STAT 574

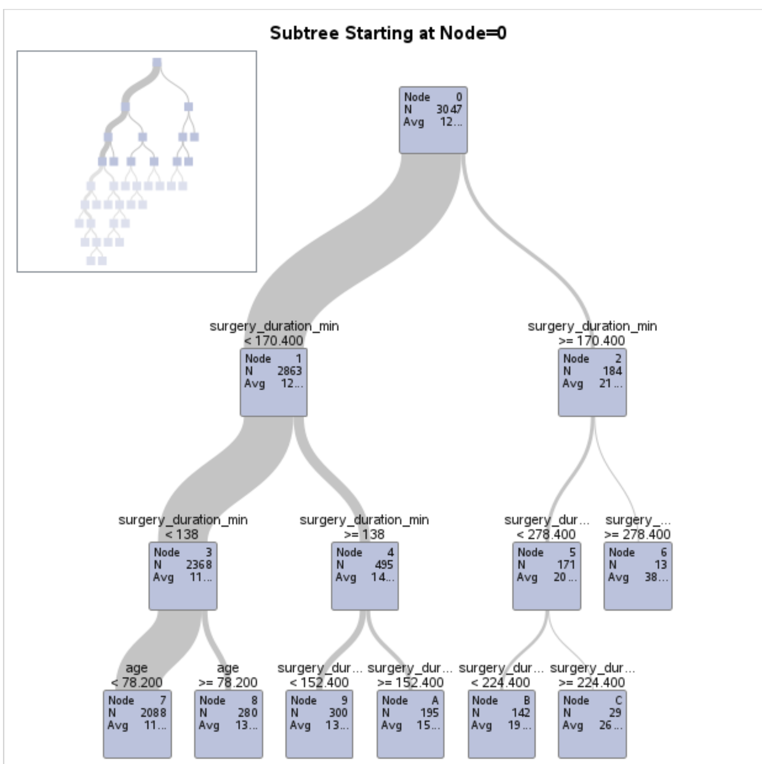
Derrick Edwards

Homework 1

Problem 1. The data file "hospital_data.csv" contains data on patients who underwent hip replacement surgery. The variables in the data set are: medical ID, gender, age, BMI, ASA score (explained below), surgery duration (in minutes), and surgery cost. The ASA (American Society of Anesthesiology) score is a metric to determine if someone is healthy enough to tolerate surgery and anesthesia ('1'=healthy, '2'=no significant functional limitations, '3'=significant functional limitations, '4'=constant threat to life).

(a) Split the data into 80% training and 20% testing sets and build a regression tree on the training set with the RSS splitting criterion to model surgery cost. Use all the other variables except medical ID as splitting variables. Apply the cost-complexity pruning algorithm to produce a reasonably-sized tree. Give the graphical output.

SAS:

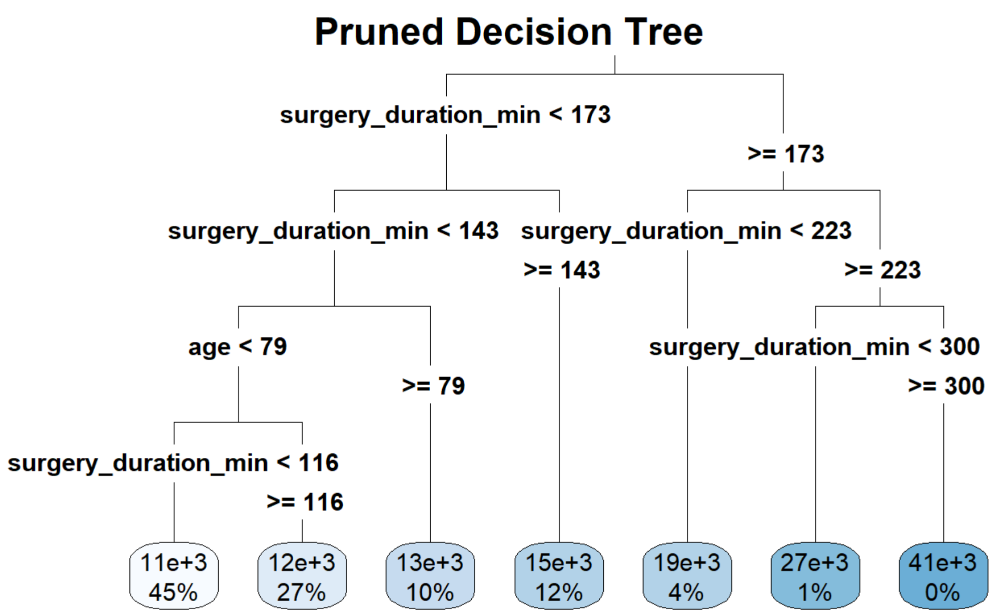


SAS CODE:

```
/* split data into 80% train 20% test */
proc surveyselect data=hospital rate=0.8 seed=192837
out=hospital outall method=srs;
```

```
run;
/*RSS SPLITTING AND COST-COMPLEXITY PRUNING*/
proc hpsplit data=hospital;
class gender;
model surgery_cost = gender age BMI ASA surgery_duration_min;
grow RSS;
prune costcomplexity;
partition rolevar=selected(train="1");
output out=predicted;
ID selected;
run;
```

R STUDIO

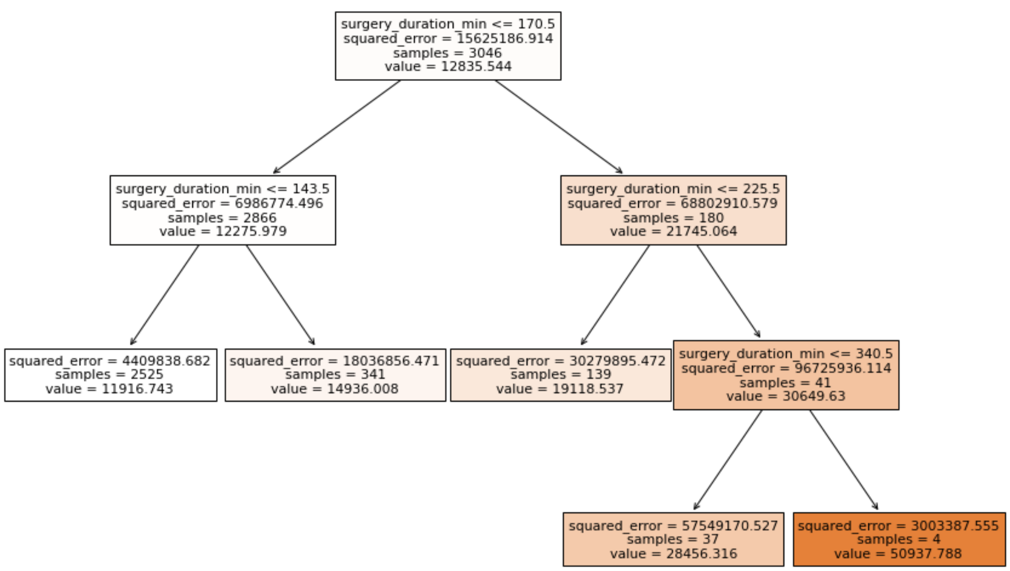


R Code:

```
# split into testing and training
set.seed(239076)
sample <- sample(c(TRUE, FALSE), nrow(hospital),
          replace = TRUE, prob = c(0.8, 0.2))
train <- hospital[sample,]
test <- hospital[!sample,]

# Apply RSS splitting Criterion to model surgery cost
library(rpart)
library(rpart.plot)
a1_tree <- rpart(surgery_cost ~ gender + age + BMI + ASA +
```

```
              surgery_duration_min, data=train,
             method = "anova", xval=10, cp=0)

printcp(a1_tree)

# plot initial decision tree
rpart.plot(a1_tree, type = 3,
        main="Initial Decision Tree")

# complexity Parameter Table Graph - find number optimal number of leaves
plotcp(a1_tree, minline = TRUE, upper = "size")
 # aprox - 7 splits

# reduced decision tree
a1_RSS <- rpart(surgery_cost ~ gender + age + BMI + ASA +
              surgery_duration_min, data = train,
            method = "anova", cp=0.0086)

# plot prunned tree
rpart.plot(a1_RSS, type=3,
        main = "Pruned Decision Tree")
```

Python



Python Code:

```python
# convert gender to binary
coding={'M': 1, 'F':0}
```

```python
hospital['gender']=hospital['gender'].map(coding)
# select independent and dependent variables
X=hospital.iloc[:,1:6].values
y=hospital.iloc[:,6].values


#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.20,
random_state=348644)

#FITTING REGRESSION TREE WITH RSS SPLITTING CRITERION
rtree = DecisionTreeRegressor(random_state=907420,
criterion="squared_error", max_leaf_nodes=5)
reg_tree_RSS = rtree.fit(X_train, y_train)

#plotting fitted tree
fig=plt.figure(figsize=(15,10))
fn=['gender', 'age', 'BMI', 'ASA', 'surgery_duration_min']
tree.plot_tree(reg_tree_RSS, feature_names=fn, filled=True)
```

(b) Use the fitted model to predict surgery cost for the testing data. Compute proportions of predicted values within 10%, 15%, and 20% of the observed values.

SAS

accuracy10	accuracy15	accuracy20
0.519054	1	1

SAS CODE:

```
/*COMPUTING PREDICTION ACCURACY FOR TESTING DATA*/
data test;
set predicted;
if(selected="0");
keep _leaf_ surgery_cost P_surgery_cost;
run;

data accuracy;
set test;
if(abs(surgery_cost-P_surgery_cost)<0.10*surgery_cost)
then ind10=1; else ind10=0;
if(abs(surgery_cost-Psurgery_cost)<0.15*surgery_cost)
then ind15=1; else ind15=0;
if(abs(surgery_cost-Psurgery_cost)<0.20*surgery_cost)
then ind20=1; else ind20=0;
```

run;

proc sql;
 select mean(ind10) as accuracy10, mean(ind15) as accuracy15,
 mean(ind20) as accuracy20
 from accuracy;
 quit;

R STUDIO

```
[1] 0.4872483
[1] 0.685906
[1] 0.8134228
```

R CODE:

```
# Compute prediction accuracy for testing data
pred_surg_cost <- predict(a1_RSS, newdata = test)

# accuracy within 10%
accuracy10 <- ifelse(abs(test$surgery_cost - pred_surg_cost) <
                 0.10*test$surgery_cost, 1, 0)
print(mean(accuracy10))

# accuracy withing 15%
accuracy15 <- ifelse(abs(test$surgery_cost - pred_surg_cost) <
                 0.15*test$surgery_cost, 1, 0)
print(mean(accuracy15))

# accuracy withing 20%
accuracy20 <- ifelse(abs(test$surgery_cost - pred_surg_cost) <
                 0.20*test$surgery_cost, 1, 0)
print(mean(accuracy20))
```

PYTHON

```
Accuracy Scores RSS Splitting Criterion Accuracy within 10%: 0.44750656167979
Accuracy within 15%: 0.6286089238845144 Accuracy within 20%: 0.7874015748031497
```

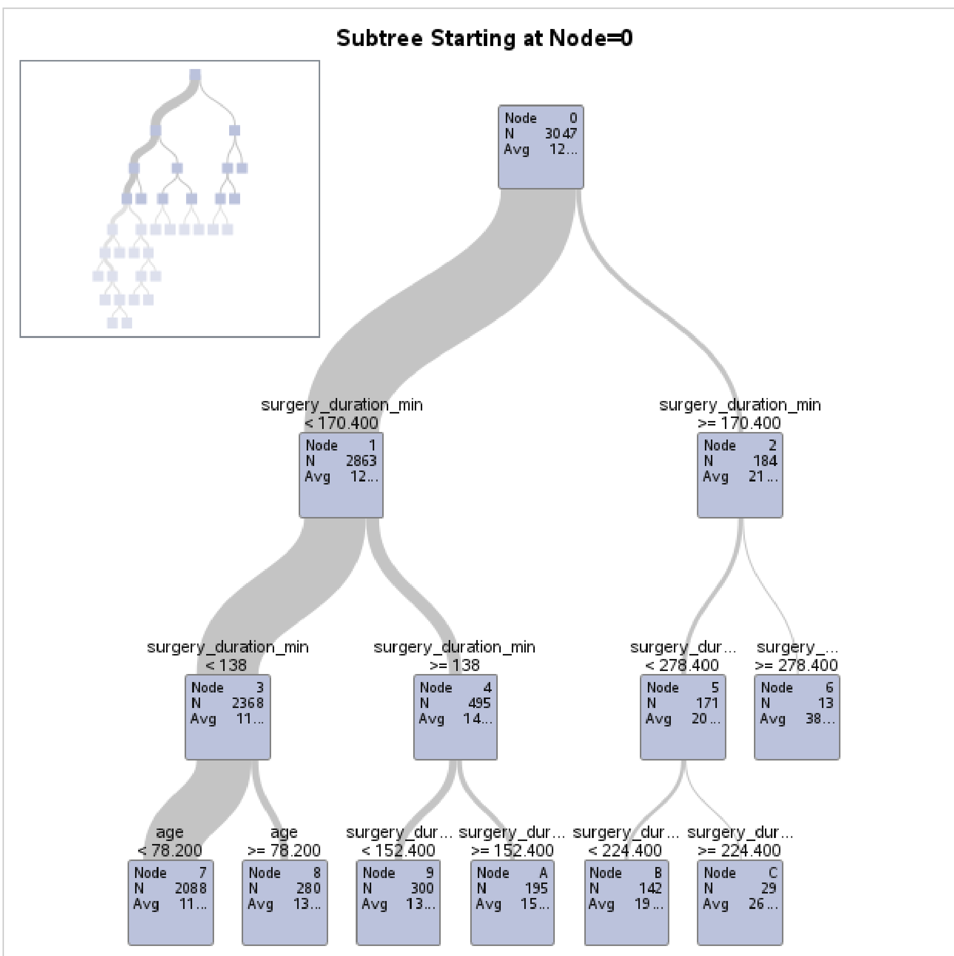
PYTHON CODE:

```python
y_pred=reg_tree_RSS.predict(X_test)

ind10=[]
ind15=[]
ind20=[]

for sub1, sub2 in zip(y_pred, y_test):
    ind10.append(1) if abs(sub1-sub2)<0.10*sub2 else ind10.append(0)
    ind15.append(1) if abs(sub1-sub2)<0.15*sub2 else ind15.append(0)
```

```
    ind20.append(1) if abs(sub1-sub2)<0.20*sub2 else ind20.append(0)

prop10=sum(ind10)/len(ind10)
prop15=sum(ind15)/len(ind15)
prop20=sum(ind20)/len(ind20)

print("Accuracy Scores RSS Splitting Criterion")
print("Accuracy within 10%:\n {}".format(prop10))
print("Accuracy within 15%:\n {}".format(prop15))
print("Accuracy within 20%:\n {}".format(prop20))
```

(c) Build a regression tree on the training data based on the CHAID splitting criterion and cost-complexity pruning. Give the graphical output.

