity_decrease=0.0,
                        min_impurity_split=None, min_samples_leaf=1,
                        min_samples_split=2, min_weight_fraction_leaf=0.0,
                        presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.00022224966045190832, class_weight=None,
                        criterion='gini', max_depth=None, max_features=None,
                        max_leaf_nodes=None, min_impurity_decrease=0.0,
                        min_impurity_split=None, min_samples_leaf=1,
                        min_samples_split=2, min_weight_fraction_leaf=0.0,
                        presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.0004258890433780514, class_weight=None,
                        criterion='gini', max_depth=None, max_features=None,
                        max_leaf_nodes=None, min_impurity_decrease=0.0,
                        min_impurity_split=None, min_samples_leaf=1,
                        min_samples_split=2, min_weight_fraction_leaf=0.0,
                        presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.0004390116749667314, class_weight=None,
                        criterion='gini', max_depth=None, max_features=None,
                        max_leaf_nodes=None, min_impurity_decrease=0.0,
                        min_impurity_split=None, min_samples_leaf=1,
                        min_samples_split=2, min_weight_fraction_leaf=0.0,
                        presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.0005041774704696064, class_weight=None,
                        criterion='gini', max_depth=None, max_features=None,
                        max_leaf_nodes=None, min_impurity_decrease=0.0,
                        min_impurity_split=None, min_samples_leaf=1,
                        min_samples_split=2, min_weight_fraction_leaf=0.0,
                        presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.0008544825960556308, class_weight=None,
                        criterion='gini', max_depth=None, max_features=None,
                        max_leaf_nodes=None, min_impurity_decrease=0.0,
                        min_impurity_split=None, min_samples_leaf=1,
                        min_samples_split=2, min_weight_fraction_leaf=0.0,
                        presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.0009383874552413869, class_weight=None,
                        criterion='gini', max_depth=None, max_features=None,
                        max_leaf_nodes=None, min_impurity_decrease=0.0,
                        min_impurity_split=None, min_samples_leaf=1,
```

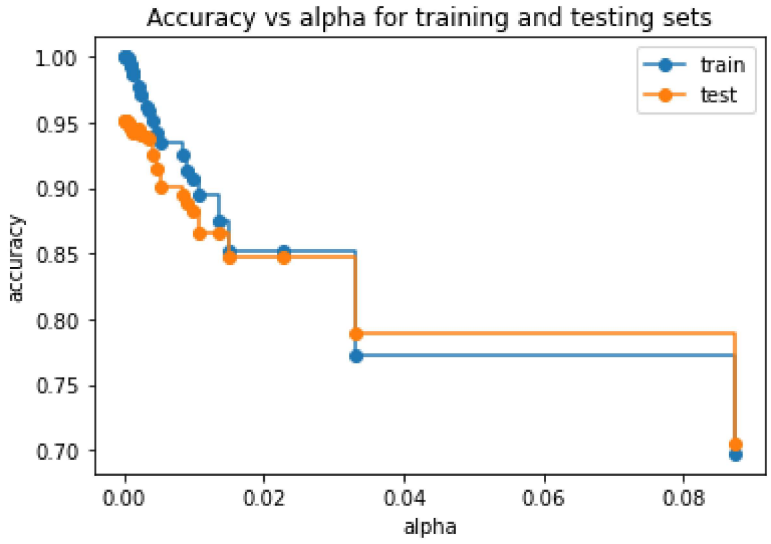
```
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.0011853315224101737, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.0012021686179898955, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.0012964563526361278, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.0020285728811835889, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.0024662747221087776, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.0031587376821157163, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.0034212042639008935, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.0040873830312728280, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.0046376235743185610, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.0051595204990417505, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.0084214856622674940, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.0089159319965220040, class_weight=None,
```

```
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.009910255327168393, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.010616975100101064, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.013679692722193944, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.014853478525823582, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.022731378437123598, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.033175884178617304, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best'),
 DecisionTreeClassifier(ccp_alpha=0.08738102722582908, class_weight=None,
                       criterion='gini', max_depth=None, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       presort='deprecated', random_state=0, splitter='best')]
```

In [9]:

```
#EVALUATION OF ALPHA BASED ON TRAIN AND TEST ERRORS

#Evaluates prediction accuracy and plots it against ccp_alphas
train_scores = [dt.score(X_train, y_train) for dt in dts]
test_scores = [dt.score(X_test, y_test) for dt in dts]
fig, ax = plt.subplots()
ax.set_xlabel("alpha")
ax.set_ylabel("accuracy")
ax.set_title("Accuracy vs alpha for training and testing sets")
ax.plot(ccp_alphas, train_scores, marker='o', label="train",
        drawstyle="steps-post")
ax.plot(ccp_alphas, test_scores, marker='o', label="test",
        drawstyle="steps-post")
ax.legend()
plt.show()
```



In [10]:

```
#FINAL MODEL BUILDING
dt=DecisionTreeClassifier(criterion='gini',
    min_samples_leaf=5 ,max_depth=10,
    ccp_alpha=0.0025)
dt.fit(X_train, y_train)
```

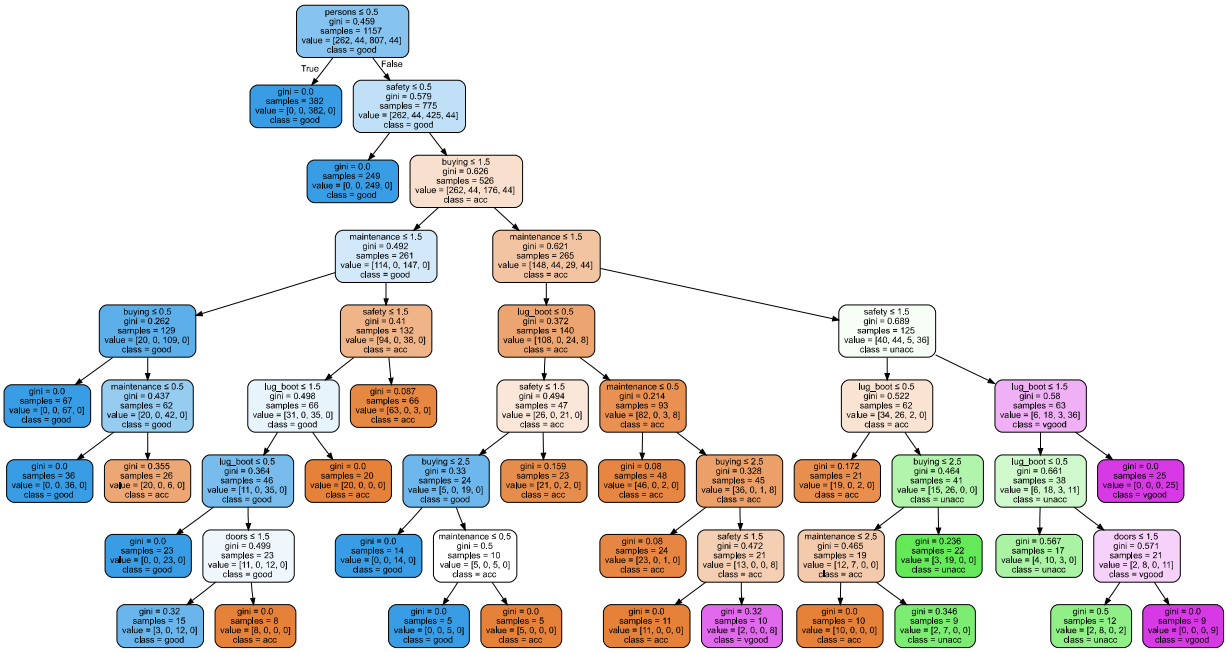
Out[10]:

```
DecisionTreeClassifier(ccp_alpha=0.0025, class_weight=None, criterion='gini',
                       max_depth=10, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=5, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

In [11]:

```
#VISUALIZING THE TREE USING GRAPHVIZ
dot_data = tree.export_graphviz(dt, out_file=None,feature_names=X_train.columns,clas
graph = graphviz.Source(dot_data)
graph
#Scikit learn CART algorithm treats ordinal data also as integers and performs divis
```

