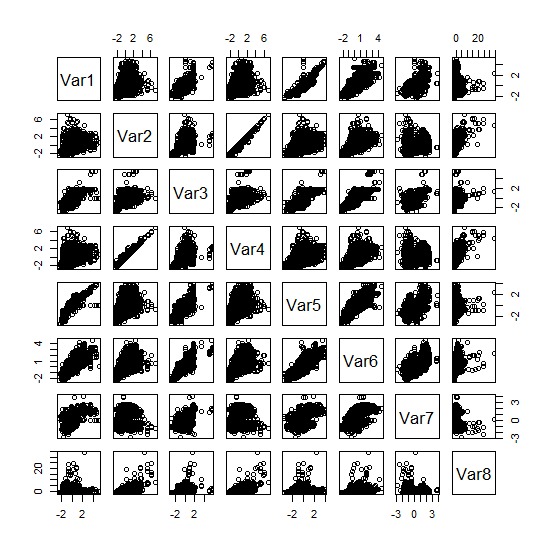
**Pre-processing of data:**

Since we are dealing with a data consisting 100,000 observations of 34 variables to start with, and the targeted predictions (number of claims) are only 720, the ratio of positive outcomes becomes extremely low (~0.007%). This demands for an extensive pre-processing of data.

**Summary of pre-processing:**  
1. The data contains over 150,000 “?”s (missing values). Since R treats these “?”s as characters and accounts them as factors, we replaced them with logical NA values. This paved way for na.action options.  
2. We plotted the pairs function to see any close correlation amongst variables. Variables Var1, and Var2 showed high collinearity with Var5 and Var4 respectively. Similarly, variables Blind\_Make, Blind\_Model, Blind\_submodel were also found to be extremely colinear. Also, Blind\_Model and Blind\_Submodel had too many factors which would result in expensive calculations. So, we decided to drop Var1, Var2, Blind\_Model and Blind\_Submodel.

3. Variables Cat2, Cat4, Cat5, Cat7 contain more than 40,000 missing values each. Hence, we decided to remove them from the model.

4. C\_Claim – a binary response variable was created according to the Claim\_Amount variable.



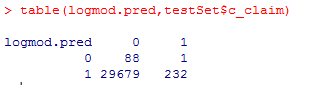
**1. Logistic Model**

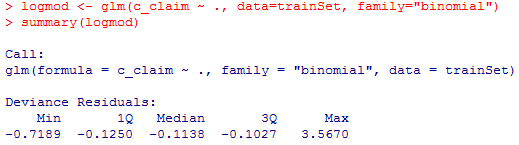
We began our initial analysis with the logistic model using the **glm** function on our training data set. For this model, we split our data into 70% training and 30% testing data.

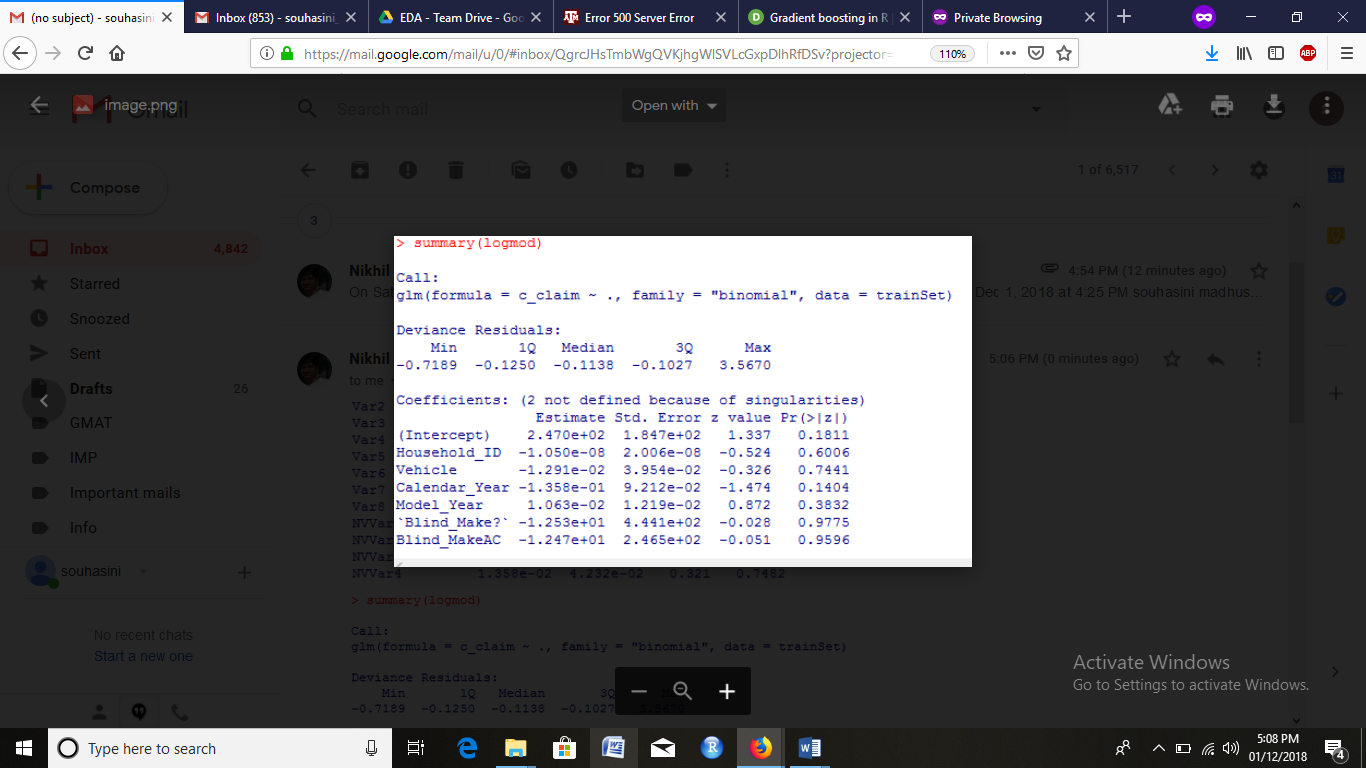
We then ran a summary of the probabilities of the testing model to find the median value to determine the threshold.

According to the confusion matrix, our test error rate is 79%.

Confusion matrix:







**2. Generalized boosted model (gbm model):**

Packages used in this model:

* Caret - for modeling wrapper, functions and commands
* pROC - for Area under the Curve (AUC) , ROC functions

We used the caret function to model our binary outcomes for C\_Claim.

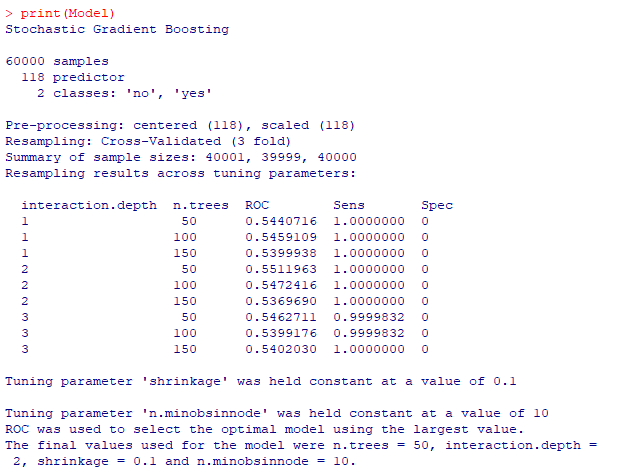
This model utilizes boosted trees.

Gbm can deal with factor variables since it can dummify them internally. We ensured each unique factor has its own separate column.

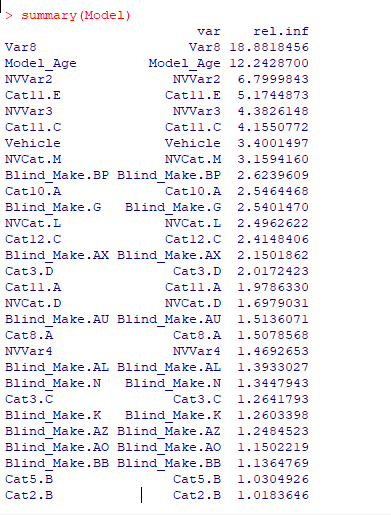
We used the trainControl function provided by the Caret library to control resampling our data. Also, this function internally determined the best fit setting for our model through internal runs for training and testing.

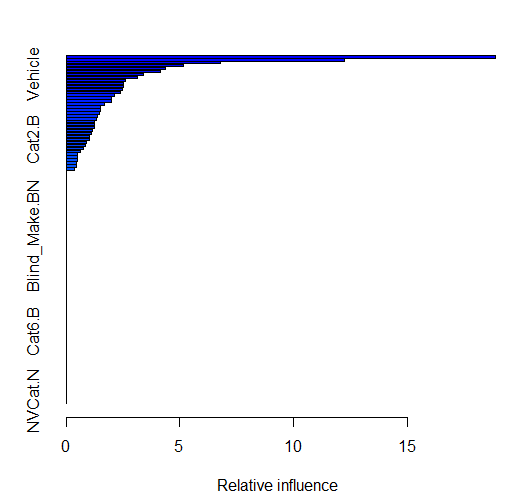
In our model, we cross validated our data thrice. The function automatically set the best parameters for tuning such as shrinkage, trees and interaction depth (for gbm model).

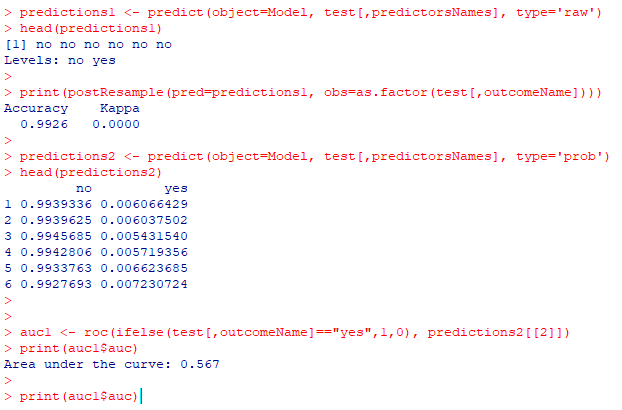
We checked which tuning parameters served to be most important.



We used the summary function to determine variable importance.







For evaluating our gbm model, we used "raw" to give a class prediction of "yes" or "no".

For our classification model, we are not opting to use **RMSE**, instead we want to use **ROC**.

We used "prob" to check AUC value through probabilities.

postResample function of the caret package was used to check the accuracy score.

**Evaluation Criteria:**

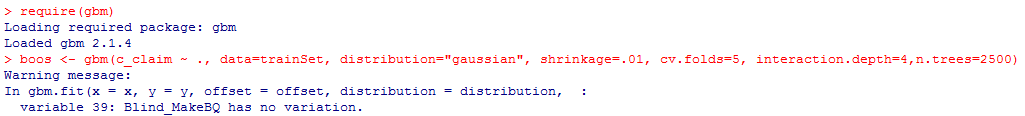
**The AUC value for this model is 0.567.**

**3. Boosted model (Best model):**

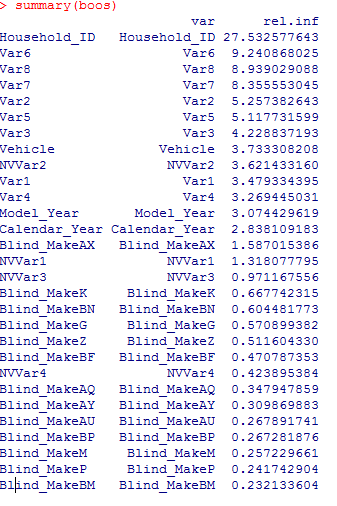
We used boosting model keeping certain goals in mind. Boosting helped us in variable selection. Since we are dealing with 65 factors in Blind\_Make variables, we decided to convert them into dummy variables and then converted them into numeric variables.

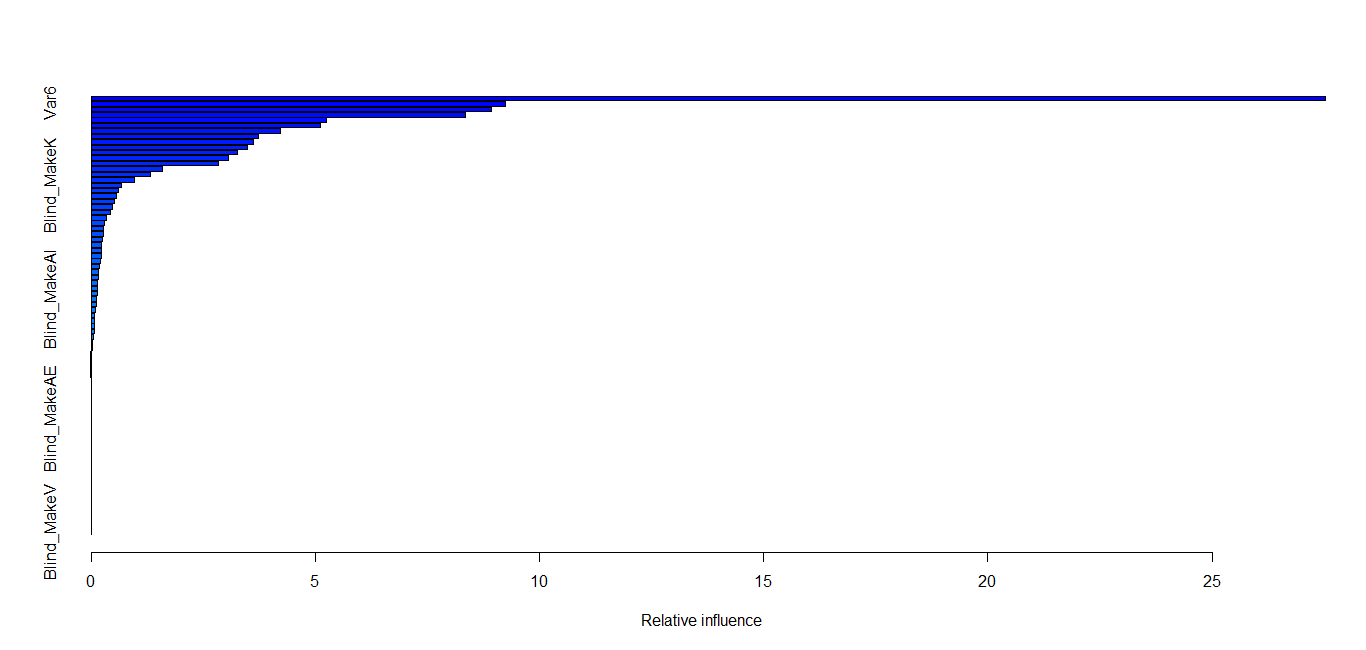
We decided to use shrinkage (step size) of 0.01 and maximum number of trees as 2500 (considering our computational limitations). For error rate fitting, we used cross validation with cv folds set to 5.

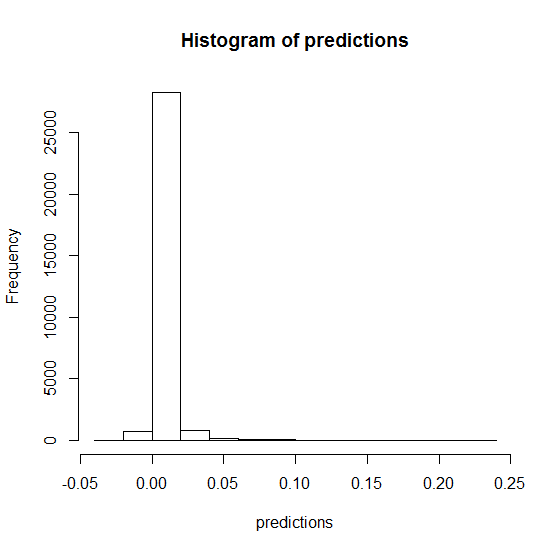
After converting Blind\_Make factors into 65 numeric variables, we were dealing with 82 variables. To test boosting, we first decided to drop Cat.. variables and then computed boosting model.



**List of important variables:**





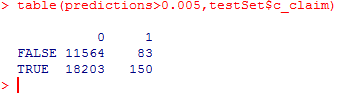


**Reasons for choosing best model:**

**Model 1 (Logistic regression)**: Resulted in a higher False Positive Rate. Thus, we pursued boosting methods to increase our prediction accuracy.

**Model 2:** Resulted in AUC value of 0.567. The range of AUC is 0.5 to 1, since our AUC was on the lower end of the range, thus, we chose to refine our boosting method.

**Model 3:** We eliminated less significant character variables to refine our model and it resulted in a lower test error rate compared to our two previous models. Thus, Model 3 is our best model.



Test error rate: (11564+83)/30,000 = ~38%