Coursework (CM3111)
Big Data Analytics

Alistair Quinn

December 14, 2017

---

1 Data Exploration

1.1 Dataset Choice

I have chosen a dataset that was used in the CoIL 2000 Challenge, which is called the Caravan Insurance Challenge dataset available here: https://www.kaggle.com/uciml/caravan-insurance-challenge

The data set is free to use for non-comercial use. Although sourced from the UCL Machine Learning organization's kaggle page, the dataset is owned by Sentient Machine Research.

1.2 Technology-Platform

The dataset is contained in a csv file that is 245KB in size. As the dataset is so small I not need to use Big Data technology such as Hadoop. Instead I will use RStudio on a windows PC. I chose the dataset based on my current ability, and I found the idea of the dataset interesting.

1.3 Problem Statement Data Exploration

Each row in the table corresponds to a post code.The task with this dataset is to identify potential purchasers of caravan insurance policies. The class label in the dataset is called CARAVAN and has two values, 0 or 1. CARAVARN is 1 when that row would potentially purchase a caravan insurance policy.

First I'll load the dataset I'm using.

```r
#Set WD
setwd("D:/RGU/3rdYear/Semester1/Big Data Analytics/Coursework/wd")
#Load Data
df <- read.csv("Data/caravan-insurance-challenge.csv")
```

During my data exploration, I will be visualising my data using a library called ggplot2. This is a plotting system for R. I have chosen to use this library as I have experience with it from the labs in the course.

```
## Warning:  package 'ggplot2' was built under R version 3.4.2
```

```r
library(ggplot2)
```

Going to take a look at the number of rows and columns

```r
#Rows and Cols
nrow(df)
```

```
## [1] 9822
```

```r
ncol(df)
```

```
## [1] 87
```

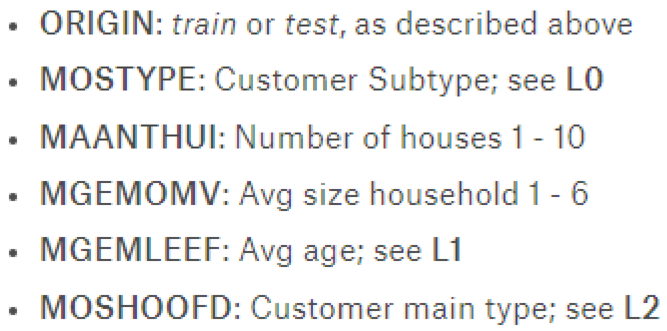
There are currently 9822 rows, and 87 columns in the dataset

Going to take a look at the features of the dataset

```r
#Names of columns
names(df)
```

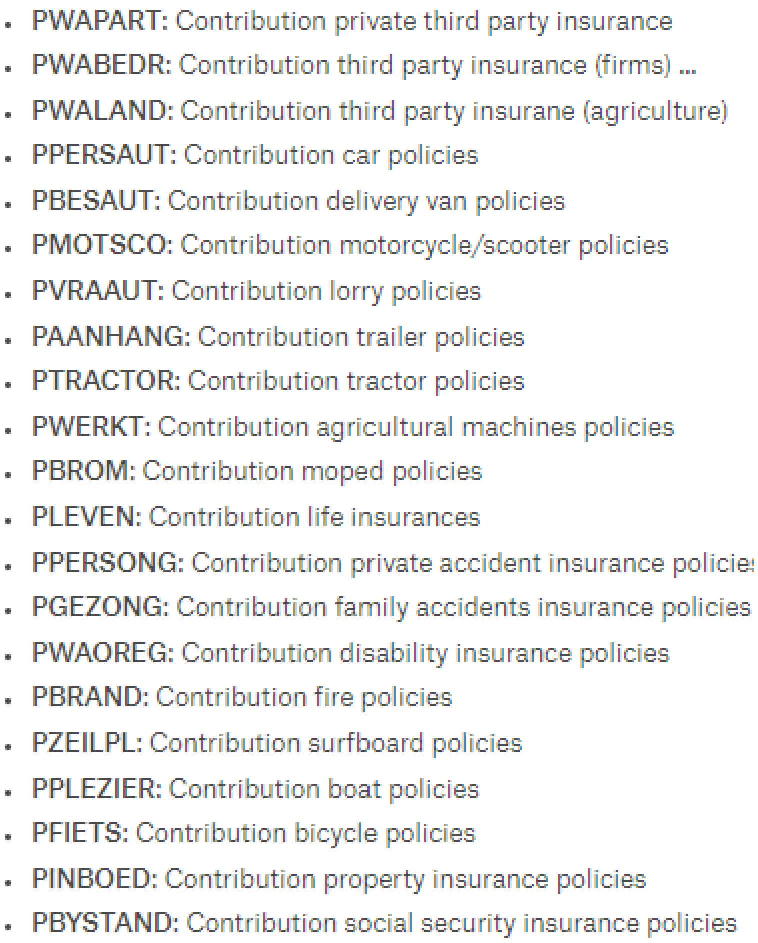
```
##  [1] "ORIGIN"   "MOSTYPE"  "MAANTHUI" "MGEMOMV"  "MGEMLEEF" "MOSHOOFD"
##  [7] "MGODRK"   "MGODPR"   "MGODOV"   "MGODGE"   "MRELGE"   "MRELSA"
## [13] "MRELOV"   "MFALLEEN" "MFGEKIND" "MFWEKIND" "MOPLHOOG" "MOPLMIDD"
## [19] "MOPLLAAG" "MBERHOOG" "MBERZELF" "MBERBOER" "MBERMIDD" "MBERARBG"
## [25] "MBERARBO" "MSKA"     "MSKB1"    "MSKB2"    "MSKC"     "MSKD"
## [31] "MHHUUR"   "MHKOOP"   "MAUT1"    "MAUT2"    "MAUT0"    "MZFONDS"
## [37] "MZPART"   "MINKM30"  "MINK3045" "MINK4575" "MINK7512" "MINK123M"
## [43] "MINKGEM"  "MKOOPKLA" "PWAPART"  "PWABEDR"  "PWALAND"  "PPERSAUT"
## [49] "PBESAUT"  "PMOTSCO"  "PVRAAUT"  "PAANHANG" "PTRACTOR" "PWERKT"
## [55] "PBROM"    "PLEVEN"   "PPERSONG" "PGEZONG"  "PWAOREG"  "PBRAND"
## [61] "PZEILPL"  "PPLEZIER" "PFIETS"   "PINBOED"  "PBYSTAND" "AWAPART"
## [67] "AWABEDR"  "AWALAND"  "APERSAUT" "ABESAUT"  "AMOTSCO"  "AVRAAUT"
## [73] "AAANHANG" "ATRACTOR" "AWERKT"   "ABROM"    "ALEVEN"   "APERSONG"
## [79] "AGEZONG"  "AWAOREG"  "ABRAND"   "AZEILPL"  "APLEZIER" "AFIETS"
## [85] "AINBOED"  "ABYSTAND" "CARAVAN"
```

Variables beginning with M are demographic statistics of the postal code.



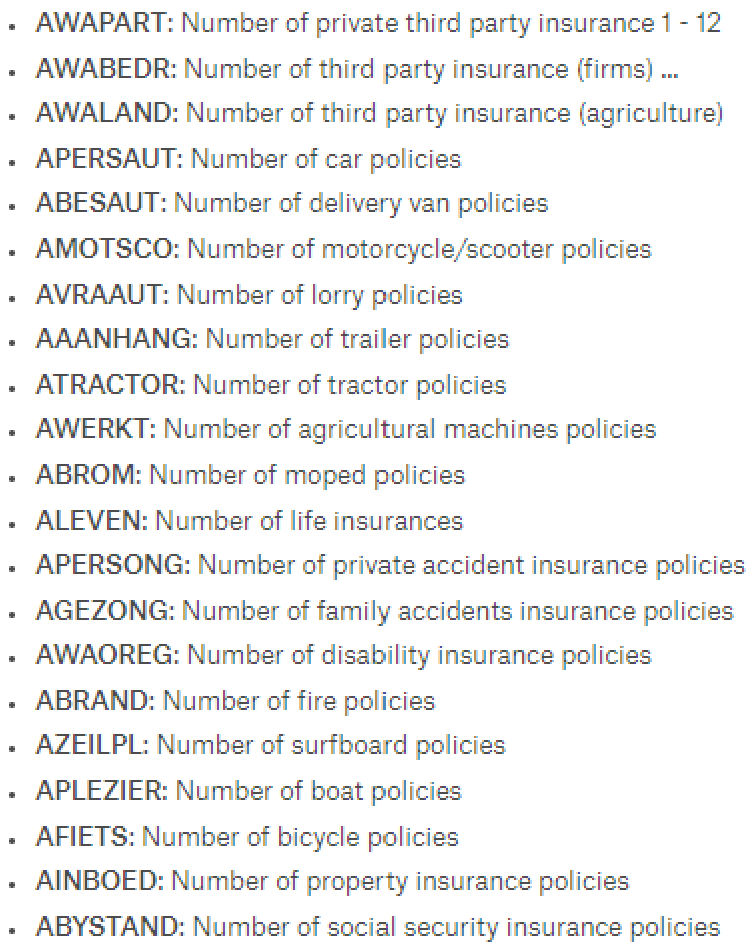
- **ORIGIN:** *train* or *test*, as described above
- **MOSTYPE:** Customer Subtype; see **L0**
- **MAANTHUI:** Number of houses 1 - 10
- **MGEMOMV:** Avg size household 1 - 6
- **MGEMLEEF:** Avg age; see **L1**
- **MOSHOOFD:** Customer main type; see **L2**

Variables beginning with P and A refer to product ownership and insurance statistics of the postal code. Variables beginning with P refer to contribution policies.