Out[11]:



In [12]:

```
#MODEL PREDICTION AND PERFORMANCE
y_pred=dt.predict(X_test)
from sklearn.metrics import classification_report
cmtx = confusion_matrix(y_test, y_pred)
```

```
cmtx
#print(classification_report(y_test, y_pred, target_names=['Buyer','Nonbuyer']))
```

Out[12]:
```
array([[112,   8,   1,   1],
       [  0,  25,   0,   0],
       [ 18,   1, 384,   0],
       [  0,   2,   0,  19]], dtype=int64)
```

In [13]:
```
def ConfMat(ranCM):
    ranCM = pd.DataFrame(ranCM,columns=["acc","unacc", "good","vgood"], index=["acc"
    FP = ranCM.sum(axis=0) - np.diag(ranCM)
    FN = ranCM.sum(axis=1) - np.diag(ranCM)
    TP = np.diag(ranCM)
    TN = ranCM.values.sum() - (FP + FN + TP)

    P = TP + FN
    N = FP + TN

    # Sensitivity,
    TPR = TP/(TP+FN)
    # Specificity
    TNR = TN/(TN+FP)
    # Overall accuracy
    ACC = (TP+TN)/(TP+FP+FN+TN)
    # error
    ERR = (FP+FN)/(TP+FP+FN+TN)
    TP = pd.DataFrame(TP,index=["acc","unacc", "good","vgood"],columns = [4])
    Output =pd.concat({'FP':FP,'FN':FN,'TN':TN,'TP':TP,'P':P,'N':N,'TNR':TNR,'ACC':A

    return print(Output)
ConfMat(cmtx)
```

```
        FP  FN   TN   TP    P    N       TNR       ACC       TPR       ERR
         0   1    2    4    3    4         5         6         7         8
acc     18  10  431  112  122  449  0.959911  0.950963  0.918033  0.049037
unacc   11   0  535   25   25  546  0.979853  0.980736  1.000000  0.019264
good     1  19  167  384  403  168  0.994048  0.964974  0.952854  0.035026
vgood    1   2  549   19   21  550  0.998182  0.994746  0.904762  0.005254
```

Bagging

In [14]:
```
#Bagging Classifier

from sklearn.ensemble import ExtraTreesClassifier

model = ExtraTreesClassifier(50 , random_state= 10)

model = model.fit(X_train , y_train)

y_pred = model.predict(X_test)
```

In [15]:
```
from sklearn.metrics import confusion_matrix , accuracy_score , classification_repor

cfm = confusion_matrix(y_test , y_pred)
print(cfm)
print("Classification report :")

print(classification_report(y_test , y_pred))

acc = accuracy_score(y_test , y_pred)
print("Accuracy of the model:" , acc)
```

```
[[116   2   4   0]
 [  4  20   0   1]
 [  7   0 396   0]
 [  2   0   0  19]]
Classification report :
              precision    recall  f1-score   support

         acc       0.90      0.95      0.92       122
        good       0.91      0.80      0.85        25
       unacc       0.99      0.98      0.99       403
       vgood       0.95      0.90      0.93        21

    accuracy                           0.96       571
   macro avg       0.94      0.91      0.92       571
weighted avg       0.97      0.96      0.96       571

Accuracy of the model: 0.9649737302977233
```

In [16]:

```
def ConfMat(ranCM):
    ranCM = pd.DataFrame(ranCM,columns=["acc","unacc", "good","vgood"], index=["acc"
    FP = ranCM.sum(axis=0) - np.diag(ranCM)
    FN = ranCM.sum(axis=1) - np.diag(ranCM)
    TP = np.diag(ranCM)
    TN = ranCM.values.sum() - (FP + FN + TP)

    P = TP + FN
    N = FP + TN

    # Sensitivity,
    TPR = TP/(TP+FN)
    # Specificity
    TNR = TN/(TN+FP)
    # Overall accuracy
    ACC = (TP+TN)/(TP+FP+FN+TN)
    # error
    ERR = (FP+FN)/(TP+FP+FN+TN)
    TP = pd.DataFrame(TP,index=["acc","unacc", "good","vgood"],columns = [4])
    Output =pd.concat({'FP':FP,'FN':FN,'TN':TN,'TP':TP,'P':P,'N':N,'TNR':TNR,'ACC':A

    return print(Output)
ConfMat(cfm)
```

```
        FP FN   TN   TP    P    N       TNR       ACC       TPR       ERR
         0  1    2    4    3    4         5         6         7         8
acc      13  6  436  116  122  449  0.971047  0.966725  0.950820  0.033275
unacc     2  5  544   20   25  546  0.996337  0.987741  0.800000  0.012259
good      4  7  164  396  403  168  0.976190  0.980736  0.982630  0.019264
vgood     1  2  549   19   21  550  0.998182  0.994746  0.904762  0.005254
```

Boosting

In [17]:

```
#boosting - Ada boost - generic boosting algo, less used for DT. can be used for oth

from sklearn.ensemble import AdaBoostClassifier

model_Ada = AdaBoostClassifier(base_estimator=DecisionTreeClassifier() ,n_estimators

model_Ada = model_Ada.fit(X_train , y_train)

y_pred = model_Ada.predict(X_test)
```

In [18]:

```
from sklearn.metrics import confusion_matrix , accuracy_score , classification_repor

cfmt = confusion_matrix(y_test , y_pred)
print(cfmt)
print("Classification report :")

print(classification_report(y_test , y_pred))

acc = accuracy_score(y_test , y_pred)
print("Accuracy of the model:" , acc)
```

```
[[108   3  10   1]
 [  5  20   0   0]
 [  8   0 395   0]
 [  0   1   0  20]]
Classification report :
              precision    recall  f1-score   support

         acc       0.89      0.89      0.89       122
        good       0.83      0.80      0.82        25
       unacc       0.98      0.98      0.98       403
       vgood       0.95      0.95      0.95        21

    accuracy                           0.95       571
   macro avg       0.91      0.90      0.91       571
weighted avg       0.95      0.95      0.95       571

Accuracy of the model: 0.9509632224168126
```

In [19]:

```
def ConfMat(ranCM):
    ranCM = pd.DataFrame(ranCM,columns=["acc","unacc", "good","vgood"], index=["acc"
    FP = ranCM.sum(axis=0) - np.diag(ranCM)
    FN = ranCM.sum(axis=1) - np.diag(ranCM)
    TP = np.diag(ranCM)
    TN = ranCM.values.sum() - (FP + FN + TP)

    P = TP + FN
    N = FP + TN

    # Sensitivity,
    TPR = TP/(TP+FN)
    # Specificity
    TNR = TN/(TN+FP)
    # Overall accuracy
    ACC = (TP+TN)/(TP+FP+FN+TN)
    # error
    ERR = (FP+FN)/(TP+FP+FN+TN)
    TP = pd.DataFrame(TP,index=["acc","unacc", "good","vgood"],columns = [4])
    Output =pd.concat({'FP':FP,'FN':FN,'TN':TN,'TP':TP,'P':P,'N':N,'TNR':TNR,'ACC':A

    return print(Output)
ConfMat(cfmt)
```

```
         FP  FN   TN   TP    P    N       TNR       ACC       TPR       ERR
          0   1    2    4    3    4         5         6         7         8
acc      13  14  436  108  122  449  0.971047  0.952715  0.885246  0.047285
unacc     4   5  542   20   25  546  0.992674  0.984238  0.800000  0.015762
good     10   8  158  395  403  168  0.940476  0.968476  0.980149  0.031524
vgood     1   1  549   20   21  550  0.998182  0.996497  0.952381  0.003503
```

Random forest

In [20]:

```
#RANDOM FOREST
```

```
X, y = make_classification(n_samples=1000, n_features=4,
                           n_informative=2, n_redundant=0,
                           random_state=0, shuffle=False)
#Generates 1000 subsamples for tree
dt=RandomForestClassifier(n_estimators=5, criterion='gini',
                          min_samples_leaf=5 ,max_depth=10,
                          ccp_alpha=0.0)
dt.fit(X_train, y_train)
y_pred=dt.predict(X_test)
from sklearn.metrics import classification_report
cmtx_rn = confusion_matrix(y_test, y_pred)
cmtx_rn
```

