Prediction of Potential Customers for Insurance Product

Arunava Munshi

22 April 2019

Project Description:

Introduction:

The project is related to extracting knowledge from the car insurance data from an insurance company. In this project we are going to analyze the real world business data. The problem that will be solved by this project is mentioned below.

Prediction:

Need to predict the customers who would be interested to buy the car insurance product.

Explanation:

Need to explain why such prediction is necessary to understand the potential customers.

Data Set:

This project consists of three data-set.

TICDATA2000.txt:

This is the training data-set having 5822 rows and 86 columns, containing customer records. Out of the 86 columns first 43 columns are Sociology-Demographic data for the customers and rest are product ownership information. The Socio-Demographic data is derived from the zip code information that implies that areas with same zip codes have same Socio-Demographic attributes. Attribute 86: "CARAVAN" is the target variable.

TICEVAL2000.txt:

This is the test data set having 4000 rows and 85 columns. This data set does not have Attribute 86: "CARAVAN".

TICTGTS2000.txt:

This data set has the information about the customers who actually bought the car insurance, which is the basis to evaluate our prediction accuracy.

The data description can be found at: Data Description

Prediction Task:

The prediction task here will be to find a set of 800 customers most likely to buy the car insurance policy that mostly match with the customers actually bought later on as per TICTGTS2000.txt.

Reading The Data:

In this section the data sets are **ticdata2000.txt**, **ticeval2000.txt** and **tictgts2000.txt** are read into three dataframes such as **ticdata2000_df**, **ticeval2000_df** and **tictgts2000_df** respectively.

Exploratory Data Analysis:

Exploratory data analysis is required to understand the data well. Let's to some basic EDA on the training data.

Data Summary:

Let's implement the summary function to understand the training data.

```
summary(ticdata2000_df)
```

```
##        V1               V2               V3               V4
##  Min.   : 1.00   Min.   : 1.000   Min.   :1.000   Min.   :1.000
##  1st Qu.:10.00   1st Qu.: 1.000   1st Qu.:2.000   1st Qu.:2.000
##  Median :30.00   Median : 1.000   Median :3.000   Median :3.000
##  Mean   :24.25   Mean   : 1.111   Mean   :2.679   Mean   :2.991
##  3rd Qu.:35.00   3rd Qu.: 1.000   3rd Qu.:3.000   3rd Qu.:3.000
##  Max.   :41.00   Max.   :10.000   Max.   :5.000   Max.   :6.000
##        V5               V6               V7               V8
##  Min.   : 1.000   Min.   :0.0000   Min.   :0.000   Min.   :0.00
##  1st Qu.: 3.000   1st Qu.:0.0000   1st Qu.:4.000   1st Qu.:0.00
##  Median : 7.000   Median :0.0000   Median :5.000   Median :1.00
##  Mean   : 5.774   Mean   :0.6965   Mean   :4.627   Mean   :1.07
##  3rd Qu.: 8.000   3rd Qu.:1.0000   3rd Qu.:6.000   3rd Qu.:2.00
##  Max.   :10.000   Max.   :9.0000   Max.   :9.000   Max.   :5.00
##        V9              V10              V11               V12
##  Min.   :0.000   Min.   :0.000   Min.   :0.0000   Min.   :0.00
##  1st Qu.:2.000   1st Qu.:5.000   1st Qu.:0.0000   1st Qu.:1.00
##  Median :3.000   Median :6.000   Median :1.0000   Median :2.00
##  Mean   :3.259   Mean   :6.183   Mean   :0.8835   Mean   :2.29
##  3rd Qu.:4.000   3rd Qu.:7.000   3rd Qu.:1.0000   3rd Qu.:3.00
##  Max.   :9.000   Max.   :9.000   Max.   :7.0000   Max.   :9.00
##       V13              V14             V15             V16
##  Min.   :0.000   Min.   :0.00   Min.   :0.0   Min.   :0.000
##  1st Qu.:0.000   1st Qu.:2.00   1st Qu.:3.0   1st Qu.:0.000
##  Median :2.000   Median :3.00   Median :4.0   Median :1.000
##  Mean   :1.888   Mean   :3.23   Mean   :4.3   Mean   :1.461
##  3rd Qu.:3.000   3rd Qu.:4.00   3rd Qu.:6.0   3rd Qu.:2.000
```

```
##  Max.   :9.000   Max.   :9.00   Max.   :9.0   Max.   :9.000  
##       V17             V18             V19             V20       
##  Min.   :0.000   Min.   :0.000   Min.   :0.000   Min.   :0.000  
##  1st Qu.:2.000   1st Qu.:3.000   1st Qu.:0.000   1st Qu.:0.000  
##  Median :3.000   Median :5.000   Median :2.000   Median :0.000  
##  Mean   :3.351   Mean   :4.572   Mean   :1.895   Mean   :0.398  
##  3rd Qu.:4.000   3rd Qu.:6.000   3rd Qu.:3.000   3rd Qu.:1.000  
##  Max.   :9.000   Max.   :9.000   Max.   :9.000   Max.   :5.000  
##       V21              V22             V23            V24       
##  Min.   :0.0000   Min.   :0.000   Min.   :0.00   Min.   :0.000  
##  1st Qu.:0.0000   1st Qu.:2.000   1st Qu.:1.00   1st Qu.:1.000  
##  Median :0.0000   Median :3.000   Median :2.00   Median :2.000  
##  Mean   :0.5223   Mean   :2.899   Mean   :2.22   Mean   :2.306  
##  3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:3.00   3rd Qu.:3.000  
##  Max.   :9.0000   Max.   :9.000   Max.   :9.00   Max.   :9.000  
##       V25             V26             V27             V28       
##  Min.   :0.000   Min.   :0.000   Min.   :0.000   Min.   :0.000  
##  1st Qu.:0.000   1st Qu.:1.000   1st Qu.:1.000   1st Qu.:2.000  
##  Median :1.000   Median :2.000   Median :2.000   Median :4.000  
##  Mean   :1.621   Mean   :1.607   Mean   :2.203   Mean   :3.759  
##  3rd Qu.:2.000   3rd Qu.:2.000   3rd Qu.:3.000   3rd Qu.:5.000  
##  Max.   :9.000   Max.   :9.000   Max.   :9.000   Max.   :9.000  
##       V29             V30             V31             V32       
##  Min.   :0.000   Min.   :0.000   Min.   :0.000   Min.   :0.00  
##  1st Qu.:0.000   1st Qu.:2.000   1st Qu.:2.000   1st Qu.:5.00  
##  Median :1.000   Median :4.000   Median :5.000   Median :6.00  
##  Mean   :1.067   Mean   :4.237   Mean   :4.772   Mean   :6.04  
##  3rd Qu.:2.000   3rd Qu.:7.000   3rd Qu.:7.000   3rd Qu.:7.00  
##  Max.   :9.000   Max.   :9.000   Max.   :9.000   Max.   :9.00  
##       V33             V34             V35             V36       
##  Min.   :0.000   Min.   :0.000   Min.   :0.000   Min.   :0.000  
##  1st Qu.:0.000   1st Qu.:1.000   1st Qu.:5.000   1st Qu.:1.000  
##  Median :1.000   Median :2.000   Median :7.000   Median :2.000  
##  Mean   :1.316   Mean   :1.959   Mean   :6.277   Mean   :2.729  
##  3rd Qu.:2.000   3rd Qu.:3.000   3rd Qu.:8.000   3rd Qu.:4.000  
##  Max.   :7.000   Max.   :9.000   Max.   :9.000   Max.   :9.000  
##       V37             V38             V39             V40        
##  Min.   :0.000   Min.   :0.000   Min.   :0.000   Min.   :0.0000  
##  1st Qu.:1.000   1st Qu.:2.000   1st Qu.:1.000   1st Qu.:0.0000  
##  Median :2.000   Median :4.000   Median :3.000   Median :0.0000  
##  Mean   :2.574   Mean   :3.536   Mean   :2.731   Mean   :0.7961  
##  3rd Qu.:4.000   3rd Qu.:5.000   3rd Qu.:4.000   3rd Qu.:1.0000  
##  Max.   :9.000   Max.   :9.000   Max.   :9.000   Max.   :9.0000  
##       V41              V42             V43             V44        
##  Min.   :0.0000   Min.   :0.000   Min.   :1.000   Min.   :0.0000  
##  1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:3.000   1st Qu.:0.0000  
##  Median :0.0000   Median :4.000   Median :4.000   Median :0.0000  
##  Mean   :0.2027   Mean   :3.784   Mean   :4.236   Mean   :0.7712  
##  3rd Qu.:0.0000   3rd Qu.:4.000   3rd Qu.:6.000   3rd Qu.:2.0000  
##  Max.   :9.0000   Max.   :9.000   Max.   :8.000   Max.   :3.0000  
```

```
##       V45                 V46                 V47            V48         
##  Min.   :0.00000   Min.   :0.00000   Min.   :0.00   Min.   :0.00000  
##  1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.00   1st Qu.:0.00000  
##  Median :0.00000   Median :0.00000   Median :5.00   Median :0.00000  
##  Mean   :0.04002   Mean   :0.07162   Mean   :2.97   Mean   :0.04827  
##  3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:6.00   3rd Qu.:0.00000  
##  Max.   :6.00000   Max.   :4.00000   Max.   :8.00   Max.   :7.00000  
##       V49              V50                 V51                 V52         
##  Min.   :0.0000   Min.   :0.000000   Min.   :0.00000   Min.   :0.00000  
##  1st Qu.:0.0000   1st Qu.:0.000000   1st Qu.:0.00000   1st Qu.:0.00000  
##  Median :0.0000   Median :0.000000   Median :0.00000   Median :0.00000  
##  Mean   :0.1754   Mean   :0.009447   Mean   :0.02096   Mean   :0.09258  
##  3rd Qu.:0.0000   3rd Qu.:0.000000   3rd Qu.:0.00000   3rd Qu.:0.00000  
##  Max.   :7.0000   Max.   :9.000000   Max.   :5.00000   Max.   :6.00000  
##       V53                 V54             V55              V56         
##  Min.   :0.00000   Min.   :0.000   Min.   :0.0000   Min.   :0.00000  
##  1st Qu.:0.00000   1st Qu.:0.000   1st Qu.:0.0000   1st Qu.:0.00000  
##  Median :0.00000   Median :0.000   Median :0.0000   Median :0.00000  
##  Mean   :0.01305   Mean   :0.215   Mean   :0.1948   Mean   :0.01374  
##  3rd Qu.:0.00000   3rd Qu.:0.000   3rd Qu.:0.0000   3rd Qu.:0.00000  
##  Max.   :6.00000   Max.   :6.000   Max.   :9.0000   Max.   :6.00000  
##       V57                 V58                 V59             V60           
##  Min.   :0.00000   Min.   :0.00000   Min.   :0.000   Min.   :0.0000000  
##  1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.000   1st Qu.:0.0000000  
##  Median :0.00000   Median :0.00000   Median :2.000   Median :0.0000000  
##  Mean   :0.01529   Mean   :0.02353   Mean   :1.828   Mean   :0.0008588  
##  3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:4.000   3rd Qu.:0.0000000  
##  Max.   :3.00000   Max.   :7.00000   Max.   :8.000   Max.   :3.0000000  
##       V61                 V62                 V63                 V64         
##  Min.   :0.00000   Min.   :0.00000   Min.   :0.00000   Min.   :0.00000  
##  1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.00000  
##  Median :0.00000   Median :0.00000   Median :0.00000   Median :0.00000  
##  Mean   :0.01889   Mean   :0.02525   Mean   :0.01563   Mean   :0.04758  
##  3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:0.00000  
##  Max.   :6.00000   Max.   :1.00000   Max.   :6.00000   Max.   :5.00000  
##       V65             V66                 V67                 V68        
##  Min.   :0.000   Min.   :0.00000   Min.   :0.00000   Min.   :0.0000  
##  1st Qu.:0.000   1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.0000  
##  Median :0.000   Median :0.00000   Median :0.00000   Median :1.0000  
##  Mean   :0.403   Mean   :0.01477   Mean   :0.02061   Mean   :0.5622  
##  3rd Qu.:1.000   3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:1.0000  
##  Max.   :2.000   Max.   :5.00000   Max.   :1.00000   Max.   :7.0000  
##       V69                 V70                 V71                  V72         
##  Min.   :0.00000   Min.   :0.00000   Min.   :0.000000   Min.   :0.00000  
##  1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.000000   1st Qu.:0.00000  
##  Median :0.00000   Median :0.00000   Median :0.000000   Median :0.00000  
##  Mean   :0.01048   Mean   :0.04105   Mean   :0.002233   Mean   :0.01254  
##  3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:0.000000   3rd Qu.:0.00000  
##  Max.   :4.00000   Max.   :8.00000   Max.   :3.000000   Max.   :3.00000  
##       V73                 V74                  V75                 V76         
```

```
##  Min.   :0.00000   Min.   :0.000000   Min.   :0.00000   Min.   :0.00000
##  1st Qu.:0.00000   1st Qu.:0.000000   1st Qu.:0.00000   1st Qu.:0.00000
##  Median :0.00000   Median :0.000000   Median :0.00000   Median :0.00000
##  Mean   :0.03367   Mean   :0.006183   Mean   :0.07042   Mean   :0.07661
##  3rd Qu.:0.00000   3rd Qu.:0.000000   3rd Qu.:0.00000   3rd Qu.:0.00000
##  Max.   :4.00000   Max.   :6.000000   Max.   :2.00000   Max.   :8.00000
##       V77                V78                V79                V80
##  Min.   :0.000000   Min.   :0.000000   Min.   :0.000000   Min.   :0.0000
##  1st Qu.:0.000000   1st Qu.:0.000000   1st Qu.:0.000000   1st Qu.:0.0000
##  Median :0.000000   Median :0.000000   Median :0.000000   Median :1.0000
##  Mean   :0.005325   Mean   :0.006527   Mean   :0.004638   Mean   :0.5701
##  3rd Qu.:0.000000   3rd Qu.:0.000000   3rd Qu.:0.000000   3rd Qu.:1.0000
##  Max.   :1.000000   Max.   :1.000000   Max.   :2.000000   Max.   :7.0000
##       V81                 V82                V83
##  Min.   :0.0000000   Min.   :0.000000   Min.   :0.00000
##  1st Qu.:0.0000000   1st Qu.:0.000000   1st Qu.:0.00000
##  Median :0.0000000   Median :0.000000   Median :0.00000
##  Mean   :0.0005153   Mean   :0.006012   Mean   :0.03178
##  3rd Qu.:0.0000000   3rd Qu.:0.000000   3rd Qu.:0.00000
##  Max.   :1.0000000   Max.   :2.000000   Max.   :3.00000
##       V84                V85            V86
##  Min.   :0.000000   Min.   :0.00000   0:5474
##  1st Qu.:0.000000   1st Qu.:0.00000   1: 348
##  Median :0.000000   Median :0.00000
##  Mean   :0.007901   Mean   :0.01426
##  3rd Qu.:0.000000   3rd Qu.:0.00000
##  Max.   :2.000000   Max.   :2.00000
```