- **PWAPART:** Contribution private third party insurance
- **PWABEDR:** Contribution third party insurance (firms) ...
- **PWALAND:** Contribution third party insurane (agriculture)
- **PPERSAUT:** Contribution car policies
- **PBESAUT:** Contribution delivery van policies
- **PMOTSCO:** Contribution motorcycle/scooter policies
- **PVRAAUT:** Contribution lorry policies
- **PAANHANG:** Contribution trailer policies
- **PTRACTOR:** Contribution tractor policies
- **PWERKT:** Contribution agricultural machines policies
- **PBROM:** Contribution moped policies
- **PLEVEN:** Contribution life insurances
- **PPERSONG:** Contribution private accident insurance policie
- **PGEZONG:** Contribution family accidents insurance policies
- **PWAOREG:** Contribution disability insurance policies
- **PBRAND:** Contribution fire policies
- **PZEILPL:** Contribution surfboard policies
- **PPLEZIER:** Contribution boat policies
- **PFIETS:** Contribution bicycle policies
- **PINBOED:** Contribution property insurance policies
- **PBYSTAND:** Contribution social security insurance policies

variables beggining with A refer to number of policies.



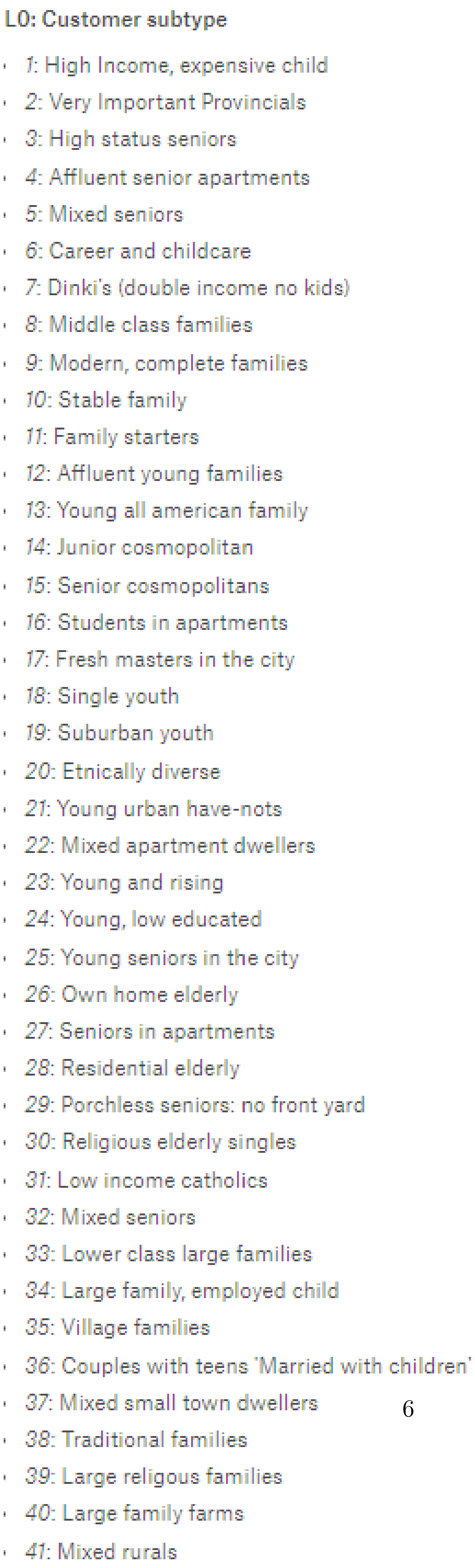
- **AWAPART:** Number of private third party insurance 1 - 12
- **AWABEDR:** Number of third party insurance (firms) ...
- **AWALAND:** Number of third party insurance (agriculture)
- **APERSAUT:** Number of car policies
- **ABESAUT:** Number of delivery van policies
- **AMOTSCO:** Number of motorcycle/scooter policies
- **AVRAAUT:** Number of lorry policies
- **AAANHANG:** Number of trailer policies
- **ATRACTOR:** Number of tractor policies
- **AWERKT:** Number of agricultural machines policies
- **ABROM:** Number of moped policies
- **ALEVEN:** Number of life insurances
- **APERSONG:** Number of private accident insurance policies
- **AGEZONG:** Number of family accidents insurance policies
- **AWAOREG:** Number of disability insurance policies
- **ABRAND:** Number of fire policies
- **AZEILPL:** Number of surfboard policies
- **APLEZIER:** Number of boat policies
- **AFIETS:** Number of bicycle policies
- **AINBOED:** Number of property insurance policies
- **ABYSTAND:** Number of social security insurance policies

Going to check the factors of the dataset

```
#Names of columns
sapply(df,levels)
```

I have ommited the result of the above code, as it was far too large. Most of the columns in the current data are numeric values but are actually supposed to be factors. I will refactor these columns during pre-processing. The only variable that has been turned into a factor by R is the first one, ORIGIN. I will explore this factor later.

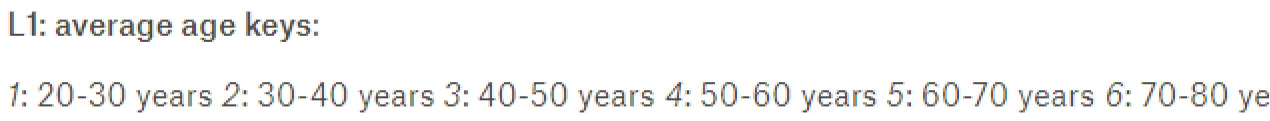
There are 4 keys that relate to this datset. A key for customer subtype:



L0: Customer subtype

- *1*: High Income, expensive child
- *2*: Very Important Provincials
- *3*: High status seniors
- *4*: Affluent senior apartments
- *5*: Mixed seniors
- *6*: Career and childcare
- *7*: Dinki's (double income no kids)
- *8*: Middle class families
- *9*: Modern, complete families
- *10*: Stable family
- *11*: Family starters
- *12*: Affluent young families
- *13*: Young all american family
- *14*: Junior cosmopolitan
- *15*: Senior cosmopolitans
- *16*: Students in apartments
- *17*: Fresh masters in the city
- *18*: Single youth
- *19*: Suburban youth
- *20*: Etnically diverse
- *21*: Young urban have-nots
- *22*: Mixed apartment dwellers
- *23*: Young and rising
- *24*: Young, low educated
- *25*: Young seniors in the city
- *26*: Own home elderly
- *27*: Seniors in apartments
- *28*: Residential elderly
- *29*: Porchless seniors: no front yard
- *30*: Religious elderly singles
- *31*: Low income catholics
- *32*: Mixed seniors
- *33*: Lower class large families
- *34*: Large family, employed child
- *35*: Village families
- *36*: Couples with teens 'Married with children'
- *37*: Mixed small town dwellers
- *38*: Traditional families
- *39*: Large religous families
- *40*: Large family farms
- *41*: Mixed rurals

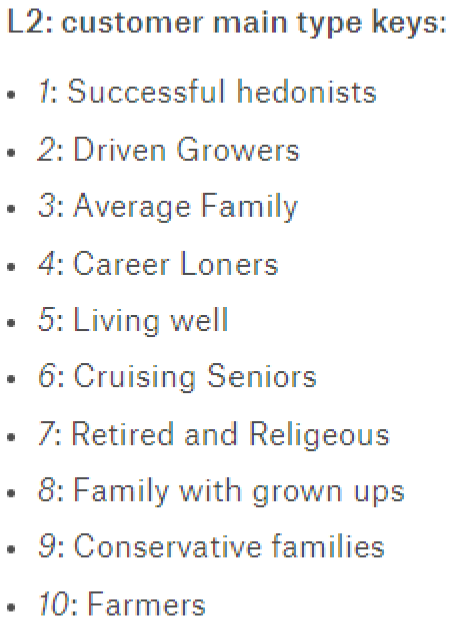
A key for average age:



L1: average age keys:

1: 20-30 years *2*: 30-40 years *3*: 40-50 years *4*: 50-60 years *5*: 60-70 years *6*: 70-80 ye

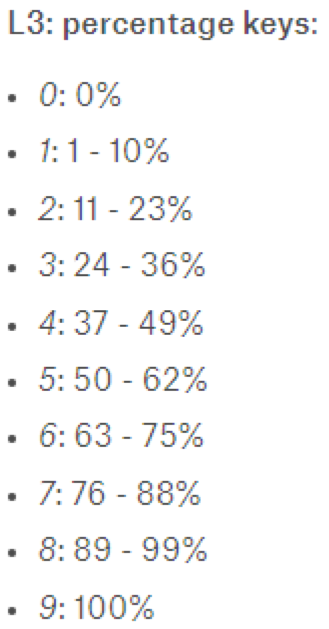
A key of customer main types:



L2: customer main type keys:

- *1*: Successful hedonists
- *2*: Driven Growers
- *3*: Average Family
- *4*: Career Loners
- *5*: Living well
- *6*: Cruising Seniors
- *7*: Retired and Religeous
- *8*: Family with grown ups
- *9*: Conservative families
- *10*: Farmers

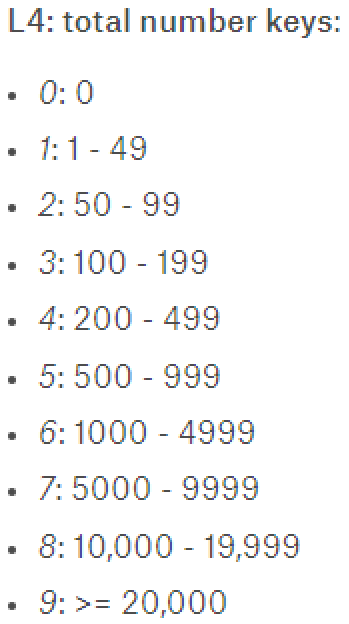
A key of percentage ranges:



L3: percentage keys:

- *0*: 0%
- *1*: 1 - 10%
- *2*: 11 - 23%
- *3*: 24 - 36%
- *4*: 37 - 49%
- *5*: 50 - 62%
- *6*: 63 - 75%
- *7*: 76 - 88%
- *8*: 89 - 99%
- *9*: 100%

and a key of total number ranges:



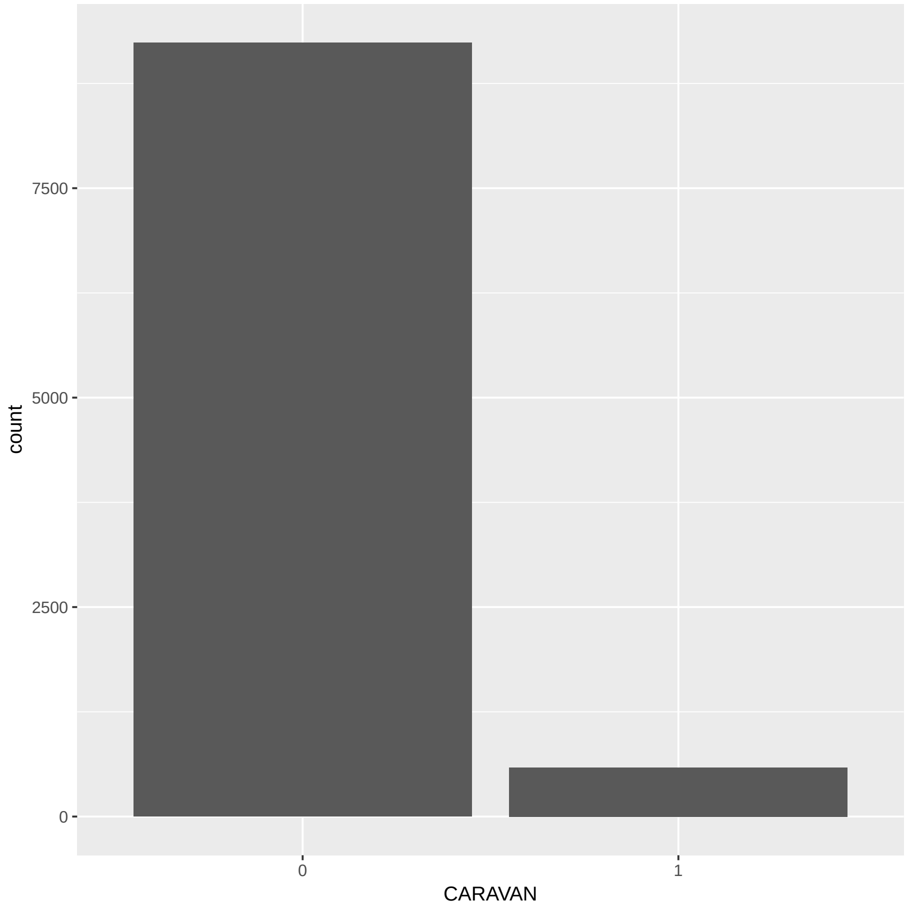
L4: total number keys:

- *0*: 0
- *1*: 1 - 49
- *2*: 50 - 99
- *3*: 100 - 199
- *4*: 200 - 499
- *5*: 500 - 999
- *6*: 1000 - 4999
- *7*: 5000 - 9999
- *8*: 10,000 - 19,999
- *9*: >= 20,000

I will use these keys to turn the appropriate columns into factors later, by maping the numeric values in the dataset to the appropritae value from the keys.

I will now take a look at the class label distribution

```
#Temp refactor
classLabelFreq <- data.frame(df$CARAVAN)
classLabelFreq$df.CARAVAN <- as.factor(df$CARAVAN)
#Class label Distribution Plot
ggplot(classLabelFreq,aes(x=df.CARAVAN)) + geom_bar() + labs(x="CARAVAN")
```



```
#Size of each factor level
length(classLabelFreq[classLabelFreq$df.CARAVAN=="0",])
```

```
## [1] 9236
```

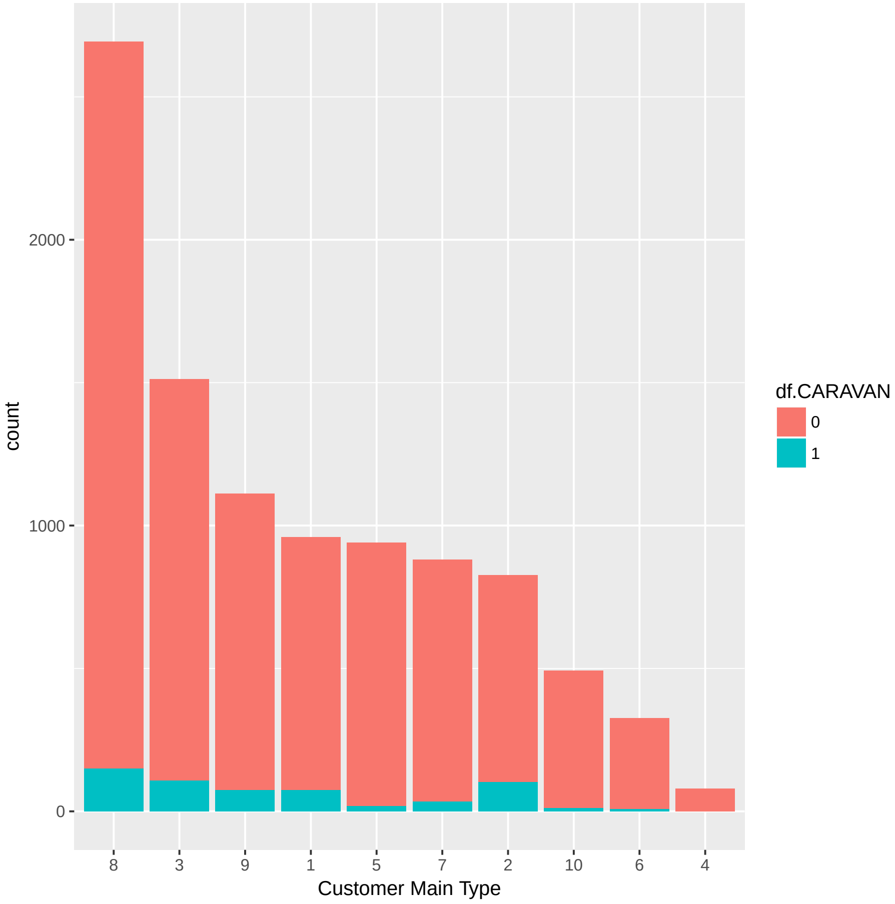
```
length(classLabelFreq[classLabelFreq$df.CARAVAN=="1",])
```

```
## [1] 586
```

There are 586 records that are likely to want caravan insurance. 9236 records that do not. Dataset has an imbalanced distribution in the class label. I will use a resampling technique during pre-processing to compensate for this.

Lets take a look at the disribution of Main Customer Type

```
#Temp refactor
custMainType <- data.frame(df$MOSHOOFD,df$CARAVAN)
custMainType$df.MOSHOOFD <- as.factor(custMainType$df.MOSHOOFD)
custMainType$df.CARAVAN <- as.factor(custMainType$df.CARAVAN)
#Plot of Customer Main Type
ggplot(custMainType,aes(x=reorder(df.MOSHOOFD,df.MOSHOOFD,function(x)-length(x)),fill=df.CA
```



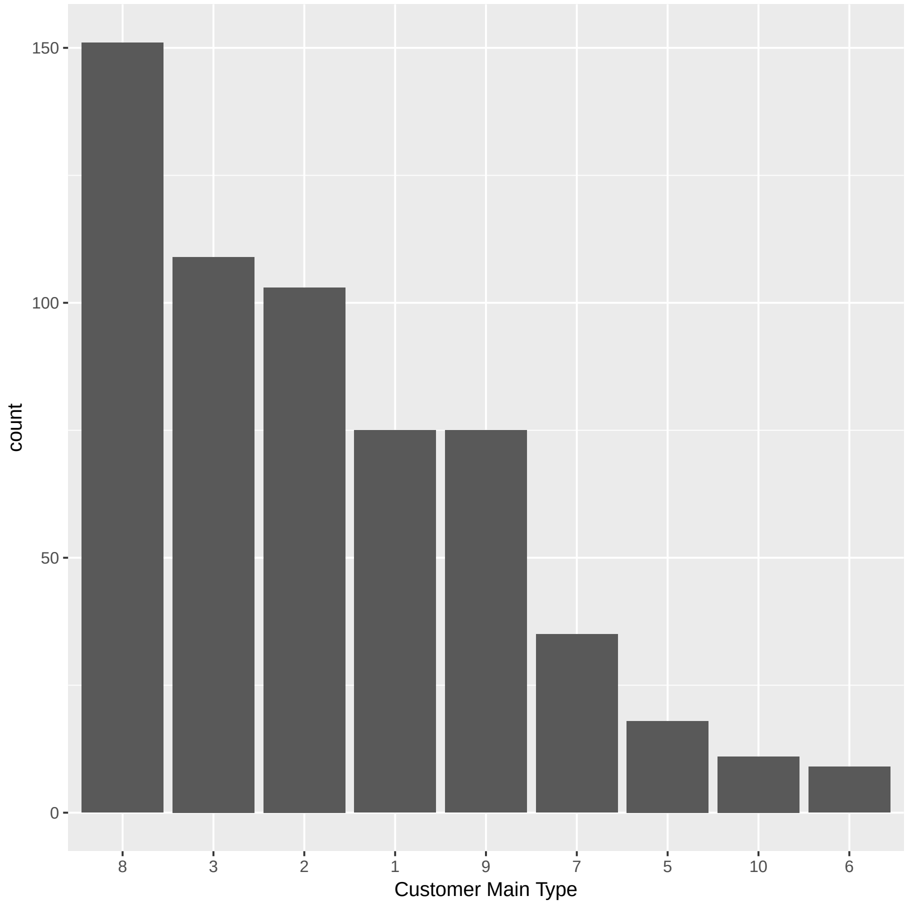
Most frequent Main Customer Type is 8:Family with Grown Ups, 2nd Most frequent is 3:Average Family and the 3rd most frequent is 9:conservative families.

