FIT5149 S1 2009 Assessment 1: Choosing and Explaining Likely Caravan Insurance Customers

Table of Contents

```r
library(caret)
library(data.table)
library(lattice)
library(ggplot2)
library(e1071)
library(ISLR)
library(dplyr)
library(tidyverse)
library(MASS)
library(shiny)
library(plotly)
library(webshot)
library(corrplot)
library(RColorBrewer)
library(MASS)
library(clusterGeneration)
library(glmnet)
library(ca)
library(car)
library(usdm)
library(ROCR)


#webshot::install_phantomjs()

#Reading the data

data=read.table("ticdata2000.txt")
testdata=read.table("ticeval2000.txt")
targetdata=read.table("tictgts2000.txt")
```

1. Data Exploration

1.1 Summary of Train Dataset

There are 5822 Customers in the dataset with 86 attributes associated to each of them.

1.2 Checking for Missing values

```r
paste0("Missing values in the Dataset is : ", sum(is.na(data)))

## [1] "Missing values in the Dataset is : 0"
```

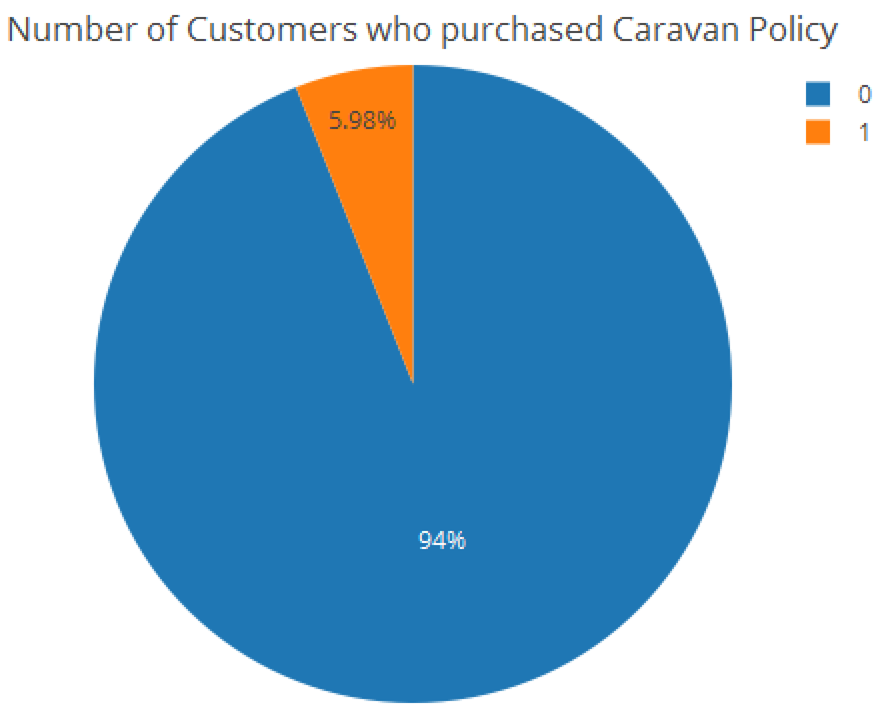
1.3 Exploring different variable types

```r
str(data)

## 'data.frame':    5822 obs. of  86 variables:
##  $ V1 : int  33 37 37 9 40 23 39 33 33 11 ...
##  $ V2 : int  1 1 1 1 1 1 2 1 1 2 ...
##  $ V3 : int  3 2 2 3 4 2 3 2 2 3 ...
##  $ V4 : int  2 2 2 3 2 1 2 3 4 3 ...
##  $ V5 : int  8 8 8 3 10 5 9 8 8 3 ...
##  $ V6 : int  0 1 0 2 1 0 2 0 0 3 ...
##  $ V7 : int  5 4 4 3 4 5 2 7 1 5 ...
##  $ V8 : int  1 1 2 2 1 0 0 0 3 0 ...
##  $ V9 : int  3 4 4 4 4 5 5 2 6 2 ...
##  $ V10: int  7 6 3 5 7 0 7 7 6 7 ...
##  $ V11: int  0 2 2 2 1 6 2 2 0 0 ...
##  $ V12: int  2 2 4 2 2 3 0 0 3 2 ...
##  $ V13: int  1 0 4 2 2 3 0 0 3 2 ...
##  $ V14: int  2 4 4 3 4 5 3 5 3 2 ...
##  $ V15: int  6 5 2 4 4 2 6 4 3 6 ...
##  $ V16: int  1 0 0 3 5 0 0 0 0 0 ...
##  $ V17: int  2 5 5 4 4 5 4 3 1 4 ...
##  $ V18: int  7 4 4 2 0 4 5 6 8 5 ...
##  $ V19: int  1 0 0 4 0 2 0 2 1 2 ...
##  $ V20: int  0 0 0 0 5 0 0 0 1 0 ...
##  $ V21: int  1 0 0 0 4 0 0 0 0 0 ...
##  $ V22: int  2 5 7 3 0 4 4 2 1 3 ...
##  $ V23: int  5 0 0 1 0 2 1 5 8 3 ...
##  $ V24: int  2 4 2 2 0 2 5 2 1 3 ...
##  $ V25: int  1 0 0 3 9 2 0 2 1 1 ...
##  $ V26: int  1 2 5 2 0 2 1 1 1 2 ...
##  $ V27: int  2 3 0 1 0 2 4 2 0 1 ...
##  $ V28: int  6 5 4 4 0 4 5 5 8 4 ...
##  $ V29: int  1 0 0 0 0 2 0 2 1 2 ...
##  $ V30: int  1 2 7 5 4 9 6 0 9 0 ...
##  $ V31: int  8 7 2 4 5 0 3 9 0 9 ...
##  $ V32: int  8 7 7 9 6 5 8 4 5 6 ...
##  $ V33: int  0 1 0 0 2 3 0 4 2 1 ...
##  $ V34: int  1 2 2 0 1 3 1 2 3 2 ...
##  $ V35: int  8 6 9 7 5 9 9 6 7 6 ...
##  $ V36: int  1 3 0 2 4 0 0 3 2 3 ...
##  $ V37: int  0 2 4 1 0 5 4 2 7 2 ...
##  $ V38: int  4 0 5 5 0 2 3 5 2 3 ...
##  $ V39: int  5 5 0 3 9 3 3 3 1 3 ...
##  $ V40: int  0 2 0 0 0 0 0 0 0 1 ...
##  $ V41: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V42: int  4 5 3 4 6 3 3 3 2 4 ...
##  $ V43: int  3 4 4 4 3 3 5 3 3 7 ...
##  $ V44: int  0 2 2 0 0 0 0 0 0 2 ...
##  $ V45: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V46: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V47: int  6 0 6 6 0 6 6 0 5 0 ...
##  $ V48: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V49: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V50: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V51: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V52: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V53: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V54: int  0 0 0 0 0 0 0 3 0 0 ...
##  $ V55: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V56: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V57: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V58: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V59: int  5 2 2 2 6 0 0 0 0 3 ...
##  $ V60: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V61: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V62: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V63: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V64: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V65: int  0 2 1 0 0 0 0 0 0 1 ...
##  $ V66: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V67: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V68: int  1 0 1 1 0 1 1 0 1 0 ...
##  $ V69: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V70: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V71: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V72: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V73: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V74: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V75: int  0 0 0 0 0 0 0 1 0 0 ...
##  $ V76: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V77: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V78: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V79: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V80: int  1 1 1 1 1 0 0 0 0 1 ...
##  $ V81: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V82: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V83: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V84: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V85: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V86: int  0 0 0 0 0 0 0 0 0 0 ...
```

We can see all the variables in the dataset are integers with different levels

1.4 Determining the ratio of Caravan Policy holders to Non caravan Policy Holders.

```
pie<-data.frame(table(data$V86))
 plot_ly(pie, labels = ~Var1, values = ~Freq, type = 'pie')%>%
  layout(title = 'Number of Customers who purchased Caravan Policy',
         xaxis = list(showgrid = FALSE, zeroline = FALSE, showticklabels =
FALSE),
         yaxis = list(showgrid = FALSE, zeroline = FALSE, showticklabels =
FALSE))
```

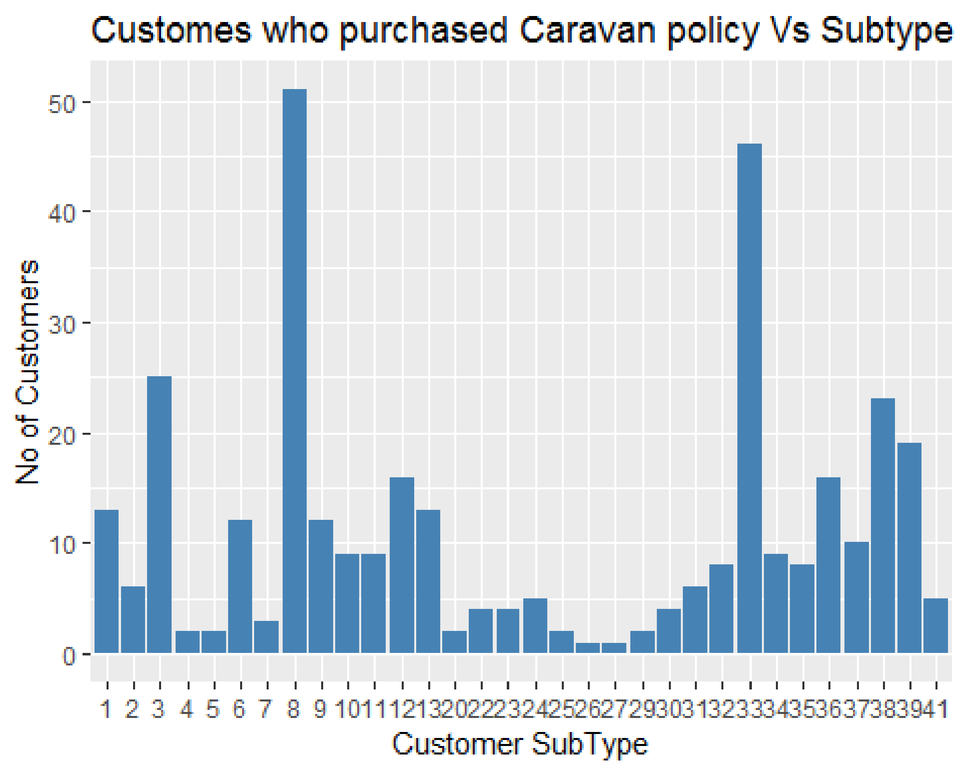


We can determine from the above pie chart that number of people who are purchasing Caravan Policies are less when compared to people who are not purchasing. Approximately 6% of the whole customers list buy the policy.

1.5 Determing the potential policy buyers based on Socio Demographic information provided.

1.5.1 Policy Buyers Vs Customer Subtype.

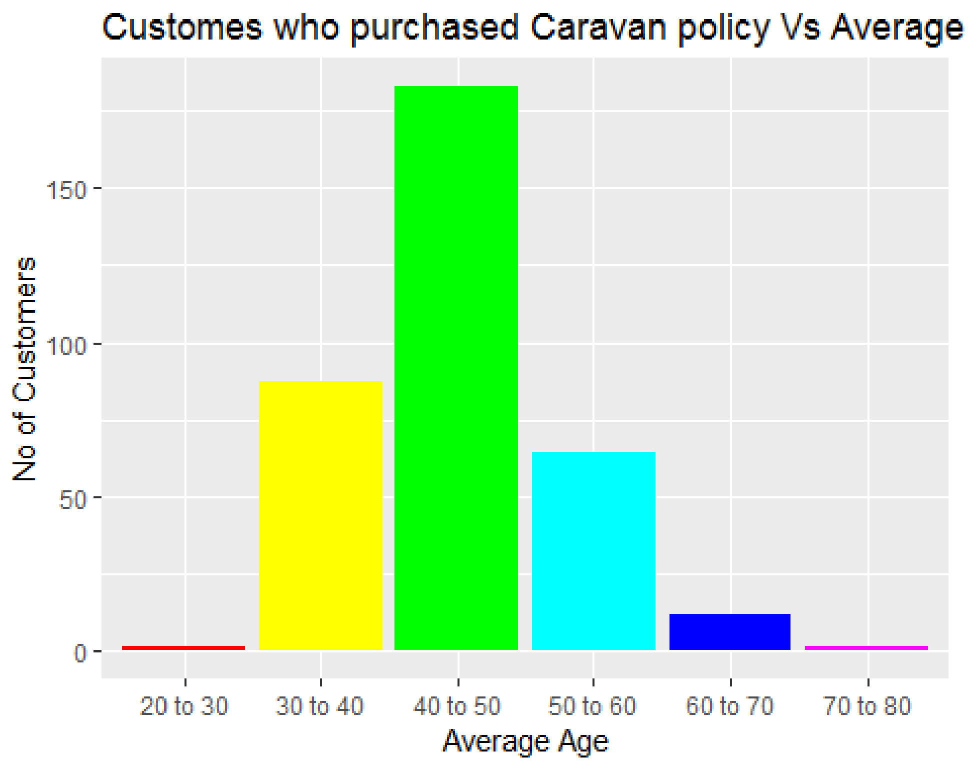
```
Csubtype=data.frame(table(data$V1[data$V86==1]))
ggplot(data=Csubtype,aes(x=Var1, y=Freq)) +
geom_bar(stat="identity",fill="steelblue")+xlab("Customer SubType")+ylab("No
of Customers")+
ggtitle("Customes who purchased Caravan policy Vs Subtype")
```



The above graph shows potential policy holders based on Customer Subtype where we can say that Middle Class families and lower class with large families are more likely to buy the policy.