From the above summary information, we have the following understandings -

1. The column 1 **MOSTYPE** (Customer Subtype) has the maximum variance between 1 to 41.
2. Other than that, all other data have low variance.
3. The target variable **CARAVAN** only has two different values 0 and 1, from which we can logically conclude that either a policy is taken or not taken by a customer.

Data Types:

Now let's check the data types for each and every column.

```
str(ticdata2000_df)
```

```
## '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: Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
```

From the above data-type information, the important takeaways are as follows.

1. All the variables are integer type except for the column 86 (That we changed to factor).
2. Social Classes (Columns 25 - 29) are represented as integer, indicating that they are categorical variables.
3. Same concept goes for Income ranges (Columns 37 - 41), Columns related to Religion (Columns 6 - 9), as they seem to be categorical variables.

Unique Values:

Now Let's check the unique values of different columns.

```
lapply(ticdata2000_df, unique)

## $V1
##  [1] 33 37  9 40 23 39 11 10 41 38 22 13 31 34 24  8  7  3 36 25 20 12 35
## [24] 30 29 32  1 26  2  4 16  5 21  6 18 27 28 17 15 19
##
## $V2
## [1]  1  2  3 10  5  7  4  8  6
```

```
##
## $V3
## [1] 3 2 4 1 5
##
## $V4
## [1] 2 3 1 4 5 6
##
## $V5
##  [1]  8  3 10  5  9  7  2  1  6  4
##
## $V6
##  [1] 0 1 2 3 4 6 5 9 7 8
##
## $V7
##  [1] 5 4 3 2 7 1 6 9 0 8
##
## $V8
## [1] 1 2 0 3 4 5
##
## $V9
##  [1] 3 4 5 2 6 7 0 1 9 8
##
## $V10
##  [1] 7 6 3 5 0 1 9 8 2 4
##
## $V11
## [1] 0 2 1 6 4 3 5 7
##
## $V12
##  [1] 2 4 3 0 1 6 5 7 8 9
##
## $V13
##  [1] 1 0 4 2 3 5 6 7 8 9
##
## $V14
##  [1] 2 4 3 5 6 0 1 7 8 9
##
## $V15
##  [1] 6 5 2 4 3 1 7 9 8 0
##
## $V16
##  [1] 1 0 3 5 4 2 6 7 8 9
##
## $V17
##  [1] 2 5 4 3 1 7 6 0 8 9
##
## $V18
##  [1] 7 4 2 0 5 6 8 3 1 9
##
## $V19
```

```
##  [1] 1 0 4 2 3 6 5 7 9 8
##
## $V20
## [1] 0 5 1 2 3 4
##
## $V21
##  [1] 1 0 4 3 2 5 6 7 8 9
##
## $V22
##  [1] 2 5 7 3 0 4 1 9 6 8
##
## $V23
##  [1] 5 0 1 2 8 3 4 6 7 9
##
## $V24
##  [1] 2 4 0 5 1 3 7 6 9 8
##
## $V25
##  [1] 1 0 3 9 2 4 5 6 7 8
##
## $V26
##  [1] 1 2 5 0 3 4 8 6 9 7
##
## $V27
##  [1] 2 3 0 1 4 6 5 7 8 9
##
## $V28
##  [1] 6 5 4 0 8 1 2 7 3 9
##
## $V29
## [1] 1 0 2 5 3 4 7 6 9
##
## $V30
##  [1] 1 2 7 5 4 9 6 0 8 3
##
## $V31
##  [1] 8 7 2 4 5 0 3 9 1 6
##
## $V32
##  [1] 8 7 9 6 5 4 3 2 1 0
##
## $V33
## [1] 0 1 2 3 4 6 5 7
##
## $V34
##  [1] 1 2 0 3 4 6 5 7 8 9
##
## $V35
##  [1] 8 6 9 7 5 4 3 1 2 0
##
```

```
## $V36
##  [1] 1 3 0 2 4 5 6 8 7 9
##
## $V37
##  [1] 0 2 4 1 5 7 3 9 6 8
##
## $V38
##  [1] 4 0 5 2 3 1 6 7 8 9
##
## $V39
##  [1] 5 0 3 9 1 2 4 6 8 7
##
## $V40
##  [1] 0 2 1 4 3 5 9 6 8 7
##
## $V41
## [1] 0 2 1 3 4 9 5 7
##
## $V42
##  [1] 4 5 3 6 2 8 1 7 9 0
##
## $V43
## [1] 3 4 5 7 2 6 1 8
##
## $V44
## [1] 0 2 1 3
##
## $V45
## [1] 0 1 3 4 2 6 5
##
## $V46
## [1] 0 3 4 2
##
## $V47
## [1] 6 0 5 7 8 4
##
## $V48
## [1] 0 5 6 7
##
## $V49
## [1] 0 4 5 6 7 3
##
## $V50
## [1] 0 6 4 9
##
## $V51
## [1] 0 2 1 3 5 4
##
## $V52
## [1] 0 3 5 4 6
```

```
##
## $V53
## [1] 0 2 6 4 3
##
## $V54
## [1] 0 3 2 4 5 6
##
## $V55
##  [1] 0 4 3 2 5 9 1 7 6 8
##
## $V56
## [1] 0 2 4 1 5 6 3
##
## $V57
## [1] 0 2 3
##
## $V58
## [1] 0 6 4 7 5
##
## $V59
## [1] 5 2 6 0 3 4 1 7 8
##
## $V60
## [1] 0 1 3
##
## $V61
## [1] 0 4 1 5 2 6 3
##
## $V62
## [1] 0 1
##
## $V63
## [1] 0 2 1 3 6 5 4
##
## $V64
## [1] 0 4 3 2 5
##
## $V65
## [1] 0 2 1
##
## $V66
## [1] 0 1 5
##
## $V67
## [1] 0 1
##
## $V68
## [1] 1 0 2 7 3 6 4
##
## $V69
```

```
## [1] 0 1 4 2 3
##
## $V70
## [1] 0 1 8 2
##
## $V71
## [1] 0 1 2 3
##
## $V72
## [1] 0 1 2 3
##
## $V73
## [1] 0 1 2 3 4
##
## $V74
## [1] 0 1 3 2 6
##
## $V75
## [1] 0 1 2
##
## $V76
## [1] 0 1 2 3 8 4
##
## $V77
## [1] 0 1
##
## $V78
## [1] 0 1
##
## $V79
## [1] 0 1 2
##
## $V80
## [1] 1 0 2 3 5 4 7
##
## $V81
## [1] 0 1
##
## $V82
## [1] 0 2 1
##
## $V83
## [1] 0 1 2 3
##
## $V84
## [1] 0 1 2
##
## $V85
## [1] 0 1 2
##
```

```
## $V86
## [1] 0 1
## Levels: 0 1
```

We have the below observations from the above output.

1. There is no null values, NaN or missing values in any column, indicating that the data is possibly properly cleaned up or wrangled already.
2. Column 1 (MOSTYPE) has most of the unique values from 1-41.
3. We also see that every columns from 44 - 64 that are listed as **“Contribution to policies”** being integer indicates that they are categorical variables.
4. We also notice that columns 65-85 are number of policies, which should not be categorical.
5. Within the socio-demographic variables, incomes, religion etc. being integers indicates that they are categorical variables.

