Q1 - a,b

In [1]:
```
from sklearn.model_selection import train_test_split
import pandas as pd
cars = pd.read_csv("claim_history.csv")
train, test = train_test_split(cars, test_size=0.25, random_state=60616)
```

In [2]:
```
print(len(train))
print(len(test))
```

```
7726
2576
```

In [3]:
```
print(len(train[train['CAR_USE']=='Commercial']))
print(len(train[train['CAR_USE']=='Commercial'])/len(train))
print(len(train[train['CAR_USE']=='Private']))
print(len(train[train['CAR_USE']=='Private'])/len(train))
```

```
2851
0.3690137199068082
4875
0.6309862800931918
```

In [4]:
```
print(len(test[test['CAR_USE']=='Commercial']))
print(len(test[test['CAR_USE']=='Commercial'])/len(test))
print(len(test[test['CAR_USE']=='Private']))
print(len(test[test['CAR_USE']=='Private'])/len(test))
print(len(train[train['CAR_USE']=='Commercial']))
```

```
938
0.3641304347826087
1638
0.6358695652173914
2851
```

Q1 - c,d

In [5]:
```
print("Probability that the Car_use in train is commercial:")
tot_commercial = len(train[train['CAR_USE']=='Commercial']) + len(test[test['C
AR_USE']=='Commercial'])
commercial_train_prob = len(train[train['CAR_USE']=='Commercial'])/tot_commerc
ial
print(round(commercial_train_prob,2))
print("Probability that the Car_use in test is Private:")
tot_private = len(test[test['CAR_USE']=='Private']) + len(train[train['CAR_US
E']=='Private'])
private_test_prob = len(test[test['CAR_USE']=='Private'])/tot_private
print(round(private_test_prob,2))
```

```
Probability that the Car_use in train is commercial:
0.75
Probability that the Car_use in test is Private:
0.25
```

Q2 - a

In [6]:
```
from math import log
private_train_prob = len(train[train['CAR_USE']=='Private'])/tot_commercial
print(round(private_train_prob,2))
root_entropy = -1 * ((commercial_train_prob)*log(commercial_train_prob,2) + (p
rivate_train_prob)*log(private_train_prob,2))
print("\n\nEntropy of root node is " + str(root_entropy))
```

```
1.29


Entropy of root node is -0.1590319278104924
```

Q2 - b

In [7]:
```
def calcEntropy(total, commercial, private):
    p_commercial = commercial/total
    p_private = private/total
    log_p_commercial = log(p_commercial,2) if p_commercial != 0 else 0
    log_p_private = log(p_private,2) if p_private != 0 else 0
    entropy = -1*(p_commercial * log_p_commercial + p_private * log_p_private)
    return entropy
```

In [8]:
```
import itertools
import pandas as pd
import numpy as np
import math
from math import log
import sklearn.metrics as metrics

def allPossibleSets(S, varType):
    if varType == "Nominal":
            relS=set()
            n = len(S)
            k=int(n/2)
            for i in range(1,k):
                relS.update(set(itertools.combinations(S, i)))
            kth_subset = set(itertools.combinations(S, k))
            kth_subset = set(itertools.islice(kth_subset, int(len(kth_subset)/
2)))
            relS.update(kth_subset)
            return relS
    elif varType == "Ordinal":
        relL = []
        n=len(S)
        for i in range(1,n):
            relL.append(set(itertools.islice(S, i)))
        relS = set(frozenset(i) for i in relL)
        return [list(x) for x in relS]
```

In [9]:
```
def EntropyPredictor(data, pred, varType):
    possibleSets = allPossibleSets(set(pred), varType)
    commercialTotal = len(data[data['CAR_USE'] == 'Commercial'])
    privateTotal = len(data[data['CAR_USE'] == 'Private'])
    nTotal = commercialTotal + privateTotal
    entropyList=[]
    for pset in possibleSets:
        filtData = data[pred.isin(pset)]
        total = len(filtData)
        commercial = len(filtData[filtData['CAR_USE'] == 'Commercial'])
        private = len(filtData[filtData['CAR_USE'] == 'Private'])
        entropy = calcEntropy(total, commercial, private)
        entropy2 = calcEntropy((nTotal - total), (commercialTotal - commercial
), (privateTotal - private))
        tt = nTotal - total
        splitEntropy = (total/nTotal)*entropy + (tt/nTotal)*entropy2
        entropyList.append([pset, splitEntropy])
    splitEntropys = np.array(entropyList)[:,1]
    minSplitEntropy = min(splitEntropys)
    return entropyList[np.where(splitEntropys == minSplitEntropy)[0][0]]
```