The least frequent type is 4:Career Loners ,2nd least frequent is 6:Cruising Seniors and the 3rd least frequent is 10:Farmers.

Comparing to where CARAVAN is TRUE, you can see that the two most frequent main types are the same as the whole dataset. Customer Main Type 2: Driven Growers seems to be a bit more prominent where CARAVAN is true. You can also see that there are no instances of group 4: Career Loners in the rows where CARAVAN is true.

Lets take a closer look at the rows where CARAVAN is TRUE:

```
#Wants caravan
wantsCaravan <- df[df$CARAVAN==1,]
wantsCaravan$MOSHOOFD <- as.factor(wantsCaravan$MOSHOOFD)
wantsCaravan$MOSTYPE <- as.factor(wantsCaravan$MOSTYPE)

#Plot of Customer Main Type where wants caravan
ggplot(wantsCaravan,aes(x=reorder(MOSHOOFD,MOSHOOFD,function(x)-length(x)))) + geom_bar() +
```



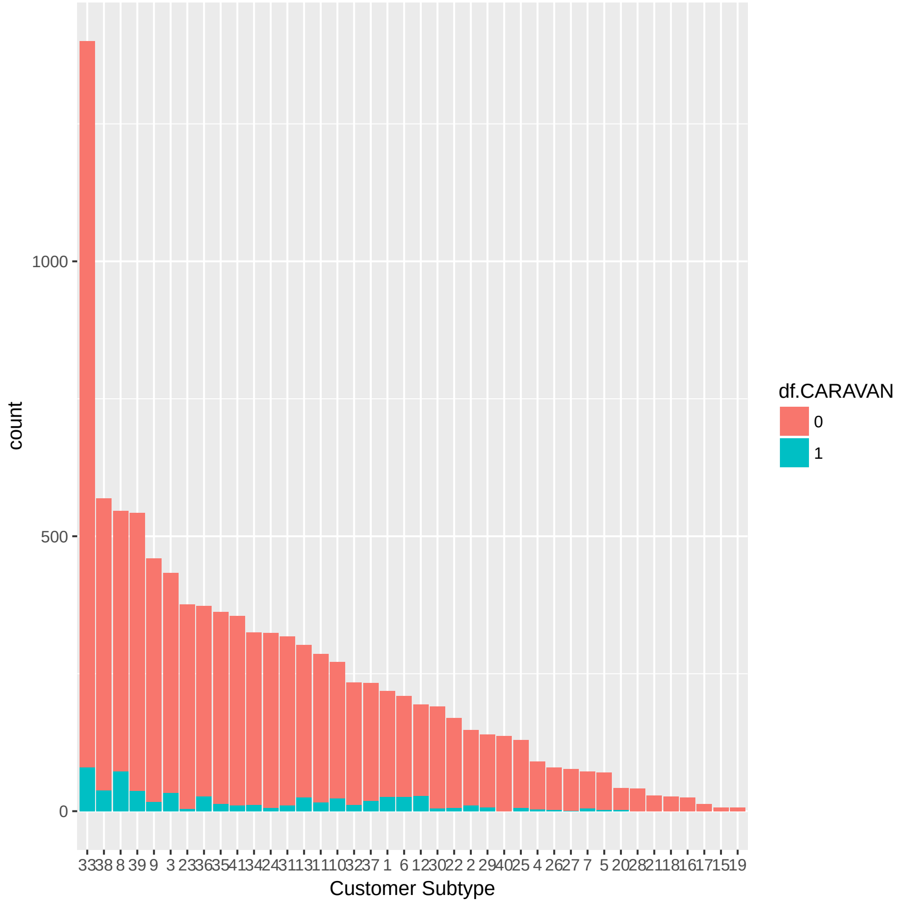
```
#Max and Min
mainCustType = table(wantsCaravan$MOSHOOFD)
names(which.max(mainCustType))

## [1] "8"

names(which.min(mainCustType))

## [1] "6"
```

The top 3 Main Customer Types are 8:Family with Grown Ups, 3: Average Family and 2:Driven Growers. As stated before the first 2 types are the same as the dataset as a whole. Type 2:Driven Growers has now overtaken 9:conservative families. 9:Conservative Families is joint 4th. Interesting that category 1: Successful Hedonists has the same number of occurances as 9:Conservaite Families. Hedonists are people who devote their lives to the pursuit of pleasure. There is perhaps a connection there between the idea of traveling by caravan and being a hedonist.

The 3 least frequent Main Custom Types 6:Cruising Seniors ,10:Farmers and 5:Living Well. As stated before there are no instances of 4:Career Loners when CARAVAN is TRUE. There is too small a number of instances of 4:Career Loners in the whole dataset to really say there is correlation but it is possible.

Now going to take a look at Customer Subtype

```
#Temp refactor
subCustType <- data.frame(df$MOSTYPE,df$CARAVAN)
subCustType$df.MOSTYPE <- as.factor(subCustType$df.MOSTYPE)
subCustType$df.CARAVAN <- as.factor(subCustType$df.CARAVAN)
#Plot of Customer subtype
ggplot(subCustType,aes(x=reorder(df.MOSTYPE,df.MOSTYPE,function(x)-length(x)),fill=df.CARAVA
```



The top 3 subtypes are 33:Lower class large families ,38:Traditional families and 8:Middle class families.

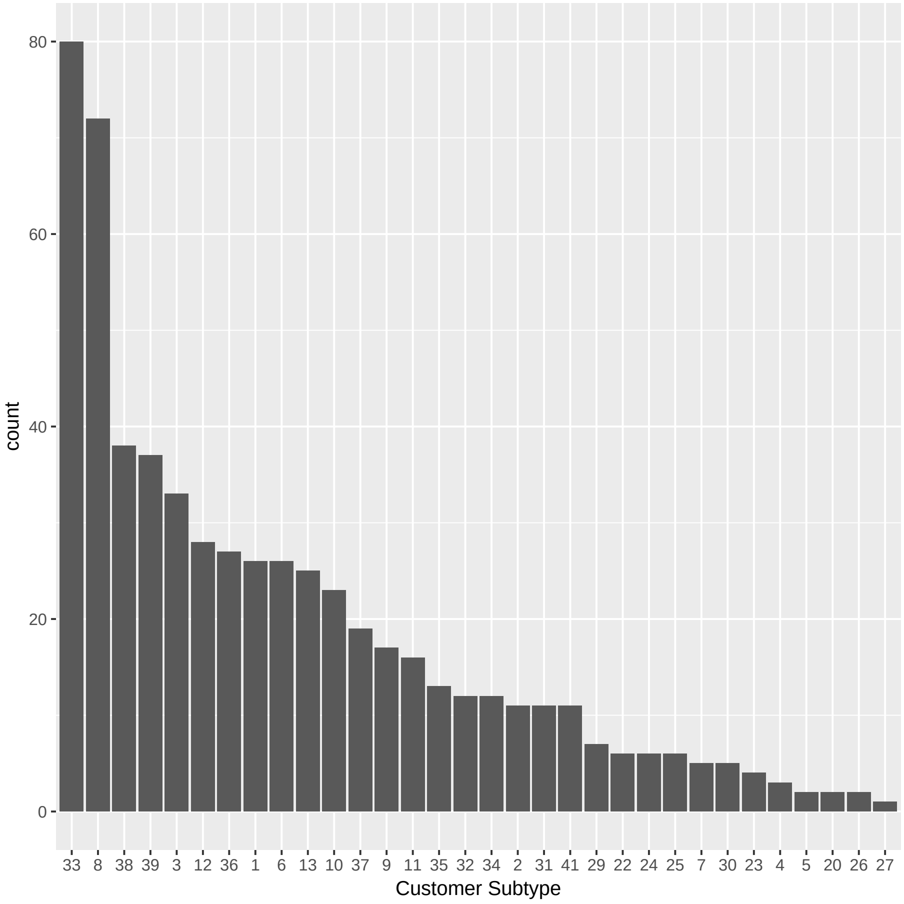
The bottom 3 are 19:Suburban Youth ,15:Senior cosmopolitans and 17:Fresh masters in the city. The dataset contains mostly data about families.

Taking a look at when CARAVAN is TRUE, you can see that It doesn't follow the same trend in frequency as the dataset as a whole This might mean that It could be used as a useful predictor. You can also see that there are no rows where CARAVAN is true that have subtypes of 40:Large family farms, 21:Young urban have-nots, 18:Single youth, 16:Students in apartments, 17:Fresh masters in the city, 15:Senior cosmopolitans and 19:Suburban youth.

This could suggest that postcode areas most comprised with familes are more likely to purchase caravan insurance.