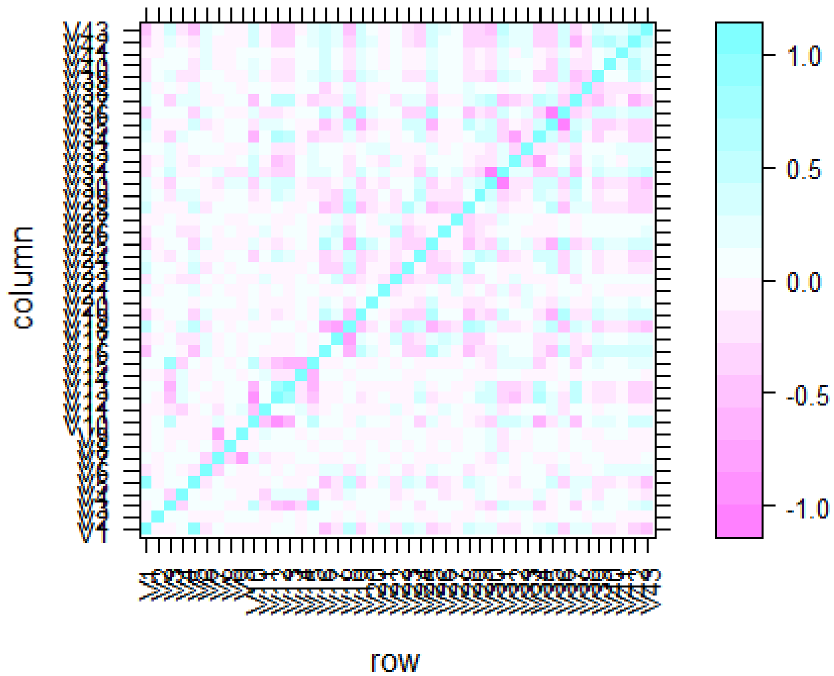
Correlation Among Variables:

Now Let’s try to find the correlation or level of association among variables.

Correlation Among Socio-DemoGraphic Variables:

Let’s try to find the correlation among Sociology-Demographic Variables.

```
library(lattice)
library(dplyr)

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:data.table':
##
##     between, first, last

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

levelplot(cor(select(ticdata2000_df, c(V1:V43))), scales = list(x = list(rot
= 90)))
```

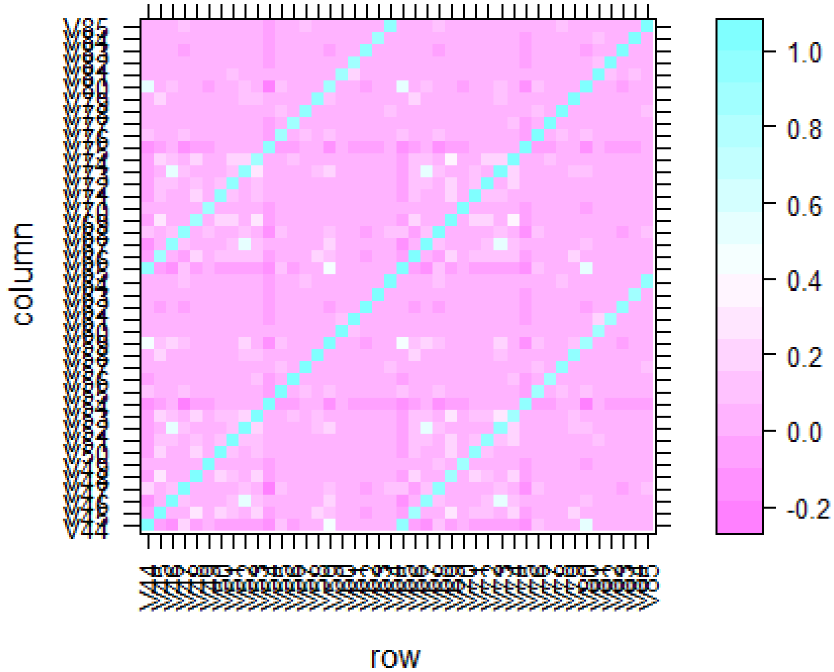


As we can see here, there is no significant correlations between any two variables. So we can ignore this.

Correlation Among Product Ownership Variables:

Let's now try to find the correlation among product ownership Variables.

```
library(lattice)
levelplot(cor(select(ticdata2000_df, c(V44:V85))), scales = list(x = list(rot
= 90)))
```



Now, here we find the correlation between few variables such as between 44-65, 45-66, 47-68 and so on to 64-85. So we can see that Contribution of respective insurance policies are strongly correlated to the corresponding number of those policies, which is logically correct. The Pearson coefficients are shown below -

```
cor(ticdata2000_df$V44,ticdata2000_df$V65)

## [1] 0.9813692

cor(ticdata2000_df$V45,ticdata2000_df$V66)

## [1] 0.8954072

cor(ticdata2000_df$V46,ticdata2000_df$V67)

## [1] 0.9875786

cor(ticdata2000_df$V47,ticdata2000_df$V68)

## [1] 0.9161545

cor(ticdata2000_df$V48,ticdata2000_df$V69)

## [1] 0.9029956

cor(ticdata2000_df$V49,ticdata2000_df$V70)

## [1] 0.9048552
```

```r
cor(ticdata2000_df$V50,ticdata2000_df$V71)

## [1] 0.9486633

cor(ticdata2000_df$V51,ticdata2000_df$V72)

## [1] 0.9660805

cor(ticdata2000_df$V52,ticdata2000_df$V73)

## [1] 0.9298178

cor(ticdata2000_df$V53,ticdata2000_df$V74)

## [1] 0.9096707

cor(ticdata2000_df$V54,ticdata2000_df$V75)

## [1] 0.9697076

cor(ticdata2000_df$V55,ticdata2000_df$V76)

## [1] 0.8501711

cor(ticdata2000_df$V56,ticdata2000_df$V77)

## [1] 0.8975617

cor(ticdata2000_df$V57,ticdata2000_df$V78)

## [1] 0.9799685

cor(ticdata2000_df$V58,ticdata2000_df$V79)

## [1] 0.9484299

cor(ticdata2000_df$V59,ticdata2000_df$V80)

## [1] 0.8655359

cor(ticdata2000_df$V60,ticdata2000_df$V81)

## [1] 0.8703339

cor(ticdata2000_df$V61,ticdata2000_df$V82)

## [1] 0.9044364

cor(ticdata2000_df$V62,ticdata2000_df$V83)

## [1] 0.9358543

cor(ticdata2000_df$V63,ticdata2000_df$V84)

## [1] 0.8752565
```

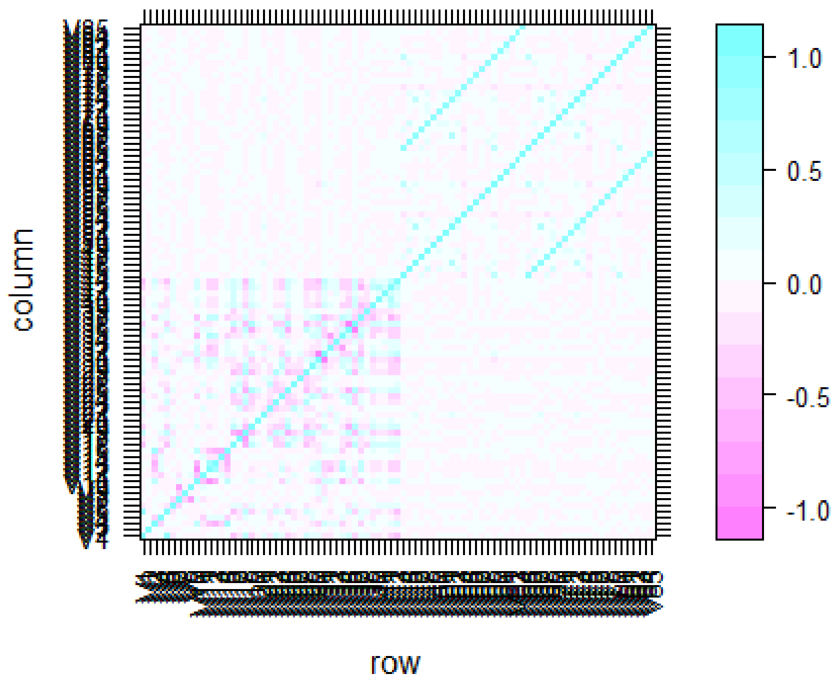
```r
cor(ticdata2000_df$V64,ticdata2000_df$V85)

## [1] 0.9662388
```

So, it is proven that these variables are highly correlated among each other.

Correlation Among Socio-Demographic and Product Ownership Variables:

If we plot the same graph for the whole dataframe, we don't find any correlation between Socio-Demographic and Product Ownership set. So we can discard any possibility of such relationship.

```r
library(lattice)
levelplot(cor(ticdata2000_df[,-86]), scales = list(x = list(rot = 90)))
```



The above statement is proven from the above plot. So we may ignore any further possibility of revisiting these variables.

Variable Interactions:

Now it comes to the initial interactions, we need to consider the interactions among the predictors and the interactions between the predictors and the target. We will do that in two steps.

Evaluating Interactions among the Product Ownership Variables:

Because we know that there are strong correlation among some of the product ownership variables mentioned earlier, we need to check if there could be an interaction among these variables through any transformation. Logically speaking, Contribution of each insurance policy should be multiplied with its number to realize the best effect (I have checked the other interactions such as addition, subtraction etc. and no other transformations worked as good as multiplication.). The interactions are mentioned below and multiplications are added into the new columns of the dataframe **ticdata2000_df**.

```
ticdata2000_df$V44V65 <- with(ticdata2000_df, V44 * V65)
ticdata2000_df$V45V66 <- with(ticdata2000_df, V45 * V66)
ticdata2000_df$V46V67 <- with(ticdata2000_df, V46 * V67)
ticdata2000_df$V47V68 <- with(ticdata2000_df, V47 * V68)
ticdata2000_df$V48V69 <- with(ticdata2000_df, V48 * V69)
ticdata2000_df$V49V70 <- with(ticdata2000_df, V49 * V70)
ticdata2000_df$V50V71 <- with(ticdata2000_df, V50 * V71)
ticdata2000_df$V51V72 <- with(ticdata2000_df, V51 * V72)
ticdata2000_df$V52V73 <- with(ticdata2000_df, V52 * V73)
ticdata2000_df$V53V74 <- with(ticdata2000_df, V53 * V74)
ticdata2000_df$V54V75 <- with(ticdata2000_df, V54 * V75)
ticdata2000_df$V55V76 <- with(ticdata2000_df, V55 * V76)
ticdata2000_df$V56V77 <- with(ticdata2000_df, V56 * V77)
ticdata2000_df$V57V78 <- with(ticdata2000_df, V57 * V78)
ticdata2000_df$V58V79 <- with(ticdata2000_df, V58 * V79)
ticdata2000_df$V59V80 <- with(ticdata2000_df, V59 * V80)
ticdata2000_df$V60V81 <- with(ticdata2000_df, V60 * V81)
ticdata2000_df$V61V82 <- with(ticdata2000_df, V61 * V82)
ticdata2000_df$V62V83 <- with(ticdata2000_df, V62 * V83)
ticdata2000_df$V63V84 <- with(ticdata2000_df, V63 * V84)
ticdata2000_df$V64V85 <- with(ticdata2000_df, V64 * V85)
```

Chi Square Test on All the Variables and Their Interactions:

Now once the correlations are known among the predictors, it's the time to identify the relation between each predictors & their interactions and target variable. The best way to find the relation is to do a Chi Square test. Why this is best? Chi Square test between two potential variables first seeks to reject a null hypothesis "There is no relationship between two variables". It works best for categorical variables. We already know that the target variable CARAVAN is a categorical variable and from the previous analysis we came to know that no other table column is continuous. So we can conclude that a Chi Square test should be wise decision here.

```
CHI <- function(sppx, sppy)
{test <- chisq.test(sppx, sppy, correct=FALSE)
return(test)
}
colnames_vec = c()
count = 0
```

```
for (colnames in colnames(ticdata2000_df)){
    test <- CHI(ticdata2000_df[colnames], ticdata2000_df$V86)
    if (test[3] < 0.01){
    colnames_vec <- c(colnames_vec, colnames)
    count = count + 1
        }
}
print(colnames_vec)

##  [1] "V1"     "V5"     "V7"     "V10"    "V12"    "V16"    "V18"
##  [8] "V19"    "V22"    "V23"    "V24"    "V25"    "V28"    "V29"
## [15] "V30"    "V31"    "V32"    "V34"    "V35"    "V36"    "V37"
## [22] "V39"    "V40"    "V42"    "V43"    "V44"    "V47"    "V49"
## [29] "V57"    "V59"    "V61"    "V64"    "V65"    "V68"    "V75"
## [36] "V76"    "V80"    "V82"    "V85"    "V86"    "V44V65" "V47V68"
## [43] "V49V70" "V57V78" "V59V80" "V61V82" "V64V85"
```

The above code conducts a Chi Square test between every predictor or every interaction and the target variable. The significance level for the test is set to 0.01, means with 99% confidence, the null hypothesis is attempted to be rejected. Because we want to consider as much information relevant to the target, so we are becoming a bit more conservative. The variables which are related to the target have been mentioned in the summary.

```
length(colnames_vec)

## [1] 47
```

So there are 47 such variables. All these variables are put into a separate dataframe named **ticdata2000_df_subset1**. The test dataframe **ticeval2000_df** is also updated with the new features.

```
library(dplyr)
ticdata2000_df_subset1 <- select(ticdata2000_df, colnames_vec)
colnames(ticdata2000_df_subset1)

##  [1] "V1"     "V5"     "V7"     "V10"    "V12"    "V16"    "V18"
##  [8] "V19"    "V22"    "V23"    "V24"    "V25"    "V28"    "V29"
## [15] "V30"    "V31"    "V32"    "V34"    "V35"    "V36"    "V37"
## [22] "V39"    "V40"    "V42"    "V43"    "V44"    "V47"    "V49"
## [29] "V57"    "V59"    "V61"    "V64"    "V65"    "V68"    "V75"
## [36] "V76"    "V80"    "V82"    "V85"    "V86"    "V44V65" "V47V68"
## [43] "V49V70" "V57V78" "V59V80" "V61V82" "V64V85"

head(ticdata2000_df_subset1)

##   V1 V5 V7 V10 V12 V16 V18 V19 V22 V23 V24 V25 V28 V29 V30 V31 V32 V34 V35
## 1 33  8  5   7   2   1   7   1   2   5   2   1   6   1   1   8   8   1   8
## 2 37  8  4   6   2   0   4   0   5   0   4   0   5   0   2   7   7   2   6
## 3 37  8  4   3   4   0   4   0   7   0   2   0   4   0   7   2   7   2   9
## 4  9  3  3   5   2   3   2   4   3   1   2   3   4   0   5   4   9   0   7
## 5 40 10  4   7   2   5   0   0   0   0   0   9   0   0   4   5   6   1   5
```

```
## 6 23  5  5   0   3   0   4   2   4   2   2   2   4   2   9   0   5   3   9
##   V36 V37 V39 V40 V42 V43 V44 V47 V49 V57 V59 V61 V64 V65 V68 V75 V76 V80
## 1   1   0   5   0   4   3   0   6   0   0   5   0   0   0   1   0   0   1
## 2   3   2   5   2   5   4   2   0   0   0   2   0   0   2   0   0   0   1
## 3   0   4   0   0   3   4   2   6   0   0   2   0   0   1   1   0   0   1
## 4   2   1   3   0   4   4   0   6   0   0   2   0   0   0   1   0   0   1
## 5   4   0   9   0   6   3   0   0   0   0   6   0   0   0   0   0   0   1
## 6   0   5   3   0   3   3   0   6   0   0   0   0   0   0   1   0   0   0
##   V82 V85 V86 V44V65 V47V68 V49V70 V57V78 V59V80 V61V82 V64V85
## 1   0   0   0      0      6      0      0      5      0      0
## 2   0   0   0      4      0      0      0      2      0      0
## 3   0   0   0      2      6      0      0      2      0      0
## 4   0   0   0      0      6      0      0      2      0      0
## 5   0   0   0      0      0      0      0      6      0      0
## 6   0   0   0      0      6      0      0      0      0      0
```

```
ticeval2000_df$V44V65 <- with(ticeval2000_df, V44 * V65)
ticeval2000_df$V47V68 <- with(ticeval2000_df, V47 * V68)
ticeval2000_df$V49V70 <- with(ticeval2000_df, V49 * V70)
ticeval2000_df$V57V78 <- with(ticeval2000_df, V57 * V78)
ticeval2000_df$V59V80 <- with(ticeval2000_df, V59 * V80)
ticeval2000_df$V61V82 <- with(ticeval2000_df, V61 * V82)
ticeval2000_df$V64V85 <- with(ticeval2000_df, V64 * V85)

head(ticeval2000_df)
```

```
##   V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 V15 V16 V17 V18 V19 V20
## 1 33  1  4  2  8  0  6  0  3   5   0   4   1   1   8   2   2   6   0   0
## 2  6  1  3  2  2  0  5  0  4   5   2   2   1   4   5   5   4   0   5   0
## 3 39  1  3  3  9  1  4  2  3   5   2   3   2   3   6   2   4   4   2   1
## 4  9  1  2  3  3  2  3  2  4   5   4   1   2   4   4   2   4   4   2   1
## 5 31  1  2  4  7  0  2  0  7   9   0   0   0   6   3   0   0   9   0   0
## 6 30  1  2  4  7  1  4  2  3   5   0   4   4   3   2   1   2   6   1   0
##   V21 V22 V23 V24 V25 V26 V27 V28 V29 V30 V31 V32 V33 V34 V35 V36 V37 V38
## 1   1   2   6   1   0   2   1   5   3   1   8   8   1   1   8   1   3   3
## 2   0   4   0   0   4   3   0   2   1   3   6   9   0   0   7   2   1   1
## 3   1   3   2   2   1   1   5   2   1   1   8   6   2   2   6   3   2   4
## 4   1   5   1   2   3   1   3   2   2   3   6   7   2   1   7   2   2   5
## 5   0   2   4   4   0   0   0   7   2   9   0   7   2   0   9   0   5   4
## 6   1   3   3   3   1   1   2   5   1   5   4   5   1   4   9   0   2   5
##   V39 V40 V41 V42 V43 V44 V45 V46 V47 V48 V49 V50 V51 V52 V53 V54 V55 V56
## 1   3   0   0   3   3   1   0   0   0   0   0   0   0   0   0   0   0   0
## 2   5   4   0   6   8   2   0   0   6   0   4   0   0   0   0   0   3   0
## 3   3   1   0   3   5   2   0   0   6   0   0   0   0   0   0   0   4   0
## 4   3   1   0   4   4   2   0   0   5   0   0   0   0   0   0   0   0   0
## 5   0   0   0   3   1   2   0   0   0   0   0   0   0   0   0   0   0   0
## 6   2   1   0   4   2   0   0   0   0   0   0   0   0   0   0   0   0   0
##   V57 V58 V59 V60 V61 V62 V63 V64 V65 V66 V67 V68 V69 V70 V71 V72 V73 V74
## 1   0   0   4   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0
## 2   0   0   4   0   0   0   0   0   1   0   0   1   0   1   0   0   0   0
```

```
## 3   0   0   4   0   0   0   0   0   1   0   0   1   0   0   0   0   0   0
## 4   0   0   3   0   0   0   0   0   1   0   0   1   0   0   0   0   0   0
## 5   0   0   1   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0
## 6   0   0   4   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##    V75 V76 V77 V78 V79 V80 V81 V82 V83 V84 V85 V44V65 V47V68 V49V70 V57V78
## 1   0   0   0   0   0   1   0   0   0   0   0      1      0      0      0
## 2   0   2   0   0   0   1   0   0   0   0   0      2      6      4      0
## 3   0   1   0   0   0   1   0   0   0   0   0      2      6      0      0
## 4   0   0   0   0   0   1   0   0   0   0   0      2      5      0      0
## 5   0   0   0   0   0   1   0   0   0   0   0      2      0      0      0
## 6   0   0   0   0   0   2   0   0   0   0   0      0      0      0      0
##    V59V80 V61V82 V64V85
## 1       4      0      0
## 2       4      0      0
## 3       4      0      0
## 4       3      0      0
## 5       1      0      0
## 6       8      0      0
```

Feature Selection:

After we get an idea on which are the potential variables which could finally affect the target variable from the Chi square Test, it's time to do the Feature Selection. Feature selection is more insightful and nuanced technique to pick up the potential predictors. Feature Selection picks up the features for a target variable through some standardized algorithm. For the case of this project, we examine some famous feature selection techniques.

1. Recursive Feature Elimination
2. Forward Subset Selection
3. Backward Subset Selection
4. Lasso

We also have another feature selection named **"Best Subset Selection"** Algorithm. But because this algorithm makes

$$2^p$$

iterations and any number of iterations more than

$$2^{10}$$

is not advisable, so it is not practical for 47 predictors.

Recursive Feature Elimination (RFE)

RFE is very efficient feature selection method that recursively removes the weakest features. The features are ranked according to the attributes of the model and it attempts to eliminate some features in every loop. The best thing with this algorithm is that it also handles col-linearity among the predictors itself. But one disadvantage of RFE is that it is computationally expensive because we may have to train increasing number of models. The below code is for RFE for "Naive Bayes" (We can select ML techniques in RFE as well, this is another example) below, that we will use later on during model building. For the purpose of this project we are considering only 1st 10 ranked variables thrown by RFE.

```
library(caret)

## Loading required package: ggplot2

subsets <- c(10, 20)
drops <- c("V86")
ticdata2000_df_subset1$V86 <- factor(ticdata2000_df_subset1$V86)
ctrl <- rfeControl(functions = nbFuncs,
                   method = "cv",
                   number = 10,
                   verbose = FALSE)

nbProfile <- rfe(ticdata2000_df_subset1[, !(names(ticdata2000_df_subset1) %in
% drops)], ticdata2000_df_subset1$V86,
                 sizes = subsets,
                 rfeControl = ctrl)

choose_colm_RFE <- nbProfile$optVariables[1:10]
choose_colm_RFE

##  [1] "V47V68" "V47"    "V68"    "V42"    "V59"    "V43"    "V59V80"
##  [8] "V18"    "V44V65" "V44"
```

The above output shows the best 10 predictors as per the algorithm.

Forward Subset Selection (FSS)

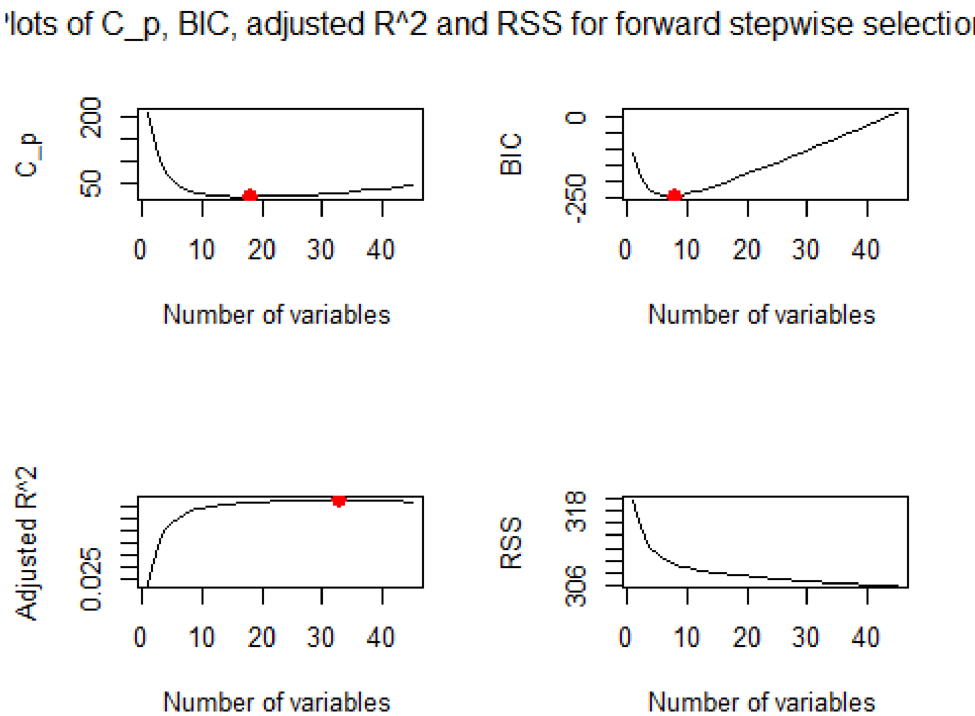
FSS is a feature selection technique that recursively selects feature starting from just one feature. In the 1st iteration the algorithm adds every other predictors to itself and check and compares the accuracy of these newly created models and keeps the best one in that iteration. The next iteration will start with the best model selected in the previous iteration and will add each of remaining predictor one by one and evaluate the each models one by one and again keep the best one among then. This process will keep on going until there is no predictor remaining. The advantage of this method is that it is computationally very efficient and takes very less time for large number of predictors. But, there are few disadvantages of this method -

1. It is greedy method, so it can lead to over-fitting sometimes.
2. It cannot discard any variable selected previously and is somewhat conservative.
3. It cannot handle multi-collinearity.

This algorithm gives penalty to the bias.The code is written below.

```
library(leaps)
```

```r
regfit.fwd <- regsubsets(V86 ~ ., data = ticdata2000_df_subset1,
                                  nvmax = NULL, method = "forward")

## Reordering variables and trying again:

reg.summary.fwd <- summary(regfit.fwd)

par(mfrow = c(2, 2))

plot(reg.summary.fwd$cp, xlab = "Number of variables",
                         ylab = "C_p", type = "l")

points(which.min(reg.summary.fwd$cp),          reg.summary.fwd$cp[which.min(r
eg.summary.fwd$cp)],
       col = "red", cex = 2, pch = 20)

plot(reg.summary.fwd$bic, xlab = "Number of variables",
                          ylab = "BIC", type = "l")

points(which.min(reg.summary.fwd$bic), reg.summary.fwd$bic[which.min(reg.summ
ary.fwd$bic)],
       col = "red", cex = 2, pch = 20)

plot(reg.summary.fwd$adjr2, xlab = "Number of variables",
                            ylab = "Adjusted R^2", type = "l")

points(which.max(reg.summary.fwd$adjr2), reg.summary.fwd$adjr2[which.max(reg.
summary.fwd$adjr2)],
       col = "red", cex = 2, pch = 20)

plot(reg.summary.fwd$rss, xlab = "Number of variables",
                          ylab = "RSS", type = "l")

mtext("Plots of C_p, BIC, adjusted R^2 and RSS for forward stepwise selection
", side = 3, line = -2, outer = TRUE)
```



We have evaluated the model against four measurement criteria.

$$R^2$$

is the amount of variation in the model outcome that are being explained by the predictors. BIC is a measure of penalty for inclusion of new feature using Bayesian technique. C_P is is also another measure of penalty for inclusion of new feature.

$$R^2$$

Adjusted is a variation

$$R^2$$

which gives penalty on feature inclusion. The red points on the above grasp show the optimal number of features using each measurement criteria. It highest number of optimal features is advised by

$$R^2$$

Adjusted and we are keeping this number in order to avoid any feature miss, which may otherwise affects prediction accuracy of final model.

The optimal features selected by

$$R^2$$

Adjusted is shown below.

```r
print("Number of Optimal Coefficients by Adjusted R^2: ")

## [1] "Number of Optimal Coefficients by Adjusted R^2: "

max_adjr2_fwd = which.max(reg.summary.fwd$adjr2)
print(max_adjr2_fwd)

## [1] 33

print("The Features selected: ")

## [1] "The Features selected: "

coef(regfit.fwd, max_adjr2_fwd)

##  (Intercept)           V7          V10          V12          V16
##  1.782384873  0.003836875  0.008906678  0.006046679  0.005913736
##          V18          V19          V22          V23          V24
## -0.004764874  0.005687063  0.006698985  0.002453586  0.004762263
##          V28          V30          V31          V32          V35
##  0.003845645 -0.049909418 -0.048821151  0.003432252 -0.055524663
##          V36          V40          V42          V43          V44
## -0.058629741  0.002457609  0.002746280  0.002968700  0.063499890
##          V47          V57          V59          V61          V64
##  0.011355921  0.027283274  0.009767043 -0.016270429  0.092657925
##          V65          V68          V80          V82          V85
## -0.032950536 -0.094432907 -0.016827072  0.327712702  0.223899581
##       V44V65       V47V68       V64V85       V57V78
## -0.033865100  0.014769022 -0.136229917  0.000000000

choose_colm_FSS <- c("V7", "V10", "V12", "V16", "V18", "V19", "V22", "V23", "
V24", "V28", "V30", "V31", "V32", "V35", "V36", "V40", "V42", "V43", "V59", "
V61", "V64", "V80", "V82", "V85", "V44V65", "V47V68", "V64V85", "V57V78")
print("Final Features selected: ")

## [1] "Final Features selected: "

print(choose_colm_FSS)

##  [1] "V7"     "V10"    "V12"    "V16"    "V18"    "V19"    "V22"   
##  [8] "V23"    "V24"    "V28"    "V30"    "V31"    "V32"    "V35"   
## [15] "V36"    "V40"    "V42"    "V43"    "V59"    "V61"    "V64"   
## [22] "V80"    "V82"    "V85"    "V44V65" "V47V68" "V64V85" "V57V78"

print("Final Length: ")

## [1] "Final Length: "

length(choose_colm_FSS)

## [1] 28
```

So I can see that, Adjusted

$$R^2$$

Chooses 33 variables but, out of them some are interactions which indicates that the independent variables of those interactions can be removed. So the final set of variables from

$$R^2$$

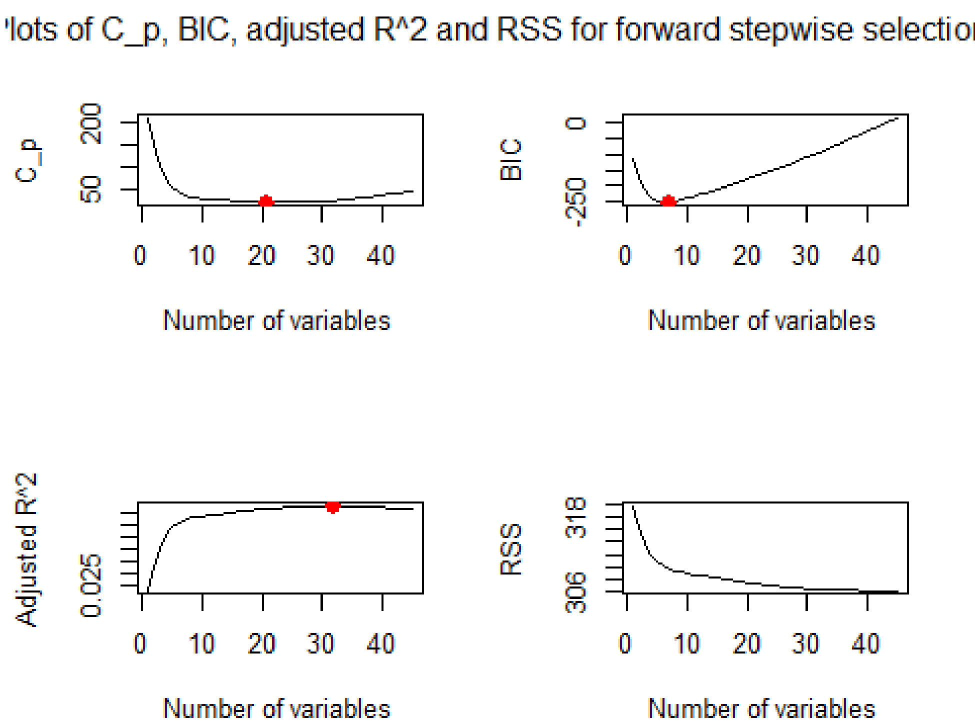
Adjusted are shown finally, the count of which is 28.

Backward Subset Selection (BSS)

The BSS algorithm is almost similar to Forward Selection but only difference is that the algo starts with all features at first and iteratively removes one by one in the same way FSS does it from the front. The advantages and disadvantages are similar as of FSS. The code for the same is given below.

```
library(leaps)
regfit.bwd <- regsubsets(V86 ~ ., data = ticdata2000_df_subset1,
                                  nvmax = NULL, method = "backward")

## Warning in leaps.setup(x, y, wt = wt, nbest = nbest, nvmax = nvmax,
## force.in = force.in, : 1 linear dependencies found

## Reordering variables and trying again:

## Warning in rval$lopt[] <- rval$vorder[rval$lopt]: number of items to
## replace is not a multiple of replacement length

reg.summary.bwd <- summary(regfit.bwd)

par(mfrow = c(2, 2))

plot(reg.summary.bwd$cp, xlab = "Number of variables",
                        ylab = "C_p", type = "l")

points(which.min(reg.summary.bwd$cp), reg.summary.bwd$cp[which.min(reg.summar
y.bwd$cp)],
                col = "red", cex = 2, pch = 20)

plot(reg.summary.bwd$bic, xlab = "Number of variables",
                          ylab = "BIC", type = "l")

points(which.min(reg.summary.bwd$bic), reg.summary.bwd$bic[which.min(reg.summ
ary.bwd$bic)],
                 col = "red", cex = 2, pch = 20)

plot(reg.summary.bwd$adjr2, xlab = "Number of variables",
                            ylab = "Adjusted R^2", type = "l")

points(which.max(reg.summary.bwd$adjr2), reg.summary.bwd$adjr2[which.max(reg.
```

```
summary.bwd$adjr2)],
                   col = "red", cex = 2, pch = 20)

plot(reg.summary.bwd$rss, xlab = "Number of variables",
                            ylab = "RSS", type = "l")

mtext("Plots of C_p, BIC, adjusted R^2 and RSS for forward stepwise selection
", side = 3, line = -2, outer = TRUE)
```