1.5.2 Policy Buyers Vs Age group

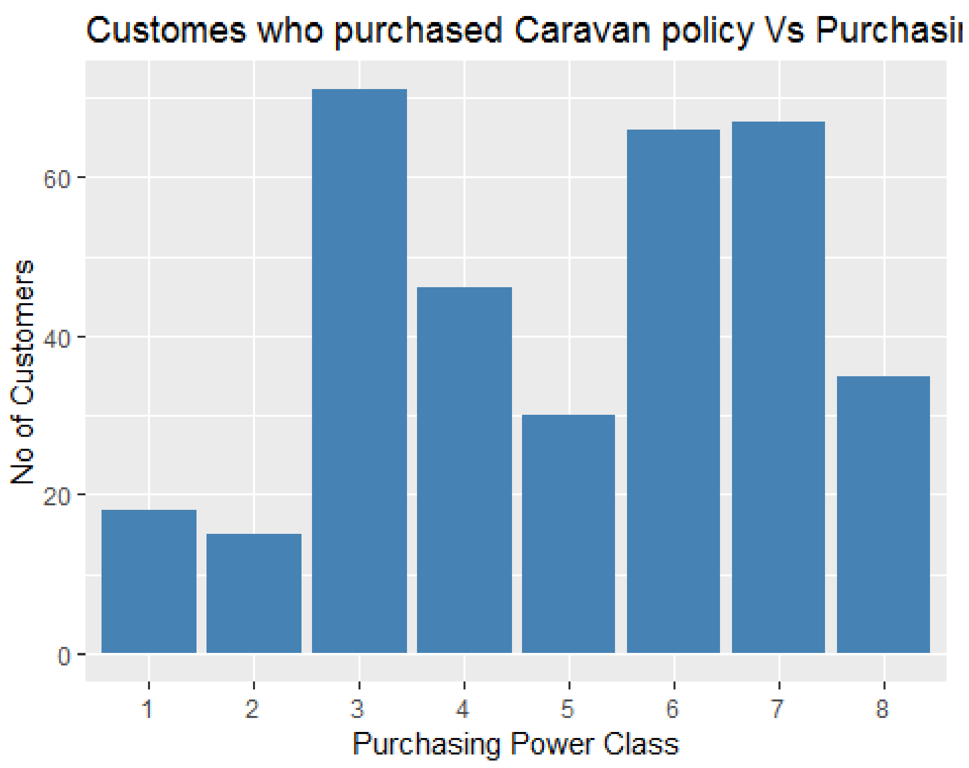
```
agegrp=table(data$V4[data$V86==1])
names(agegrp)=c("20 to 30","30 to 40","40 to 50","50 to 60","60 to 70","70 to 
80")
AverageAge=data.frame(agegrp)
ggplot(data=AverageAge,aes(x=Var1, y=Freq)) +
geom_bar(stat="identity",fill=rainbow(6) )+xlab("Average Age")+ylab("No of 
Customers")+
ggtitle("Customes who purchased Caravan policy Vs Average Age ")
```



The above graph shows that customers with the average age group 40 t0 50 are more likely to buy Policy.

1.5.3 Policy Buyers Vs Purchasing Power Class

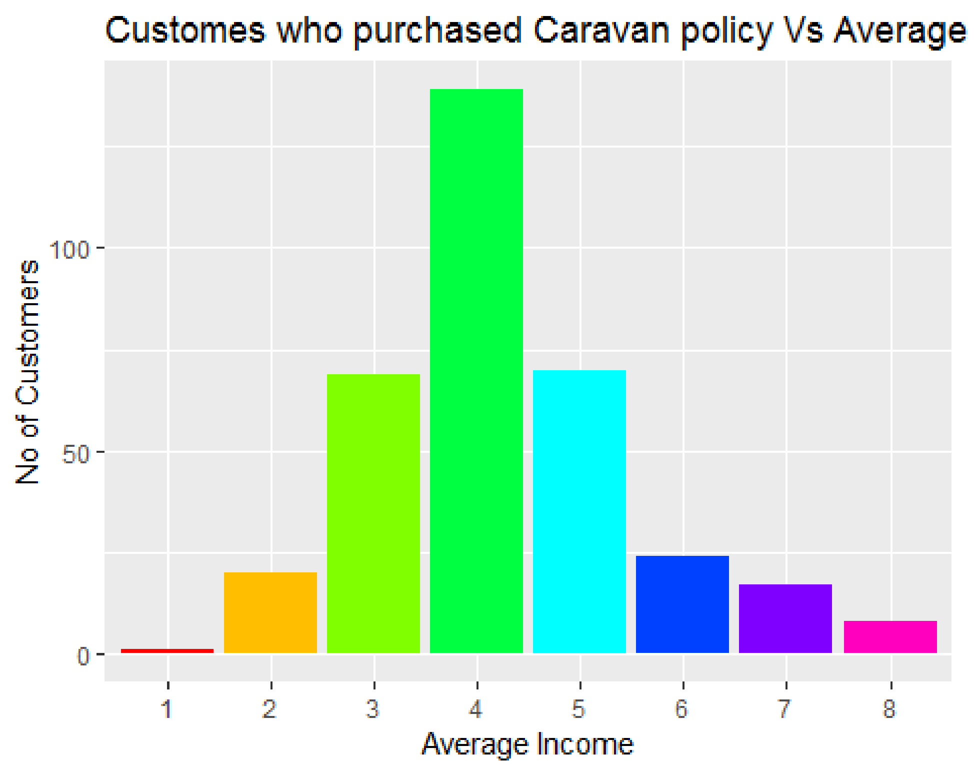
```
powerclass=data.frame(table(data$V43[data$V86==1]))
ggplot(data=powerclass,aes(x=Var1, y=Freq)) +
geom_bar(stat="identity",fill="steelblue")+xlab("Purchasing Power
Class")+ylab("No of Customers")+
ggtitle("Customes who purchased Caravan policy Vs Purchasing Power Class")
```



From The Above Graph we can observe that Customers who are of having purchaing power class (3) are more likely to purchase the Caravan policy. Marketing professional can also target people segment purchaing power class(6 and 7)

1.5.4 Policy Buyers Vs Average income of the Customers

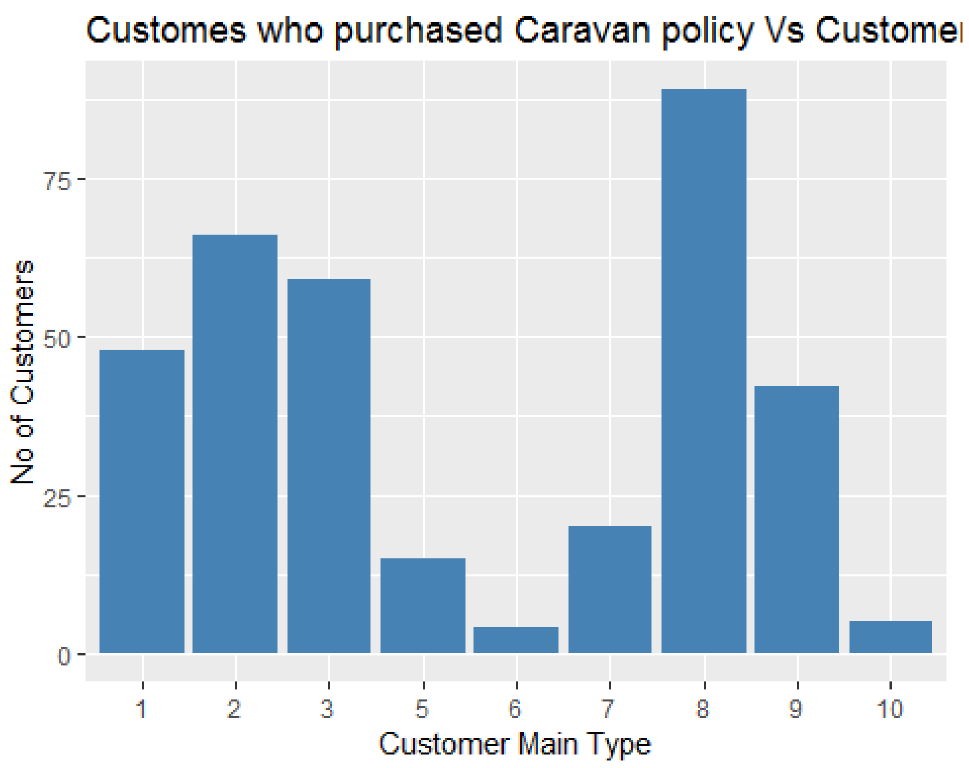
```
aveincome=data.frame(table(data$V42[data$V86==1]))
ggplot(data=aveincome,aes(x=Var1, y=Freq)) +
geom_bar(stat="identity",fill=rainbow(8))+xlab("Average Income")+ylab("No of
Customers")+
ggtitle("Customes who purchased Caravan policy Vs Average Income")
```



From the above Graph we can observe the average Income of the people between these ranges ($100–$199,$200–$499,$500–$999) are likely to buy the Policy where $200-$499 being the highest.

1.5.5 Customer Main Type

```
Customer_MainType=data.frame(table(data$V5[data$V86==1]))
ggplot(data=Customer_MainType,aes(x=Var1, y=Freq)) +
geom_bar(stat="identity",fill="steelblue")+xlab("Customer Main
Type")+ylab("No of Customers")+
ggtitle("Customes who purchased Caravan policy Vs Customer Main Type")
```

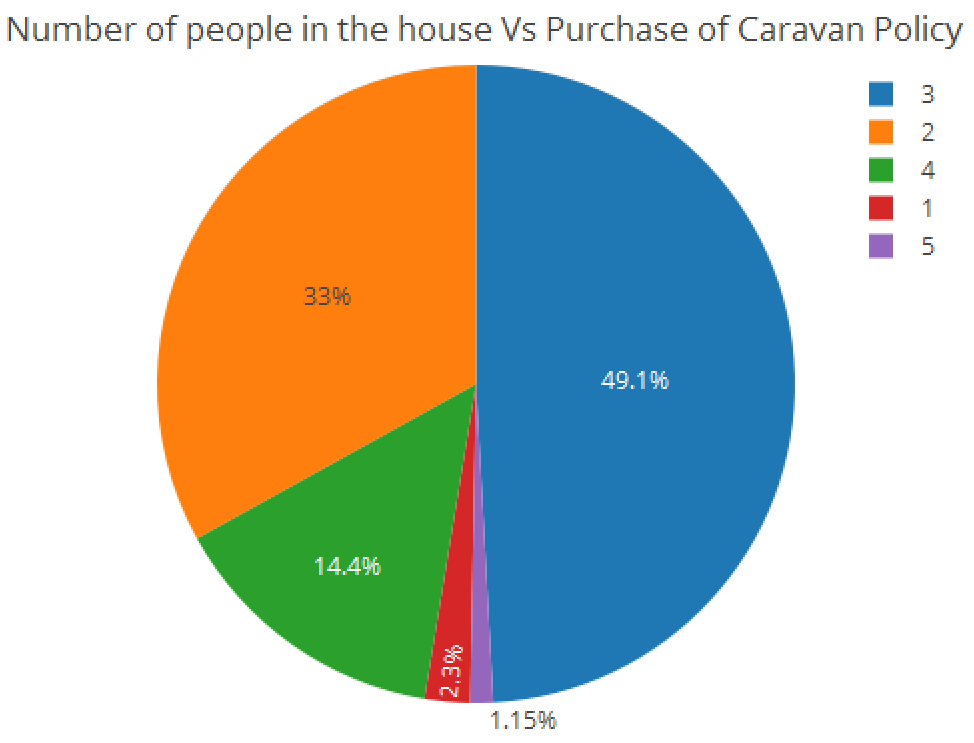


From the above bar plot is slightly clear that Family with grown-ups(8) main type are likely to buy caravan policies than others.

1.6 Determing the potential policy buyers based on Policy ownership information provided.

1.6.1 Policy Buyers Vs Household Members

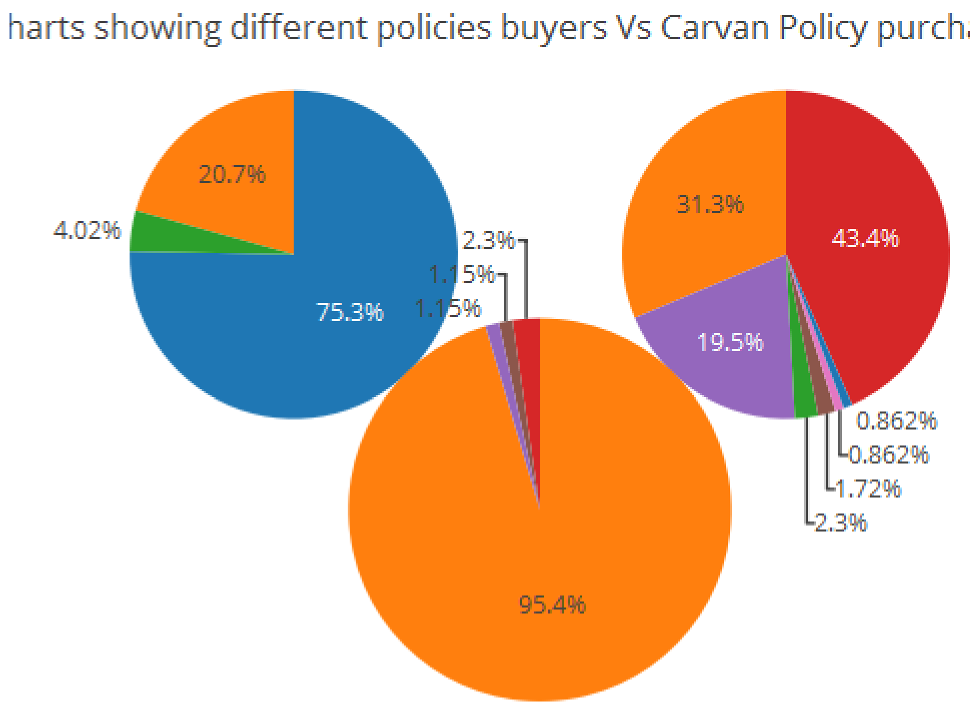
```
avgsize=data.frame(table(data$V3[data$V86==1]))
p <- plot_ly(avgsize, labels = ~Var1, values = ~Freq, type = 'pie') %>%
  layout(title = 'Number of people in the house Vs Purchase of Caravan
Policy',
         xaxis = list(showgrid = FALSE, zeroline = FALSE, showticklabels =
FALSE),
         yaxis = list(showgrid = FALSE, zeroline = FALSE, showticklabels =
FALSE))
p
```



From the above Pie chart we can observe that average sizes of 3 are likely to buy when compared to the others.