Going to take a look at customer subtype when CARAVAN is TRUE

```r
#Plot of Customer Subtype where wants caravan
ggplot(wantsCaravan,aes(x=reorder(MOSTYPE,MOSTYPE,function(x)-length(x)))) + geom_bar() + la
```



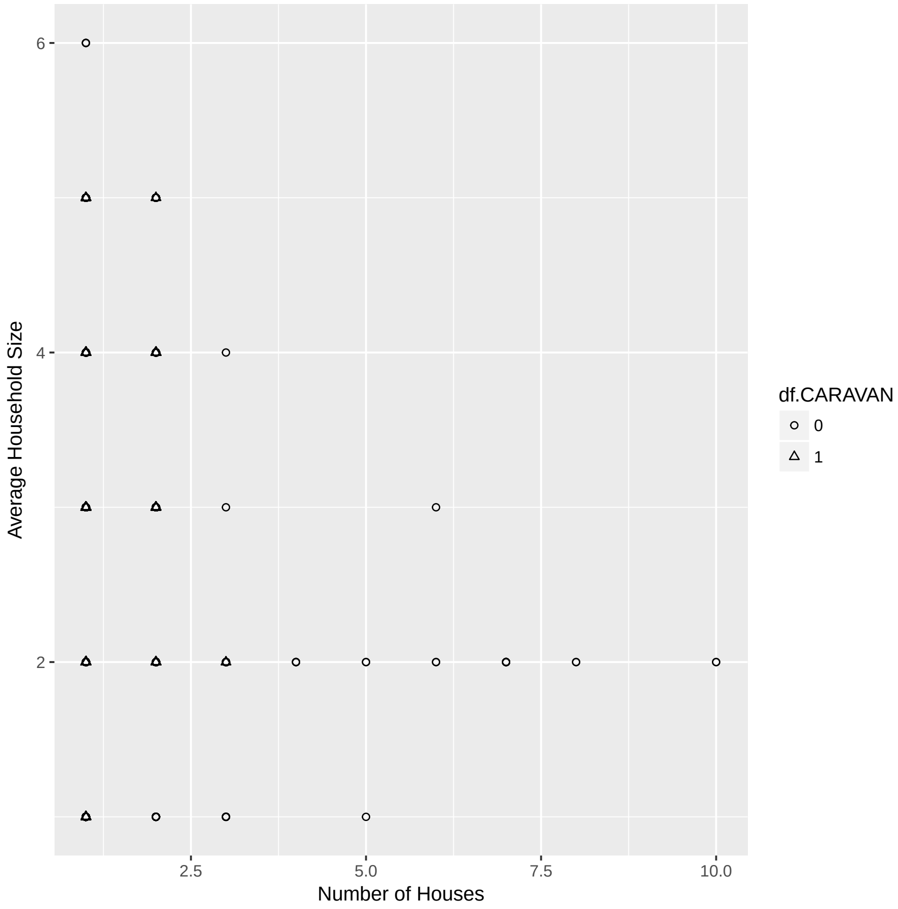
```
#Max and Min
subCustType = table(wantsCaravan$MOSTYPE)
names(which.max(subCustType))

## [1] "33"

names(which.min(subCustType))

## [1] "27"
```

Top 3 subtypes are the same as the datset as a whole but subtype 8:Middle class families is now more frequent than 38:Traditional families. This might mean that areas consisting of lower to middle class families are more likely to purchase caravan insurance than areas with mostly traditional families.

The bottom 3 subtypes are 27:Seniors in apartments ,26:Own home elderly and 20:Ethnically diverse. This supports the theory that areas with higher amounts of families are the most likely to purchase caravan insurance.

I am not going to take a look at two columns. MAANTHUI(Number of houses) and MGEMOMV(Avg size of household). MAATHUI is in the range of 1-10 and MGEMOMV is in the range of 1-6. They are the only two numeric values in the dataset the rest are factors. I will look at them together to see if there is any corelation. I will use ggplot2 again to make a scatter plot. These are integer values there will be overlap.

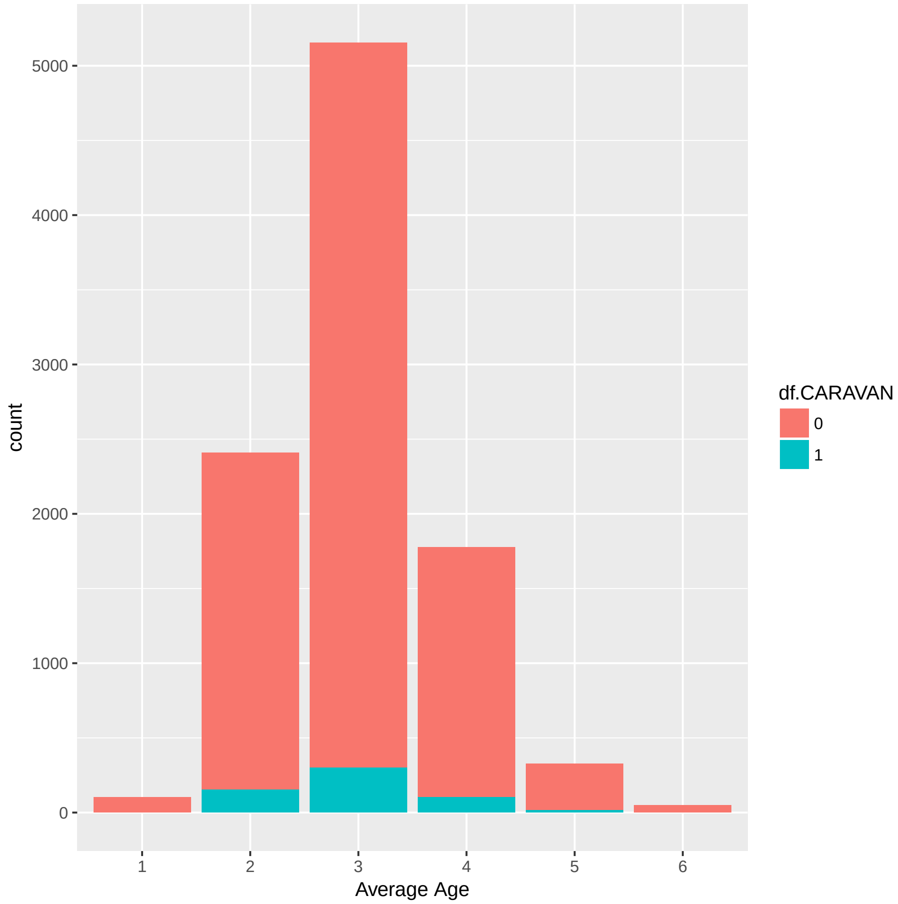
```
#Number of Houses and Avg size of household
houseData<-data.frame(df$MAANTHUI,df$MGEMOMV,df$CARAVAN)
houseData$df.CARAVAN<-as.factor(houseData$df.CARAVAN)
#ScatterPlot of both
ggplot(houseData,aes(x=df.MAANTHUI,y=df.MGEMOMV)) + geom_point(aes(shape=df.CARAVAN)) + scal
```



Taking a look at the results, average size of household decreases as the number of houses increases which makes sense. Looking at the points where CARAVAN is TRUE, There are no points when the number of houses is greater than 3. There are also no points when the average house size is greater than 5. This shows a potential connection between number of houses and CARAVAN. If I had to remove one of the two variables I would remove Average house size as I think number of houses has a greater corelation to CARAVAN equaling TRUE

I will now take a look at the average age variable, MGEMLEEF.

```
#Average Age
averageAge <- data.frame(df$MGEMLEEF,df$CARAVAN)
averageAge$df.MGEMLEEF <- as.factor(averageAge$df.MGEMLEEF)
averageAge$df.CARAVAN <- as.factor(averageAge$df.CARAVAN)
#Plot of Average Age
ggplot(averageAge,aes(x=df.MGEMLEEF,fill=df.CARAVAN)) + geom_bar() + labs(x="Average Age")
```



Average age is a factor, where each level is a range of ages. Lowest age range is 1:20-30 years, highest is 6:70-80 years. Looking at the graph, levels 3:40-50, 2:30-40 and 4:50-60 are the top 3 most frequent values. The extremes of 1 and 6 are the two lowest and do not have very many occurances. Looking at instances where CARAVAN is TRUE, They are in the same order as the whole dataset. Except there are no instances where CARAVAN is TRUE that contain 1 and 6 for average age.There might be a trend that areas were the average age is 1:20-30 or 6:70-80 would not buy caravan insurance. There isn't enough data to be sure and as the data where CARAVAN is true follows a similar trend to the data as a whole its unlikely there is a corelation between average age and CARAVAN.

The first column in the data set is ORIGIN. It has two values

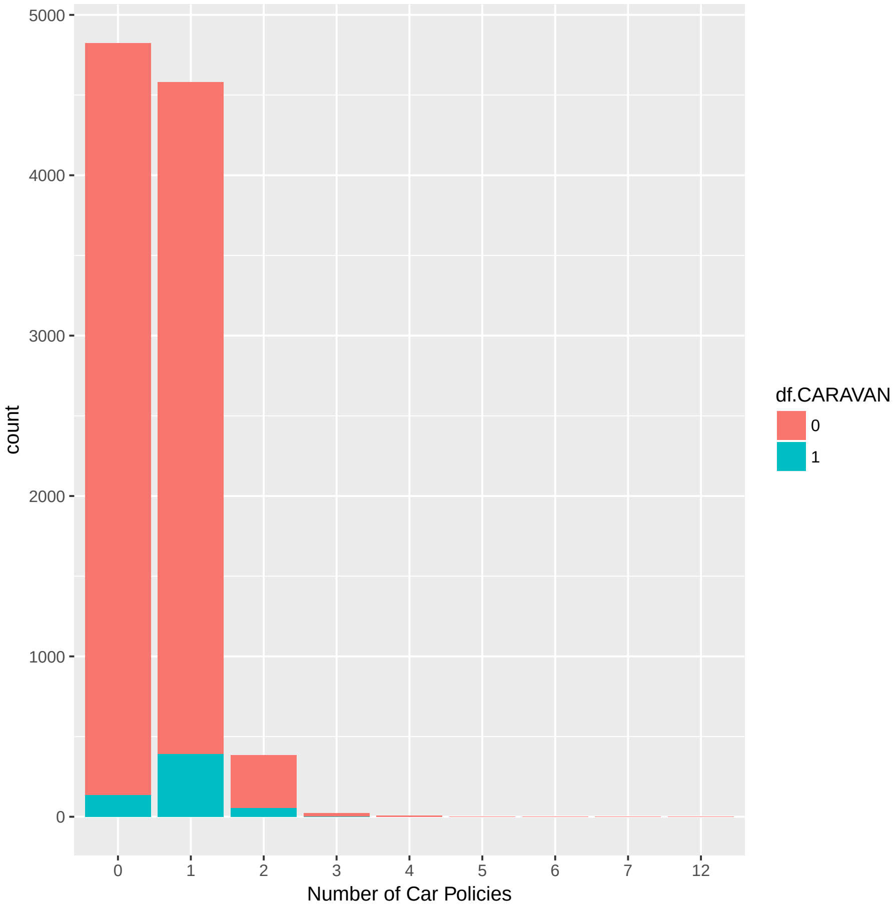
```
#Class label Distribution Plot
levels(df$ORIGIN)

## [1] "test"  "train"
```

This is the original source of the row from the challenge. The rows are already split into a train set and test set. I will remove this column later during preprocessing, as I plan to resample the data and split the data into train and test sets myself.

