Prudential Life Insurance Assessment

1. Introduction

There is a sense of vulnerability when thinking about what the future would bring. Somehow, we need protection against the possibility of any losses. The history of Insurance contracts is dated back to medieval Europe when commerce and trading were expanding.

According with First American Life Insurance Company, "The first life insurance company in the United States began in 1735, for the benefit of the families of Presbyterian ministers."[1]. On those days, various religious authorities were outraged at the practice of putting a value on human life, but it was finally accepted as it was seen as a way to protect widows and orphans. Benjamin Franklin played a key role in popularizing the practice of insurance.

The internet changed the insurance industry. Cheapest rates can be found online for the right coverage. The process can be consider antiquated when customers provide extensive information to identify risk classification and eligibility, including scheduling medical exams, a process that takes an average of 30 days. That is why only 40% of U.S. households own individual life insurance.

The purpose of this project is to predict the rating in the existing Prudential Life Insurance assessment. If we can find out the significant attributes that determine the assessment, Prudential could streamline the process to make it quicker and less labor intensive. The task is to predict "Response", an ordinal variable with 8 levels relating to the final decision associated with an application. Since, "Response" is an ordinal variable; I have chosen to apply **Classification Tree and Random Forest** method for the analysis. I decided to add a binomial output variable called 'Approved' that is set to 1 when Response = 8. Otherwise Approved = 0. I could predict 'Approved' using Logistic Regression and figure out which independent variables are significant.

Life Insurance details insurers want to know

The underwriters don't just want to know about the applicant state of health. They want to know about his or her family's medical history as well. If a close member in the family has had any serious conditions (coronary heart disease, stroke, cancer and diabetes), the insurer could bump up your premiums. The Prudential Life Insurance applicant information is provided by the Kaggle competition.[2] It consists of 2 datasets: train (59,381 obs.) and test (19,765 obs.).

[1] First America Life Insurance: http://www.globelifeinsurance.com/article/the-importance-of-life-insurance#sthash.SRuYRhQx.dpuf

[2] https://www.kaggle.com/c/prudential-life-insurance-assessment/data

2. Prudential Data Set

train.csv - the training set, contains the Response values.
test.csv - the test set, you must predict the Response variable for all rows in this file.

Variable	Description
Id	A unique identifier associated with an application.
Product_Info_1-7	A set of normalized variables relating to the product applied for
Ins_Age	Normalized age of applicant
Ht	Normalized height of applicant
Wt	Normalized weight of applicant
BMI	Normalized BMI of applicant
Employment_Info_1-6	A set of normalized variables relating to the employment history of the applicant.
InsuredInfo_1-6	A set of normalized variables providing information about the applicant.
Insurance_History_1-9	A set of normalized variables relating to the insurance history of the applicant.
Family_Hist_1-5	A set of normalized variables relating to the family history of the applicant.
Medical_History_1-41	A set of normalized variables relating to the medical history of the applicant.
Medical_Keyword_1-48	A set of dummy variables relating to the presence of/absence of a medical keyword being associated with the application.
Response	This is the target variable, an ordinal variable relating to the final decision associated with an application

The following variables are all categorical (nominal): -

Product_Info_1, Product_Info_2, Product_Info_3, Product_Info_5, Product_Info_6, Product_Info_7, Employment_Info_2, Employment_Info_3, Employment_Info_5, InsuredInfo_1, InsuredInfo_2, InsuredInfo_3, InsuredInfo_4, InsuredInfo_5, InsuredInfo_6, InsuredInfo_7, Insurance_History_1, Insurance_History_2, Insurance_History_3, Insurance_History_4, Insurance_History_7, Insurance_History_8, Insurance_History_9, Family_Hist_1, Medical_History_2, Medical_History_3, Medical_History_4, Medical_History_5, Medical_History_6, Medical_History_7, Medical_History_8, Medical_History_9, Medical_History_11, Medical_History_12, Medical_History_13, Medical_History_14, Medical_History_16, Medical_History_17, Medical_History_18, Medical_History_19, Medical_History_20, Medical_History_21, Medical_History_22, Medical_History_23, Medical_History_25, Medical_History_26, Medical_History_27, Medical_History_28, Medical_History_29, Medical_History_30, Medical_History_31, Medical_History_33, Medical_History_34, Medical_History_35, Medical_History_36, Medical_History_37, Medical_History_38, Medical_History_39, Medical_History_40, Medical_History_41

The following variables are continuous:

Product_Info_4, Ins_Age, Ht, Wt, BMI, Employment_Info_1, Employment_Info_4, Employment_Info_6, Insurance_History_5, Family_Hist_2, Family_Hist_3, Family_Hist_4, Family_Hist_5

The following variables are discrete:

Medical_History_1, Medical_History_10, Medical_History_15, Medical_History_24, Medical_History_32

Medical_Keyword_1-48 are dummy variables.