1.6.2 Different Policy holders Vs Caravan Policy Holders

```
carpolicy=data.frame(table(data$V47[data$V86==1]))
firepolicy=data.frame(table(data$V59[data$V86==1]))
SocialSecurity=data.frame(table(data$V64[data$V86==1]))
TrailerPolicy=data.frame(table(data$V72[data$V86==1]))

p <- plot_ly() %>%
  add_pie(data = carpolicy, labels = ~Var1, values = ~Freq,
          name = "Car Policy", domain = list(x = c(0, 0.4), y = c(0.4, 1)))
%>%
  add_pie(data = firepolicy, labels = ~Var1, values = ~Freq,
          name = "Fire Policy ", domain = list(x = c(0.6, 1), y = c(0.4, 1)))
%>%
  add_pie(data = SocialSecurity, labels = ~Var1, values = ~Freq,
          name = "Social Security Policy", domain = list(x = c(0.25, 0.75), y
= c(0, 0.6))) %>%
  layout(title = "Pie Charts showing different policies buyers Vs Carvan
Policy purchasers ",
         showlegend = F,
         xaxis = list(showgrid = FALSE, zeroline = FALSE, showticklabels =
FALSE),
         yaxis = list(showgrid = FALSE, zeroline = FALSE, showticklabels =
FALSE))

p
```



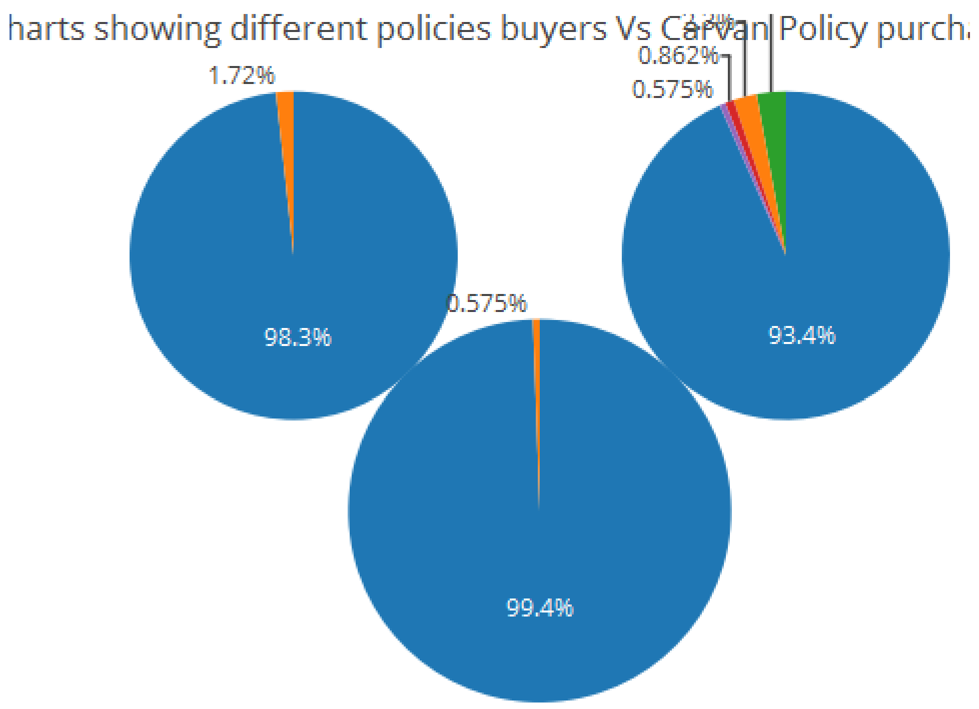
1. The Chart on the left shows 75.3% of the Customers who buy Car policy with 6 levels would buy Carvan policy.

2. The chart on the right displays 43% of the customers who opt fire polies with level 4 would buy caravan Policy

3. The Pie in between shows as customers who buy no social securities (0) are tending to buy caravan policies.

```
TrailerPolicy=data.frame(table(data$V72[data$V86==1]))
life=data.frame(table(data$V76[data$V86==1]))
van=data.frame(table(data$V69[data$V86==1]))
```

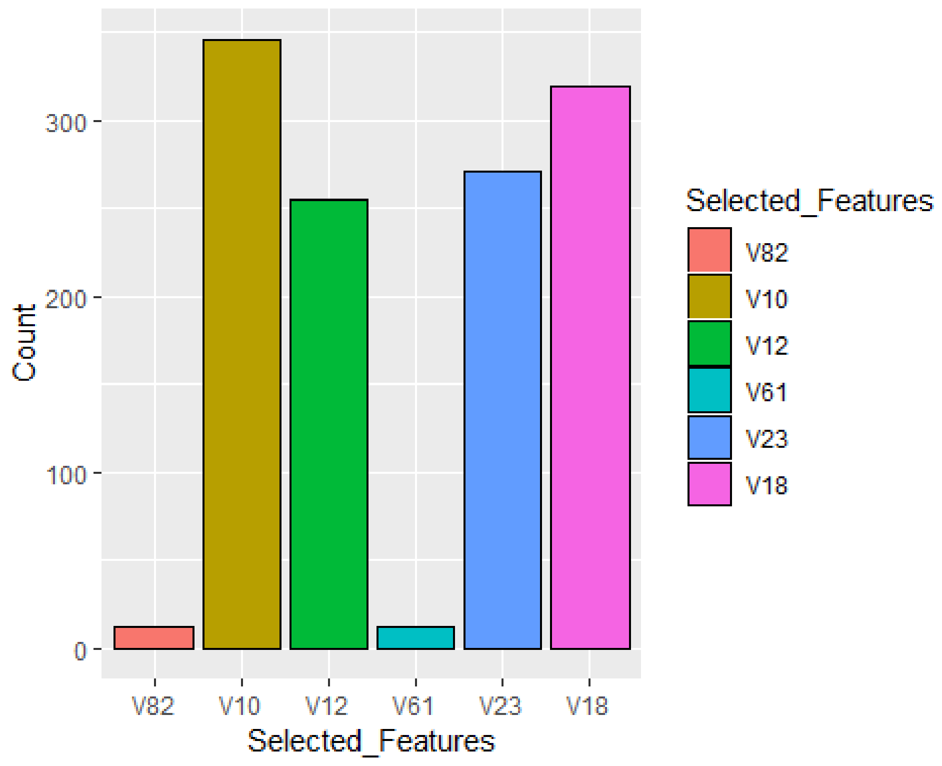
```
p <- plot_ly() %>%
  add_pie(data = TrailerPolicy, labels = ~Var1, values = ~Freq,
          name = "Trailer Policy", domain = list(x = c(0, 0.4), y = c(0.4,
1))) %>%
  add_pie(data = life, labels = ~Var1, values = ~Freq,
          name = "Life Insurance Policy ", domain = list(x = c(0.6, 1), y =
c(0.4, 1))) %>%
  add_pie(data = van, labels = ~Var1, values = ~Freq,
          name = "Van Delivery Policy", domain = list(x = c(0.25, 0.75), y =
c(0, 0.6))) %>%

  layout(title = "Pie Charts showing different policies buyers Vs Carvan
Policy purchasers ",
         showlegend = F,
         xaxis = list(showgrid = FALSE, zeroline = FALSE, showticklabels =
FALSE),
         yaxis = list(showgrid = FALSE, zeroline = FALSE, showticklabels =
FALSE))

p
```



From the above pie charts we can say that Trailer (Left), Life insurance (right),Van delivery(middle) Policies are inversly related to Caravan Policy as Customers who dont buy these policies will tend to .buy Caravan Policy.

1.6.3 Few Selected Vs Caravan Policy Holders

```
No_of_boat_policies <- sum(data$V86 == 1 & data$V82 != 0)
Married <- sum(data$V86 == 1 & data$V10  != 0)
Other_relation <- sum(data$V86 == 1 & data$V12  != 0)
boat_policies <- sum(data$V86 == 1 & data$V61  != 0)
Skilled_labourers <- sum(data$V86 == 1 & data$V23  != 0)
Lower_level_education <- sum(data$V86 == 1 & data$V18  != 0)


dat <- data.frame(
  Selected_Features = factor(c("V82","V10" , "V12" , "V61","V23", "V18" ),
levels=c("V82","V10" , "V12" , "V61","V23", "V18")),
  Count = c(No_of_boat_policies, Married , Other_relation , boat_policies ,
Skilled_labourers, Lower_level_education)
)

ggplot(data=dat, aes(x=Selected_Features, y=Count, fill=Selected_Features)) +
  geom_bar(colour="black", stat="identity")
```



From the above graph it is partially clear that married people and low level education may buy Caravan policy.

2 Prediction Task (Model Methodology and Development)

2.1 Desired Variable Identification

We are interested in looking at the model which has minimum number of predictors(reduced complexity) and best accuracy. Hence choosing the predictors would be first step in developing a model.

Data provided has both Nominal and Ordinal Variables. Hence identifying them would be initial step in the process of elimination of variables.

Looking at the Data dictionary we try to understand each variable's description, significance and distinct levels provided. Accordingly if the levels given have some kind of order we categorise it as Ordinal else nominal.

The below are list of the variables which I think should be Nominal

1. Customer Subtype
2. Customer main type
3. Protestant
4. Other religion
5. No religion
6. Married
7. Living together
8. Other relation
9. Singles
10. 1 car
11. 2 car
12. No car
13. Purchasing Power Class
14. Caravan Policy

```r
# Converting the above Listed variables to factors
data[,'V86']<-factor(data[,'V86'])
data[,'V1']<-factor(data[,'V1'])
data[,'V5']<-factor(data[,'V5'])
data[,'V7']<-factor(data[,'V7'])
data[,'V8']<-factor(data[,'V8'])
data[,'V9']<-factor(data[,'V9'])
data[,'V10']<-factor(data[,'V10'])
data[,'V11']<-factor(data[,'V11'])
data[,'V12']<-factor(data[,'V12'])
data[,'V13']<-factor(data[,'V13'])
data[,'V43']<-factor(data[,'V43'])
data[,'V32']<-factor(data[,'V32'])
data[,'V33']<-factor(data[,'V33'])
data[,'V34']<-factor(data[,'V34'])
```

```
var=c('V1','V5','V7','V8','V9','V10','V11','V12','V13','V43','V32','V33','V34
','V86')
var1=c('V1','V5','V7','V8','V9','V10','V11','V12','V13','V43','V32','V33','V3
4')

# Creating a separate Dataset for Nominal Variables
data_sub_nominal=data[,c(var)]
# Creating a data set for Ordinal Variables
data[ ,c(var1)] <- list(NULL)
```

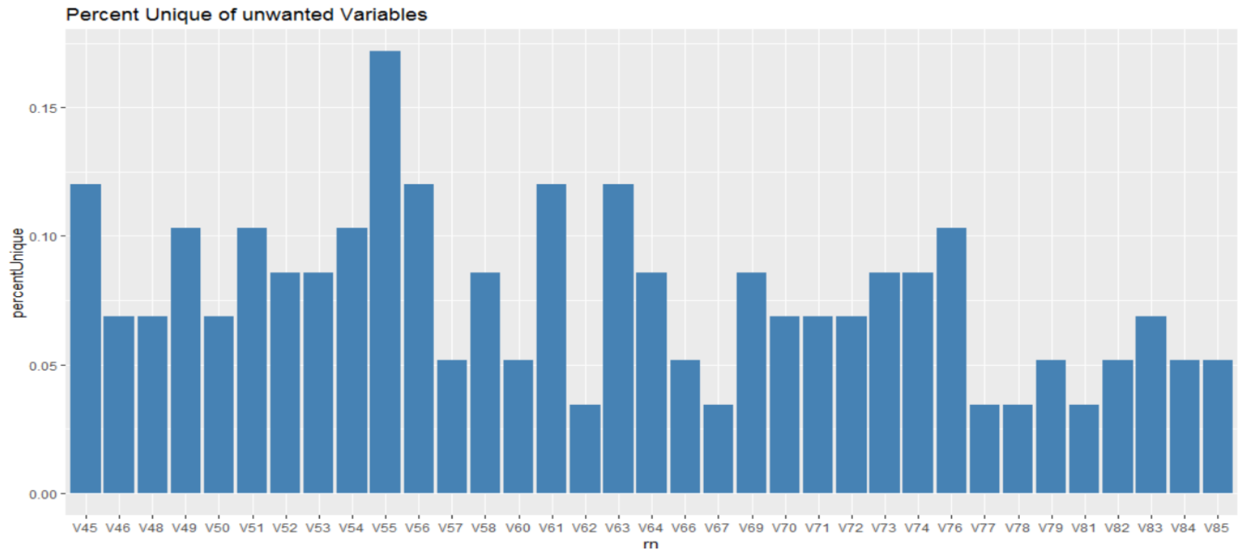
2.2 Oridinal Variables Reduction

2.2.1 Identification of near zero variance predictors for Oridinal Variables

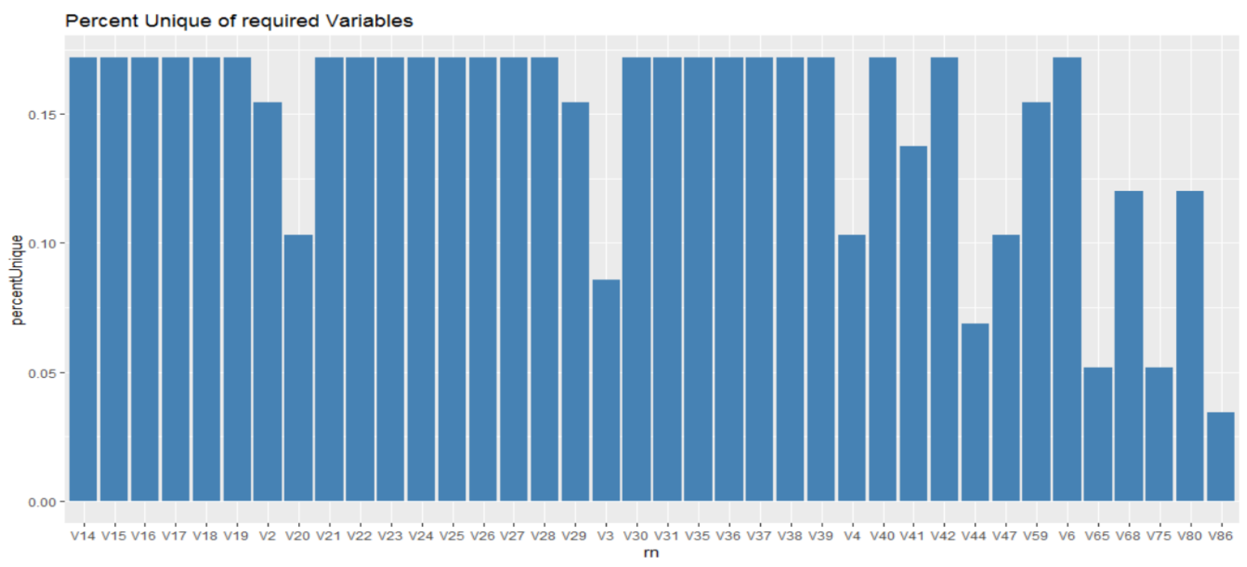
Let's check the near zero variance predictors, which have the following two characteristics 1. They have very few unique values relative to the number of samples. 2. The ratio of the frequency of the most common value to the frequency of the second most common value is large.

