KU7 - Prepare a report documenting the decisions taken with supporting evidence.

Decision Tree

Step 1: I researched on the best way to configure this model.

I used these two articles to help me find the best configuration for the decision tree:

https://rpubs.com/maulikpatel/229337

http://dataaspirant.com/2017/02/03/decision-tree-classifier-implementation-in-r/

These articles both used the train and trainControl functions from the caret library in r to find out how best to configure the decision tree to output the best results. The code used is show in the last step.

Step 2: I analysed the data

I cleaned and explored the data and since I know that corplots and decision trees require numeric values, I converted my data.

Below, I am renaming the columns to make more sense visually.

```
# Data cleaning -------------------------------------------------------------

cars <- read.csv("C:\\Users\\Fran\\Documents\\Bsc\\3BSC\\3bsc_pendrive\\FirstSem\\ACI\\Assignment\\ACI-A02\\Q01\\cars.csv", sep=",")

str(cars)

names(cars) <- c("BuyingPrice", "PriceOfMaintenance", "Doors", "Capacity",
                 "SizeOfBoot", "EstimatedSafety", "class")

summary(cars)
```

Figure 1: Renaming column names

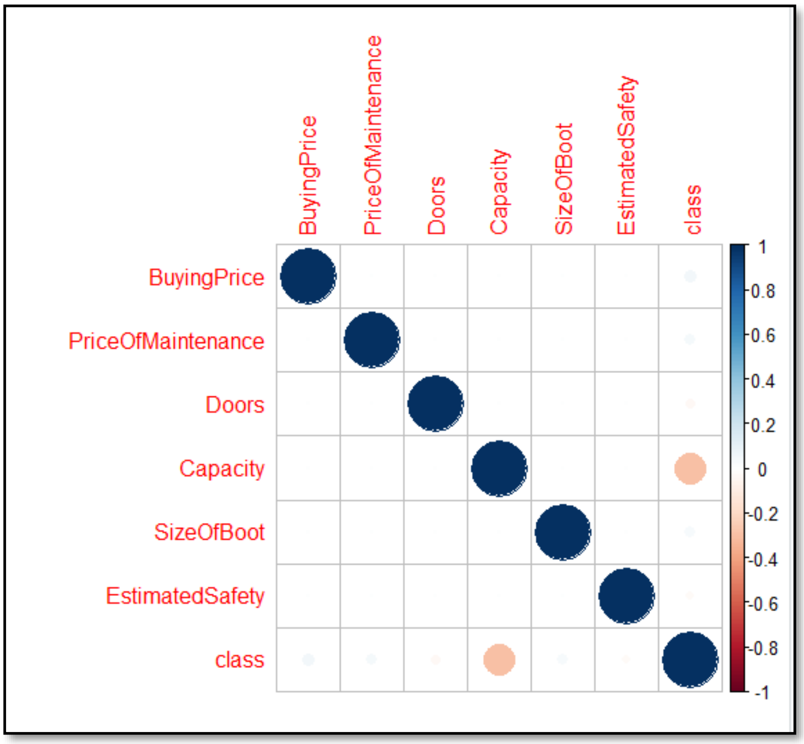
```
> summary(cars)
  BuyingPrice   PriceOfMaintenance      Doors          Capacity    SizeOfBoot EstimatedSafety class
 Min.   :1.00   Min.   :1.00       Min.   :1.00   Min.   :1   Min.   :1   Min.   :1       1: 384
 1st Qu.:1.75   1st Qu.:1.75       1st Qu.:1.75   1st Qu.:1   1st Qu.:1   1st Qu.:1       2:  69
 Median :2.50   Median :2.50       Median :2.50   Median :2   Median :2   Median :2       3:1210
 Mean   :2.50   Mean   :2.50       Mean   :2.50   Mean   :2   Mean   :2   Mean   :2       4:  65
 3rd Qu.:3.25   3rd Qu.:3.25       3rd Qu.:3.25   3rd Qu.:3   3rd Qu.:3   3rd Qu.:3
 Max.   :4.00   Max.   :4.00       Max.   :4.00   Max.   :3   Max.   :3   Max.   :3
>
```

Figure 2:Summary of data attributes

```r
#checks for na's
apply(cars,2,function(x) sum(is.na(x)))


#Convert all to numeric for corrplot usage.
cars$BuyingPrice <- as.numeric(cars$BuyingPrice)
cars$PriceOfMaintenance <- as.numeric(cars$PriceOfMaintenance)
cars$SizeOfBoot <- as.numeric(cars$SizeOfBoot)
cars$EstimatedSafety <- as.numeric(cars$EstimatedSafety)
cars$class <- as.numeric(cars$class)
cars$Capacity <- as.numeric(cars$Capacity)
cars$Doors <- as.numeric(cars$Doors)
```

Figure 3: Checking for any NA values



Step 3: Applying what I learnt

```
# Decision tree - Prediction ---------------------------------------------

cars.model_dt70 <- C5.0(cars.train_dt70[-7],cars.train_dt70$class, trials=10)
summary(cars.model_dt70)

cars.model_dt75 <- C5.0(cars.train_dt75[-7],cars.train_dt75$class, trials=10)
summary(cars.model_dt75)

cars.model_dt80 <- C5.0(cars.train_dt80[-7],cars.train_dt80$class, trials=10)
summary(cars.model_dt80)

cars.predict_dt70 <- predict(cars.model_dt70, cars.test_dt70[-7])
cars.predict_dt75 <- predict(cars.model_dt75, cars.test_dt75[-7])
cars.predict_dt80 <- predict(cars.model_dt80, cars.test_dt80[-7])

#Best
cars.best_model_dt <- C5.0(cars.train_dt75[-7],cars.train_dt75$class, trials=20, winnow=FALSE, model="rules")
cars.predict_best_model_dt <- predict(cars.best_model_dt, cars.test_dt75)
```

Figure 4: Creating decision trees

I chose to create 3 samples, 70%, 75% and 80% and after splitting them into train and test, I created my decision trees with a default of 10 trails and one with 20 trails since my research lead me to find the supposedly best configuration.

```
#Inbuilt (in caret): Accuracy Metric
control <- trainControl(method="repeatedcv", number=5, repeats=5)
set.seed(123)
fit.c50 <- caret::train(class~., data=cars, method="C5.0", metric='Accuracy', trControl=control)
fit.c50 #The final values used for the model were trials = 20, model = rules and winnow = FALSE which obtained accuracy of 99%.
trellis.par.set(caretTheme())
plot(fit.c50, metric='Accuracy')
```

Figure 5: Testing which configuration should be used

```
1728 samples
   6 predictor
   4 classes: '1', '2', '3', '4'

No pre-processing
Resampling: Cross-Validated (5 fold, repeated 5 times)
Summary of sample sizes: 1383, 1383, 1382, 1382, 1382, 1382, ...
Resampling results across tuning parameters:

  model  winnow  trials  Accuracy      Kappa
  rules  FALSE    1      0.9725716562  0.9403990001
  rules  FALSE   10      0.9888883246  0.9757404942
  rules  FALSE   20      0.9896972361  0.9775003784
  rules   TRUE    1      0.9675931859  0.9303633136
  rules   TRUE   10      0.9708311913  0.9360057765
  rules   TRUE   20      0.9734931647  0.9422448673
  tree   FALSE    1      0.9656255126  0.9255746668
  tree   FALSE   10      0.9832138534  0.9634091916
  tree   FALSE   20      0.9854130698  0.9682792137
  tree    TRUE    1      0.9633086826  0.9212366537
  tree    TRUE   10      0.9688621737  0.9321870731
  tree    TRUE   20      0.9731443255  0.9420353360

Accuracy was used to select the optimal model using the largest value.
The final values used for the model were trials = 20, model = rules and winnow = FALSE.
```

Figure 6:Result

The result is seen the picture above, and it says that 20 trials, winnow false outputs the best accuracy. Which in fact it does, and I obtained this result using the information found from my research.

Neural network

Step 1: Normalization

I used this article to help me understand which normalization methods should be used: https://datascienceplus.com/fitting-neural-network-in-r/

I chose to use the min-max method as this linearly transforms x to y=(x-min)/(max-min) and the entire range of values of X from min to max are mapped to range 0 to 1.

I found this to be the easiest to understand and use.

```
# Neural Network - Normalize --------------------------------------------

normalize <- function(x) {
  return (
    (x-min(x)) / (max(x)-min(x))
  )
}


carsNN <- cars
carsNN$BuyingPrice <- as.numeric(as.factor(carsNN$BuyingPrice))
carsNN$PriceOfMaintenance <- as.numeric(as.factor(carsNN$PriceOfMaintenance))
carsNN$SizeOfBoot <- as.numeric(as.factor(carsNN$SizeOfBoot))
carsNN$EstimatedSafety <- as.numeric(as.factor(carsNN$EstimatedSafety))
carsNN$class <- carsNN$class
carsNN$Capacity <- as.numeric(as.factor(carsNN$Capacity))
carsNN$Doors <- as.numeric(as.factor(carsNN$Doors))
```

Step 2: Creating dummy columns

I used this article to help me make dummy variables:

https://cran.r-project.org/web/packages/dummies/dummies.pdf

The reason I needed dummy variables is because even though the data contains one column for the attribute ‘class’, the attribute contains 4 different results, being: unacceptable, good, very good and acceptable.

In order for me to output the correct metrics, and classify as best possible, I would need to feed the neural network these 4 results as 4 separate classes, and in order to do that, I used the dummies library in R, which simply takes the attribute that you need to split and splits it accordingly wherever it finds a different result (so 4 different columns). Now each column is classified as 1 or 0, depending on the data.

```
newCols <- cbind(carsNN[,-7], dummy(carsNN$class))
colnames(newCols) <- c("BuyingPrice", "PriceOfMaintenance", "Doors", "Capacity",
                       "SizeOfBoot", "EstimatedSafety", "acc", "good", "unacc", "vgood")
```

Vehicle.r* newCols

Filter

	BuyingPrice	PriceOfMaintenance	Doors	Capacity	SizeOfBoot	EstimatedSafety	acc	good	unacc	vgood
1	4	4	1	1	3	2	0	0	1	0
2	4	4	1	1	3	3	0	0	1	0
3	4	4	1	1	3	1	0	0	1	0
4	4	4	1	1	2	2	0	0	1	0
5	4	4	1	1	2	3	0	0	1	0
6	4	4	1	1	2	1	0	0	1	0

Step 3: Hidden Layers

I used this article to choose the number of hidden layers:

https://www.r-bloggers.com/selecting-the-number-of-neurons-in-the-hidden-layer-of-a-neural-network/

In the article it stated that the important thing when creating neural networks is to choose a number of hidden neurons between 1 and the number of input variables.

So, I first tried with two hidden layers, one of 2 and one of 4.

```
cars_nn_Model <- neuralnet(class1 + class2 + class3 + class4~BuyingPrice+PriceOfMaintenance+Doors+Capacity+SizeOfBoot+EstimatedSafety
                          ,data=cars_train_nn,
                          hidden=c(2,4), linear.output = FALSE, threshold = 0.1)
```

This resulted in 74% accuracy.

I then tried another configuration, and used a sample of 60-20%, with 3 hidden layers of 6,6,4. These gave me better accuracy of 95%.