IE7275: Data Mining in Engineering

Homework-4

Prerit Samria
Seungyeon Ko

```
library(readxl)
library(rpart)
library(rpart.plot)
library(ggplot2)
library (caret)
```

```
## Loading required package: lattice
```

Problem-1

```
car.df <- read_excel("ToyotaCorolla.xlsx", sheet = "data")
```

```
str(car.df)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    1436 obs. of  39 variables:
##  $ Id               : num  1 2 3 4 5 6 7 8 9 10 ...
##  $ Model            : chr  "TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors" "TOYOTA Corolla 2.0 D4D HA
##  $ Price            : num  13500 13750 13950 14950 13750 ...
##  $ Age_08_04        : num  23 23 24 26 30 32 27 30 27 23 ...
##  $ Mfg_Month        : num  10 10 9 7 3 1 6 3 6 10 ...
##  $ Mfg_Year         : num  2002 2002 2002 2002 2002 ...
##  $ KM               : num  46986 72937 41711 48000 38500 ...
##  $ Fuel_Type        : chr  "Diesel" "Diesel" "Diesel" "Diesel" ...
##  $ HP               : num  90 90 90 90 90 90 90 90 192 69 ...
##  $ Met_Color        : num  1 1 1 0 0 0 1 1 0 0 ...
##  $ Color            : chr  "Blue" "Silver" "Blue" "Black" ...
##  $ Automatic        : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ CC               : chr  "2000" "2000" "2000" "2000" ...
##  $ Doors            : num  3 3 3 3 3 3 3 3 3 3 ...
##  $ Cylinders        : num  4 4 4 4 4 4 4 4 4 4 ...
##  $ Gears            : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ Quarterly_Tax    : num  210 210 210 210 210 210 210 210 100 185 ...
##  $ Weight           : num  1165 1165 1165 1165 1170 ...
##  $ Mfr_Guarantee    : num  0 0 1 1 1 0 0 1 0 0 ...
##  $ BOVAG_Guarantee  : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ Guarantee_Period : num  3 3 3 3 3 3 3 3 3 3 ...
##  $ ABS              : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ Airbag_1         : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ Airbag_2         : num  1 1 1 1 1 1 1 1 0 1 ...
##  $ Airco            : num  0 1 0 0 1 1 1 1 1 1 ...
##  $ Automatic_airco  : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Boardcomputer    : num  1 1 1 1 1 1 1 1 0 1 ...
##  $ CD_Player        : num  0 1 0 0 0 0 0 1 0 0 ...
##  $ Central_Lock     : num  1 1 0 0 1 1 1 1 1 0 ...
##  $ Powered_Windows  : num  1 0 0 0 1 1 1 1 1 0 ...
##  $ Power_Steering   : num  1 1 1 1 1 1 1 1 1 1 ...
```

```
##  $ Radio            : num  0 0 0 0 0 0 0 0 1 0 ...
##  $ Mistlamps        : num  0 0 0 0 1 1 0 0 0 0 ...
##  $ Sport_Model      : num  0 0 0 0 0 0 1 0 0 0 ...
##  $ Backseat_Divider : num  1 1 1 1 1 1 1 1 0 1 ...
##  $ Metallic_Rim     : num  0 0 0 0 0 0 0 0 1 0 ...
##  $ Radio_cassette   : num  0 0 0 0 0 0 0 0 1 0 ...
##  $ Parking_Assistant: num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Tow_Bar          : num  0 0 0 0 0 0 0 0 0 0 ...
```

We do not need predictors: ID and Model.

```
car.df <- car.df [, -c(1, 2)]
str(car.df)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    1436 obs. of  37 variables:
##  $ Price            : num  13500 13750 13950 14950 13750 ...
##  $ Age_08_04        : num  23 23 24 26 30 32 27 30 27 23 ...
##  $ Mfg_Month        : num  10 10 9 7 3 1 6 3 6 10 ...
##  $ Mfg_Year         : num  2002 2002 2002 2002 2002 ...
##  $ KM               : num  46986 72937 41711 48000 38500 ...
##  $ Fuel_Type        : chr  "Diesel" "Diesel" "Diesel" "Diesel" ...
##  $ HP               : num  90 90 90 90 90 90 90 90 192 69 ...
##  $ Met_Color        : num  1 1 1 0 0 0 1 1 0 0 ...
##  $ Color            : chr  "Blue" "Silver" "Blue" "Black" ...
##  $ Automatic        : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ CC               : chr  "2000" "2000" "2000" "2000" ...
##  $ Doors            : num  3 3 3 3 3 3 3 3 3 3 ...
##  $ Cylinders        : num  4 4 4 4 4 4 4 4 4 4 ...
##  $ Gears            : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ Quarterly_Tax    : num  210 210 210 210 210 210 210 210 100 185 ...
##  $ Weight           : num  1165 1165 1165 1165 1170 ...
##  $ Mfr_Guarantee    : num  0 0 1 1 1 0 0 1 0 0 ...
##  $ BOVAG_Guarantee  : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ Guarantee_Period : num  3 3 3 3 3 3 3 3 3 3 ...
##  $ ABS              : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ Airbag_1         : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ Airbag_2         : num  1 1 1 1 1 1 1 1 0 1 ...
##  $ Airco            : num  0 1 0 0 1 1 1 1 1 1 ...
##  $ Automatic_airco  : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Boardcomputer    : num  1 1 1 1 1 1 1 1 0 1 ...
##  $ CD_Player        : num  0 1 0 0 0 0 0 1 0 0 ...
##  $ Central_Lock     : num  1 1 0 0 1 1 1 1 1 0 ...
##  $ Powered_Windows  : num  1 0 0 0 1 1 1 1 1 0 ...
##  $ Power_Steering   : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ Radio            : num  0 0 0 0 0 0 0 0 1 0 ...
##  $ Mistlamps        : num  0 0 0 0 1 1 0 0 0 0 ...
##  $ Sport_Model      : num  0 0 0 0 0 0 1 0 0 0 ...
##  $ Backseat_Divider : num  1 1 1 1 1 1 1 1 0 1 ...
##  $ Metallic_Rim     : num  0 0 0 0 0 0 0 0 1 0 ...
##  $ Radio_cassette   : num  0 0 0 0 0 0 0 0 1 0 ...
##  $ Parking_Assistant: num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Tow_Bar          : num  0 0 0 0 0 0 0 0 0 0 ...
```

Converting categorical predictors with m classes into m dummies. Fuel_Type and Color are the categorical predictors.

```r
car.df$Fuel_Type <- as.factor(car.df$Fuel_Type)
summary(car.df$Fuel_Type)
```

```
##    CNG Diesel Petrol
##     17    155   1264
```

```r
car.df [, c("Fuel_Type_CNG", "Fuel_Type_Diesel", "Fuel_Type_Petrol")] <-
  model.matrix( ~ Fuel_Type - 1, data = car.df)
```

```r
car.df$Color <- as.factor (car.df$Color)
summary(car.df$Color)
```

```
##  Beige  Black   Blue  Green   Grey    Red Silver Violet  White Yellow
##      3    191    283    220    301    278    122      4     31      3
```

```r
car.df [, c("Color_Beige", "Color_Black", "Color_Blue", "Color_Green", "Color_Grey",
            "Color_Red", "Color_Silver", "Color_Violet", "Color_White", "Color_Yellow")] <-
  model.matrix( ~ Color - 1, data = car.df)
```

Partitioning the data into training (50%), validation (30%) and test (20%).

```r
set.seed(101)
train.index <- sample (row.names(car.df), 0.5 * dim (car.df)[1])
valid.index <- sample (setdiff(row.names(car.df), train.index), 0.3 * dim (car.df)[1])
test.index <- setdiff(row.names(car.df), union (train.index, valid.index))
train.df <- car.df [train.index, ]
valid.df <- car.df [valid.index, ]
test.df <- car.df [test.index, ]
```