This kind of predictor is non-informative, it can break some models which you may want to fit to your data. Hence we need to remove them.

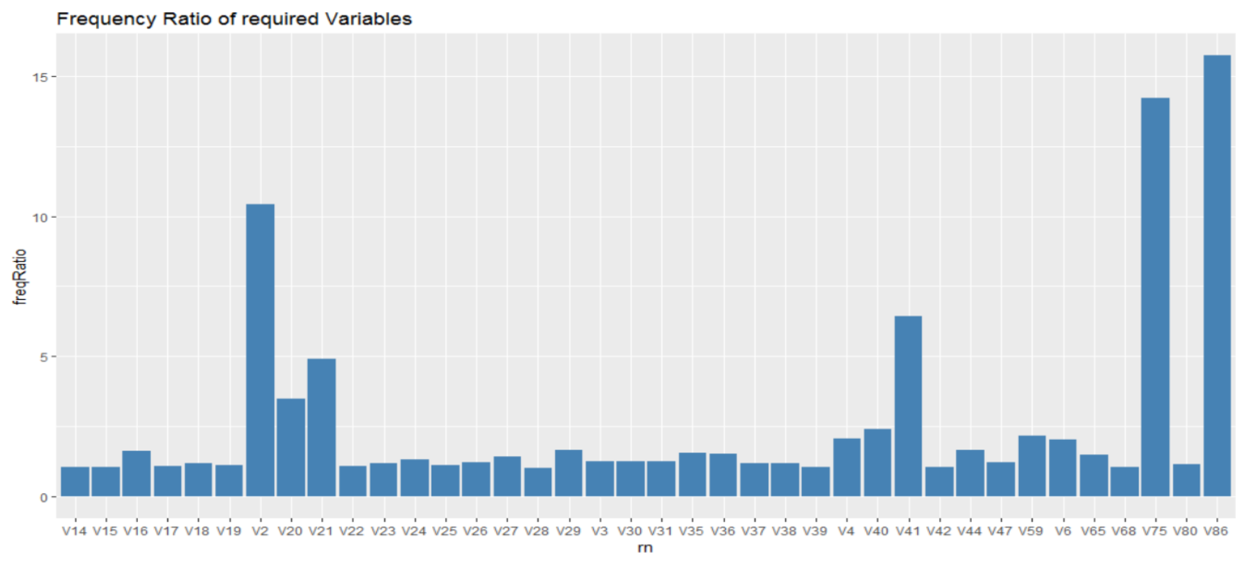
```
nzv <- nearZeroVar(data, saveMetrics = TRUE)
tmp=nzv[nzv$nzv==FALSE,] # temporary variable to store required variables
setDT(tmp,keep.rownames = TRUE)
tmp1=nzv[nzv$nzv==TRUE,] # temporary variable to store unwanted variables
setDT(tmp1,keep.rownames = TRUE)

ggplot(data=tmp1,aes(x=rn, y=percentUnique))+
  geom_bar(stat="identity",fill="steelblue")+ ggtitle("Percent Unique of
unwanted Variables ")
```



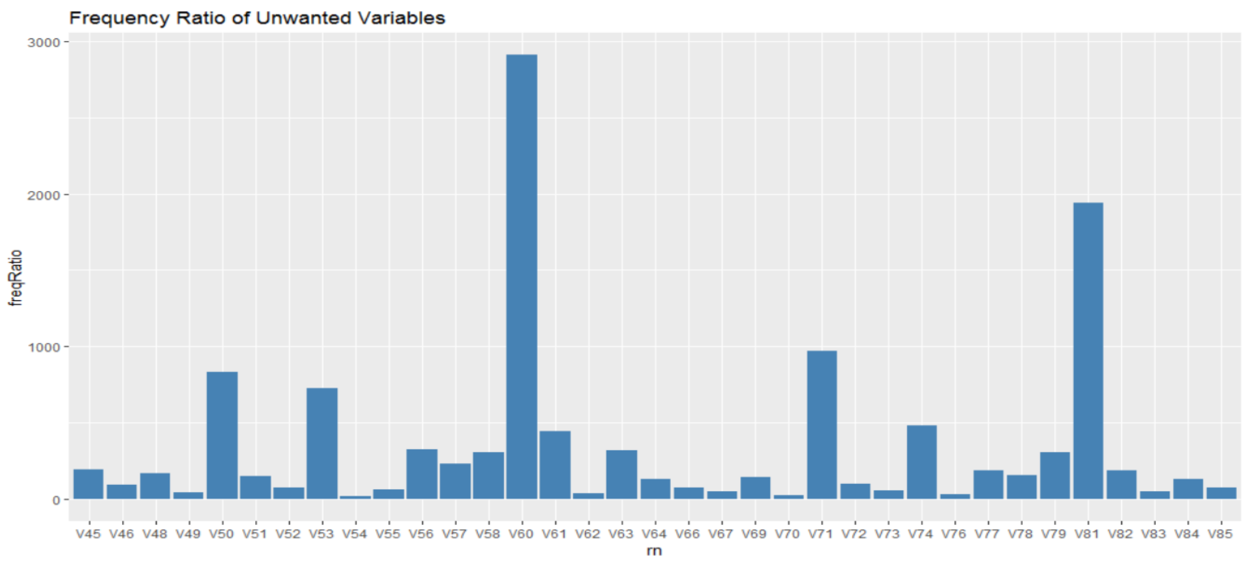
```r
ggplot(data=tmp,aes(x=rn, y=percentUnique))+
  geom_bar(stat="identity",fill="steelblue")+ ggtitle("Percent Unique of
required Variables ")
```



```r
ggplot(data=tmp,aes(x=rn, y=freqRatio))+
  geom_bar(stat="identity",fill="steelblue")+ ggtitle("Frequency Ratio of
required Variables ")
```



```r
ggplot(data=tmp1,aes(x=rn, y=freqRatio))+
  geom_bar(stat="identity",fill="steelblue")+ ggtitle("Frequency Ratio of
Unwanted Variables ")
```



The above four Graphs shows the output of NZV function with the required variables for the model and variables which has to thrown out. The nearZerovariance() has been set to default cutoffs with frequency cutoff as 95/5 and UniqueCut to be 10.

Chart 1 and 2 Displays uniqueness in the variables where as chart 3 and 4 shows frequency Ratio that is the ratio of frequencies for the most common value over the second most common value.

We can also see there are no zerovariance Variables,however there are near zero variance.

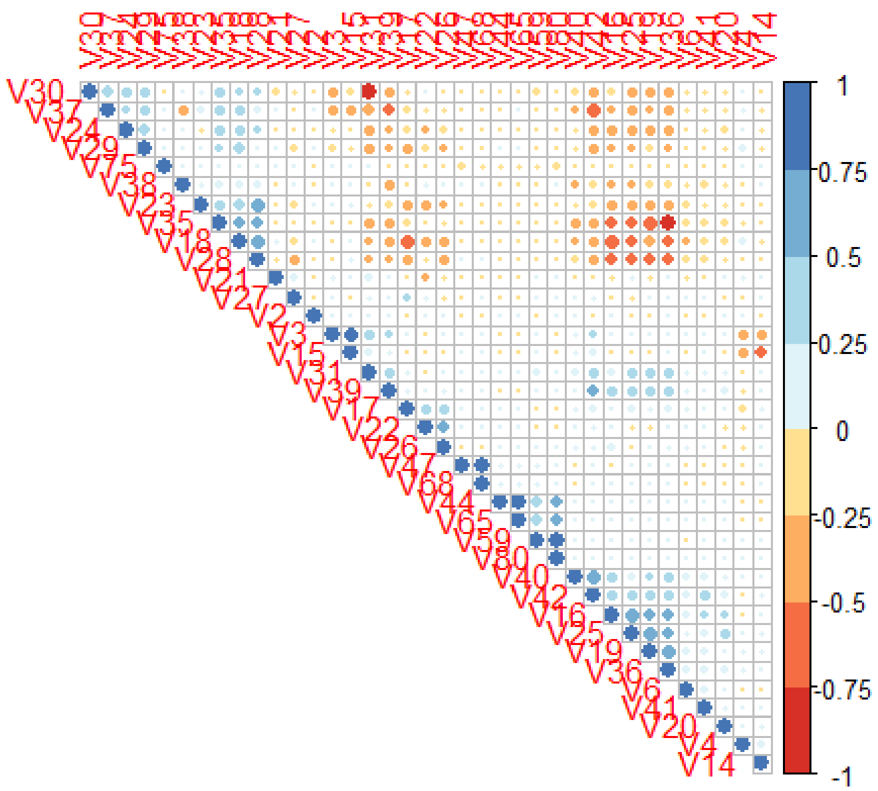
```
# Considering Only the required variables as per nearzerovariacce() function
data_subset=data[,unlist(tmp[,1])]
```

2.2.2 Determining the inter-correlation between variables(Collinearity).

These inter variable interations degrade the model performance. Hence We need to remove one of the variables.

Lets us look into correaltions visually

```
correlation=cor(data_subset[,-length(unlist(tmp[,1]))])
corrplot(correlation, type="upper", order="hclust",
         col=brewer.pal(n=8, name="RdYlBu"))
```



From the above Correlation plot we can observe that there are attributes which are inter-correlated among themselves. Hence keeping these correlated attributes in the model may affect prediction results.

findCorrelation() searches through a correlation matrix and returns a vector of integers corresponding to columns to remove to reduce pair-wise correlations.

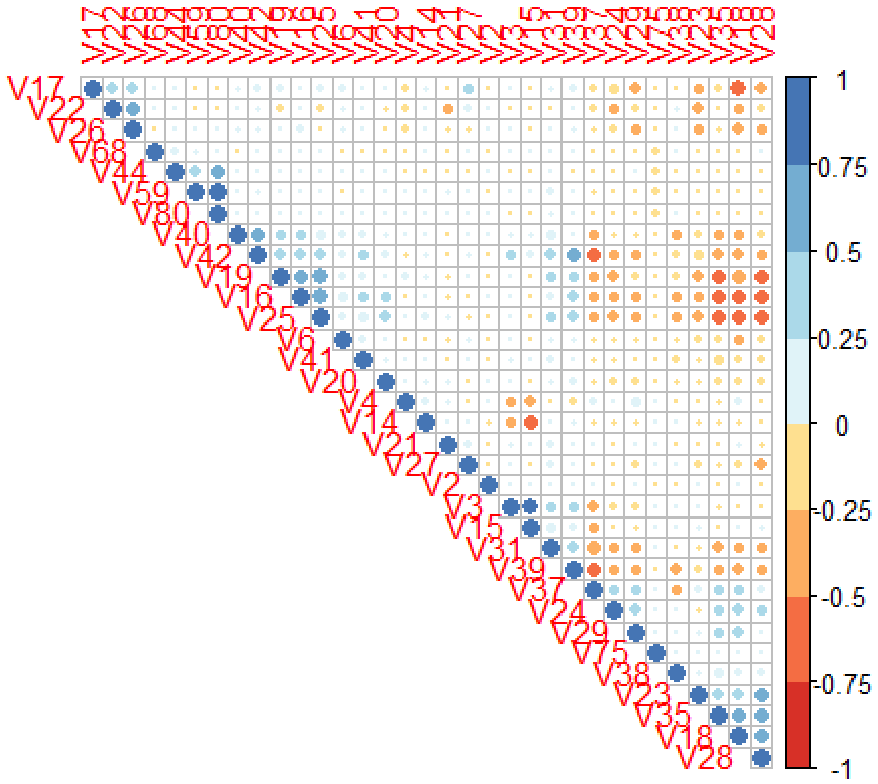
```
#Identifying correlated variables and removing them.
highCor <- findCorrelation(correlation, cutoff = 0.9)# Setting Cutoff to 0.75
as any pair-wise correlation above 0.75 is considered high and may break the
model

data=data_subset[,-highCor] #removing them
```

The highly correlated variables in the dataset are 24, 21, 34, 32.