Again, on the same ground we go for the features selected by the

R^2

Adjusted. The optimal features selected by

R^2

Adjusted is shown below.

```
print("Number of Optimal Coefficients by Adjusted R^2: ")

## [1] "Number of Optimal Coefficients by Adjusted R^2: "

max_adjr2_bwd = which.max(reg.summary.bwd$adjr2)
print(max_adjr2_bwd)

## [1] 32

print("The Features selected: ")
```

```
## [1] "The Features selected: "
```

```r
coef(regfit.bwd, max_adjr2_bwd)
```

```
##  (Intercept)           V1           V5           V7          V10 
##  1.819206814  0.004306614 -0.019607208  0.003659493  0.009746566 
##          V12          V16          V18          V19          V22 
##  0.006111257  0.005730977 -0.004589974  0.004233635  0.004802613 
##          V24          V28          V30          V31          V32 
##  0.003217243  0.004427362 -0.053232386 -0.051911082  0.003321400 
##          V35          V36          V42          V43          V44 
## -0.054791804 -0.058072799  0.003920756  0.003567282  0.052360919 
##          V47          V57          V59          V61          V64 
##  0.011427275  0.027795504  0.010322498 -0.016283343  0.095344966 
##          V68          V80          V82          V85       V44V65 
## -0.092996478 -0.018902274  0.325625991  0.228096705 -0.039517549 
##       V47V68       V64V85       V57V78 
##  0.014508789 -0.140103702  0.000000000
```

```r
choose_colm_BSS <- c("V1", "V5", "V7", "V10", "V12", "V16", "V18", "V19", "V2
2", "V24", "V28", "V30", "V31", "V32", "V35", "V36", "V42", "V43", "V57", "V5
9", "V61", "V80", "V82", "V44V65", "V47V68", "V64V85", "V57V78")
print("Final Features selected: ")
```

```
## [1] "Final Features selected: "
```

```r
print(choose_colm_BSS)
```

```
##  [1] "V1"     "V5"     "V7"     "V10"    "V12"    "V16"    "V18"   
##  [8] "V19"    "V22"    "V24"    "V28"    "V30"    "V31"    "V32"   
## [15] "V35"    "V36"    "V42"    "V43"    "V57"    "V59"    "V61"   
## [22] "V80"    "V82"    "V44V65" "V47V68" "V64V85" "V57V78"
```

```r
print("Final Length: ")
```

```
## [1] "Final Length: "
```

```r
length(choose_colm_BSS)
```

```
## [1] 27
```

So I can see that, Adjusted

$$R^2$$

Chooses 32 variables but, out of them some are interactions which indicates that the independent variables of those interactions can be removed. So the final set of variables from

$$R^2$$

Adjusted are shown finally, the count of which is 27.

Lasso

Lasso is a shrinkage technique that tries to do feature selection by shrinking the regression coefficients towards zero by adding a penalty expression on variance. While Subset selection methods penalizes bias, Lasso concentrates into variance. Lasso has an advantage over subset selection and that is, it never concentrates into eliminating predictors to make a better fit. But it has some disadvantages too. 1. Because it does not concentrate into model fitting, it sometimes leads to under-fitting. 2. It cannot also handle multi-colinearity.

```
str(ticdata2000_df_subset1)
```

```
## 'data.frame':    5822 obs. of  47 variables:
##  $ V1   : int  33 37 37 9 40 23 39 33 33 11 ...
##  $ V5   : int  8 8 8 3 10 5 9 8 8 3 ...
##  $ V7   : int  5 4 4 3 4 5 2 7 1 5 ...
##  $ V10  : int  7 6 3 5 7 0 7 7 6 7 ...
##  $ V12  : int  2 2 4 2 2 3 0 0 3 2 ...
##  $ V16  : int  1 0 0 3 5 0 0 0 0 0 ...
##  $ V18  : int  7 4 4 2 0 4 5 6 8 5 ...
##  $ V19  : int  1 0 0 4 0 2 0 2 1 2 ...
##  $ 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 ...
##  $ 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 ...
##  $ 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 ...
##  $ V39  : int  5 5 0 3 9 3 3 3 1 3 ...
##  $ V40  : int  0 2 0 0 0 0 0 0 0 1 ...
##  $ 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 ...
##  $ V47  : int  6 0 6 6 0 6 6 0 5 0 ...
##  $ V49  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V57  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V59  : int  5 2 2 2 6 0 0 0 0 3 ...
##  $ V61  : 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 ...
##  $ V68  : int  1 0 1 1 0 1 1 0 1 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 ...
##  $ V80  : int  1 1 1 1 1 0 0 0 0 1 ...
##  $ V82  : int  0 0 0 0 0 0 0 0 0 0 ...
```

```
##  $ V85   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V86   : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ V44V65: int  0 4 2 0 0 0 0 0 0 2 ...
##  $ V47V68: int  6 0 6 6 0 6 6 0 5 0 ...
##  $ V49V70: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V57V78: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V59V80: int  5 2 2 2 6 0 0 0 0 3 ...
##  $ V61V82: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V64V85: int  0 0 0 0 0 0 0 0 0 0 ...
```

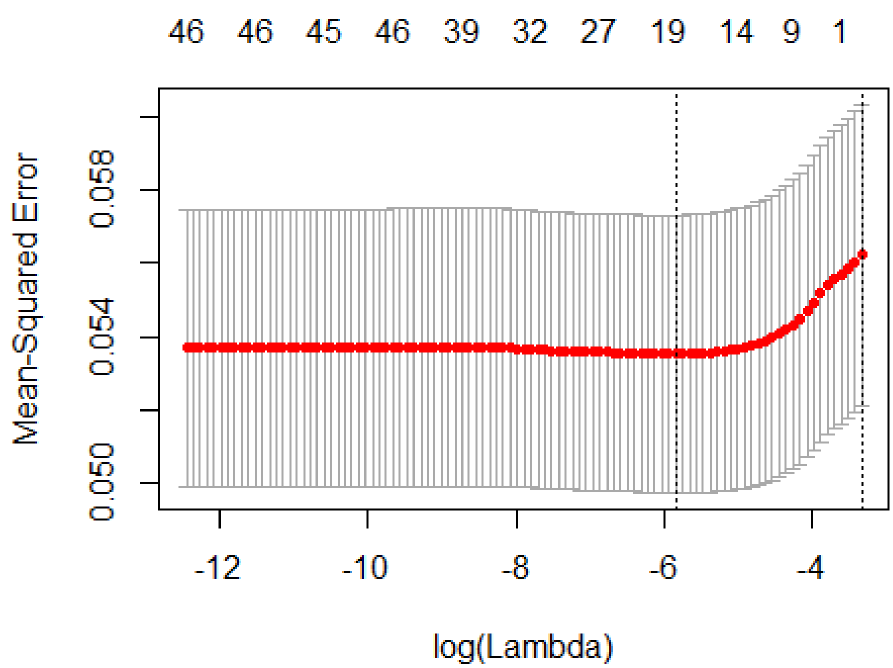
```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loading required package: foreach
```

```
## Loaded glmnet 2.0-16
```

```
temp_df <-ticdata2000_df_subset1
temp_df$V86 <- as.integer(temp_df$V86)
xmat <- model.matrix(V86 ~ ., data = temp_df)[, -1]
cv.lasso <- cv.glmnet(xmat, temp_df$V86, alpha = 1)
plot(cv.lasso)
```



Now, we choose the best value of

$$log(Lambda)$$

by minimizing the Mean-Squared_Error. Clearly, we don’t need to consider the value of

$$log(Lambda)$$

one standard error away from

$$min(log(Lambda))$$

because that will lead to the number of predictors close to zero. The R code for the same is mentioned below.

```
bestlam <- cv.lasso$lambda.min
print("Best Log(Lambda): ")

## [1] "Best Log(Lambda): "

bestlam

## [1] 0.00290187

fit.lasso <- glmnet(xmat, temp_df$V86, alpha = 1)
predict(fit.lasso, s = bestlam, type = "coefficients")[1:47, ]

##   (Intercept)            V1            V5            V7           V10
##  9.481818e-01  0.000000e+00  0.000000e+00  1.864899e-03  2.768592e-03
##           V12           V16           V18           V19           V22
##  0.000000e+00  4.313793e-03 -2.362184e-03  0.000000e+00  2.081881e-03
##           V23           V24           V25           V28           V29
##  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
##           V30           V31           V32           V34           V35
## -6.831939e-04  0.000000e+00  2.701924e-03  0.000000e+00  0.000000e+00
##           V36           V37           V39           V40           V42
##  0.000000e+00  0.000000e+00  0.000000e+00  1.257305e-03  2.514800e-03
##           V43           V44           V47           V49           V57
##  2.865682e-03  1.049848e-02  8.529382e-03  0.000000e+00  1.368781e-02
##           V59           V61           V64           V65           V68
##  4.749891e-03  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
##           V75           V76           V80           V82           V85
##  0.000000e+00  0.000000e+00  0.000000e+00  2.510284e-01  5.684784e-02
##        V44V65        V47V68        V49V70        V57V78        V59V80
##  0.000000e+00  1.088648e-03  0.000000e+00  1.818843e-06  0.000000e+00
##        V61V82        V64V85
##  0.000000e+00  0.000000e+00
```

So we have the features now and we can also validate that if we take

$$log(Lambda)$$

value one standard error away, we end up with having no features. The code for the same is below.

```
bestlam <- cv.lasso$lambda.1se
print("Best Log(Lambda): ")

## [1] "Best Log(Lambda): "
```

```r
bestlam
```

```
## [1] 0.03577561
```

```r
fit.lasso <- glmnet(xmat, temp_df$V86, alpha = 1)
predict(fit.lasso, s = bestlam, type = "coefficients")[1:47, ]
```

```
## (Intercept)          V1          V5          V7         V10         V12 
##    1.059773    0.000000    0.000000    0.000000    0.000000    0.000000 
##         V16         V18         V19         V22         V23         V24 
##    0.000000    0.000000    0.000000    0.000000    0.000000    0.000000 
##         V25         V28         V29         V30         V31         V32 
##    0.000000    0.000000    0.000000    0.000000    0.000000    0.000000 
##         V34         V35         V36         V37         V39         V40 
##    0.000000    0.000000    0.000000    0.000000    0.000000    0.000000 
##         V42         V43         V44         V47         V49         V57 
##    0.000000    0.000000    0.000000    0.000000    0.000000    0.000000 
##         V59         V61         V64         V65         V68         V75 
##    0.000000    0.000000    0.000000    0.000000    0.000000    0.000000 
##         V76         V80         V82         V85      V44V65      V47V68 
##    0.000000    0.000000    0.000000    0.000000    0.000000    0.000000 
##      V49V70      V57V78      V59V80      V61V82      V64V85 
##    0.000000    0.000000    0.000000    0.000000    0.000000
```

```r
choose_colm_LASSO <- c("V5", "V7", "V10", "V16", "V18", "V22",
"V28", "V30", "V32", "V40", "V42", "V43", "V44",
"V57", "V59", "V76", "V82", "V85", "V47V68")
print("Final Features selected: ")
```

```
## [1] "Final Features selected: "
```

```r
print(choose_colm_LASSO)
```

```
##  [1] "V5"     "V7"     "V10"    "V16"    "V18"    "V22"    "V28"   
##  [8] "V30"    "V32"    "V40"    "V42"    "V43"    "V44"    "V57"   
## [15] "V59"    "V76"    "V82"    "V85"    "V47V68"
```

```r
print("Final Length: ")
```

```
## [1] "Final Length: "
```

```r
length(choose_colm_LASSO)
```

```
## [1] 19
```

So I can see that, We are coming up with 19 variables only.