In [10]:
```
def SplitCondition(node):
    car_type_entropy = EntropyPredictor(node, node['CAR_TYPE'], "Nominal")
    occupation_entropy = EntropyPredictor(node, node['OCCUPATION'], "Nominal")
    education_entropy = EntropyPredictor(node, node['EDUCATION'], "Ordinal")
    min_Ent_Pred = [car_type_entropy, occupation_entropy, education_entropy]
    all_entropy = np.array(min_Ent_Pred)[:,1]
    splitCondition = min_Ent_Pred[np.where(all_entropy == min(all_entropy))[0]
[0]]
    return splitCondition
```

In [11]:
```
splitCondition = SplitCondition(train)
print("\n\nSplit condition is " + str(splitCondition[0]))
```

```
Split condition is ('Student', 'Blue Collar', 'Unknown')
```

In [12]:
```
True_node = train[(train['OCCUPATION'] == 'Blue Collar') | (train['OCCUPATION'
] == 'Unknown') | (train['OCCUPATION'] == 'Student') ]
```

In [13]:
```
False_node = train[~train.isin(True_node)].dropna()

print("True predictor name and values:")
print("Car Type: " + str(set(True_node['CAR_TYPE'])) + "\nOccupation: " + str(
set(True_node['OCCUPATION'])) + "\nEducation: " + str(set(True_node['EDUCATIO
N'])))

print("False predictor name and values: ")
print("Car Type: " + str(set(False_node['CAR_TYPE'])) + "\nOccupation: " + str
(set(False_node['OCCUPATION'])) + "\nEducation: " + str(set(False_node['EDUCAT
ION'])))
```

```
True predictor name and values:
Car Type: {'Van', 'Panel Truck', 'Minivan', 'Sports Car', 'SUV', 'Pickup'}
Occupation: {'Blue Collar', 'Unknown', 'Student'}
Education: {'Below High School', 'High School', 'Doctors', 'Bachelors', 'Mast
ers'}
False predictor name and values:
Car Type: {'Van', 'Panel Truck', 'Minivan', 'Sports Car', 'SUV', 'Pickup'}
Occupation: {'Professional', 'Home Maker', 'Manager', 'Doctor', 'Lawyer', 'Cl
erical'}
Education: {'Below High School', 'High School', 'Doctors', 'Bachelors', 'Mast
ers'}
```

Q2 - c

In [14]:
```
print("\n\nSplit condition of First layer is " + str(splitCondition[1]))
```

```
Split condition of First layer is 0.7138723890228706
```

Q2 - d

In [15]:
```
split_Condition_True_Node =  SplitCondition(True_node)
print(split_Condition_True_Node)
split_Condition_False_Node =  SplitCondition(False_node)
print(split_Condition_False_Node)
print("The total no. of leaves = 4")
```

```
[['Below High School'], 0.6736439321725546]
[('Van', 'Panel Truck', 'Pickup'), 0.3293722168415045]
The total no. of leaves = 4
```