I have not looked at all the main variables.

I am not going to take a look at the APERSAUT column, which is the Number of Car Policies column. This is a factor, where each level equates to a range of values. The ranges are defined in the total number key. I have a feeling that it might be an important predictor, so want to try and confirm my theory.

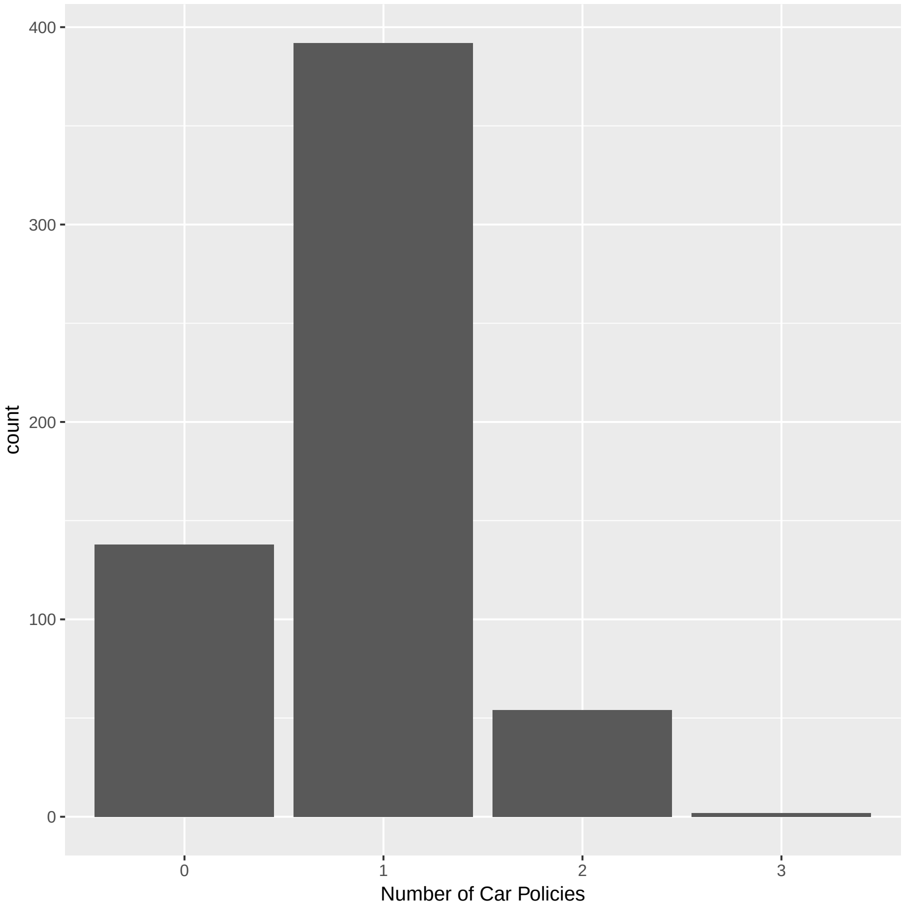
```
#Temp refactor
numberOfCarPolicies <- data.frame(df$APERSAUT,df$CARAVAN)
numberOfCarPolicies$df.APERSAUT <- as.factor(numberOfCarPolicies$df.APERSAUT)
numberOfCarPolicies$df.CARAVAN <- as.factor(numberOfCarPolicies$df.CARAVAN)
#Plot of APERSAUT
ggplot(numberOfCarPolicies,aes(x=reorder(df.APERSAUT,df.APERSAUT,function(x)-length(x)),fil
```



There is a factor level of 12 here on the graph. There is only supposed to be levels 0-9. This is likely a mistake in the dataset. When I factor this column later during pre processing and remove n/a values this wrong value should get removed.

Looking at when CARAVAN is TRUE, there is a possible relation between when the number of car policies are the level 1:1-49 and when CARAVAN is TRUE.

```r
#Comparing levels of APERSAUT(Number of car policies)
wantsCaravan$APERSAUT <- as.factor(wantsCaravan$APERSAUT)
ggplot(wantsCaravan,aes(x=APERSAUT)) + geom_bar() + labs(x="Number of Car Policies")
```



This factor is supposed to have 9 levels in total, but this column factored only contains the lowest 4 possible factors, 0,1,2 and 3. 1 is the most frequent. In this case 1 represents the range of values 1-49. 0 represents 0, 2 represents the range of values 50-99 and 3 represents the range of values 100-199.

This would suggest that post codes that contain a number of cars in the range of 1-49 are most likey to purchase caravan insurance. There are only 2 rows in the dataset that contain 3 for this column so this could be considered an outlier. Based on the graph the range of values for number of cars could be 0-99, or 0-199 if you include the two rows that have the value 3. There isn't enough data here to be sure.

I was originally confused by this result. I had assumed that I would find the opposite, that areas with large numbers of cars would likey need caravan insurance as I assumed you would need a car to tow a caravan. After further study of the information about the dataset, in this case CARAVAN actually refers to mobile homes. This would suggest the data is based on american post codes (or area codes) and that 'caravan' refers to a self driving vehicle that you can sleep in rather than a traditional british caravan.

Based on the graph its possible there is a correlation between this variable and CARAVAN being TRUE. This variable might be an important predictor.

1.4 Pre-Proccessing

First I will refactor all the appropriate columns

```
#Refactor
#Customer Subtype
df$MOSTYPE <- factor(df$MOSTYPE,levels=c(1:41),labels=c("High Income, expensive child","Very
#Average Age
df$MGEMLEEF <- factor(df$MGEMLEEF,levels=c(1:6),labels=c("20-30 years","30-40 years","40-50
#Custom Main Type
df$MOSHOOFD <- factor(df$MOSHOOFD,levels=(1:10),labels=c("Successful hedonists","Driven Grow
#Percentages
for (i in which(colnames(df)=="MGODRK"):which(colnames(df)=="MKOOPKLA")){
  df[,i] <- factor(df[,i],levels=c(0:9),labels=c("0%","1-10%","11-23%","24-36%","37-49%","50
}
#Number of
for (i in which(colnames(df)=="PWAPART"):which(colnames(df)=="ABYSTAND")){
  df[,i] <- factor(df[,i],levels=c(0:9),labels=c("0","1-49","50-99","100-199","200-499","500
}
#Set class label as factor
df$CARAVAN <- factor(df$CARAVAN,levels=c("0","1"))
```

I will now remove the column ORIGIN. The column origin is a factor with two values, TRAIN and TEST. It is the original set that the data came from in the challenge that this dataset was creaeted for. TRAIN data was given to contestants, and TEST was the data used to test the submitted models. As I am going to be resampling the data and partitioning my own train and test sets this column is useless so I will remove it.

```
#Get rid of ORIGIN
df$ORIGIN <- NULL
```

I will now remove any rows with missing values

```
#Remove NA's
df<-df[complete.cases(df),]
```

I will now resample the dataset to balance the distribution of the class label. I will do this using a function called ovun.sample from the ROSE library.

```
## Warning: package 'ROSE' was built under R version 3.4.3

## Loaded ROSE 0.0-3
```

```
library(ROSE)
```

```
#Resample Train(Oversampling)
df<-ovun.sample(CARAVAN~.,data=df,method="over")$data
```

2 Modelling/Classification

I will now create my random forest model. To do this I will be using the caret and randomForest R libraries.

```
## [1] 1e+05
```

```
## Warning:  package 'caret' was built under R version 3.4.2

## Loading required package:  lattice

## Warning:  package 'randomForest' was built under R version 3.4.2

## randomForest 4.6-12

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package:  'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(caret)
library(randomForest)
```

First I will create a train and test set.

```
part<-createDataPartition(y=df$CARAVAN,p=0.7,list=FALSE)
train<-df[part,]
test<-df[-part,]
```

Now I will build my model using the randomForest library. I will create a function to do this, that will also return error rates and accuracies.

```
buildModel<-function(trainData,testData,ntrees=100,nodeSize=1){
  #build random forest model
  model<-randomForest(trainData[,-ncol(trainData)],trainData[,ncol(trainData)],xtest=testDat

  #Print results
  print("TRAIN")
  #Train OOB Error
  print(paste("Train OOB Error: ",model$err.rate[nrow(model$test$err.rate),1,drop=FALSE],sep
  #Train Factor Level 0 Error
  print(paste("Train CARAVAN=0 Error: ",model$err.rate[nrow(model$test$err.rate),2,drop=FALS
  #Train Factor Level 1 Error
  print(paste("Train CARAVAN=1 Error: ",model$err.rate[nrow(model$test$err.rate),3,drop=FALS
  #Train Accuracy
  trainAuc<-sum(diag(model$confusion))/nrow(trainData)
  print(paste("Train Accuracy: ",trainAuc,"%",sep=""))

  #Print blank line between train and test results
  print(" ")

  print("TEST")
  #Test Error
  print(paste("Test Error: ",model$test$err.rate[nrow(model$test$err.rate),1,drop=FALSE],sep
  #Train Factor Level 0 Error
  print(paste("Test CARAVAN=0 Error: ",model$test$err.rate[nrow(model$test$err.rate),2,drop=
  #Train Factor Level 1 Error
  print(paste("Test CARAVAN=1 Error: ",model$test$err.rate[nrow(model$test$err.rate),3,drop=
  #Test Accuracy
  testAuc<-sum(diag(model$test$confusion))/nrow(testData)
  print(paste("Test Accuracy: ",testAuc,sep=""))
  return(model)
}
model<-buildModel(train,test)
```

```
## [1] "TRAIN"
## [1] "Train OOB Error: 0.362205334987593"
## [1] "Train CARAVAN=0 Error: 0.00850734725444702"
## [1] "Train CARAVAN=1 Error: 0.717773285647644"
## [1] "Train Accuracy: 0.637794665012407%"
## [1] " "
## [1] "TEST"
## [1] "Test Error: 0.382555193630112"
## [1] "Test CARAVAN=0 Error: 0.00866425992779783"
## [1] "Test CARAVAN=1 Error: 0.758345428156749"
## [1] "Test Accuracy: 0.617444806369888"
```

Now I will take a look at the error rates and accuracies of my model. During initial testing, accuracies where around 55-57 range. This isn't great and could be improved. Mode data where CARAVAN=TRUE is really needed. Perhaps different resampling methods like bootstraping might generate better results

I have written a function validate the model. I will use a 10 fold cross validation method.

```r
#Function to perform 10 fold cross validation
validate <- function(data,ntrees=100,nodeSize=1){
  #Frame to hold results
  results<-data.frame(OOB=as.numeric(),trainFalseError=as.numeric(),trainTrueError=as.numeri
  #Folds generated using Caret packages createFolds
  folds<-createFolds(data$CARAVAN,k=10,list=TRUE,returnTrain=FALSE)
  for (i in 1:10){
    #Keep one set for testing, rest training
    trainData<-data[-c(folds[[i]]),]
    testData<-data[c(folds[[i]]),]
    model<-randomForest(trainData[,-ncol(trainData)],trainData[,ncol(trainData)],xtest=testD
    #TRAIN
    oob<-model$err.rate[nrow(model$test$err.rate),1,drop=FALSE]
    trainFalseError<-model$err.rate[nrow(model$test$err.rate),2,drop=FALSE]
    trainTrueError<-model$err.rate[nrow(model$test$err.rate),3,drop=FALSE]
    trainAccuracy<-sum(diag(model$confusion))/nrow(trainData)
    #TEST
    testError<-model$test$err.rate[nrow(model$test$err.rate),1,drop=FALSE]
    testFalseError<-model$test$err.rate[nrow(model$test$err.rate),2,drop=FALSE]
    testTrueError<-model$test$err.rate[nrow(model$test$err.rate),3,drop=FALSE]
    testAccuracy<-sum(diag(model$test$confusion))/nrow(testData)
    #Create new Row in results with values
    results<-rbind(results,data.frame(OOB=oob,trainFalseError=trainFalseError,trainTrueError
  }
  #Display results
  #TRAIN
  print("TRAIN")
  #OOB
  print(results$OOB)
  print(paste("Average OOB: ",sum(results$OOB)/nrow(results),sep=""))
  #Train CARAVAN=0 Error
  print(results$trainFalseError)
  print(paste("Average CARAVAN=0 Error: ",sum(results$trainFalseError)/nrow(results),sep="")
  #Train Caravan=1 Error
  print(results$trainTrueError)
  print(paste("Average CARAVAN=1 Error: ",sum(results$trainTrueError)/nrow(results),sep=""))
  #Train Accuracy
  print(results$trainAccuracy)
  print(paste("Average Train Accuracy: ",sum(results$trainAccuracy)/nrow(results),sep=""))

  #Print blank line between train and test results
  print(" ")

  #TEST
  print("TEST")
  #Test Error
  print(results$testError)
  print(paste("Average Test Error: ",sum(results$testError)/nrow(results),sep=""))
  #Test CARAVAN=0 Error
  print(results$testFalseError)
  print(paste("Average CARAVAN=0 Error: ",sum(results$testFalseError)/nrow(results),sep=""))
  #Test CARAVAN=1 Error
  print(results$testTrueError)
  print(paste("Average CARAVAN=1 Error: ",sum(results$testTrueError)/nrow(results),sep=""))
  #Test Accuracy
  print(results$testAccuracy)
  print(paste("Average Test Accuracy: ",sum(results$testAccuracy)/nrow(results),sep=""))

  #Return results
  return(results)
}
```

I will now use it to validate my model

```r
validateResult<-validate(df);
```

```
## [1] "TRAIN"
##  [1] 0.4152241 0.4274427 0.4173452 0.4298293 0.4243667 0.4385669 0.4085289
##  [8] 0.4095543 0.4212050 0.4286748
## [1] "Average OOB: 0.422073776655107"
## NULL
## [1] "Average CARAVAN=0 Error: 0"
## NULL
## [1] "Average CARAVAN=1 Error: 0"
##  [1] 0.5847759 0.5725573 0.5826548 0.5701707 0.5756333 0.5614331 0.5914711
##  [8] 0.5904457 0.5787950 0.5713252
## [1] "Average Train Accuracy: 0.577926223344893"
## [1] " "
## [1] "TEST"
## NULL
## [1] "Average Test Error: 0"
## NULL
## [1] "Average CARAVAN=0 Error: 0"
## NULL
## [1] "Average CARAVAN=1 Error: 0"
##  [1] 0.5632122 0.5488599 0.5621945 0.5529603 0.5537459 0.5425936 0.5708085
##  [8] 0.5680955 0.5583922 0.5648399
## [1] "Average Test Accuracy: 0.55857024958404"
```

3 Imporving Performance

I will start by tring to fine tune the ntrees attribute of my model by testing my model with values between 100 and 1000 for ntree.

```r
#Using same train and test set as before
#Tweak number of trees
testNTrees <- function(trainData,testData){
  ntrees<-100
  results<-NULL
  results<-data.frame(NTrees=as.numeric(),OOB=as.numeric(),trainFalseError=as.numeric(),trai
  for (i in 1:10){
    trainData=train
    testData=test
    model<-randomForest(trainData[,-ncol(trainData)],trainData[,ncol(trainData)],xtest=testD
    #TRAIN
    oob<-model$err.rate[nrow(model$test$err.rate),1,drop=FALSE]
    trainFalse<-model$err.rate[nrow(model$test$err.rate),2,drop=FALSE]
    trainTrue<-model$err.rate[nrow(model$test$err.rate),3,drop=FALSE]
    trainAccuracy<-sum(diag(model$confusion))/nrow(trainData)
    #TEST
    testError<-model$test$err.rate[nrow(model$test$err.rate),1,drop=FALSE]
    testFalse<-model$test$err.rate[nrow(model$test$err.rate),2,drop=FALSE]
    testTrue<-model$test$err.rate[nrow(model$test$err.rate),3,drop=FALSE]
    testAccuracy<-sum(diag(model$test$confusion))/nrow(testData)
    #Create new row in results with new data
    results[nrow(results)+1,]<-c(ntrees,oob,trainFalse,trainTrue,testError,testFalse,testTru
    results
    ntrees <-ntrees + 100
  }
  #Display results
  results
  #TRAIN
  #OOB
  print(paste("Average OOB: ",sum(results$OOB)/nrow(results),sep=""))
  #Train CARAVAN=0 Error
  print(paste("Average CARAVAN=0 Error: ",sum(results$trainFalseError)/nrow(results),sep="")
  #Train Caravan=1 Error
  print(paste("Average CARAVAN=1 Error: ",sum(results$trainTrueError)/nrow(results),sep=""))
  #Train Accuracy
  print(paste("Average Train Accuracy: ",sum(results$trainAccuracy)/nrow(results),sep=""))

  #Print blank line between train and test results
  print(" ")

  #Test Error
  print(paste("Average Test Error: ",sum(results$testError)/nrow(results),sep=""))
  #Test CARAVAN=0 Error
  print(paste("Average CARAVAN=0 Error: ",sum(results$testFalseError)/nrow(results),sep=""))
  #Test CARAVAN=1 Error
  print(paste("Average CARAVAN=1 Error: ",sum(results$testTrueError)/nrow(results),sep=""))
  #Test Accuracy
  print(paste("Average Test Accuracy: ",sum(results$testAccuracy)/nrow(results),sep=""))

  #return max row
  ntrees<-results$NTrees[which.max(results$Accuracy)]
  return(ntrees)
}
#Get ntrees
ntrees<-testNTrees(train,test)
```

```
## [1] "Average OOB: 0.422549627791563"
## [1] "Average CARAVAN=0 Error: 0.00505800464037123"
## [1] "Average CARAVAN=1 Error: 0.84224848390608"
## [1] "Average Train Accuracy: 0.577450372208437"
## [1] " "
## [1] "Average Test Error: 0.428610206297503"
## [1] "Average CARAVAN=0 Error: 0.00631768953068592"
## [1] "Average CARAVAN=1 Error: 0.853047895500726"
## [1] "Average Test Accuracy: 0.571389793702497"
```

```
ntrees
```

```
## numeric(0)
```