3. Initial Data Wrangling

Using function 'missmap' to plot missing values, I got the following columns with missing values in a percentage greater than 30%. For a detail of the function invoked and the percentage calculations, you can visit https://moscosof.github.io/Prudential_DataWrangling/Prudential_DataWrangling.html

Train data set missing data	**Test data set missing data**
Insurance_History_5 (42.76%)	Insurance_History_5 (41.00%)
Family_Hist_2(48.25%)	Family_Hist_2(49.98%)
Family_Hist_3(57.66%)	Family_Hist_3(55.9%)
Family_Hist_4(32.30%)	Family_Hist_4(33.89%)
Family_Hist_5(70.41)	Family_Hist_5(68.92%)
Medical_History_10(99.06%)	Medical_History_10(98.98%)
Medical_History_15(75.10%)	Medical_History_15(75.20%)
Medical_History_24(93.59%)	Medical_History_24(94.02%)
Medical_History_32(98.13%)	Medical_History_32(98.22%)

As you can see, the percentage of missing data in the train and test data set is almost similar. The variable names are not completely descriptive to make our own assumptions regarding the importance of the variable to discard them or not from the analysis. Therefore, I decided to populate the missing values using the median. [3] You can see the R code to populate using median value:

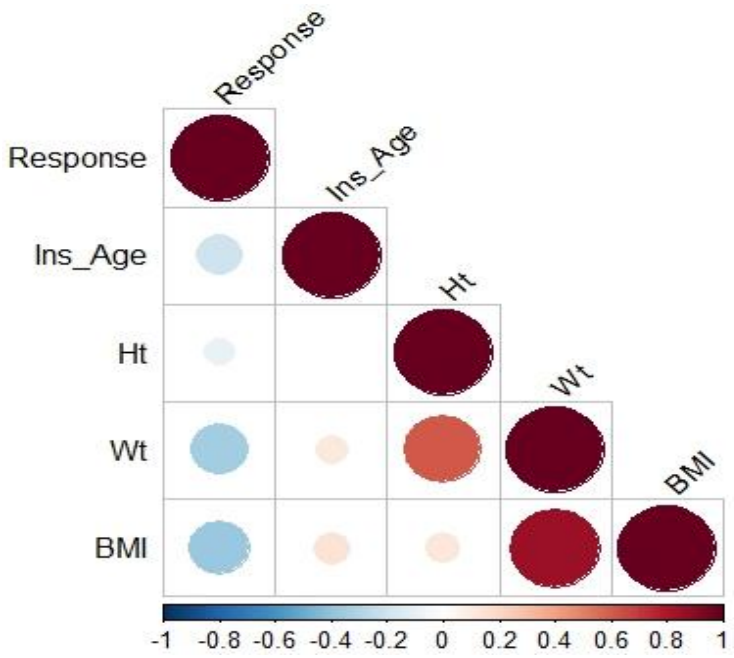
```
Populate_na <- function(datafra)
{
        for(i in 1:ncol(datafra))
        {
                if(is.numeric(datafra[,i]))
                {
                        datafra[is.na(datafra[,i]),i] <- median(datafra[!is.na(datafra[,i]),i])
                }
        }
        datafra
}
```

3 http://stackoverflow.com/questions/23242389/median-imputation-using-sapply

The Response variable is described as an "ordinal measure of risk that has 8 levels" but it does not specify if the top ratings indicates a high risk OR a positive score associated to a final decision. For that reason, I decided to figured out how Response is correlated to variables that are assumed to be negative in life insurance scores as they increase in value: weight, BMI, Age.

Correlation between Response and independent variables Weight, BMI and Age

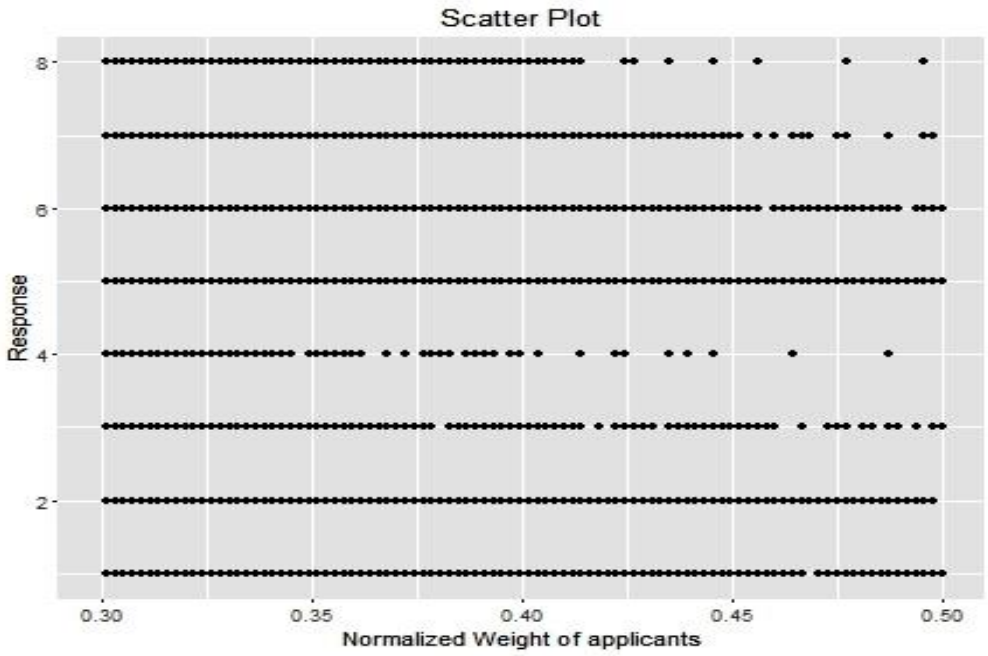
```
> rquery.cormat(M)$r
          Response Ins_Age    Ht    Wt BMI
Response         1
Ins_Age      -0.21       1
Ht           -0.094   0.0084     1
Wt           -0.35    0.11 0.61     1
BMI          -0.38    0.14 0.12 0.85   1
```



As you can see, there is a negative correlation between Response and the independent variables BMI, Weight and Age. In other words, as BMI, Weight and Age increases; Response decreases. It tells us that a high score in Response indicates a lower BMI, Weight and Age. The higher the Response value, the lower the insurance premium.

Just to confirm what I just stated, the following plot shows that as Weight increases, the top ratings in Response decrease.

```
qplot(x=Wt, y=Response, data=PersonalData,  # Cannot use binwidth=0.01,
    main="Scatter Plot",
    xlab ="Normalized Weight of applicants", xlim=c(0.30,0.50))
```



4. Picking a predictive method

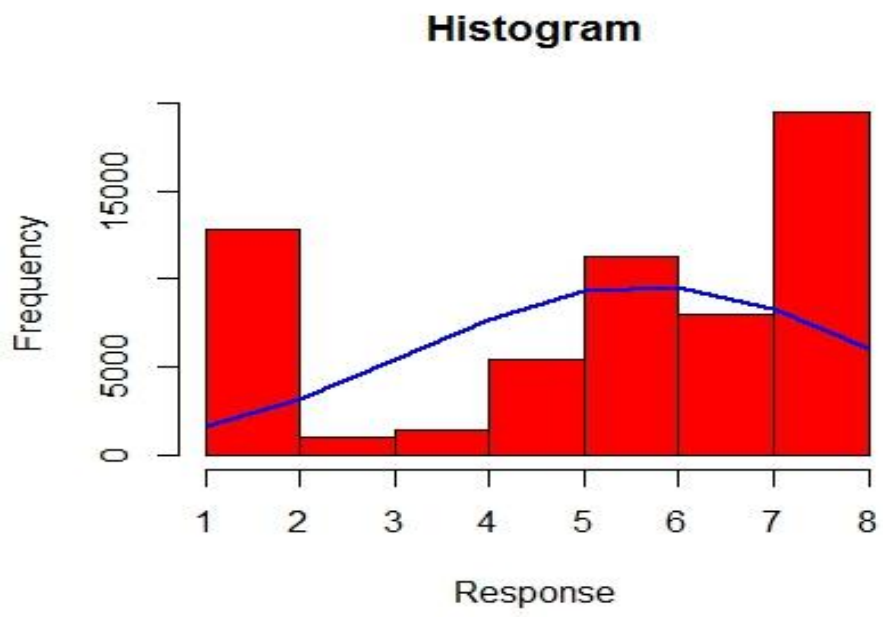
Since "Response" is an ordinal variable; I have chosen to apply **Classification Tree** and **Random Forest** for the analysis. **Logistic Regression** does not apply to this case, however, I decided to add a binary variable to evaluate Response as binary and at least have an idea of the variables that are significant in getting a high score of 8.

"Response" variable distribution:

```
summary(train$Response)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   4.000   6.000   5.637   8.000   8.000
table(train$Response)

    1     2     3     4     5     6     7     8
 6207  6552  1013  1428  5432 11233  8027 19489
```

Drawing Response Distribution

```
TrainResp <- train$Response
h<-hist(TrainResp, breaks=8, col="red", xlab="Response",
     main="Histogram")
xfit<-seq(min(TrainResp),max(TrainResp),length=8)
yfit<-dnorm(xfit,mean=mean(TrainResp),sd=sd(TrainResp))
yfit <- yfit*diff(h$mids[1:2])*length(TrainResp) lines(xfit, yfit, col="blue", lwd=2)
```



Classification Tree https://moscosof.github.io/Prudential_CapstoneProject/ClassTree.html

Our test data set does not contain a "Response" variable to evaluate our final model and for that reason, the train data set was split into a 75% allocated for training and 25% for testing.

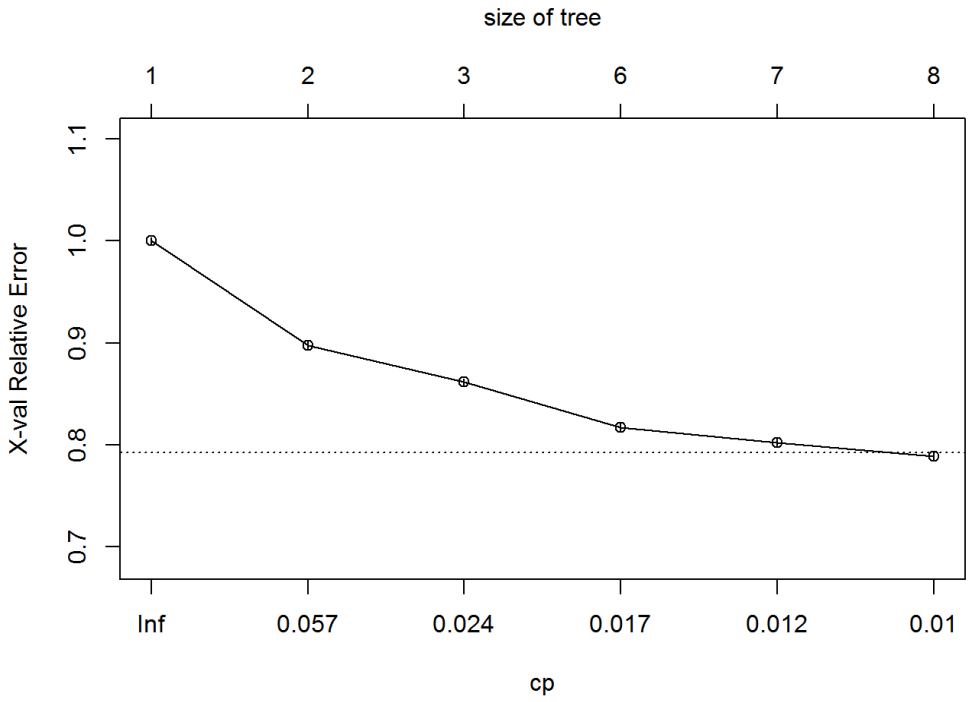
Classification Tree via "rpart"

```
# grow tree
fit <- rpart(Response ~ ., method="class", data=train)
printcp(fit) # display the results
```

```
Classification tree:
rpart(formula = Response ~ ., data = train, method = "class")

Variables actually used in tree construction:

[1] BMI              Medical_History_15 Medical_History_23
[4] Medical_History_4  Product_Info_4

Root node error: 29919/44536 = 0.67179
n= 44536

        CP nsplit rel error  xerror      xstd
1 0.103011      0   1.00000 1.00000 0.0033121
2 0.031585      1   0.89699 0.89742 0.0034513
3 0.018060      2   0.86540 0.86156 0.0034827
4 0.015174      5   0.81122 0.81727 0.0035098
5 0.010027      6   0.79605 0.80217 0.0035161
6 0.010000      7   0.78602 0.78870 0.0035205
```

```
# visualize cross-validation results
plotcp(fit)
```

plotcp shows the cp values indicating that the lowest cp value is reached at 0.01.



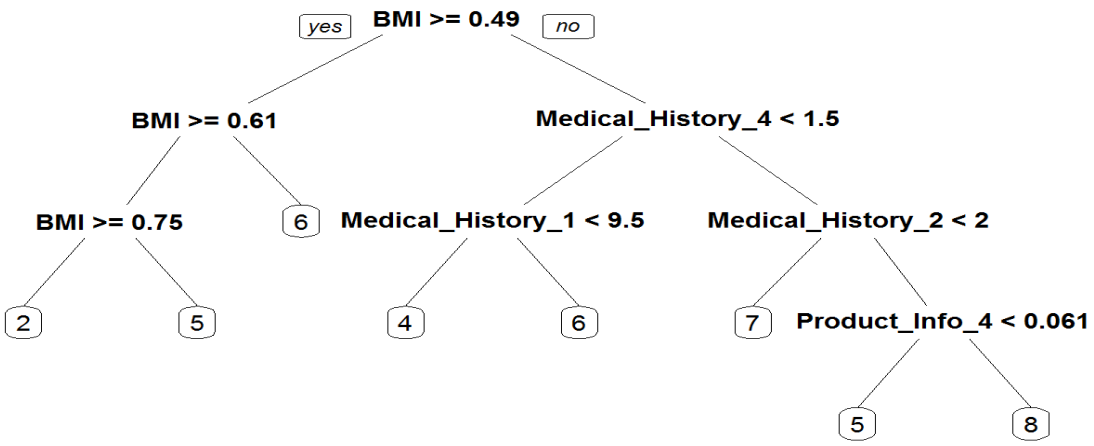
Prune the tree Prune back the tree to avoid overfitting the data. Typically, you will want to select a tree size that minimizes the cross-validated error, the xerror column printed by printcp().

```
pfit<- prune(fit, cp=  fit$cptable[which.min(fit$cptable[,"xerror"]),"CP"])

printcp(pfit) # display the results
```

```
Classification tree:
rpart(formula = Response ~ ., data = train, method = "class")

Variables actually used in tree construction:
[1] BMI                Medical_History_15 Medical_History_23
[4] Medical_History_4  Product_Info_4

Root node error: 29919/44536 = 0.67179

n= 44536

        CP nsplit rel error  xerror      xstd
1 0.103011      0   1.00000 1.00000 0.0033121
2 0.031585      1   0.89699 0.89742 0.0034513
3 0.018060      2   0.86540 0.86156 0.0034827
4 0.015174      5   0.81122 0.81727 0.0035098
5 0.010027      6   0.79605 0.80217 0.0035161
6 0.010000      7   0.78602 0.78870 0.0035205
```

Plot the pruned tree

```
prp(pfit)
```



Predicting on testLog data set.

```
#dimension of testLog data set
dim(testLog)
[1] 14845   128
pfit.predict = predict(pfit, newdata = testLog, type = "class")

table(pfit.predict)
pfit.predict
  1   2   3   4    5    6    7    8
  0 432   0 513 1509 5898  968 5525
```

Calculate the Sum of Square Errors to evaluate the model.

```
#Prediction vs Actual
pfit.sse = sum((as.numeric(pfit.predict) - testLog$Response)^2)
pfit.sse
[1] 82345
```

Calculate accuracy to evaluate the model.

```
table(pfit.predict)
   1    2    3    4    5    6    7    8
```

```
   0  432    0  513 1509 5898  968 5525
table(testLog$Response)

   1    2    3    4    5    6    7    8
1552 1638  253  357 1358 2808 2007 4872
table(testLog$Response, as.numeric(pfit.predict))

      2    4    5    6    7    8
  1 106   89  244  668  135  310
  2 212   56  273  664  133  300
  3   5   99   56   88    2    3
  4   0  234    1   92    1   29
  5  92    1  613  476   74  102
  6  14   16  274 1885  149  470
  7   1    3   41 1054  368  540
  8   2   15    7  971  106 3771
```

Overall accuracy = (212 + 234 + 613 + 1885 + 368 + 3771) / 14,845 = 47.77%

Conclusion: The tree obtained using Classification method is easy to read and hence very interpretative. However, the number of observations in our training data set might be too small for an accurate prediction. Predictions for Response outputs 8, 6 and 4 are very accurate. Nonetheless, the overall accuracy is 47.44% .

Random Forest https://moscosof.github.io/Prudential_CapstoneProject/RandomForest.html

Our test data set does not contain a "Response" variable to evaluate our final model and for that reason, the train data set was split into a 75% allocated for training and 25% for testing.

Populate missing values with the median

```
trainLog <- manage_na(trainLog[,-c(1)])    #  Except columns 1 (ID)
testLog  <- manage_na(testLog[,-c(1)])
```

Random Forest

```
#randomForest does not run with na values
fit <- randomForest(Response ~ .,  nodesize = 25, ntree = 200, data=trainLog)
print(fit) # view results
```

```
Call:
 randomForest(formula = Response ~ ., data = trainLog, nodesize = 25,      ntree = 200)
               Type of random forest: regression
```

```
                Number of trees: 200
No. of variables tried at each split: 42

      Mean of squared residuals: 3.498086
            % Var explained: 42.04
importance(fit) # importance of each predictor
                          IncNodePurity
```

bmi	27737.38079
Wt	12167.35153
Medical_History_23	10228.93635
Medical_Keyword_3	9406.69365
Product_Info_2	8957.23896
Product_Info_4	6610.62719
Ins_Age	6016.96877
Medical_History_4	5375.22888
Medical_Keyword_15	4000.91795
Family_Hist_3	2736.29612
Family_Hist_5	2577.72095
Medical_History_1	2513.09601
Medical_History_2	2301.66765
Family_Hist_4	2126.93436
……	…….

Making Predictions on testLog

```
dim(testLog)
[1] 14845   127
predictForest <- predict(fit, newdata = testLog )
```

Evaluating the model

```
predictForestRound <- round(predictForest,0)
table(testLog$Response)
  1   2   3   4   5   6   7   8
1552 1638  253  357 1358 2808 2007 4872

table(predictForestRound)
predictForestRound
  1   2   3   4   5   6   7   8
  4 435 1254 1770 2788 3180 4351 1063
```

```
table(testLog$Response, predictForestRound)
```

```
  predictForestRound
     1   2   3   4   5   6   7   8
1    4 161 301 353 317 246 153  17
2    0 208 317 335 374 251 144   9
3    0  25 173  29  16   6   4   0
4    0   6 208  77  11  19  32   4
5    0  24 146 497 491 115  82   3
6    0  11 104 355 915 946 447  30
7    0   0   3 102 463 811 588  40
8    0   0   2  22 201 786 901 960
```

Overall Accuracy = (4+208+173+77+ 491+ 946+588+960) / 14845 = 21.40 %

Logistic Regression https://moscosof.github.io/Prudential_CapstoneProject/Binomial_glm.html

Logistic Regression applies to binary outcomes only and for that reason a new column "Approved" was created. Approved is set to 1 when the applicant gets the max rating of 8 (Response). Otherwise, its value is zero.

```
train$Approved <- 0
train$Approved[train$Response==8] <- 1   # Approved = 1 when train$Response = 8

# Approved values
table(train$Approved)
```

```
  0   1
39892 19489
```

Splitting data with ratio 75/25.

```
set.seed(3000)
# 75% of data in Training set
split <- sample.split(train$Response, SplitRatio = 0.75)

#Splitting data
trainLog <- subset(train, split==TRUE)
testLog  <- subset(train, split==FALSE)
```

Populate trainLog missing values with the median

```
trainLog <- manage_na(trainLog[,-c(1)])    #   Except columns 1 (ID)

testLog <- manage_na(testLog[,-c(1)])
```

Alphabetic Categorical variable in the trainLog data set: Product_Info_2. Getting the frequency

```
ProductInfo2 <- data.frame(table(trainLog$Product_Info_2))
ProductInfo2_Sorted <- ProductInfo2 %>% arrange(desc(Freq))
head(ProductInfo2_Sorted)
```

```
 Var1  Freq
1   D3 10778
2   D4  8155
3   A8  5113
4   D1  4895
5   D2  4731
6   E1  1974
```

Approved ratio

```
table(trainLog$Approved)
```

```
    0     1
29919 14617
```

Percentage of Approved observations (Response = 8)

```
table(trainLog$Approved)[2]/nrow(trainLog)
```

```
    1
0.3282064
```

We need a baseline method to compare our predictions. In this case, we can say that 14,617 out of 44,536 obs. were Approved (Response=8) in our training data set. Therefore, our base line method has an accuracy of 32.8% and that is what we will try to beat with a logistic regression model. The AIC value was improved by removing variables that were not significant in the logistic model:

```
trainFit = glm(Approved ~ Product_Info_1 + Product_Info_2 +
                  Product_Info_4 + Product_Info_5 +
                  InsuredInfo_2  + InsuredInfo_4  +
                  InsuredInfo_5  + InsuredInfo_6  + InsuredInfo_7 +
                  Insurance_History_1 + Insurance_History_2 +
                  Family_Hist_1 + Family_Hist_3 +
                  Family_Hist_4 + Family_Hist_5 +
                  Medical_History_3 +
                  Medical_History_4  + Medical_History_5 +
                  Medical_History_7  + Medical_History_10 +
                  Medical_History_13 + Medical_History_14 +
                  Medical_History_15 +
                  Medical_History_17 + Medical_History_18 +
                  Medical_History_20 + Medical_History_23 +
                  Medical_History_24 + Medical_History_30 +
                  Medical_History_31 + Medical_History_32 +
```

```
                Medical_History_39 + Medical_History_40 +
                Medical_Keyword_3  + Medical_Keyword_12  +
                Medical_Keyword_15 + Medical_Keyword_22  +
                Ht + Wt +  Ins_Age,
                data = trainLog, family=binomial)

summary(trainFit)
```

```
Deviance Residuals:
   Min      1Q  Median      3Q     Max
-2.8971  -0.5655  -0.1359   0.6145   4.1564

Coefficients:
                      Estimate Std. Error z value      Pr(>|z|)
(Intercept)       -9.052e+00  1.605e+00  -5.639 1.71e-08 ***
Product_Info_1     -8.113e-01  8.530e-02  -9.511  < 2e-16 ***
Product_Info_2A2   -3.068e-01  1.041e-01  -2.946 0.003217 **
Product_Info_2A3   -1.919e-03  1.163e-01  -0.017 0.986832
Product_Info_2A4    3.024e-01  2.206e-01   1.371 0.170280
Product_Info_2A5    1.964e-01  1.249e-01   1.573 0.115692
Product_Info_2A6    2.513e-01  9.059e-02   2.775 0.005528 **
Product_Info_2A7   -3.247e+00  2.298e-01 -14.132  < 2e-16 ***
Product_Info_2A8   -3.623e-01  7.952e-02  -4.556 5.21e-06 ***
Product_Info_2B1   -1.732e-01  5.114e-01  -0.339 0.734887
Product_Info_2B2    2.575e-01  1.106e-01   2.329 0.019884 *
Product_Info_2C1   -4.309e-01  2.101e-01  -2.051 0.040274 *
Product_Info_2C2   -2.594e-01  2.809e-01  -0.923 0.355763
Product_Info_2C3   -1.228e-01  2.013e-01  -0.610 0.541870
Product_Info_2C4    3.541e-01  2.180e-01   1.624 0.104276
Product_Info_2D1   -3.329e-01  8.276e-02  -4.022 5.76e-05 ***
Product_Info_2D2   -1.574e-01  8.171e-02  -1.926 0.054076 .
Product_Info_2D3   -2.003e-01  7.168e-02  -2.795 0.005193 **
Product_Info_2D4    1.928e-01  7.059e-02   2.731 0.006315 **
Product_Info_2E1    1.161e-01  8.953e-02   1.297 0.194755
Product_Info_4      7.016e-01  5.200e-02  13.492  < 2e-16 ***
Product_Info_5     -7.038e-01  1.669e-01  -4.217 2.48e-05 ***
InsuredInfo_2      -3.550e+00  6.027e-01  -5.890 3.85e-09 ***
InsuredInfo_4       1.516e-01  4.351e-02   3.486 0.000491 ***
InsuredInfo_5      -3.732e-01  7.700e-02  -4.847 1.26e-06 ***
InsuredInfo_6       4.808e-01  4.068e-02  11.820  < 2e-16 ***
InsuredInfo_7      -3.823e-01  6.959e-02  -5.494 3.93e-08 ***
Insurance_History_1 -1.690e-01  3.258e-02  -5.188 2.12e-07 ***
Insurance_History_2 -3.861e-01  5.453e-02  -7.082 1.42e-12 ***
Family_Hist_1       1.951e-01  3.096e-02   6.302 2.94e-10 ***
Family_Hist_3       1.374e+00  1.653e-01   8.313  < 2e-16 ***
Family_Hist_4       7.003e-01  1.306e-01   5.362 8.21e-08 ***
Family_Hist_5       1.517e+00  2.175e-01   6.974 3.09e-12 ***
```

```
Medical_History_3    1.997e-01  5.915e-02   3.377 0.000734 ***
Medical_History_4    1.532e+00  3.613e-02  42.414  < 2e-16 ***
Medical_History_5   -1.951e+00  2.141e-01  -9.110  < 2e-16 ***
Medical_History_7    4.973e-01  1.155e-01   4.304 1.67e-05 ***
Medical_History_10   6.135e-03  1.694e-03   3.622 0.000292 ***
Medical_History_13   1.992e-01  2.387e-02   8.346  < 2e-16 ***
Medical_History_14   3.571e-01  9.919e-02   3.601 0.000318 ***
Medical_History_15   1.485e-02  3.893e-04  38.157  < 2e-16 ***
Medical_History_17   6.545e-01  1.161e-01   5.637 1.73e-08 ***
Medical_History_18  -3.234e-01  6.744e-02  -4.796 1.62e-06 ***
Medical_History_20   1.145e+00  1.353e-01   8.462  < 2e-16 ***
Medical_History_23   4.359e-01  3.842e-02  11.344  < 2e-16 ***
Medical_History_24   4.424e-03  6.415e-04   6.896 5.34e-12 ***
Medical_History_30  -1.525e+00  9.273e-02 -16.450  < 2e-16 ***
Medical_History_31   5.461e-01  9.271e-02   5.891 3.85e-09 ***
Medical_History_32   8.450e-03  2.343e-03   3.606 0.000310 ***
Medical_History_39   1.453e-01  3.944e-02   3.686 0.000228 ***
Medical_History_40   7.864e-01  1.118e-01   7.037 1.97e-12 ***
Medical_Keyword_3   -3.590e+00  2.522e-01 -14.232  < 2e-16 ***
Medical_Keyword_12   5.776e-01  1.561e-01   3.699 0.000216 ***
Medical_Keyword_15  -2.145e+00  1.054e-01 -20.358  < 2e-16 ***
Medical_Keyword_22   3.044e-01  8.410e-02   3.619 0.000295 ***
Ht                1.191e+01  3.230e-01  36.885  < 2e-16 ***
Wt               -2.037e+01  3.022e-01 -67.391  < 2e-16 ***
Ins_Age           -1.059e+00  1.160e-01  -9.125  < 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: 56374  on 44535  degrees of freedom
Residual deviance: 33324  on 44478  degrees of freedom
AIC: 33440

Number of Fisher Scoring iterations: 7
```

As you can see, the model created dummy variables for the categorical variable Product_Info_2.

Predictions with different Thresholds:

To view the detail in the thresholds calculations, visit

Threshold = 0.7

```
  FALSE  TRUE
0 28316  1603
1  7751  6866
```

Sensivity = 0.47 False_Positive_Error_Rate = 0.05 Specificity = 0.95

Threshold = 0.5

```
    FALSE  TRUE
0   26056  3863
1    3972 10645
```

Sensitivity = 0.72 False_Positive_Error_Rate = 0.13 Specificity = 0.87

Threshold = 0.2

```
   FALSE   TRUE
0 20504    9415
1  940    13677
```

Sensitivity = 0.93 False_Positive_Error_Rate = 0.31 Specificity = 0.69

Which Threshold to pick?

After calculating the Sensitivity and Specificity with Threshold 0.7, 0.5, and 0.2, I would pick a Threshold of 0.2 because I would like to have a high True Positive Rate while having a low False Positive Error Rate.

Making predictions on testLog data set

```
dim(testLog)
[1] 14845   128
predictTest = predict(trainFit, type="response", newdata = testLog)

summary(predictTest)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.00000 0.01491 0.14960 0.26740 0.48920 0.97410
```

Calculate accuracy of the model

```
ConfusionMatrix <- table(testLog$Approved,predictTest > 0.2)
ConfusionMatrix

    FALSE TRUE
  0  7563 2410
  1   554 4318
Accuracy <- (ConfusionMatrix[1,1] + ConfusionMatrix[2,2]) / nrow(testLog)
Accuracy
[1] 0.8003368
False_Positive_Error_Rate = ConfusionMatrix[1,2] / (ConfusionMatrix[1,1] +
ConfusionMatrix[1,2])
False_Positive_Error_Rate
[1] 0.2416525
#Sensitivity
True_Positive_Rate = ConfusionMatrix[2,2] / (ConfusionMatrix[2,1] +
ConfusionMatrix[2,2])
True_Positive_Rate
[1] 0.886289
```

Conclusion:
The logistic model can accurately predict a **binary** output with the variable Approved to indicate the highest score (Approved = 1 when Response = 8). A Threshold of **0.2** gives a high True Positive Rate of 88.6 % while keeping a low False Positive Error Rate of 24.1%.

Comparing models:
All models were created under 44,536 obs. in the training log and 14,845 obs. in the test one.
Among all 3 models, only Classification Tree did not required to populate missing values.
Classification Tree gave us overall accuracy of 47.77% with a tree that was easy to read in 7 splits for the variables: BMI, Medical_History_4, Medical_History_1, Medical_History_2 and Product_Info_4.
Random Forest gave us 200 tress with 42 splits, mean squared residuals of 3.49 and overall accuracy of 21.40 %. The top 8 variables with more importance for the model were: BMI, Wt, Medical_History_23, Medical_Keyword_3, Product_Info_2, Product_Info_4, Ins_age, Medical_History_4.
The model with the highest accuracy was obtain using Logistic Regression after creating a binary outcome variable (Approved) to be predicted instead of a nominal (Response), giving us an overall accuracy of 80%. There were around 40 significant coefficients that probably could have been reduced to improve the AIC.
The variables with significant importance among all 3 models were: BMI, Medical_History_4, Medical_History_1, Medical_History_2 and Product_Info_4, Wt, Medical_History_23, Medical_Keyword_3, Product_Info_2, Ins_age.

INDEX

R code sources are located at https://github.com/moscosof/Prudential_CaptsoneProject

Reference for details in R code output in html format:

Classification Tree: https://moscosof.github.io/Prudential_CapstoneProject/ClassTree.html
Random Forest: https://moscosof.github.io/Prudential_CapstoneProject/RandomForest.html
Logistic Regression: https://moscosof.github.io/Prudential_CapstoneProject/Binomial_glm.html