Model Building and Checking Prediction Accuracy:

Now the next step will be to build models on the selected features and compare the models on their accuracy. As part of this project we will follow two Classification techniques - 1. Naive Bayes 2. Logistics Regression

Because the target here is a categorical variable, so this is a classification problem.

Naive Bayes:

Naive Bayes is a classification technique that takes into account Bayes Theorem of probability with an assumption of independence among the predictors. It assumes that a certain feature in a model within a class is not related to any other features present in that class. Though sounds a bit extreme, but this feature of Naive Bayes provides extreme flexibility. My choice for Naive Bayes algorithm in this case goes for the below reasons.

1. The model is very easy to build and at the same time, it is very useful for small to very very large data sets. So it is quite flexible.
2. The nature of conditional independence, though too extreme, gives better output than other complex models very often.
3. There is no constraint on training data. Because here, we have very less number of positive (People who bought insurance plan) cases in target variable as compared to the negative cases (People who did not buy insurance plan), this situation does not affect the Naive Bayes Model.

We will only use the feature selected by Recursive Feature Elimination because all other methods are meant for linear models (which we shall use later on in Logistic Regression). The code for Naive Bayes is written below. We will use 10 fold cross validation repeated 10 times below because it is good for trading of variance.

```
library(naivebayes)

##
## Attaching package: 'naivebayes'

## The following object is masked from 'package:data.table':
##
##     tables

ctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 10)

nb_train_df <- ticdata2000_df_subset1[,c(choose_colm_RFE, 'V86')]
model <- train(V86 ~ ., data = nb_train_df, method = "naive_bayes", trControl
= ctrl)

nb_test_df <- ticeval2000_df[,c(choose_colm_RFE)]
nb_pred <- predict(model, nb_test_df, type="prob")

top_800_customers_nb <- head(nb_pred[order(-nb_pred[, c(2)]), ], n=800)
nb_test_df$V86 <- as.numeric(0)
nb_test_df[as.numeric(rownames(top_800_customers_nb)), c("V86")] <- 1
predicted_values <- nb_test_df[, c("V86")]
table(tictgts2000_df$V86, predicted_values)

##    predicted_values
##        0    1
```

```
##   0 3072  690
##   1  128  110

mean(tictgts2000_df$V86 == predicted_values)

## [1] 0.7955
```

The above code predicts the 1st 800 customers who are likely to buy the car insurance policy and checks how many of these customers actually bought it. It can be seen that out of these 800 predictions 110 got accurate from 238 actual predictions, means the prediction accuracy for customers who actually bought the insurance is 46% approximately. The overall prediction accuracy is however very high i.e 79.5%.

We also tried it through bootstrapping, which also gives the same result.

```
library(naivebayes)
ctrl <- trainControl(method = "boot")

nb_train_df <- ticdata2000_df_subset1[,c(choose_colm_RFE, 'V86')]
model <- train(V86 ~ ., data = nb_train_df, method = "naive_bayes", trControl
= ctrl)

nb_test_df <- lm_test_df <- ticeval2000_df[,c(choose_colm_RFE)]
nb_pred <- predict(model, nb_test_df, type="prob")

top_800_customers_nb <- head(nb_pred[order(-nb_pred[, c(2)]), ], n=800)
nb_test_df$V86 <- as.numeric(0)
nb_test_df[as.numeric(rownames(top_800_customers_nb)), c("V86")] <- 1
predicted_values <- nb_test_df[, c("V86")]
table(tictgts2000_df$V86, predicted_values)

##    predicted_values
##        0    1
##   0 3072  690
##   1  128  110

mean(tictgts2000_df$V86 == predicted_values)

## [1] 0.7955
```

Logistic Regression:

Logistic Regression is a linear classification algorithm. The log-off for the Logistic regression is a linear model. The reason for doing logistic regression is mentioned below - 1. It is very simple method and the model is very easy to build. 2. It works really well when there are only two classes within the predictor, which is the case here.

Here, we could have also used Linear Discriminant Analysis (LDA) and Quadratic Discriminant Analysis (QDA) techniques as well. But, considering that Logistic Regression works pretty well for target variable having exactly 2 classes, we will not get any advantage of doing LDA and QDA here. So we are not using them in this case.

We would be using all feature selection techniques here.

Model Using Features Selected From Recursive Feature Elimination:

Because RFE is algorithm specific, we need to select the feature using the option **functions =lmFuncs**.

```
library(caret)
subsets <- c(10, 20)
drops <- c("V86")
ticdata2000_df_subset1$V86 <- as.integer(ticdata2000_df_subset1$V86)
ctrl <- rfeControl(functions =lmFuncs,
                   method = "cv",
                   number = 10,
                   verbose = FALSE)

lmProfile <- rfe(ticdata2000_df_subset1[, !(names(ticdata2000_df_subset1) %in
% drops)], ticdata2000_df_subset1$V86,
                 sizes = subsets,
                 rfeControl = ctrl)
choose_colm_RFE <- lmProfile$optVariables[1:10]
choose_colm_RFE

##  [1] "V82"    "V85"    "V64V85" "V64"    "V68"    "V61V82" "V44"
##  [8] "V36"    "V35"    "V61"
```

After the features have been selected we can put them into the model with 10 fold cross validation with 10 repetitions same as done before.

```
ctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 10)
lm_train_df <- ticdata2000_df_subset1[,c(choose_colm_RFE, 'V86')]
lm_model <- train(V86 ~ ., data = lm_train_df, method = "glm", trControl = ct
rl)
lm_test_df <- ticeval2000_df[,c(choose_colm_RFE)]

lm_test_df <- cbind(lm_test_df, tictgts2000_df)

lm_pred <- predict(lm_model, lm_test_df)
lm_pred <- cbind(nb_pred, as.data.frame(lm_pred))
top_800_customers_lm <- head(lm_pred[order(-lm_pred[, c(3)]), ], n=800)
lm_test_df$V86 <- as.numeric(0)
lm_test_df[as.numeric(rownames(top_800_customers_lm)), c("V86")] <- 1
predicted_values <- lm_test_df[, c("V86")]
table(tictgts2000_df$V86, predicted_values)

##    predicted_values
##        0    1
##   0 3063  699
##   1  137  101

mean(tictgts2000_df$V86 == predicted_values)
```

```
## [1] 0.791
```

The above code predicts the 1st 800 customers who are likely to buy the car insurance policy and checks how many of these customers actually bought it. It can be seen that out of these 800 predictions 101 got accurate from 238 actual predictions, means the prediction accuracy for customers who actually bought the insurance is 42% approximately. The overall prediction accuracy is however very high i.e 79.1%. So it can be seen that Naive Bayes predicted with better accuracy than did Logistic Regression.

We also tried it through bootstrapping, which also gives the same result.

```
ctrl <- trainControl(method = "boot")
lm_train_df <- ticdata2000_df_subset1[,c(choose_colm_RFE, 'V86')]
lm_model <- train(V86 ~ ., data = lm_train_df, method = "glm", trControl = ct
rl)
lm_test_df <- ticeval2000_df[,c(choose_colm_RFE)]

lm_test_df <- cbind(lm_test_df, tictgts2000_df)

lm_pred <- predict(lm_model, lm_test_df)
lm_pred <- cbind(nb_pred, as.data.frame(lm_pred))
top_800_customers_lm <- head(lm_pred[order(-lm_pred[, c(3)]), ], n=800)
lm_test_df$V86 <- as.numeric(0)
lm_test_df[as.numeric(rownames(top_800_customers_lm)), c("V86")] <- 1
predicted_values <- lm_test_df[, c("V86")]
table(tictgts2000_df$V86, predicted_values)

##    predicted_values
##        0    1
##   0 3063  699
##   1  137  101

mean(tictgts2000_df$V86 == predicted_values)

## [1] 0.791
```

Model Using Features Selected From Forward Subset Selection:

Let's now do the same experiment on the subset got from Forward Subset Selection with 10 fold cross validation with repeats 10 times.

```
ctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 10)
lm_train_df <- ticdata2000_df_subset1[,c(choose_colm_FSS, 'V86')]
lm_model <- train(V86 ~ ., data = lm_train_df, method = "glm", trControl = ct
rl)
lm_test_df <- ticeval2000_df[,c(choose_colm_FSS)]

lm_test_df <- cbind(lm_test_df, tictgts2000_df)

lm_pred <- predict(lm_model, lm_test_df)
lm_pred <- cbind(nb_pred, as.data.frame(lm_pred))
```

```
top_800_customers_lm <- head(lm_pred[order(-lm_pred[, c(3)]), ], n=800)
lm_test_df$V86 <- as.numeric(0)
lm_test_df[as.numeric(rownames(top_800_customers_lm)), c("V86")] <- 1
predicted_values <- lm_test_df[, c("V86")]
table(tictgts2000_df$V86, predicted_values)
```

```
##    predicted_values
##        0    1
##   0 3075  687
##   1  125  113
```

```
mean(tictgts2000_df$V86 == predicted_values)
```

```
## [1] 0.797
```

The above code predicts the 1st 800 customers who are likely to buy the car insurance policy and checks how many of these customers actually bought it. It can be seen that out of these 800 predictions 113 got accurate from 238 actual predictions, means the prediction accuracy for customers who actually bought the insurance is 47.4% approximately. The overall prediction accuracy is however very high i.e 79.7%. So it can be seen that Logistic Regression with Forward subset selection as feature selection method predicted with better accuracy than did Naive Bayes.

We also tried it through bootstrapping, which also gives the same result.

```
ctrl <- trainControl(method = "boot")
lm_train_df <- ticdata2000_df_subset1[,c(choose_colm_FSS, 'V86')]
lm_model <- train(V86 ~ ., data = lm_train_df, method = "glm", trControl = ct
rl)
lm_test_df <- ticeval2000_df[,c(choose_colm_FSS)]

lm_test_df <- cbind(lm_test_df, tictgts2000_df)

lm_pred <- predict(lm_model, lm_test_df)
lm_pred <- cbind(nb_pred, as.data.frame(lm_pred))
top_800_customers_lm <- head(lm_pred[order(-lm_pred[, c(3)]), ], n=800)
lm_test_df$V86 <- as.numeric(0)
lm_test_df[as.numeric(rownames(top_800_customers_lm)), c("V86")] <- 1
predicted_values <- lm_test_df[, c("V86")]
table(tictgts2000_df$V86, predicted_values)
```

```
##    predicted_values
##        0    1
##   0 3075  687
##   1  125  113
```

```
mean(tictgts2000_df$V86 == predicted_values)
```

```
## [1] 0.797
```

Model Using Features Selected From Backward Subset Selection:

Let's now do the same experiment on the subset got from Backward Subset Selection with 10 fold cross validation with repeats 10 times.

```
ctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 10)
lm_train_df <- ticdata2000_df_subset1[,c(choose_colm_BSS, 'V86')]
lm_model <- train(V86 ~ ., data = lm_train_df, method = "glm", trControl = ct
rl)
lm_test_df <- ticeval2000_df[,c(choose_colm_BSS)]

lm_test_df <- cbind(lm_test_df, tictgts2000_df)

lm_pred <- predict(lm_model, lm_test_df)
lm_pred <- cbind(nb_pred, as.data.frame(lm_pred))
top_800_customers_lm <- head(lm_pred[order(-lm_pred[, c(3)]), ], n=800)
lm_test_df$V86 <- as.numeric(0)
lm_test_df[as.numeric(rownames(top_800_customers_lm)), c("V86")] <- 1
predicted_values <- lm_test_df[, c("V86")]
table(tictgts2000_df$V86, predicted_values)

##    predicted_values
##        0    1
##   0 3080  682
##   1  120  118

mean(tictgts2000_df$V86 == predicted_values)

## [1] 0.7995
```

The above code predicts the 1st 800 customers who are likely to buy the car insurance policy and checks how many of these customers actually bought it. It can be seen that out of these 800 predictions 113 got accurate from 238 actual predictions, means the prediction accuracy for customers who actually bought the insurance is 50% approximately. The overall prediction accuracy is however very high i.e 79.9%. So it can be seen that Logistic Regression with Backward subset selection as feature selection method predicted with better accuracy than did Logistic Regression with Forward subset selection as feature selection.