I will not try and fine tune the randomForest function variable nodesize.

```
#Tweek Nodesize
testNodeSize <- function(trainData,testData){
  nsize<-0
  results<-data.frame(NTrees=as.numeric(),OOB=as.numeric(),trainFalseError=as.numeric(),trai
  for (i in 1:(nrow(trainData)/100)){
    model<-randomForest(trainData[,-ncol(trainData)],trainData[,ncol(trainData)],xtest=testD
    #TRAIN
    oob<-model$err.rate[nrow(model$test$err.rate),1,drop=FALSE]
    trainFalse<-model$err.rate[nrow(model$test$err.rate),2,drop=FALSE]
    trainTrue<-model$err.rate[nrow(model$test$err.rate),3,drop=FALSE]
    trainAccuracy<-sum(diag(model$confusion))/nrow(trainData)
    #TEST
    testError<-model$test$err.rate[nrow(model$test$err.rate),1,drop=FALSE]
    testFalse<-model$test$err.rate[nrow(model$test$err.rate),2,drop=FALSE]
    testTrue<-model$test$err.rate[nrow(model$test$err.rate),3,drop=FALSE]
    testAccuracy<-sum(diag(model$test$confusion))/nrow(testData)
    results[nrow(results)+1,]<-c(Nodesize=nsize,OOB=oob,trainFalseError=trainFalse,trainTrue
    nsize<-nsize+1
  }
  #Display results
  results
  #TRAIN
  #OOB
  print(paste("Average OOB: ",sum(results$OOB)/nrow(results),sep=""))
  #Train CARAVAN=0 Error
  print(paste("Average CARAVAN=0 Error: ",sum(results$trainFalseError)/nrow(results),sep="")
  #Train Caravan=1 Error
  print(paste("Average CARAVAN=1 Error: ",sum(results$trainTrueError)/nrow(results),sep=""))
  #Train Accuracy
  print(paste("Average Train Accuracy: ",sum(results$trainAccuracy)/nrow(results),sep=""))

  #Print blank line between train and test results
  print(" ")

  #Test Error
  print(paste("Average Test Error: ",sum(results$testError)/nrow(results),sep=""))
  #Test CARAVAN=0 Error
  print(paste("Average CARAVAN=0 Error: ",sum(results$testFalseError)/nrow(results),sep=""))
  #Test CARAVAN=1 Error
  print(paste("Average CARAVAN=1 Error: ",sum(results$testTrueError)/nrow(results),sep=""))
  #Test Accuracy
  print(paste("Average Test Accuracy: ",sum(results$testAccuracy)/nrow(results),sep=""))

  #Return node size
  nodeSize<-results$Nodesize[which.max(result$Accuracy)]
  return(nodeSize)
}
#Get node size
nodeSize<-testNodeSize(train,test)
```

```
## Error in double((nclass + 1) * ntree):  invalid 'length' argument
```

```
nodeSize
```

```
## Error in eval(expr, envir, enclos):  object 'nodeSize' not found
```

I will use the importance function, from the randomForest library, to create plots of mean decrease in accuracy and mean decrease in Gini

```
#Mean Decrease in accuracy
meanDecreaseAccuracy<-importance(model,type=1)
#Order highest to lowest
meanDecreaseAccuracy<-meanDecreaseAccuracy[order(-meanDecreaseAccuracy),,drop=FALSE]
```

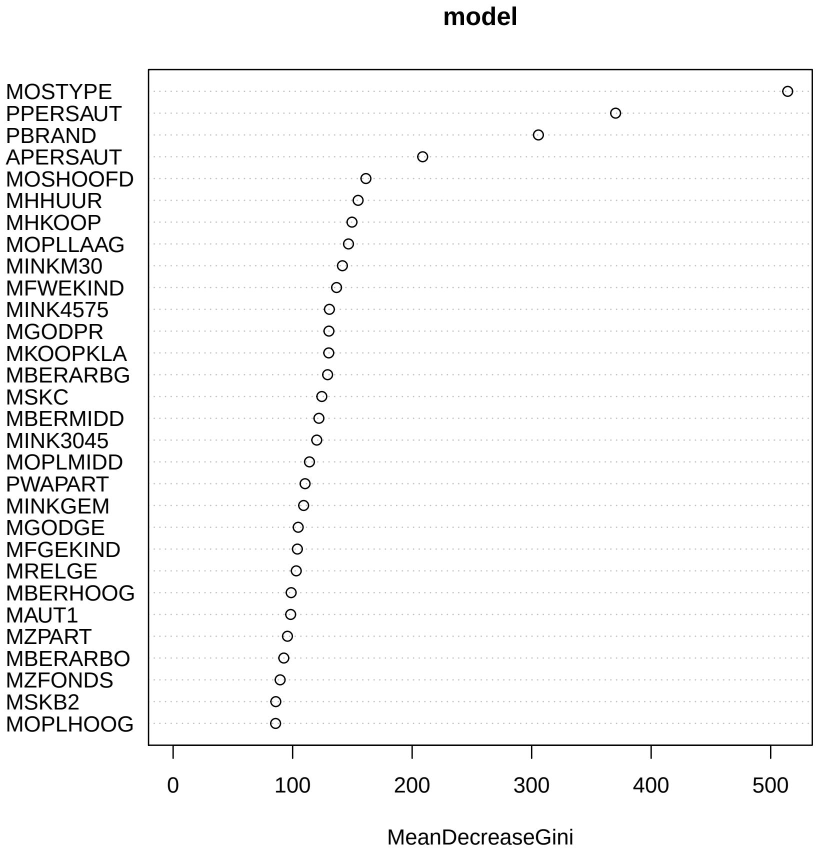
```
#Plot
varImpPlot(model,type=1)
```

```
## Error in imp[, i]:  subscript out of bounds
```

```
#Mean decrease in node impurity
meanDecreaseGini<-importance(model,type=2)
#Order highest to lowest
meanDecreaseGini<-meanDecreaseGini[order(-meanDecreaseGini),,drop=FALSE]
meanDecreaseGini
```

```
##          MeanDecreaseGini
## MOSTYPE      514.26800333
## PPERSAUT     370.26215024
## PBRAND       305.66971244
## APERSAUT     208.81773222
## MOSHOOFD     161.32473741
## MHHUUR       154.79469939
## MHKOOP       149.66084904
## MOPLLAAG     146.74084494
## MINKM30      141.66576562
## MFWEKIND     136.77010095
## MINK4575     130.76822521
## MGODPR       130.37139497
## MKOOPKLA     130.23277675
## MBERARBG     129.28026711
## MSKC         124.42695572
## MBERMIDD     122.00730519
## MINK3045     120.19269919
## MOPLMIDD     114.08357985
## PWAPART      110.41765838
## MINKGEM      109.21871514
## MGODGE       104.65557120
## MFGEKIND     103.97875152
## MRELGE       103.03417653
## MBERHOOG      98.78448596
## MAUT1         98.36829996
## MZPART        95.68301473
## MBERARBO      92.62159389
## MZFONDS       89.55541854
## MSKB2         85.92392806
## MOPLHOOG      85.74890114
## MSKA          84.47253577
## MRELOV        83.18051770
## MFALLEEN      81.49311923
## MAUT0         80.81203940
## ABRAND        73.23774627
## AWAPART       71.54612093
## MSKB1         69.92456095
## MBERBOER      53.55881001
## MAUT2         50.63775358
## MSKD          50.49648342
## MINK7512      49.70288136
## MGODOV        46.66511908
## PLEVEN        44.14741477
## MGODRK        42.81072563
## PBROM         34.60628618
## MRELSA        33.94497460
## MGEMLEEF      33.36079179
## MGEMOMV       32.52862786
## ALEVEN        31.53586151
## ABROM         29.35470172
## MBERZELF      26.81663983
## APLEZIER      21.84613140
## PMOTSCO       20.30707770
## MINK123M      19.60645067
## PPLEZIER      19.07050425
## AFIETS        16.03789983
## PBYSTAND      15.58267446
## AMOTSCO       15.37859781
## PFIETS        14.23052072
## MAANTHUI      12.29398277
## ABYSTAND      11.90435419
## ATRACTOR       8.24869617
## PTRACTOR       8.07572433
## PBESAUT        6.03866473
## PINBOED        6.00321450
## PWABEDR        5.95243114
## PGEZONG        5.92276444
## AWALAND        5.81081551
## PWALAND        5.61343505
## AINBOED        5.03817651
## AGEZONG        4.91896252
## ABESAUT        4.88233957
## PAANHANG       4.73417231
## AAANHANG       4.68527866
## AWABEDR        4.61539748
## AWAOREG        3.52134592
## PWAOREG        2.96717627
## PPERSONG       2.46006079
## APERSONG       2.43607783
## PZEILPL        1.84359286
## AZEILPL        1.15680138
## AWERKT         0.21646573
## PWERKT         0.17357383
## AVRAAUT        0.01080517
## PVRAAUT        0.00405516
```

```
#Plot
varImpPlot(model,type=2)
```



I will remove any columns that have a negative mean decrease in accuracy value(if any).

```
#Get negative or 0 MDA
cols<-rownames(meanDecreaseAccuracy[meanDecreaseAccuracy<0,,drop=FALSE])
#Show cols being removed
cols
```

```
#Remove cols
if (!is.null(cols)){
  df<-df[,!(colnames(df) %in% cols)]
}
```

I will now re-test the accuracy of my model

```
model2<-buildModel(train,test,ntrees=ntrees,nodeSize=nodeSize)
```

```
## Error in randomForest.default(trainData[, -ncol(trainData)], trainData[,
:  object 'nodeSize' not found
```

I will now validate the model again using the function I created ealier

```
validateResults2<-validate(df,ntrees,nodeSize)
```

```
## Error in randomForest.default(trainData[, -ncol(trainData)], trainData[,
:  object 'nodeSize' not found
```

4 Conclusions

5 References

- http://ggplot2.org/
- https://cran.r-project.org/web/packages/ROSE/ROSE.pdf
- https://www.analyticsvidhya.com/blog/2016/03/practical-guide-deal-imbalanced-classification-problems/
- https://cran.r-project.org/web/packages/randomForest/randomForest.pdf
- http://topepo.github.io/caret/index.html