Q2 - e

In [16]:
```
nodeTT = True_node[ (True_node['EDUCATION'] == 'Below High School') ]
nodeTF = True_node[~True_node.isin(nodeTT)].dropna()
nodeFT = False_node[ (False_node['CAR_TYPE'] == 'Minivan') | (False_node['CAR_
TYPE'] == 'SUV') | (False_node['CAR_TYPE'] == 'Sports Car')]
nodeFF = False_node[~False_node.isin(nodeFT)].dropna()

def countTargetVals(node):
    print("Number of values where Car Use is Commercial " + str(len(node[node[
'CAR_USE'] == 'Commercial'])))
    print("Number of values where Car Use is Private " + str(len(node[node['CA
R_USE'] == 'Private'])))
    com = len(node[node['CAR_USE'] == 'Commercial'])
    pri = len(node[node['CAR_USE'] == 'Private'])
    total_value = com + pri
    return com / total_value

print("Condition True - True")
ptt = countTargetVals(nodeTT)

print("\nCondition True - False")
ptf = countTargetVals(nodeTF)

print("\nCondition False - True")
pft = countTargetVals(nodeFT)

print("\nCondition False - False")
pff = countTargetVals(nodeFF)
```

```
Condition True - True
Number of values where Car Use is Commercial 173
Number of values where Car Use is Private 460

Condition True - False
Number of values where Car Use is Commercial 994
Number of values where Car Use is Private 131

Condition False - True
Number of values where Car Use is Commercial 14
Number of values where Car Use is Private 2049

Condition False - False
Number of values where Car Use is Commercial 438
Number of values where Car Use is Private 397
```

Q2 - f

In [30]:
```
#Kolmogorov Smirnov cutoff
import numpy
threshold = float((cars.groupby('CAR_USE').size() / cars.shape[0])['Commercia
l'])
cutoff = numpy.where(threshold > 1.0, numpy.nan,threshold)
print(cutoff)
```

```
0.36779266161910307
```

Q3 - a,c,d

In [18]:
```
from sklearn.metrics import accuracy_score
```

In [19]:
```
testData = test[['CAR_TYPE', 'OCCUPATION', 'EDUCATION', 'CAR_USE']].dropna()
nodeTrueTest = test[ (test['OCCUPATION'] == 'Blue Collar') | (testData['OCCUPA
TION'] == 'Student') |
        (test['OCCUPATION'] == 'Unknown') ]
nodeFalseTest = test[~test.isin(nodeTrueTest)].dropna()
nodeTTtest = nodeTrueTest[ (nodeTrueTest['EDUCATION'] == 'Below High School')
]
nodeTFtest = nodeTrueTest[~nodeTrueTest.isin(nodeTTtest)].dropna()
nodeFTtest = nodeFalseTest[ (nodeFalseTest['CAR_TYPE'] == 'Minivan') | (nodeFa
lseTest['CAR_TYPE'] == 'SUV') | (nodeFalseTest['CAR_TYPE'] == 'Sports Car') ]
nodeFFtest = nodeFalseTest[~nodeFalseTest.isin(nodeFTtest)].dropna()

threshold = float((train.groupby('CAR_USE').size() / train.shape[0])['Commerci
al'])
print("Threshold is", threshold)
testData['Predicted_Probability'] = 0
nodeTTtest['Predicted_Probability'] = ptt
nodeTFtest['Predicted_Probability'] = ptf
nodeFTtest['Predicted_Probability'] = pft
nodeFFtest['Predicted_Probability'] = pff
leafNodes = pd.concat([nodeTTtest, nodeTFtest, nodeFTtest, nodeFFtest])
leafNodes.loc[leafNodes['Predicted_Probability'] >= threshold, 'Predicted_Clas
s'] = "Commercial"
leafNodes.loc[leafNodes['Predicted_Probability'] < threshold, 'Predicted_Clas
s'] = "Private"
```

```
Threshold is 0.3690137199068082

C:\Users\segar\Anaconda3\lib\site-packages\ipykernel_launcher.py:13: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  del sys.path[0]
C:\Users\segar\Anaconda3\lib\site-packages\ipykernel_launcher.py:15: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  from ipykernel import kernelapp as app
```