Part-a

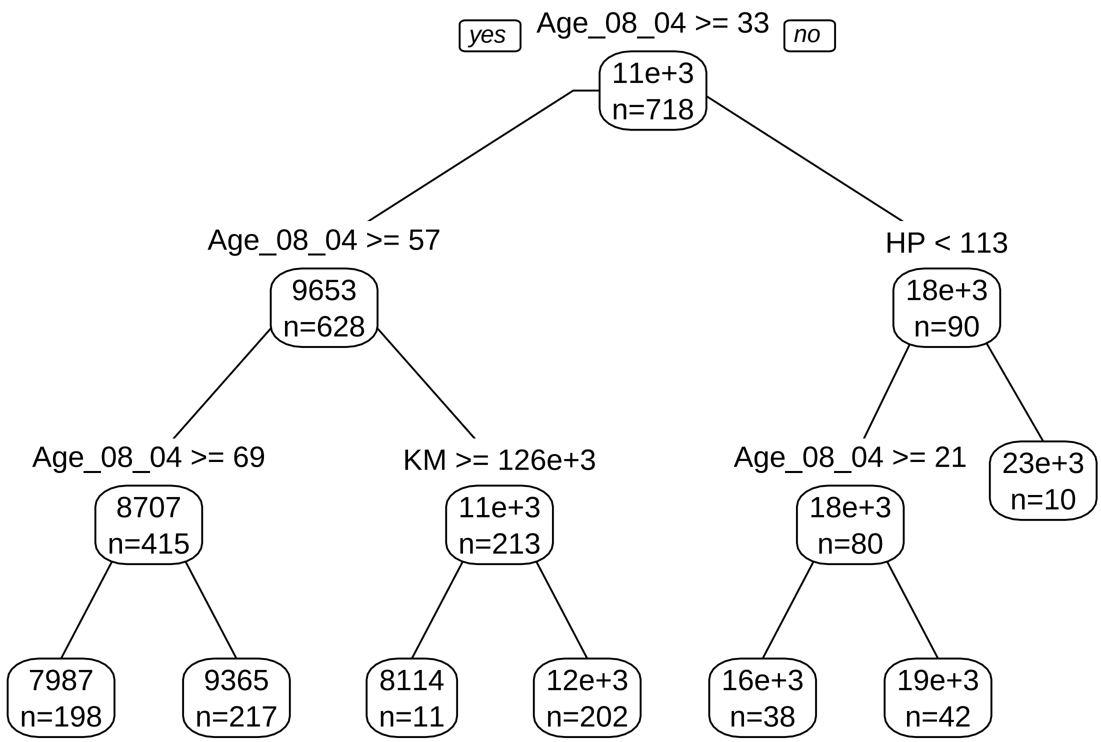
```r
car.tree <- rpart (Price ~ Age_08_04 + KM + Fuel_Type + HP + Automatic + Doors +
                     Quarterly_Tax +  Mfr_Guarantee + Guarantee_Period + Airco +
                     Automatic_airco + CD_Player + Powered_Windows + Sport_Model +
                     Tow_Bar, data = train.df, method="anova")
```

(i)

```r
# number of leaves in the tree
length_regression <-length(car.tree$frame$var [car.tree$frame$var == "<leaf>"])
length_regression
```

```
## [1] 7
```

```r
prp (car.tree, type = 1, extra = 1, split.font = 1, varlen = -10)
```




```
car.tree$variable.importance [1:4]
```

```
##       Age_08_04 Automatic_airco              KM              HP
##      7267193388      1951879777      1723410086      1016849241
```

Number of leaves in the tree is 7. Also, we can observe from the tree that the most important car specifications for predicting the price are Age_08_04, HP and KM. In addition, using the importance of variable for the tree, we see that the most important car specifications for predicting the price are Age_08_04, Automatic_airco, KM and HP.

(ii) The selected variables are: Price, Age_08_04, KM, Fuel_Type, HP, Automatic, Doors, Quarterly_Tax, Mfr_Guarantee, Guarantee_Period, Airco, Automatic_airco, CD_Player, Powered_Windows, Sport_Model, Tow_Bar.

```
selected.var <- c (1, 2, 5, 6, 7, 10, 12, 15, 17, 19, 23, 24, 26, 28, 32, 37)
```

```
car.df <- car.df [, selected.var]
```

Using the car.tree to predict Price for training, validation and test data.

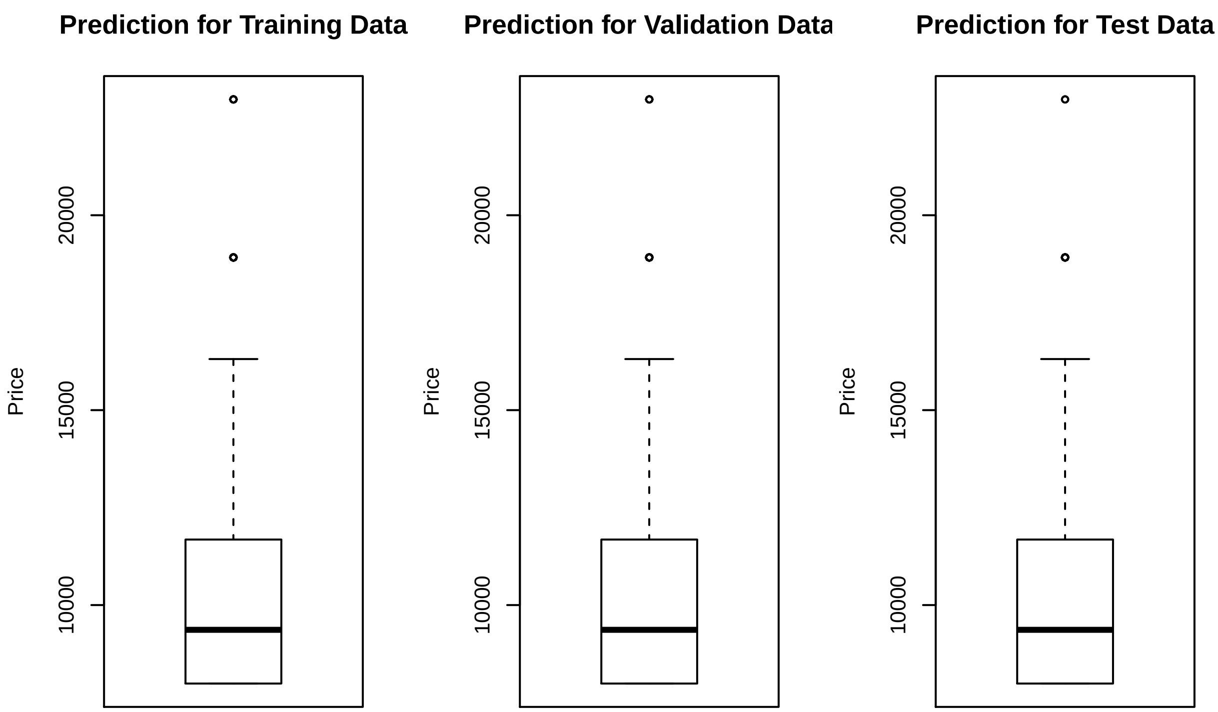
```
train.df <- train.df [, selected.var]
valid.df <- valid.df [, selected.var]
test.df <- test.df [, selected.var]
train.df_pred <- predict (car.tree, data = train.df)
valid.df_pred <- predict (car.tree, newdata = valid.df)
test.df_pred <- predict (car.tree, newdata = test.df)
```

We will use RMSE to calcualte the performance of the Regression Tree 'car.tree'

```
train.df_RMSE <- sqrt(sum((train.df$Price - as.array(train.df_pred))^2)/nrow(as.array(train.df_pred)))
valid.df_RMSE <- sqrt(sum((valid.df$Price - as.array(valid.df_pred))^2)/nrow(as.array(valid.df_pred)))
test.df_RMSE <- sqrt(sum((test.df$Price - as.array(test.df_pred))^2)/nrow(as.array(test.df_pred)))
```

Creating boxplots of the three RMSEs.

```r
par (mfrow = c(1, 3))
boxplot (train.df_pred, main = "Prediction for Training Data", ylab = "Price")
boxplot (valid.df_pred, main = "Prediction for Validation Data", ylab = "Price")
boxplot (test.df_pred, main = "Prediction for Test Data", ylab = "Price")
```




```r
par (mfrow = c(1, 1))
```

```r
RMSE <- data.frame(Training = train.df_RMSE, Validation = valid.df_RMSE,
                   Test = test.df_RMSE, row.names = "RMSE")
RMSE
```

```
##      Training Validation   Test
## RMSE  1361.91   1477.113 1386.8
```