The Total number of attributes remaining after removal of Near Zero variance predictors and highCorrelated Variables is 34

Lets Plot Correlation plot again Visualize the relations.

```
corrplot(cor(data[-ncol(data)]), type="upper", order="hclust",
         col=brewer.pal(n=8, name="RdYlBu"))
```

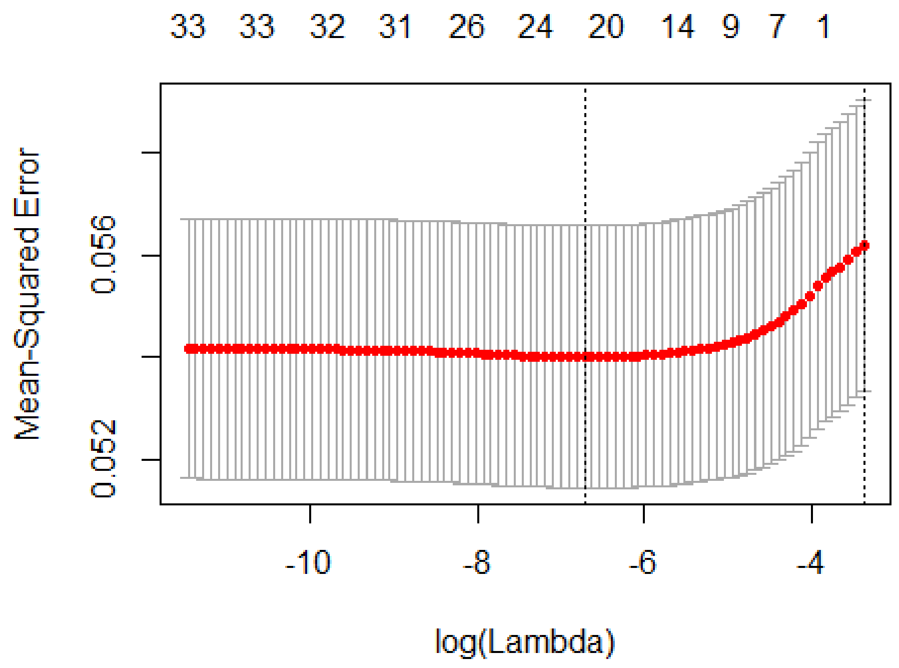


We observe that the attributes which were highly correlated, for instance V47 and V68 (car policies) in the previous cor-plot , One of them has been removed there by reducing inter-variable interactions.However there are still correlated which is due to multi-Collinearity.

2.2.3 Interpretting the Good set of predictors Using Lasso Regression.

Lasso is a regression analysis method that performs both variable selection and regularization in order to enhance the prediction accuracy of the model.It forces the predictor intercepts to Zero. Hence we can select best predictors by selecting only non Zero predictors.

```
data$V86=as.numeric(as.character(data$V86))
xmat <- model.matrix(V86 ~., data = data)
xmat <- xmat[, -1]
cv.lasso <- cv.glmnet(xmat, data$V86, alpha = 1)
plot(cv.lasso)
```



The plot above shows log(lambda) Vs Mean Squred Error. We can see the two vertical lines where fisrt from left showslambda value which has least Mean-Squared Error and second line shows lambda with 1 stadard devaition error.

We can access the min value by using the code below and fit the model using glmnet.

```
bestlam <- cv.lasso$lambda.min
fit.lasso <- glmnet(xmat, data$V86, alpha = 1)
```

The best lambda value obtained is 0.0012004 and we predict the values based on best lambda obtained.

```
variables=data.frame(predict(fit.lasso, s = bestlam, type =
"coefficients")[1:ncol(data),])
setDT(variables,keep.rownames = TRUE)
names(variables) <- c("variables", "Intercept")
```

The variables which have to be removed are
V14, V17, V19, V20, V23, V24, V25, V26, V27, V37, V39

and variables which are having some level of significance are (Intercept),

V2, V3, V4, V6, V15, V16, V18, V21, V22, V28, V29, V31, V35, V38, V40, V41, V42, V44, V59, V68, V75, V80.

```
data=data[,c("V2","V3","V4"
,"V6","V15","V16","V18","V21","V22","V23","V26","V28","V31","V35","V38","V40"
, "V41","V42" ,"V44", "V59", "V68", "V75" ,"V80","V86")]
```

2.2.3 Variance Inflation Factors

Let us Cross-Validate this using Variance Inflation Factors which identifies if there are multiple-collinearity between the preditors

A VIF of 1 means that there is no correlation among the k-th predictor and the remaining predictor variables, and hence the variance of the k-th coefficient is not inflated at all. The general rule of thumb is that VIFs exceeding 4 warrant further investigation, while VIFs exceeding 10 are signs of serious multicollinearity requiring correction.

```
inflation_var=vifcor(cor(data[,-24]),th=0.8)
inflation_var

## 7 variables from the 23 input variables have collinearity problem:
##
## V80 V15 V16 V28 V42 V18 V22
##
## After excluding the collinear variables, the linear correlation
coefficients ranges between:
## min correlation ( V59 ~ V26 ):  9.448888e-06
## max correlation ( V59 ~ V44 ):  0.7791747
##
## ---------- VIFs of the remained variables --------
##    Variables      VIF
## 1         V2 1.253844
## 2         V3 3.307789
## 3         V4 3.395178
## 4         V6 2.326028
## 5        V21 1.806059
## 6        V23 5.000058
## 7        V26 3.869114
## 8        V31 4.049060
## 9        V35 8.928974
## 10       V38 2.669794
## 11       V40 3.775307
## 12       V41 2.613321
## 13       V44 3.319629
## 14       V59 3.500338
## 15       V68 1.529744
## 16       V75 1.873178
```

From the Above we can say that there are multi-collinearity in the variables which are obtained.

```
a=inflation_var@excluded
data[ ,c(a)] <- list(NULL)
```

###2.3 Nominal Variables Reduction

Nominal Variables Significance are determined using Chi-Sq test where hypothesis is stated and depending on the p-value we accept or reject.

2.3.1 Chi-Square test for Nominal Variables(factors)

Our hypothesis for all the variables with target variable will be

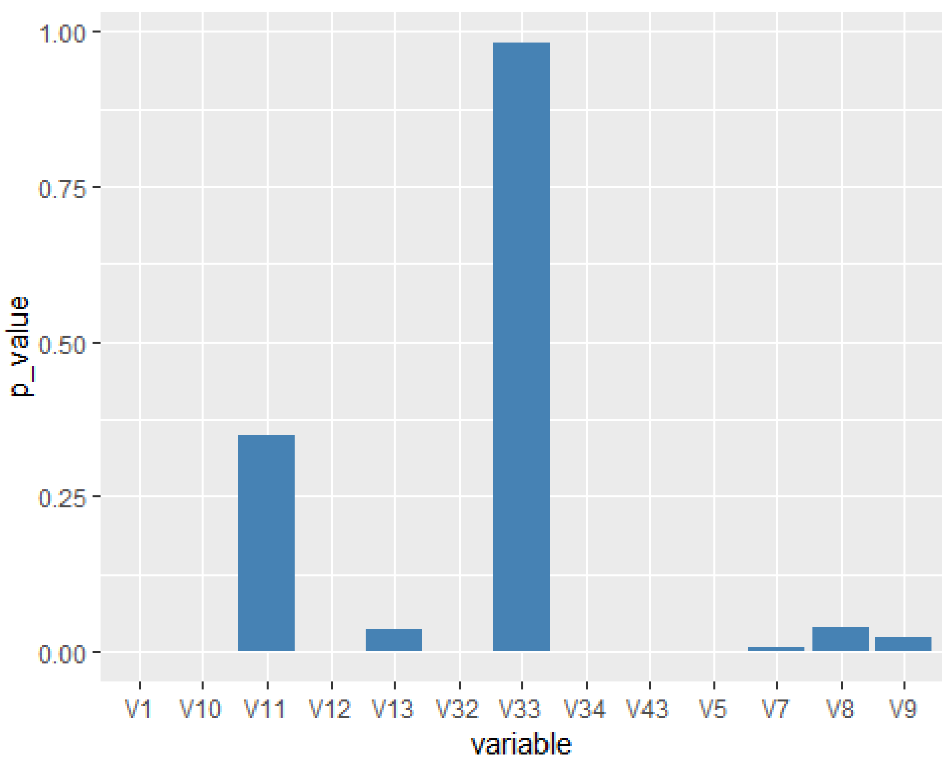
H0: Caravan Policy is Independant of variable X

HA: caravan policy is not Independant of Variable X

```
chisq_t=data.frame(mapply(function(x, y) chisq.test(x, y)$p.value,
data_sub_nominal[, -14],
                      MoreArgs=list(data_sub_nominal[,14])))
```

```
## Warning in chisq.test(x, y): Chi-squared approximation may be incorrect

## Warning in chisq.test(x, y): Chi-squared approximation may be incorrect

## Warning in chisq.test(x, y): Chi-squared approximation may be incorrect

## Warning in chisq.test(x, y): Chi-squared approximation may be incorrect

## Warning in chisq.test(x, y): Chi-squared approximation may be incorrect

## Warning in chisq.test(x, y): Chi-squared approximation may be incorrect

## Warning in chisq.test(x, y): Chi-squared approximation may be incorrect

## Warning in chisq.test(x, y): Chi-squared approximation may be incorrect

## Warning in chisq.test(x, y): Chi-squared approximation may be incorrect

## Warning in chisq.test(x, y): Chi-squared approximation may be incorrect

## Warning in chisq.test(x, y): Chi-squared approximation may be incorrect

## Warning in chisq.test(x, y): Chi-squared approximation may be incorrect
```

```
setDT(chisq_t,keep.rownames = TRUE)
names(chisq_t) <- c("variable", "p_value")

ggplot(data=chisq_t,aes(x=variable, y=p_value)) +
  geom_bar(stat="identity",fill="steelblue")
```



The Chi-Square Test shows some varaibles are dependant and others are not. Cosidering the variables for regression based on the above graph which is P-value Vs Variables, we get V1,V10,V32,V34,V43,V5 as dependant.

```
# Subsetting only the required variables

data_sub_nominal=data_sub_nominal[,c('V1','V10','V12','V32','V34','V43','V5')
]
```