SAS:

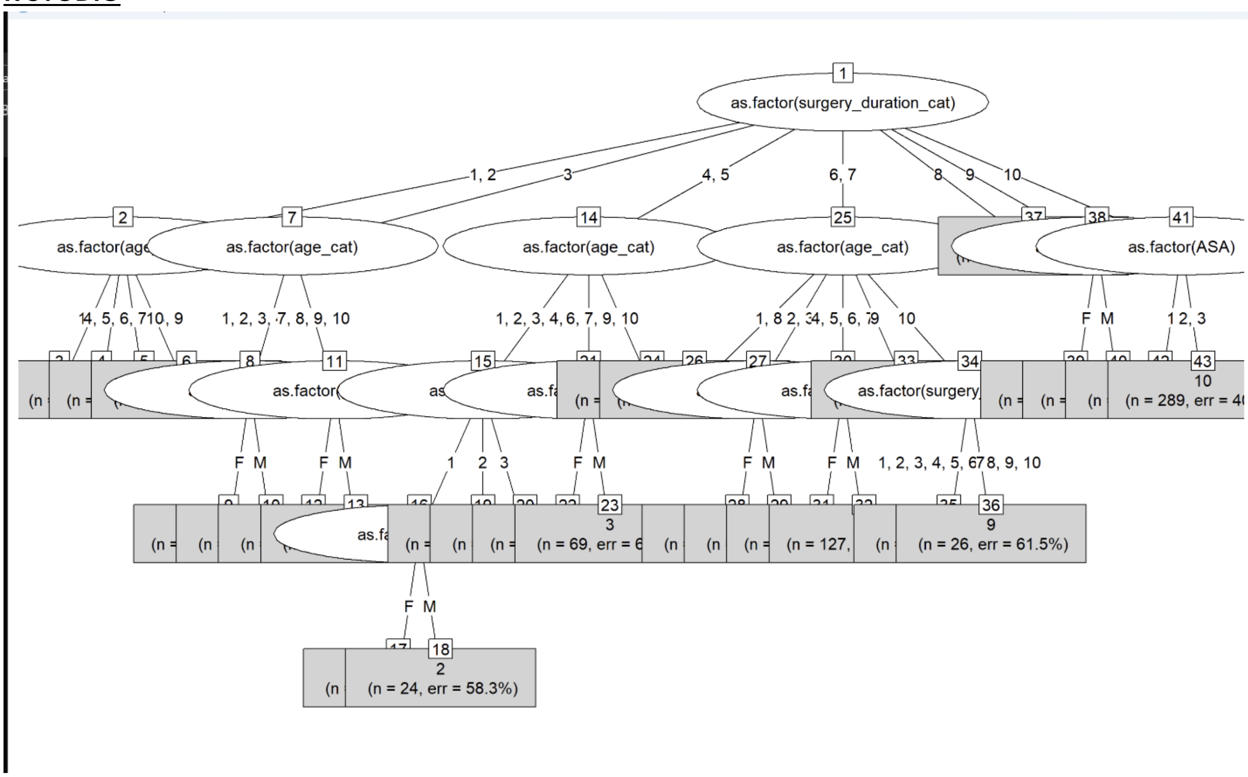


SAS CODE:

proc hpsplit data=hospital seed=501231;
class gender;
model surgery_cost = gender age BMI ASA surgery_duration_min;

```
grow CHAID;
prune costcomplexity;
partition rolevar=selected(train="1");
output out=predicted;
ID selected;
run;
```

R STUDIO



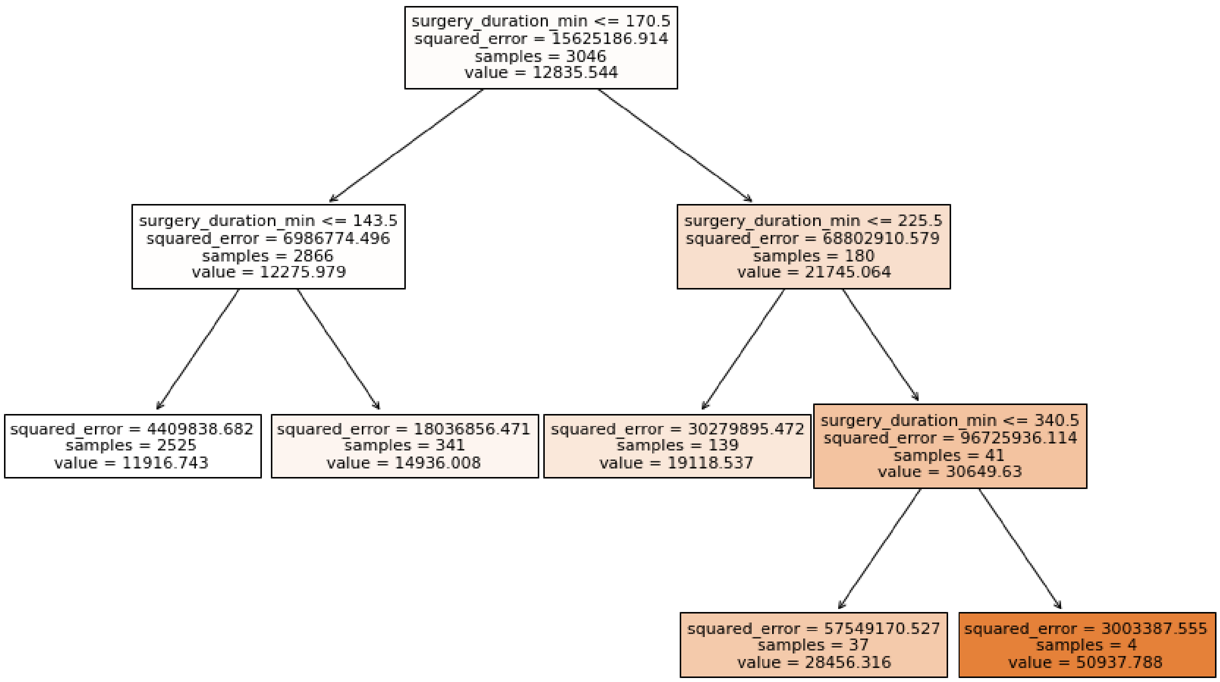
R CODE:

```
 # mutate continous variables for CHAID splitting
hospital_cat <- mutate(hospital, age_cat=ntile(age,10),
                BMI_cat=ntile(BMI,10),
                surgery_cost_cat=ntile(surgery_cost,10),
                surgery_duration_cat=ntile(surgery_duration_min,10))

# split 80% train 20% test for mutated data set
set.seed(239076)
sample2 <- sample(c(TRUE, FALSE), nrow(hospital_cat),
            replace = TRUE, prob = c(0.8, 0.2))
train_cat <- hospital_cat[sample2,]
test_cat <- hospital_cat[!sample2,]

# Fit regression tree for CHAID splitting
a1_chaid <- chaid(as.factor(surgery_cost_cat) ~ as.factor(gender) + as.factor(age_cat) +
```

as.factor(BMI_cat) + as.factor(ASA) +
as.factor(surgery_duration_cat), data = train_cat,
control = chaid_control(maxheight = 4))

plot(a1_chaid, type="simple")

PYTHON



Python CODE

```
# convert age to deciles
hospital['deciles']=pandas.qcut(hospital['surgery_cost'], 10, labels=False)
deciles_coding={0:'0th',1:'1st',2:'2nd',3:'3rd',4:'4th',5:'5th',6:'6th',7:'7th',8
:'8th',9:'9th'}
hospital['deciles']=hospital['deciles'].map(deciles_coding)

X=hospital.iloc[:,1:6].values
y=hospital.iloc[:,6:8].values


#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.20,
random_state=348644)
```

```python
X_train=pandas.DataFrame(X_train, columns=['gender','age',
'BMI','ASA','surgery_duration_min'])
y_train=pandas.DataFrame(y_train[:,1], columns=['deciles'])
train_data=pandas.concat([X_train, y_train],axis=1)

# fitting tree
from chefboost import Chefboost

config={'algorithm': 'CHAID'}
tree_chaid=Chefboost.fit(train_data, config, target_label='deciles')
```

(d) Use the fitted CHAID tree to predict surgery cost for the data in the testing set. Compute proportions of predicted values within 10%, 15%, and 20% of the observed values. Which of the two models, RSS or CHAID, give better prediction?

SAS:

accuracy10	accuracy15	accuracy20
0.519054	1	1

SAS CODE:

```
data accuracy;
set test;
if(abs(surgery_cost-P_surgery_cost)<0.10*surgery_cost)
then ind10=1; else ind10=0;
if(abs(surgery_cost-Psurgery_cost)<0.15*surgery_cost)
then ind15=1; else ind15=0;
if(abs(surgery_cost-Psurgery_cost)<0.20*surgery_cost)
then ind20=1; else ind20=0;
run;

proc sql;
 select mean(ind10) as accuracy10, mean(ind15) as accuracy15,
 mean(ind20) as accuracy20
 from accuracy;
 quit;
```

R STUDIO

```
[1] 0.442953
[1] 0.6134228
[1] 0.7409396
```

R CODE:

```
# accuracy within 10%
chaid_acc10 <- ifelse(abs(test_cat$surgery_cost - test_cat$p_median_surgery_cost)
                < 0.10*test_cat$surgery_cost, 1, 0)
print(mean(chaid_acc10))

# accuracy within 15%
chaid_acc15 <- ifelse(abs(test_cat$surgery_cost - test_cat$p_median_surgery_cost)
                < 0.15*test_cat$surgery_cost, 1, 0)
print(mean(chaid_acc15))

# accuracy within 20%
chaid_acc20 <- ifelse(abs(test_cat$surgery_cost - test_cat$p_median_surgery_cost)
                < 0.20*test_cat$surgery_cost, 1, 0)
print(mean(chaid_acc20))
```

Python:

CHAID Accuracy Scores
Accuracy within 10%
0.442257217847769
Accuracy within 15%
0.6220472440944882
Accuracy within 20%
0.7887139107611548

Python Code

```
y_test=pandas.DataFrame(y_test[:,0], columns=['surgery_cost'])
y_pred=pandas.DataFrame(y_pred, columns=['predclass'])
pred_data=pandas.concat([y_test,y_pred],axis=1)

df_new=pred_data.groupby('predclass')['surgery_cost'].mean()#predicted
value=class mean
inner_join = pandas.merge(pred_data, df_new, on='predclass', how ='inner')

ind10=[]
ind15=[]
ind20=[]
#median_house_value_x=observed value, median_house_value_y=predicted value
for sub1, sub2 in zip(inner_join['surgery_cost_x'],
inner_join['surgery_cost_y']):
    ind10.append(1) if abs(sub1-sub2)<0.10*sub1 else ind10.append(0)
    ind15.append(1) if abs(sub1-sub2)<0.15*sub1 else ind15.append(0)
    ind20.append(1) if abs(sub1-sub2)<0.20*sub1 else ind20.append(0)

prop10=sum(ind10)/len(ind10)
```

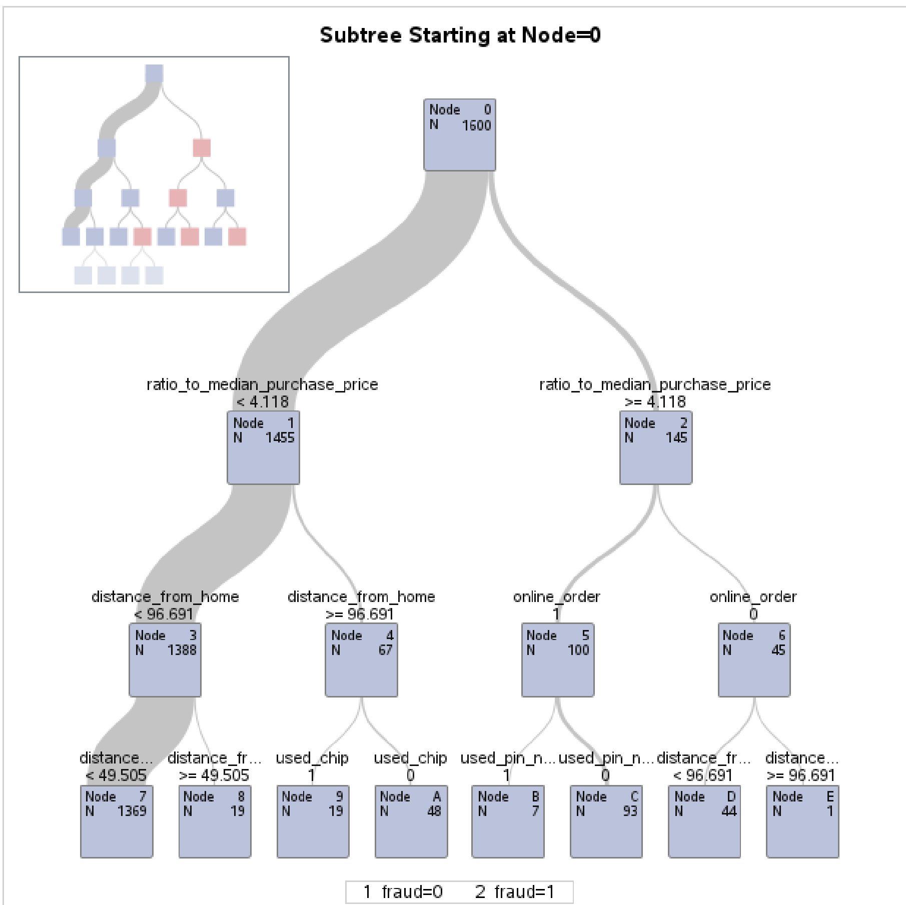
```
prop15=sum(ind15)/len(ind15)
prop20=sum(ind20)/len(ind20)

print("CHAID Accuracy Scores")
print("Accuracy within 10%\n {}".format(prop10))
print("Accuracy within 15%\n {}".format(prop15))
print("Accuracy within 20%\n {}".format(prop20))
```

RESULTS: Overall the RSS splitting criterion produced a more accurate model.