We also tried it through bootstrapping, which also gives the same result.

```
ctrl <- trainControl(method = "boot")
lm_train_df <- ticdata2000_df_subset1[,c(choose_colm_BSS, 'V86')]
lm_model <- train(V86 ~ ., data = lm_train_df, method = "glm", trControl = ct
rl)
lm_test_df <- ticeval2000_df[,c(choose_colm_BSS)]

lm_test_df <- cbind(lm_test_df, tictgts2000_df)

lm_pred <- predict(lm_model, lm_test_df)
```

```
lm_pred <- cbind(nb_pred, as.data.frame(lm_pred))
top_800_customers_lm <- head(lm_pred[order(-lm_pred[, c(3)]), ], n=800)
lm_test_df$V86 <- as.numeric(0)
lm_test_df[as.numeric(rownames(top_800_customers_lm)), c("V86")] <- 1
predicted_values <- lm_test_df[, c("V86")]
table(tictgts2000_df$V86, predicted_values)
```

```
##    predicted_values
##        0    1
##   0 3080  682
##   1  120  118
```

```
mean(tictgts2000_df$V86 == predicted_values)
```

```
## [1] 0.7995
```

Model Using Features Selected From LASSO:

Let's now do the same experiment on the subset got from LASSO with 10 fold cross validation with repeats 10 times.

```
ctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 10)
lm_train_df <- ticdata2000_df_subset1[,c(choose_colm_LASSO, 'V86')]
lm_model <- train(V86 ~ ., data = lm_train_df, method = "glm", trControl = ct
rl)
lm_test_df <- ticeval2000_df[,c(choose_colm_LASSO)]

lm_test_df <- cbind(lm_test_df, tictgts2000_df)

lm_pred <- predict(lm_model, lm_test_df)
lm_pred <- cbind(nb_pred, as.data.frame(lm_pred))
top_800_customers_lm <- head(lm_pred[order(-lm_pred[, c(3)]), ], n=800)
lm_test_df$V86 <- as.numeric(0)
lm_test_df[as.numeric(rownames(top_800_customers_lm)), c("V86")] <- 1
predicted_values <- lm_test_df[, c("V86")]
table(tictgts2000_df$V86, predicted_values)
```

```
##    predicted_values
##        0    1
##   0 3079  683
##   1  121  117
```

```
mean(tictgts2000_df$V86 == predicted_values)
```

```
## [1] 0.799
```

The above code predicts the 1st 800 customers who are likely to buy the car insurance policy and checks how many of these customers actually bought it. It can be seen that out of these 800 predictions 113 got accurate from 238 actual predictions, means the prediction accuracy for customers who actually bought the insurance is 49.5% approximately. The overall prediction accuracy is however very high i.e 79.9%. So, in this case Logistic

Regression with LASSO is working almost as accurately as, if not completely, the case for Logistic Regression with Backward Subset Selection as feature selection technique. But, in this case we are getting much simpler model with just 19 features as compared to the model with 27 features observed by Backward Subset Selection. So we keep this model observed by LASSO as our main model to consider.

We also tried it through bootstrapping, which also gives the same result.

```
ctrl <- trainControl(method = "boot")
lm_train_df <- ticdata2000_df_subset1[,c(choose_colm_LASSO, 'V86')]
lm_model <- train(V86 ~ ., data = lm_train_df, method = "glm", trControl = ct
rl)
lm_test_df <- ticeval2000_df[,c(choose_colm_LASSO)]

lm_test_df <- cbind(lm_test_df, tictgts2000_df)

lm_pred <- predict(lm_model, lm_test_df)
lm_pred <- cbind(nb_pred, as.data.frame(lm_pred))
top_800_customers_lm <- head(lm_pred[order(-lm_pred[, c(3)]), ], n=800)
lm_test_df$V86 <- as.numeric(0)
lm_test_df[as.numeric(rownames(top_800_customers_lm)), c("V86")] <- 1
predicted_values <- lm_test_df[, c("V86")]
table(tictgts2000_df$V86, predicted_values)

##    predicted_values
##        0    1
##   0 3079  683
##   1  121  117

mean(tictgts2000_df$V86 == predicted_values)

## [1] 0.799
```

Now, let's see if can further optimize the number of features by not hampering the model significantly.

Doing Further Model Investigation:

Now let's try to investigate the model named **lm_model** closely. In order to do that we need to do a summary on that.

```
summary(lm_model)

## 
## Call:
## NULL
## 
## Deviance Residuals: 
##      Min        1Q    Median        3Q       Max  
## -0.72676  -0.08206  -0.04662  -0.01488   1.03745  
## 
```

```
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.9264279  0.0279471  33.149  < 2e-16 ***
## V5          -0.0007786  0.0014223  -0.547 0.584131
## V7           0.0033747  0.0018556   1.819 0.069020 .
## V10          0.0035548  0.0019674   1.807 0.070837 .
## V16          0.0063053  0.0026659   2.365 0.018054 *
## V18         -0.0037925  0.0021369  -1.775 0.075984 .
## V22          0.0034815  0.0018226   1.910 0.056155 .
## V28          0.0043293  0.0021035   2.058 0.039621 *
## V30         -0.0011588  0.0012441  -0.931 0.351682
## V32          0.0031704  0.0022443   1.413 0.157814
## V40          0.0032539  0.0033505   0.971 0.331493
## V42          0.0019235  0.0034420   0.559 0.576293
## V43          0.0024050  0.0021045   1.143 0.253171
## V44          0.0130863  0.0036757   3.560 0.000374 ***
## V57          0.0253602  0.0159660   1.588 0.112255
## V59          0.0050560  0.0018978   2.664 0.007741 **
## V76          0.0041025  0.0081877   0.501 0.616347
## V82          0.2786462  0.0371434   7.502 7.24e-14 ***
## V85          0.0698300  0.0256172   2.726 0.006432 **
## V47V68       0.0076049  0.0008416   9.037  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.05321718)
##
##     Null deviance: 327.20  on 5821  degrees of freedom
## Residual deviance: 308.77  on 5802  degrees of freedom
## AIC: -534.02
##
## Number of Fisher Scoring iterations: 2
```

Let's 1st target the variables very high P-values and remove them. So accordingly, we try removing the variables V5, V42 and V76 and checking the model's prediction accuracy.

```r
choose_colm_LASSO_subset1 <- c("V7", "V10", "V16", "V18", "V22",
"V28", "V30", "V32", "V40", "V43", "V44",
"V57", "V59", "V82", "V85", "V47V68")
ctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 10)
lm_train_df <- ticdata2000_df_subset1[,c(choose_colm_LASSO_subset1, 'V86')]
lm_model_sb1 <- train(V86 ~ ., data = lm_train_df, method = "glm", trControl
= ctrl)
lm_test_df <- ticeval2000_df[,c(choose_colm_LASSO_subset1)]

lm_test_df <- cbind(lm_test_df, tictgts2000_df)

lm_pred <- predict(lm_model_sb1, lm_test_df)
lm_pred <- cbind(nb_pred, as.data.frame(lm_pred))
top_800_customers_lm <- head(lm_pred[order(-lm_pred[, c(3)]), ], n=800)
```

```
lm_test_df$V86 <- as.numeric(0)
lm_test_df[as.numeric(rownames(top_800_customers_lm)), c("V86")] <- 1
predicted_values <- lm_test_df[, c("V86")]
table(tictgts2000_df$V86, predicted_values)

##    predicted_values
##        0    1
##   0 3077  685
##   1  123  115

mean(tictgts2000_df$V86 == predicted_values)

## [1] 0.798
```

The above code predicts the 1st 800 customers who are likely to buy the car insurance policy and checks how many of these customers actually bought it. It can be seen that out of these 800 predictions 115 got accurate from 238 actual predictions, means the prediction accuracy for customers who actually bought the insurance is 48.31% approximately. The overall prediction accuracy is however very high i.e 79.8%. So there is a slight decrease in accuracy against a simpler model, which is pretty acceptable. So we keep this model.

We have tried to make the model simpler further with several other combinations of existing features and for each and every case, it is badly hampering the prediction accuracy. so we keep this as our final model.

Conclusion

So it can be concluded that **"Logistic Regression"** is the best classification algorithm for the given data considering the prediction accuracy. The final features which are significantly affecting target variable -

1. MGODPR Protestant - Column 7
2. MRELGE Married - column 10
3. MOPLHOOG High level education - Column 16
4. MOPLLAAG Lower level education - column 18
5. MBERMIDD Middle management - column 22
6. MSKC Social class C - column 28
7. MHHUUR Rented house - column 30
8. MAUT1 1 car - column 32
9. MINK7512 Income 75-122.000 - column 40
10. MKOOPKLA Purchasing power class - column 43
11. PWAPART Contribution private third party insurance - column 44
12. PGEZONG Contribution family accidents insurance policies - column 57
13. PBRAND Contribution fire policies - column 59
14. APLEZIER Number of boat policies - column 82
15. ABYSTAND Number of social security insurance policies - column 85

16. PPERSAUT Contribution car policies (column 47) multiplied APERSAUT Number of car policies (column 68)

So, it can be seen that both socio-demographic and product ownership variables and their interactions are important for predicting the customer CARAVAN insurance policy buying behavior.

The below list of customer ids should be sent mail for Caravan policy -

```
predicted_customers <- as.numeric(rownames(top_800_customers_lm))
actual_customers <- which(tictgts2000_df[ , "V86"] == 1)
intersect(predicted_customers,actual_customers)
```

```
##   [1]  576 2622 3500 3658  948   92 1499   89 2516 3727  687   12  829  57
2
##  [15] 3081    2 2047 2136 1822 2357 1468 3093 2982   30 2718 2633  210 241
7
##  [29] 2165 3077 1334 2119  596 1562 2930 3048 1509 2511 1418 1703 3001   31
4
##  [43] 1868  239  337 3069 2635 2872 2310 2541 3476 2200 2771  787 2501 169
1
##  [57]  297 1436 2738  279 1170  918 3229 3664 2162  578 3616 1368 1118 171
1
##  [71] 2830  719  660 3892 3487 1852  856 2443  232 2850 3760 1861 1706 196
7
##  [85] 2864 2673 2218 2201 3088 3604 1305  828 3325 2147 3676 1824 1086 302
1
##  [99] 1907 2019 2615 1883 1121 2032 3260 3468 2354  797 2935  383 1452 276
8
## [113] 2149  737 2750
```

This concludes the project. Thank you for taking your time to read this document.