2.3.2 Inter-correlation(Interaction) between these categorical variables.

Interactions can be determined by plotting the two variables as shown below

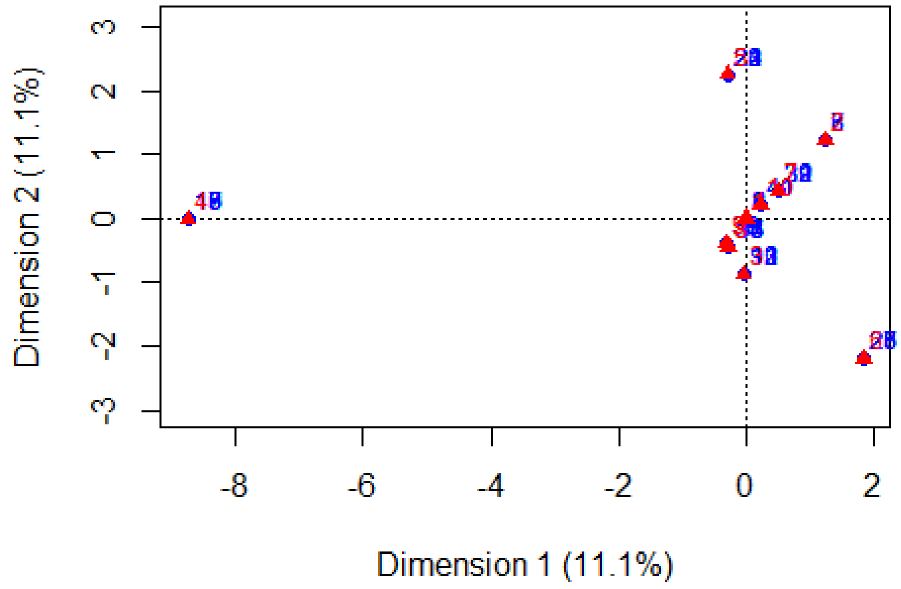
```
mytable <- with(data_sub_nominal, table(V1,V5)) # create a 2 way table
prop.table(mytable, 1) # row percentages

##     V5
## V1    1 2 3 4 5 6 7 8 9 10
##   1   1 0 0 0 0 0 0 0 0  0
##   2   1 0 0 0 0 0 0 0 0  0
##   3   1 0 0 0 0 0 0 0 0  0
##   4   1 0 0 0 0 0 0 0 0  0
##   5   1 0 0 0 0 0 0 0 0  0
##   6   0 1 0 0 0 0 0 0 0  0
##   7   0 1 0 0 0 0 0 0 0  0
##   8   0 1 0 0 0 0 0 0 0  0
##   9   0 0 1 0 0 0 0 0 0  0
##   10  0 0 1 0 0 0 0 0 0  0
##   11 0 0 1 0 0 0 0 0 0  0
##   12 0 0 1 0 0 0 0 0 0  0
##   13 0 0 1 0 0 0 0 0 0  0
##   15 0 0 0 1 0 0 0 0 0  0
##   16 0 0 0 1 0 0 0 0 0  0
##   17 0 0 0 1 0 0 0 0 0  0
##   18 0 0 0 1 0 0 0 0 0  0
##   19 0 0 0 1 0 0 0 0 0  0
##   20 0 0 0 0 1 0 0 0 0  0
##   21 0 0 0 0 1 0 0 0 0  0
##   22 0 0 0 0 1 0 0 0 0  0
##   23 0 0 0 0 1 0 0 0 0  0
##   24 0 0 0 0 1 0 0 0 0  0
##   25 0 0 0 0 0 1 0 0 0  0
##   26 0 0 0 0 0 1 0 0 0  0
##   27 0 0 0 0 0 1 0 0 0  0
##   28 0 0 0 0 0 1 0 0 0  0
##   29 0 0 0 0 0 0 1 0 0  0
##   30 0 0 0 0 0 0 1 0 0  0
##   31 0 0 0 0 0 0 1 0 0  0
##   32 0 0 0 0 0 0 1 0 0  0
##   33 0 0 0 0 0 0 0 1 0  0
##   34 0 0 0 0 0 0 0 1 0  0
##   35 0 0 0 0 0 0 0 1 0  0
##   36 0 0 0 0 0 0 0 1 0  0
##   37 0 0 0 0 0 0 0 1 0  0
##   38 0 0 0 0 0 0 0 0 1  0
##   39 0 0 0 0 0 0 0 0 1  0
##   40 0 0 0 0 0 0 0 0 0  1
##   41 0 0 0 0 0 0 0 0 0  1
```

```
prop.table(mytable, 2) # column percentages
```

```
##     V5
## V1            1          2          3          4          5          6
##   1  0.22463768 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
##   2  0.14855072 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
##   3  0.45108696 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
##   4  0.09420290 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
##   5  0.08152174 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
##   6  0.00000000 0.23705179 0.00000000 0.00000000 0.00000000 0.00000000
##   7  0.00000000 0.08764940 0.00000000 0.00000000 0.00000000 0.00000000
##   8  0.00000000 0.67529880 0.00000000 0.00000000 0.00000000 0.00000000
##   9  0.00000000 0.00000000 0.31376975 0.00000000 0.00000000 0.00000000
##   10 0.00000000 0.00000000 0.18623025 0.00000000 0.00000000 0.00000000
##   11 0.00000000 0.00000000 0.17268623 0.00000000 0.00000000 0.00000000
##   12 0.00000000 0.00000000 0.12528217 0.00000000 0.00000000 0.00000000
##   13 0.00000000 0.00000000 0.20203160 0.00000000 0.00000000 0.00000000
##   15 0.00000000 0.00000000 0.00000000 0.09615385 0.00000000 0.00000000
##   16 0.00000000 0.00000000 0.00000000 0.30769231 0.00000000 0.00000000
##   17 0.00000000 0.00000000 0.00000000 0.17307692 0.00000000 0.00000000
##   18 0.00000000 0.00000000 0.00000000 0.36538462 0.00000000 0.00000000
##   19 0.00000000 0.00000000 0.00000000 0.05769231 0.00000000 0.00000000
##   20 0.00000000 0.00000000 0.00000000 0.00000000 0.04393673 0.00000000
##   21 0.00000000 0.00000000 0.00000000 0.00000000 0.02636204 0.00000000
##   22 0.00000000 0.00000000 0.00000000 0.00000000 0.17223199 0.00000000
##   23 0.00000000 0.00000000 0.00000000 0.00000000 0.44112478 0.00000000
##   24 0.00000000 0.00000000 0.00000000 0.00000000 0.31634446 0.00000000
##   25 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.40000000
##   26 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.23414634
##   27 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.24390244
##   28 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.12195122
##   29 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
##   30 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
##   31 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
##   32 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
##   33 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
##   34 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
##   35 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
##   36 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
##   37 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
##   38 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
##   39 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
##   40 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
##   41 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
##     V5
## V1            7          8          9         10
##   1  0.00000000 0.00000000 0.00000000 0.00000000
##   2  0.00000000 0.00000000 0.00000000 0.00000000
##   3  0.00000000 0.00000000 0.00000000 0.00000000
##   4  0.00000000 0.00000000 0.00000000 0.00000000
##   5  0.00000000 0.00000000 0.00000000 0.00000000
##   6  0.00000000 0.00000000 0.00000000 0.00000000
##   7  0.00000000 0.00000000 0.00000000 0.00000000
##   8  0.00000000 0.00000000 0.00000000 0.00000000
##   9  0.00000000 0.00000000 0.00000000 0.00000000
##   10 0.00000000 0.00000000 0.00000000 0.00000000
##   11 0.00000000 0.00000000 0.00000000 0.00000000
##   12 0.00000000 0.00000000 0.00000000 0.00000000
##   13 0.00000000 0.00000000 0.00000000 0.00000000
##   15 0.00000000 0.00000000 0.00000000 0.00000000
##   16 0.00000000 0.00000000 0.00000000 0.00000000
##   17 0.00000000 0.00000000 0.00000000 0.00000000
##   18 0.00000000 0.00000000 0.00000000 0.00000000
##   19 0.00000000 0.00000000 0.00000000 0.00000000
##   20 0.00000000 0.00000000 0.00000000 0.00000000
##   21 0.00000000 0.00000000 0.00000000 0.00000000
##   22 0.00000000 0.00000000 0.00000000 0.00000000
##   23 0.00000000 0.00000000 0.00000000 0.00000000
##   24 0.00000000 0.00000000 0.00000000 0.00000000
##   25 0.00000000 0.00000000 0.00000000 0.00000000
##   26 0.00000000 0.00000000 0.00000000 0.00000000
##   27 0.00000000 0.00000000 0.00000000 0.00000000
##   28 0.00000000 0.00000000 0.00000000 0.00000000
##   29 0.15636364 0.00000000 0.00000000 0.00000000
##   30 0.21454545 0.00000000 0.00000000 0.00000000
##   31 0.37272727 0.00000000 0.00000000 0.00000000
##   32 0.25636364 0.00000000 0.00000000 0.00000000
##   33 0.00000000 0.51823417 0.00000000 0.00000000
##   34 0.00000000 0.11644274 0.00000000 0.00000000
##   35 0.00000000 0.13691619 0.00000000 0.00000000
##   36 0.00000000 0.14395393 0.00000000 0.00000000
##   37 0.00000000 0.08445298 0.00000000 0.00000000
##   38 0.00000000 0.00000000 0.50824588 0.00000000
##   39 0.00000000 0.00000000 0.49175412 0.00000000
##   40 0.00000000 0.00000000 0.00000000 0.25724638
##   41 0.00000000 0.00000000 0.00000000 0.74275362
```

```
fit <- ca(mytable)
print(fit) # basic results
```

```
##
##  Principal inertias (eigenvalues):
##            1      2      3      4      5      6      7      8      9
## Value      1      1      1      1      1      1      1      1      1
## Percentage 11.11% 11.11% 11.11% 11.11% 11.11% 11.11% 11.11% 11.11% 11.11%
##
##
##  Rows:
##                1        2        3        4        5        6        7
## Mass    0.021299 0.014085 0.042769 0.008932 0.007729 0.020440 0.007558
## ChiDist 3.089838 3.089838 3.089838 3.089838 3.089838 3.255397 3.255397
## Inertia 0.203339 0.134466 0.408318 0.085271 0.073792 0.216612 0.080092
## Dim. 1  0.000000 0.000000 0.000000 0.000000 0.000000 1.241202 1.241202
## Dim. 2  0.000000 0.000000 0.000000 0.000000 0.000000 1.214970 1.214970
##                8         9        10        11        12        13
## Mass    0.058227  0.047750  0.028341  0.026280  0.019066  0.030745
## ChiDist 3.255397  2.360319  2.360319  2.360319  2.360319  2.360319
## Inertia 0.617071  0.266020  0.157889  0.146407  0.106217  0.171286
## Dim. 1  1.241202 -0.049025 -0.049025 -0.049025 -0.049025 -0.049025
## Dim. 2  1.214970 -0.870969 -0.870969 -0.870969 -0.870969 -0.870969
##                15         16         17         18         19        20
## Mass     0.000859   0.002748   0.001546   0.003263   0.000515  0.004294
## ChiDist 10.533828 10.533828 10.533828 10.533828 10.533828  3.038418
## Inertia  0.095295   0.304944   0.171531   0.362121   0.057177  0.039643
## Dim. 1  -8.742565 -8.742565 -8.742565 -8.742565 -8.742565 -0.302948
## Dim. 2  -0.019431 -0.019431 -0.019431 -0.019431 -0.019431  2.254888
##                21         22         23         24         25        26
## Mass     0.002576   0.016833   0.043112   0.030917   0.014085  0.008245
## ChiDist  3.038418   3.038418   3.038418   3.038418   5.234501  5.234501
## Inertia  0.023786  0.155399  0.398012  0.285427  0.385915  0.225902
## Dim. 1  -0.302948 -0.302948 -0.302948 -0.302948  1.825175  1.825175
## Dim. 2   2.254888  2.254888  2.254888  2.254888 -2.196991 -2.196991
##                27        28        29        30        31        32        33
## Mass     0.008588  0.004294 0.014772 0.020268 0.035211 0.024218  0.139127
## ChiDist  5.234501  5.234501 3.096039 3.096039 3.096039 3.096039  1.650723
## Inertia  0.235314  0.117657 0.141592 0.194278 0.337516 0.232145  0.379107
## Dim. 1   1.825175  1.825175 0.488025 0.488025 0.488025 0.488025 -0.278422
## Dim. 2  -2.196991 -2.196991 0.425619 0.425619 0.425619 0.425619 -0.446308
##                34        35        36        37        38        39
## Mass     0.031261  0.036757  0.038647  0.022673  0.058227  0.056338
## ChiDist  1.650723  1.650723  1.650723  1.650723  2.780042  2.780042
## Inertia  0.085182  0.100159  0.105307  0.061780  0.450018  0.435416
## Dim. 1  -0.278422 -0.278422 -0.278422 -0.278422 -0.328659 -0.328659
## Dim. 2  -0.446308 -0.446308 -0.446308 -0.446308 -0.394279 -0.394279
##               40       41
## Mass    0.012195 0.035211
## ChiDist 4.482656 4.482656
## Inertia 0.245051 0.707542
## Dim. 1  0.214339 0.214339
## Dim. 2  0.205066 0.205066
##
##
##  Columns:
##               1        2         3         4         5         6        7
## Mass    0.094813 0.086225  0.152181  0.008932  0.097733  0.035211 0.094469
## ChiDist 3.089838 3.255397  2.360319 10.533828  3.038418  5.234501 3.096039
## Inertia 0.905187 0.913775  0.847819  0.991068  0.902267  0.964789 0.905531
## Dim. 1  0.000000 1.241202 -0.049025 -8.742565 -0.302948  1.825175 0.488025
## Dim. 2  0.000000 1.214970 -0.870969 -0.019431  2.254888 -2.196991 0.425619
##                 8         9       10
## Mass     0.268464  0.114565 0.047406
## ChiDist  1.650723  2.780042 4.482656
## Inertia  0.731536  0.885435 0.952594
## Dim. 1  -0.278422 -0.328659 0.214339
## Dim. 2  -0.446308 -0.394279 0.205066

plot(fit) # symmetric map
```



From the above graph we can say that both the dimension points lie exactly one top of another. Hence V1 and V5 are highly correlated.

Likewise for V32 and V34

```
mytable <- with(data_sub_nominal, table(V32,V34)) # create a 2 way table
prop.table(mytable, 1) # row percentages
```

```
##     V34
## V32            0            1            2            3            4
##    0 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.1052631579
##    1 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000
##    2 0.0172413793 0.0000000000 0.0000000000 0.1206896552 0.0000000000
##    3 0.0389610390 0.0000000000 0.0519480519 0.1515151515 0.3333333333
##    4 0.0558035714 0.0401785714 0.2075892857 0.1316964286 0.2566964286
##    5 0.1479338843 0.0330578512 0.1190082645 0.3743801653 0.3247933884
##    6 0.1082381239 0.1834034877 0.4004810583 0.3078773301 0.0000000000
##    7 0.3389950460 0.1585279547 0.5024769993 0.0000000000 0.0000000000
##    8 0.2758620690 0.7241379310 0.0000000000 0.0000000000 0.0000000000
##    9 1.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000
##     V34
## V32            5            6            7            8            9
##    0 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.8947368421
##    1 0.0000000000 0.0714285714 0.0000000000 0.9285714286 0.0000000000
##    2 0.2241379310 0.2068965517 0.4310344828 0.0000000000 0.0000000000
##    3 0.0952380952 0.3290043290 0.0000000000 0.0000000000 0.0000000000
##    4 0.3080357143 0.0000000000 0.0000000000 0.0000000000 0.0000000000
##    5 0.0008264463 0.0000000000 0.0000000000 0.0000000000 0.0000000000
##   6 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000
##   7 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000
##   8 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000
##   9 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000
```

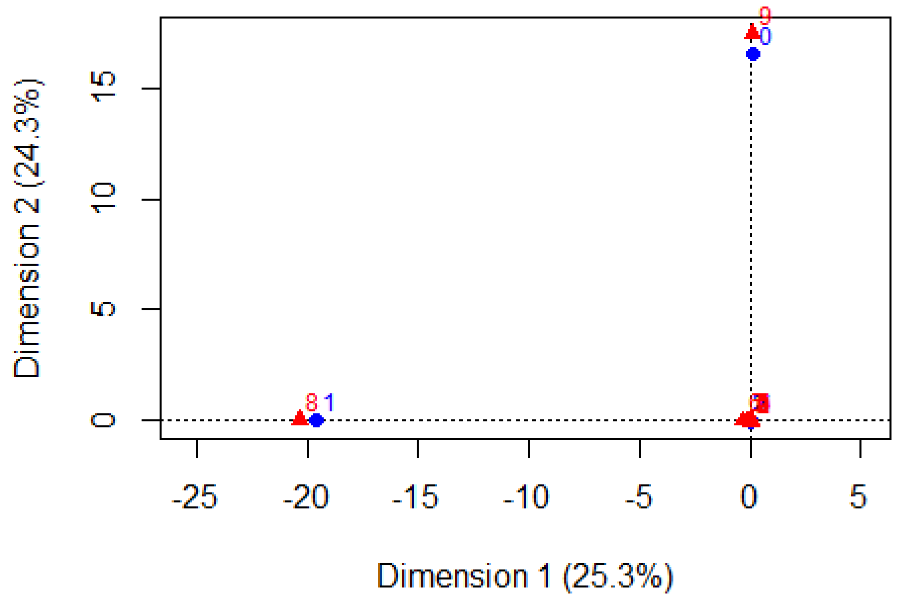
```
prop.table(mytable, 2) # column percentages
```

```
##    V34
## V32            0            1            2            3            4
##   0 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0034071550
##   1 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000
##   2 0.0006896552 0.0000000000 0.0000000000 0.0065666041 0.0000000000
##   3 0.0062068966 0.0000000000 0.0073846154 0.0328330206 0.1311754685
##   4 0.0172413793 0.0231958763 0.0572307692 0.0553470919 0.1959114140
##   5 0.1234482759 0.0515463918 0.0886153846 0.4249530957 0.6695059625
##   6 0.1241379310 0.3930412371 0.4098461538 0.4803001876 0.0000000000
##   7 0.3303448276 0.2886597938 0.4369230769 0.0000000000 0.0000000000
##   8 0.0496551724 0.2435567010 0.0000000000 0.0000000000 0.0000000000
##   9 0.3482758621 0.0000000000 0.0000000000 0.0000000000 0.0000000000
##    V34
## V32            5            6            7            8            9
##   0 0.0000000000 0.0000000000 0.0000000000 0.0000000000 1.0000000000
##   1 0.0000000000 0.0112359551 0.0000000000 1.0000000000 0.0000000000
##   2 0.0747126437 0.1348314607 1.0000000000 0.0000000000 0.0000000000
##   3 0.1264367816 0.8539325843 0.0000000000 0.0000000000 0.0000000000
##   4 0.7931034483 0.0000000000 0.0000000000 0.0000000000 0.0000000000
##   5 0.0057471264 0.0000000000 0.0000000000 0.0000000000 0.0000000000
##   6 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000
##   7 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000
##   8 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000
##   9 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000
```

```
fit <- ca(mytable)
print(fit) # basic results
```

```
##
##  Principal inertias (eigenvalues):
##            1        2        3        4        5        6        7
## Value      0.929696 0.894925 0.582677 0.413529 0.291328 0.209759 0.19892
## Percentage 25.27%   24.32%   15.84%   11.24%   7.92%    5.7%     5.41%
##            8        9
## Value      0.122866 0.035447
## Percentage 3.34%    0.96%
##
##
##  Rows:
##                 0          1         2         3         4         5
## Mass     0.003263   0.002405  0.009962  0.039677  0.076950  0.207832
## ChiDist 16.531065  19.633818  6.843160  2.761794  1.761289  0.979085
## Inertia  0.891832   0.926969  0.466519  0.302637  0.238708  0.199230
## Dim. 1   0.082799 -20.356288 -0.115444 -0.089267  0.036307  0.047518
## Dim. 2  17.472343   0.030165 -0.035053 -0.011502 -0.030272 -0.031259
##                 6         7         8         9
## Mass     0.285641  0.242700  0.044830  0.086740
## ChiDist  0.625872  0.744682  1.799923  1.736425
## Inertia  0.111890  0.134590  0.145237  0.261536
## Dim. 1   0.059198  0.063133  0.064879  0.064126
## Dim. 2  -0.068012 -0.074324 -0.076841 -0.075528
##
##
##  Columns:
##                 0         1         2         3        4         5
## Mass     0.249055  0.133288  0.279114  0.183099 0.100824  0.029887
## ChiDist  1.017440  1.107765  0.675543  0.864767 1.446633  2.852673
## Inertia  0.257818  0.163563  0.127376  0.136925 0.211000  0.243210
## Dim. 1   0.061831  0.062833  0.059609  0.048688 0.028519  0.009496
## Dim. 2  -0.071450 -0.073165 -0.068642 -0.050986 0.032942 -0.029875
##                 6         7          8         9
## Mass     0.015287  0.004294   0.002233  0.002920
## ChiDist  4.388136  9.968917  20.368042 17.476300
## Inertia  0.294359  0.426740   0.926339  0.891817
## Dim. 1  -0.332414 -0.119730 -21.111937  0.085873
## Dim. 2  -0.015020 -0.037054   0.031886 18.469619
```

```
plot(fit) # symmetric map
```



Even V34 and V32 are also closely related.Hence these are intercorrelated.

Similarly when we compute for other combinations we get V1+V5+V43,v32+V34 , V32+V10 , V32+V12 are all correlated. Therefore we just keep one of the variable as predictors. I chose V43 and V32 as predictors from this group of nominal variables.

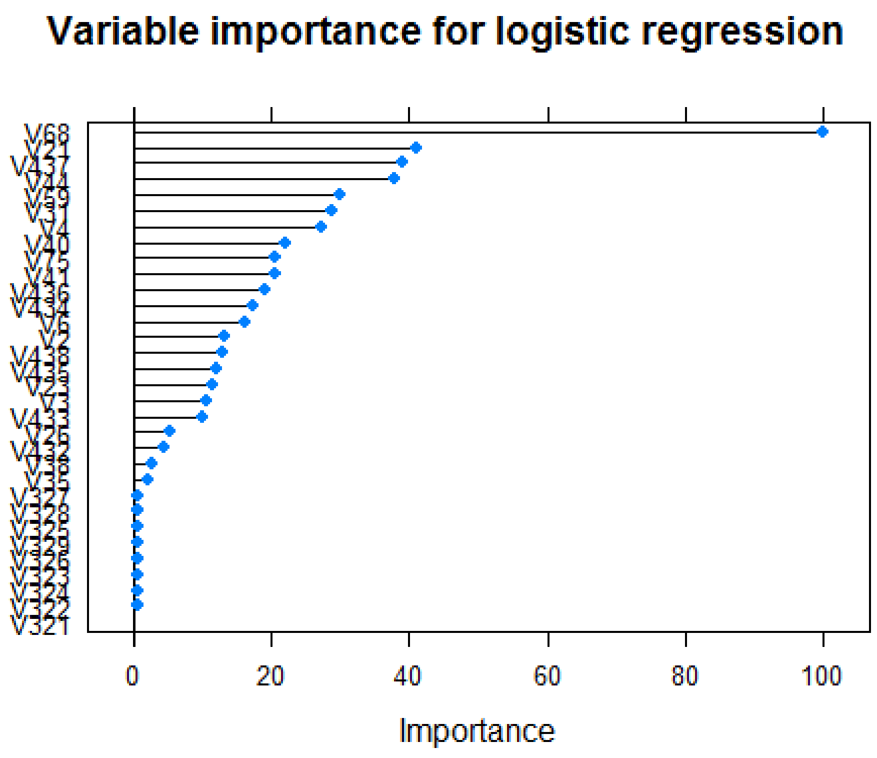
```
# Merging the two Dataframes of Nominal and Ordinal

data_sub_nominal=data_sub_nominal[,c('V43','V32')]
data=cbind(data_sub_nominal,data)
```

Using GLM to find the significance of the variables which also helps in variable selection

```
logit <- train(V86~., data=data, method='glm', family=binomial(link='logit'),
               preProcess=c('scale', 'center'))

## Warning in train.default(x, y, weights = w, ...): You are trying to do
## regression and your outcome only has two possible values Are you trying to
## do classification? If so, use a 2 level factor as your outcome column.

summary(logit)

## 
## Call:
## NULL
## 
## Deviance Residuals: 
##     Min       1Q   Median       3Q      Max  
## -1.9098  -0.3760  -0.2758  -0.1995   2.9841  
## 
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)    
## (Intercept)    -3.17936    1.39621  -2.277 0.022778 *  
## V432            0.03410    0.09608   0.355 0.722661    
## V433            0.10450    0.12427   0.841 0.400398    
## V434            0.16100    0.10849   1.484 0.137787    
## V435            0.09732    0.09488   1.026 0.305014    
## V436            0.17013    0.10526   1.616 0.106034    
## V437            0.28744    0.08642   3.326 0.000880 ***
## V438            0.09883    0.09006   1.097 0.272437    
## V321            0.01515   24.50844   0.001 0.999507    
## V322            1.11731   31.71926   0.035 0.971900    
## V323            2.28996   62.34418   0.037 0.970700    
## V324            3.07714   85.12025   0.036 0.971162    
## V325            4.90316  129.59321   0.038 0.969819    
## V326            5.40994  144.27315   0.037 0.970088    
## V327            5.30045  136.92604   0.039 0.969121    
## V328            2.51413   66.09091   0.038 0.969655    
## V329            3.37235   89.89261   0.038 0.970074    
## V2             -0.08315    0.07357  -1.130 0.258368    
## V3              0.06399    0.07169   0.893 0.372078    
## V4              0.16270    0.06956   2.339 0.019330 *  
## V6             -0.08556    0.06211  -1.378 0.168355    
## V21          -0.30437    0.08676  -3.508 0.000451 ***
## V23          -0.06593    0.06802  -0.969 0.332440    
## V26           0.02691    0.05974   0.450 0.652403    
## V31           0.18295    0.07425   2.464 0.013741 *  
## V35          -0.01179    0.06939  -0.170 0.865069    
## V38           0.01348    0.06336   0.213 0.831562    
## V40           0.10727    0.05698   1.883 0.059748 .  
## V41          -0.11693    0.06694  -1.747 0.080667 .  
## V44           0.21187    0.06561   3.229 0.001241 ** 
## V59           0.17679    0.06911   2.558 0.010520 *  
## V68           0.44375    0.05190   8.549  < 2e-16 ***
## V75          -0.16879    0.09641  -1.751 0.079981 .  
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## (Dispersion parameter for binomial family taken to be 1)
## 
##     Null deviance: 2635.5  on 5821  degrees of freedom
## Residual deviance: 2365.4  on 5789  degrees of freedom
## AIC: 2431.4
## 
## Number of Fisher Scoring iterations: 14
```

From the Summary it is clear that V44,V68,V59,V21,V31,V43 are more significant when compared to others. Variable importance can also be viewed through the plot below.

```
plot(varImp(logit, scale = TRUE), main = "Variable importance for logistic
regression")
```



The Graph Above shows the variable significance in the model development. Hence I consider variables which are having more 40% for prediction task.

3. Model Development

1. First we divide the data into Train and Test

```
# Dividing the dataset 75/25 ratio where 75 is Train and  25 is Test
inTrain <- createDataPartition(y = data$V86, p = 0.75, list = FALSE)
train <- data[inTrain,]
test <- data[-inTrain,]
train[,'V86']<-factor(train[,'V86'])
test[,'V86']<-factor(test[,'V86'])
```

3.1 Predicting using Generalised Linear Model

Model Description:

Logistic regression model is generalised version of linear model which can be applied when we dont have any idea of data and attribute significance. It is suggested that if the output is binary using GLM would be give better results, It also provides the flexibilty to set the probablity threshold. It works good when we have both mix and nominal,ordinal and continous attributes.

Applying GLM on the Train dataset with the selected predictors

```
m1=glm(V86~V44+V68+V21+V31+V43,data=train,family="binomial")
summary(m1)
```

```
##
## Call:
## glm(formula = V86 ~ V44 + V68 + V21 + V31 + V43, family = "binomial",
##     data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0417  -0.3771  -0.2824  -0.2164   2.8933
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -4.17039    0.28971 -14.395  < 2e-16 ***
## V44          0.29698    0.06666   4.455 8.37e-06 ***
## V68          0.81064    0.10085   8.038 9.14e-16 ***
## V21         -0.26683    0.08946  -2.983  0.00286 **
## V31          0.06450    0.02476   2.605  0.00919 **
## V432        -0.13313    0.41495  -0.321  0.74833
## V433         0.15808    0.30553   0.517  0.60487
## V434         0.29902    0.32372   0.924  0.35564
## V435         0.25996    0.34453   0.755  0.45053
## V436         0.60734    0.31056   1.956  0.05051 .
## V437         1.00471    0.33085   3.037  0.00239 **
## V438         0.21042    0.37719   0.558  0.57694
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1993.2  on 4366  degrees of freedom
## Residual deviance: 1825.4  on 4355  degrees of freedom
## AIC: 1849.4
##
## Number of Fisher Scoring iterations: 6
```

- Deviance Residual : In the first part of the summary tells badness of the fit. Higher the residual deviance worse is the fit, The residual deviance shown in the summary result above looks roughly symmetric.

The next part of the output shows the coefficients, their standard errors, the z-statistic, and the associated p-values

- Coeficients :

    1. Intercepts:The logistic regression coefficients gives the change in the log odds of the outcome for a one unit increase in the predictor variable.For Instance every unit change in V44 log odds of V86 change by 0.322

2. Z-value: The z-value is the regression coefficient divided by its standard error.

3. P-value : Shows the probablity value and stars next to that identifies significance level.

4. Std.Error - Error in the estimates

- Null deviance: shows how well our target variable is predicted given only intercept. In our case its 1993.2

- Residual deviance : Shows how well the model fits into train set

- AIC : It is the way of accessing the model peformance. It penalises the model if there are many predictors. Less the AIC better the model. In our case it is 1844.4

```
#calucating the probablities using predict
probabilities <- predict(m1, test[-ncol(test)], type = "response")
predicted <- ifelse(probabilities > 0.5,1,0)
#observed.classes=targetdata$V1
mean(predicted == test$V86)

## [1] 0.9415808

# Displaying the confusion Matrix of all possible True false positve and
negative cases
u <- union(predicted, test$V86)
t <- table(factor(predicted, u), factor(test$V86, u))
confusionMatrix(t,positive='1')

## Confusion Matrix and Statistics
##
##
##        0    1
##   0 1370   84
##   1    1    0
##
##                Accuracy : 0.9416
##                  95% CI : (0.9283, 0.9531)
##     No Information Rate : 0.9423
##     P-Value [Acc > NIR] : 0.5733
##
##                   Kappa : -0.0014
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.0000000
##             Specificity : 0.9992706
##          Pos Pred Value : 0.0000000
##          Neg Pred Value : 0.9422283
##              Prevalence : 0.0577320
##          Detection Rate : 0.0000000
##    Detection Prevalence : 0.0006873
##       Balanced Accuracy : 0.4996353
##
##        'Positive' Class : 1
##
```

The model shows accuracy of 94% where it has specificity is high and sensitivity is low. This model is not all predicting the correct 1's. Its more like saying just add zero to all targets and still you end up in more accurate results.

It may be because of dataset is unbalanced with very less number of 1's than 0's; We shall try putting this into testset provided and compare the results.

Anova Test for Model Attributes:

```
anova(m1, test = "Chisq")

## Analysis of Deviance Table
##
## Model: binomial, link: logit
##
## Response: V86
##
## Terms added sequentially (first to last)
##
##
##      Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
## NULL                  4366     1993.2
## V44   1   37.122      4365     1956.1 1.110e-09 ***
## V68   1   72.778      4364     1883.3 < 2.2e-16 ***
## V21   1   13.032      4363     1870.3 0.0003061 ***
## V31   1   23.107      4362     1847.2 1.532e-06 ***
## V43   7   21.729      4355     1825.5 0.0028282 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Anova Function gives the deviance table where one can analyse as how deviance varies when an attribute is added to model.

Here we can see all predictors are significant with significance value very much less than 0.05.