Out[20]:
```
array([[115,   4,   3,   0],
       [ 11,  13,   0,   1],
       [ 15,   0, 388,   0],
       [  5,   1,   0,  15]], dtype=int64)
```

In [21]:
```
print(classification_report(y_test, y_pred, target_names=['acc','unacc','good','vgoo
```

```
              precision    recall  f1-score   support

         acc       0.79      0.94      0.86       122
       unacc       0.72      0.52      0.60        25
        good       0.99      0.96      0.98       403
       vgood       0.94      0.71      0.81        21

    accuracy                           0.93       571
   macro avg       0.86      0.78      0.81       571
weighted avg       0.93      0.93      0.93       571
```

In [22]:
```
def ConfMat(ranCM):
    ranCM = pd.DataFrame(ranCM,columns=["acc","unacc", "good","vgood"], index=["acc"
    FP = ranCM.sum(axis=0) - np.diag(ranCM)
    FN = ranCM.sum(axis=1) - np.diag(ranCM)
    TP = np.diag(ranCM)
    TN = ranCM.values.sum() - (FP + FN + TP)

    P = TP + FN
    N = FP + TN

    # Sensitivity,
    TPR = TP/(TP+FN)
    # Specificity
    TNR = TN/(TN+FP)
    # Overall accuracy
    ACC = (TP+TN)/(TP+FP+FN+TN)
    # error
    ERR = (FP+FN)/(TP+FP+FN+TN)
    TP = pd.DataFrame(TP,index=["acc","unacc", "good","vgood"],columns = [4])
    Output =pd.concat({'FP':FP,'FN':FN,'TN':TN,'TP':TP,'P':P,'N':N,'TNR':TNR,'ACC':A

    return print(Output)
ConfMat(cmtx_rn)
```

```
         FP  FN   TN   TP    P    N       TNR       ACC       TPR       ERR
          0   1    2    4    3    4         5         6         7         8
acc      31   7  418  115  122  449  0.930958  0.933450  0.942623  0.066550
unacc     5  12  541   13   25  546  0.990842  0.970228  0.520000  0.029772
good      3  15  165  388  403  168  0.982143  0.968476  0.962779  0.031524
vgood     1   6  549   15   21  550  0.998182  0.987741  0.714286  0.012259
```

In [23]:
```
'''''#CROSS VALIDATION

#simple validation set approach: https://scikit-learn.org/stable/modules/model_evalu
#by default splits into five folds
from sklearn import model_selection
from sklearn.model_selection import cross_validate
from sklearn.metrics import recall_score
scores = cross_validate(dt, X_test, y_test)
k=scores['test_score']
k
'''
```

Out[23]:
```
"''#CROSS VALIDATION\n\n#simple validation set approach: https://scikit-learn.org/st
able/modules/model_evaluation.html#scoring-parameter  \n#by default splits into five
folds\nfrom sklearn import model_selection\nfrom sklearn.model_selection import cros
s_validate\nfrom sklearn.metrics import recall_score\nscores = cross_validate(dt, X_
test, y_test)\nk=scores['test_score']\nk\n"
```

In [24]:
```
''' #K-fold

kf = KFold(n_splits=500)
scores=[]
for train_index, test_index in kf.split(X,y):
    X_train, X_test, y_train, y_test = X[train_index], X[test_index],y[train_index],
    dt.fit(X_train, y_train)
    scores.append(dt.score(X_test, y_test))
scores
'''
```

Out[24]:
```
' #K-fold\n\nkf = KFold(n_splits=500)\nscores=[]\nfor train_index, test_index in kf.
split(X,y):\n    X_train, X_test, y_train, y_test = X[train_index], X[test_index],y
[train_index], y[test_index]\n    dt.fit(X_train, y_train)\n    scores.append(dt.sco
re(X_test, y_test)) \nscores\n'
```

In [ ]:

In [ ]: