Capstone Project

Table of Contents

Table of Contents

Introduction

Premium paid by the customer is the major revenue source for insurance companies. Default in premium payments results in significant revenue losses and hence insurance companies would like to know upfront which type of customers would default premium payments. The objective of this project us to

1. Build a model that can predict the likelihood of a customer defaulting on premium payments (Who is likely to default)
2. Identify the factors that drive higher default rate (Are there any characteristics of the customers who are likely to default?)
3. Propose a strategy for reducing default rates by using the model and other insights from the analysis (What should be done to reduce the default rates?)

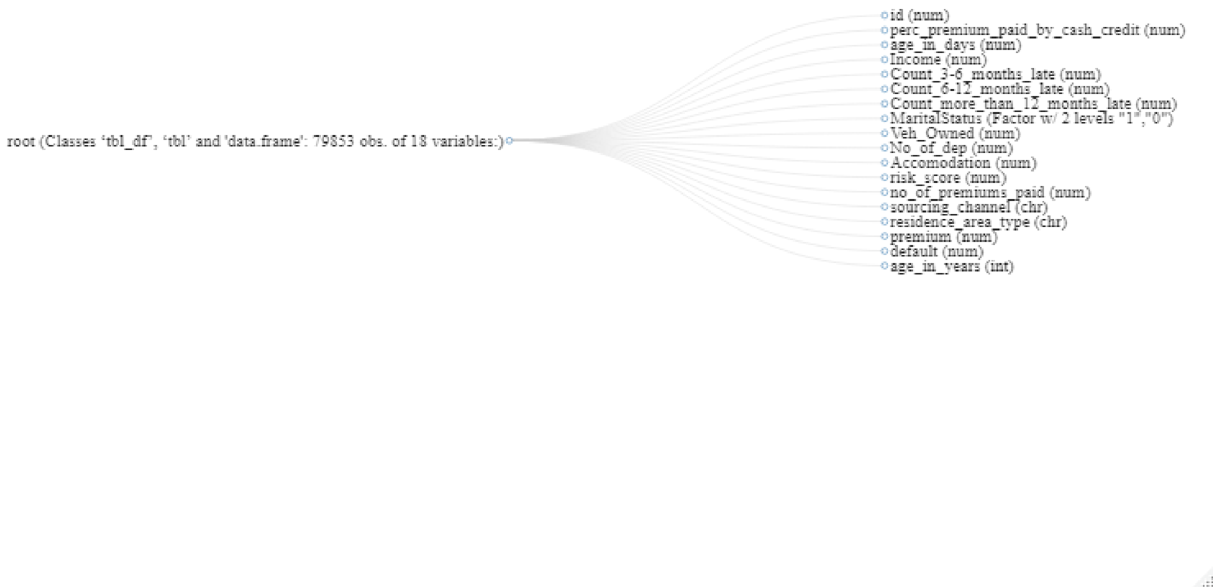
Data Report

Variable Identification

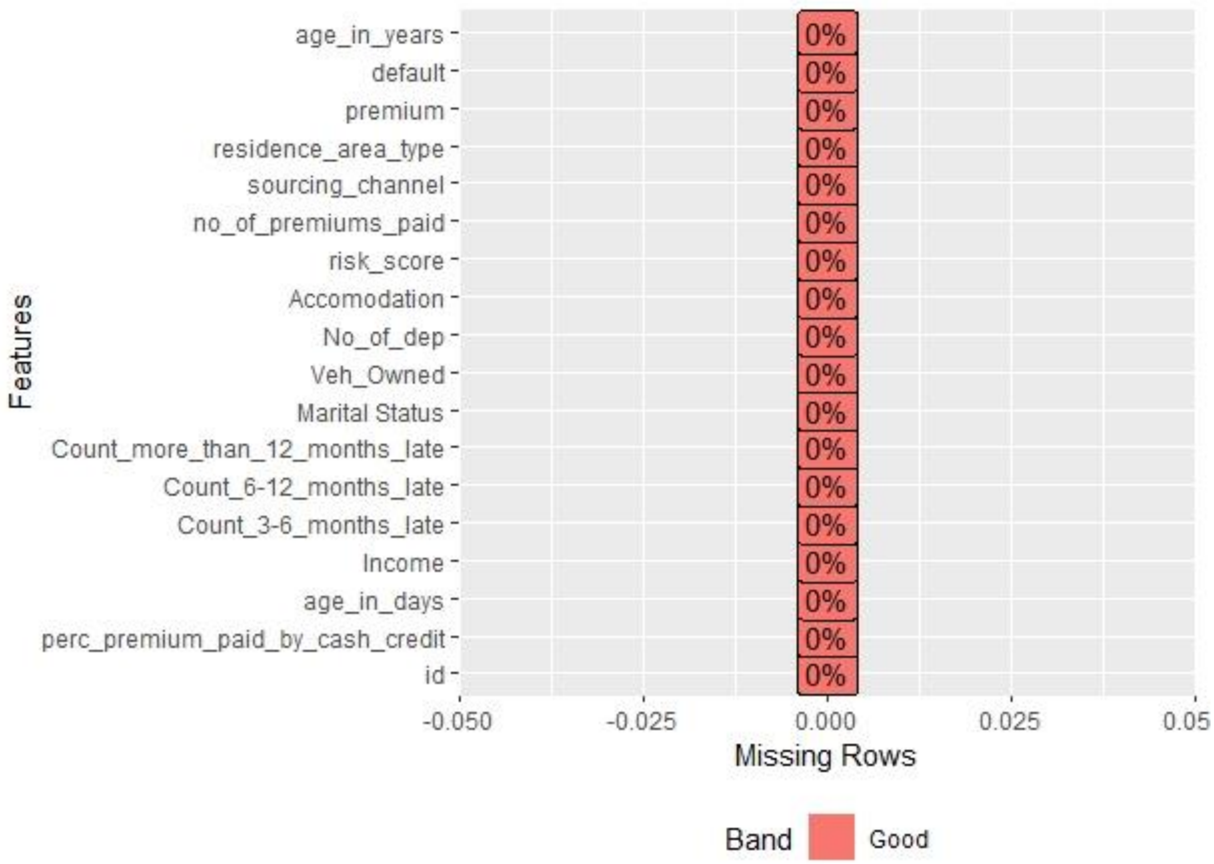
We cane use R functions to do as follows

- dim: we see that we have 79853 different observation in 17 variables.
- names: we see that all the names looking good and straightforward to work with accept some change we will come on soon.
- str: we identifying that:
  - perc_premium_paid_by_cash_credit: num
  - age_in_days : num
  - Income : num
  - Count_3-6_months_late : num
  - Count_6-12_months_late : num
  - Count_more_than_12_months_late : num
  - Marital Status : num
  - Veh_Owned : num
  - No_of_dep : num
  - Accomodation : num
  - risk_score : num
  - no_of_premiums_paid : num
  - sourcing_channel : chr
  - residence_area_type : chr
  - premium : num
  - default : num
- head & tail: shows that we are lucky we have quite bet a clear data.
- anyNA: we see that we don't have missing value at whole dataset.

Please refer Appendix A for Source Code.



Dataset structure



Missing Value

Initial Exploratory Data Analysis

Univariate Analysis

Five Numbers

Variable	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
age_in_years	21.0	42.0	52.0	56.4	63.0	104.0
Premium	1200	5400	7500	17580	13800	60000
no_of_premiums_paid	2.0	7.0	10.0	18.6	14.0	60.0
risk_score	91.90	98.83	99.18	97.86	99.52	99.89
No_of_dep	1	2	3	2.6	3	4
Veh_Owned	1	1	2	2	3	3
Income	24030	108010	166560	18162658	252090	90262600

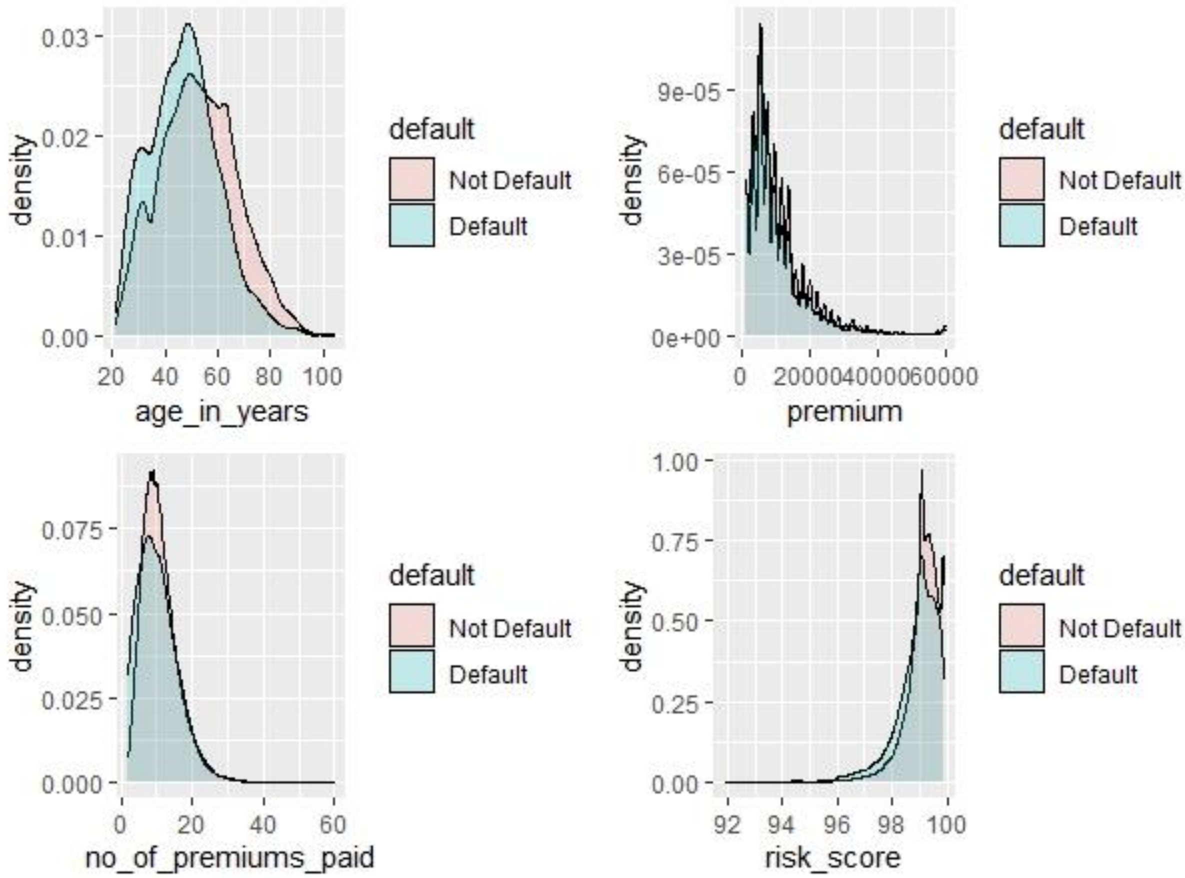
In Dataset we fix the variable age_in_days to be age_in_years to easiest the dealing with years.

Charts

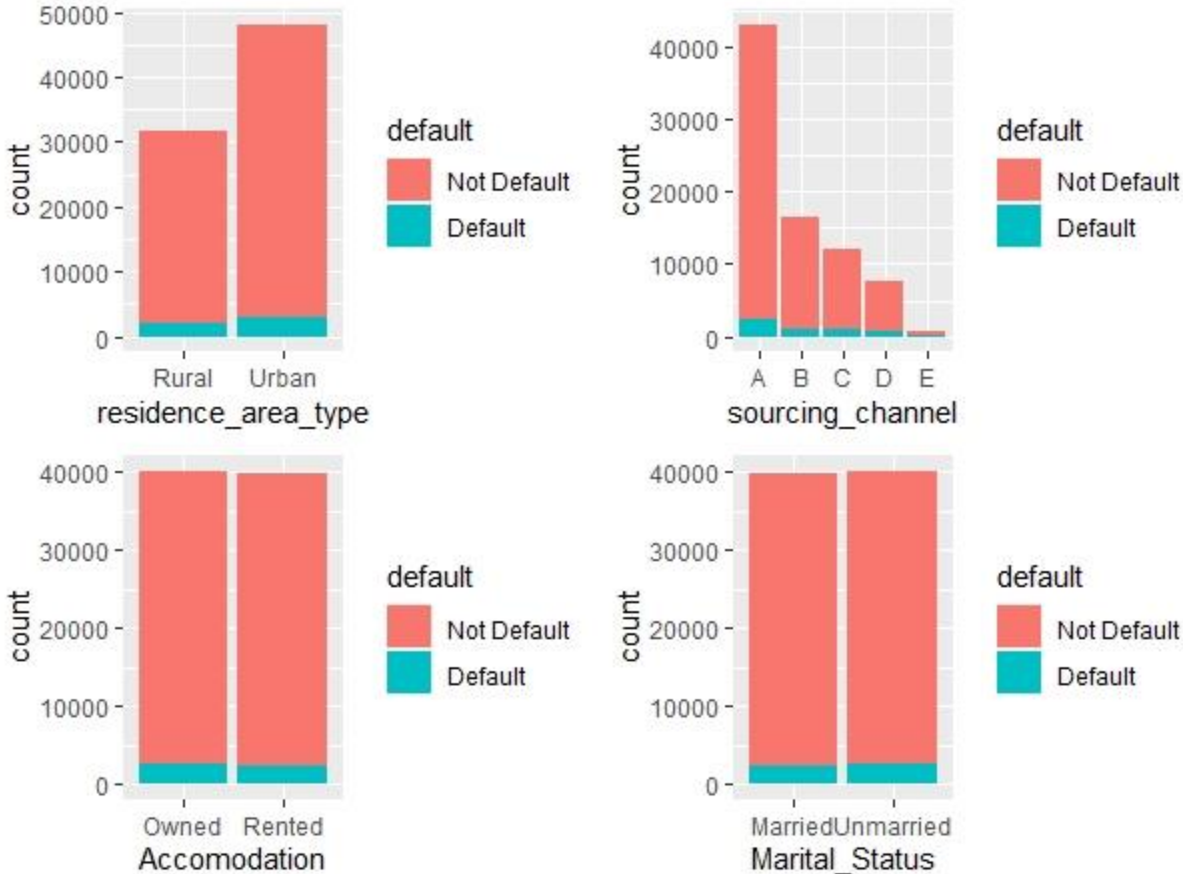

count
60000
40000
20000
0
Default
Not Default
default
count
50000
40000
30000
20000
10000
0
Rural
Urban
residence_area_type
count
40000
30000
20000
10000
0
A
B
C
D
E
sourcing_channel
count
40000
30000
20000
10000
0
Owned
Rented
Accomodation
count
40000
30000
20000
10000
0
Married
Unmarried
Marital_Status
count
60000
40000
20000
0
0
5
10
Count_3-6_months_late
count
60000
40000
20000
0
0
5
10
15
Count_6-12_months_late
count
60000
40000
20000
0
0
3
6
9
12
Count_more_than_12_months_late

Bivariate Analysis

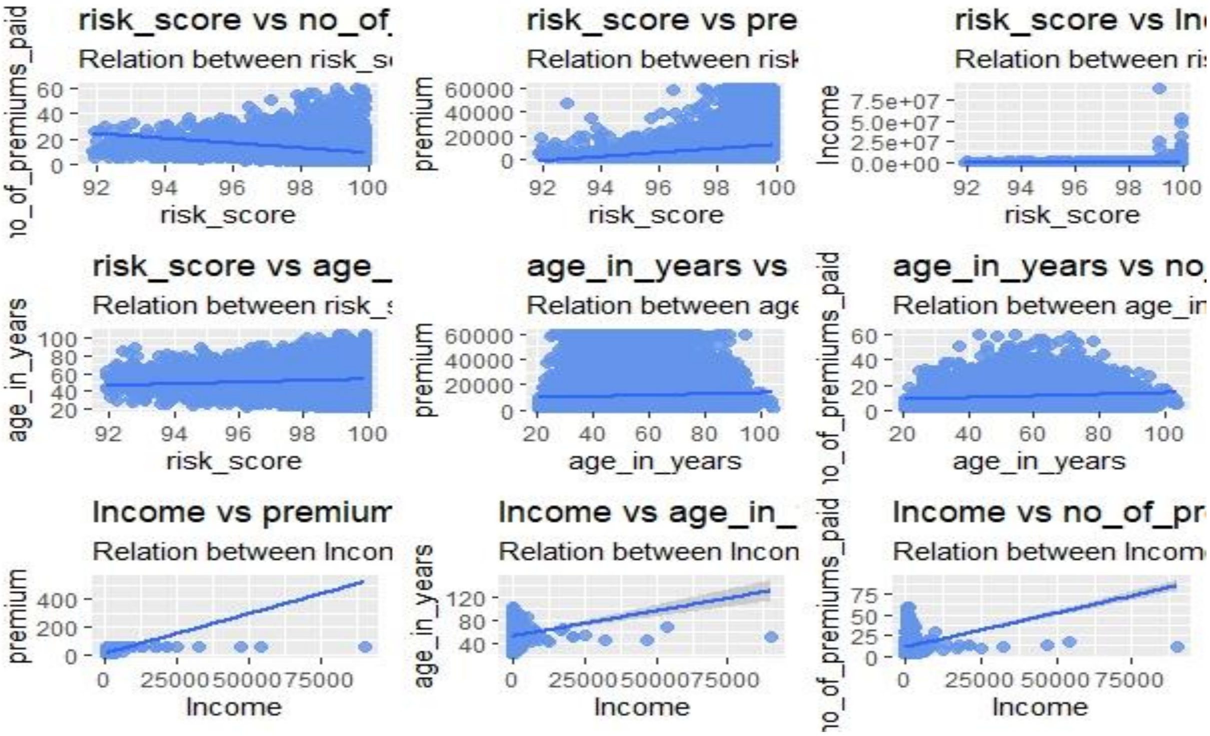
Default vs Numerical Variables

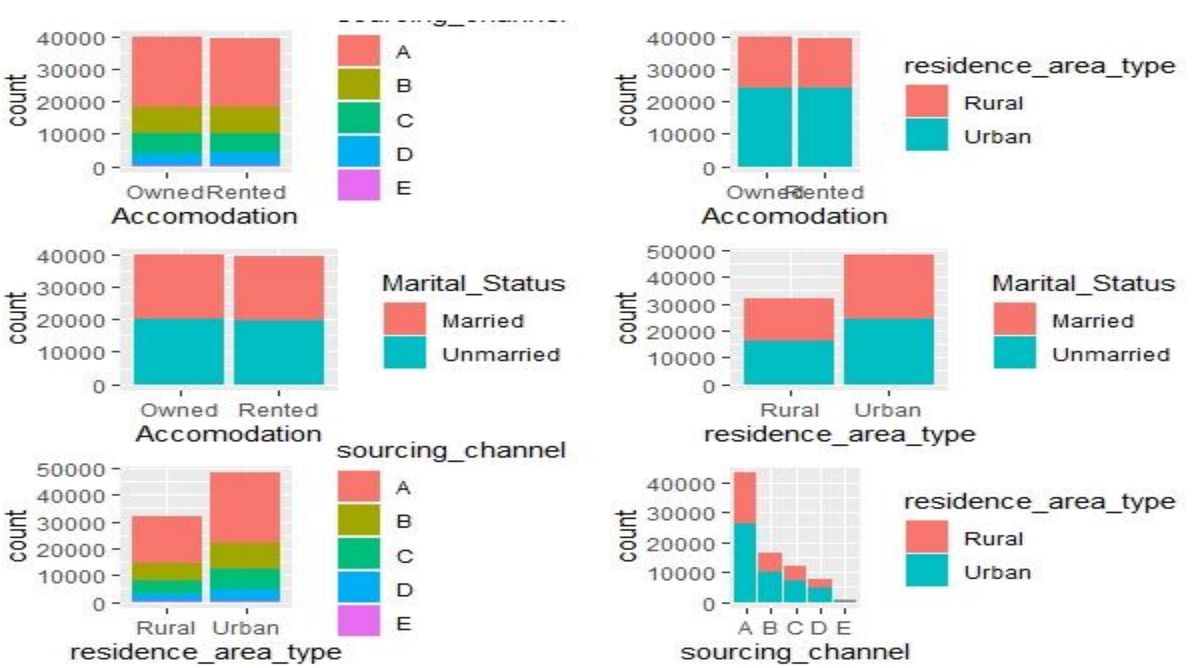


Default vs Categorical Variables



Correlation Analysis





Data pre-processing

Removal of Unwanted Variables

Variable to be removed are:

- Id: we don't need it.
- Age_in_days: while we are generateing new variable from this variable but in years format, so we don't need it any more.

Missing Value Treatment

There are no messing data in the dataset.

Outlier Treatment

The Method of outliers detection we are going to use is based on the **percentiles**. With the percentiles method, all observations that lie outside the interval formed by the 1 and 99. percentiles will be considered as potential outlier, after that simply we are going to remove the observation with outlier.

Please refer Appendix A for Source Code.

Variable Transformation

We realize that we need to maintain (change type, rename) some variable as follows:

- #Marital_Status
- #Count_3_6_months_late
- # Count_6_12_months_late
- #Veh_Owned
- #No_of_dep
- #Accomodation
- #sourcing_channel
- #residence_area_type
- #default

Addition of new variables

We add the following:

- Age_in_years

Please refer Appendix A for Source Code.

Exploratory Data Analysis

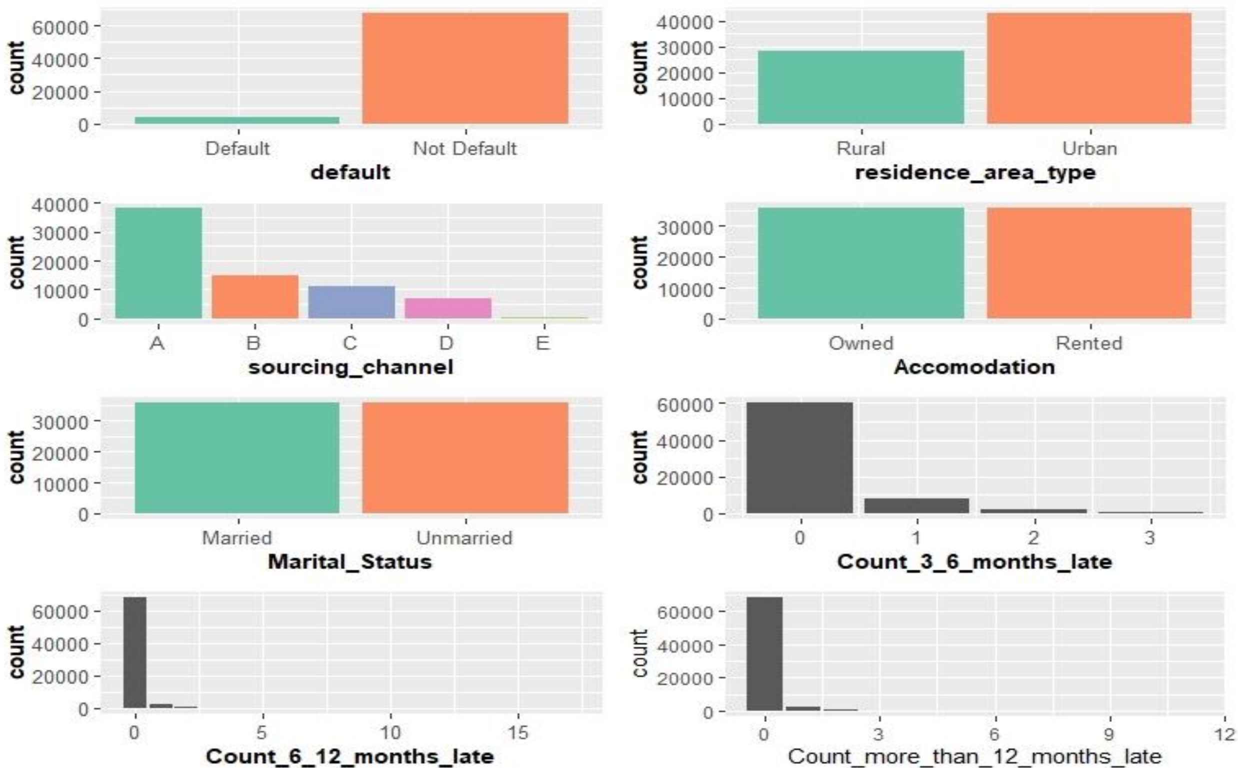
Relationship among variables, important variables

Five Numbers

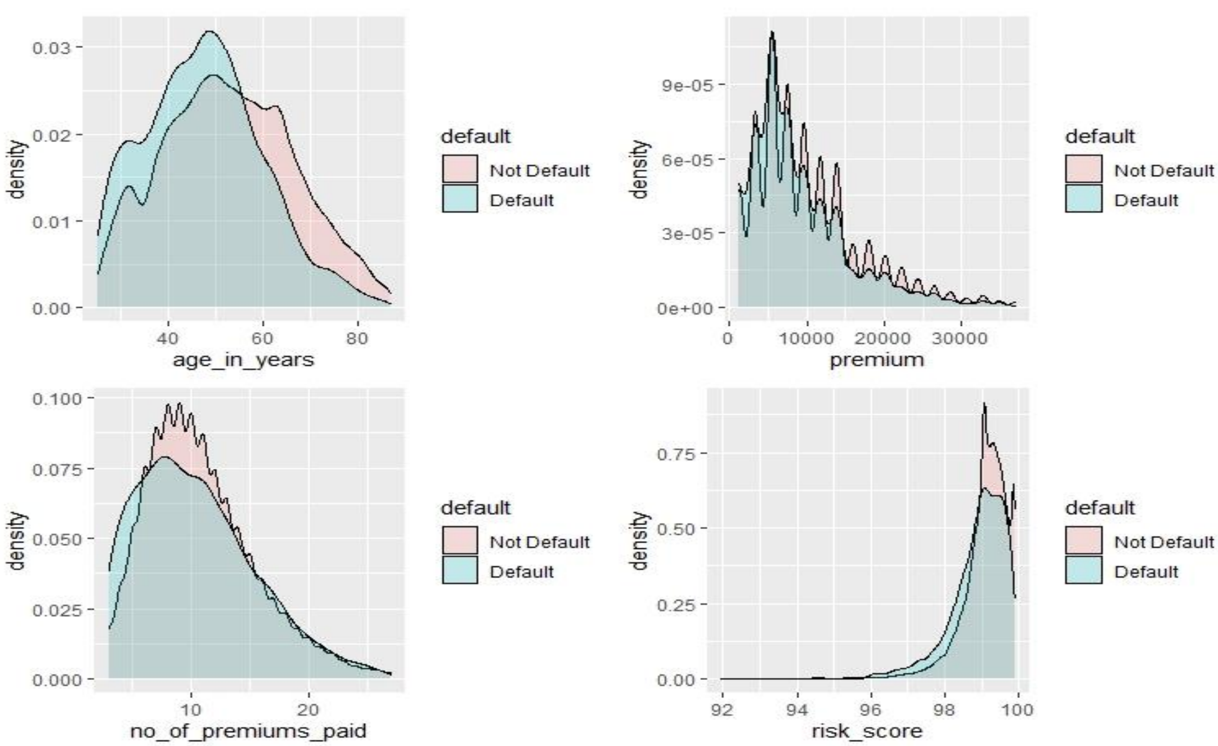
Variable	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
age_in_years	25.0	42.0	52.0	53.6	62.0	87
Premium	1200	5400	7500	12960	13800	36900
no_of_premiums_paid	3.0	7.0	10.0	12.0	13.0	27.0
risk_score	91.96	98.83	99.18	97.87	99.51	99.89
No_of_dep	1	2	3	2.6	3	4
Veh_Owned	1	1	2	2	3	3

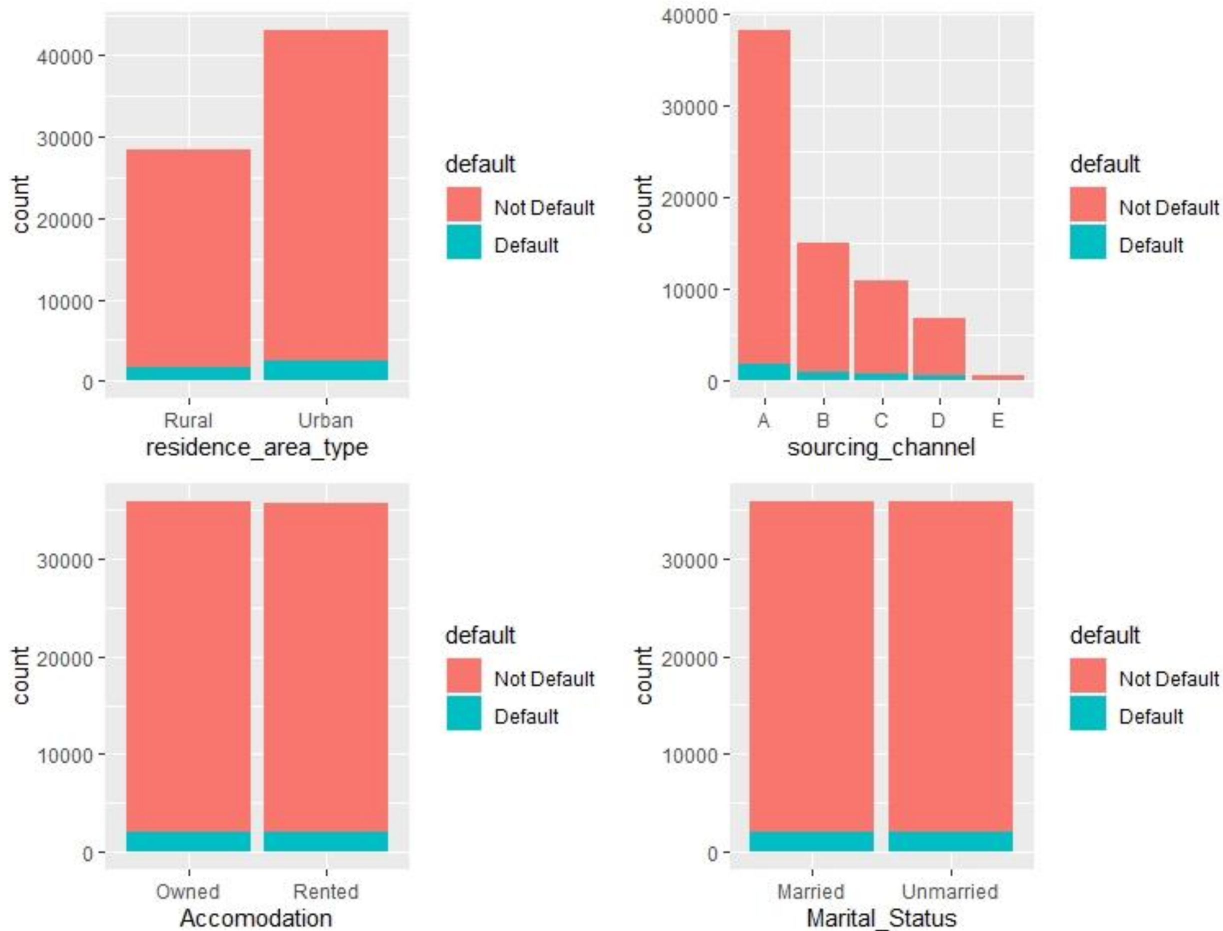
Insightful Visualizations

UNIVARIATE ANALYSIS

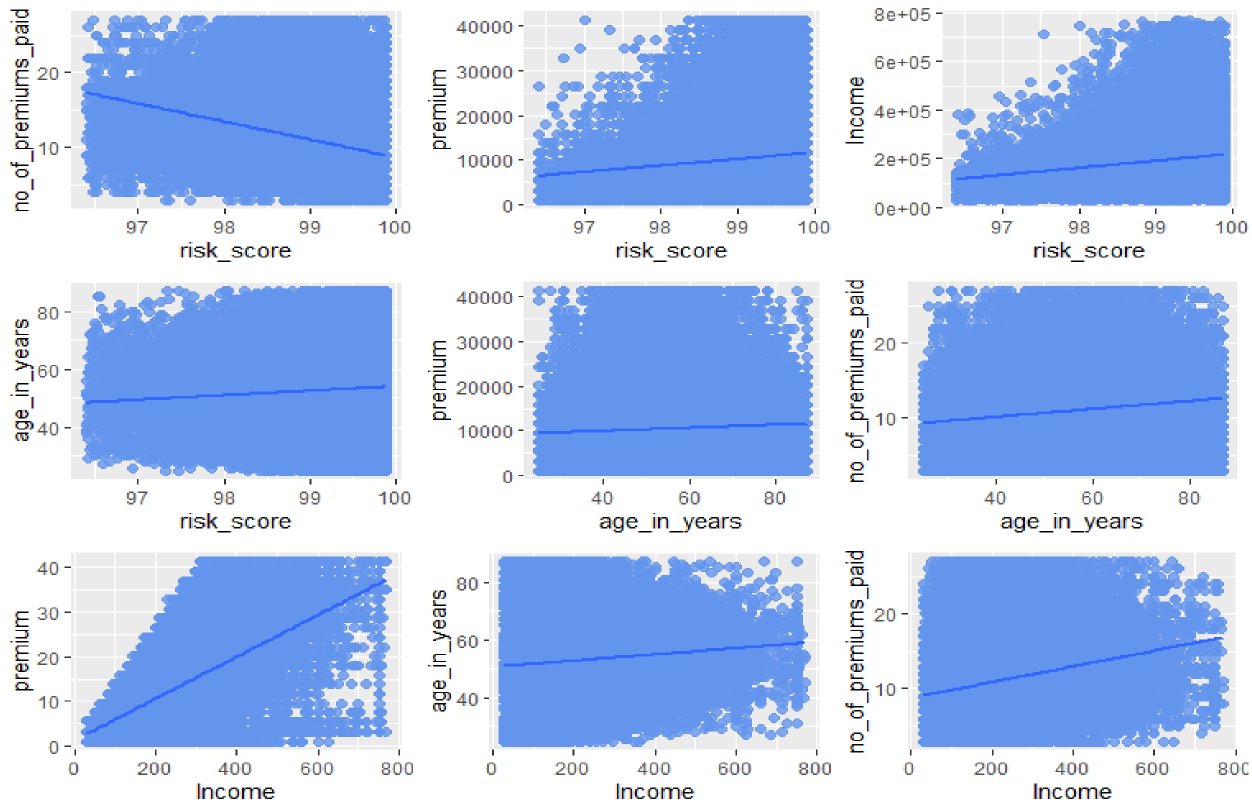


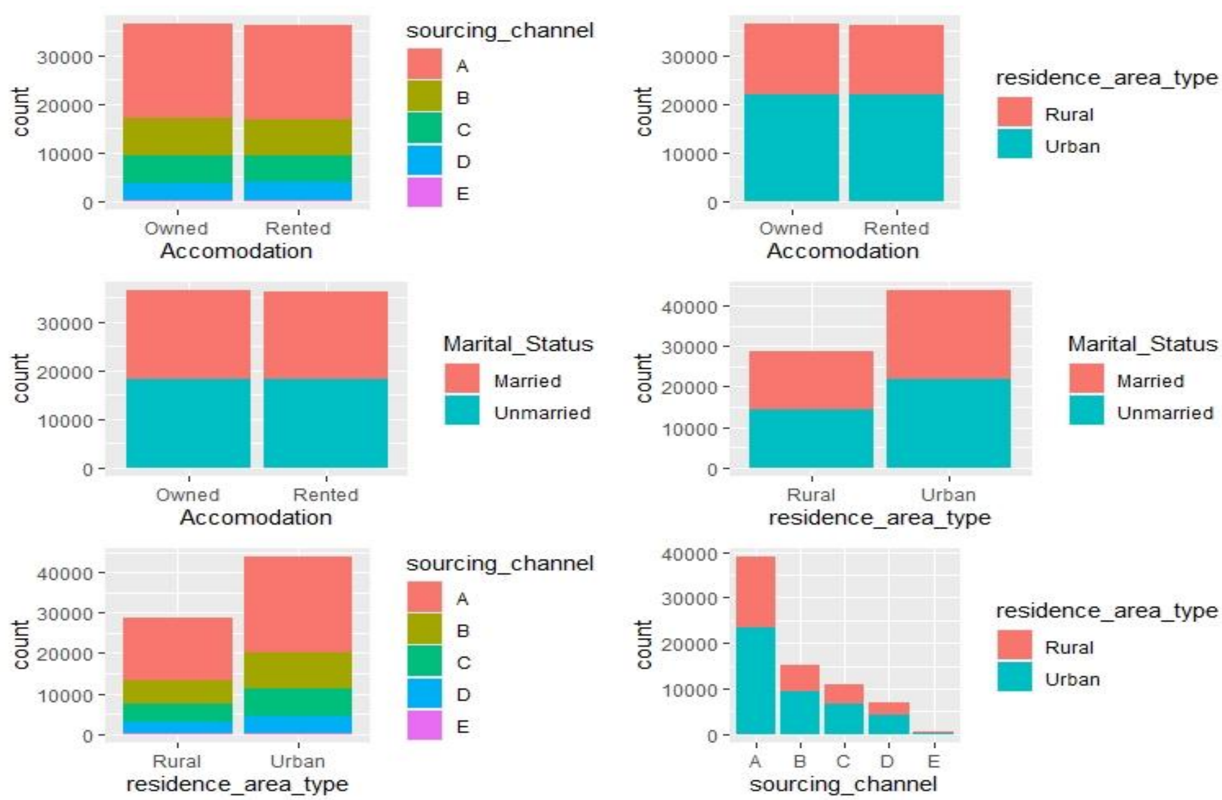
BIVARIATE ANALYSIS




count
40000
30000
20000
10000
0
Rural
Urban
residence_area_type
default
Not Default
Default
sourcing_channel
A
B
C
D
E
Accomodation
Owned
Rented
Marital_Status
Married
Unmarried

Correlation Analysis





Analytical approach

We are going to use Random forest and logistic regression techniques to build the models and then we will compare the results to choice the best model.

We will divide dataset into two parts as known as train data and test data as (80 %, 20%) respectively.

We will use confusion matrix, AUC, sensitivity and specificity techniques to compare the models and chose the best one for our case.

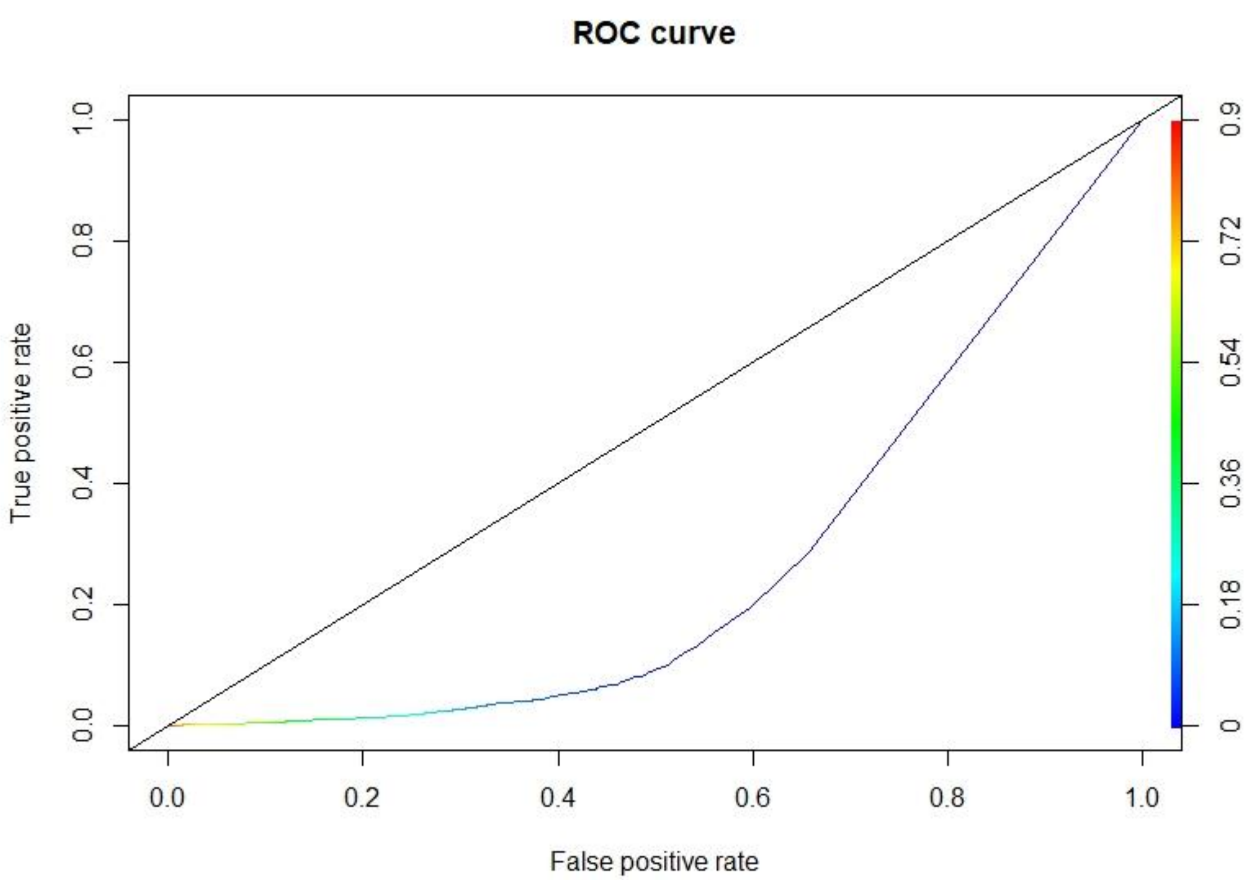
Modelling Process

As we mention previously, we are going to use Random forest and logistic regression techniques to solve this problem.

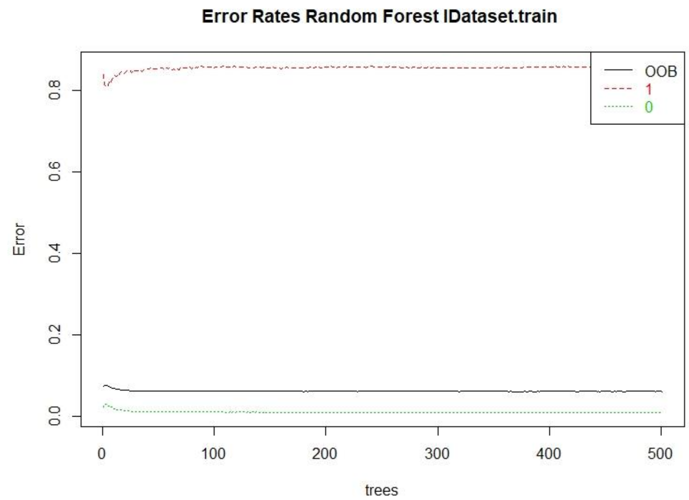
Random Forest

After we split data to train and test dataset (80%, 20%), we start with randomForest method in randomForest package, first we realize that using calculated AUC the best combination of variables as follows:

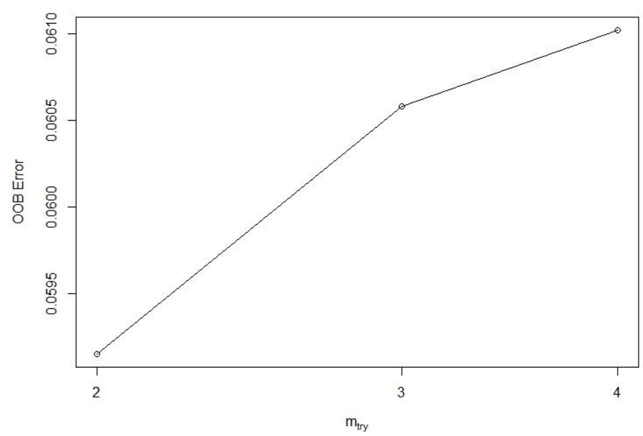
ncome+Count_3_6_months_late+Count_6_12_months_late+Count_more_than_12_months_late+risk_score



so we start building model and then find the best number of trees is 251.



ow we will "tune" the Random Forest by trying different m values, We will stick with 251 trees (odd number of trees are preferable). The returned forest, "tRndFor" is the one corresponding to the best m



List the importance of the variables. Larger the MeanDecrease values the more important the variable.

```
> importance(tRndFor)
                                      0         1 MeanDecreaseAccuracy MeanDecreaseGini
Income                         2.087238 16.657514             16.05497         430.8145
Count_3_6_months_late         57.145825 -2.797534             27.21822         316.4784
Count_6_12_months_late       121.721833 26.158026             68.58116         571.8026
Count_more_than_12_months_late 65.178440  4.587669            34.77560         285.7453
risk_score                     9.831201 12.097804             16.12197         363.0098
```

Lets make predictions on the training data and measure the prediction error rate.

```
> print('accuracy is ')
```

```
[1] "accuracy is "
> sum(diag(tb.train))/sum(tb.train)
[1] 0.9512413
```

Now using the tuned Random Forest from the previous step, and make prediction on test data

```
> print('accuracy is ')
[1] "accuracy is "
> sum(diag(tb.test))/sum(tb.test)
[1] 0.9419294
```

Finaly we get this result

- #trainData accuracy = 95
- #trainData sensitivity = 99
- #trainData specificity = 20
- #testData accuracy = 94
- #testData sensitivity = 99
- #testData specificity = 11

Please refer Appendix A for Source Code.

Logistic Regression

After Split the data into train and test dataset. (80%, 20%) We apply logistic regression known function glm using calculated AUC the best combination of variables as follows:

ncome+Count_3_6_months_late+Count_6_12_months_late+Count_more_than_12_months_late+risk_score

we get this summary

```
> summary(LRmodel)

Call:
glm(formula = default ~ Income + Count_3_6_months_late + Count_6_12_months_la
te + 
    Count_more_than_12_months_late + risk_score, family = binomial, 
    data = IDataset.train2)

Deviance Residuals: 
    Min       1Q   Median       3Q      Max  
-2.9891   0.2245   0.2466   0.2867   2.4023  

Coefficients:
                                   Estimate Std. Error z value Pr(>|z|)    
(Intercept)                      -3.469e+01  2.883e+00 -12.032  < 2e-16 ***
Income                            1.273e-06  1.939e-07   6.567 5.12e-11 ***
Count_3_6_months_late1           -9.283e-01  4.815e-02 -19.280  < 2e-16 ***
Count_3_6_months_late2           -1.365e+00  6.792e-02 -20.094  < 2e-16 ***
Count_3_6_months_late3           -1.578e+00  9.898e-02 -15.946  < 2e-16 ***
Count_3_6_months_late4           -1.962e+00  1.414e-01 -13.878  < 2e-16 ***
Count_3_6_months_late5           -1.782e+00  2.124e-01  -8.386  < 2e-16 ***
```

```
Count_3_6_months_late6          -1.955e+00  3.451e-01  -5.663 1.49e-08 ***
Count_3_6_months_late7          -2.174e+00  5.493e-01  -3.958 7.56e-05 ***
Count_3_6_months_late8          -2.130e+00  8.028e-01  -2.653  0.00798 **
Count_3_6_months_late9          -2.950e+00  1.408e+00  -2.096  0.03611 *
Count_3_6_months_late10         -1.214e+01  5.354e+02  -0.023  0.98191
Count_3_6_months_late11         -1.412e+01  5.354e+02  -0.026  0.97896
Count_3_6_months_late12         -1.570e+01  5.354e+02  -0.029  0.97661
Count_3_6_months_late13         -1.726e+01  5.354e+02  -0.032  0.97429
Count_6_12_months_late1         -1.561e+00  6.085e-02 -25.652  < 2e-16 ***
Count_6_12_months_late2         -2.214e+00  1.065e-01 -20.790  < 2e-16 ***
Count_6_12_months_late3         -2.656e+00  1.532e-01 -17.333  < 2e-16 ***
Count_6_12_months_late4         -2.821e+00  2.351e-01 -12.001  < 2e-16 ***
Count_6_12_months_late5         -3.167e+00  3.857e-01  -8.211  < 2e-16 ***
Count_6_12_months_late6         -1.937e+00  5.208e-01  -3.719  0.00020 ***
Count_6_12_months_late7         -3.679e+00  8.919e-01  -4.125 3.71e-05 ***
Count_6_12_months_late8         -3.159e+00  1.461e+00  -2.163  0.03057 *
Count_6_12_months_late9          1.012e+01  3.780e+02   0.027  0.97865
Count_6_12_months_late10         1.102e+01  5.354e+02   0.021  0.98358
Count_6_12_months_late11        -2.249e+00  1.936e+00  -1.162  0.24538
Count_6_12_months_late12        -1.456e+01  5.354e+02  -0.027  0.97830
Count_6_12_months_late13        -1.566e+01  5.354e+02  -0.029  0.97667
Count_6_12_months_late14         1.204e+01  5.354e+02   0.022  0.98205
Count_6_12_months_late15         1.090e+01  5.354e+02   0.020  0.98376
Count_6_12_months_late17        -1.699e+01  5.354e+02  -0.032  0.97468
Count_more_than_12_months_late1  -1.090e+00  6.130e-02 -17.785  < 2e-16 ***
Count_more_than_12_months_late2  -1.516e+00  1.219e-01 -12.430  < 2e-16 ***
Count_more_than_12_months_late3  -1.202e+00  2.290e-01  -5.247 1.55e-07 ***
Count_more_than_12_months_late4  -4.271e-01  3.679e-01  -1.161  0.24571
Count_more_than_12_months_late5  -1.713e+00  8.648e-01  -1.981  0.04763 *
Count_more_than_12_months_late6  -1.459e+01  2.353e+02  -0.062  0.95057
Count_more_than_12_months_late7  -1.711e+00  2.585e+00  -0.662  0.50809
Count_more_than_12_months_late8  -1.294e+01  5.354e+02  -0.024  0.98072
Count_more_than_12_months_late11 -1.481e+01  5.354e+02  -0.028  0.97793
risk_score                       3.826e-01  2.920e-02  13.104  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 26927  on 59373  degrees of freedom
Residual deviance: 21931  on 59333  degrees of freedom
AIC: 22013

Number of Fisher Scoring iterations: 12
```

important variables

```
                         newColName     Overall
15          Count_6_12_months_late1 25.65161232
16          Count_6_12_months_late2 20.78954626
3            Count_3_6_months_late2 20.09360270
2            Count_3_6_months_late1 19.27993773
31  Count_more_than_12_months_late1 17.78483640
17          Count_6_12_months_late3 17.33317429
4            Count_3_6_months_late3 15.94593960
5            Count_3_6_months_late4 13.87846002
40                       risk_score 13.10430549
32  Count_more_than_12_months_late2 12.42975245
18          Count_6_12_months_late4 12.00071310
6            Count_3_6_months_late5  8.38646951
19          Count_6_12_months_late5  8.21052702
```

```
1                               Income   6.56730612
7               Count_3_6_months_late6   5.66314699
33  Count_more_than_12_months_late3   5.24663015
21             Count_6_12_months_late7   4.12512121
8               Count_3_6_months_late7   3.95806124
20             Count_6_12_months_late6   3.71908827
9               Count_3_6_months_late8   2.65283042
22             Count_6_12_months_late8   2.16268819
10              Count_3_6_months_late9   2.09569159
35  Count_more_than_12_months_late5   1.98066902
25            Count_6_12_months_late11   1.16164609
34  Count_more_than_12_months_late4   1.16084006
37  Count_more_than_12_months_late7   0.66181236
36  Count_more_than_12_months_late6   0.06199110
14             Count_3_6_months_late13   0.03223092
30            Count_6_12_months_late17   0.03174036
13             Count_3_6_months_late12   0.02932111
27            Count_6_12_months_late13   0.02924970
39 Count_more_than_12_months_late11   0.02766113
26            Count_6_12_months_late12   0.02719766
23             Count_6_12_months_late9   0.02676606
12             Count_3_6_months_late11   0.02637302
38  Count_more_than_12_months_late8   0.02416165
11             Count_3_6_months_late10   0.02267659
28            Count_6_12_months_late14   0.02249392
24            Count_6_12_months_late10   0.02058355
29            Count_6_12_months_late15   0.02035602
```

lets see model performances on train data

```
> print('accuracy is ')
[1] "accuracy is "
> sum(diag(tb.train2))/sum(tb.train2)
[1] 0.9423653
```

let us check accuracy on test data set

```
> print('accuracy is ')
[1] "accuracy is "
> sum(diag(tb.test2))/sum(tb.test2)
[1] 0.9399084
```

Finaly we get this result

- #trainData accuracy = 94
- #trainData sensitivity = 94
- #trainData specificity = 57
- #testData accuracy = 93
- #testData sensitivity = 94
- #testData specificity = 51

Please refer Appendix A for Source Code.

Model comparisons

Model	Accuracy on train data	Accuracy on test data	sensitivity on train data	sensitivity on test data	specificity on train data	specificity on test data
Random Forest	95	94	99	99	20	11
Logistic Regression	94	93	94	94	57	51

Please refer Appendix A for Source Code.

Interpretation from the best model

While the Accuracy of first technique (Random Forest) is better than the second one (Logistic Regression), also sensitivity do well.

We can say the first technique (Random Forest) is the best.

Business insights and Recommendations

The important variable is

- Count_6_12_months_late
- Count_3_6_months_late
- Count_more_than_12_months_late
- risk_score

this is mean that the **history of the customer premium paid** is significantly and has big impact on the target variable Default.

Appendix A – Source Code

```
library(readxl)
library(plyr)
library(ggplot2)


Insurance_Premium_Default_Dataset <- read_excel("Data/Capstone
Project/Insurance Premium Default-Dataset.xlsx")
View(Insurance_Premium_Default_Dataset)



#Retrieve the dimension of an object.
dim(Insurance_Premium_Default_Dataset)

#Get the names of an object.
names(Insurance_Premium_Default_Dataset)

#Display the internal structure of an dataset.
str(Insurance_Premium_Default_Dataset)

#Returns the first 10 rows of the dataset.
head(Insurance_Premium_Default_Dataset, 10)

#Returns the last 10 rows of the dataset.
tail(Insurance_Premium_Default_Dataset, 10)

#Return a summary of the dataset variables.
summary(Insurance_Premium_Default_Dataset)

#check if ther is any NA value in dataset
anyNA(Insurance_Premium_Default_Dataset)


#preparing variables
```

```r
#generate age_in_years
Insurance_Premium_Default_Dataset$age_in_years =
as.integer(format(round(Insurance_Premium_Default_Dataset$age_in_days/360,
0), nsmall = 0))



#convert from quantitative to qualitative

#Marital_Status
Insurance_Premium_Default_Dataset$`Marital Status` <-
factor(Insurance_Premium_Default_Dataset$`Marital Status`, order = F, levels
=c("1","0"))
Insurance_Premium_Default_Dataset$Marital_Status <-
factor(mapvalues(Insurance_Premium_Default_Dataset$`Marital Status`, from =
c("1", "0"), to = c("Married","Unmarried")))


#Accomodation
Insurance_Premium_Default_Dataset$Accomodation <-
factor(Insurance_Premium_Default_Dataset$Accomodation, order = F, levels
=c("1","0"))
Insurance_Premium_Default_Dataset$Accomodation <-
factor(mapvalues(Insurance_Premium_Default_Dataset$Accomodation, from =
c("1", "0"), to = c("Owned","Rented")))

#sourcing_channel
Insurance_Premium_Default_Dataset$sourcing_channel =
as.factor(Insurance_Premium_Default_Dataset$sourcing_channel)

#residence_area_type
Insurance_Premium_Default_Dataset$residence_area_type =
as.factor(Insurance_Premium_Default_Dataset$residence_area_type)

#default

Insurance_Premium_Default_Dataset$default =
as.factor(Insurance_Premium_Default_Dataset$default)
Insurance_Premium_Default_Dataset$default <-
factor(mapvalues(Insurance_Premium_Default_Dataset$default, from = c("1",
"0"), to = c("Not Default", "Default")))


#objects in the dataset can be accessed by simply giving their names
attach(Insurance_Premium_Default_Dataset)


summary(Insurance_Premium_Default_Dataset)



# Load DataExplorer for exploratory data analysis.
library(DataExplorer)

# This function helps to visualize data structure in network graph format.
```

```r
plot_str(Insurance_Premium_Default_Dataset, type="d", fontSize = 25)
# plot missing data
plot_missing(Insurance_Premium_Default_Dataset)


# Check the fivenumber summary of variables
summary(fivenum(Insurance_Premium_Default_Dataset$age_in_years))
summary(fivenum(Insurance_Premium_Default_Dataset$premium))
summary(fivenum(Insurance_Premium_Default_Dataset$no_of_premiums_paid))
summary(fivenum(Insurance_Premium_Default_Dataset$risk_score))
summary(fivenum(Insurance_Premium_Default_Dataset$No_of_dep))
summary(fivenum(Insurance_Premium_Default_Dataset$Veh_Owned))
summary(fivenum(Insurance_Premium_Default_Dataset$Income))




### UNIVARIATE ANALYSIS

library(ggplot2)
library(grid)
library(gridExtra)
## visualize properties of all categorical variables


# Setting up the aesthetics
unipar = theme(legend.position = "none") +
  theme(axis.text = element_text(size = 10),
        axis.title = element_text(size = 11),
        title = element_text(size = 13, face = "bold"))

# Define color brewer
col1 = "Set2"

# Plotting the bar charts
g1=ggplot(Insurance_Premium_Default_Dataset, aes(x=default, fill=default)) +
geom_bar()+ unipar + scale_fill_brewer(palette=col1)

# Plotting the bar charts
g2=ggplot(Insurance_Premium_Default_Dataset, aes(x=residence_area_type,
fill=residence_area_type)) + geom_bar()+ unipar +
scale_fill_brewer(palette=col1)

# Plotting the bar charts
g3=ggplot(Insurance_Premium_Default_Dataset, aes(x=sourcing_channel,
fill=sourcing_channel)) + geom_bar()+ unipar +
scale_fill_brewer(palette=col1)

# Plotting the bar charts
g4=ggplot(Insurance_Premium_Default_Dataset, aes(x=Accomodation,
fill=Accomodation)) + geom_bar()+ unipar + scale_fill_brewer(palette=col1)

# Plotting the bar charts
```

```
g5=ggplot(Insurance_Premium_Default_Dataset, aes(x=Marital_Status,
fill=`Marital_Status`)) + geom_bar()+ unipar +
scale_fill_brewer(palette=col1)

# Plotting the bar charts
g6=ggplot(Insurance_Premium_Default_Dataset, aes(x=`Count_3-6_months_late`,
fill=`Count_3-6_months_late`)) + geom_bar()+ unipar +
scale_fill_brewer(palette=col1)

# Plotting the bar charts
g7=ggplot(Insurance_Premium_Default_Dataset, aes(x=`Count_6-12_months_late`,
fill=`Count_6-12_months_late`)) + geom_bar()+ unipar +
scale_fill_brewer(palette=col1)




# Plotting the bar charts
g8=ggplot(Insurance_Premium_Default_Dataset) + geom_bar(aes(x =
Count_more_than_12_months_late)) + scale_fill_brewer(palette=col2)


# Partitioning the barcharts
grid.arrange(g1,g2,g3,g4,g5,g6,g7,g8,ncol=2)


### BIVARIATE ANALYSIS


# Setting up the aesthetics
bipar1 = theme(legend.position = "none") + theme_light() +
  theme(axis.text = element_text(size = 10),
        axis.title = element_text(size = 11),
        title = element_text(size = 13, face = "bold"))

# Define color brewer
col2 = "Set2"

# default vs numerical variables
p1=ggplot(Insurance_Premium_Default_Dataset,
          aes(x = age_in_years, #quantitative variable
              fill = factor(default,
                            levels = c("Not Default", "Default"),
                            labels = c("Not Default", "Default")))) +
  geom_density(alpha = 0.2) + #setting transparency of graph to keep overlaps
visible
  labs(fill = "default", # setting title of legend
       x = "age_in_years")


p2=ggplot(Insurance_Premium_Default_Dataset,
          aes(x = premium, #quantitative variable
              fill = factor(default,
                            levels = c("Not Default", "Default"),
                            labels = c("Not Default", "Default")))) +
```

```r
  geom_density(alpha = 0.2) + #setting transparency of graph to keep overlaps
visible
  labs(fill = "default", # setting title of legend
       x = "premium")


p3=ggplot(Insurance_Premium_Default_Dataset,
          aes(x = no_of_premiums_paid, #quantitative variable
              fill = factor(default,
                            levels = c("Not Default", "Default"),
                            labels = c("Not Default", "Default")))) +
  geom_density(alpha = 0.2) + #setting transparency of graph to keep overlaps
visible
  labs(fill = "default", # setting title of legend
       x = "no_of_premiums_paid")


p4=ggplot(Insurance_Premium_Default_Dataset,
          aes(x = risk_score, #quantitative variable
              fill = factor(default,
                            levels = c("Not Default", "Default"),
                            labels = c("Not Default", "Default")))) +
  geom_density(alpha = 0.2) + #setting transparency of graph to keep overlaps
visible
  labs(fill = "default", # setting title of legend
       x = "risk_score")

# Partitioning the boxplots
grid.arrange(p1,p2,p3,p4,ncol=2)




# Setting up the aesthetics
bipar2 = theme(legend.position = "top",
               legend.direction = "horizontal",
               legend.title = element_text(size = 10),
               legend.text = element_text(size = 8)) +
  theme(axis.text = element_text(size = 10),
        axis.title = element_text(size = 11),
        title = element_text(size = 13, face = "bold"))


library(dplyr)

# default vs categorical variables
# stacked bar chart
p8 = ggplot(Insurance_Premium_Default_Dataset,
         aes(x = residence_area_type,
             fill = factor(default,
                           levels = c("Not Default", "Default"),
                           labels = c("Not Default", "Default")))) +
  labs(fill = "default", # setting title of legend
       x = "residence_area_type",
       title = "Custome default by residence_area_type") +
  geom_bar(position = "stack") #specifying the type of bar chart as stacked
```

```r
p9 = ggplot(Insurance_Premium_Default_Dataset,
            aes(x = sourcing_channel,
                fill = factor(default,
                              levels = c("Not Default", "Default"),
                              labels = c("Not Default", "Default")))) +
  labs(fill = "default", # setting title of legend
       x = "sourcing_channel",
       title = "Custome default by sourcing_channel") +
  geom_bar(position = "stack") #specifying the type of bar chart as stacked


p10 = ggplot(Insurance_Premium_Default_Dataset,
            aes(x = Accomodation,
                fill = factor(default,
                              levels = c("Not Default", "Default"),
                              labels = c("Not Default", "Default")))) +
  labs(fill = "default", # setting title of legend
       x = "Accomodation",
       title = "Custome default by Accomodation") +
  geom_bar(position = "stack") #specifying the type of bar chart as stacked

p11 = ggplot(Insurance_Premium_Default_Dataset,
             aes(x = Marital_Status,
                 fill = factor(default,
                               levels = c("Not Default", "Default"),
                               labels = c("Not Default", "Default")))) +
  labs(fill = "default", # setting title of legend
       x = "Marital_Status",
       title = "Custome default by Marital_Status") +
  geom_bar(position = "stack") #specifying the type of bar chart as stacked

# Partitioning the boxplots
grid.arrange(p8,p9,p10,p11,ncol=2)



# removing unwante
IDataset = Insurance_Premium_Default_Dataset[,c(2, 4 ,5, 6, 7, 9, 10, 11, 12,
13, 14, 15, 16, 18, 19, 17 ) ]



IDataset = IDataset %>%
  rename(
    Count_3_6_months_late = `Count_3-6_months_late`,
    Count_6_12_months_late = `Count_6-12_months_late`
  )


attach(IDataset)

#outlier treatment
```

```
#income
lower_bound <- quantile(IDataset$Income, 0.01)
upper_bound <- quantile(IDataset$Income, 0.99)

outlier_ind <- which(IDataset$Income < lower_bound | IDataset$Income >
upper_bound)

if( length(outlier_ind) > 0)
IDataset = IDataset[-outlier_ind, ]




#perc_premium_paid_by_cash_credit
lower_bound <- quantile(IDataset$perc_premium_paid_by_cash_credit, 0.01)
upper_bound <- quantile(IDataset$perc_premium_paid_by_cash_credit, 0.99)

outlier_ind <- which(IDataset$perc_premium_paid_by_cash_credit < lower_bound
| IDataset$perc_premium_paid_by_cash_credit > upper_bound)

if( length(outlier_ind) > 0)
  IDataset = IDataset[-outlier_ind, ]




#Count_3_6_months_late
lower_bound <- quantile(IDataset$Count_3_6_months_late, 0.01)
upper_bound <- quantile(IDataset$Count_3_6_months_late, 0.99)

outlier_ind <- which(IDataset$Count_3_6_months_late < lower_bound |
IDataset$Count_3_6_months_late > upper_bound)

if( length(outlier_ind) > 0)
  IDataset = IDataset[-outlier_ind, ]




#Count_6_12_months_late
lower_bound <- quantile(IDataset$Count_6_12_months_late, 0.01)
upper_bound <- quantile(IDataset$Count_6_12_months_late, 0.99)

outlier_ind <- which(IDataset$Count_6_12_months_late < lower_bound |
IDataset$Count_6_12_months_late > upper_bound)

if( length(outlier_ind) > 0)
  IDataset = IDataset[-outlier_ind, ]




#Count_more_than_12_months_late
lower_bound <- quantile(IDataset$Count_more_than_12_months_late, 0.01)
upper_bound <- quantile(IDataset$Count_more_than_12_months_late, 0.99)
```

```r
outlier_ind <- which(IDataset$Count_more_than_12_months_late < lower_bound |
IDataset$Count_more_than_12_months_late > upper_bound)

if( length(outlier_ind) > 0)
  IDataset = IDataset[-outlier_ind, ]



#Veh_Owned
lower_bound <- quantile(IDataset$Veh_Owned, 0.01)
upper_bound <- quantile(IDataset$Veh_Owned, 0.99)

outlier_ind <- which(IDataset$Veh_Owned < lower_bound | IDataset$Veh_Owned >
upper_bound)

if( length(outlier_ind) > 0)
  IDataset = IDataset[-outlier_ind, ]



#No_of_dep
lower_bound <- quantile(IDataset$No_of_dep, 0.01)
upper_bound <- quantile(IDataset$No_of_dep, 0.99)

outlier_ind <- which(IDataset$No_of_dep < lower_bound | IDataset$No_of_dep >
upper_bound)

if( length(outlier_ind) > 0)
  IDataset = IDataset[-outlier_ind, ]




#risk_score
lower_bound <- quantile(IDataset$risk_score, 0.01)
upper_bound <- quantile(IDataset$risk_score, 0.99)

outlier_ind <- which(IDataset$risk_score < lower_bound | IDataset$risk_score
> upper_bound)

if( length(outlier_ind) > 0)
  IDataset = IDataset[-outlier_ind, ]



#no_of_premiums_paid
lower_bound <- quantile(IDataset$no_of_premiums_paid, 0.01)
upper_bound <- quantile(IDataset$no_of_premiums_paid, 0.99)

outlier_ind <- which(IDataset$no_of_premiums_paid < lower_bound |
IDataset$no_of_premiums_paid > upper_bound)

if( length(outlier_ind) > 0)
```

```r
  IDataset = IDataset[-outlier_ind, ]




#premium
lower_bound <- quantile(IDataset$premium, 0.01)
upper_bound <- quantile(IDataset$premium, 0.99)

outlier_ind <- which(IDataset$premium < lower_bound | IDataset$premium >
upper_bound)

if( length(outlier_ind) > 0)
  IDataset = IDataset[-outlier_ind, ]



#age_in_years
lower_bound <- quantile(IDataset$age_in_years, 0.01)
upper_bound <- quantile(IDataset$age_in_years, 0.99)

outlier_ind <- which(IDataset$age_in_years < lower_bound |
IDataset$age_in_years > upper_bound)

if( length(outlier_ind) > 0)
  IDataset = IDataset[-outlier_ind, ]

#EDA again

# Check the fivenumber summary of variables
summary(fivenum(IDataset$age_in_years))
summary(fivenum(IDataset$premium))
summary(fivenum(IDataset$no_of_premiums_paid))
summary(fivenum(IDataset$risk_score))
summary(fivenum(IDataset$No_of_dep))
summary(fivenum(IDataset$Veh_Owned))
summary(fivenum(IDataset$Income))




### UNIVARIATE ANALYSIS

library(ggplot2)
library(grid)
library(gridExtra)
## visualize properties of all categorical variables


# Setting up the aesthetics
unipar = theme(legend.position = "none") +
  theme(axis.text = element_text(size = 10),
        axis.title = element_text(size = 11),
```

```
        title = element_text(size = 13, face = "bold"))

# Define color brewer
col1 = "Set2"

# Plotting the bar charts
g1=ggplot(IDataset, aes(x=default, fill=default)) + geom_bar()+ unipar +
scale_fill_brewer(palette=col1)

# Plotting the bar charts
g2=ggplot(IDataset, aes(x=residence_area_type, fill=residence_area_type)) +
geom_bar()+ unipar + scale_fill_brewer(palette=col1)

# Plotting the bar charts
g3=ggplot(IDataset, aes(x=sourcing_channel, fill=sourcing_channel)) +
geom_bar()+ unipar + scale_fill_brewer(palette=col1)

# Plotting the bar charts
g4=ggplot(IDataset, aes(x=Accomodation, fill=Accomodation)) + geom_bar()+
unipar + scale_fill_brewer(palette=col1)

# Plotting the bar charts
g5=ggplot(IDataset, aes(x=Marital_Status, fill=Marital_Status)) + geom_bar()+
unipar + scale_fill_brewer(palette=col1)

# Plotting the bar charts
g6=ggplot(IDataset, aes(x=Count_3_6_months_late, fill=Count_3_6_months_late))
+ geom_bar()+ unipar + scale_fill_brewer(palette=col1)

# Plotting the bar charts
g7=ggplot(IDataset, aes(x=Count_6_12_months_late,
fill=Count_6_12_months_late)) + geom_bar()+ unipar +
scale_fill_brewer(palette=col1)




# Plotting the bar charts
g8=ggplot(IDataset) + geom_bar(aes(x = Count_more_than_12_months_late)) +
scale_fill_brewer(palette=col2)


# Partitioning the barcharts
grid.arrange(g1,g2,g3,g4,g5,g6,g7,g8,ncol=2)



### BIVARIATE ANALYSIS


# Setting up the aesthetics
bipar1 = theme(legend.position = "none") + theme_light() +
  theme(axis.text = element_text(size = 10),
        axis.title = element_text(size = 11),
        title = element_text(size = 13, face = "bold"))
```

```
# Define color brewer
col2 = "Set2"

# default vs numerical variables
p1=ggplot(IDataset,
          aes(x = age_in_years, #quantitative variable
              fill = factor(default,
                            levels = c("Not Default", "Default"),
                            labels = c("Not Default", "Default")))) +
  geom_density(alpha = 0.2) + #setting transparency of graph to keep overlaps
visible
  labs(fill = "default", # setting title of legend
       x = "age_in_years")


p2=ggplot(IDataset,
          aes(x = premium, #quantitative variable
              fill = factor(default,
                            levels = c("Not Default", "Default"),
                            labels = c("Not Default", "Default")))) +
  geom_density(alpha = 0.2) + #setting transparency of graph to keep overlaps
visible
  labs(fill = "default", # setting title of legend
       x = "premium")


p3=ggplot(IDataset,
          aes(x = no_of_premiums_paid, #quantitative variable
              fill = factor(default,
                            levels = c("Not Default", "Default"),
                            labels = c("Not Default", "Default")))) +
  geom_density(alpha = 0.2) + #setting transparency of graph to keep overlaps
visible
  labs(fill = "default", # setting title of legend
       x = "no_of_premiums_paid")


p4=ggplot(IDataset,
          aes(x = risk_score, #quantitative variable
              fill = factor(default,
                            levels = c("Not Default", "Default"),
                            labels = c("Not Default", "Default")))) +
  geom_density(alpha = 0.2) + #setting transparency of graph to keep overlaps
visible
  labs(fill = "default", # setting title of legend
       x = "risk_score")

# Partitioning the boxplots
grid.arrange(p1,p2,p3,p4,ncol=2)




# Setting up the aesthetics
bipar2 = theme(legend.position = "top",
               legend.direction = "horizontal",
```

```r
                legend.title = element_text(size = 10),
                legend.text = element_text(size = 8)) +
  theme(axis.text = element_text(size = 10),
        axis.title = element_text(size = 11),
        title = element_text(size = 13, face = "bold"))


library(dplyr)

# default vs categorical variables
# stacked bar chart
p8 = ggplot(IDataset,
            aes(x = residence_area_type,
                fill = factor(default,
                              levels = c("Not Default", "Default"),
                              labels = c("Not Default", "Default")))) +
  labs(fill = "default", # setting title of legend
       x = "residence_area_type") +
  geom_bar(position = "stack") #specifying the type of bar chart as stacked


p9 = ggplot(IDataset,
            aes(x = sourcing_channel,
                fill = factor(default,
                              levels = c("Not Default", "Default"),
                              labels = c("Not Default", "Default")))) +
  labs(fill = "default", # setting title of legend
       x = "sourcing_channel") +
  geom_bar(position = "stack") #specifying the type of bar chart as stacked


p10 = ggplot(IDataset,
             aes(x = Accomodation,
                 fill = factor(default,
                               levels = c("Not Default", "Default"),
                               labels = c("Not Default", "Default")))) +
  labs(fill = "default", # setting title of legend
       x = "Accomodation") +
  geom_bar(position = "stack") #specifying the type of bar chart as stacked

p11 = ggplot(IDataset,
             aes(x = Marital_Status,
                 fill = factor(default,
                               levels = c("Not Default", "Default"),
                               labels = c("Not Default", "Default")))) +
  labs(fill = "default", # setting title of legend
       x = "Marital_Status") +
  geom_bar(position = "stack") #specifying the type of bar chart as stacked

# Partitioning the boxplots
grid.arrange(p8,p9,p10,p11,ncol=2)



# correlation analysis
#scatter plot
```

```
c1 = ggplot(IDataset,aes(x = risk_score,y = no_of_premiums_paid)) +
  geom_point(color="cornflowerblue", #setting the colour, size and
transparency(alpha) of the points
             size = 2,
             alpha=.8) +
  labs(x = "risk_score", #specifying the labels of axes and title of plot
       y = "no_of_premiums_paid") +
  geom_smooth(method = "lm") # this adds a linear trend line which is useful
to summarize the relationship between the two variables



#scatter plot
c2 = ggplot(IDataset,aes(x = risk_score,y = premium)) +
  geom_point(color="cornflowerblue", #setting the colour, size and
transparency(alpha) of the points
             size = 2,
             alpha=.8) +
  labs(x = "risk_score", #specifying the labels of axes and title of plot
       y = "premium") +
  geom_smooth(method = "lm") # this adds a linear trend line which is useful
to summarize the relationship between the two variables


#scatter plot
c3 = ggplot(IDataset,aes(x = risk_score,y = Income)) +
  geom_point(color="cornflowerblue", #setting the colour, size and
transparency(alpha) of the points
             size = 2,
             alpha=.8) +
  labs(x = "risk_score", #specifying the labels of axes and title of plot
       y = "Income") +
  geom_smooth(method = "lm") # this adds a linear trend line which is useful
to summarize the relationship between the two variables



#scatter plot
c4 = ggplot(IDataset,aes(x = risk_score,y = age_in_years)) +
  geom_point(color="cornflowerblue", #setting the colour, size and
transparency(alpha) of the points
             size = 2,
             alpha=.8) +
  labs(x = "risk_score", #specifying the labels of axes and title of plot
       y = "age_in_years") +
  geom_smooth(method = "lm") # this adds a linear trend line which is useful
to summarize the relationship between the two variables



#scatter plot
c5 = ggplot(IDataset,aes(x = age_in_years,y = premium)) +
  geom_point(color="cornflowerblue", #setting the colour, size and
transparency(alpha) of the points
             size = 2,
             alpha=.8) +
```

```
  labs(x = "age_in_years", #specifying the labels of axes and title of plot
       y = "premium") +
  geom_smooth(method = "lm") # this adds a linear trend line which is useful
to summarize the relationship between the two variables


#scatter plot
c6 = ggplot(IDataset,aes(x = age_in_years,y = no_of_premiums_paid)) +
  geom_point(color="cornflowerblue", #setting the colour, size and
transparency(alpha) of the points
             size = 2,
             alpha=.8) +
  labs(x = "age_in_years", #specifying the labels of axes and title of plot
       y = "no_of_premiums_paid") +
  geom_smooth(method = "lm") # this adds a linear trend line which is useful
to summarize the relationship between the two variables



#scatter plot
c7 = ggplot(IDataset,aes(x = Income/1000,y = premium/1000)) +
  geom_point(color="cornflowerblue", #setting the colour, size and
transparency(alpha) of the points
             size = 2,
             alpha=.8) +
  labs(x = "Income", #specifying the labels of axes and title of plot
       y = "premium") +
  geom_smooth(method = "lm") # this adds a linear trend line which is useful
to summarize the relationship between the two variables




#scatter plot
c8 = ggplot(IDataset,aes(x = Income/1000,y = age_in_years)) +
  geom_point(color="cornflowerblue", #setting the colour, size and
transparency(alpha) of the points
             size = 2,
             alpha=.8) +
  labs(x = "Income", #specifying the labels of axes and title of plot
       y = "age_in_years") +
  geom_smooth(method = "lm") # this adds a linear trend line which is useful
to summarize the relationship between the two variables



#scatter plot
c9 = ggplot(IDataset,aes(x = Income/1000,y = no_of_premiums_paid)) +
  geom_point(color="cornflowerblue", #setting the colour, size and
transparency(alpha) of the points
             size = 2,
             alpha=.8) +
  labs(x = "Income", #specifying the labels of axes and title of plot
       y = "no_of_premiums_paid") +
```

```
  geom_smooth(method = "lm") # this adds a linear trend line which is useful
to summarize the relationship between the two variables


grid.arrange(c1,c2,c3,c4,c5,c6,c7,c8,c9,ncol=3)


# stacked bar chart
cc1 = ggplot(IDataset,
             aes(x = Accomodation,
                 fill = factor(sourcing_channel))) +
  labs(fill = "sourcing_channel", # setting title of legend
       x = "Accomodation") +
  geom_bar(position = "stack") #specifying the type of bar chart as stacked


cc2 = ggplot(IDataset,
             aes(x = Accomodation,
                 fill = factor(residence_area_type))) +
  labs(fill = "residence_area_type", # setting title of legend
       x = "Accomodation") +
  geom_bar(position = "stack") #specifying the type of bar chart as stacked


cc3 = ggplot(IDataset,
             aes(x = Accomodation,
                 fill = factor(Marital_Status))) +
  labs(fill = "Marital_Status", # setting title of legend
       x = "Accomodation") +
  geom_bar(position = "stack") #specifying the type of bar chart as stacked


cc4 = ggplot(IDataset,
             aes(x = residence_area_type,
                 fill = factor(Marital_Status))) +
  labs(fill = "Marital_Status", # setting title of legend
       x = "residence_area_type") +
  geom_bar(position = "stack") #specifying the type of bar chart as stacked


cc5 = ggplot(IDataset,
             aes(x = residence_area_type,
                 fill = factor(sourcing_channel))) +
  labs(fill = "sourcing_channel", # setting title of legend
       x = "residence_area_type") +
  geom_bar(position = "stack") #specifying the type of bar chart as stacked


cc6 = ggplot(IDataset,
```

```
             aes(x = sourcing_channel,
                 fill = factor(residence_area_type))) +
  labs(fill = "residence_area_type", # setting title of legend
       x = "sourcing_channel") +
  geom_bar(position = "stack") #specifying the type of bar chart as stacked

grid.arrange(cc1,cc2,cc3,cc4,cc5,cc6,ncol=2)
```

```
seed = 134

set.seed(seed)
index = sample(1:nrow(IDataset),0.80*nrow(IDataset))
IDataset.train1 = IDataset[index,]
IDataset.test1 = IDataset[-index,]


library(randomForest)

seed=1000

set.seed(seed)
```

```
#auc -> 0.1714639
#Income+Count_3_6_months_late+Count_6_12_months_late+Count_more_than_12_month
s_late
#IDataset.train1[, c(2,3,4,5)]


#auc -> 0.0501028
#Income+Count_3_6_months_late+Count_6_12_months_late+Count_more_than_12_month
s_late+risk_score
#IDataset.train1[, c(2,3,4,5,9)]


#auc -> 0.03329917
#age_in_years+Income+Count_3_6_months_late+Count_6_12_months_late+Count_more_
than_12_months_late+risk_score
#IDataset.train1[, c(14, 2,3,4,5,9]

#auc -> 0.05851644
#age_in_years+Income+Count_3_6_months_late+Count_6_12_months_late+Count_more_
than_12_months_late
#IDataset.train1[, c(14, 2,3,4,5)]


#auc -> 0.03141114
#age_in_years+Income+Count_3_6_months_late+Count_6_12_months_late+Count_more_
than_12_months_late+sourcing_channel+risk_score
#c(14, 2,3,4,5,11,9)
```

```
#auc -> 0.0266754
#age_in_years+Income+Count_3_6_months_late+Count_6_12_months_late+Count_more_
than_12_months_late+sourcing_channel+premium+risk_score
#c(14, 2,3,4,5,11,9,13)



#auc -> 0.01334393
#All

#build our random forest
rndFor = randomForest(default ~
Income+Count_3_6_months_late+Count_6_12_months_late+Count_more_than_12_months
_late+risk_score, data = IDataset.train1 ,
                      ntree=501, mtry = 3, nodesize = 10,
                      importance=TRUE)

#The error rate plot w.r.t number of trees reveals that anything more than,
say 51
# trees is really not that valuable.
rndFor$err.rate

plot(rndFor, main="")

legend("topright", c("OOB", "1", "0"), text.col=1:6, lty=1:3, col=1:3)

title(main="Error Rates Random Forest IDataset.train ")

set.seed(seed)


#Now we will "tune" the Random Forest by trying different m values.
#We will stick with 51 trees (odd number of trees are preferable).
#The returned forest, "tRndFor" is the one corresponding to the best m
tRndFor = tuneRF(x = IDataset.train1[, c(2,3,4,5,9)],
                 y= IDataset.train1 $default,
                 mtryStart = 3,
                 ntreeTry = 251,
                 stepFactor = 1.5,
                 improve = 0.0001,
                 trace=TRUE,
                 plot = TRUE,
                 doBest = TRUE,
                 nodesize = 10,
                 importance=TRUE
)

#List the importance of the variables. Larger the MeanDecrease values
#the more important the variable.
importance(tRndFor)


#Lets make predictions on the training data and measure the prediction error
rate.
```

```
IDataset.train1 $predict.class = predict(tRndFor, IDataset.train1 ,
type="class")

IDataset.train1 $prob1 = predict(tRndFor, IDataset.train1 ,
type="prob")[,"0"]

tb.train=table(IDataset.train1 $default, IDataset.train1 $predict.class)


print('accuracy is ')
sum(diag(tb.train))/sum(tb.train)

library(ROCR)
pred_ROCR <- prediction(IDataset.train1 $prob1, IDataset.train1 $default)
roc_ROCR <- performance(pred_ROCR, measure = "tpr", x.measure = "fpr")
plot(roc_ROCR, main = "ROC curve", colorize = T)
abline(a = 0, b = 1)

auc_ROCR <- performance(pred_ROCR, measure = "auc")
auc_ROCR <- auc_ROCR@y.values[[1]]


print('AUC is ')
auc_ROCR




trainsensitivity2 = tb.train[2,2] / sum(tb.train[2, ])
trainsensitivity2

trainspecificity2 = tb.train[1,1] / sum(tb.train[1, ])
trainspecificity2

#Now using the tuned Random Forest from the previous step,
#and redo our errors and top decile calculations for the previously
identified threshold.

IDataset.test1$predict.class = predict(tRndFor, IDataset.test1 ,
type="class")

IDataset.test1$prob1 = predict(tRndFor, IDataset.test1 , type="prob")[,"0"]

tb.test=table(IDataset.test1 $default, IDataset.test1 $predict.class)

print('accuracy is ')
sum(diag(tb.test))/sum(tb.test)




testsensitivity2 = tb.test[2,2] / sum(tb.test[2, ])
testsensitivity2

testspecificity2 = tb.test[1,1] / sum(tb.test[1, ])
testspecificity2
```

```
#result
#train accuracy = 95
#train sensitivity = 99
#train specificity = 20


#test accuracy = 94
#train sensitivity = 99
#train specificity = 11




##########################################
#with smoote
##########################################

# library(DMwR)

# set.seed(seed)
# index = sample(1:nrow(IDataset),0.80*nrow(IDataset))
# IDataset.train1 = IDataset[index,]
# IDataset.test1 = IDataset[-index,]


# ## Smote : Synthetic Minority Oversampling Technique To Handle default
Imbalancy In Binary Classification
# IDataset.train.balanced.data1 <- SMOTE(default ~.,
as.data.frame(IDataset.train1[, c(2,3,4,5,9,16)]), perc.over = 4800, k = 5,
perc.under = 1000)

# as.data.frame(table(IDataset.train.balanced.data1$default))



# ## Smote : Synthetic Minority Oversampling Technique To Handle default
Imbalancy In Binary Classification
# IDataset.test.balanced.data1 <- SMOTE(default ~.,
as.data.frame(IDataset.test1[, c(2,3,4,5,9,16)]), perc.over = 4800, k = 5,
perc.under = 1000)

# as.data.frame(table(IDataset.test.balanced.data1$default))





# library(randomForest)

# seed=1000

# set.seed(seed)
```

```
# #build our random forest
# rndFor = randomForest(default ~
Income+Count_3_6_months_late+Count_6_12_months_late+Count_more_than_12_months
_late+risk_score, data = IDataset.train.balanced.data1 ,
#                        ntree=501, mtry = 3, nodesize = 10,
#                        importance=TRUE)

# #The error rate plot w.r.t number of trees reveals that anything more than,
say 51
# # trees is really not that valuable.
# rndFor$err.rate

# plot(rndFor, main="")

# legend("topright", c("OOB", "1", "0"), text.col=1:6, lty=1:3, col=1:3)

# title(main="Error Rates Random Forest IDataset.train ")

# set.seed(seed)

# head(IDataset.train.balanced.data1)
# names(IDataset.train.balanced.data1)
# #Now we will "tune" the Random Forest by trying different m values.
# #We will stick with 51 trees (odd number of trees are preferable).
# #The returned forest, "tRndFor" is the one corresponding to the best m
# tRndFor = tuneRF(x = IDataset.train.balanced.data1[, -6],
#                   y= IDataset.train.balanced.data1 $default,
#                   mtryStart = 3,
#                   ntreeTry = 251,
#                   stepFactor = 1.5,
#                   improve = 0.0001,
#                   trace=TRUE,
#                   plot = TRUE,
#                   doBest = TRUE,
#                   nodesize = 10,
#                   importance=TRUE
# )

# #List the importance of the variables. Larger the MeanDecrease values
# #the more important the variable.
# importance(tRndFor)


# #Lets make predictions on the training data and measure the prediction
error rate.
# IDataset.train.balanced.data1 $predict.class = predict(tRndFor,
IDataset.train.balanced.data1 , type="class")

# IDataset.train.balanced.data1 $prob1 = predict(tRndFor,
IDataset.train.balanced.data1 , type="prob")[,"0"]

# tbl=table(IDataset.train.balanced.data1 $default,
IDataset.train.balanced.data1 $predict.class)
```

```
# print('accuracy is ')
# sum(diag(tbl))/sum(tbl)

# library(ROCR)
# pred_ROCR <- prediction(IDataset.train.balanced.data1 $prob1,
IDataset.train.balanced.data1 $default)
# roc_ROCR <- performance(pred_ROCR, measure = "tpr", x.measure = "fpr")
# plot(roc_ROCR, main = "ROC curve", colorize = T)
# abline(a = 0, b = 1)

# auc_ROCR <- performance(pred_ROCR, measure = "auc")
# auc_ROCR <- auc_ROCR@y.values[[1]]


# print('AUC is ')
# auc_ROCR

# #Now using the tuned Random Forest from the previous step,
# #and redo our errors and top decile calculations for the previously
identified threshold.

# IDataset.test.balanced.data1$predict.class = predict(tRndFor,
IDataset.test.balanced.data1 , type="class")

# IDataset.test.balanced.data1$prob1 = predict(tRndFor,
IDataset.test.balanced.data1 , type="prob")[,"0"]

# tbl=table(IDataset.test.balanced.data1 $default,
IDataset.test.balanced.data1 $predict.class)

# print('accuracy is ')
# sum(diag(tbl))/sum(tbl)

# pred_ROCR <- prediction(IDataset.test.balanced.data1 $prob1,
IDataset.test.balanced.data1 $default)
# roc_ROCR <- performance(pred_ROCR, measure = "tpr", x.measure = "fpr")
# plot(roc_ROCR, main = "ROC curve", colorize = T)
# abline(a = 0, b = 1)

# auc_ROCR <- performance(pred_ROCR, measure = "auc")
# auc_ROCR <- auc_ROCR@y.values[[1]]


# print('AUC is ')
# auc_ROCR





# #result
# #train accuracy = 0.93
# #test accuracy = 0.94


## Split the data into train & test dataset. Split80:20
```

```
seed = 101

set.seed(seed)
index = sample(1:nrow(IDataset),0.80*nrow(IDataset))
IDataset.train2 = IDataset[index,]
IDataset.test2 = IDataset[-index,]




## Let's check the count of unique value in the target variable
as.data.frame(table(IDataset.train2$default))

## Let's check the count of unique value in the target variable
as.data.frame(table(IDataset.test2$default))




# let us build the model with all varaibles

LRmodel = glm(default~
Income+Count_3_6_months_late+Count_6_12_months_late+Count_more_than_12_months
_late+risk_score , data = IDataset.train2, family= binomial)
summary(LRmodel)

# Using stepwise algorithm for removing insignificant variables

library(MASS)
log_model = stepAIC(LRmodel, direction = "both",k=5)
summary(log_model)


# lets see important variables

library(caret)
varImp(log_model)

# convert to data frame
l = data.frame(varImp(log_model))
l <- cbind(newColName = rownames(l), l)
rownames(l) <- 1:nrow(l)

# soritng the imprtance of varaible
l[with(l, order(-Overall)), ]

# lets see model performances on train data
# prediction on test dataset
IDataset.train2$prob1 = predict(log_model, newdata= IDataset.train2,
type="response")
tb.train2 = table(IDataset.train2$prob1>0.50, IDataset.train2$default)
print('accuracy is ')
sum(diag(tb.train2))/sum(tb.train2)
```

```
trainsensitivity2 = tb.train2[2,2] / sum(tb.train2[2, ])
trainsensitivity2

trainspecificity2 = tb.train2[1,1] / sum(tb.train2[1, ])
trainspecificity2

# let us check accuracy on test data set
# prediction on test dataset
IDataset.test2$prob1 = predict(log_model, newdata= IDataset.test2,
type="response")
tb.test2 = table(IDataset.test2$prob1>0.50, IDataset.test2$default)
print('accuracy is ')
sum(diag(tb.test2))/sum(tb.test2)




testsensitivity2 = tb.test2[2,2] / sum(tb.test2[2, ])
testsensitivity2

testspecificity2 = tb.test2[1,1] / sum(tb.test2[1, ])
testspecificity2


library(ineq)









#result
#train accuracy = 94
#train sensitivity = 94
#train specificity = 57


#test accuracy = 93
#train sensitivity = 94
#train specificity = 51
```