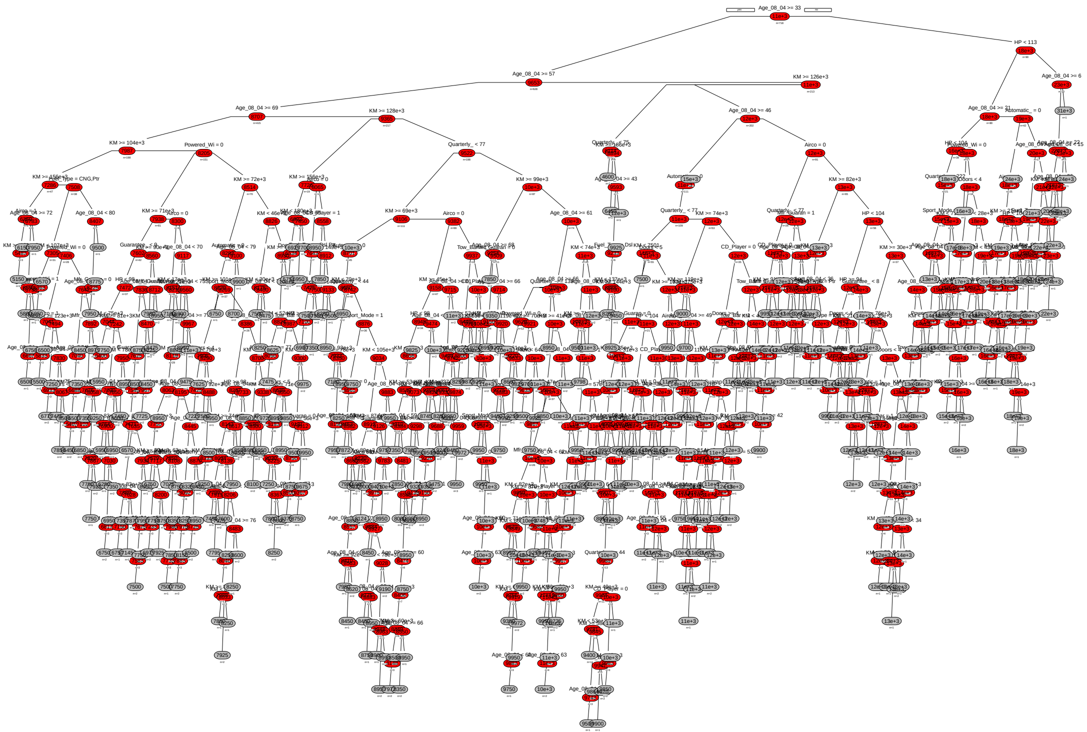
We can observe that the RMSE for training data is the smallest of the three datasets. The validation RMSE is larger than the training RMSE since the tree is modeled using the training data. The test RMSE data is larger than the training RMSE due to the same reason as for validation RMSE. However, the test RMSE is smaller than the validation RMSE.

****(iv)**** Now we are going to get the best pruned tree and a full grown tree.

```r
# full grown tree
car.deeper <- rpart(Price ~ ., data = train.df, method = "anova",
                    minbucket = 1, cp = 0)
```

```r
prp (car.deeper, type = 1, extra = 1, split.font = 1, varlen = -10, under = TRUE,
     box.col = ifelse (car.deeper$frame$var == "<leaf>", "gray", "red"))
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



```r
length (car.deeper$frame$var [car.deeper$frame$var == "<leaf>"])
```

```
## [1] 469
```

We can see that the number of leaves in the full grown tree is 469.

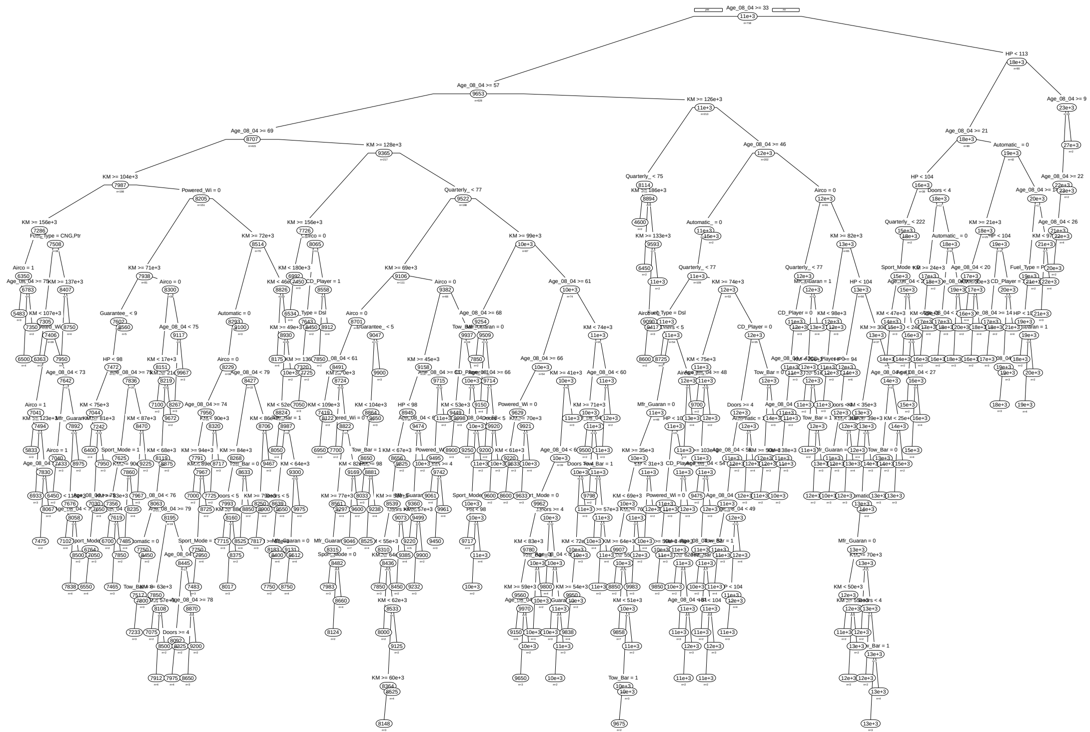
```r
deeper.pred_train <- predict(car.deeper, data = train.df)
deeper.pred_valid <- predict (car.deeper, newdata = valid.df)
deeper.pred_test <- predict (car.deeper, newdata = test.df)
```

```r
deeper.train.df_RMSE <- sqrt(sum((train.df$Price -
                                    as.array(deeper.pred_train))^2)/nrow(as.array(deeper.pred_train)))
deeper.valid.df_RMSE <- sqrt(sum((valid.df$Price -
                                    as.array(deeper.pred_valid))^2)/nrow(as.array(deeper.pred_valid)))
deeper.test.df_RMSE <- sqrt(sum((test.df$Price -
                                   as.array(deeper.pred_test))^2)/nrow(as.array(deeper.pred_test)))
```

```r
# best pruned tree
# using cross-validation: put xval = the number of partitions that you want
# cp: sets the smallest value for the complexity parameter
car.cross_validation <- rpart(Price ~ ., data = train.df, method = "anova",
                              minsplit = 5, xval = 5, cp = 0.00001)
```

```r
prp (car.cross_validation, type = 1, extra = 1, split.font = 1, varlen = -10, under = TRUE)
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



```r
# printcp: Displays the cp table for fitted rpart object.
printcp(car.cross_validation)
```

```
##
## Regression tree:
## rpart(formula = Price ~ ., data = train.df, method = "anova",
##     minsplit = 5, xval = 5, cp = 1e-05)
##
## Variables actually used in tree construction:
##  [1] Age_08_04        Airco            Automatic        Automatic_airco
##  [5] CD_Player        Doors            Fuel_Type        Guarantee_Period
##  [9] HP               KM               Mfr_Guarantee    Powered_Windows
## [13] Quarterly_Tax    Sport_Model      Tow_Bar
##
## Root node error: 8980524175/718 = 12507694
##
## n= 718
##
##              CP nsplit rel error  xerror     xstd
## 1    6.5032e-01      0  1.000000 1.00101 0.085084
## 2    1.2191e-01      1  0.349680 0.38793 0.035319
## 3    2.7713e-02      2  0.227766 0.25878 0.029799
## 4    2.1891e-02      3  0.200053 0.25021 0.029616
## 5    1.5092e-02      4  0.178162 0.23030 0.029583
## 6    1.4778e-02      5  0.163070 0.22383 0.027320
## 7    7.5154e-03      6  0.148293 0.20704 0.025748
## 8    6.2235e-03      7  0.140777 0.18750 0.021969
## 9    5.9050e-03      8  0.134554 0.18416 0.022116
## 10   5.8798e-03      9  0.128649 0.18416 0.022116
## 11   4.8680e-03     10  0.122769 0.18586 0.022137
## 12   4.4652e-03     11  0.117901 0.18285 0.022255
```

```
## 13  4.1797e-03      12  0.113436 0.18136 0.022274
## 14  3.3677e-03      13  0.109256 0.18057 0.022256
## 15  3.3604e-03      14  0.105888 0.17883 0.022149
## 16  2.9563e-03      15  0.102528 0.17736 0.022136
## 17  2.0598e-03      16  0.099572 0.17610 0.021751
## 18  1.8027e-03      17  0.097512 0.17787 0.021525
## 19  1.7979e-03      18  0.095709 0.17708 0.021501
## 20  1.7825e-03      19  0.093911 0.17708 0.021501
## 21  1.7109e-03      20  0.092129 0.17686 0.021446
## 22  1.6849e-03      21  0.090418 0.17701 0.021442
## 23  1.3853e-03      22  0.088733 0.17738 0.021459
## 24  1.3809e-03      23  0.087347 0.17493 0.020526
## 25  1.1124e-03      24  0.085967 0.17272 0.020360
## 26  1.0973e-03      27  0.082629 0.17354 0.020391
## 27  1.0863e-03      28  0.081532 0.17347 0.020337
## 28  1.0859e-03      29  0.080446 0.17347 0.020337
## 29  9.6664e-04      30  0.079360 0.17383 0.020341
## 30  9.3703e-04      32  0.077427 0.16852 0.019771
## 31  9.3500e-04      33  0.076490 0.16934 0.019780
## 32  9.1409e-04      34  0.075555 0.16934 0.019780
## 33  7.9113e-04      35  0.074640 0.17120 0.019869
## 34  7.7264e-04      36  0.073849 0.17426 0.019920
## 35  7.4873e-04      37  0.073077 0.17618 0.019969
## 36  7.2322e-04      38  0.072328 0.17531 0.019945
## 37  7.0628e-04      39  0.071605 0.17535 0.019947
## 38  6.9458e-04      41  0.070192 0.17564 0.019932
## 39  6.7369e-04      42  0.069498 0.17700 0.020105
## 40  6.2289e-04      43  0.068824 0.17617 0.019570
## 41  6.2073e-04      44  0.068201 0.17587 0.019587
## 42  6.1596e-04      45  0.067580 0.17399 0.018556
## 43  6.1117e-04      46  0.066964 0.17399 0.018556
## 44  5.8490e-04      48  0.065742 0.17417 0.018556
## 45  5.7799e-04      50  0.064572 0.17211 0.018452
## 46  5.7156e-04      51  0.063994 0.17211 0.018452
## 47  5.5853e-04      53  0.062851 0.17183 0.018451
## 48  5.3793e-04      54  0.062293 0.17177 0.018469
## 49  5.2685e-04      55  0.061755 0.17140 0.018472
## 50  5.2678e-04      56  0.061228 0.17132 0.018465
## 51  5.1870e-04      57  0.060701 0.17201 0.018464
## 52  4.9275e-04      58  0.060182 0.17229 0.018464
## 53  4.8775e-04      59  0.059690 0.17323 0.018463
## 54  4.8271e-04      60  0.059202 0.17337 0.018462
## 55  4.7438e-04      61  0.058719 0.17323 0.018463
## 56  4.3472e-04      62  0.058245 0.17433 0.018457
## 57  4.2205e-04      63  0.057810 0.17706 0.018475
## 58  4.1563e-04      66  0.056544 0.17622 0.018253
## 59  4.0321e-04      69  0.055297 0.17728 0.018265
## 60  3.8559e-04      71  0.054491 0.17888 0.018306
## 61  3.8243e-04      72  0.054105 0.17929 0.018307
## 62  3.7637e-04      74  0.053340 0.18000 0.018546
## 63  3.6845e-04      75  0.052964 0.18096 0.018554
## 64  3.6749e-04      76  0.052595 0.18157 0.018564
## 65  3.3469e-04      77  0.052228 0.18138 0.018568
## 66  3.1985e-04      78  0.051893 0.18175 0.018554
```

```
## 67  3.1849e-04     81  0.050934 0.18158 0.018550
## 68  3.0151e-04     82  0.050615 0.18304 0.018714
## 69  2.9939e-04     83  0.050314 0.18537 0.018733
## 70  2.9816e-04     85  0.049715 0.18537 0.018733
## 71  2.9511e-04     86  0.049417 0.18542 0.018732
## 72  2.9217e-04     87  0.049121 0.18599 0.018738
## 73  2.9070e-04     88  0.048829 0.18639 0.018736
## 74  2.8959e-04     89  0.048539 0.18636 0.018736
## 75  2.8513e-04     94  0.046990 0.18699 0.018798
## 76  2.8426e-04     97  0.046135 0.18726 0.018787
## 77  2.7594e-04     98  0.045850 0.18746 0.018786
## 78  2.7509e-04    100  0.045298 0.18688 0.018767
## 79  2.7225e-04    101  0.045023 0.18698 0.018760
## 80  2.7070e-04    102  0.044751 0.18694 0.018769
## 81  2.7007e-04    103  0.044480 0.18707 0.018778
## 82  2.6247e-04    104  0.044210 0.18719 0.018779
## 83  2.5581e-04    105  0.043948 0.18723 0.018779
## 84  2.5389e-04    106  0.043692 0.18615 0.018760
## 85  2.4717e-04    107  0.043438 0.18632 0.018760
## 86  2.4529e-04    108  0.043191 0.18619 0.018764
## 87  2.4416e-04    109  0.042946 0.18614 0.018765
## 88  2.4165e-04    110  0.042702 0.18614 0.018765
## 89  2.3758e-04    111  0.042460 0.18618 0.018765
## 90  2.3498e-04    112  0.042222 0.18618 0.018765
## 91  2.3436e-04    113  0.041987 0.18608 0.018765
## 92  2.3198e-04    114  0.041753 0.18620 0.018768
## 93  2.3191e-04    115  0.041521 0.18622 0.018768
## 94  2.2866e-04    116  0.041289 0.18627 0.018770
## 95  2.2862e-04    117  0.041060 0.18649 0.018769
## 96  2.2814e-04    118  0.040832 0.18649 0.018769
## 97  2.2670e-04    119  0.040604 0.18649 0.018769
## 98  2.2288e-04    121  0.040150 0.18631 0.018767
## 99  2.2150e-04    122  0.039927 0.18684 0.018767
## 100 2.1869e-04    123  0.039706 0.18684 0.018767
## 101 2.1750e-04    124  0.039487 0.18684 0.018767
## 102 2.1722e-04    125  0.039270 0.18684 0.018774
## 103 2.1454e-04    126  0.039052 0.18684 0.018774
## 104 2.1158e-04    127  0.038838 0.18669 0.018771
## 105 2.0909e-04    128  0.038626 0.18631 0.018773
## 106 2.0339e-04    130  0.038208 0.18671 0.018772
## 107 2.0312e-04    131  0.038005 0.18690 0.018779
## 108 2.0149e-04    132  0.037802 0.18690 0.018779
## 109 1.9919e-04    133  0.037600 0.18582 0.018777
## 110 1.8767e-04    135  0.037202 0.18529 0.018767
## 111 1.8370e-04    138  0.036639 0.18587 0.018777
## 112 1.8349e-04    139  0.036455 0.18610 0.018776
## 113 1.8344e-04    140  0.036272 0.18603 0.018776
## 114 1.7672e-04    141  0.036088 0.18527 0.018763
## 115 1.7467e-04    142  0.035911 0.18491 0.018764
## 116 1.6787e-04    143  0.035737 0.18510 0.018760
## 117 1.6761e-04    144  0.035569 0.18471 0.018756
## 118 1.5981e-04    145  0.035401 0.18472 0.018755
## 119 1.5979e-04    146  0.035241 0.18494 0.018756
## 120 1.5797e-04    147  0.035082 0.18503 0.018764
```

```
## 121 1.5716e-04    150  0.034608 0.18477 0.018765
## 122 1.5689e-04    151  0.034451 0.18477 0.018765
## 123 1.5682e-04    156  0.033645 0.18477 0.018765
## 124 1.5648e-04    157  0.033488 0.18477 0.018765
## 125 1.4550e-04    158  0.033331 0.18434 0.018767
## 126 1.4268e-04    159  0.033186 0.18598 0.018785
## 127 1.4025e-04    160  0.033043 0.18588 0.018782
## 128 1.3989e-04    161  0.032903 0.18586 0.018782
## 129 1.3871e-04    162  0.032763 0.18587 0.018781
## 130 1.3141e-04    165  0.032321 0.18586 0.018780
## 131 1.3140e-04    166  0.032190 0.18603 0.018778
## 132 1.2980e-04    167  0.032058 0.18616 0.018777
## 133 1.2921e-04    168  0.031928 0.18616 0.018777
## 134 1.2816e-04    169  0.031799 0.18616 0.018777
## 135 1.2217e-04    171  0.031543 0.18697 0.019244
## 136 1.2168e-04    172  0.031421 0.18617 0.019080
## 137 1.2021e-04    175  0.031056 0.18621 0.019082
## 138 1.1433e-04    176  0.030935 0.18585 0.019078
## 139 1.1306e-04    177  0.030821 0.18621 0.019078
## 140 1.1228e-04    178  0.030708 0.18648 0.019079
## 141 1.1225e-04    179  0.030596 0.18648 0.019079
## 142 1.1036e-04    180  0.030484 0.18638 0.019080
## 143 1.0727e-04    181  0.030373 0.18647 0.019082
## 144 1.0701e-04    182  0.030266 0.18641 0.019083
## 145 1.0582e-04    183  0.030159 0.18671 0.019086
## 146 1.0197e-04    185  0.029947 0.18671 0.019086
## 147 1.0184e-04    188  0.029620 0.18676 0.019085
## 148 9.2793e-05    189  0.029518 0.18686 0.019084
## 149 9.0253e-05    190  0.029425 0.18723 0.019100
## 150 9.0104e-05    191  0.029335 0.18751 0.019111
## 151 8.9174e-05    192  0.029245 0.18789 0.019113
## 152 8.9119e-05    193  0.029155 0.18789 0.019113
## 153 8.7209e-05    194  0.029066 0.18803 0.019112
## 154 8.5772e-05    195  0.028979 0.18827 0.019112
## 155 8.5518e-05    197  0.028808 0.18839 0.019110
## 156 8.3515e-05    198  0.028722 0.18825 0.019109
## 157 8.2464e-05    199  0.028639 0.18845 0.019109
## 158 8.0651e-05    200  0.028556 0.18850 0.019108
## 159 7.8540e-05    201  0.028475 0.18864 0.019107
## 160 7.8124e-05    202  0.028397 0.18867 0.019106
## 161 7.6073e-05    203  0.028319 0.18863 0.019115
## 162 7.0642e-05    205  0.028167 0.18922 0.019124
## 163 6.5475e-05    206  0.028096 0.18908 0.019125
## 164 6.5164e-05    207  0.028031 0.18926 0.019120
## 165 6.2775e-05    208  0.027965 0.18926 0.019120
## 166 6.2728e-05    210  0.027840 0.18907 0.019117
## 167 6.1280e-05    211  0.027777 0.18896 0.019116
## 168 6.0882e-05    213  0.027655 0.18900 0.019116
## 169 5.9450e-05    214  0.027594 0.18900 0.019116
## 170 5.7744e-05    215  0.027534 0.18895 0.019117
## 171 5.5268e-05    216  0.027476 0.18898 0.019114
## 172 5.5078e-05    217  0.027421 0.18900 0.019114
## 173 5.3635e-05    219  0.027311 0.18904 0.019113
## 174 5.3080e-05    220  0.027257 0.18871 0.019022
```

```
## 175 5.2927e-05    221  0.027204 0.18866 0.019024
## 176 5.1245e-05    222  0.027151 0.18866 0.019024
## 177 5.0711e-05    223  0.027100 0.18861 0.019029
## 178 4.9894e-05    224  0.027049 0.18861 0.019029
## 179 4.8521e-05    225  0.027000 0.18861 0.019029
## 180 4.7722e-05    226  0.026951 0.18868 0.019030
## 181 4.6777e-05    227  0.026903 0.18869 0.019030
## 182 4.4594e-05    228  0.026856 0.18871 0.019030
## 183 4.4179e-05    229  0.026812 0.18868 0.019030
## 184 4.2575e-05    230  0.026768 0.18862 0.019028
## 185 4.1571e-05    231  0.026725 0.18861 0.019028
## 186 4.1345e-05    232  0.026684 0.18861 0.019028
## 187 4.0677e-05    233  0.026642 0.18862 0.019028
## 188 4.0421e-05    235  0.026561 0.18866 0.019028
## 189 3.8008e-05    236  0.026520 0.18875 0.019028
## 190 3.6537e-05    237  0.026482 0.18884 0.019029
## 191 3.2091e-05    238  0.026446 0.18878 0.019029
## 192 2.7083e-05    240  0.026382 0.18902 0.019027
## 193 2.7059e-05    241  0.026355 0.18897 0.019026
## 194 2.5091e-05    242  0.026328 0.18902 0.019026
## 195 2.4136e-05    243  0.026302 0.18896 0.019026
## 196 1.9806e-05    244  0.026278 0.18890 0.019027
## 197 1.8188e-05    245  0.026259 0.18903 0.019028
## 198 1.8188e-05    246  0.026240 0.18903 0.019028
## 199 1.7157e-05    247  0.026222 0.18903 0.019028
## 200 1.6912e-05    248  0.026205 0.18905 0.019028
## 201 1.5682e-05    249  0.026188 0.18908 0.019028
## 202 1.0000e-05    250  0.026172 0.18922 0.019028
```

```
car.cross_validation$cptable [which.min(car.cross_validation$cptable [, "xerror"]), "nsplit"]
```

```
## [1] 32
```

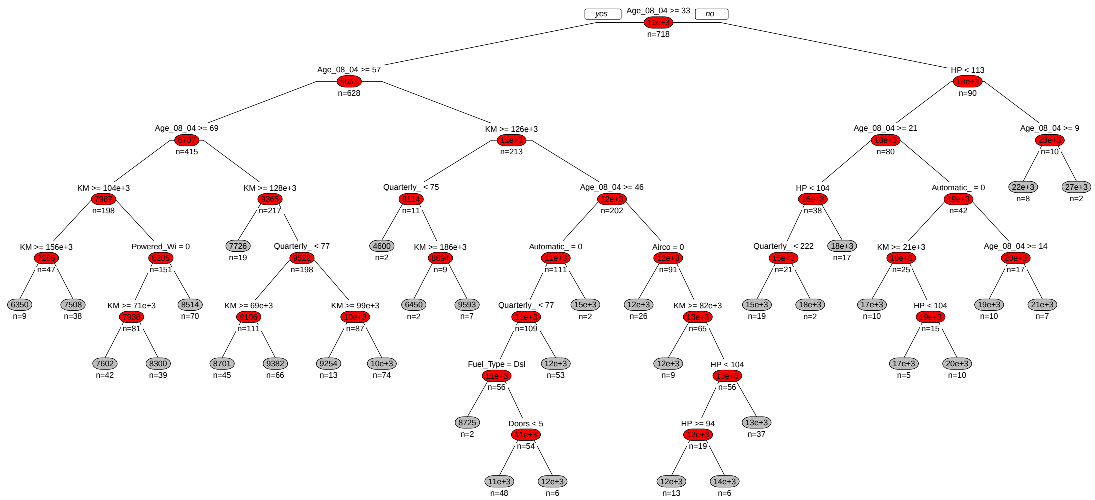
```
# minimum error
car.cross_validation$cptable [which.min(car.cross_validation$cptable [, "xerror"]), "xerror"]
```

```
## [1] 0.1685235
```

We can see that the minimum error tree is with number of splits = 32 and the minimum error = 0.1685235.

```
# best-pruned tree
car.pruned <- prune (car.cross_validation,
                     cp = car.cross_validation$cptable [which.min(car.cross_validation$cptable
                                                         [, "xerror"]), "CP"])
```

```
prp (car.pruned, type = 1, extra = 1, split.font = 1, varlen = -10, under = TRUE,
     box.col = ifelse (car.pruned$frame$var == "<leaf>", "gray", "red"))
```



```
length(car.pruned$frame$var [car.pruned$frame$var == "<leaf>"])
```

```
## [1] 33
```

We can see that the number of leaves in the full grown tree is 33.

```
pruned.pred_train <- predict(car.pruned, data = train.df)
pruned.pred_valid <- predict (car.pruned, newdata = valid.df)
pruned.pred_test <- predict (car.pruned, newdata = test.df)
```

```
pruned.train.df_RMSE <- sqrt(sum((train.df$Price -
                                   as.array(pruned.pred_train))^2)/nrow(as.array(pruned.pred_train)))
pruned.valid.df_RMSE <- sqrt(sum((valid.df$Price -
                                   as.array(pruned.pred_valid))^2)/nrow(as.array(pruned.pred_valid)))
pruned.test.df_RMSE <- sqrt(sum((test.df$Price -
                                  as.array(pruned.pred_test))^2)/nrow(as.array(pruned.pred_test)))
```

```
RMSE_deeper_pruned <- data.frame(Training = c(deeper.train.df_RMSE, pruned.train.df_RMSE),
                                 Validation = c(deeper.valid.df_RMSE, pruned.valid.df_RMSE),
                                 Test = c(deeper.test.df_RMSE, pruned.test.df_RMSE),
                                 row.names = c("Full Grown Tree RMSE", "Best Pruned Tree RMSE"))
RMSE_deeper_pruned
```

```
##                        Training Validation     Test
## Full Grown Tree RMSE   272.7552   1401.600 1418.342
## Best Pruned Tree RMSE  984.0874   1172.967 1221.987
```

We can observe that the validation RMSE and the test RMSE for the best pruned tree is lower than that of the Full grown tree. This is because the full grown tree overfits the data. So if we use a full grown tree instead of the best pruned tree, the predictive performance for the validation data decreases.

Part-b

```
summary(car.df$Price)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    4350    8450    9900   10731   11950   32500
```

```
# categorizes Price into 20 bins of equal counts
car.df$Binned_Price <- cut(x = car.df$Price, breaks = 20, labels = c(1:20))
str(car.df)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    1436 obs. of  17 variables:
##  $ Price           : num  13500 13750 13950 14950 13750 ...
##  $ Age_08_04       : num  23 23 24 26 30 32 27 30 27 23 ...
##  $ KM              : num  46986 72937 41711 48000 38500 ...
##  $ Fuel_Type       : Factor w/ 3 levels "CNG","Diesel",..: 2 2 2 2 2 2 2 2 3 2 ...
##  $ HP              : num  90 90 90 90 90 90 90 90 192 69 ...
##  $ Automatic       : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Doors           : num  3 3 3 3 3 3 3 3 3 3 ...
##  $ Quarterly_Tax   : num  210 210 210 210 210 210 210 210 100 185 ...
##  $ Mfr_Guarantee   : num  0 0 1 1 1 0 0 1 0 0 ...
##  $ Guarantee_Period: num  3 3 3 3 3 3 3 3 3 3 ...
##  $ Airco           : num  0 1 0 0 1 1 1 1 1 1 ...
##  $ Automatic_airco : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ CD_Player       : num  0 1 0 0 0 0 0 1 0 0 ...
##  $ Powered_Windows : num  1 0 0 0 1 1 1 1 1 0 ...
##  $ Sport_Model     : num  0 0 0 0 0 0 1 0 0 0 ...
##  $ Tow_Bar         : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Binned_Price    : Factor w/ 20 levels "1","2","3","4",..: 7 7 7 8 7 7 9 11 13 7 ...
```

```r
# removing Price
car.df <- car.df [, -1]
str(car.df)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    1436 obs. of  16 variables:
##  $ Age_08_04       : num  23 23 24 26 30 32 27 30 27 23 ...
##  $ KM              : num  46986 72937 41711 48000 38500 ...
##  $ Fuel_Type       : Factor w/ 3 levels "CNG","Diesel",..: 2 2 2 2 2 2 2 2 3 2 ...
##  $ HP              : num  90 90 90 90 90 90 90 90 192 69 ...
##  $ Automatic       : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Doors           : num  3 3 3 3 3 3 3 3 3 3 ...
##  $ Quarterly_Tax   : num  210 210 210 210 210 210 210 210 100 185 ...
##  $ Mfr_Guarantee   : num  0 0 1 1 1 0 0 1 0 0 ...
##  $ Guarantee_Period: num  3 3 3 3 3 3 3 3 3 3 ...
##  $ Airco           : num  0 1 0 0 1 1 1 1 1 1 ...
##  $ Automatic_airco : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ CD_Player       : num  0 1 0 0 0 0 0 1 0 0 ...
##  $ Powered_Windows : num  1 0 0 0 1 1 1 1 1 0 ...
##  $ Sport_Model     : num  0 0 0 0 0 0 1 0 0 0 ...
##  $ Tow_Bar         : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Binned_Price    : Factor w/ 20 levels "1","2","3","4",..: 7 7 7 8 7 7 9 11 13 7 ...
```

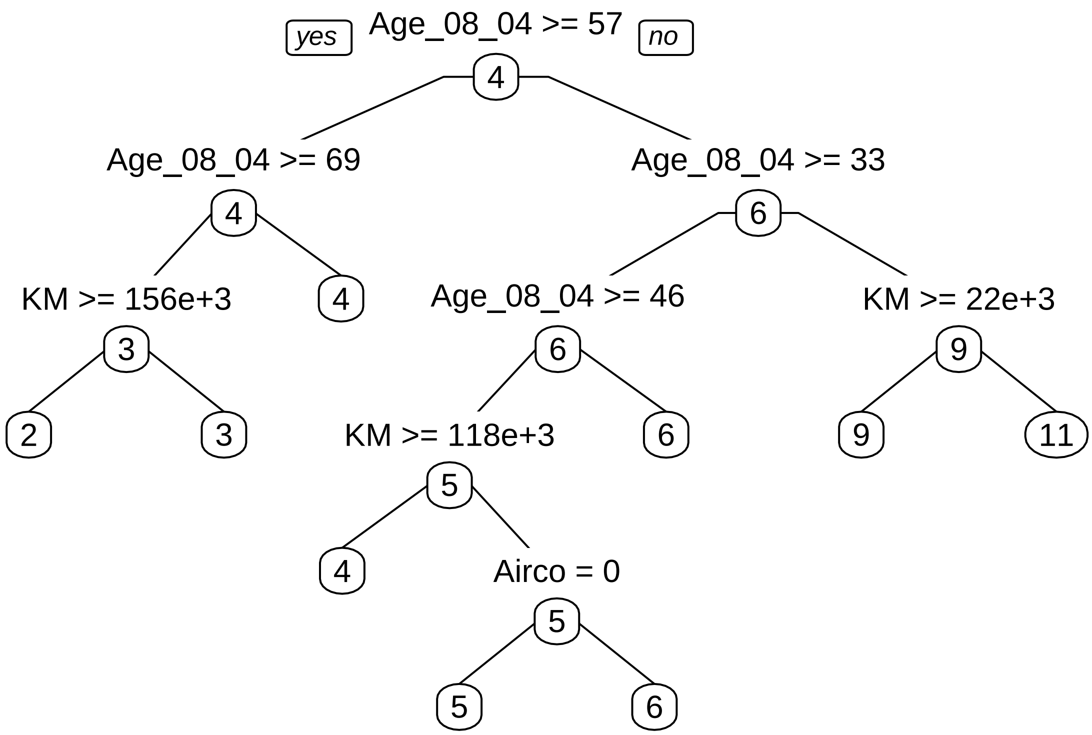
Partitioning the data into training (50%), validation (30%) and test (20%).

```r
set.seed(101)
train1.index <- sample (row.names(car.df), 0.5 * dim (car.df)[1])
valid1.index <- sample (setdiff(row.names(car.df), train1.index), 0.3 * dim (car.df)[1])
test1.index <- setdiff(row.names(car.df), union (train1.index, valid1.index))
train1.df <- car.df [train1.index, ]
valid1.df <- car.df [valid1.index, ]
test1.df <- car.df [test1.index, ]
```

****(i)****

```r
car.classification <- rpart(Binned_Price ~ ., data = train1.df, method = "class")
```

```r
# plot default tree
prp (car.classification, type = 1, split.font = 1, varlen = -10, under = TRUE)
```




```
length_classification <- length(car.classification$frame$var [car.classification$frame$var == "<leaf>"])
length_classification
```

```
## [1] 9
```

```
# lengths
length_difference <- c(length_regression, length_classification)
length_difference
```

```
## [1] 7 9
```

```
# Top predictors
car.tree$variable.importance [1:4]
```

```
##       Age_08_04 Automatic_airco              KM              HP
##      7267193388      1951879777      1723410086      1016849241
```

```
car.classification$variable.importance[1:4]
```

```
## Age_08_04        KM CD_Player     Airco
## 115.24176  49.59624  20.06086  15.60647
```

The differences between the Regression Tree (car.tree) and Classification Tree (car.classification) are: 1. Lengths of the repective trees are 7 and 9. 2. The common top predictors are Age_08_04 and KM. Also, since the choice of a split depends on the ordering of observation values and not on the absolute magnitudes of these values, the splits are sensitive to changes in the data and even a slight change can cause very different splits. This is a reason for the change in the Regression Tree (car.tree) and Classification Tree (car.classification).

****(ii)****

```
new_data <- data.frame(Age_08_04 = 77, KM = 117000, Fuel_Type = "Petrol", HP = 110, Automatic = 0,
                       Doors = 5, Quarterly_Tax = 100, Mfr_Guarantee = 0, Guarantee_Period = 3,
                       Airco = 1, Automatic_airco = 0, CD_Player = 0, Powered_Windows = 0,
                       Sport_Model = 0, Tow_Bar = 1)
```

```
new_data.pred_reg <- predict (car.tree, newdata = new_data)
new_data.pred_reg
```

```
##        1
## 7986.692
```

The predicted Price using Regression tree (car.tree) is $7986.692

```
new_data.pred_class <- predict(car.classification, newdata = new_data, type = "class")
new_data.pred_class
```

```
## 1
## 3
## Levels: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

The predicted Binned Price using Classification tree (car.classification) is 3.

```
real_price <- ((32500 - 4350)/20) * as.numeric(new_data.pred_class) + 4350
real_price
```

```
## [1] 8572.5
```

Since the summary statistics of Price is: min = 4350 and max = 32500. Thus, when Price is divided into 20 equal groups, the predicted Binned Price of class '3' is equal to $8572.5

****(iii)****

```
difference <- abs(new_data.pred_reg - real_price)
difference
```

```
##        1
## 585.8081
```

The prediction from Regression Tree is 7986.692 and from Classification Tree is 8572.5. The difference in the predictions is $585.8081, which is due to the higher accuracy of Regression Tree than that of Classification Tree. This is because in Classification Tree, we binned the outcome into 20 bins while in Regression Tree, we used the actual numbers which is more accurate. Also, the number of leaves in Regression Tree (7) is less than that of in Classification Tree (9). The advantage of using regression tree is when you want to find an accurate price of a car. The advantage of using classification tree is when you want to find the bracket of the price of a car.

Problem-2

```
bank.df <- read_excel("Banks.xlsx")
str(bank.df)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    20 obs. of  5 variables:
##  $ Obs                : num  1 2 3 4 5 6 7 8 9 10 ...
##  $ Financial Condition: num  1 1 1 1 1 1 1 1 1 1 ...
##  $ TotCap/Assets      : num  8.1 6.6 5.8 12.3 4.5 9.1 1.1 8.9 0.7 9.8 ...
##  $ TotExp/Assets      : num  0.13 0.1 0.11 0.09 0.11 0.14 0.12 0.12 0.16 0.12 ...
##  $ TotLns&Lses/Assets : num  0.64 1.04 0.66 0.8 0.69 0.74 0.63 0.75 0.56 0.65 ...
```

Running the logistic regression model with predictors X1 as TotLns&Lses/Assets, X2 as TotExp/Assets and Y as Financial Condition. Changing the names of these variable to X1, X2 and Y respectively.

```
colnames (bank.df) [c(2, 4, 5)] <- c("Y", "X2", "X1")
str(bank.df)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    20 obs. of  5 variables:
```

```
##  $ Obs          : num  1 2 3 4 5 6 7 8 9 10 ...
##  $ Y            : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ TotCap/Assets: num  8.1 6.6 5.8 12.3 4.5 9.1 1.1 8.9 0.7 9.8 ...
##  $ X2           : num  0.13 0.1 0.11 0.09 0.11 0.14 0.12 0.12 0.16 0.12 ...
##  $ X1           : num  0.64 1.04 0.66 0.8 0.69 0.74 0.63 0.75 0.56 0.65 ...
```

```r
bank.logit <- glm(Y ~ X1 + X2, data = bank.df, family = "binomial")
summary (bank.logit)
```

```
##
## Call:
## glm(formula = Y ~ X1 + X2, family = "binomial", data = bank.df)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.64035  -0.35514   0.02079   0.53234   1.03373
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -14.188      6.122  -2.317   0.0205 *
## X1             9.173      6.864   1.336   0.1814
## X2            79.964     39.263   2.037   0.0417 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 27.726  on 19  degrees of freedom
## Residual deviance: 12.831  on 17  degrees of freedom
## AIC: 18.831
##
## Number of Fisher Scoring iterations: 6
```

Part-a

(i)

Logit Format: Logit(Y = 1) = -14.188 + 9.173 * X1 + 79.964 * X2

(ii)

```r
bank.logit$Odds <- exp(bank.logit$coefficients)
bank.logit$Odds
```

```
##  (Intercept)           X1           X2
## 6.893258e-07 9.635549e+03 5.344393e+34
```

Odds Format: Odds(Y = 1) = e ^ (-14.188 + 9.173 * X1 + 79.964 * X2) = (6.8932 * 10^(-7)) * (9.635 * 10^(3)) ^ X1 * (5.344 * 10^(34)) ^ X2

iii.

Probability Format: P (Y = 1) = 1 / (1 + e ^ (14.188 - 9.173 * X1 - 79.964 * X2))

Part-b

```r
new_data <- data.frame (X1 = 0.6, X2 = 0.11)
new_data
```

```
##    X1   X2
## 1 0.6 0.11
```

```
new_data.pred <- predict(bank.logit, newdata = new_data)
new_data.pred
```

```
##         1
## 0.1124105
```

Thus the logit (Y = 1) = 0.1124105

```
odds <- exp(new_data.pred)
odds
```

```
##        1
## 1.118972
```

The Odds (Y = 1) = 1.118972

```
prob <- odds/ (1 + odds)
prob
```

```
##         1
## 0.5280731
```

The P (Y = 1) = 0.5280731 Since the P (Y = 1) is greater than the cutoff value of 0.5, the new data is recorded as Y = 1, i.e., the new bank is Financially weak.

Part-c The cutoff value of 0.5 is in conjunction with P (Y = 1) for classifying a bank being Financially weak. Thus, to make a classification based on odds and logit, the threshold should be 1 and 0 respectively.

Part-d X1 as TotLns&Lses/Assets, X2 as TotExp/Assets and Y as Financial Condition.

Odds Format: Odds(Y = 1) = e ^ (-14.188 + 9.173 * X1 + 79.964 * X2) = (6.8932 * 10^(-7)) * (9.635 * 10^(3)) ^ X1 * (5.344 * 10^(34)) ^ X2

As we increase a unit of X1 keeping X2 constant, the odds of belonging to Y = 1 will increase with a factor of e^(9.173) = 9.635 * 10^(3). Similarly, if we increase a unit of X2 keeping X1 constant, the odds of belonging to Y = 1 will increase with a factor of e^(79.964) = 5.344 * 10^(34). This is because the coefficients for both X1 and X2 are positive.

Part-e

If a bank in poor financial condition is misclassified as financially strong, the cutoff value for classification (at 0.5) should be decreased to avoid the misclassification.

Problem-3

```
system.df <- read_excel("System Administrators.xlsx")
str(system.df)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    75 obs. of  3 variables:
##  $ Experience    : num  10.9 9.9 10.4 13.7 9.4 12.4 7.9 8.9 10.2 11.4 ...
##  $ Training      : num  4 4 6 6 8 4 6 4 6 4 ...
##  $ Completed task: chr  "Yes" "Yes" "Yes" "Yes" ...
```

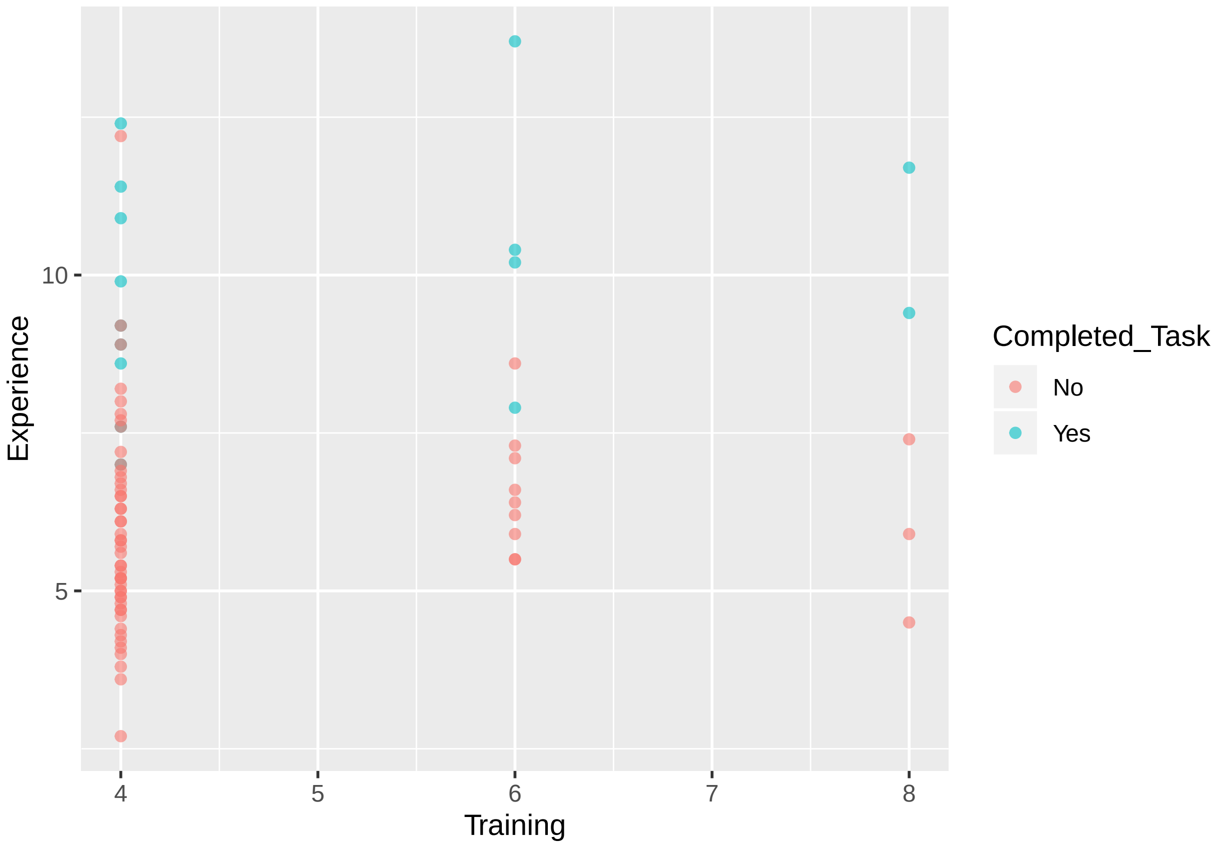
```
colnames (system.df) [3] <- "Completed_Task"
str(system.df)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    75 obs. of  3 variables:
##  $ Experience    : num  10.9 9.9 10.4 13.7 9.4 12.4 7.9 8.9 10.2 11.4 ...
```

```
##  $ Training      : num  4 4 6 6 8 4 6 4 6 4 ...
##  $ Completed_Task: chr  "Yes" "Yes" "Yes" "Yes" ...
```

Part-a

```r
# scatterplot
ggplot(system.df, aes(y = Experience, x = Training, colour = Completed_Task)) +
  geom_point(alpha = 0.6)
```



We can observe from the scatterplot that Experience appears to be potentially useful predictor for classifying task completion.

Part-b

```r
system.df$Completed_Task <- factor(system.df$Completed_Task)
levels(system.df$Completed_Task) <- c(0, 1)
summary (system.df$Completed_Task)
```

```
##  0  1
## 60 15
```

```r
system.df_logit <- glm (Completed_Task ~ ., data = system.df, family = "binomial")
summary (system.df_logit)
```

```
##
## Call:
## glm(formula = Completed_Task ~ ., family = "binomial", data = system.df)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
```

```
## -2.65306  -0.34959  -0.17479  -0.08196   2.21813
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -10.9813     2.8919  -3.797 0.000146 ***
## Experience    1.1269     0.2909   3.874 0.000107 ***
## Training      0.1805     0.3386   0.533 0.593970
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 75.060  on 74  degrees of freedom
## Residual deviance: 35.713  on 72  degrees of freedom
## AIC: 41.713
##
## Number of Fisher Scoring iterations: 6
```

```r
# predicting the P (Completed_Task = 1)
system.df_logit_pred <- predict (system.df_logit, data = system.df, type = "response")
str(system.df_logit_pred)
```

```
##  Named num [1:75] 0.883 0.71 0.861 0.996 0.742 ...
##  - attr(*, "names")= chr [1:75] "1" "2" "3" "4" ...
```

```r
confusionMatrix(as.factor (ifelse(system.df_logit_pred > 0.5, 1, 0)), system.df$Completed_Task)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 58  5
##          1  2 10
##
##                Accuracy : 0.9067
##                  95% CI : (0.8171, 0.9616)
##     No Information Rate : 0.8
##     P-Value [Acc > NIR] : 0.01041
##
##                   Kappa : 0.6847
##
##  Mcnemar's Test P-Value : 0.44969
##
##             Sensitivity : 0.9667
##             Specificity : 0.6667
##          Pos Pred Value : 0.9206
##          Neg Pred Value : 0.8333
##              Prevalence : 0.8000
##          Detection Rate : 0.7733
##    Detection Prevalence : 0.8400
##       Balanced Accuracy : 0.8167
##
##        'Positive' Class : 0
##
```

We can see that 7 records are misclassified. Among those who complete the task, what is the percentage of programmers who are incorrectly classified as failing to complete the task = 5 / (5 + 10) = 33.33 %

Part-c

In order to decrease the percantage in part-b, the cutoff value should be decreased.

```r
# cutoff value decreased from 0.5 to 0.4
confusionMatrix(as.factor (ifelse(system.df_logit_pred > 0.4, 1, 0)), system.df$Completed_Task)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 56  4
##          1  4 11
##
##                Accuracy : 0.8933
##                  95% CI : (0.8006, 0.9528)
##     No Information Rate : 0.8
##     P-Value [Acc > NIR] : 0.0243
##
##                   Kappa : 0.6667
##
##  Mcnemar's Test P-Value : 1.0000
##
##             Sensitivity : 0.9333
##             Specificity : 0.7333
##          Pos Pred Value : 0.9333
##          Neg Pred Value : 0.7333
##              Prevalence : 0.8000
##          Detection Rate : 0.7467
##    Detection Prevalence : 0.8000
##       Balanced Accuracy : 0.8333
##
##        'Positive' Class : 0
##
```

We can see that the percentage in part-b has decreased to = 4 / (4 + 11) = 26.67 %

Part-d P (Y = 1) > 0.5 -> Implies, Odds (Y = 1) > 1 -> Further implies, logit > 0. Thus, -10.9813 + 1.1269 * Experience + 0.1805 * Training > 0. Therefore, Experience > (10.9813 - 0.1805 * Training) / 1.1269. For Training = 4 years, Experience > 9.104 years