```
with(m1, pchisq(null.deviance - deviance, df.null - df.residual, lower.tail =
FALSE))

## [1] 3.394139e-30
```

The chi-square of 172.8 with 11 degrees of freedom and an associated p-value of less than tells us that our model as a whole fits significantly better than an empty model.

Applying Model to Original Test Data

```
#Converting the variables in Test data.
testdata[,'V1']<-factor(testdata[,'V1'])
testdata[,'V5']<-factor(testdata[,'V5'])
testdata[,'V7']<-factor(testdata[,'V7'])
testdata[,'V8']<-factor(testdata[,'V8'])
testdata[,'V9']<-factor(testdata[,'V9'])
```

```
testdata[,'V10']<-factor(testdata[,'V10'])
testdata[,'V11']<-factor(testdata[,'V11'])
testdata[,'V12']<-factor(testdata[,'V12'])
testdata[,'V13']<-factor(testdata[,'V13'])
testdata[,'V43']<-factor(testdata[,'V43'])
testdata[,'V32']<-factor(testdata[,'V32'])
testdata[,'V33']<-factor(testdata[,'V33'])
testdata[,'V34']<-factor(testdata[,'V34'])
```

Finding the probablities of test set

```
probabilities <- predict(m1, testdata, type = "response")

# Code Block to set threshold
testdata1=testdata
testdata1$prob=probabilities
testdata1 <- testdata1[order(-testdata1$prob),] # arranging in Descending
order of their probablities
threshold=testdata1$prob[800] # Picking the threshold to be 800th customer's
probablity

predicted <- ifelse(probabilities >=threshold,1,0)
observed.classes=targetdata$V1
mean(predicted == observed.classes)

## [1] 0.7895

# Creation of Confusion Matrix
u <- union(predicted, observed.classes)
t <- table(factor(predicted, u), factor(observed.classes, u))
confusionMatrix(t,positive='1')

## Confusion Matrix and Statistics
##
##
##         0    1
##   0 3059  139
##   1  703   99
##
##                Accuracy : 0.7895
##                  95% CI : (0.7765, 0.802)
##     No Information Rate : 0.9405
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1086
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.41597
##             Specificity : 0.81313
##          Pos Pred Value : 0.12344
##          Neg Pred Value : 0.95654
##              Prevalence : 0.05950
##          Detection Rate : 0.02475
##    Detection Prevalence : 0.20050
##       Balanced Accuracy : 0.61455
##
##        'Positive' Class : 1
##
```

The Confusion Matrix Displays out of top 809 Customers who are potential, 105 are correctly predicted and hence the sensitivity and specificity of the model turns out to be 44.1% and 81.2% respectivly. The overall accuarcy of the model gives approximately 79%

3.2 Predicting Using Naive Bayes

Model Description:

It is machine learning technique which is purely based on Bayes theorem with strong Independence assumptions among the predictors. This Model would be worth of applying as we have removed all the intercorrelations/Collinearity between the model and also we have case of classifying the customers.

```
modelnaive <- naiveBayes(V86~V44+V68+V21+V31+V43, data=train)
modelnaive

##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##         0         1
## 0.9395466 0.0604534
##
## Conditional probabilities:
##    V44
## Y        [,1]      [,2]
##   0 0.7440897 0.9514410
##   1 1.1212121 0.9906813
##
##    V68
## Y        [,1]      [,2]
##   0 0.5378991 0.5932862
##   1 0.9090909 0.5712310
##
##    V21
## Y        [,1]      [,2]
##   0 0.5564221 1.0996973
##   1 0.2992424 0.7279322
##
##    V31
## Y       [,1]     [,2]
##   0 4.730441 3.077334
##   1 5.628788 3.071778
##
##    V43
## Y            1          2          3          4          5          6
##   0 0.10187668 0.07457958 0.26444065 0.15573970 0.10382647 0.15647087
##   1 0.05681818 0.04166667 0.20454545 0.12500000 0.09090909 0.21969697
##    V43
## Y            7          8
##   0 0.07360468 0.06946137
##   1 0.18560606 0.07575758
```

1. The model summary gives the conditional probablities of the predictors which is in tables
2. Level : Displays the level in the output.
3. apriori : Displays A-priori probabilities.

```
probabilities <- predict(modelnaive, test[-ncol(test)], type = "class")
observed.classes=test$V86
mean(probabilities == observed.classes)

## [1] 0.9402062

u <- union(probabilities, observed.classes)
t <- table(factor(probabilities, u), factor(observed.classes, u))
confusionMatrix(t,positive='1')

## Confusion Matrix and Statistics
##
##
##        0    1
##   0 1366   82
##   1    5    2
##
##                Accuracy : 0.9402
##                  95% CI : (0.9268, 0.9518)
##     No Information Rate : 0.9423
##     P-Value [Acc > NIR] : 0.6581
##
##                   Kappa : 0.0354
##
##  Mcnemar's Test P-Value : 3.698e-16
##
##             Sensitivity : 0.023810
##             Specificity : 0.996353
##          Pos Pred Value : 0.285714
##          Neg Pred Value : 0.943370
##                Prevalence : 0.057732
##            Detection Rate : 0.001375
##      Detection Prevalence : 0.004811
##         Balanced Accuracy : 0.510081
##
##          'Positive' Class : 1
##
```

The Confusion Matix shows Sensitivity as Zero and Specificity to 99.85% and the overall percentage fit is 94.09%

Appplying the Model on Original TestData

```
probabilities <- data.frame(predict(modelnaive, testdata, type = "raw"))
# Reordering the dataframe to get 800 customers
testdata1=testdata
testdata1$prob=probabilities$X1
testdata1 <- testdata1[order(-testdata1$prob),] # arranging in Descending
order of their probablities
threshold=testdata1$prob[800] # Picking the threshold to be 800th customer's
probablity

predicted.classes <- ifelse(probabilities$X1 >= threshold, "1", "0")
#Predicting the probablities
observed.classes=targetdata$V1
mean(predicted.classes == observed.classes)

## [1] 0.789

u <- union(predicted.classes, observed.classes)
t <- table(factor(predicted.classes, u), factor(observed.classes, u))
confusionMatrix(t,positive='1')

## Confusion Matrix and Statistics
##
##
##         0    1
##    0 3055  137
##    1  707  101
##
##                  Accuracy : 0.789
##                    95% CI : (0.776, 0.8016)
##       No Information Rate : 0.9405
##       P-Value [Acc > NIR] : 1
##
##                     Kappa : 0.1114
##
##    Mcnemar's Test P-Value : <2e-16
##
##               Sensitivity : 0.42437
##               Specificity : 0.81207
##          Pos Pred Value : 0.12500
##          Neg Pred Value : 0.95708
##              Prevalence : 0.05950
##          Detection Rate : 0.02525
##    Detection Prevalence : 0.20200
##       Balanced Accuracy : 0.61822
##
##        'Positive' Class : 1
##
```

The Confusion Matrix Displays out of top 805 Customers who are potential, 104 are correctly predicted and hence the sensitivity and specificity of the model turns out to be 43.7% and 81.39% respectivly. The overall accuarcy of the model gives approximately 79%

4 Model Comparison

ROC Curve shows the trade off between sensitvity and Specificity; Hence we try to plot ROC curve for both the methods and Check for Area under curve.

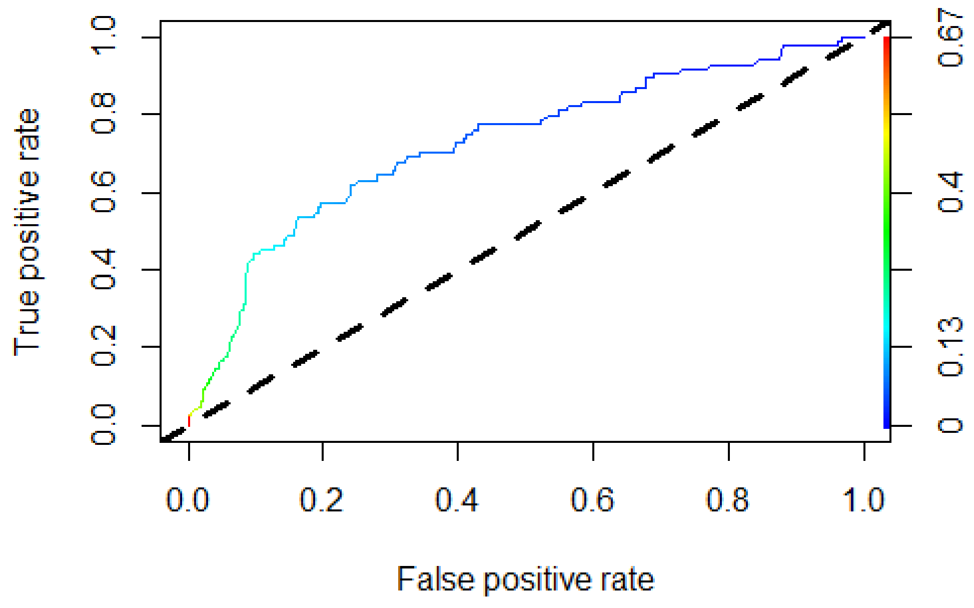
This is the test with perfect classification has a ROC curve that passes through the upper left corner. This proves that it has 100% both Sensisitivity and Specificity. Therefore the closer the ROC curve is to the upper left corner, the higher the overall accuracy of the test.

Sensitivity of a model is portion of correctly positively predicted cases (True Positives) whereas specificity is the portion of correctly negatively predicted cases(False Positives).We will further discuss on numbers which model has more sensitivity and specificity

Naive Bayes Classification:

ROC Curve for Naive Bayes

```
probabilities <- predict(modelnaive, test[-ncol(test)], type = "raw")
NBPred <- prediction(probabilities[,2], test$V86)
NBPerf <- performance(NBPred, "tpr", "fpr")
plot(NBPerf, colorize=TRUE)
abline(a=0, b=1, lty=2, lwd=3, col="black")
```



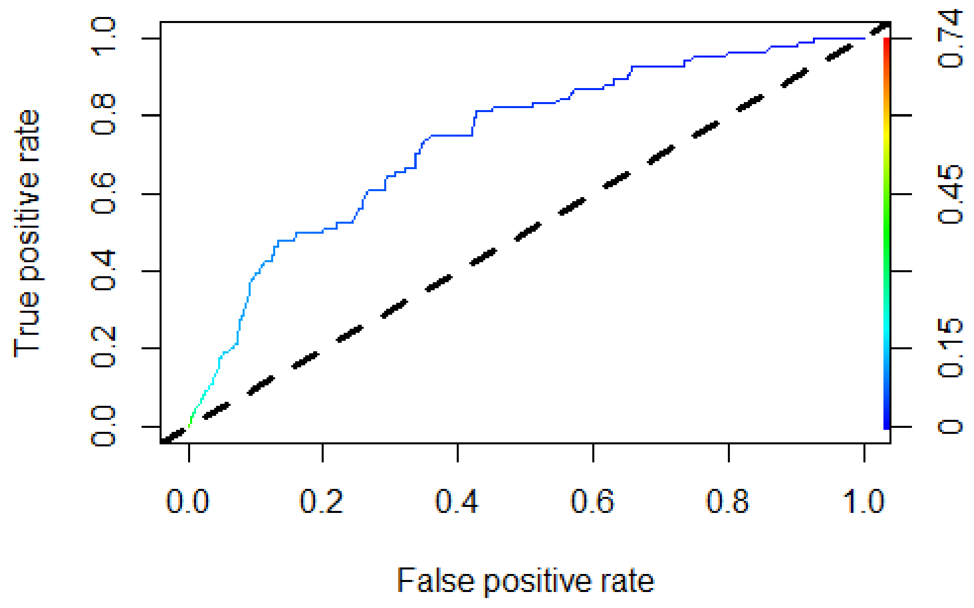
The Graph above shows ROC curve with False Positive and True Positive Cases for Naïve Bayes Classification where area under determines goodness of the model

```
# The Area Under the Curve
performance(NBPred, "auc")
```

```
## An object of class "performance"
## Slot "x.name":
## [1] "None"
##
## Slot "y.name":
## [1] "Area under the ROC curve"
##
## Slot "alpha.name":
## [1] "none"
##
## Slot "x.values":
## list()
##
## Slot "y.values":
## [[1]]
## [1] 0.7260689
##
##
## Slot "alpha.values":
## list()
```

Generarlised Linear Model

```r
probs=predict(m1, test, type = "response")
LRPred <- prediction(probs, test$V86)
LRPerf <- performance(LRPred, "tpr", "fpr")
plot(LRPerf, colorize=TRUE)
abline(a=0, b=1, lty=2, lwd=3, col="black")
```



The Graph above shows ROC curve with False Positive and True Positive Cases for GLM model where area under the curve determines goodness of the model

```r
performance(LRPred, "auc")

## An object of class "performance"
## Slot "x.name":
## [1] "None"
##
## Slot "y.name":
## [1] "Area under the ROC curve"
##
## Slot "alpha.name":
## [1] "none"
##
## Slot "x.values":
## list()
##
## Slot "y.values":
```

```
## [[1]]
## [1] 0.7379042
##
##
## Slot "alpha.values":
## list()
```

```
Naive bayes considers 805 Customers who are potential out of which 104 are
correctly predicted and hence the sensitivity the model is 43.7% .Inversely
specificity is 81.39% where as GLM considers 809 Customers who are potential
out of which it has correctly predicted 105 Customers and hence sensitiity is
44.1% and inversely Specificity is 81.2%,

ROC curve also proves the GLM has more Area Under the Curve (72.4%) than
compared to Naive Bayes (71.8%).As per this GLM is giving more accuracy.
Hence GLM would be best for the current Scenario.

However both the models are giving nearly same accuracy. Trying the same with
different training and test partition would be usefull in better selection of
model.
```

5 Summary for Marketing

```
Clearly from the Exploratory Analysis we have found that the one best
potential indicator of buying Caravan Policy is the customer who is having a
car insurance policy where contribution is level 6. Customer Subtype is hard
to find as there are few customers buying it. However EDA shows customer
subtype with level 8 and 33 are potential.


Secondly Household with three people have 50% of chances to buy the policy.
Other variables like Customer Main type, Education level, Status, Average
Income, Average Age have good correlations but in the model they are
eliminated as they might be linear/multiple-collinearity. From the model
parameters we get Contribution private third party insurance, Home Owners,
Contribution to fire policy shows significance impact on the predicting
model.

In conclusion we can say person who is having Car insurance policy /
Contribution towards it, have level 7 Purchasing power class, a private third
party insurance policy and Social security policy have high tendency to buy.

Likewise Customers who are having no trailer/tractor/life insurance policies
are intended to buy.



The Process followed for eliminating the variables is as follows,
```

Divide the Dataset into two Parts, Ordinal and Nominal Variables where both have predictors. For Ordinal variables remove Near Zero Variance and high Correlation between each other through functions defined earlier. Applied lasso reduction technique followed by Variance Inflation Checker for reducing Multi-collinearity. Then used GLM to check significance of the variables

On the other hand, for Nominal Variables I used chi-Sq test to determine p-values followed by removing inter-correlated variables through graphical methods. Then merged the two dataframes and used modeling techniques to predict the Caravan policy.