Problem 2. The data file "card_transdata.csv" downloaded from Kaggle.com contains data on fraudulent credit card activities. The variables are: distance_from_home (the distance between credit card holder's home and where the transaction happened), distance_from_last_transaction (distance between current and last transactions' locations), ratio_to_median_purchase_price (amount of current transaction over the median transaction amount on the credit card account), repeat_retailer (if purchases were made from the same retailer before), used_chip (if the chip on the credit card was used during transaction), used_pin_number (if PIN code was used during transaction), online_order (if the transaction was an online order), and fraud (if the transaction was fraudulent).

(a) Split the data into 80% training and 20% testing sets and build a binary classification tree for fraudulent activity on the training set using the Gini splitting criterion. Prune the tree using the cost-complexity pruning algorithm. Give the graphical output.

SAS

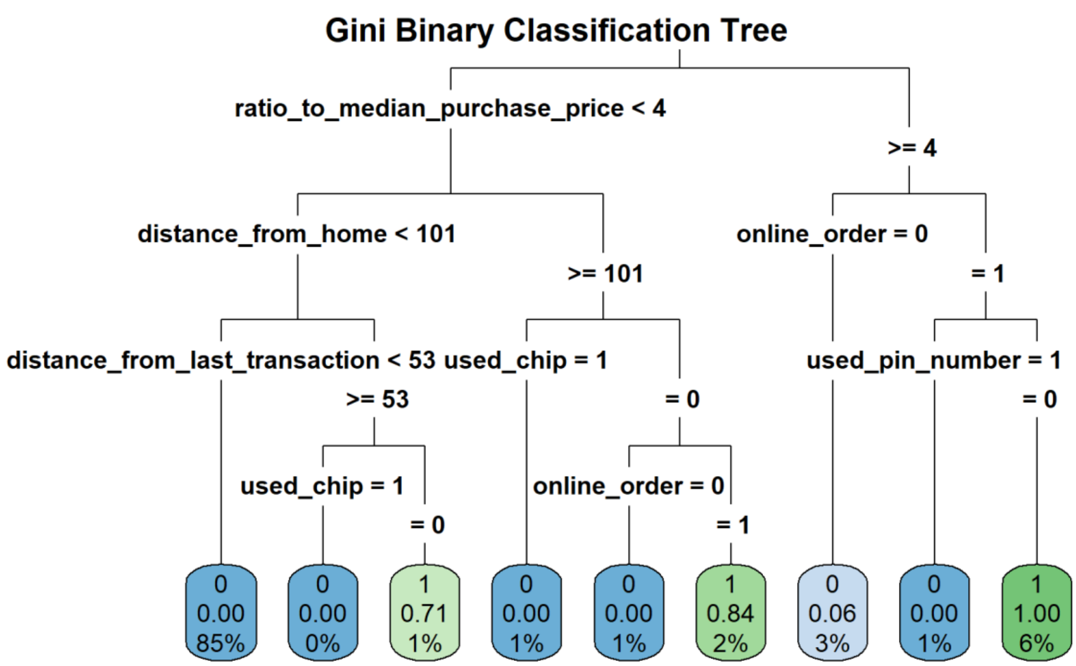


SAS CODE:

```
proc surveyselect data=card_data rate=0.8 seed=109238
out=card_data outall method=srs;
run;
```

```
/*GINI SPLITTING AND COST-COMPLEXITY PRUNING */
proc hpsplit data=card_data maxdepth=4;
class fraud repeat_retailer used_chip used_pin_number online_order;
model fraud(event="1") = distance_from_home distance_from_last_transaction
ratio_to_median_purchase_price repeat_retailer used_chip
used_pin_number online_order;
grow gini;
prune costcomplexity;
partition rolevar=selected(train="1");
output out=predicted;
ID selected;
run;
```

R STUDIO



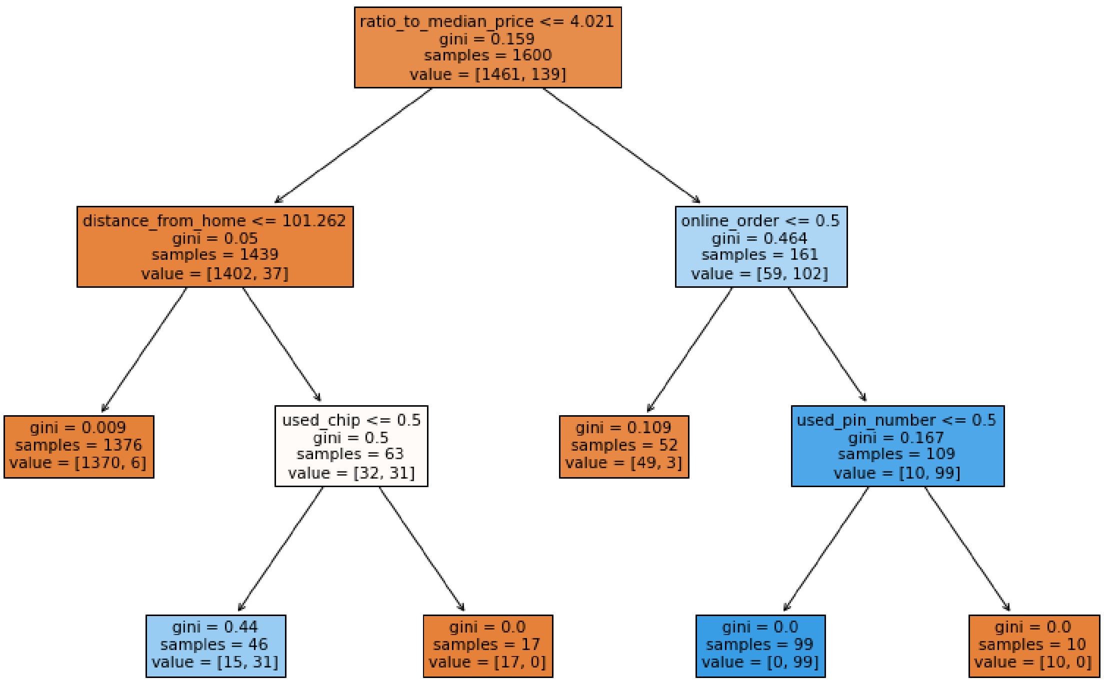
R CODE

```
# split data 80% train 20% test
sample3 <- sample(c(TRUE, FALSE), nrow(credit_data),
             replace=TRUE, prob=c(0.8, 0.2))
train_credit <- credit_data[sample3,]
test_credit <- credit_data[!sample3,]

# fitting tree with gini criterion
a2_gini <- rpart(fraud ~ distance_from_home + distance_from_last_transaction +
```

ratio_to_median_purchase_price + repeat_retailer +
used_chip + used_pin_number + online_order, data=credit_data,
method = "class", parms = list(split="Gini"), maxdepth=4)

# plot pruned tree (gini)
rpart.plot(a2_gini, type=3, main="Gini Binary Classification Tree")

Python



Python Code

```
# mark predictors and predicted variables
X=card_data.iloc[:,0:7]
y=card_data.iloc[:,7]

# split data to 80% train 20% test
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.20,
random_state=289022)

#FITTING BINARY TREE WITH GINI SPLITTING CRITERION
gini_tree=DecisionTreeClassifier(max_leaf_nodes=6, criterion='gini',
random_state=199233)
gini_tree.fit=gini_tree.fit(X_train,y_train)
```

```python
#PLOTTING FITTED TREE
fig = plt.figure(figsize=(15,10))
tree.plot_tree(gini_tree.fit,
feature_names=['distance_from_home','distance_from_last_transaction',
'ratio_to_median_price','repeat_retailer', 'used_chip', 'used_pin_number',
'online_order'], filled=True)
```

(b) Compute the prediction accuracy for the training data, using the range of classification thresholds between 0.01 and 0.99. What thresholds correspond to the largest prediction accuracy?

SAS

cutoff	trueclassrate
0.01	0.985
0.02	0.985
0.03	0.985

SAS Code

```
data test;
set predicted;
if(selected="0");
keep fraud P_fraud1;
run;

data cutoffs;
set test;
do i=1 to 99;
tp=(P_fraud1 > 0.01*i and fraud="1");
tn=(P_fraud1 < 0.01*i and fraud="0");
output;
end;
run;

proc sql;
create table rates as
select i, sum(tp+tn)/count(*) as trueclassrate
from cutoffs
group by i;
select 0.01*i as cutoff, trueclassrate
```

```
 from rates
  having trueclassrate=max(trueclassrate);
quit;
```

R Studio

```
     [,1]       [,2]
[1,] 0.94 0.09296482
[2,] 0.95 0.09296482
```

R Code

```
# compute prediction accuracy for testing data
pred_gini <- predict(a2_gini, test_credit)
test_pred <- cbind(test_credit, pred_gini)

test_pred <- test_pred %>% rename("no" = "0")
test_pred <- test_pred %>% rename("yes" = "1")



tp <- matrix(NA, nrow = nrow(test_pred), ncol = 99)
tn <- matrix(NA, nrow = nrow(test_pred), ncol = 99)

for (i in 1:99) {
  tp[,i] <- ifelse(test_pred$fraud==1 & test_pred$yes > 0.01*i, 1, 0)
  tn[,i] <- ifelse(test_pred$fraud==0 & test_pred$no <= 0.01*i, 1, 0)
}

trueclassrate <- matrix(NA, nrow = 99, ncol = 2)
for(i in 1:99){
  trueclassrate[i,1] <- 0.01*i
  trueclassrate[i,2] <- sum(tp[,i] + tn[,i])/nrow(test_pred)
}

trueclassrateFinal <- trueclassrate[which(trueclassrate[,2]==max(trueclassrate[,2])),]

print(trueclassrateFinal)
```

Python

```
trueclassrate cutoff
5 0.9725     0.06
6 0.9725     0.07
```

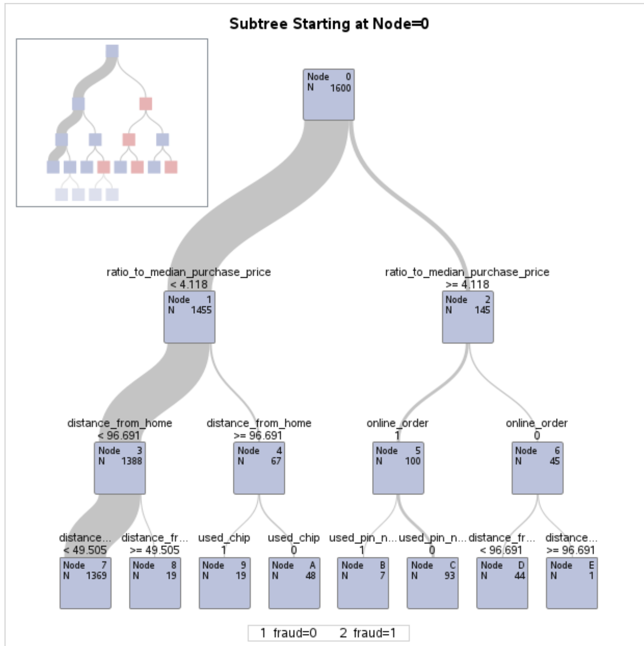
Python Code

```
#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
y_pred=gini_tree.predict_proba(X_test)
```

```
#y_pred[::,1] are predicted probabilities of "yes"

total=len(y_pred)
trueclassrate=[]
cutoff=[]

for i in range(99):
    tp=0
    tn=0
    cutoff.append(0.01*(i+1))
    for sub1, sub2 in zip(y_pred[::,1], y_test):
        tp_ind=1 if (sub1>0.01*(i+1) and sub2==1) else 0
        tn_ind=1 if (sub1<0.01*(i+1) and sub2==0) else 0
        tp+=tp_ind
        tn+=tn_ind
    rate=(tp+tn)/total
    trueclassrate.append(rate)

df=pandas.DataFrame({'trueclassrate': trueclassrate,'cutoff': cutoff})
max_rate=max(trueclassrate)
optimal=df[df['trueclassrate']==max_rate]
print(optimal)
```

(c) Fit the binary classification tree using the entropy splitting criterion and cost-complexity pruning algorithm. Display the tree.

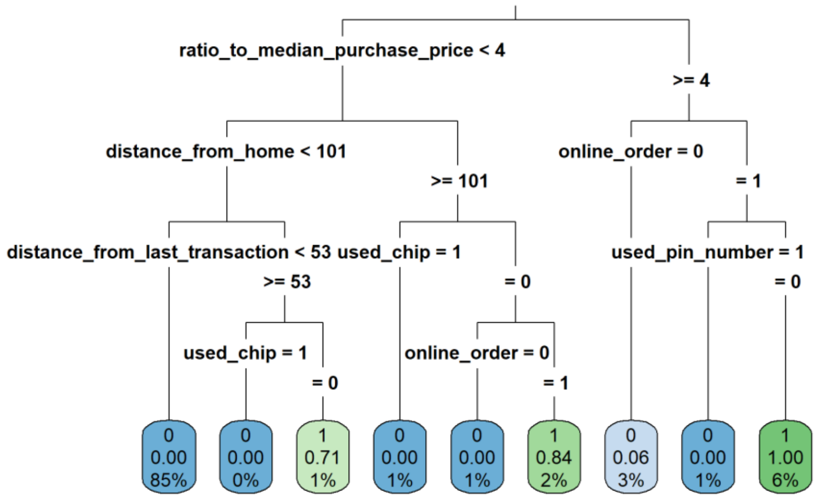
SAS



SAS Code

```
proc hpsplit data=card_data maxdepth=4;
class fraud repeat_retailer used_chip used_pin_number online_order;
model fraud(event="1") = distance_from_home distance_from_last_transaction
ratio_to_median_purchase_price repeat_retailer used_chip
used_pin_number online_order;
grow entropy;
prune costcomplexity;
partition rolevar=selected(train="1");
output out=predicted;
ID selected;
run;
```

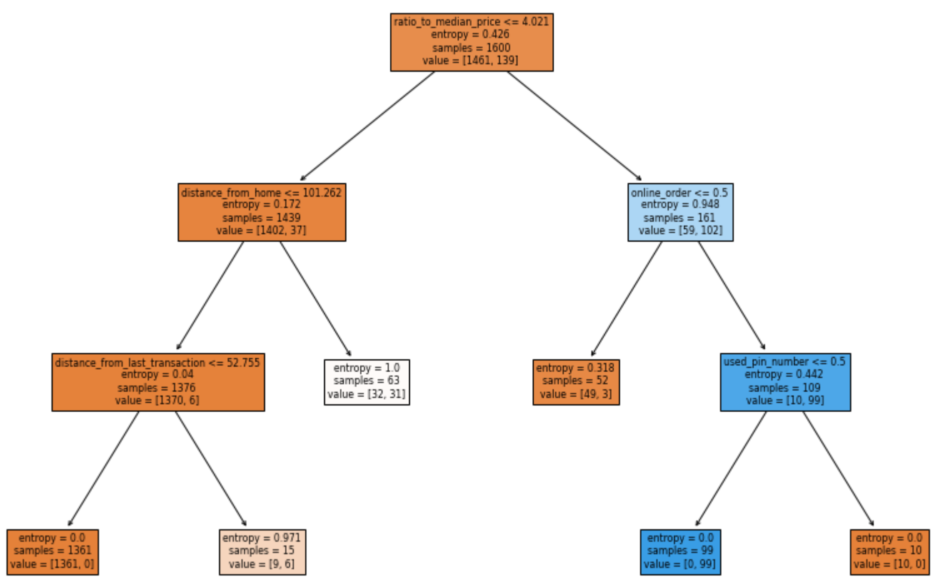
R Studio



R Code

```
c2_entropy <- rpart(fraud ~ distance_from_home + distance_from_last_transaction +
            ratio_to_median_purchase_price + repeat_retailer +
            used_chip + used_pin_number + online_order, data=credit_data,
          method = "class", parms = list(split="entropy"), maxdepth=4)

rpart.plot(c2_entropy, type=3)
```

Python



Python Code

```
#FITTING BINARY TREE WITH ENTROPY SPLITTING CRITERION
gini_tree=DecisionTreeClassifier(max_leaf_nodes=6, criterion='entropy',
random_state=199233)
gini_tree.fit=gini_tree.fit(X_train,y_train)

#PLOTTING FITTED TREE
fig = plt.figure(figsize=(15,10))
tree.plot_tree(gini_tree.fit,
feature_names=['distance_from_home','distance_from_last_transaction',
'ratio_to_median_price','repeat_retailer', 'used_chip', 'used_pin_number',
'online_order'], filled=True)
```

(d) Compute the prediction accuracy of the entropy tree for the training data, using the cut-offs for predicted probability of fraud ranging between 0.01 and 0.99. List the cut-offs that give the maximum prediction accuracy.

SAS

cutoff	trueclassrate
0.01	0.985
0.02	0.985

SAS Code

```
data test;
set predicted;
if(selected="0");
keep fraud P_fraud1;
run;

data cutoffs;
set test;
do i=1 to 99;
tp=(P_fraud1 > 0.01*i and fraud="1");
tn=(P_fraud1 < 0.01*i and fraud="0");
output;
end;
run;

proc sql;
create table rates as
select i, sum(tp+tn)/count(*) as trueclassrate
from cutoffs
group by i;
```

```
select 0.01*i as cutoff, trueclassrate
 from rates
  having trueclassrate=max(trueclassrate);
quit;
```

R Studio

```
      [,1]       [,2]
[1,] 0.94 0.09296482
```

R Code:

```
# compute prediction accuracy for testing data
pred_entropy <- predict(c2_entropy, test_credit)
test_pred <- cbind(test_credit, pred_entropy)

test_pred <- test_pred %>% rename("no" = "0")
test_pred <- test_pred %>% rename("yes" = "1")


tp <- matrix(NA, nrow = nrow(test_pred), ncol = 99)
tn <- matrix(NA, nrow = nrow(test_pred), ncol = 99)

for (i in 1:99) {
  tp[,i] <- ifelse(test_pred$fraud==1 & test_pred$yes > 0.01*i, 1, 0)
  tn[,i] <- ifelse(test_pred$fraud==0 & test_pred$no <= 0.01*i, 1, 0)
}

trueclassrate <- matrix(NA, nrow = 99, ncol = 2)
for(i in 1:99){
  trueclassrate[i,1] <- 0.01*i
  trueclassrate[i,2] <- sum(tp[,i] + tn[,i])/nrow(test_pred)
}

trueclassrateFinal <- trueclassrate[which(trueclassrate[,2]==max(trueclassrate[,2])),]

print(trueclassrateFinal)
```

Python

```
trueclassrate cutoff
0.9675       0.50
```

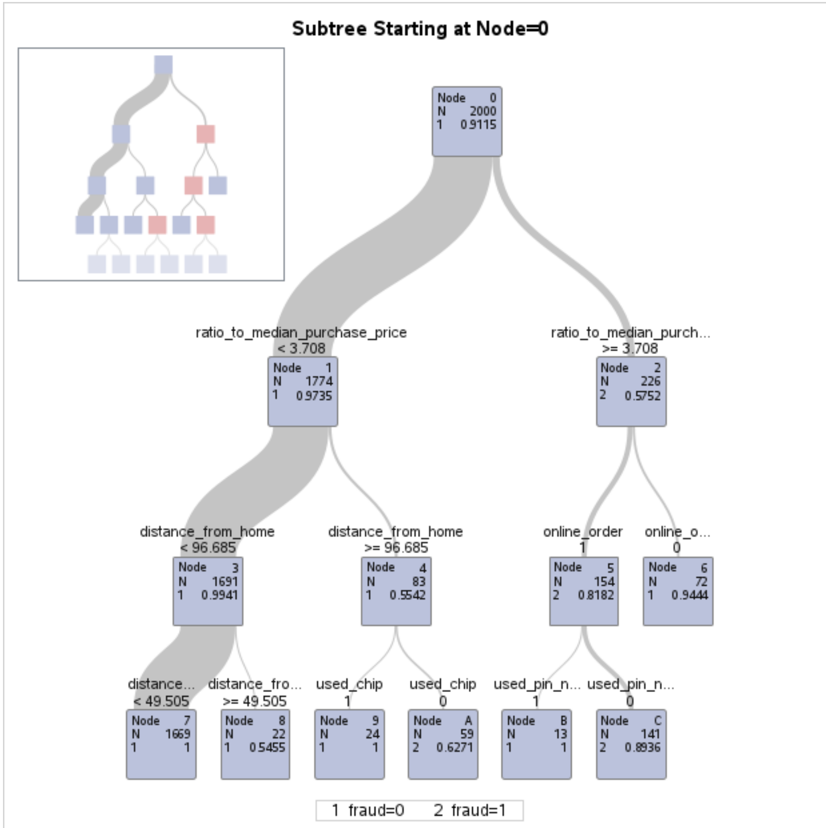
Python Code

```
#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
y_pred=gini_tree.predict_proba(X_test)

#y_pred[::,1] are predicted probabilities of "yes"
```

```
total=len(y_pred)
trueclassrate=[]
cutoff=[]

for i in range(99):
    tp=0
    tn=0
    cutoff.append(0.01*(i+1))
    for sub1, sub2 in zip(y_pred[::,1], y_test):
        tp_ind=1 if (sub1>0.01*(i+1) and sub2==1) else 0
        tn_ind=1 if (sub1<0.01*(i+1) and sub2==0) else 0
        tp+=tp_ind
        tn+=tn_ind
    rate=(tp+tn)/total
    trueclassrate.append(rate)


df=pandas.DataFrame({'trueclassrate': trueclassrate,'cutoff': cutoff})
max_rate=max(trueclassrate)
optimal=df[df['trueclassrate']==max_rate]
print(optimal)
```

(e) Fit the binary classification tree using the CHAID splitting criterion and cost-complexity pruning algorithm. Display the tree.

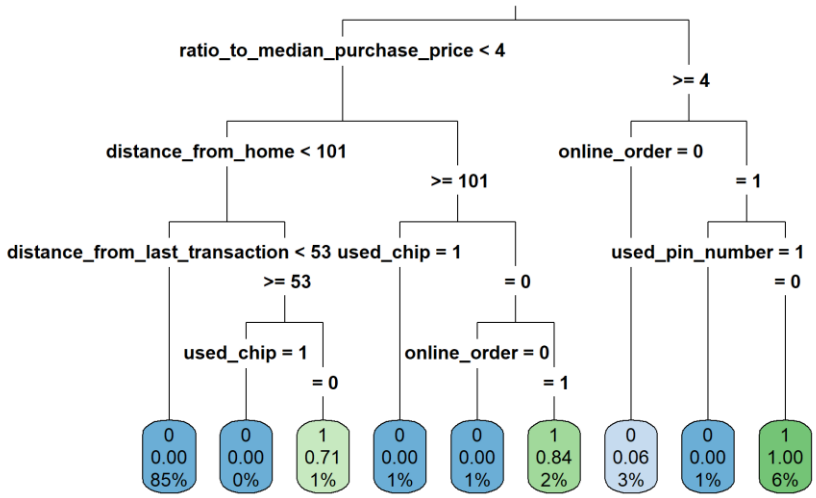
SAS



SAS Code

```
proc hpsplit data=card_data maxdepth=4;
class fraud repeat_retailer used_chip used_pin_number online_order;
model fraud(event="1") = distance_from_home distance_from_last_transaction
ratio_to_median_purchase_price repeat_retailer used_chip
used_pin_number online_order;
grow CHAID;
prune costcomplexity;
partition rolevar=selected(train="1");
output out=predicted;
ID selected;
run;
```

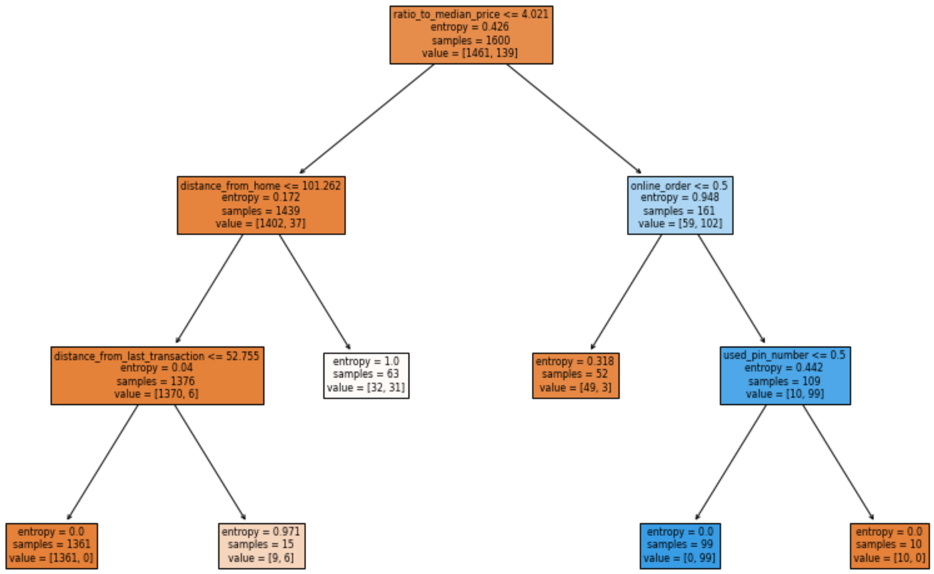
R Studio



R Code:

```
cred_mut <- mutate(credit_data, dist.home.cat=ntile(distance_from_home,10),
           dist.trans.cat=ntile(distance_from_last_transaction,10),
           ratio.cat=ntile(ratio_to_median_purchase_price,10))

# split data 80-20 split
sample_chaid <- sample(c(TRUE,FALSE),nrow(cred_mut),
             replace=TRUE, prob=c(0.8,0.2))

train_chaid <- cred_mut[sample_chaid,]
test_chaid <- cred_mut[!sample_chaid,]


# fit binary classification tree
library(CHAID)
tree.chaid <- chaid(as.factor(fraud) ~ as.factor(dist.home.cat) +
              as.factor(dist.trans.cat) + as.factor(ratio.cat) +
              as.factor(repeat_retailer) + as.factor(used_chip) +
              as.factor(used_pin_number) + as.factor(online_order),
             data = cred_mut, control = chaid_control(maxheight = 3))

plot(tree.chaid, type = "simple")
```

Python



Python Code

```
train_data=pandas.concat([X_train, y_train], axis=1) #one-to-one concatenation


config={'algorithm': 'CHAID', 'max_depth': 4}
tree_chaid=Chefboost.fit(train_data, config, target_label='fraud')
```

(f) Compute the prediction accuracy of the CHAID tree for the training data, using the cut-offs for predicted probability of fraud ranging between 0.01 and 0.99. List the cut-offs that give the maximum prediction accuracy. Which of the three trees (Gini, entropy, or CHAID) produces the largest maximum prediction accuracy?

SAS

cutoff	trueclassrate
0.03	0.985

SAS Code

```
data test;
set predicted;
if(selected="0");
keep fraud P_fraud1;
run;
```

```
data cutoffs;
set test;
do i=1 to 99;
tp=(P_fraud1 > 0.01*i and fraud="1");
tn=(P_fraud1 < 0.01*i and fraud="0");
output;
end;
run;

proc sql;
create table rates as
select i, sum(tp+tn)/count(*) as trueclassrate
from cutoffs
group by i;
select 0.01*i as cutoff, trueclassrate
 from rates
  having trueclassrate=max(trueclassrate);
quit;
```

R Studio

```
0.9726368
```

R Code

```
# compute predictin accuracy for testing data
pred_bin_chaid <- predict(tree.chaid, newdata = test_chaid)
test_new <- cbind(test_chaid, pred_bin_chaid)

truepred <-c()
n <- nrow(test_new)
for (i in 1:n) {
  truepred[i] <- ifelse(test_new$fraud[i]==test_new$pred_bin_chaid[i],1,0)
}

print(truepredrate <- mean(truepred))
```

Problem 3. Consider the Gini classification tree built in Problem 2. For the predicted classifications on the training data,

(a) Compute the confusion matrix (the number of true positive, false positive, true negative, and false negative predictions). Use the 0.5 cut-off for predicted probability of fraud.

SAS

Model-Based Confusion Matrix			
	Predicted		
Actual	0	1	Error Rate
0	1454	13	0.0089
1	6	127	0.0451

SAS Code

```
/*SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS*/
proc surveyselect data=card_data rate=0.8 seed=109238
out=card_data outall method=srs;
run;
/*GINI SPLITTING AND COST-COMPLEXITY PRUNING */
proc hpsplit data=card_data maxdepth=4;
class fraud repeat_retailer used_chip used_pin_number online_order;
model fraud(event="1") = distance_from_home distance_from_last_transaction
ratio_to_median_purchase_price repeat_retailer used_chip
used_pin_number online_order;
grow gini;
prune costcomplexity;
partition rolevar=selected(train="1");
output out=predicted;
ID selected;
run;



/*COMPUTING CONFUSION MATRIX AND PERFORMANCE MEASURES
FOR TESTING DATA*/
data test;
set predicted;
if(selected="0");
tp=(P_fraud1 > 0.5 and fraud="1");
fp=(P_fraud1> 0.5 and fraud="0");
```

```
tn=(P_fraud0> 0.5 and fraud="0");
fn=(P_fraud0> 0.5 and fraud="1");
run;

proc sql;
create table confusion as
select sum(tp) as tp, sum(fp) as fp, sum(tn) as tn,
sum(fn) as fn, count(*) as total
from test;
select * from confusion;
quit;
```

R Studio

Tp=27	fp=1
tn=347	fn=5

R Code

```
#COMPUTING CONFUSION MATRIX AND PERFORMANCE MEASURES FOR TESTING DATA
pred.values<- predict(gini_3a, test_3a)
test<- cbind(test_3a,pred.values)

pred.3a <- predict(gini_3a, test_3a)
test_3a <- cbind(test_3a, pred.3a)

View(test_3a)

test_3a <- test_3a %>% rename("no"="0")
test_3a <- test_3a %>% rename("yes"="1")

tp<- c()
fp<- c()
tn<- c()
fn<- c()

total<- nrow(test_3a)
for (i in 1:total){
  tp[i]<- ifelse(test_3a$yes[i]>0.5 & test_3a$fraud[i]=="1",1,0)
  fp[i]<- ifelse(test_3a$yes[i]>0.5 & test_3a$fraud[i]=="0",1,0)
  tn[i]<- ifelse(test_3a$no[i]>0.5 & test_3a$fraud[i]=="0",1,0)
  fn[i]<- ifelse(test_3a$no[i]>0.5 & test_3a$fraud[i]=="1",1,0)
}

print(tp<- sum(tp))
```

```
print(fp<- sum(fp))
print(tn<- sum(tn))
print(fn<- sum(fn))
```

Python

```
tp: 31 fp: 4 tn: 358 fn: 7 total: 400
```

Python Code

```
#FITTING BINARY TREE WITH GINI SPLITTING CRITERION
gini_tree=DecisionTreeClassifier(max_leaf_nodes=6, criterion='gini',
random_state=199233)
gini_tree.fit=gini_tree.fit(X_train,y_train)

#COMPUTING CONFUSION MATRIX AND PERFORMANCE MEASURES FOR TESTING SET
y_pred=gini_tree.predict_proba(X_test)

total=len(y_pred)

tpos=[]
fpos=[]
tneg=[]
fneg=[]

for sub1, sub2 in zip(y_pred[::,1], y_test):
    tpos.append(1) if (sub1>0.5 and sub2==1) else tpos.append(0)
    fpos.append(1) if (sub1>0.5 and sub2==0) else fpos.append(0)
    tneg.append(1) if (sub1<0.5 and sub2==0) else tneg.append(0)
    fneg.append(1) if (sub1<0.5 and sub2==1) else fneg.append(0)
    tp=sum(tpos)
    fp=sum(fpos)
    tn=sum(tneg)
    fn=sum(fneg)

print('tp:', tp)
print('fp:', fp)
print('tn:', tn)
print('fn:', fn)
print('total:', total)
```

(b) Compute the prediction performance measures (accuracy, misclassification rate, sensitivity, false negative rate, specificity, false positive rate, precision, negative predictive value, and F1-score).

SAS

accuracy	misclassrate	sensitivity	FNR	specificity	FPR	precision	NPV	F1score
0.985	0.015	0.863636	0.136364	1	0	1	0.983425	0.926829

SAS Code

```
proc sql;
select (tp+tn)/total as accuracy, (fp+fn)/total as
misclassrate, tp/(tp+fn) as sensitivity,
fn/(tp+fn) as FNR, tn/(fp+tn) as specificity,
fp/(fp+tn) as FPR, tp/(tp+fp) as precision,
tn/(fn+tn) as NPV, 2*tp/(2*tp+fn+fp) as F1score
from confusion;
quit;
```

R Studio

```
> print(accuracy<- (tp+tn)/total)
[1] 0.9842105
> print(misclassrate<- (fp+fn)/total)
[1] 0.01578947
> print(sensitivity<- tp/(tp+fn))
[1] 0.84375
> print(FNR<- fn/(tp+fn))
[1] 0.15625
> print(specificity<- tn/(fp+tn))
[1] 0.9971264
> print(FPR<- fp/(fp+tn))
[1] 0.002873563
> print(precision<- tp/(tp+fp))
[1] 0.9642857
> print(NPV<- tn/(fn+tn))
[1] 0.9857955
> print(F1score<- 2*tp/(2*tp+fn+fp))
[1] 0.9
```

R Code

```
print(accuracy<- (tp+tn)/total)
print(misclassrate<- (fp+fn)/total)
print(sensitivity<- tp/(tp+fn))
print(FNR<- fn/(tp+fn))
print(specificity<- tn/(fp+tn))
print(FPR<- fp/(fp+tn))
print(precision<- tp/(tp+fp))
print(NPV<- tn/(fn+tn))
print(F1score<- 2*tp/(2*tp+fn+fp))
```

Python

```
accuracy: 0.9725 misclassrate: 0.0275 sensitivity: 0.8157894736842105 FNR:
0.18421052631578946 specificity: 0.988950276243094 FPR: 0.011049723756906077
precision: 0.8857142857142857 NPV: 0.9808219178082191 F1score: 0.8493150684931506
```

Python Code

```
accuracy=(tp+tn)/total
misclassrate=(fp+fn)/total
sensitivity=tp/(tp+fn)
FNR=fn/(tp+fn)
specificity=tn/(fp+tn)
FPR=fp/(fp+tn)
precision=tp/(tp+fp)
NPV=tn/(fn+tn)
F1score=2*tp/(2*tp+fn+fp)

print('accuracy:', accuracy)
print('misclassrate:', misclassrate)
print('sensitivity:', sensitivity)
print('FNR:', FNR)
print('specificity:', specificity)
print('FPR:', FPR)
print('precision:', precision)
print('NPV:', NPV)
print('F1score:', F1score)
```

Problem 4. Consider the Gini classification tree built in Problem 2. For the predicted classifications on the training data:

(a) Compute prediction accuracy, misclassification rate, sensitivity, and specificity for a range of cut-offs between 0.01 and 0.99.

SAS

Obs	i	accuracy	misclassrate	sensitivity	specificity	oneminusspec
1	0	0.1100	0.8900	1.00000	0.00000	1.00000
2	1	0.9575	0.0425	0.93182	0.96067	0.03933
3	2	0.9575	0.0425	0.93182	0.96067	0.03933
4	3	0.9850	0.0150	0.86364	1.00000	0.00000
5	4	0.9850	0.0150	0.86364	1.00000	0.00000
97	96	0.9500	0.0500	0.54545	1.00000	0.00000
98	97	0.9500	0.0500	0.54545	1.00000	0.00000
99	98	0.9500	0.0500	0.54545	1.00000	0.00000
100	99	0.9500	0.0500	0.54545	1.00000	0.00000
101	100	0.9500	0.0500	0.54545	1.00000	0.00000
102	101	0.8900	0.1100	0.00000	1.00000	0.00000

SAS Code

```
/*COMPUTING CONFUSION MATRICES AND PERFORMANCE MEASURES
FOR TESTING SET FOR A RANGE OF CUTOFFS*/
data test;
set predicted;
if(selected="0");
run;

data cutoffs;
set test;
do i=0 to 101;
tp=(P_fraud1 >= 0.01*i and fraud="1");
fp=(P_fraud1>= 0.01*i and fraud="0");
```

```
tn=(P_fraud1<  0.01*i and fraud="0");
fn=(P_fraud1<  0.01*i and fraud="1");
output;
end;
run;

proc sql;
create table confusion as
select i, sum(tp) as tp, sum(fp) as fp, sum(tn) as tn,
sum(fn) as fn, count(*) as total
from cutoffs
group by i;
quit;

proc sql;
create table measures as
select i, (tp+tn)/total as accuracy, (fp+fn)/total as
misclassrate, tp/(tp+fn) as sensitivity, tn/(fp+tn) as specificity,
fp/(fp+tn) as oneminusspec
from confusion
group by i;
quit;

proc print data=measures;
run;
```

R Studio

```
       accuracy misclassrate sensitivity specificity   distance cutoff
 [1,] 0.9842105   0.01578947     0.84375   0.9971264 0.02442232   0.08
 [2,] 0.9842105   0.01578947     0.84375   0.9971264 0.02442232   0.09
 [3,] 0.9842105   0.01578947     0.84375   0.9971264 0.02442232   0.10
 [4,] 0.9842105   0.01578947     0.84375   0.9971264 0.02442232   0.11
 [5,] 0.9842105   0.01578947     0.84375   0.9971264 0.02442232   0.12
```

R Code

```
#COMPUTING CONFUSION MATRIX AND PERFORMANCE MEASURES FOR TESTING DATA
# for range of cutoffs

tpos<- matrix(NA, nrow=nrow(test_3a), ncol=102)
fpos<- matrix(NA, nrow=nrow(test_3a), ncol=102)
tneg<- matrix(NA, nrow=nrow(test_3a), ncol=102)
fneg<- matrix(NA, nrow=nrow(test_3a), ncol=102)

for (i in 0:101) {
  tpos[,i+1]<- ifelse(test_3a$fraud=="1" & test_3a$yes>=0.01*i,1,0)
```

```
  fpos[,i+1]<- ifelse(test_3a$fraud=="0" & test_3a$yes>=0.01*i, 1,0)
  tneg[,i+1]<- ifelse(test_3a$fraud=="0" & test_3a$yes<0.01*i,1,0)
  fneg[,i+1]<- ifelse(test_3a$fraud=="1" & test_3a$yes<0.01*i,1,0)
}


tp<- c()
fp<- c()
tn<- c()
fn<- c()
accuracy<- c()
misclassrate<- c()
sensitivity<- c()
specificity<- c()
oneminusspec<- c()
cutoff<- c()


for (i in 1:102) {
tp[i]<- sum(tpos[,i])
fp[i]<- sum(fpos[,i])
tn[i]<- sum(tneg[,i])
fn[i]<- sum(fneg[,i])
total<- nrow(test_3a)
accuracy[i]<- (tp[i]+tn[i])/total
misclassrate[i]<- (fp[i]+fn[i])/total
sensitivity[i]<- tp[i]/(tp[i]+fn[i])
specificity[i]<- tn[i]/(fp[i]+tn[i])
oneminusspec[i]<- fp[i]/(fp[i]+tn[i])
cutoff[i]<- 0.01*(i-1)
}
```

Python

```
accuracy misclassrate sensitivity specificity oneminusspec distance \ 0 0.95 0.05 0.842105 0.961326 0.038674 0.162562 1 0.95 0.05 0.842105 0.961326 0.038674 0.162562 2 0.95 0.05 0.842105 0.961326 0.038674 0.162562 3 0.95 0.05 0.842105 0.961326 0.038674 0.162562 4 0.95 0.05 0.842105 0.961326 0.038674 0.162562 cut-off 0 0.01 1 0.02 2 0.03 3 0.04 4 0.05
```

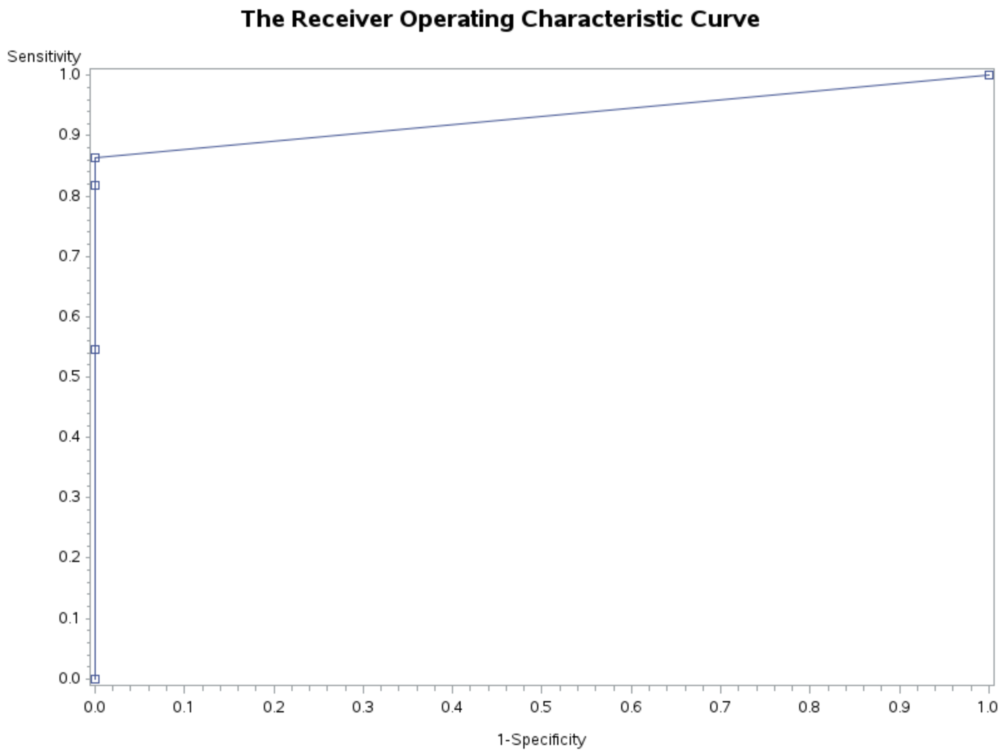
Python Code

```
#COMPUTING CONFUSION MATRICES AND PERFORMANCE MEASURES FOR TESTING SET FOR A RANGE OF CUTOFFS
y_pred=gini_tree.predict_proba(X_test)

total=len(y_pred)
```

```
cutoff=[]
accuracy=[]
misclassrate=[]
sensitivity=[]
specificity=[]
oneminusspec=[]
distance=[]

for i in range(99):
    tp=0
    fp=0
    tn=0
    fn=0
    cutoff.append(0.01*(i+1))
    for sub1, sub2 in zip(y_pred[::,1], y_test):
        tp_ind=1 if (sub1>0.01*(i+1) and sub2==1) else 0
        fp_ind=1 if (sub1>0.01*(i+1) and sub2==0) else 0
        tn_ind=1 if (sub1<0.01*(i+1) and sub2==0) else 0
        fn_ind=1 if (sub1<0.01*(i+1) and sub2==1) else 0
        tp+=tp_ind
        fp+=fp_ind
        tn+=tn_ind
        fn+=fn_ind

    accuracy_i=(tp+tn)/total
    misclassrate_i=(fp+fn)/total
    sensitivity_i=tp/(tp+fn)
    specificity_i=tn/(fp+tn)
    oneminusspec_i=fp/(fp+tn)
    distance_i=numpy.sqrt(pow(oneminusspec_i,2)+pow(1-sensitivity_i,2))

    accuracy.append(accuracy_i)
    misclassrate.append(misclassrate_i)
    sensitivity.append(sensitivity_i)
    specificity.append(specificity_i)
    oneminusspec.append(oneminusspec_i)
    distance.append(distance_i)
```

(b) Construct a Receiver Operating Characteristic (ROC) curve.

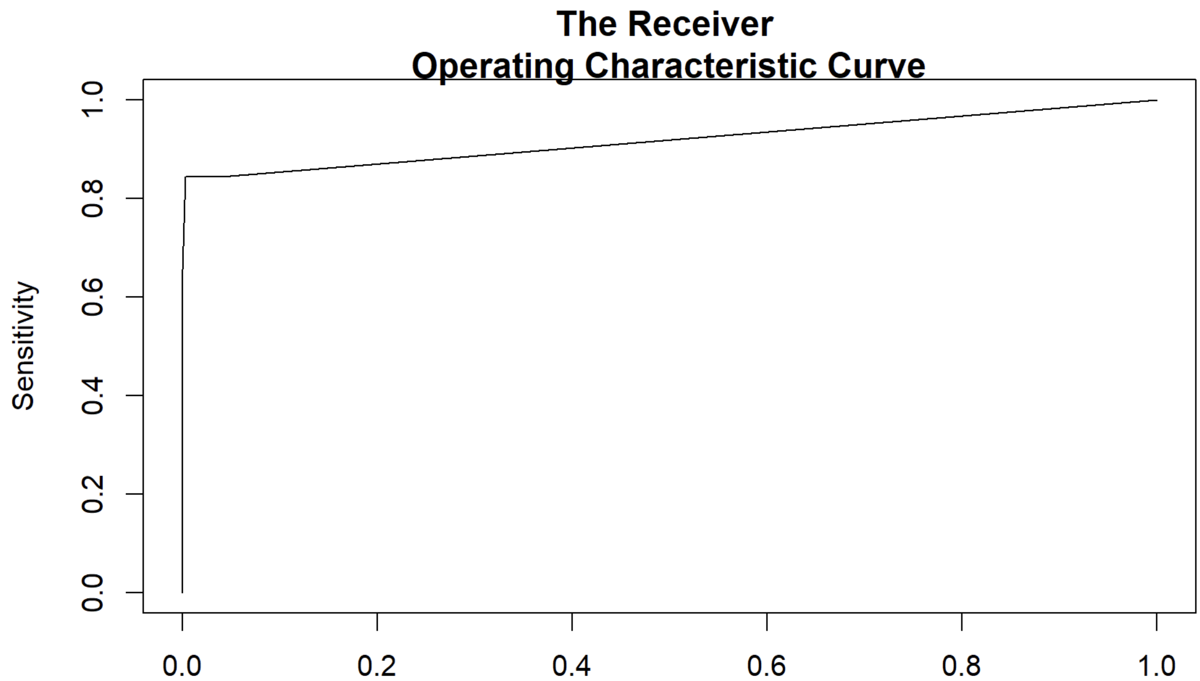
SAS



SAS Code

```
title 'The Receiver Operating Characteristic Curve';
proc gplot data=measures;
symbol v=square interpol=join;
plot sensitivity*oneminusspec/ vaxis=0 to 1 by 0.1 haxis=0 to 1 by 0.1;
label sensitivity="Sensitivity" oneminusspec="1-Specificity";
run;
```

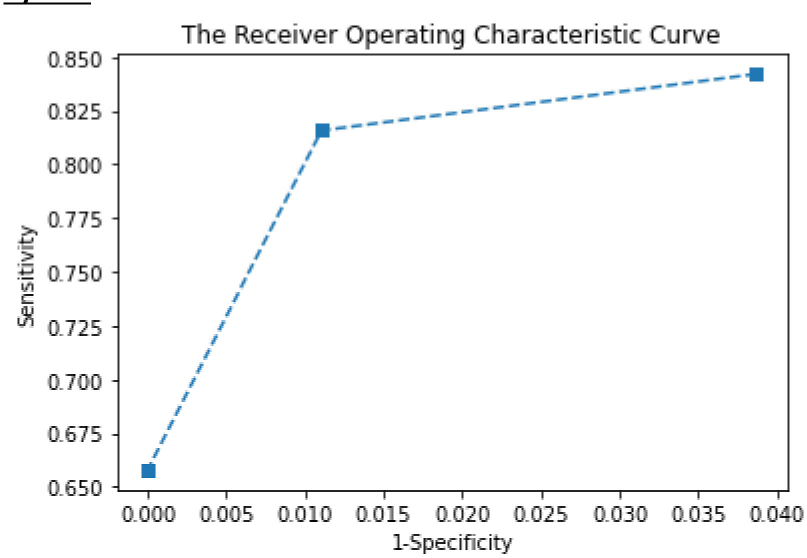
R Studio



R Code

```
#PLOTTING ROC CURVE
plot(oneminusspec, sensitivity, type="l", lty=1, main="The Receiver
Operating Characteristic Curve", xlab="1-Specificity", ylab="Sensitivity")
```

Python



Python Code

```
#PLOTTING ROC CURVE
import matplotlib.pyplot as plot
```

```
plt.plot(oneminusspec, sensitivity, linestyle='--', marker='s')
plt.title('The Receiver Operating Characteristic Curve')
plt.xlabel('1-Specificity')
plt.ylabel('Sensitivity')
```

(c) Compute the minimal distance between the ROC curve and the "ideal" point (0,1), and output accuracy, misclassification rate, sensitivity, specificity, and cut-off that correspond to the minimal distance.

SAS

accuracy	misclassrate	sensitivity	specificity	distance	cutoff
0.985	0.015	0.863636	1	0.136364	0.01
0.985	0.015	0.863636	1	0.136364	0.02
0.985	0.015	0.863636	1	0.136364	0.03

SAS Code

```
/*REPORTING MEASURES FOR THE POINT ON ROC CURVE CLOSEST
TO THE IDEAL POINT (0,1)*/
proc sql;
select accuracy, misclassrate, sensitivity, specificity,
sqrt(oneminusspec**2+(1-sensitivity)**2) as distance, i*0.01 as cutoff
from measures
having distance=min(distance);
quit;
```

R Studio

```
      accuracy misclassrate sensitivity specificity   distance cutoff
 [1,] 0.9842105   0.01578947     0.84375   0.9971264 0.02442232   0.18
```

R Code

```
#REPORTING MEASURES FOR THE POINT ON ROC CURVE CLOSEST TO THE IDEAL POINT (0,1)
distance<- c()
for (i in 1:102)
  distance[i]<- oneminusspec[i]^2+(1-sensitivity[i])^2

measures<- cbind(accuracy, misclassrate, sensitivity, specificity, distance, cutoff)
min.dist<- min(distance)
print(measures[which(measures[,5]==min.dist),])
```

Python

```
accuracy misclassrate sensitivity specificity oneminusspec distance \ 0 0.95 0.05
0.842105 0.961326 0.038674 0.162562
```

Python Code

```
#REPORTING MEASURES FOR THE POINT ON ROC CURVE CLOSEST TO THE IDEAL POINT (0,1)
df=pandas.DataFrame({'accuracy': accuracy,'misclassrate':
misclassrate,'sensitivity':
sensitivity,'specificity': specificity, 'oneminusspec': oneminusspec,'distance':
distance,'cut-off': cutoff})
min_distance=min(distance)
optimal=df[df['distance']==min_distance]
print("Optimal\n {}".format(optimal))
```

(d) Compute the area under the ROC curve.

SAS

AUC
0.5

SAS Code

```
proc sort data=measures;
by oneminusspec;
run;

data AUC;
set measures;
lagx=lag(oneminusspec);
lagy=lag(sensitivity);
if lagx=. then lagx=0;
if lagy=. then lagy=0;
trapezoid=(oneminusspec-lagx)*(sensitivity+lagy)/2;
AUC+trapezoid;
run;

proc print data=AUC (firstobs=102) noobs;
var AUC;
run;
```

R Studio

```
[1] 0.9184626
```

R Code

```
#COMPUTING AREA UNDER THE ROC CURVE
sensitivity<- sensitivity[order(sensitivity)]
oneminusspec<- oneminusspec[order(oneminusspec)]


library(Hmisc) #Harrell Miscellaneous packages
lagx<- Lag(oneminusspec,shift=1)
lagy<- Lag(sensitivity, shift=1)
lagx[is.na(lagx)]<- 0
lagy[is.na(lagy)]<- 0
trapezoid<- (oneminusspec-lagx)*(sensitivity+lagy)/2
print(AUC<- sum(trapezoid))
```

Python

```
0.031041000290782203
```

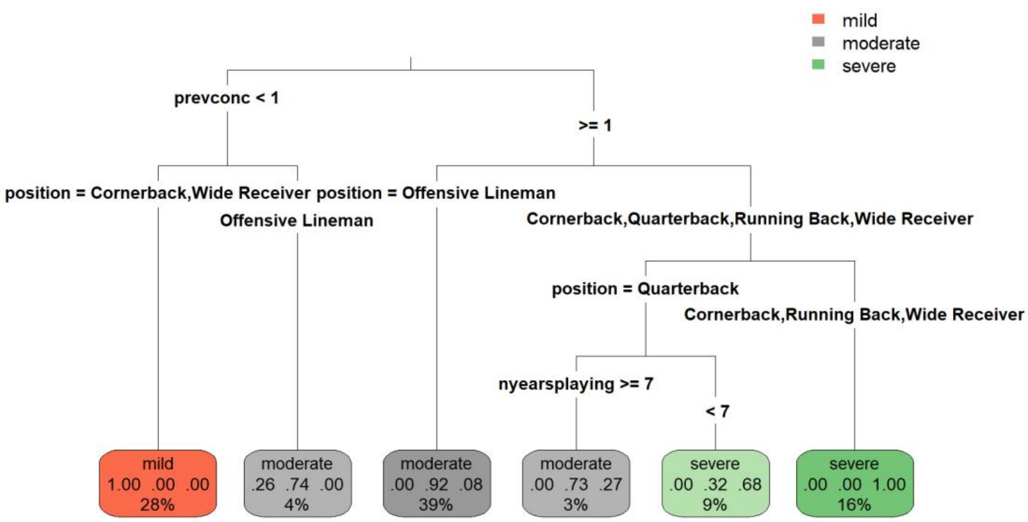
Python Code

```python
#COMPUTING AREA UNDER THE ROC CURVE
df=df.sort_values('oneminusspec', ascending=True)
df['lagx']=df['oneminusspec'].shift(1)
df['lagy']=df['sensitivity'].shift(1)
df['lagx']=numpy.nan_to_num(df['lagx'],nan=0)
df['lagy']=numpy.nan_to_num(df['lagy'],nan=0)
df['trapezoid']=(df['oneminusspec']-df['lagx'])*(df['sensitivity']+df['lagy'])/2;
AUC=sum(df['trapezoid'])
print("AUC\n {}".format(AUC))
```

Problem 5. Head concussions are not uncommon for full-contact sports like American football. The data file "concussions_data.csv" contains measurements for high-school and college aged football players who experienced a concussion during a practice or game and were admitted to a specialized clinic for treatment. The file contains patients' age, number of years playing football, position on the field, number of previous concussions, and current concussion grade (mild, moderate, or severe).

(a) Build a multinomial classification tree using the Gini splitting criterion. Compute the prediction performance measures (the number of true positive, false positive, true negative, and false negative predictions, accuracy, misclassification rate, sensitivity, false negative rate, specificity, false positive rate, precision, negative predictive value, and F1-score) for each individual class. Combine the values into micro classification measures, macro classification measures, and weighted macro classification measures.

R Studio



MICRO MEASURES

```
CLASS MEASURES:
class:moderate
[1] "tp: 49"
[1] "fp: 8"
[1] "tn: 39"
[1] "fn: 2"
[1] "accuracy: 0.897959183673469"
```

```
[1] "misclassrate: 0.102040816326531"
[1] "sensitivity: 0.96078431372549"
[1] "FNR: 0.0392156862745098"
[1] "specificity: 0.829787234042553"
[1] "FPR: 0.170212765957447"
[1] "precision: 0.859649122807018"
[1] "NPV: 0.951219512195122"
[1] "F1score: 0.907407407407407"
> class.metrics(class='mild')
CLASS MEASURES:
class:mild
[1] "tp: 20"
[1] "fp: 0"
[1] "tn: 77"
[1] "fn: 1"
[1] "accuracy: 0.989795918367347"
[1] "misclassrate: 0.0102040816326531"
[1] "sensitivity: 0.952380952380952"
[1] "FNR: 0.0476190476190476"
[1] "specificity: 1"
[1] "FPR: 0"
[1] "precision: 1"
[1] "NPV: 0.987179487179487"
[1] "F1score: 0.975609756097561"
> class.metrics(class='severe')
CLASS MEASURES:
class:severe
[1] "tp: 19"
[1] "fp: 2"
[1] "tn: 70"
[1] "fn: 7"
[1] "accuracy: 0.908163265306122"
[1] "misclassrate: 0.0918367346938776"
[1] "sensitivity: 0.730769230769231"
[1] "FNR: 0.269230769230769"
[1] "specificity: 0.972222222222222"
[1] "FPR: 0.0277777777777778"
[1] "precision: 0.904761904761905"
[1] "NPV: 0.909090909090909"
[1] "F1score: 0.808510638297872"


MACRO MEASURES:
> print(paste('accuracy:', accuracy.macro<- mean(accuracy)))
[1] "accuracy: 0.931972789115646"
> print(paste('misclassrate:', misclassrate.macro<- mean(misclassrate)))
[1] "misclassrate: 0.0680272108843537"
> print(paste('sensitivity:', sensitivity.macro<- mean(sensitivity)))
[1] "sensitivity: 0.881311498958558"
> print(paste('FNR:', FNR.macro<- mean(FNR)))
[1] "FNR: 0.118688501041442"
> print(paste('specificity:', specificity.macro<- mean(specificity)))
[1] "specificity: 0.934003152088258"
> print(paste('FPR:', FPR.macro<- mean(FPR)))
[1] "FPR: 0.0659968479117415"
> print(paste('precision:', precision.macro<- mean(precision, na.rm=TRUE)))
[1] "precision: 0.921470342522974"
> print(paste('NPV:', NPV.macro<- mean(NPV)))
[1] "NPV: 0.949163302821839"
> print(paste('F1-score:', F1score.macro<- mean(F1score)))
[1] "F1-score: 0.89717593393428"
```

```
WEIGHTED MACRO MEASURES:
[1] "accuracy: 0.920345689296127"
[1] "misclassrate: 0.0796543107038734"
[1] "sensitivity: 0.897959183673469"
[1] "FNR: 0.102040816326531"
[1] "specificity: 0.904050272591306"
[1] "FPR: 0.095949727408694"
[1] "precision: 0.90169300803028"
[1] "NPV: 0.947748244786572"
[1] "F1-score: 0.895784278077258"
```

<u>R Code</u>

```
concuss <- read.csv("C:/Users/saedw/OneDrive/Desktop/STAT 574 Data Mining/HW1STAT574S23/DATA SETS/concussions_data.csv")

# a) gini tree and compute micro and macro classification measures

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
set.seed(102938)
sample <- sample(c(TRUE, FALSE), nrow(concuss),replace=TRUE, prob=c(0.8,0.2))
train<- concuss[sample,]
test<- concuss[!sample,]

#FITTING PRUNED MULTINOMIAL CLASSIFICATION TREE WITH GINI SPLITTING
library(rpart)
tree.gini<- rpart(concussion ~ nyearsplaying + position + prevconc,
data=train, method="class", parms=list(split="Gini"), maxdepth=4)

#PLOTTING FITTED TREE
library(rpart.plot)
rpart.plot(tree.gini, type=3)
message('MICRO MEASURES:')
print(paste('accuracy:', accuracy.micro<- (tp.sum+tn.sum)/(tp.sum+fp.sum+tn.sum+fn.sum)))
print(paste('misclassrate:', misclassrate.micro<- (fp.sum+fn.sum)/(tp.sum+fp.sum+tn.sum+fn.sum)))
print(paste('sensitivity:', sensitivity.micro<- tp.sum/(tp.sum+fn.sum)))
print(paste('FNR:', FNR.micro<- fn.sum/(tp.sum+fn.sum)))
print(paste('specificity:', specificity.micro<- tn.sum/(fp.sum+tn.sum)))
print(paste('FPR:', FPR.micro<- fp.sum/(fp.sum+tn.sum)))
print(paste('precision:', precision.micro<- tp.sum/(tp.sum+fp.sum)))
print(paste('NPV:', NPV.micro<- tn.sum/(fn.sum+tn.sum)))
print(paste('F1-score:', F1score.micro<- 2*tp.sum/(2*tp.sum+fn.sum+fp.sum)))

#COMPUTING MACRO MEASURES
message('MACRO MEASURES:')
print(paste('accuracy:', accuracy.macro<- mean(accuracy)))
```

```
print(paste('misclassrate:', misclassrate.macro<- mean(misclassrate)))
print(paste('sensitivity:', sensitivity.macro<- mean(sensitivity)))
print(paste('FNR:', FNR.macro<- mean(FNR)))
print(paste('specificity:', specificity.macro<- mean(specificity)))
print(paste('FPR:', FPR.macro<- mean(FPR)))
print(paste('precision:', precision.macro<- mean(precision, na.rm=TRUE)))
print(paste('NPV:', NPV.macro<- mean(NPV)))
print(paste('F1-score:', F1score.macro<- mean(F1score)))

#COMPUTING PREDICTED VALUES FOR TESTING DATA
pred.values<- predict(tree.gini, test)

#DETERMINING PREDICTED CLASSES
test<- cbind(test, pred.values)
test$maxprob<- pmax(test$'mild',test$'moderate',test$'severe')

test$predclass<- ifelse(test$maxprob==test$'mild', 'mild',
ifelse(test$maxprob==test$'moderate','moderate', 'severe'))


#COMPUTING PERFORMANCE MEASURES FOR INDIVIDUAL CLASSES

tp<- c()
fp<- c()
tn<- c()
fn<- c()
accuracy<- c()
misclassrate<- c()
sensitivity<- c()
FNR<- c()
specificity<- c()
FPR<- c()
precision<- c()
NPV<- c()
F1score<- c()


class.metrics<- function(class) {

  tp.class<- ifelse(test$predclass==class & test$concussion==class,1,0)
  fp.class<- ifelse(test$predclass==class & test$concussion!=class,1,0)
  tn.class<- ifelse(test$predclass!=class & test$concussion!=class,1,0)
  fn.class<- ifelse(test$predclass!=class & test$concussion==class,1,0)
```

```
  message('CLASS MEASURES:')
  message('class:', class)
  print(paste('tp:', tp[class]<<- sum(tp.class)))
  #<<- is global assignment, works outside the function
  print(paste('fp:', fp[class]<<- sum(fp.class)))
  print(paste('tn:', tn[class]<<- sum(tn.class)))
  print(paste('fn:', fn[class]<<- sum(fn.class)))
  total<<- nrow(test)

  print(paste('accuracy:', accuracy[class]<<- (tp[class]+tn[class])/total))
  print(paste('misclassrate:', misclassrate[class]<<- (fp[class]+fn[class])/total))
  print(paste('sensitivity:', sensitivity[class]<<- tp[class]/(tp[class]+fn[class])))
  print(paste('FNR:', FNR[class]<<- fn[class]/(tp[class]+fn[class])))
  print(paste('specificity:', specificity[class]<<- tn[class]/(fp[class]+tn[class])))
  print(paste('FPR:', FPR[class]<<- fp[class]/(fp[class]+tn[class])))
  print(paste('precision:', precision[class]<<- tp[class]/(tp[class]+fp[class])))
  print(paste('NPV:', NPV[class]<<- tn[class]/(fn[class]+tn[class])))
  print(paste('F1score:', F1score[class]<<- 2*tp[class]/(2*tp[class]+fn[class]+fp[class])))
}

class.metrics(class='moderate')
class.metrics(class='mild')
class.metrics(class='severe')


#COMPUTING MICRO MEASURES
tp.sum<- sum(tp)
fp.sum<- sum(fp)
tn.sum<- sum(tn)
fn.sum<- sum(fn)


message('MICRO MEASURES:')
print(paste('accuracy:', accuracy.micro<- (tp.sum+tn.sum)/(tp.sum+fp.sum+tn.sum+fn.sum)))
print(paste('misclassrate:', misclassrate.micro<- (fp.sum+fn.sum)/(tp.sum+fp.sum+tn.sum+fn.sum)))
print(paste('sensitivity:', sensitivity.micro<- tp.sum/(tp.sum+fn.sum)))
print(paste('FNR:', FNR.micro<- fn.sum/(tp.sum+fn.sum)))
print(paste('specificity:', specificity.micro<- tn.sum/(fp.sum+tn.sum)))
print(paste('FPR:', FPR.micro<- fp.sum/(fp.sum+tn.sum)))
print(paste('precision:', precision.micro<- tp.sum/(tp.sum+fp.sum)))
print(paste('NPV:', NPV.micro<- tn.sum/(fn.sum+tn.sum)))
print(paste('F1-score:', F1score.micro<- 2*tp.sum/(2*tp.sum+fn.sum+fp.sum)))

#COMPUTING MACRO MEASURES
```

```
message('MACRO MEASURES:')
print(paste('accuracy:', accuracy.macro<- mean(accuracy)))
print(paste('misclassrate:', misclassrate.macro<- mean(misclassrate)))
print(paste('sensitivity:', sensitivity.macro<- mean(sensitivity)))
print(paste('FNR:', FNR.macro<- mean(FNR)))
print(paste('specificity:', specificity.macro<- mean(specificity)))
print(paste('FPR:', FPR.macro<- mean(FPR)))
print(paste('precision:', precision.macro<- mean(precision, na.rm=TRUE)))
print(paste('NPV:', NPV.macro<- mean(NPV)))
print(paste('F1-score:', F1score.macro<- mean(F1score)))

#COMPUTING WEIGHTED MACRO MEASURES
weight<- c()

for (class in 1:3)
weight[class]<- (tp[class]+fn[class])/total

message('WEIGHTED MACRO MEASURES:')
print(paste('accuracy:', accuracy.wmacro<- weight%*%accuracy))
print(paste('misclassrate:', misclassrate.wmacro<- weight%*%misclassrate))
print(paste('sensitivity:', sensitivity.wmacro<- weight%*%sensitivity))
print(paste('FNR:', FNR.wmacro<- weight%*%FNR))
print(paste('specificity:', specificity.wmacro<- weight%*%specificity))
print(paste('FPR:', FPR.wmacro<- weight%*%FPR))
precision[is.na(precision)]<- 0
print(paste('precision:', precision.wmacro<- weight%*%precision))
print(paste('NPV:', NPV.wmacro<- weight%*%NPV))
print(paste('F1-score:', F1score.wmacro<- weight%*%F1score))
```

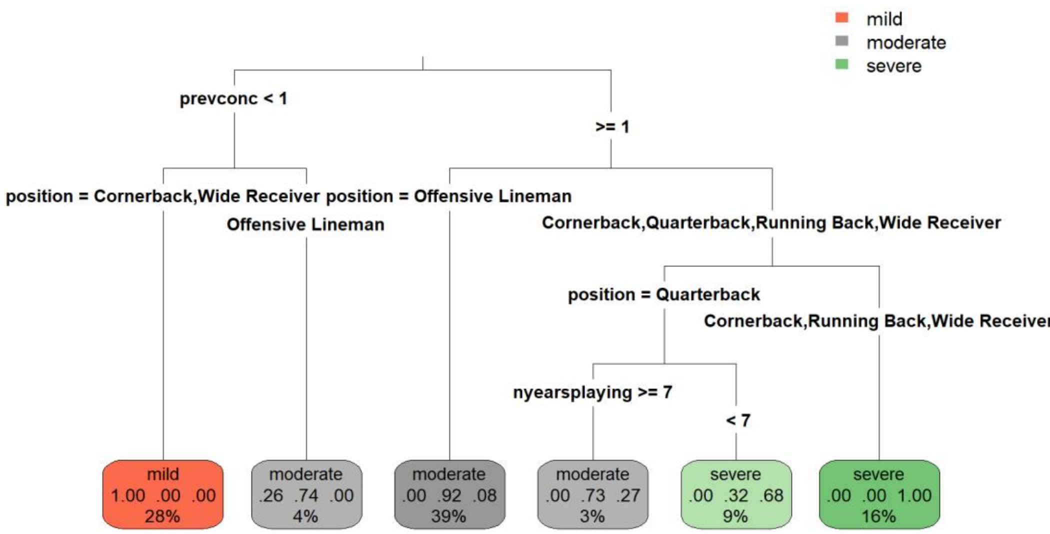
```
weight<- c()

for (class in 1:3)
weight[class]<- (tp[class]+fn[class])/total

message('WEIGHTED MACRO MEASURES:')
print(paste('accuracy:', accuracy.wmacro<- weight%*%accuracy))
print(paste('misclassrate:', misclassrate.wmacro<- weight%*%misclassrate))
print(paste('sensitivity:', sensitivity.wmacro<- weight%*%sensitivity))
print(paste('FNR:', FNR.wmacro<- weight%*%FNR))
print(paste('specificity:', specificity.wmacro<- weight%*%specificity))
```

```
print(paste('FPR:', FPR.wmacro<- weight%*%FPR))
precision[is.na(precision)]<- 0
print(paste('precision:', precision.wmacro<- weight%*%precision))
print(paste('NPV:', NPV.wmacro<- weight%*%NPV))
print(paste('F1-score:', F1score.wmacro<- weight%*%F1score))
```

Python

(b) Build a multinomial classification tree based on the entropy splitting criterion and compute the performance measures listed in part (a).

R Studio




```
CLASS MEASURES:
class:moderate
[1] "tp: 49"
[1] "fp: 8"
[1] "tn: 39"
[1] "fn: 2"
[1] "accuracy: 0.897959183673469"
[1] "misclassrate: 0.102040816326531"
[1] "sensitivity: 0.96078431372549"
[1] "FNR: 0.0392156862745098"
[1] "specificity: 0.829787234042553"
[1] "FPR: 0.170212765957447"
[1] "precision: 0.859649122807018"
```

```
[1] "NPV: 0.951219512195122"
[1] "F1score: 0.907407407407407"
CLASS MEASURES:
class:mild
[1] "tp: 20"
[1] "fp: 0"
[1] "tn: 77"
[1] "fn: 1"
[1] "accuracy: 0.989795918367347"
[1] "misclassrate: 0.0102040816326531"
[1] "sensitivity: 0.952380952380952"
[1] "FNR: 0.0476190476190476"
[1] "specificity: 1"
[1] "FPR: 0"
[1] "precision: 1"
[1] "NPV: 0.987179487179487"
[1] "F1score: 0.975609756097561"
CLASS MEASURES:
class:severe
[1] "tp: 19"
[1] "fp: 2"
[1] "tn: 70"
[1] "fn: 7"
[1] "accuracy: 0.908163265306122"
[1] "misclassrate: 0.0918367346938776"
[1] "sensitivity: 0.730769230769231"
[1] "FNR: 0.269230769230769"
[1] "specificity: 0.972222222222222"
[1] "FPR: 0.0277777777777778"
[1] "precision: 0.904761904761905"
[1] "NPV: 0.909090909090909"
[1] "F1score: 0.808510638297872"
MICRO MEASURES:
[1] "accuracy: 0.931972789115646"
[1] "misclassrate: 0.0680272108843537"
[1] "sensitivity: 0.897959183673469"
[1] "FNR: 0.102040816326531"
[1] "specificity: 0.948979591836735"
[1] "FPR: 0.0510204081632653"
[1] "precision: 0.897959183673469"
[1] "NPV: 0.948979591836735"
[1] "F1-score: 0.897959183673469"
MACRO MEASURES:
[1] "accuracy: 0.931972789115646"
[1] "misclassrate: 0.0680272108843537"
[1] "sensitivity: 0.881311498958558"
[1] "FNR: 0.118688501041442"
[1] "specificity: 0.934003152088258"
[1] "FPR: 0.0659968479117415"
[1] "precision: 0.921470342522974"
[1] "NPV: 0.949163302821839"
[1] "F1-score: 0.89717593393428"
WEIGHTED MACRO MEASURES:
[1] "accuracy: 0.920345689296127"
[1] "misclassrate: 0.0796543107038734"
[1] "sensitivity: 0.897959183673469"
[1] "FNR: 0.102040816326531"
[1] "specificity: 0.904050272591306"
[1] "FPR: 0.095949727408694"
[1] "precision: 0.90169300803028"
[1] "NPV: 0.947748244786572"
[1] "F1-score: 0.895784278077258"
WEIGHTED MACRO MEASURES:
[1] "accuracy: 0.920345689296127"
```

```
[1] "misclassrate: 0.0796543107038734"
[1] "sensitivity: 0.897959183673469"
[1] "FNR: 0.102040816326531"
[1] "specificity: 0.904050272591306"
[1] "FPR: 0.095949727408694"
[1] "precision: 0.90169300803028"
[1] "NPV: 0.947748244786572"
[1] "F1-score: 0.895784278077258"
```

R Code

```
tree.entropy<- rpart(concussion ~ nyearsplaying + position + prevconc,
data=train, method="class", parms=list(split="entropy"), maxdepth=4)

#PLOTTING FITTED TREE
rpart.plot(tree.entropy, type=3)
#Note: same as tree.gini

#COMPUTING PREDICTED VALUES FOR TESTING DATA
pred.values<- predict(tree.entropy, test)

#DETERMINING PREDICTED CLASSES
test<- cbind(test, pred.values)
test$maxprob<- pmax(test$'mild',test$'moderate',test$'severe')

test$predclass<- ifelse(test$maxprob==test$'mild', 'mild',
ifelse(test$maxprob==test$'moderate','moderate', 'severe'))


#COMPUTING PERFORMANCE MEASURES FOR INDIVIDUAL CLASSES

tp<- c()
fp<- c()
tn<- c()
fn<- c()
accuracy<- c()
misclassrate<- c()
sensitivity<- c()
FNR<- c()
specificity<- c()
FPR<- c()
precision<- c()
NPV<- c()
F1score<- c()


class.metrics<- function(class) {
```

```
  tp.class<- ifelse(test$predclass==class & test$concussion==class,1,0)
  fp.class<- ifelse(test$predclass==class & test$concussion!=class,1,0)
  tn.class<- ifelse(test$predclass!=class & test$concussion!=class,1,0)
  fn.class<- ifelse(test$predclass!=class & test$concussion==class,1,0)

  message('CLASS MEASURES:')
  message('class:', class)
  print(paste('tp:', tp[class]<<- sum(tp.class)))
  #<<- is global assignment, works outside the function
  print(paste('fp:', fp[class]<<- sum(fp.class)))
  print(paste('tn:', tn[class]<<- sum(tn.class)))
  print(paste('fn:', fn[class]<<- sum(fn.class)))
  total<<- nrow(test)

  print(paste('accuracy:', accuracy[class]<<- (tp[class]+tn[class])/total))
  print(paste('misclassrate:', misclassrate[class]<<- (fp[class]+fn[class])/total))
  print(paste('sensitivity:', sensitivity[class]<<- tp[class]/(tp[class]+fn[class])))
  print(paste('FNR:', FNR[class]<<- fn[class]/(tp[class]+fn[class])))
  print(paste('specificity:', specificity[class]<<- tn[class]/(fp[class]+tn[class])))
  print(paste('FPR:', FPR[class]<<- fp[class]/(fp[class]+tn[class])))
  print(paste('precision:', precision[class]<<- tp[class]/(tp[class]+fp[class])))
  print(paste('NPV:', NPV[class]<<- tn[class]/(fn[class]+tn[class])))
  print(paste('F1score:', F1score[class]<<- 2*tp[class]/(2*tp[class]+fn[class]+fp[class])))
}

class.metrics(class='moderate')
class.metrics(class='mild')
class.metrics(class='severe')


#COMPUTING MICRO MEASURES
tp.sum<- sum(tp)
fp.sum<- sum(fp)
tn.sum<- sum(tn)
fn.sum<- sum(fn)


message('MICRO MEASURES:')
print(paste('accuracy:', accuracy.micro<- (tp.sum+tn.sum)/(tp.sum+fp.sum+tn.sum+fn.sum)))
print(paste('misclassrate:', misclassrate.micro<- (fp.sum+fn.sum)/(tp.sum+fp.sum+tn.sum+fn.sum)))
print(paste('sensitivity:', sensitivity.micro<- tp.sum/(tp.sum+fn.sum)))
print(paste('FNR:', FNR.micro<- fn.sum/(tp.sum+fn.sum)))
print(paste('specificity:', specificity.micro<- tn.sum/(fp.sum+tn.sum)))
```

```
print(paste('FPR:', FPR.micro<- fp.sum/(fp.sum+tn.sum)))
print(paste('precision:', precision.micro<- tp.sum/(tp.sum+fp.sum)))
print(paste('NPV:', NPV.micro<- tn.sum/(fn.sum+tn.sum)))
print(paste('F1-score:', F1score.micro<- 2*tp.sum/(2*tp.sum+fn.sum+fp.sum)))

#COMPUTING MACRO MEASURES
message('MACRO MEASURES:')
print(paste('accuracy:', accuracy.macro<- mean(accuracy)))
print(paste('misclassrate:', misclassrate.macro<- mean(misclassrate)))
print(paste('sensitivity:', sensitivity.macro<- mean(sensitivity)))
print(paste('FNR:', FNR.macro<- mean(FNR)))
print(paste('specificity:', specificity.macro<- mean(specificity)))
print(paste('FPR:', FPR.macro<- mean(FPR)))
print(paste('precision:', precision.macro<- mean(precision, na.rm=TRUE)))
print(paste('NPV:', NPV.macro<- mean(NPV)))
print(paste('F1-score:', F1score.macro<- mean(F1score)))

#COMPUTING WEIGHTED MACRO MEASURES
weight<- c()

for (class in 1:3)
weight[class]<- (tp[class]+fn[class])/total

message('WEIGHTED MACRO MEASURES:')
print(paste('accuracy:', accuracy.wmacro<- weight%*%accuracy))
print(paste('misclassrate:', misclassrate.wmacro<- weight%*%misclassrate))
print(paste('sensitivity:', sensitivity.wmacro<- weight%*%sensitivity))
print(paste('FNR:', FNR.wmacro<- weight%*%FNR))
print(paste('specificity:', specificity.wmacro<- weight%*%specificity))
print(paste('FPR:', FPR.wmacro<- weight%*%FPR))
precision[is.na(precision)]<- 0
print(paste('precision:', precision.wmacro<- weight%*%precision))
print(paste('NPV:', NPV.wmacro<- weight%*%NPV))
print(paste('F1-score:', F1score.wmacro<- weight%*%F1score))
```

```
weight<- c()

for (class in 1:3)
weight[class]<- (tp[class]+fn[class])/total
```

```
message('WEIGHTED MACRO MEASURES:')
print(paste('accuracy:', accuracy.wmacro<- weight%*%accuracy))
print(paste('misclassrate:', misclassrate.wmacro<- weight%*%misclassrate))
print(paste('sensitivity:', sensitivity.wmacro<- weight%*%sensitivity))
print(paste('FNR:', FNR.wmacro<- weight%*%FNR))
print(paste('specificity:', specificity.wmacro<- weight%*%specificity))
print(paste('FPR:', FPR.wmacro<- weight%*%FPR))
precision[is.na(precision)]<- 0
print(paste('precision:', precision.wmacro<- weight%*%precision))
print(paste('NPV:', NPV.wmacro<- weight%*%NPV))
print(paste('F1-score:', F1score.wmacro<- weight%*%F1score))
```

(c) Built a multinomial classification tree based on the CHAID splitting criterion and compute the performance measures listed in part (a).

R Studio