In [27]:

```
# Determine the predicted class of Y
Y = np.array(leafNodes['CAR_USE'].tolist())
nY = Y.shape[0]
predProbY = np.array(leafNodes['Predicted_Probability'].tolist())
predY = np.empty_like(Y)
for i in range(nY):
    if (predProbY[i] > 0.5):
        predY[i] = 'Commercial'
    else:
        predY[i] = 'Private'

# Calculate the Root Average Squared Error
RASE = 0.0
for i in range(nY):
    if (Y[i] == 'Commercial'):
        RASE += (1 - predProbY[i])**2
    else:
        RASE += (0 - predProbY[i])**2
RASE = np.sqrt(RASE/nY)

# Calculate the Root Mean Squared Error
Y_true = 1.0 * np.isin(Y, ['Commercial'])
RMSE = metrics.mean_squared_error(Y_true, predProbY)
RMSE = np.sqrt(RMSE)


AUC = metrics.roc_auc_score(Y_true, predProbY)
accuracy = metrics.accuracy_score(Y, predY)

print('                  Accuracy: {:.13f}' .format(accuracy))
print('    Misclassification Rate: {:.13f}' .format(1-accuracy))
print('          Area Under Curve: {:.13f}' .format(AUC))
print('Root Average Squared Error: {:.13f}' .format(RASE))
print('   Root Mean Squared Error: {:.13f}' .format(RMSE))
```

```
                  Accuracy: 0.8576094056172
    Misclassification Rate: 0.1423905943828
          Area Under Curve: 0.9302739346984
Root Average Squared Error: 0.3099907555814
   Root Mean Squared Error: 0.3099907555814
```

Q3 - e

In [29]:

```
gini_coeff = 2 * AUC -1
print("The gini coefficient is:", gini_coeff)
```

```
The gini coefficient is: 0.8605478693967901
```

Q3 - b

In [31]:
```
import numpy
threshold = float((train.groupby('CAR_USE').size() / train.shape[0])['Commerci
al'])
cutoff = numpy.where(threshold > 1.0, numpy.nan,threshold)
print(cutoff)
```

```
0.3690137199068082
```

In [32]:
```
print(1-cutoff)
```

```
0.6309862800931918
```

Q3 - f

In [33]:
```
predY_KS = numpy.empty_like(Y)
for i in range(nY):
    if (predProbY[i] > cutoff):
        predY_KS[i] = 'Commercial'
    else:
        predY_KS[i] = 'Private'
acc_KS = accuracy_score(Y,predY_KS)
```

In [35]:
```
print(acc_KS)
```

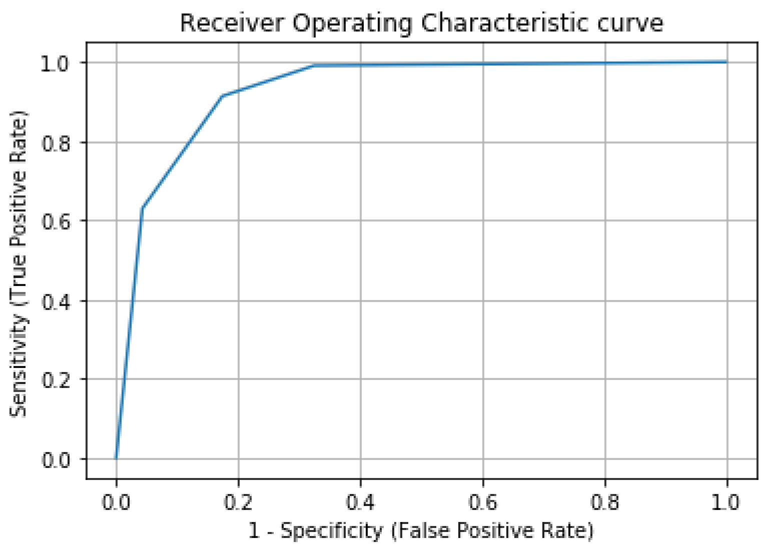
```
0.8576094056172436
```

In [36]:
```
mis_KS = 1-acc_KS
```

In [37]:
```
print(mis_KS)
```

```
0.14239059438275636
```

In [40]:
```
import matplotlib.pyplot as plt
Y = np.array(leafNodes['CAR_USE'].tolist())
OneMinusSpecificity, Sensitivity, thresholds = metrics.roc_curve(Y, predProbY,
pos_label = 'Commercial')
OneMinusSpecificity = np.append([0], OneMinusSpecificity)
Sensitivity = np.append([0], Sensitivity)
OneMinusSpecificity = np.append(OneMinusSpecificity, [1])
Sensitivity = np.append(Sensitivity, [1])
plt.plot(OneMinusSpecificity, Sensitivity)
plt.grid(True)
plt.xlabel("1 - Specificity (False Positive Rate)")
plt.ylabel("Sensitivity (True Positive Rate)")
plt.title("Receiver Operating Characteristic curve")
plt.show()
```



In [ ]: