4305ass2

Executive Summary

Data Exploration

R Markdown

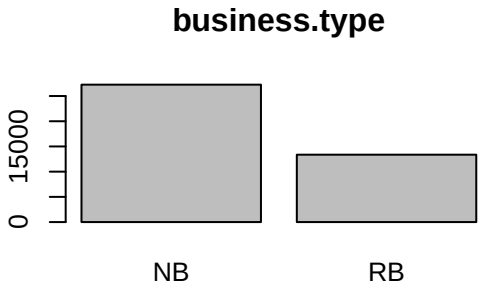
Since there seems to be two problems -> modelling frequency and modelling severity, I will try to replicate that to some extent

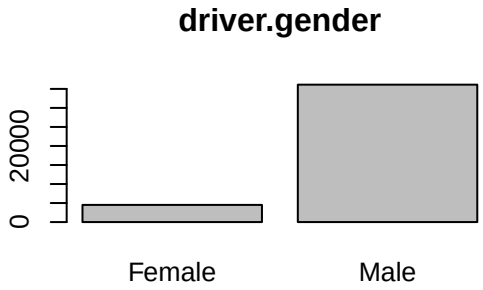
Should also do some data exploration... 1. because its needed for the individual report 2. Because it could reveal some helpful transformations

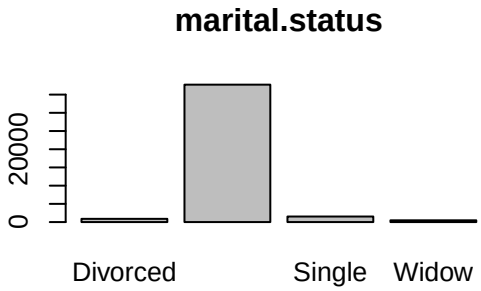
MOSTLY LOOKING AT CV ERROR RATHER THAN OOB ERROR - should be stated Check discussion question week 9 for some more random forest examples Week 9 lectorial has some helpful exploratory analysis examples

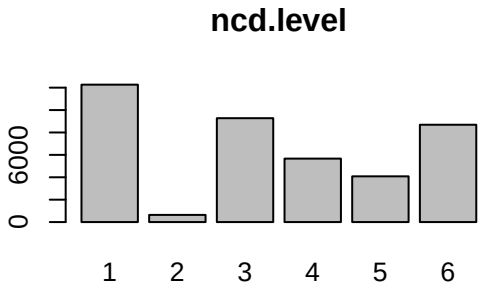
Data Exploration

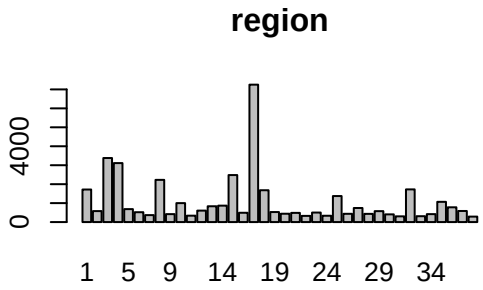
```
## tibble [40,621 x 23] (S3: tbl_df/tbl/data.frame)
##  $ year          : num [1:40621] 2016 2015 2015 2015 2015 ...
##  $ exposure      : num [1:40621] 0.917 0.167 0.917 0.167 0.917 ...
##  $ business.type : Factor w/ 2 levels "NB","RB": 1 1 1 1 1 1 1 1 1 1 ...
##  $ driver.age    : num [1:40621] 47 47 34 34 47 47 34 34 29 29 ...
##  $ driver.gender : Factor w/ 2 levels "Female","Male": 2 2 2 2 2 2 1 1 2 2 ...
##  $ marital.status: Factor w/ 4 levels "Divorced","Married",..: 2 2 2 2 1 1 2 2 2 2 ...
##  $ yrs.licensed  : num [1:40621] 3 3 1 1 4 4 2 2 1 1 ...
##  $ ncd.level     : Factor w/ 6 levels "1","2","3","4",..: 4 4 1 1 1 1 3 3 1 1 ...
##  $ region        : Factor w/ 38 levels "1","2","3","4",..: 15 15 28 28 3 3 28 28 28 28 ...
##  $ body.code     : Factor w/ 8 levels "A","B","C","D",..: 1 1 5 5 4 4 7 7 5 5 ...
##  $ vehicle.age   : num [1:40621] 3 3 5 5 4 4 6 6 3 3 ...
##  $ vehicle.value : num [1:40621] 22.6 22.6 19.2 19.2 18 18 30.9 30.9 20.4 20.4 ...
##  $ no.seats      : num [1:40621] 5 5 6 6 2 2 6 6 5 5 ...
##  $ cubic.cent    : num [1:40621] 1753 1753 2198 2198 1461 ...
##  $ horse.power   : num [1:40621] 90 90 130 130 65 65 105 105 75 75 ...
##  $ weight        : num [1:40621] 1485 1485 1753 1753 1120 ...
##  $ length        : num [1:40621] 4.3 4.3 4.86 4.86 4.04 ...
##  $ width         : num [1:40621] 1.79 1.79 1.97 1.97 1.67 ...
##  $ height        : num [1:40621] 1.81 1.81 2.07 2.07 1.82 ...
##  $ fuel.type     : Factor w/ 3 levels "Diesel","Gasoline",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ prior.claims  : num [1:40621] 0 0 0 0 2 2 1 1 0 0 ...
##  $ claim.count   : Factor w/ 6 levels "0","1","2","3",..: 1 1 2 1 1 2 1 1 2 1 ...
##  $ claim.incurred: num [1:40621] 0 0 606 0 0 ...
```

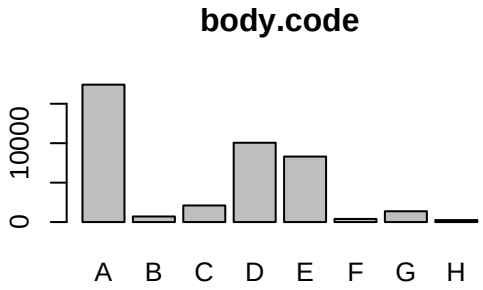

business.type
0
15000
NB
RB

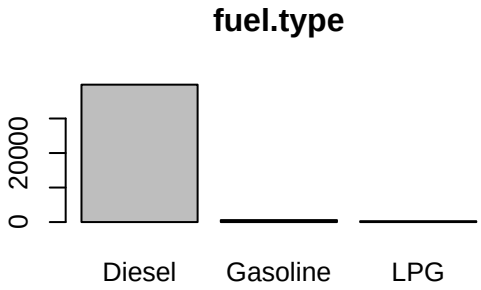

driver.gender
0
20000
Female
Male

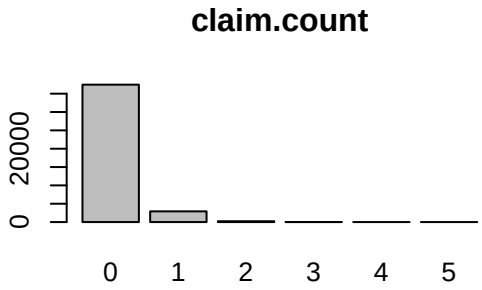

marital.status
0
20000
Divorced
Single
Widow

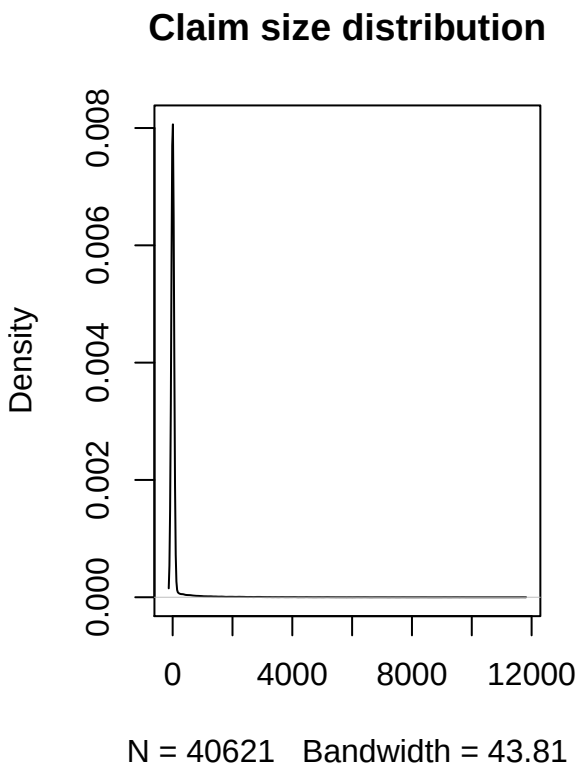

ncd.level
0
6000
1
2
3
4
5
6

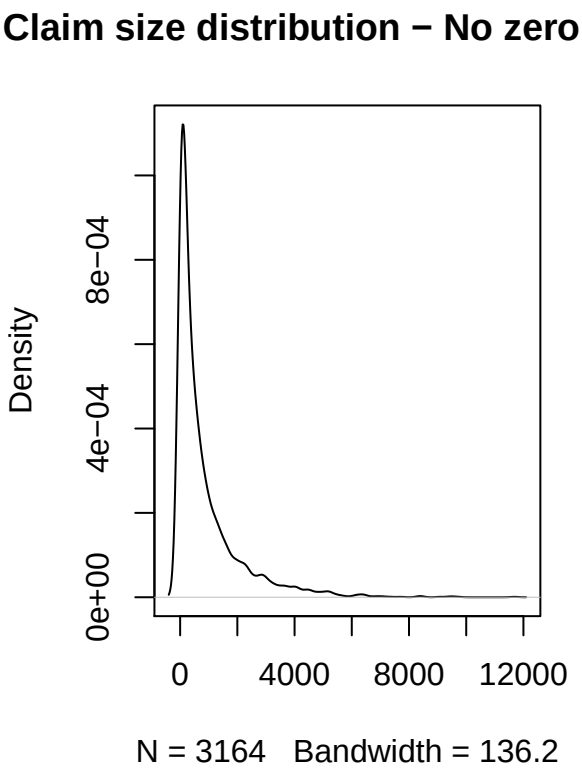

region
0
4000
1
5
9
14
19
24
29
34

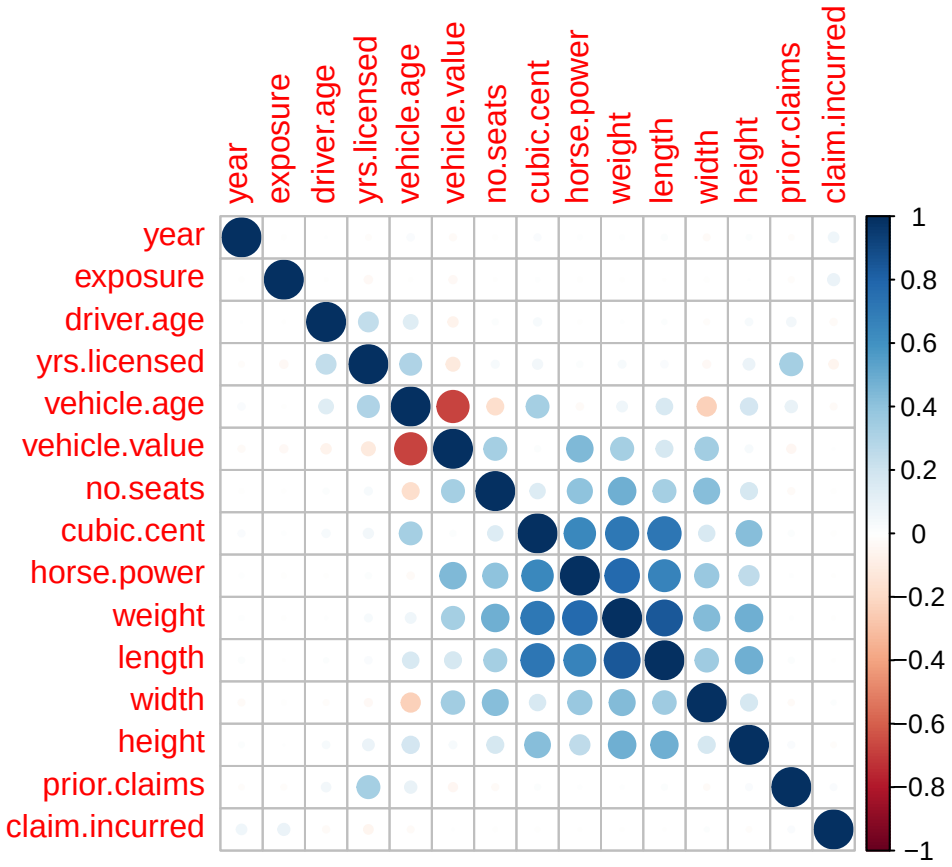

body.code
0
10000
A
B
C
D
E
F
G
H


fuel.type
0
20000
Diesel
Gasoline
LPG

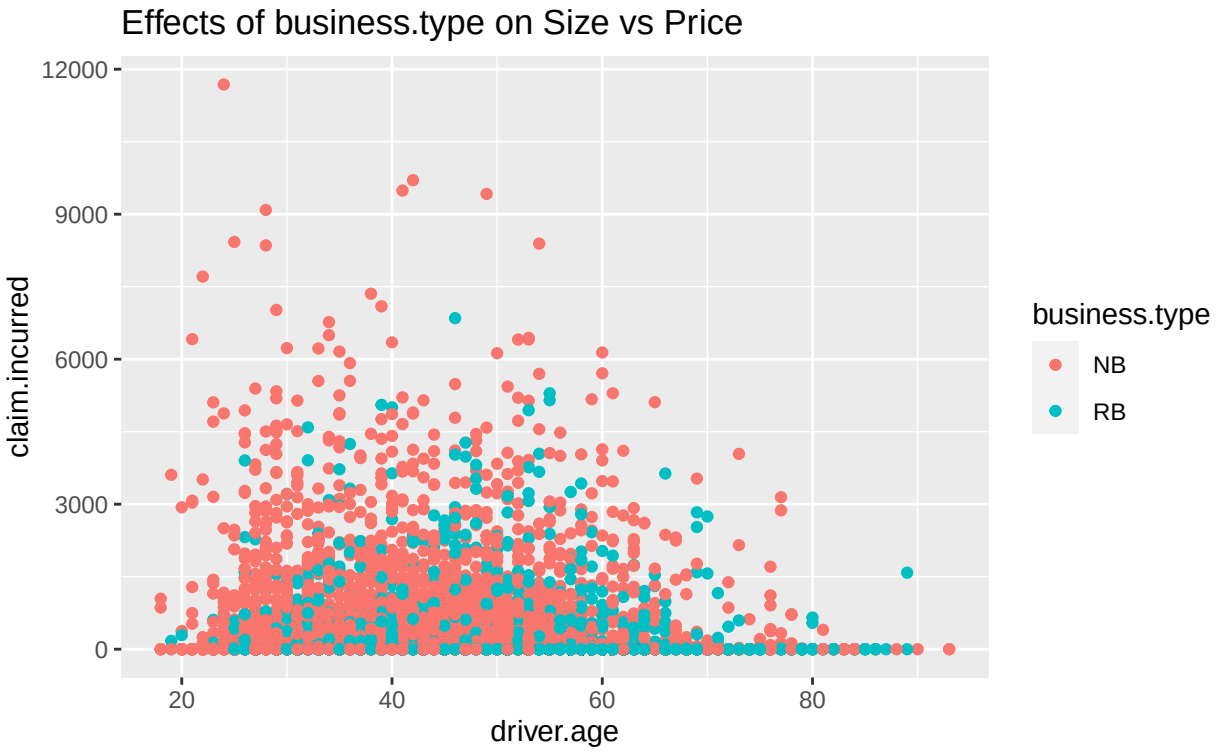

claim.count
0
20000
0
1
2
3
4
5

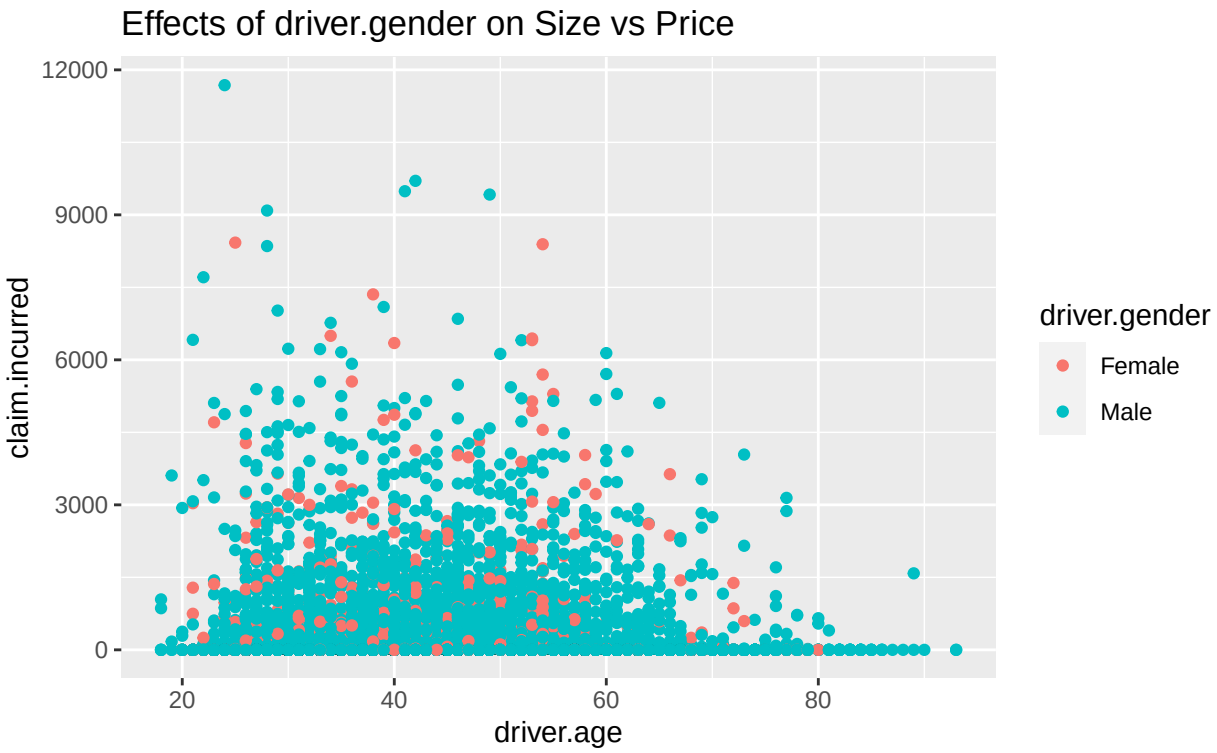

Claim size distribution
Density
0.000
0.002
0.004
0.006
0.008
0
4000
8000
12000
N = 40621 Bandwidth = 43.81

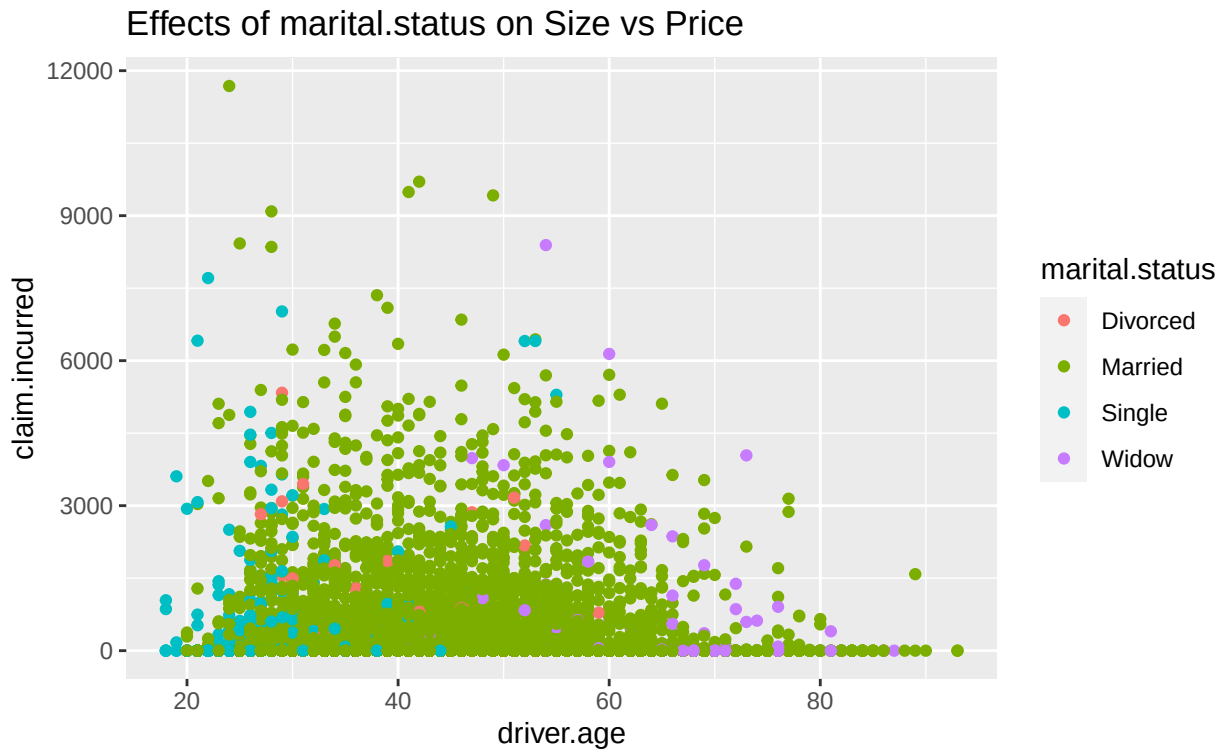

Claim size distribution − No zero te
Density
0e+00
4e−04
8e−04
0
4000
8000
12000
N = 3164 Bandwidth = 136.2

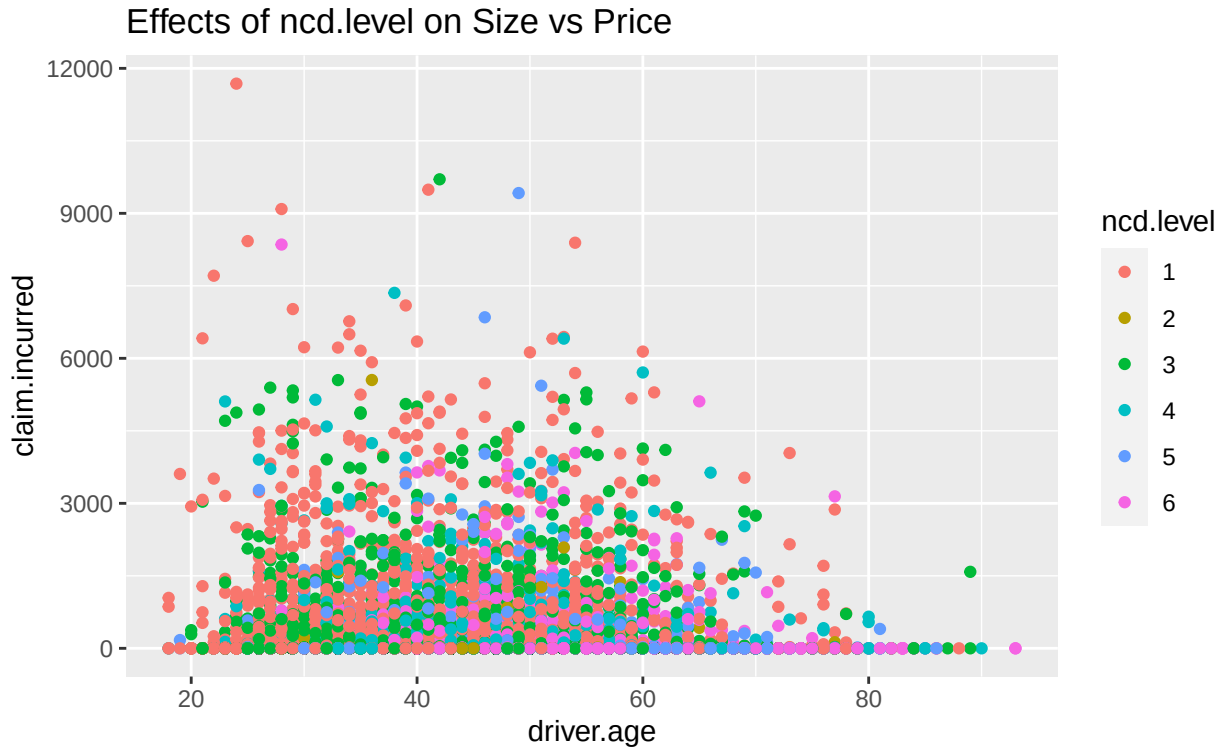

year
exposure
driver.age
yrs.licensed
vehicle.age
vehicle.value
no.seats
cubic.cent
horse.power
weight
length
width
height
prior.claims
claim.incurred
1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1

Effects of house characteristics on size and price

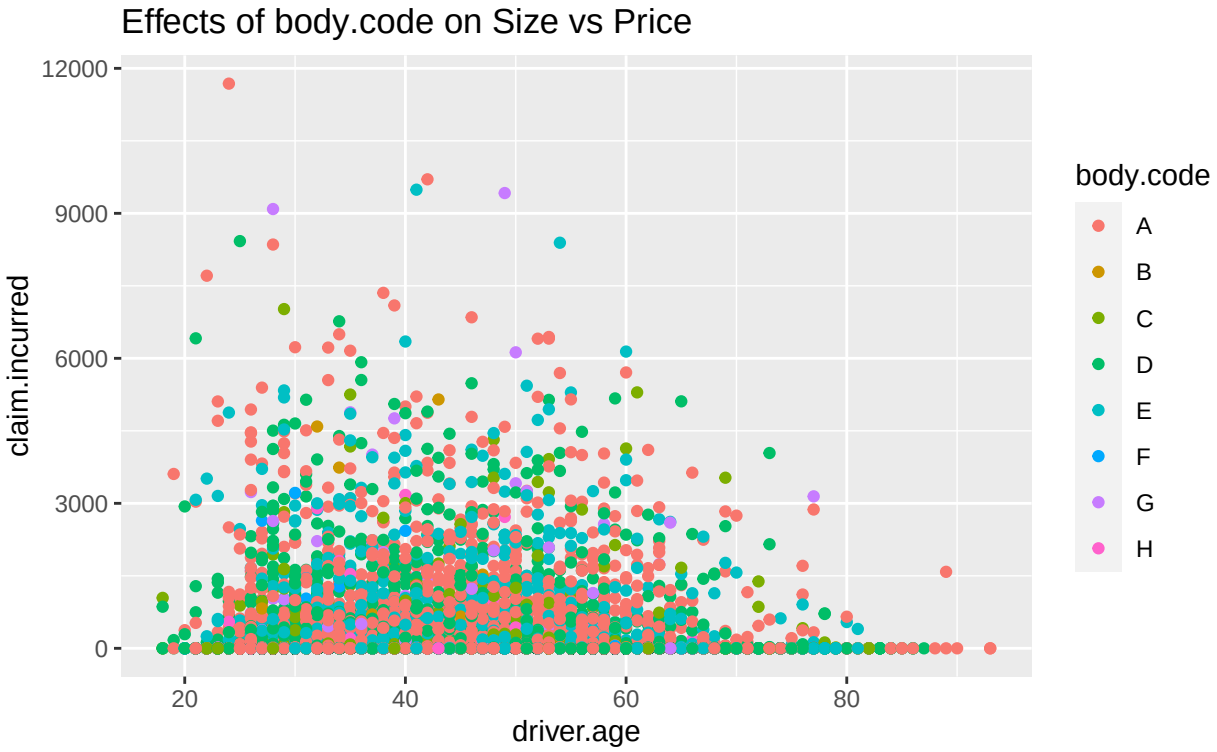


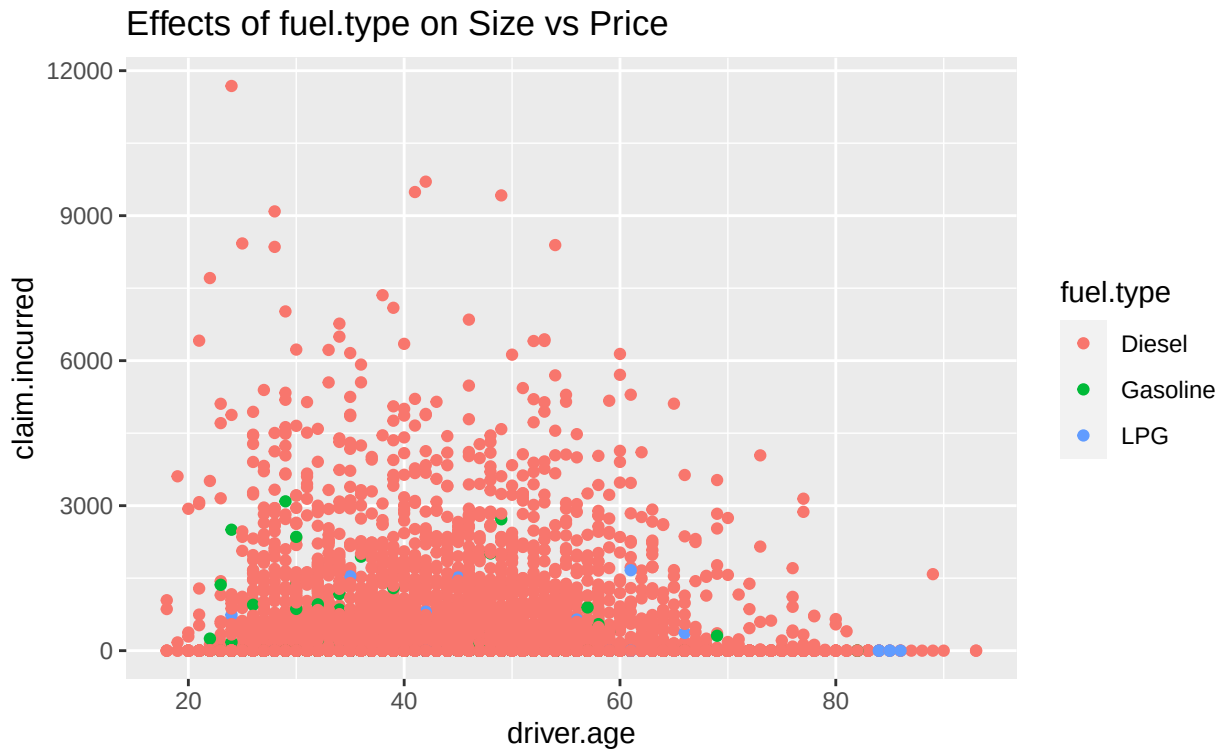


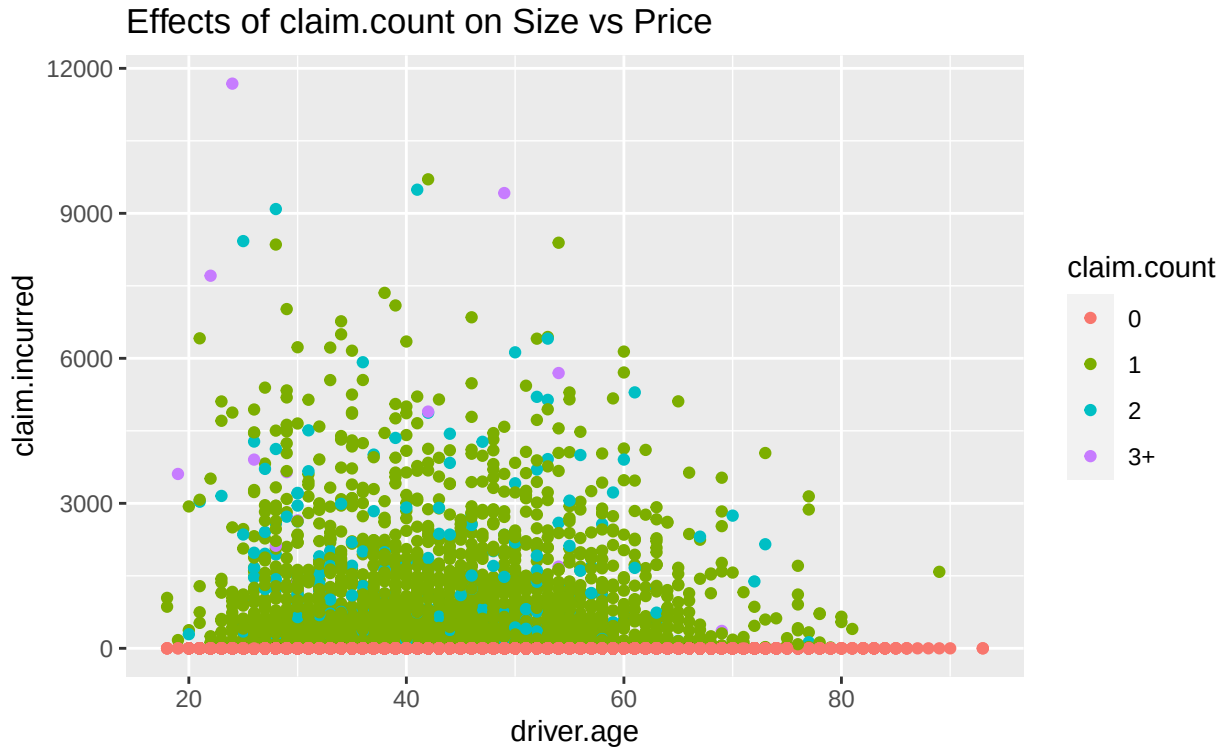

Effects of marital.status on Size vs Price
claim.incurred
driver.age
12000
9000
6000
3000
0
20
40
60
80
marital.status
Divorced
Married
Single
Widow

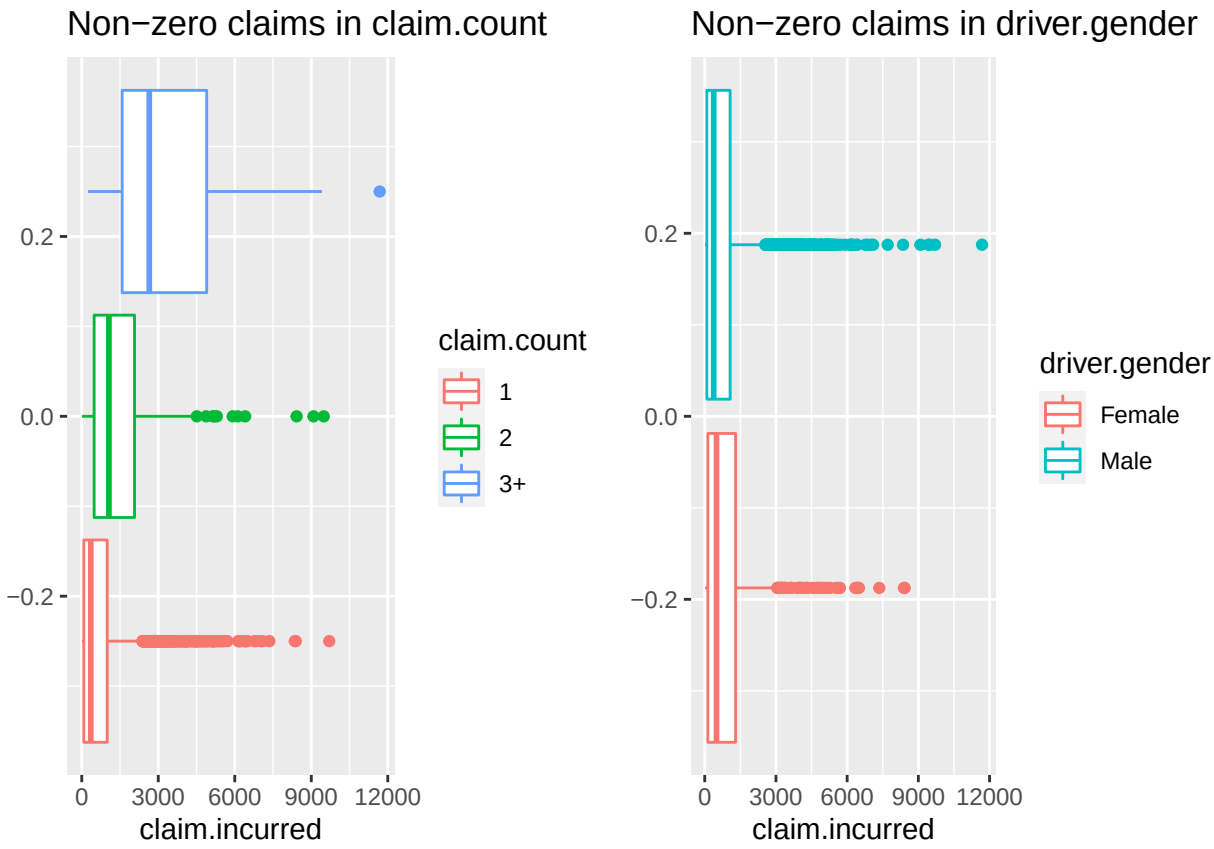

Effects of ncd.level on Size vs Price
claim.incurred
driver.age
12000
9000
6000
3000
0
20
40
60
80
ncd.level
1
2
3
4
5
6

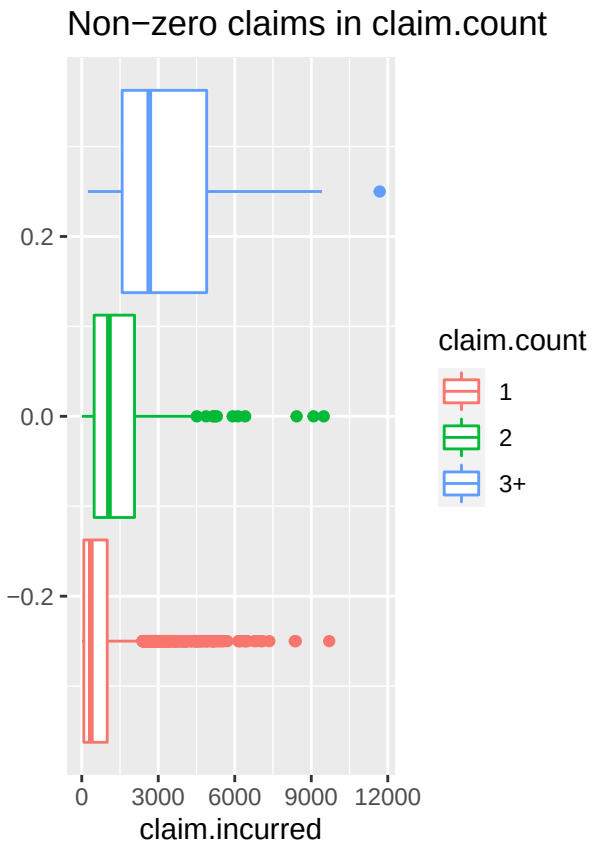

Effects of region on Size vs Price
claim.incurred
driver.age
12000
9000
6000
3000
0
20
40
60
80
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37

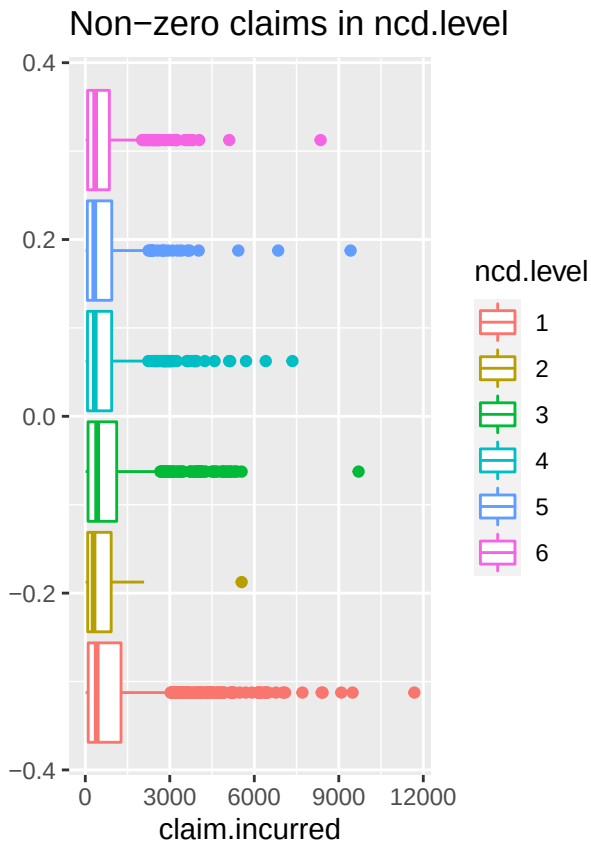

Effects of body.code on Size vs Price
claim.incurred
driver.age
12000
9000
6000
3000
0
20
40
60
80
body.code
A
B
C
D
E
F
G
H

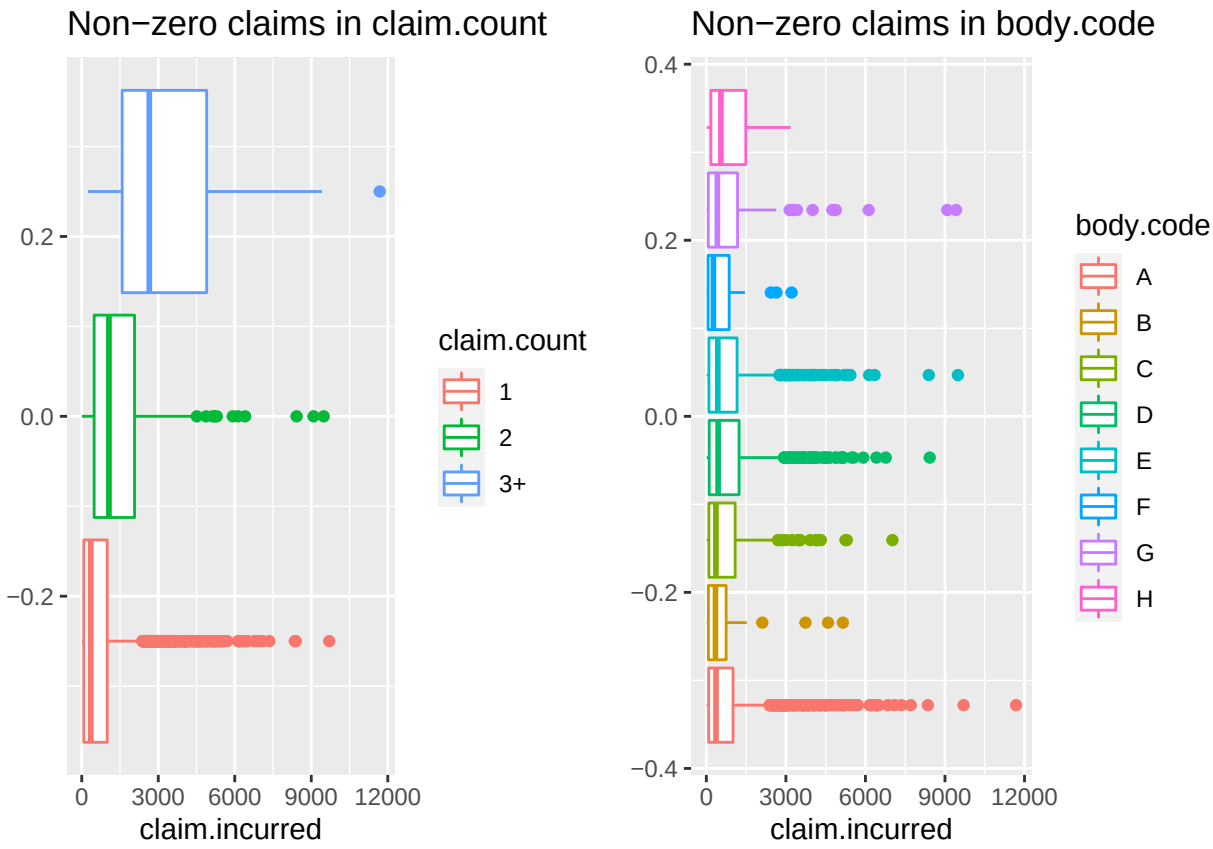

Effects of fuel.type on Size vs Price
claim.incurred
12000
9000
6000
3000
0
20
40
60
80
driver.age
fuel.type
Diesel
Gasoline
LPG

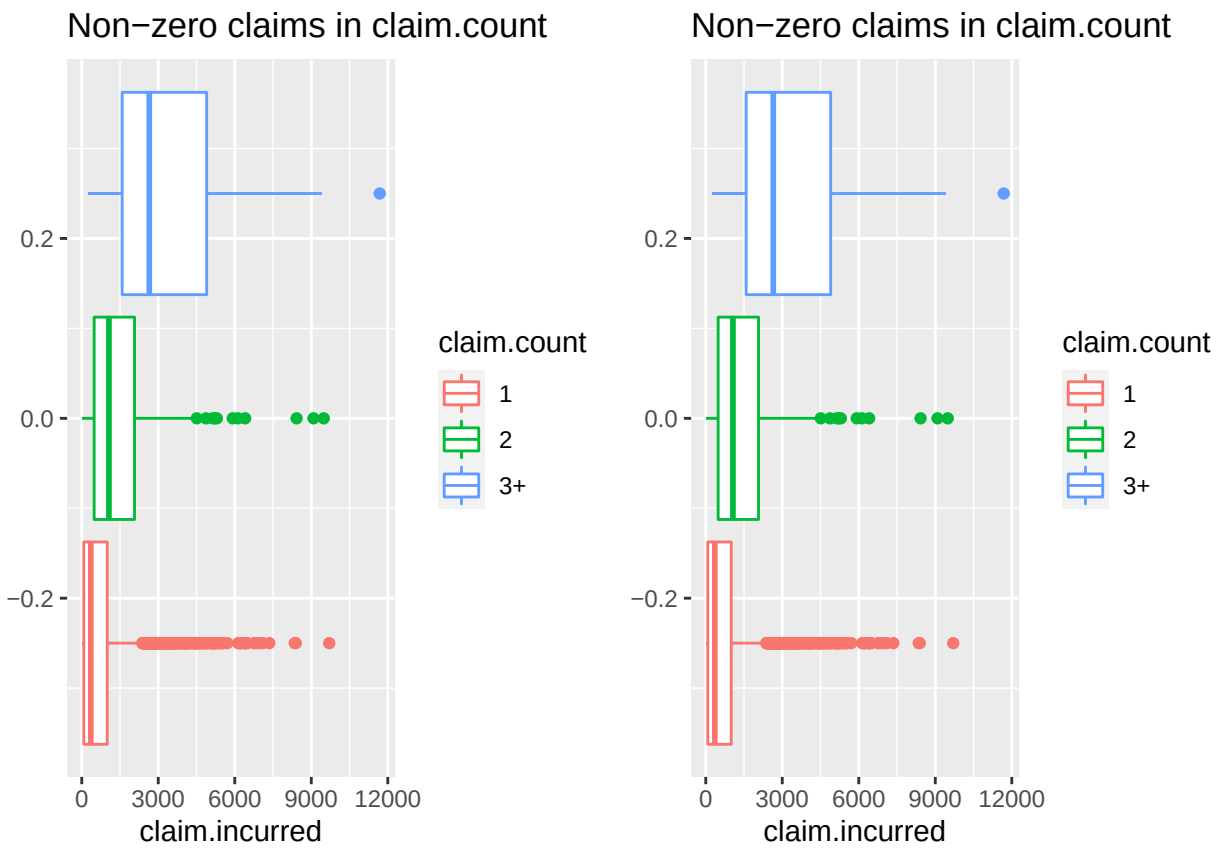

Effects of claim.count on Size vs Price
claim.incurred
12000
9000
6000
3000
0
20
40
60
80
driver.age
claim.count
0
1
2
3+


Non−zero claims in claim.count
0.2
0.0
−0.2
0
3000
6000
9000
12000
claim.incurred
claim.count
1
2
3+
Non−zero claims in driver.gender
0.2
0.0
−0.2
0
3000
6000
9000
12000
claim.incurred
driver.gender
Female
Male


Non−zero claims in claim.count
0.2
0.0
−0.2
0
3000
6000
9000
12000
claim.incurred
claim.count
1
2
3+


Non−zero claims in ncd.level
0.4
0.2
0.0
−0.2
−0.4
0
3000
6000
9000
12000
claim.incurred
ncd.level
1
2
3
4
5
6


Non−zero claims in claim.count
0.2
0.0
−0.2
0
3000
6000
9000
12000
claim.incurred
claim.count
1
2
3+
Non−zero claims in body.code
0.4
0.2
0.0
−0.2
−0.4
0
3000
6000
9000
12000
claim.incurred
body.code
A
B
C
D
E
F
G
H




```
## Warning: The 'i' argument of ''['()' can't be a matrix as of tibble 3.0.0.
## Convert to a vector.
## This warning is displayed once every 8 hours.
## Call 'lifecycle::last_warnings()' to see where this warning was generated.
```

Loading all models

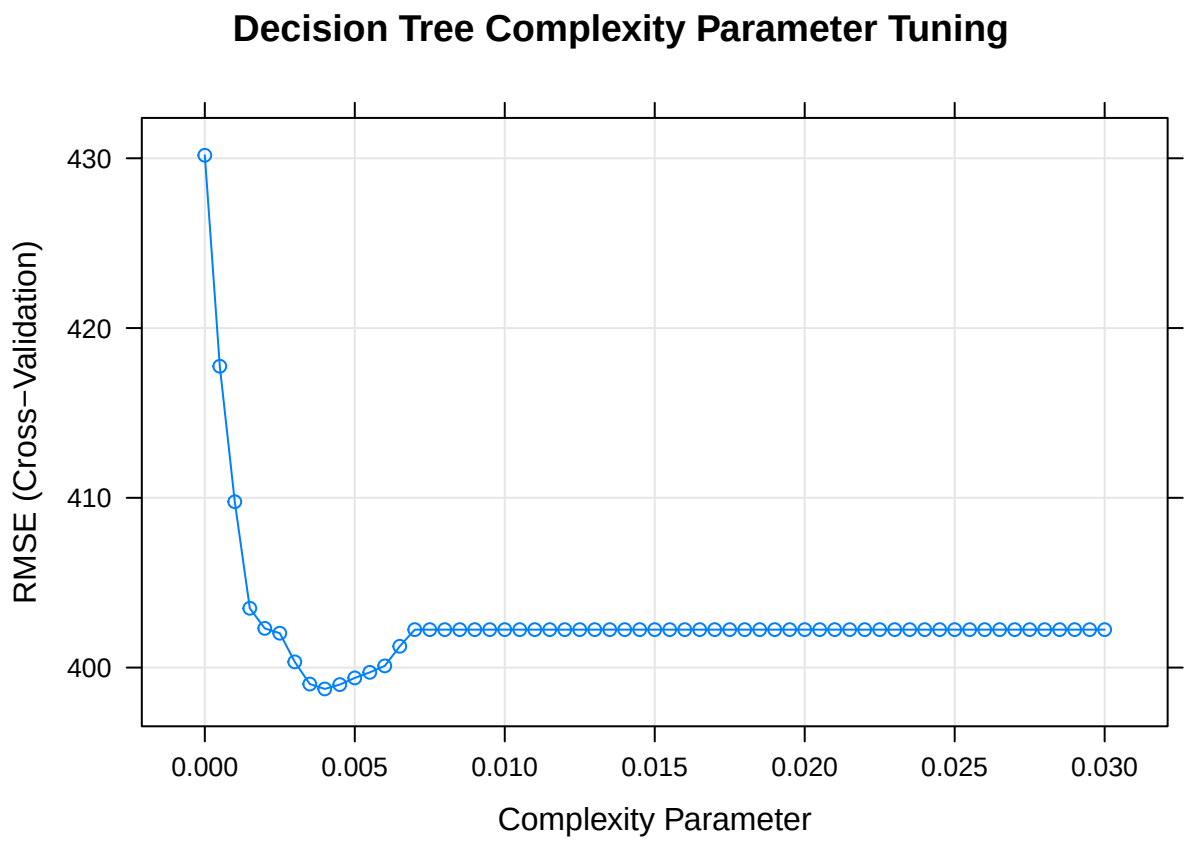
Methodology

For each model, the hyperparamters were tuned through the Caret package and a 5 fold cross validation. 5 fold cross validation was deemed adequate enough to produce results that were significant whilst also having reasonably short enough run times. Even without repeated cross validation or higher folds, each random forest model could take an hour to run.

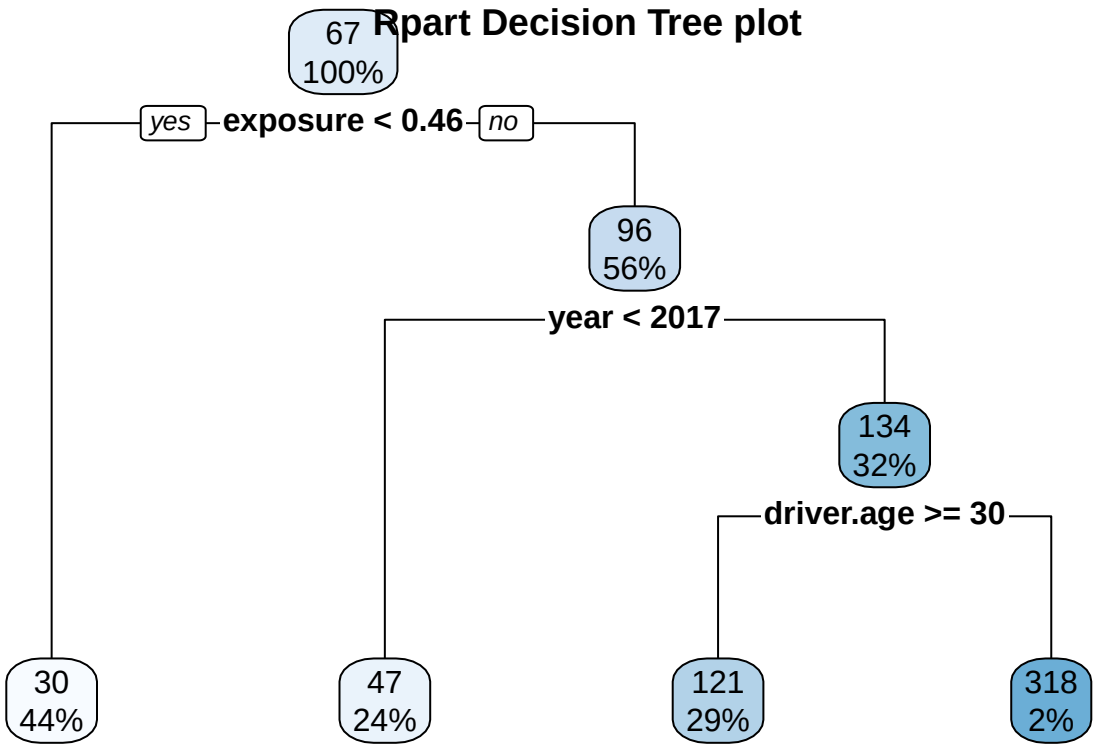
Decision Tree

The decision tree was the worst performing model, although it was only marginally worse than either random forest or bagging. The complexity parameter (CP) was tuned through a grid search and the best value was 0.004.

```r
plot(rpart1, main = "Decision Tree Complexity Parameter Tuning")
```




```r
rpart.plot(rpart1$finalModel) # requires rpart.plot package
title("Rpart Decision Tree plot")
```




```
bestcp <- rpart1$bestTune
```

Interestingly, any CP value greater than 0.007 produced a zero-split tree, which only had slightly worse predictive power than the standard tree. Therefore, this suggests that the decision tree is not a particularly strong model.

Decision Tree: Advantages

The major advantage of the decision tree is its interpretability, it is clear that high exposure, more recent vehicles and younger drivers can all be indicative of higher claims. This is information is helpful for business decisions and to generalise individuals who may be seeking coverage.

Decision Tree: Disadvantages

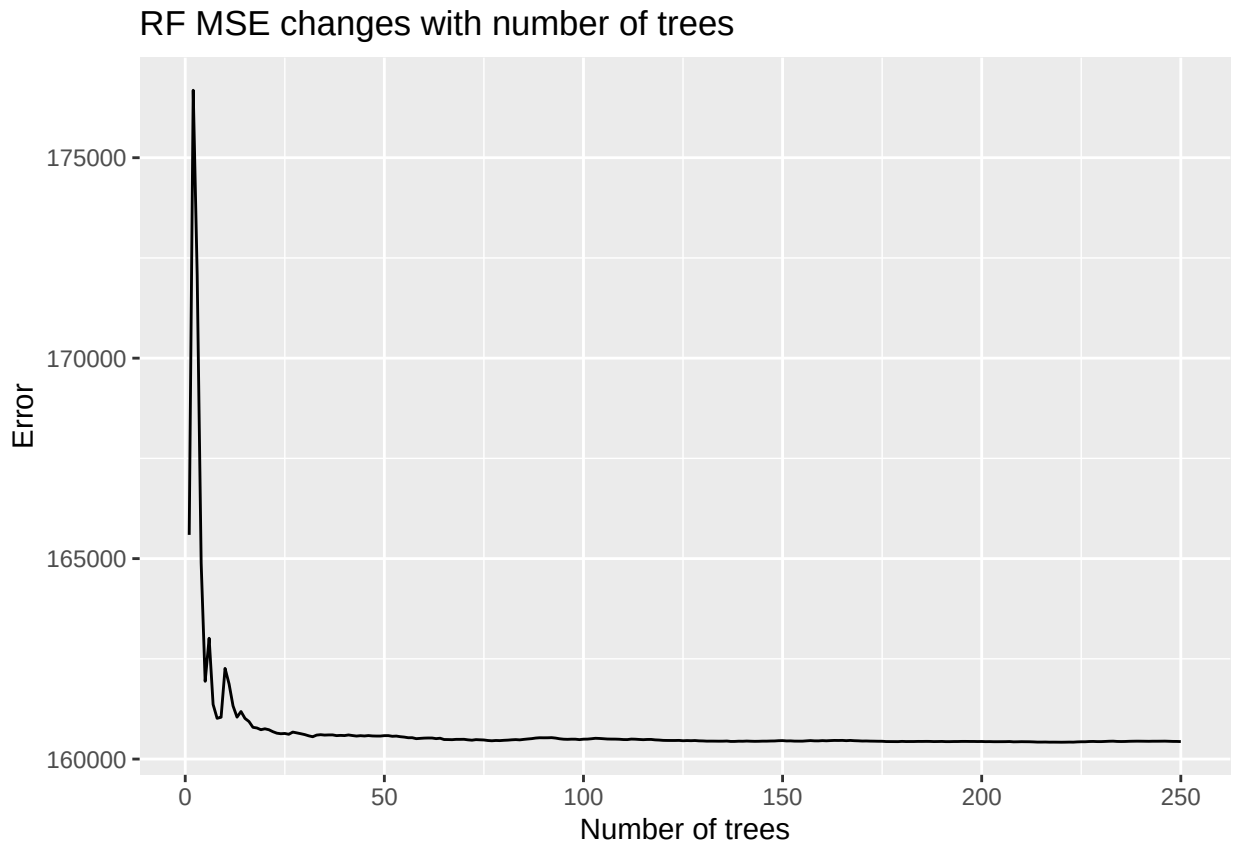
As shown from above, the decision tree is clearly a bad predictor

Decision Tree: Conclusion

Although the basic decision tree has great interpretability and can be understood by people even without a statistical background, it is also the worst predictor, and since our chosen model will be used to price insurance products, accuracy has been prioritised over interpretability.

Random Forest

There were 3 hyperparamters tuned for random forest- the number of trees, node size and the number of predictors considered at each tree split, commonly referred to as "m". Since random forest is computationally expensive, each predictor was optimised in isolation from the others. Although this is will not find the best combination of predictors, due to computational limitations it was a necessary tradeoff. ### Number of trees: Although random forest cannot be overfitted with large number of trees, in this data, any number of trees greater than 250 produced a relatively similar OOB error and so 250 was used for the final model.



Note: at 100 trees, the test error remains largely unchanged and so to create more models faster, 100 trees was deemed adequate enough to train the other parameters.

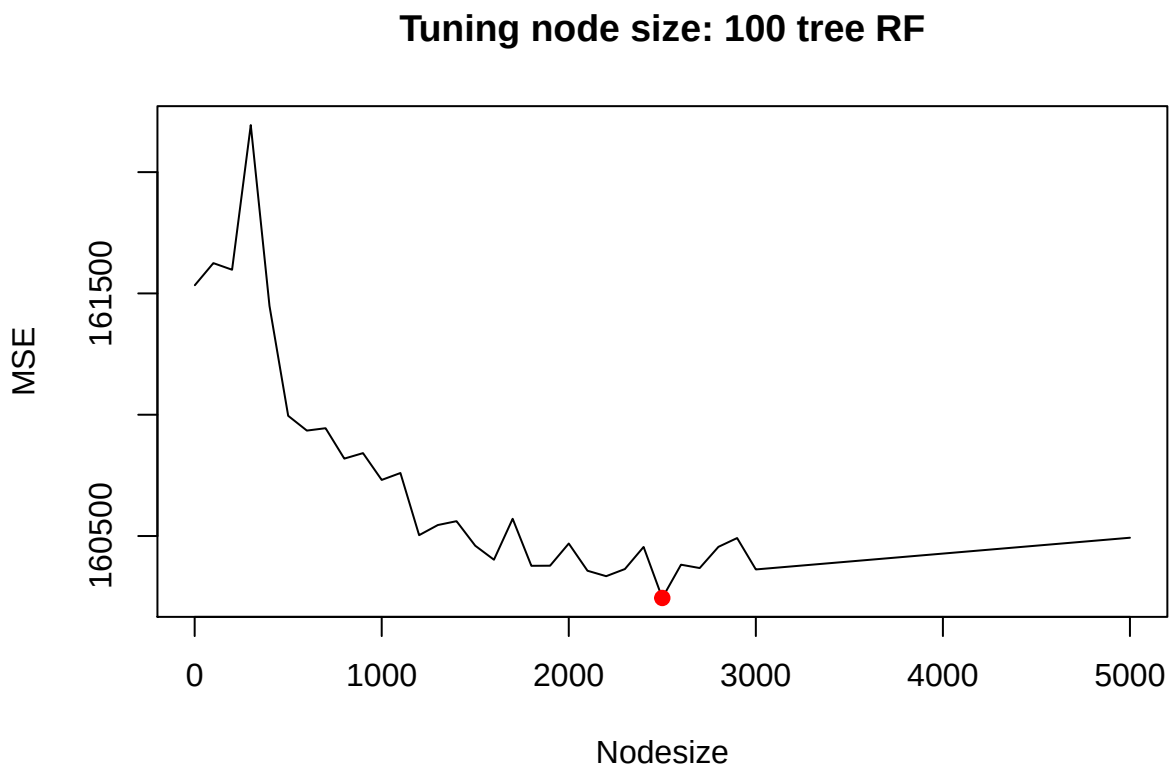
Node Size

Using the default node size value of 5 for regression in caret produced predictions that performed worse than even the standard regression tree. Therefore the node size also had to be trained in order to select the best bias-variance tradeoff. One of the limitations in this search was the computational cost of training each model. By incrementing the node size by 100 for each model, the optimal value was 2500, however, more training should be done for values greater than 3000 since it was unclear if there may have been early stopping. A value of 5000 was also used to validate that there was not any significant early stopping, however, more research should be done to ensure the most optimal node size was found.

```
bignodesizeMSE <- as.data.frame(c(1, seq(100,3000,100), 5000))
colnames(bignodesizeMSE)[1] <- "nodesize"
bignodesizeMSE$mse <- c(rep(0, nrow(bignodesizeMSE)))
bigbestnodeMSE <- Inf
```

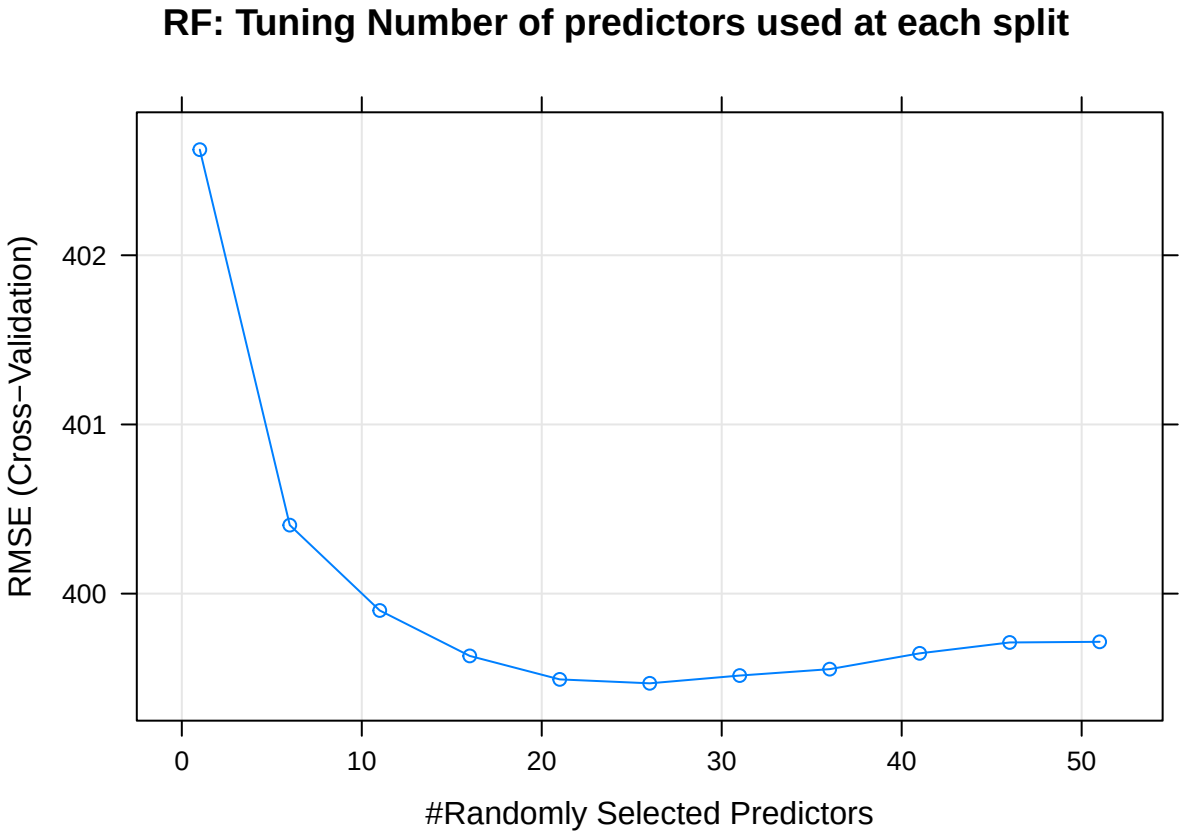
```r
bestmodel <- NULL
for(i in 1:(nrow(bignodesizeMSE)-1)) {
  j <- c(1,seq(100,3000,100))[i]
  model <- paste0("rf_", j)
  bignodesizeMSE[i, 2] = get(model)$finalModel$mse[100]
  if(bignodesizeMSE[i, 2] < bigbestnodeMSE) {
    bigbestnodesize <- j
    bigbestnodeMSE <- bignodesizeMSE[i, 2]
    bestmodel <- get(model)
  }
}
# some hardcoding
i = i+1
j <- 5000
bignodesizeMSE[i,2] <- rf_5000$finalModel$mse[100]
if(bignodesizeMSE[i, 2] < bigbestnodeMSE) {
    bigbestnodesize <- j
    bigbestnodeMSE <- bignodesizeMSE[i, 2]
}

plot(bignodesizeMSE$nodesize, bignodesizeMSE$mse, type = "l", xlab = "Nodesize", ylab = "MSE", main = "
points(bigbestnodesize, bigbestnodeMSE, col = "red", pch = 19)
```



Number of predictors considered at each split:

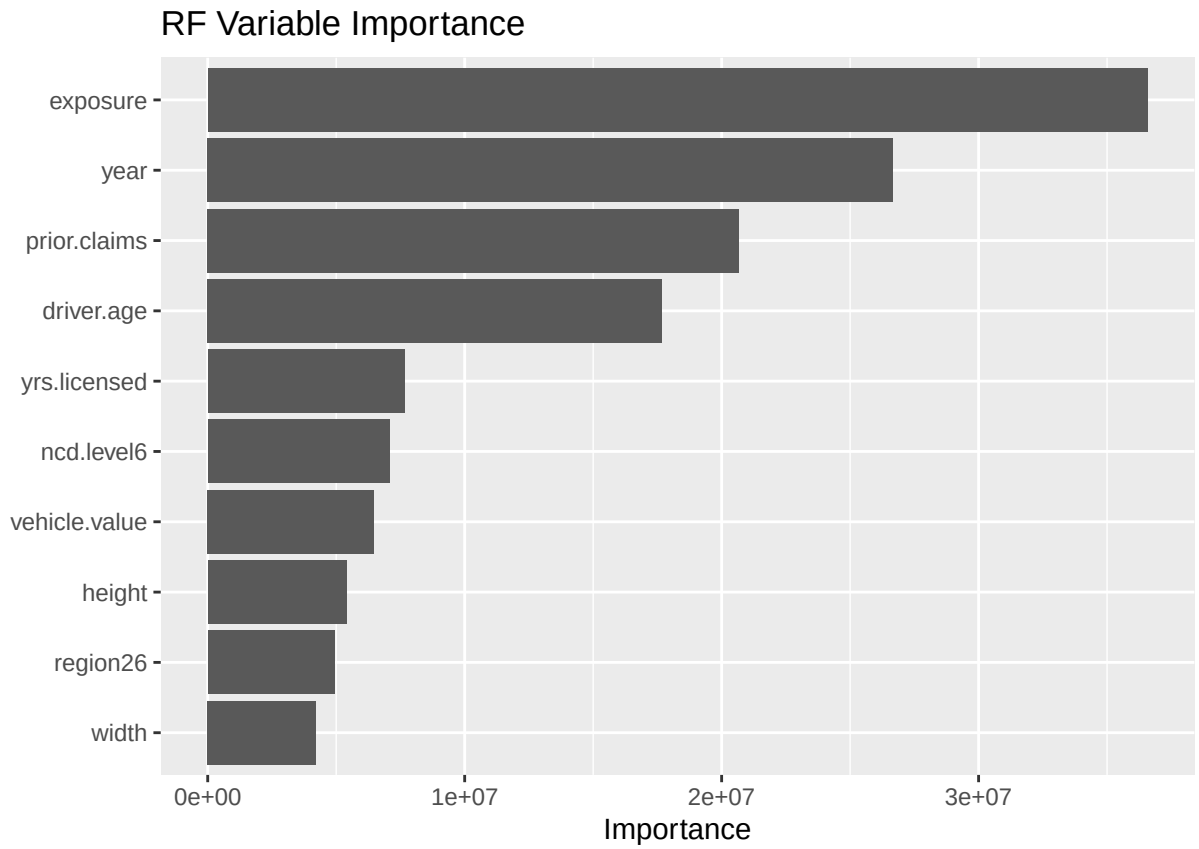
M values between 1 and 51 were modelled and the value which minimised the CV error was 26.



Finally these values were combined to create the final random forest model.

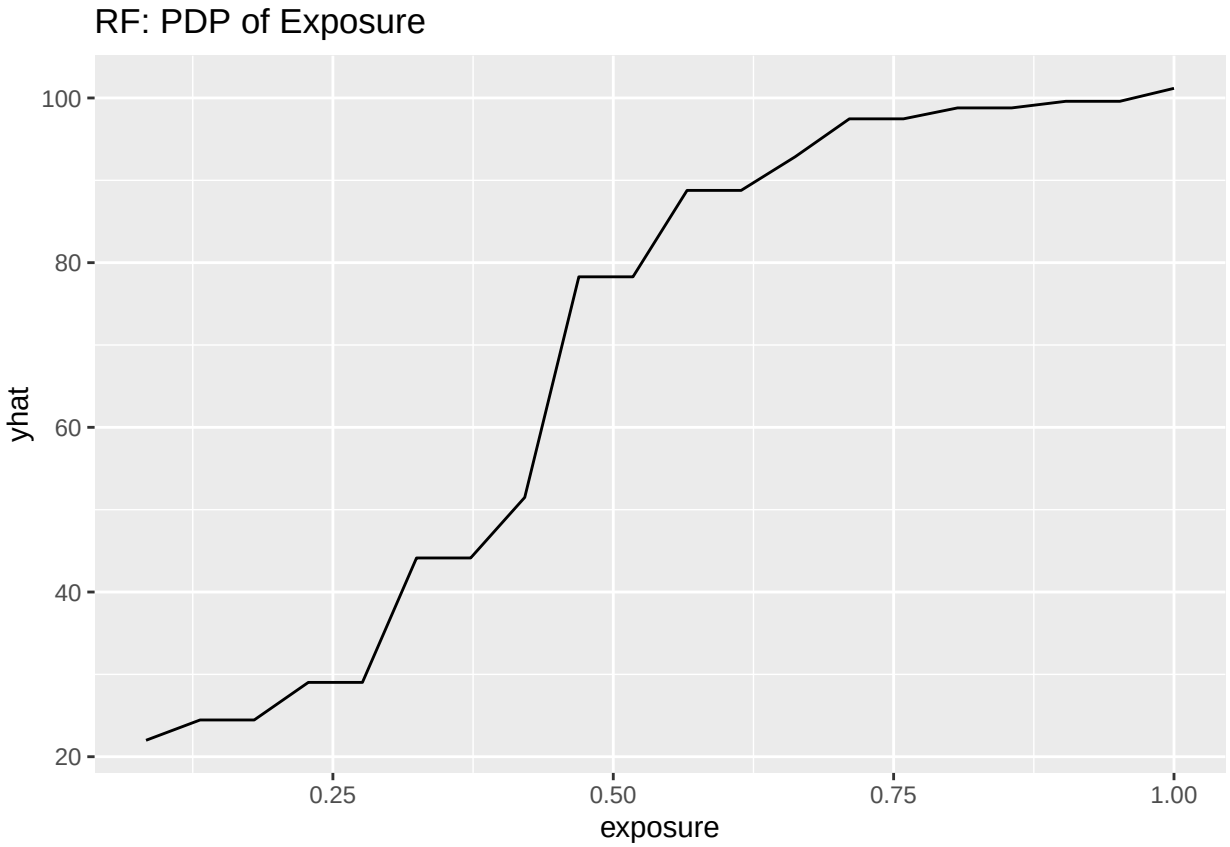
Random Forest: Advantages

Best predictive power Has some interpretation from the VIP and PDPs.



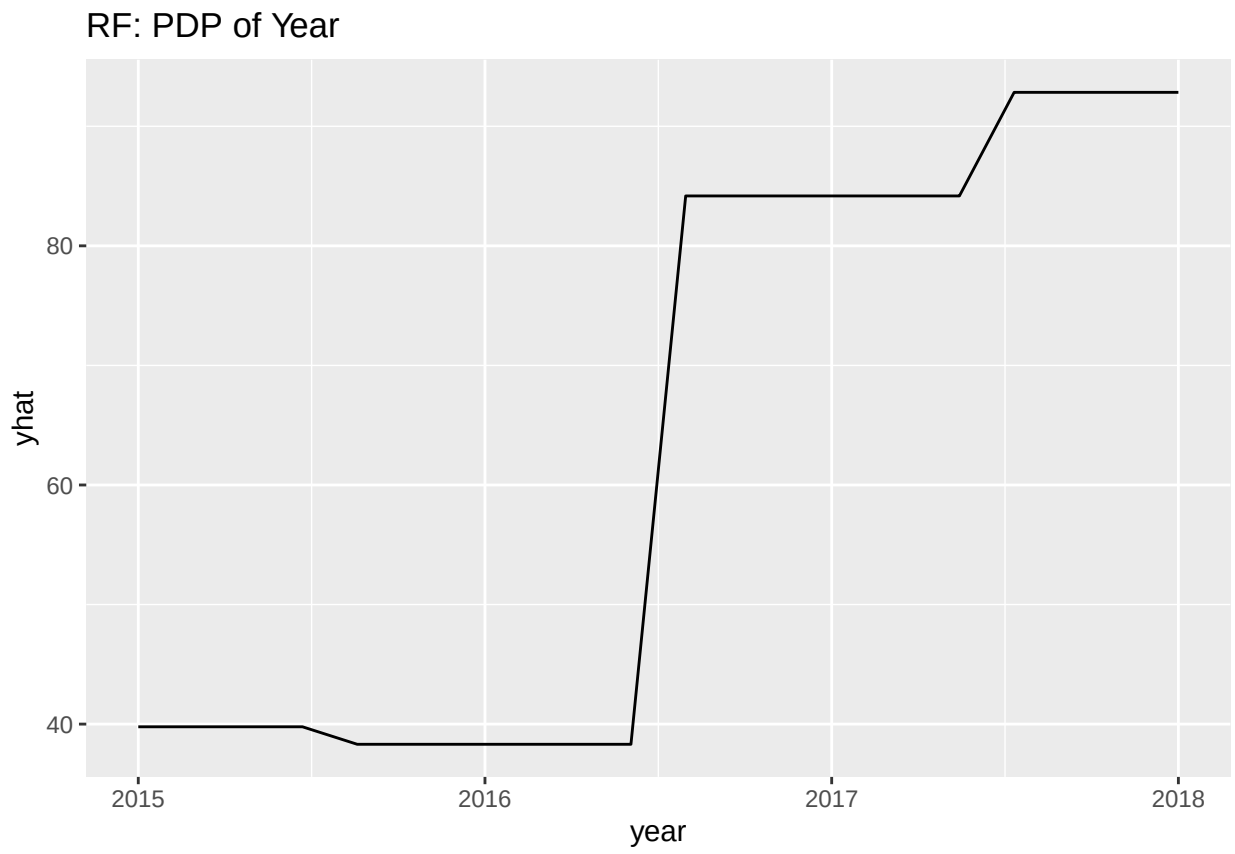

```
## Warning: Use of ‘object[[1L]]‘ is discouraged. Use ‘.data[[1L]]‘ instead.

## Warning: Use of ‘object[["yhat"]]‘ is discouraged. Use ‘.data[["yhat"]]‘
## instead.
```



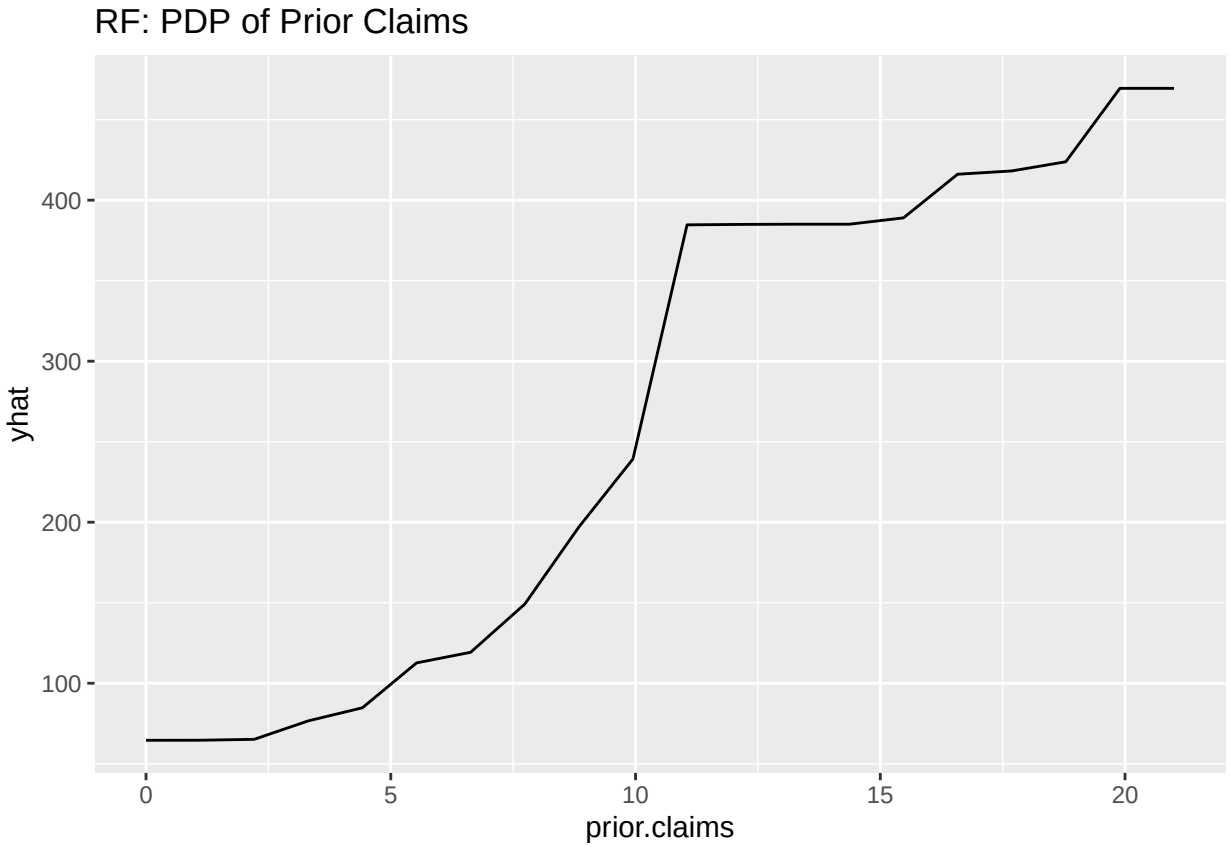

```
## Warning: Use of ‘object[[1L]]‘ is discouraged. Use ‘.data[[1L]]‘ instead.

## Warning: Use of ‘object[["yhat"]]‘ is discouraged. Use ‘.data[["yhat"]]‘
## instead.
```



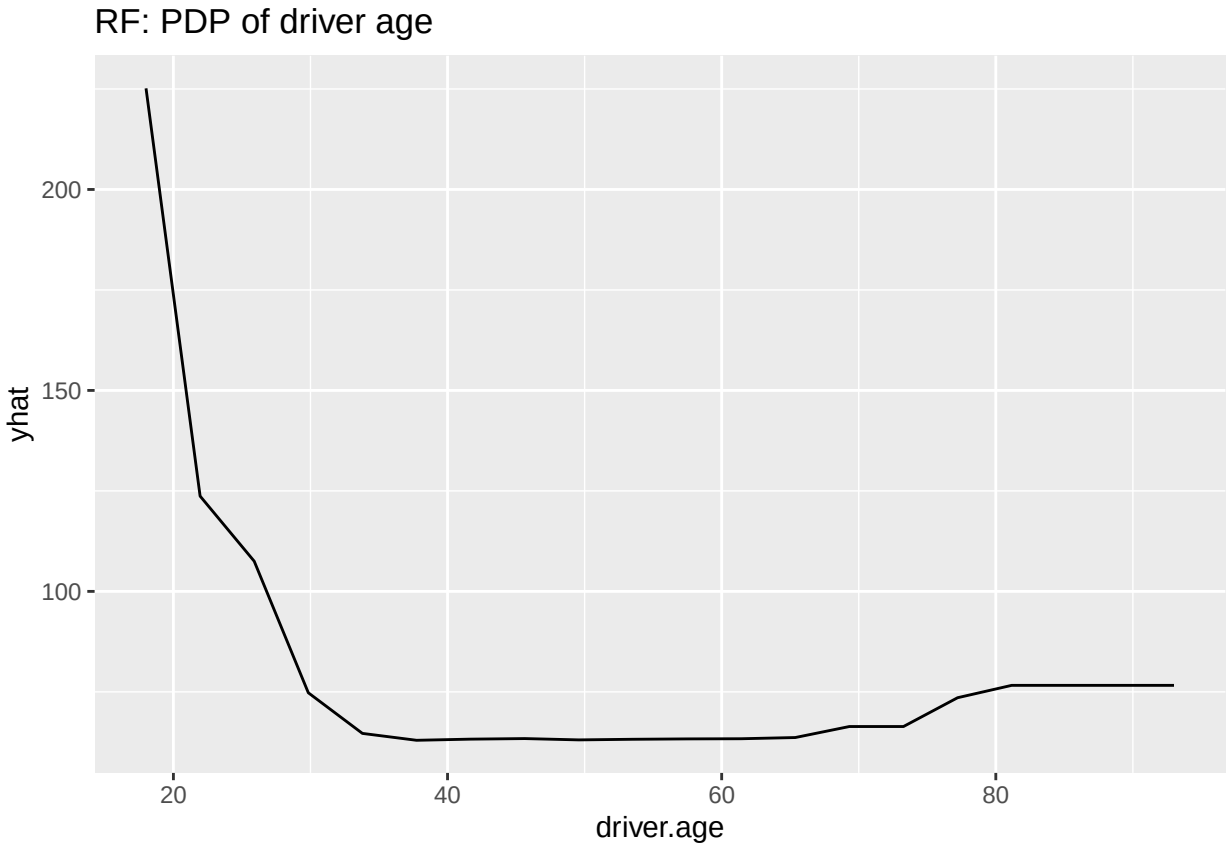

```
## Warning: Use of ‘object[[1L]]‘ is discouraged. Use ‘.data[[1L]]‘ instead.

## Warning: Use of ‘object[["yhat"]]‘ is discouraged. Use ‘.data[["yhat"]]‘
## instead.
```




```
## Warning: Use of ‘object[[1L]]‘ is discouraged. Use ‘.data[[1L]]‘ instead.

## Warning: Use of ‘object[["yhat"]]‘ is discouraged. Use ‘.data[["yhat"]]‘
## instead.
```



Random Forest: Disadvantes

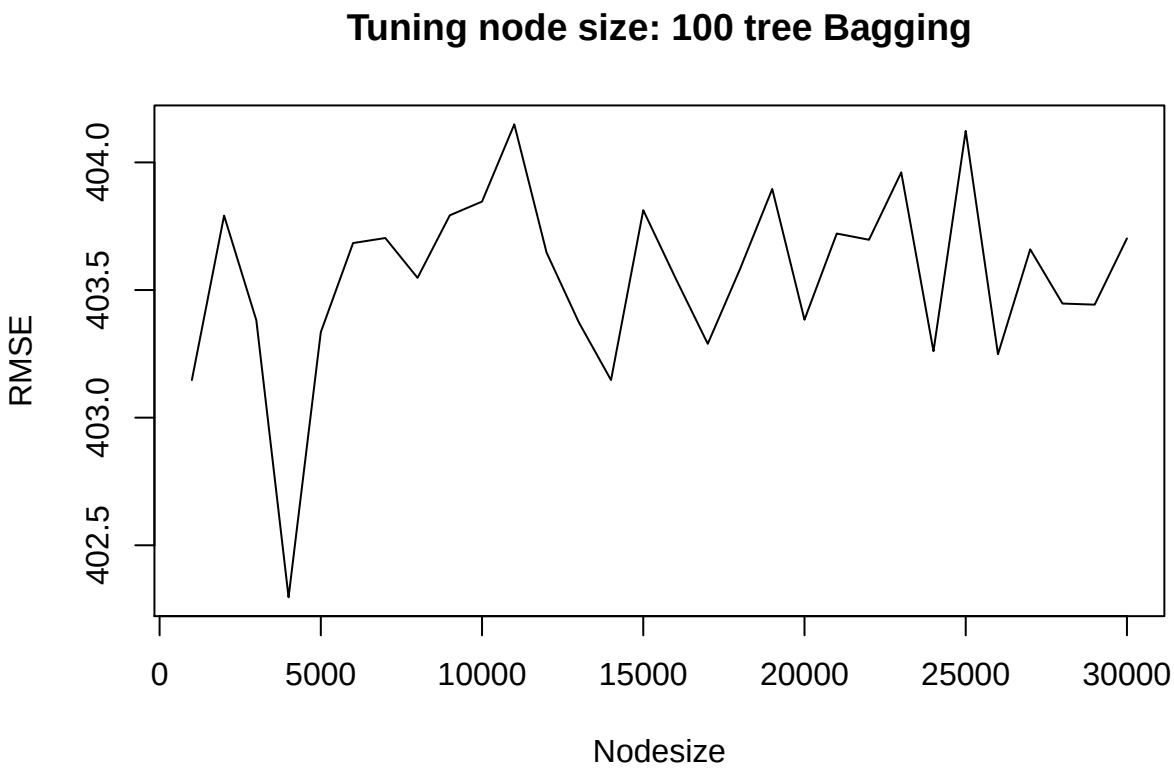
Very computationally expensive Less resposive to new information - takes long time to retrain Loses interpretability Further investigation into the nodesize

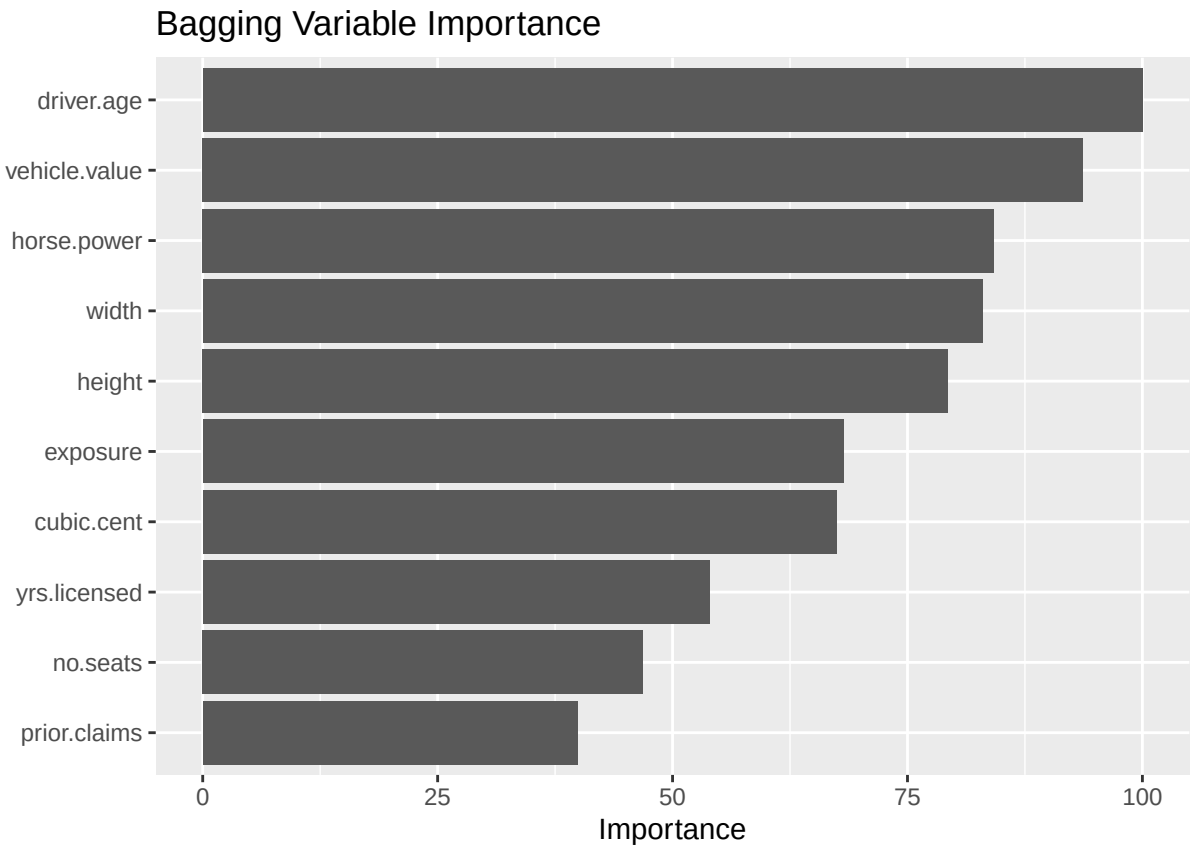
Random Forest: Conclusion

Overall, the random forest model was the best model. Although it was computationally slow to build and less interpretable than the decision tree, the improvements in the predictive power make up for these disadvantages.

Bagging

Bagging had to tune both the number of trees and the node size. However, since bagging is a subset of random forest, it was deemed appropriate to use 250 trees for both bagging and random forest. The main issue with fitting a bagging model was choosing the best node size. As seen from the plot below, there was no clear trend of the best node size. For future, more node size values should be tested. However, since bagging is growing a trees based on bootstrapped data, it was deemed appropriate to use the CP value found from the regression tree in part 1. This method also produced the lowest test RMSE.


Tuning node size: 100 tree Bagging
RMSE
402.5
403.0
403.5
404.0
0
5000
10000
15000
20000
25000
30000
Nodesize



Bagging: Advantages

More accuracte than decision tree

Bagging: Disadvantages

About same interpretability as random forest, still a worse predictor Computationally expensive, although not as bad as random forest since "m" value does not have to be trained

Bagging: Conclusion

Random forest strategy

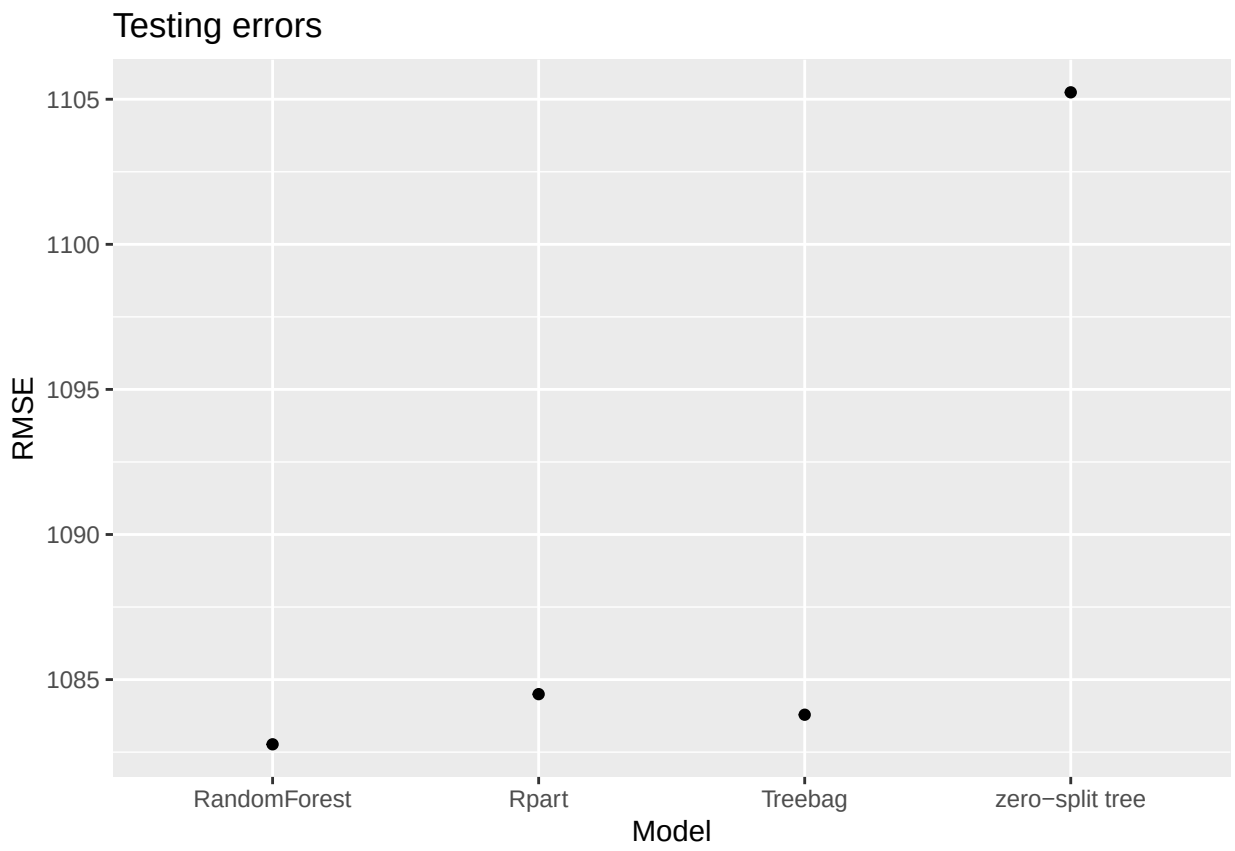
Since there is a computational time complexity constraint some of the hyper paremeters will be considered indivuadlly 1. Find best number of trees 2. Find best mtry value 3. Find best node-size value 4. Remove unimportant predictors. 5. Repeat steps 2 and 3 and see if there is a difference (tbh not sure if steps 4 and 5 are needed since overfitting is not usually an issue with random forest)

```
## # A tibble: 12,186 x 6
##    exposure claim.incurred zerosplit rpart     rf treebag
##       <dbl>          <dbl>     <dbl> <dbl>  <dbl>   <dbl>
##  1    0.917           606.      66.8  46.9 124.      45.3
```

```
##  2    0.167             0       66.8  30.0  40.7     29.5
##  3    0.917             0       66.8  46.9  51.2     47.0
##  4    0.167          2863.      66.8  30.0  12.7     29.5
##  5    0.917             0       66.8  46.9  71.7     47.0
##  6    0.25              0       66.8  30.0  14.7     29.5
##  7    0.167             0       66.8  30.0   8.70    29.5
##  8    0.167             0       66.8  30.0   8.70    29.5
##  9    0.25              0       66.8  30.0  16.1     29.5
## 10    0.917             0       66.8  46.9  68.0     47.0
## # ... with 12,176 more rows
```

```
## # A tibble: 12,186 x 6
##    exposure pure.prem zerosplit rpart    rf treebag
##       <dbl>     <dbl>     <dbl> <dbl> <dbl>   <dbl>
##  1    0.917      661.      72.9  51.2 136.     49.4
##  2    0.167        0      401.  180.  244.    177.
##  3    0.917        0       72.9  51.2  55.9    51.3
##  4    0.167    17178.     401.  180.   76.3   177.
##  5    0.917        0       72.9  51.2  78.3    51.3
##  6    0.25         0      267.  120.   58.9   118.
##  7    0.167        0      401.  180.   52.2   177.
##  8    0.167        0      401.  180.   52.2   177.
##  9    0.25         0      267.  120.   64.3   118.
## 10    0.917        0       72.9  51.2  74.2    51.3
## # ... with 12,176 more rows
```

```
##             Model     RMSE
## 1 zero-split tree 1105.238
## 2           Rpart 1084.498
## 3    RandomForest 1082.770
## 4         Treebag 1083.789
```



Appendix

R code-

```
library(randomForest)
library(rpart.plot)
library(EnvStats)
library(tidyverse)
library(corrplot)
library(ROSE)
library(caret)
library(vip)
library(pdp)
library(pROC)
library(ROCR)

library(parallel)
library(doParallel)

setwd("~/University/year3/T3/ACTL4305/Assignment/Assignment 2")

# Importing data
data <- read_csv("A2-data.csv")[,-1]
```

```r
factor_cols <- c("business.type", "driver.gender", "marital.status", "ncd.level", "region", "body.code"
data <- data %>%
  mutate_at(vars(factor_cols), funs(factor)) # makes all these factor variables


# Brief data exploration
str(data)

par(mfrow=c(1,2))
for(i in 1:ncol(data)) {
  if(class(data[,i][[1]]) == "factor") {
    var = paste0(colnames(data)[i])
    plot <- plot(data[,i], main = var)
  }
}

par(mfrow=c(1,2))
plot(density(data$claim.incurred), main = "Claim size distribution")
plot(density(filter(data, claim.incurred > 0)$claim.incurred), main = "Claim size distribution - No zer

## Collinearality check

numeric_cols <- unlist(lapply(data, is.numeric))
numeric_data <- na.omit(data[,numeric_cols])
cor_data <- cor(numeric_data)
corrplot(cor_data, method = "circle")

# remove weight, length and vehicle age
data <- data %>% select(-c(weight, length, vehicle.age))

levels(data$claim.count)[6] <- "3"
levels(data$claim.count)[5] <- "3"
levels(data$claim.count)[4] <- "3+"

## Effects of house characteristics on size and price
par(mfrow=c(1,2))
factorVar <- c(3, 5, 6, 8, 9, 10, 17, 19)
for(i in factorVar) {
  var <- colnames(data)[i]
  plot <- ggplot(data) +
    geom_point(aes(driver.age, claim.incurred, colour= factor(get(var)))) +
    labs(color=var) +
    ggtitle(paste("Effects of", var, "on Size vs Price"))
  print(plot)
}


# Box and Whisker

for(i in factorVar) {
  var <- colnames(data)[i]
  plot <- ggplot(filter(data, claim.incurred != 0)) +
    geom_boxplot(aes(claim.incurred, colour= factor(get(var)))) +
```

```
    labs(color=var) +
    ggtitle(paste("Spread of non-zero claims across", var))
  print(plot)
}


# Data Splitting
set.seed(654321)

train.index=createDataPartition(data$claim.incurred, p = 0.7, list = FALSE)

train=data[train.index,]
train <- train %>% select(-claim.count)


test=data[-train.index,]

# Loading saved models
rpart0 <- readRDS(file = "Objects/zerosplitFinal.RData")
rpart1 <- readRDS(file = "Objects/rpartFinal.RData")
rf1 <- readRDS(file = "Objects/rfFinal.RData")
treebag0 <- readRDS(file = "Objects/treebagFinal.RData")
rf0 <- readRDS(file = "Objects/mtuneRF.RData")
rf_1 <- readRDS(file = "Objects/rf_1.RData")
for(i in 1:30) {
  j <- i*100
  model_name <- paste0("rf_", j)
  print(model_name)
  filename <- paste0("Objects/", model_name, ".RData")
  assign(model_name, readRDS(file = filename))
}
rf_5000 <- readRDS(file = "Objects/rf_5000.RData")
bagnodeData <- readRDS("Objects/bagnodeData.Rda")
random_oob <- readRDS(file = "Objects/numTreeRF.RData")


# Caret's fit control parameter - held constant for duration of report
fitControl <- trainControl(
    method = "cv",
    number = 5,
    allowParallel = TRUE) # parrallel computing saves run time

################ Decision Tree ################
rpart.grid <- expand.grid(cp=seq(0,0.03,0.0005))
rpart1 <- train(claim.incurred ~.,
                data = train,
                method = "rpart",
                trControl = fitControl,
                tuneGrid=rpart.grid)

plot(rpart1, main = "Decision Tree Complexity Parameter Tuning")

rpart.plot(rpart1$finalModel) # requires rpart.plot package
```

```
title("Rpart Decision Tree plot")

bestcp <- rpart1$bestTune

# Zero Split tree
rpart0 <- train(claim.incurred ~.,
                data = train,
                method = "rpart",
                trControl = fitControl,
                tuneGrid=expand.grid(cp=0.5))

################ Random Forest ################
cluster <- makeCluster(detectCores() - 1) # convention to leave 1 core for OS
registerDoParallel(cluster)
# To find optimal number of trees
random_oob <- train(claim.incurred ~ ., data = train,
            method = "rf",
            trControl = fitControl,
            ntree = 250,
            nodesize = 2500)
stopCluster(cluster)
registerDoSEQ()

# number of trees plot
oob <- random_oob$finalModel$mse

# compare error rates
tibble::tibble(`Out of Bag Error`= oob,ntrees = 1:random_oob$finalModel$ntree)%>%
    gather(Metric, Error, -ntrees) %>%
    ggplot(aes(ntrees, Error)) +
    geom_line()+
    xlab("Number of trees") +
  ggtitle("RF MSE changes with number of trees")

# Function to quickly create models with different sizes
sizedRf <- function(size, train) {
  cluster <- makeCluster(detectCores() - 1) # convention to leave 1 core for OS
  registerDoParallel(cluster)
  out <- train(claim.incurred ~ ., data = train,
            method = "rf",
            trControl = fitControl,
            ntree = 100,
            nodesize = size)
  stopCluster(cluster)
  registerDoSEQ()
  out
}

maxNodesize <- 3000
assign("rf_1", sizedRf(1, train))
for(i in 1:(3000/100)) {
  j <- i*100
  model_name <- paste0("rf_", j)
```

```
  print(model_name)
  assign(model_name, sizedRf(j, train))
}
assign("rf_5000", sizedRf(5000, train))

# Putting different node sizes into a table
bignodesizeMSE <- as.data.frame(c(1, seq(100,3000,100), 5000))
colnames(bignodesizeMSE)[1] <- "nodesize"
bignodesizeMSE$mse <- c(rep(0, nrow(bignodesizeMSE)))
bigbestnodeMSE <- Inf
bestmodel <- NULL
for(i in 1:(nrow(bignodesizeMSE)-1)) {
  j <- c(1,seq(100,3000,100))[i]
  model <- paste0("rf_", j)
  bignodesizeMSE[i, 2] = get(model)$finalModel$mse[100]
  if(bignodesizeMSE[i, 2] < bigbestnodeMSE) {
    bigbestnodesize <- j
    bigbestnodeMSE <- bignodesizeMSE[i, 2]
    bestmodel <- get(model)
  }
}
# some hardcoding
i = i+1
j <- 5000
bignodesizeMSE[i,2] <- rf_5000$finalModel$mse[100]
if(bignodesizeMSE[i, 2] < bigbestnodeMSE) {
    bigbestnodesize <- j
    bigbestnodeMSE <- bignodesizeMSE[i, 2]
}

plot(bignodesizeMSE$nodesize, bignodesizeMSE$mse, type = "l", xlab = "Nodesize", ylab = "MSE", main = "
points(bigbestnodesize, bigbestnodeMSE, col = "red", pch = 19)

# Tuning m paramter
mtry <- bestmodel$bestTune$mtry
rfGrid <-  expand.grid(.mtry=c(seq(1,51,5)))

cluster <- makeCluster(detectCores() - 1) # convention to leave 1 core for OS
registerDoParallel(cluster)
rf0 <- train(claim.incurred ~ ., data = train,
             method = "rf",
             trControl = fitControl,
             tuneGrid = rfGrid,
             ntree = 250,
             nodesize = bigbestnodesize)
stopCluster(cluster)
registerDoSEQ()

bestm <- rf0$results$mtry[which.min(rf0$results$RMSE)]
plot(rf0, main = "RF: Tuning Number of predictors used at each split")


# Best Random Forest model
```

```
cluster <- makeCluster(detectCores() - 1) # convention to leave 1 core for OS
registerDoParallel(cluster)
rf1 <- train(claim.incurred ~ ., data = train,
             method = "rf",
             trControl = fitControl,
             tuneGrid = expand.grid(.mtry = bestm),
             ntree = 250,
             nodesize = bigbestnodesize)
stopCluster(cluster)
registerDoSEQ()


# VIP and PDP's for random forest
vip(rf1$finalModel) +
  ggtitle("RF Variable Importance")

RfExposure <-partial(rf1, pred.var = "exposure", grid.resolution = 20) %>%
  autoplot() +
  ggtitle("RF: PDP of Exposure")
RfYear <- partial(rf1, pred.var = "year", grid.resolution = 20) %>%
  autoplot() +
  ggtitle("RF: PDP of Year")
RfPriorClaims <- partial(rf1, pred.var = "prior.claims", grid.resolution = 20) %>%
  autoplot() +
  ggtitle("RF: PDP of Prior Claims")
RfAge <- partial(rf1, pred.var = "driver.age", grid.resolution = 20) %>%
  autoplot() +
  ggtitle("RF: PDP of driver age")
RfExposure
RfYear
RfPriorClaims
RfAge


################ Bagging ################
cluster <- makeCluster(detectCores() - 1) # convention to leave 1 core for OS
registerDoParallel(cluster)
treebag0 <- train(claim.incurred ~.,
                  data = train,
                  method = "treebag",
                  trControl = fitControl,
                  ntree = 250,
                  control = rpart.control(cp = bestcp))
stopCluster(cluster)
registerDoSEQ()

# Function to quickly model different sized bagged trees
sizedBag <- function(size, train) {
  cluster <- makeCluster(detectCores() - 1) # convention to leave 1 core for OS
  registerDoParallel(cluster)
  out <- train(claim.incurred ~ ., data = train,
               method = "treebag",
               trControl = fitControl,
```

```
            ntree = 100,
            nodesize = size)
  stopCluster(cluster)
  registerDoSEQ()
  out
}

# Table with errors for the different nodesizes
maxNodesize <- 30000
nodeJumps <- 1000
bagnodeData <- as.data.frame(c(seq(nodeJumps,maxNodesize,nodeJumps)))
colnames(bagnodeData)[1] <- "nodesize"
bagnodeData$mse <- c(rep(0, nrow(bagnodeData)))
bagbestnodeMSE <- Inf
bestmodel <- NULL
for(i in 1:(maxNodesize/nodeJumps)) {
  j <- i*nodeJumps
  model_name <- paste0("bag_", j)
  print(model_name)
  model <- sizedBag(j, train)

  bagnodeData[i, 2] = model$results[2]$RMSE
  if(bagnodeData[i, 2] < bagbestnodeMSE) {
    bagbestnodesize <- j
    bagbestnodeMSE <- bagnodeData[i, 2]
    bestmodel <- list(model, model_name)
  }
}


# Bagging plots
bagbestnodesize <- bagnodeData$nodesize[which.min(bagnodeData$mse)]
plot(bagnodeData$nodesize, bagnodeData$mse, type = "l", xlab = "Nodesize", ylab = "RMSE", main = "Tuning
points(bagbestnodesize, bagbestnodesize, col = "red", pch = 19)


# VIP and PDP plots for bagging
vip(treebag0) +
  ggtitle("Bagging Variable Importance")

partVehVal <- partial(treebag0, pred.var = "vehicle.value", grid.resolution = 20) %>%
  autoplot() +
  ggtitle("Bagging: PDP of Vehicle Value")
partAge <- partial(treebag0, pred.var = "driver.age", grid.resolution = 20) %>%
  autoplot() +
  ggtitle("Bagging: PDP of driver age")
partHeight <- partial(treebag0, pred.var = "height", grid.resolution = 20) %>%
  autoplot()  +
  ggtitle("Bagging: PDP of height")



# Creating testing error predictions for the tables
```

```r
predTable <- test %>% select(exposure, claim.incurred)
predTable$zerosplit <- predict(rpart0, newdata = test)
predTable$rpart <- predict(rpart1, newdata = test)
predTable$rf <- predict(rf1, newdata = test)
predTable$treebag <- predict(treebag0, newdata = test)
predTable

predPrem <- predTable
predPrem$claim.incurred <- predPrem$claim.incurred/predPrem$exposure
colnames(predPrem)[2] <- "pure.prem"
predPrem$zerosplit <- predPrem$zerosplit/predPrem$exposure
predPrem$rpart <- predPrem$rpart/predPrem$exposure
predPrem$rf <- predPrem$rf/predPrem$exposure
predPrem$treebag <- predPrem$treebag/predPrem$exposure
predPrem


# Testing RMSE
test.errors <- as.data.frame(c("zero-split tree", "Rpart", "RandomForest", "Treebag"))
test.errors$RMSE <- c(caret::RMSE(predPrem$zerosplit, predPrem$pure.prem),
                      caret::RMSE(predPrem$rpart, predPrem$pure.prem),
                      caret::RMSE(predPrem$rf, predPrem$pure.prem),
                      caret::RMSE(predPrem$treebag, predPrem$pure.prem)
                      )
colnames(test.errors)[1] <- "Model"
test.errors

ggplot(test.errors) +
  geom_point(aes(x = Model, y = RMSE)) +
  ggtitle("Testing errors")


# Saving models so that the modelling process does not have to be rerun every time
# Best models
saveRDS(rpart0,file = "Objects/zerosplitFinal.RData")
saveRDS(rpart1,file = "Objects/rpartFinal.RData")
saveRDS(rf1,file = "Objects/rfFinal.RData")
saveRDS(treebag0,file = "Objects/treebagFinal.RData")

# intermediate models
saveRDS(rf0, file = "Objects/mtuneRF.RData")

saveRDS(rf_1, file = "Objects/rf_1.RData")
for(i in 1:30) {
  j <- i*100
  model_name <- paste0("rf_", j)
  print(model_name)
  filename <- paste0("Objects/", model_name, ".RData")
  saveRDS(get(model_name), file = filename)
}
saveRDS(rf_5000, file = "Objects/rf_5000.RData")

saveRDS(bagnodeData, file="Objects/bagnodeData.Rda")
```

```r
saveRDS(random_oob, file = "Objects/numTreeRF.RData")
```