MSBA7027A Group 3 Group Project Report - Credit Card Customer Churn Prediction

LU, Yixuan 3035877749 | LIU, Shuer 3035883695 | ZHU, Huinan 3035880021 | SHEN, Shulin 3035883293

1. Executive summary

• Problem

Assume that we are a group of data analytics consultants hired by HSAC bank. Currently the credit card department of HSAC bank is facing a very high customer churn rate, which means that an increasing number of credit card holders have decided to close their credit card accounts. The department head is very confused with the current situation and concerned with the potential impact on the bank's performance. The department head would like to know the method to predict customer churn and understand the rationale behind so that the department can improve the quality of products and services provided to change the situation.

In this report, we have adopted machine learning techniques to (1) find the optimal model for the bank to predict credit card customer churn and (2) uncover the most important variables leading to the customer churn. We also proposed recommendations based on our research and findings.

• Principal findings

We fit the dataset to several machine learning models subsequent to the EDA and data-preprocessing. Our machine learning includes (1) KNN, (2) SVM, (3) Logistic Regression, (4) Random Forest, (5) Basic GBM, (6) Stochastic GBM, (7) XGBoost and (8) K-means. We generated the test AUC for each model. In conclusion, the optimal model is Stochastic GBM with the highest test AUC of 0.9578. Based on the Stochastic GBM model, we find the top 2 most important variables, *Total Transaction Count* and *Total Transaction Amount*, for HSAC to investigate further and pay more attention to.

• Recommendations

Last but not least, we proposed recommendations for HSAC to both predict and prevent customer churn. In terms of prediction, we suggest HSAC to proactively and closely monitor customer activities and pay special attention to the 2 most important variables on a timely basis.

As for preventing attrition, we recommend HSAC to focus on Customer Relationship Management. For example, HSAC should conduct customer surveys regarding the service quality regularly. To attract and retain customers, HSAC can design and launch promotion programs to different clusters of customers. More importantly, HSAC should consider providing customized service to different clusters of customers according to the clusters grouped by K-means/clusters grouped by transaction amount.

2. Introduction, Problem Description & Relevance

Banks generate income by lending money and charging interest. Credit Cards and Overdrafts are main sources of income for banks. Although credit card business generates revenue, its fluctuation can also bring a huge impact and uncertainty to the operation and financial performance of banks. Undoubtedly, customers are the overarching factor of the profitability of this business, which bluntly reveals the crucialness of prediction of credit card customer churn.

Currently the Credit Card Department of HSAC bank is facing the challenge of high customer churn rate. Therefore, in this report, our goal is to use machine learning techniques to support HSAC bank to predict and understand the customer churn. We also propose recommendations for HSAC bank to develop strategies to cope with the challenge.

3. Data Description & Preliminary Data Analysis

The dataset consists of 10,127 customers and 20 variables concerning customers' demographic attributes and bank activity attributes. Attrition Flag is the target variable and indicates whether the customer's credit card account is closed. It includes Attrited Customers and Existing Customers.

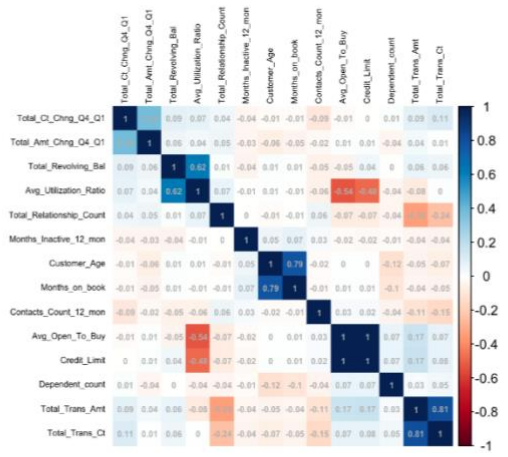
Categorical Variables
1.Gender: M=Male, F=Female
2.Education_Level: Educational Qualification of the customer.
3.Marital_Status: Married, Single, Divorced, Unknown.
4.Card_Category: Blue, Silver, Gold, Platinum.
5.Income_Category: Annual Income Category.

Numerical Variables
6.Months_on_book: Period of relationship with bank.
7.Total_Relationship_Count: Total no. of products held by the customer.
8.Months_Inactive_12_mon: No. of months in which the customer has not carried out any transactions using the card of the company
9.Contacts_Count_12_mon: No. of Contacts to bank in the last 12 months
10.Credit_Limit: Credit Limit on the Credit Card.

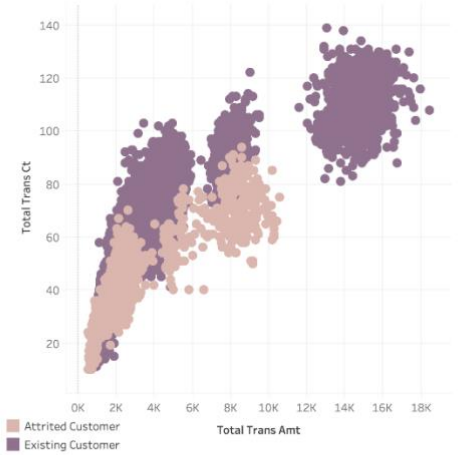
Numerical Variables(Continued)
11.Total_Revolving_Bal: The unpaid portion that carries over to the next month when a customer does not pay.
12.Dependent_count: No. of family members dependent on the customer.
13.Customer_Age: Customer's Age in Years.
14.Avg_Open_To_Buy: Open to Buy Credit Line (Average of last 12 months)
15.Total_Amt_Chng_Q4_Q1: Change in Transaction Amount (Q4 over Q1)
16.Total_Ct_Chng_Q4_Q1: Change in Transaction Count (Q4 over Q1)
17.Total_Trans_Amt: Total Transaction Amount (Last 12 months)
18.Total_Trans_Ct: Total Transaction Count (Last 12 months)
19.Average Utilization Ratio: Measures how much credit you are using compared to what you have available.

After comparison, categorical variables show similar distribution between attrited and existing customers, which means these variables have low correlation with Attrition Flag. While for several numeric variables, there are significant differences in terms of the average between 2 kinds of customers, which indicates a strong relationship with customer attrition. After comparison, there are 6 features that seem to have a strong impact on churn, including *Total Relationship Count, Total_Ct_Chng_Q4_Q1, Total_Trans_Ct, Total_Trans_Amt, Total_Revolving_Bal and Avg_Utilization_Ratio.*

From the correlation map of numerical data, there are some highly correlated variables (*Total Transaction Count & Total Transaction Amount, Customer age & months of book, Total Revolving Balance & Average Utilization*) and 2 highly collinear variables (*Credit Limit & Avg_Open_To_Buy*). Therefore, *Avg_Open_To_Buy* will be omitted in data pre-processing. By measuring the interactions between highly correlated numeric variables, the scatter plot below shows a significant difference between 2 kinds of customers, which suggests that models can have a good separation between our customers.



Picture 1. Correlation map of numerical data



Picture 2. *Total_Trans_Ct* VS *Total_Trans_Amt*

After EDA, we do data preprocessing. Firstly, we clean the data by checking missing and outlier data. Secondly, we split the data into a training set (70%) and testing set (30%). Then in the feature engineering process, we standardize numerical variables and encode categorical variables which are shown below.

Encoding Level

1. Education_Level: 1-Uneducated, 2-High School, 3-College, 4-Graduate, 5-Post-Graduate, 6-Doctorate, 7-Unknown;
2. Income_Category: 1-Less than $40K, 2-$40K - $60K, 3-$60K - $80K, 4-$80K - $120K, 5-$120K +, 6-Unknown;
3. Card_Category: 1-Blue, 2-Silver, 3-Gold, 4-Platinum;
4. Gender: 0-M, 1-F;
5. Attrition_Flag: 0-Attrited Customer, 1-Existing Customer.

During the process of EDA, we spot that, in the original dataset, the attrited customers account for 16% while the existing customers account for 84% of the total sample, which means the dataset is imbalanced. A dataset is imbalanced if the classification categories are not approximately equally represented (CHAWLA, 2002). We are working on a classification problem. Although it is quite common for a real-world dataset to be imbalanced, the machine learning models may not perform efficiently when the input is an imbalanced dataset.

We therefore introduce the SMOTE algorithm to deal with the imbalance issue. SMOTE stands for synthetic minority oversampling technique. SMOTE oversample the minority class by taking each minority class sample and introducing synthetic examples along the line segments joining any/all of the k minority class nearest neighbors. (CHAWLA, 2020). It is done without replacement. In our case, we use SMOTE to oversample the attrited customers which is our minority class.

We only perform SMOTE on the training set to prevent data leakage. We use the SMOTE function in the performanceEstimation package. The hyper-parameter *perc.over* controls the number of over-sampling and perc.under controls the number of under-sampling. Originally, we had 1,149 attrited customers and 5,939 existing customers. In the end, we have 1,149**perc.over* of attrited customer and 5,939 + [1149*(perc.over-1)]/200 existing customers in total. Hyper-parameter k can tune the KNN algorithm embedded in SMOTE and it indicates how the new examples are created. Here we use default k=5 which is the same as the default k in KNN algorithm. The percentage of attrited and existing customers is now roughly 50% to 50%, which means the classification categories are now approximately equally distributed. We create two smoted training sets. One is for SVM, and the other is for the rest of the models. We fit SVM with smoted training set having fewer samples. It is because our computers simply crashed when tuning the hyper-parameter for SVM with a training set of around 10,000 samples.

4. ML Methods, Analysis & Comparison

4.1 Supervised Model Fitting

As we are working on a binary classification problem, we use AUC instead of RMSE to select our optimal model. We have fit logistic regression, KNN, SVM, random forest, basic GBM, stochastic GBM and XGBoost. And we have tuned hyper-parameters for all models except logistic regression in order to get the best performance. Since the training AUC of these models are very similar, we choose to use test AUC as the measurement and the result is shown in Table 1. Among all the models we have fit, Stochastic GBM has the highest test AUC.

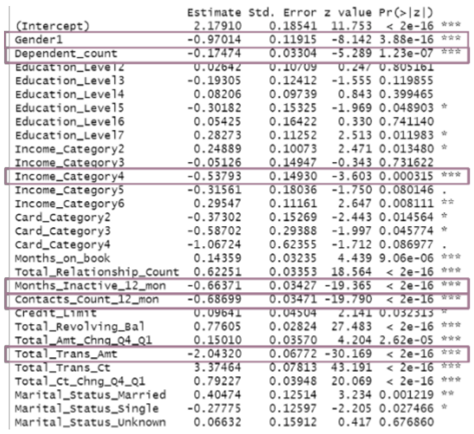
Model	KNN	SVM - Linear	SVM - Radial	SVM - Polynomial	Logistic Regression	Random Forest	Basic GBM	Stochastic GBM	XGBoost
Test AUC	0.859	0.8438	0.8905	0.8824	0.8417	0.9304	0.9574	0.9578	0.9289

Table 1. Test AUC of supervised models

- **KNN**

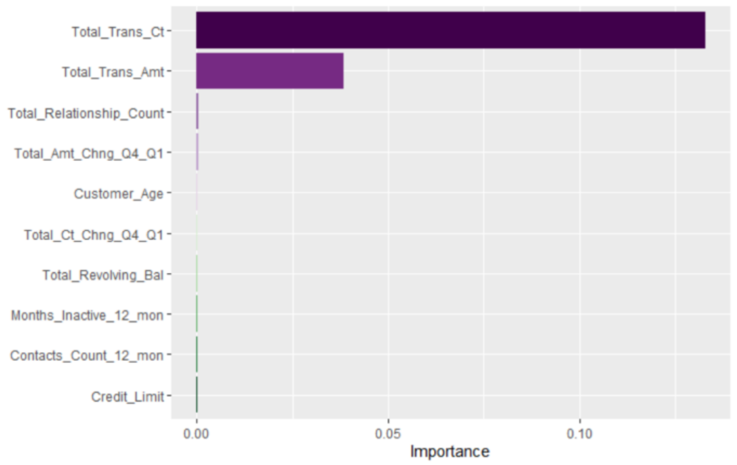
  We use 10-fold cross-validation and evaluate 10 possible values of k. The tuning process gives us the best tuning hyper-parameter k as 5, so we use 5 to fit the final model. Then we use the final model to make predictions on testing data and the AUC is around 0.859.

- **SVM**
  We fit SVM models with linear, radial and polynomial kernels. The results are summarized in Table 1. In conclusion, SVM with a radial kernel has the highest test AUC.
- **Logistic Regression**
  We exclude 2 insignificant variables (*Age* & *Book_on_Month*) after we perform backward ANOVA. The result of logistic regression is shown in Picture 3.



```
                          Estimate Std. Error z value Pr(>|z|)    
(Intercept)                2.17910    0.18541  11.753  < 2e-16 ***
Gender1                   -0.97014    0.11915  -8.142 3.88e-16 ***
Dependent_count           -0.17474    0.03304  -5.289 1.23e-07 ***
Education_Level2           0.02642    0.10709   0.247 0.805161    
Education_Level3          -0.19305    0.12412  -1.555 0.119855    
Education_Level4           0.08206    0.09739   0.843 0.399465    
Education_Level5          -0.30182    0.15325  -1.969 0.048903 *  
Education_Level6           0.05425    0.16422   0.330 0.741140    
Education_Level7           0.28273    0.11252   2.513 0.011983 *  
Income_Category2           0.24889    0.10073   2.471 0.013480 *  
Income_Category3          -0.05126    0.14947  -0.343 0.731622    
Income_Category4          -0.53793    0.14930  -3.603 0.000315 ***
Income_Category5          -0.31561    0.18036  -1.750 0.080146 .  
Income_Category6           0.29547    0.11161   2.647 0.008111 ** 
Card_Category2            -0.37302    0.15269  -2.443 0.014564 *  
Card_Category3            -0.58702    0.29388  -1.997 0.045774 *  
Card_Category4            -1.06724    0.62355  -1.712 0.086977 .  
Months_on_book             0.14359    0.03235   4.439 9.06e-06 ***
Total_Relationship_Count   0.62251    0.03353  18.564  < 2e-16 ***
Months_Inactive_12_mon    -0.66371    0.03427 -19.365  < 2e-16 ***
Contacts_Count_12_mon     -0.68699    0.03471 -19.790  < 2e-16 ***
Credit_Limit               0.09641    0.04504   2.141 0.032313 *  
Total_Revolving_Bal        0.77605    0.02824  27.483  < 2e-16 ***
Total_Amt_Chng_Q4_Q1       0.15010    0.03570   4.204 2.62e-05 ***
Total_Trans_Amt           -2.04320    0.06772 -30.169  < 2e-16 ***
Total_Trans_Ct             3.37464    0.07813  43.191  < 2e-16 ***
Total_Ct_Chng_Q4_Q1        0.79227    0.03948  20.069  < 2e-16 ***
Marital_Status_Married     0.40474    0.12514   3.234 0.001219 ** 
Marital_Status_Single     -0.27775    0.12597  -2.205 0.027466 *  
Marital_Status_Unknown     0.06632    0.15912   0.417 0.676860    
```

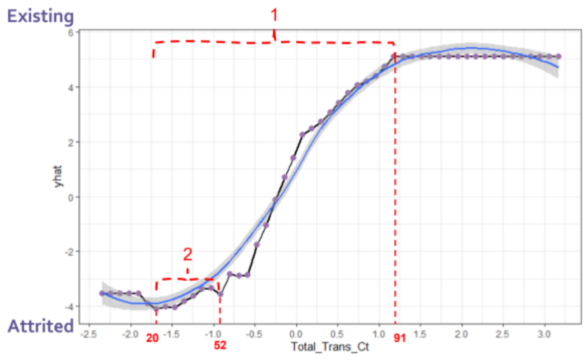
Picture 3. Output of Logistic regression

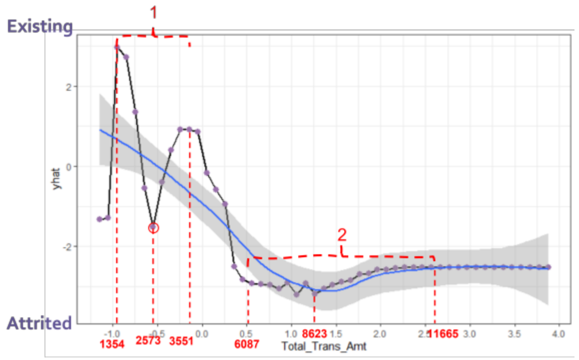


Picture 4. Important features

- **Random Forest**
  The best tuning hyper-parameter is mtry = 3(near the number of features/3) & min.node.size = 1 (the deeper tree performs better)& replace = FALSE & sample.fraction = 0.63.
- **Basic GBM**
  The distribution of the GBM function is "bernoulli" for a binary classification problem. The optimal learning rate is 0.05 and 4876 trees are required. Also, the second tuning shows that the optimal interaction.depth is 7 and n.minobsinnode is 15.
- **Stochastic GBM**
  Based on the result of Basic GBM, we tune the additional hyper-parameter(bag.fraction) and the optimal result is 0.5. We use AUC metric as our importance measurement to select important variables. As shown in Picture 4, *Total_Trans_Ct* and *Total_Trans_Amt* are the top 2 most important variables.

  Subsequently we generate the partial dependence plots (PDP) for the above mentioned 2 variables and we destandardize X axis of some special points. Y axis represents the probability of Existing or Attrition.





Picture 5. PDP of *total_transaction_count* Picture 6. PDP of *total_transaction_amount*

As shown in Picture 5, there are 2 ranges of transaction count to look into detail. First range is between 20 and 91, where transaction count has a strong positive effect on the attrition rate. Customers with more total transaction count, are more likely to stay.

The second range is between 20 and 52. The transaction count is not the lowest. However, customers who fall within this range are most likely to leave. As low total transaction count means low customer stickiness, the bank could further investigate this range to retain customers.

From Picture 6, we can see that as total transaction amount increases, the probability of attrition increases as well. Also, there are 2 ranges to highlight. First, there is a sharp V shape of probability of existing when total transaction amount is between $1,354 and $3,551. This group consists of 38% of the total customers, which is definitely a significant portion. The bank should obtain more information to understand the strange fluctuation and develop corresponding solutions so as to mitigate the attrition.

The second range is between $6,087 and $11,665. Customers who are in this range have high transaction amounts and can generate more revenue for the bank. Therefore, they are regarded as VIP customers. However, we can see that from this PDP, VIP customers tend to have a high attrition rate compared to less important customers. For these VIP customers, the bank could offer personalized products and services to retain them.

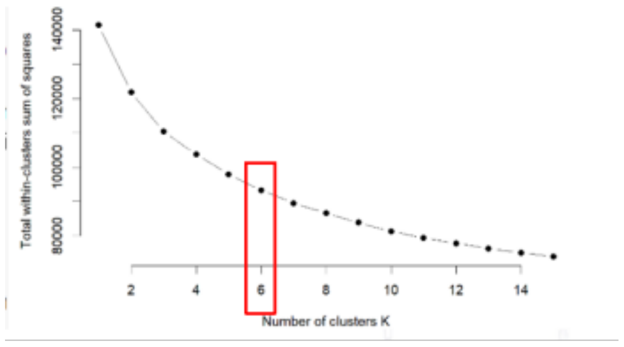
- **XGBoost**
  According to the tuning result of XGBoost, the best hyper-parameter is gamma = 0, lambda = 1e-02, alpha = 0e+00. There is a common problem for tree-based methods, that they all have overfitting problems. But for XGBoost, the overfitting problem isn't that worse.

4.2 K-means clustering

By using k-means clustering, we aim to divide customers into several groups to analyze their features, so that we can better serve their needs and potentially improve the customer experience, which may lead to less attrition.

The basic idea behind k-means clustering consists of defining clusters so that the total intra-cluster variation (known as total within-cluster variation) is minimized.

In this case, we first drop the categorical variables and response. There are mainly two reasons: 1. Numeric variables are mostly bank activity features, which are more directly correlated with the bank's action. 2. k-means clustering is only suitable for numeric data. To find the optimal number of clusters, we use the elbow method.



Picture 6

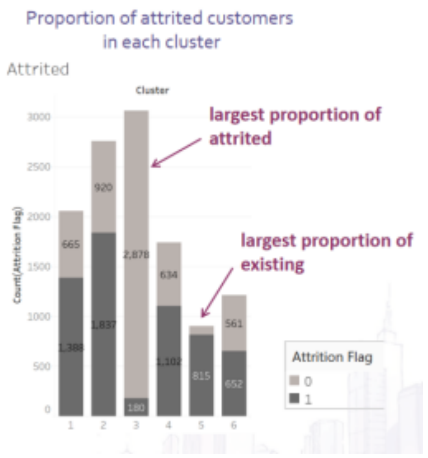
Picture 6 represents the variance within the clusters. It decreases as k increases. Since there is not a very obvious bend or elbow, we choose a number with relatively small WSS (within-clusters sum of squares) while controlling the interpretability of the model. Then, we classify the observations into 6 clusters.

After clustering, we get 6 clusters with a different mix of bank activity features.
Cluster1: Heavy users with low balance, tend to spend more on Q4 than Q1.
Cluster2: Light users, low limit and high balance.
Cluster3: Light users, low limit, inactive, low balance but high number of contacts with the bank.

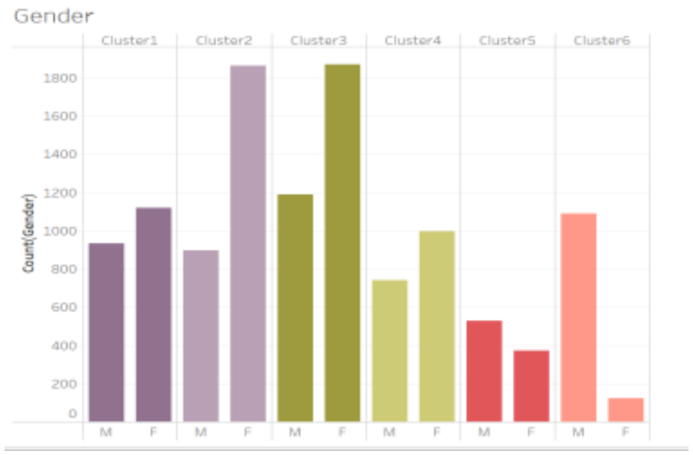
Cluster4: Long time light users with high inactive month, low limit, high balance.
Cluster5: Heavy users, high limit, high balance.
Cluster6: Light users, high limit, low balance.

Also, we can plot the bar chart of the proportion of attrited customers in each cluster (Picture 7). As shown in Picture 8, Cluster 3 has a significantly higher rate of customer attrition, while Cluster 5 has a significantly lower rate of attrition.

Combined with the bank activity features of Cluster 3, we can make some inferences, for example, that they might have some unaddressed issues, and that our customer service needs improvement. Likewise, we can get a whole picture of their demographic features and draw some conclusions. For instance, from the bar chart of the gender distribution of each cluster, we can see that there are more females in cluster 1, 2, 3 and 4, while more males in cluster 5 and 6.



Picture 7



Picture 8

By integrating the bank activity features and demographic features, the bank can clearly identify customers' labels and provide corresponding service. Consequently, with the help of clustering information, the bank can serve its customers' needs better. Instead of designing services for each individual customer, the bank can develop about 6 kinds of services only. Therefore, if we spot a customer with a high attrition rate, we can act to prevent attrition more efficiently and effectively.

5. Conclusion

When comparing all the test AUC of all models, we find that Stochastic GBM has the highest test AUC. Therefore, Stochastic GBM is the optimal model for HSAC bank to predict its credit card customer churn.

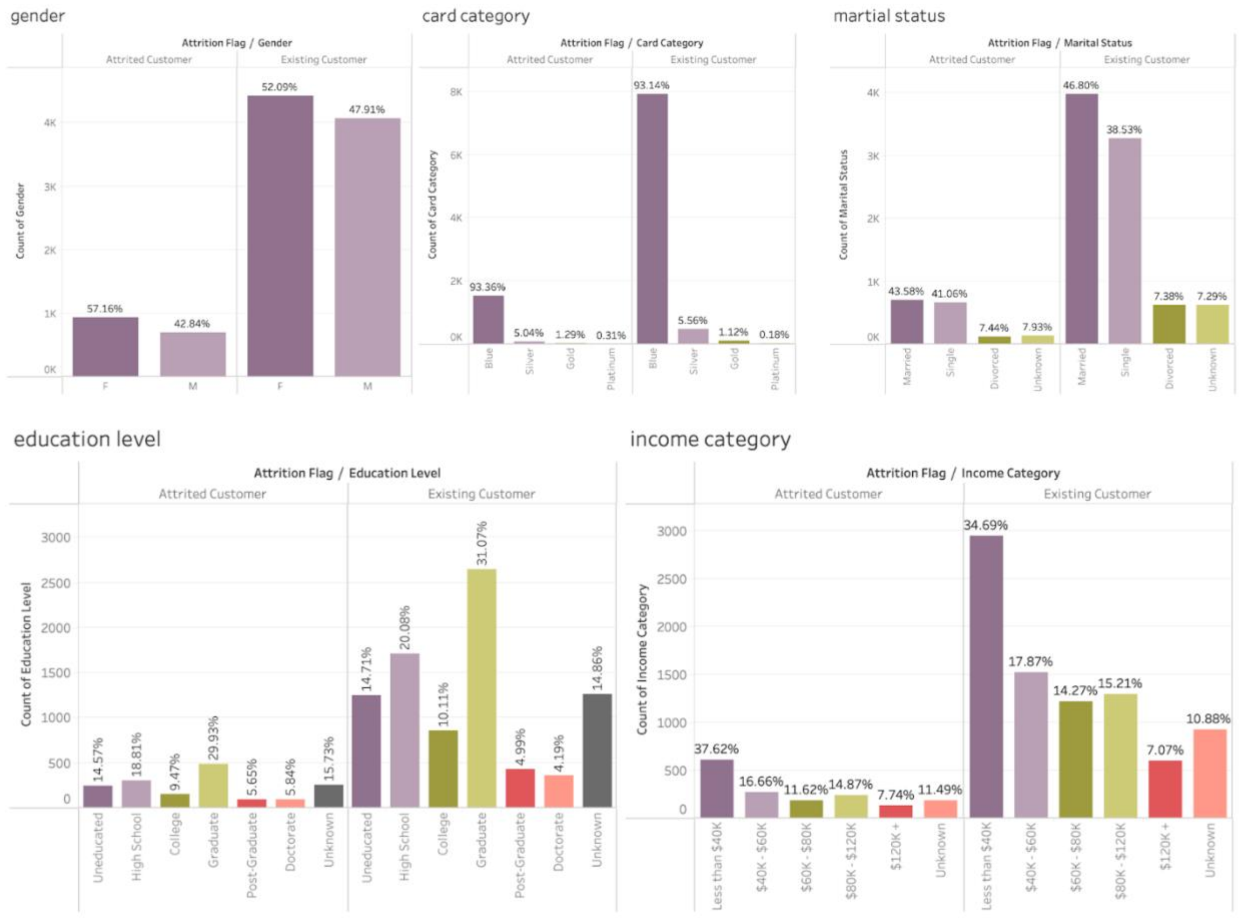
From the PDP we can conclude that the top 2 most Important Variables are *Total Transaction Count* and *Total Transaction Amount*. We found that a customer is more likely to churn if his/her transaction count ranges from 20 to 50. However, we also noticed a strange pattern. There is a huge fluctuation of attrition rate when a customer's transaction amount ranges between $1,354 and $3,551. We therefore suggest HSAC bank to further investigate the customers whose total transaction amount fall within this range and develop corresponding strategies. Last but not least, VIP customers whose transaction amount ranges from $6,087 to $11,665 have a high attrition rate. HSAC bank should pay special attention to this group of customers as VIP is more likely to generate a considerable amount of revenue for the bank.

From the Logistic Regression, we find that customers with the following attributes are more likely to churn: (1) Female, (2) Income ranges from $80K to $120K, (3) Longer inactive time, (4) More contacts with bank, (5) More dependent counts, (6) Higher total transaction amount in the past 12 months.

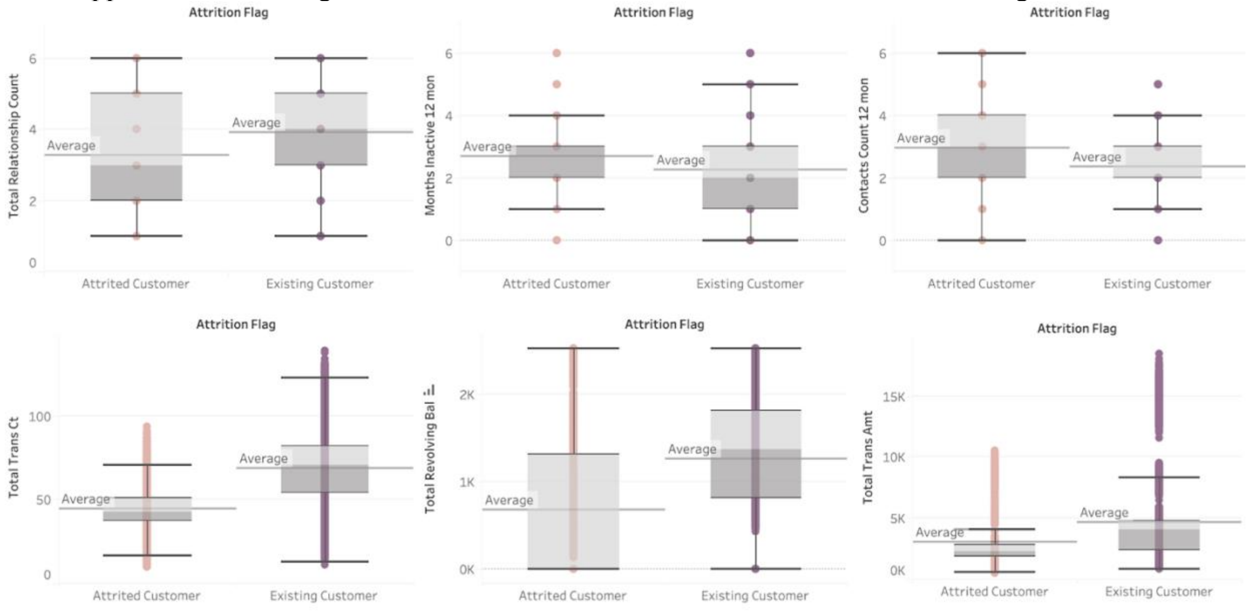
6. Reference

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, *16*, 321–357.

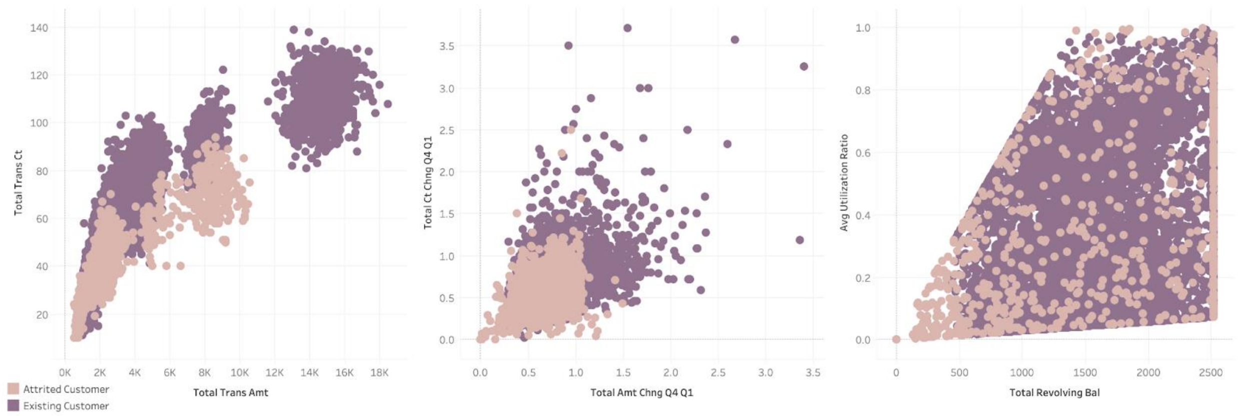
7. Appendix



Appendix 1- All categorical variables with similar distribution between two categories of customers



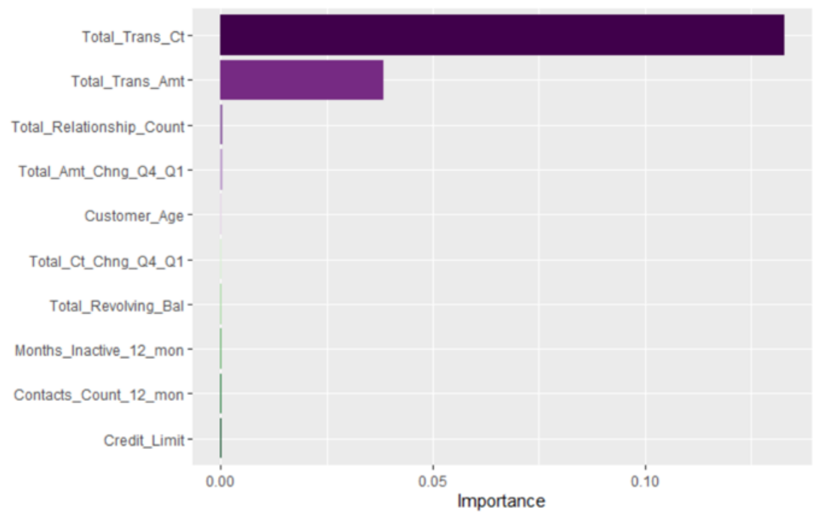
Appendix 2 - Six numerical variables with significant disparity on average between two categories of customers



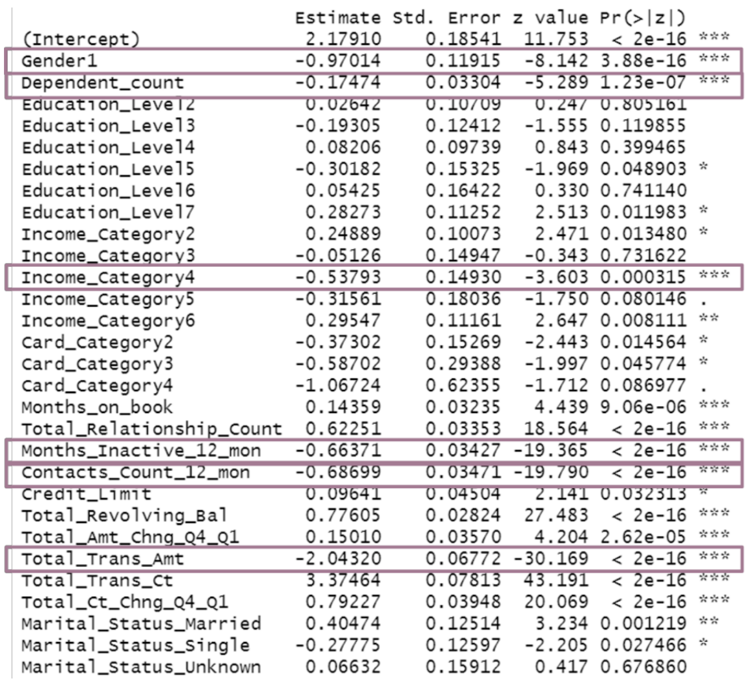
Appendix 3 - Highly correlated variables

Model	KNN	SVM - Linear	SVM - Radial	SVM - Polynomial	Logistic Regression	Random Forest	Basic GBM	Stochastic GBM	XGBoost
Test AUC	0.859	0.8438	0.8905	0.8824	0.8417	0.9304	0.9574	0.9578	0.9289

Appendix 4 - Test AUC of supervised models

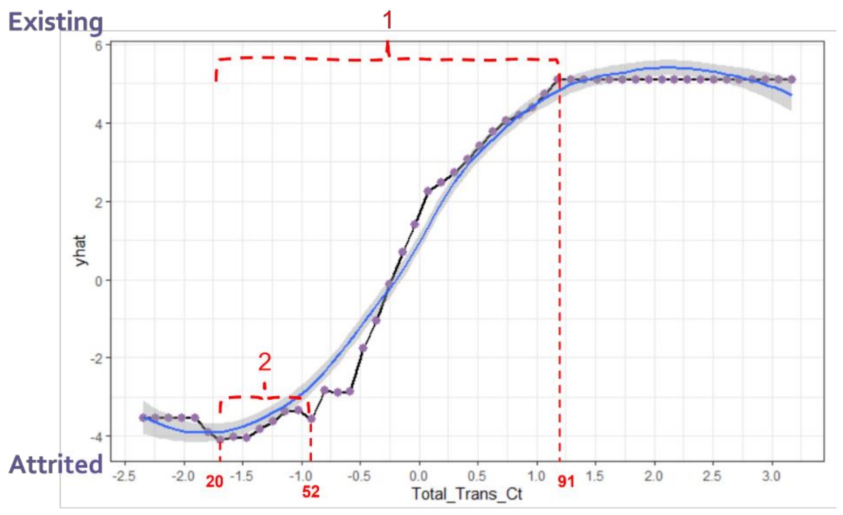


Appendix 5 - Important Features of stochastic GBM

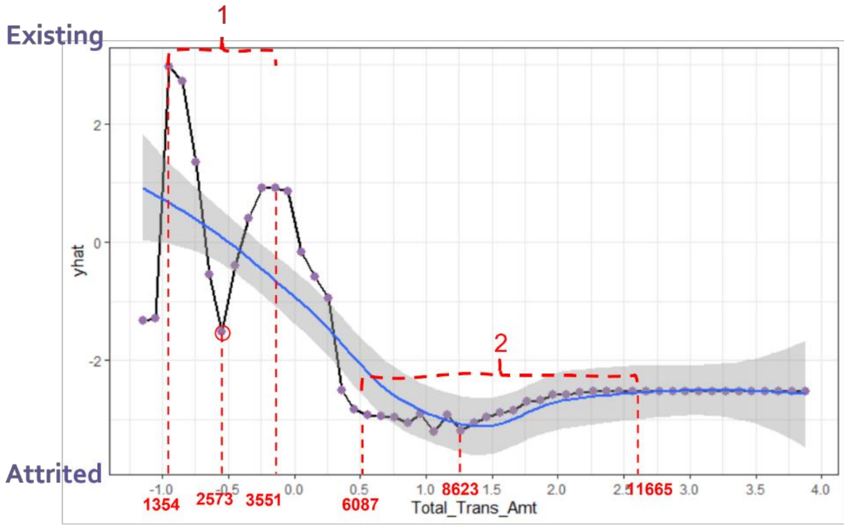


```
                        Estimate Std. Error z value Pr(>|z|)    
(Intercept)              2.17910    0.18541  11.753  < 2e-16 ***
Gender1                 -0.97014    0.11915  -8.142 3.88e-16 ***
Dependent_count         -0.17474    0.03304  -5.289 1.23e-07 ***
Education_Level2         0.02642    0.10709   0.247 0.805161    
Education_Level3        -0.19305    0.12412  -1.555 0.119855    
Education_Level4         0.08206    0.09739   0.843 0.399465    
Education_Level5        -0.30182    0.15325  -1.969 0.048903 *  
Education_Level6         0.05425    0.16422   0.330 0.741140    
Education_Level7         0.28273    0.11252   2.513 0.011983 *  
Income_Category2         0.24889    0.10073   2.471 0.013480 *  
Income_Category3        -0.05126    0.14947  -0.343 0.731622    
Income_Category4        -0.53793    0.14930  -3.603 0.000315 ***
Income_Category5        -0.31561    0.18036  -1.750 0.080146 .  
Income_Category6         0.29547    0.11161   2.647 0.008111 ** 
Card_Category2          -0.37302    0.15269  -2.443 0.014564 *  
Card_Category3          -0.58702    0.29388  -1.997 0.045774 *  
Card_Category4          -1.06724    0.62355  -1.712 0.086977 .  
Months_on_book           0.14359    0.03235   4.439 9.06e-06 ***
Total_Relationship_Count 0.62251    0.03353  18.564  < 2e-16 ***
Months_Inactive_12_mon  -0.66371    0.03427 -19.365  < 2e-16 ***
Contacts_Count_12_mon   -0.68699    0.03471 -19.790  < 2e-16 ***
Credit_Limit             0.09641    0.04504   2.141 0.032313 *  
Total_Revolving_Bal      0.77605    0.02824  27.483  < 2e-16 ***
Total_Amt_Chng_Q4_Q1     0.15010    0.03570   4.204 2.62e-05 ***
Total_Trans_Amt         -2.04320    0.06772 -30.169  < 2e-16 ***
Total_Trans_Ct           3.37464    0.07813  43.191  < 2e-16 ***
Total_Ct_Chng_Q4_Q1      0.79227    0.03948  20.069  < 2e-16 ***
Marital_Status_Married   0.40474    0.12514   3.234 0.001219 ** 
Marital_Status_Single   -0.27775    0.12597  -2.205 0.027466 *  
Marital_Status_Unknown   0.06632    0.15912   0.417 0.676860    
```

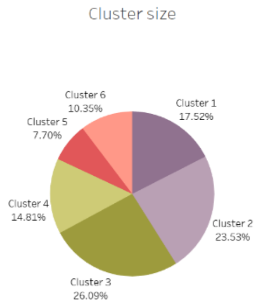
Appendix 6 – Output of logistic regression



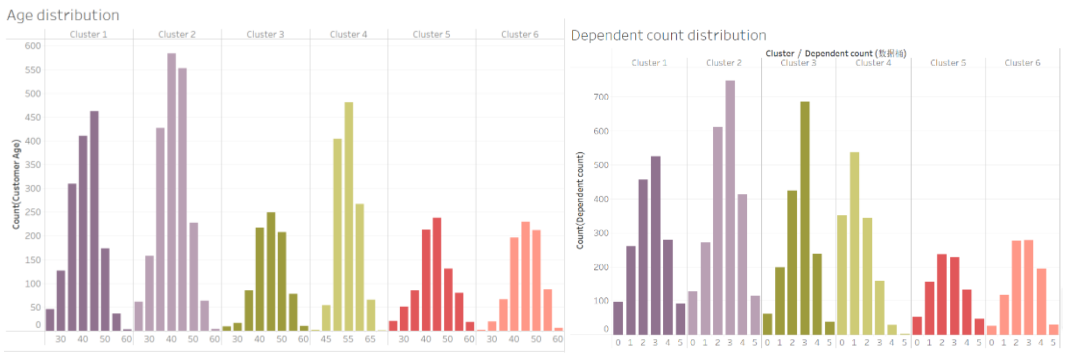
Appendix 7 - PDP of total transaction count

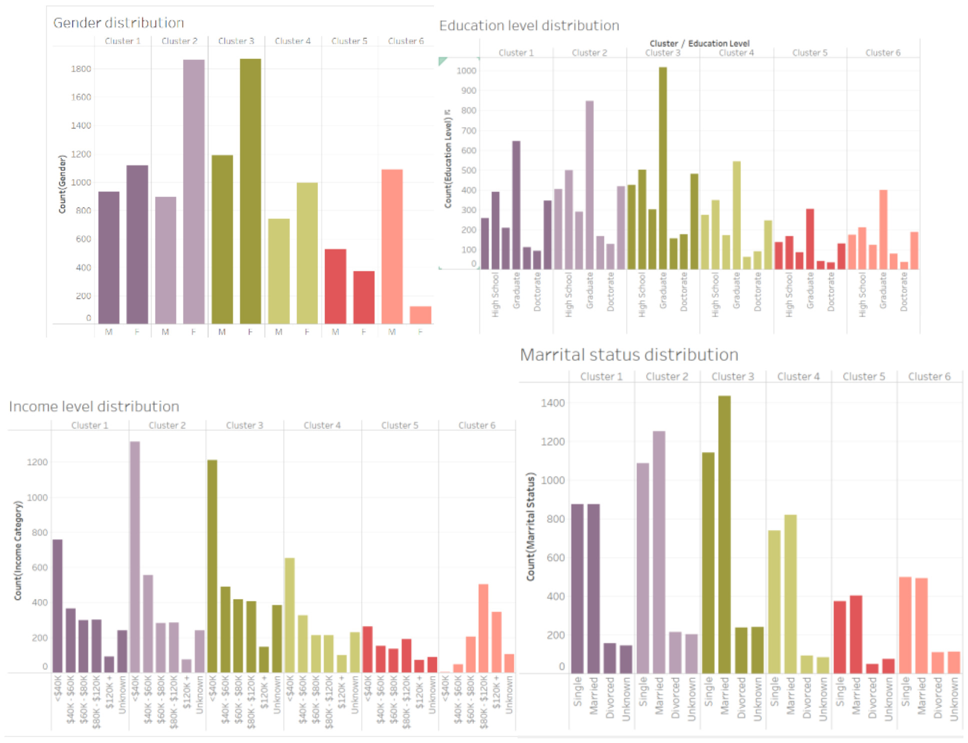


Appendix 8 - PDP of total transaction amount

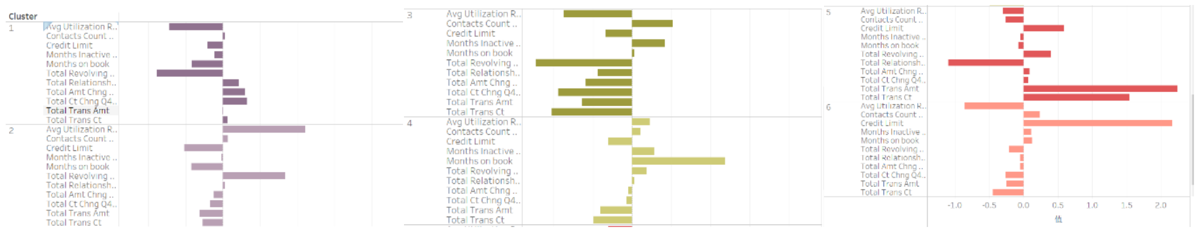


Appendix 9 - Size of each cluster





Appendix 10 - Distribution of demographic features of each cluster



Appendix 11 - Distribution of bank activity features of each cluster

GP

Group3

2021/12/14

Import Data

```
# read data and remove the first line
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 4.1.2
```

```
##
## Attaching package: 'dplyr'
```

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

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

```
customer <- read.table("Project_Data_3.txt",sep="\t",header=FALSE)
customer <- customer[-1,]
names(customer) <- c(customer[1,])
customer <- customer[-1,]
# customer
```

Data pre-processing

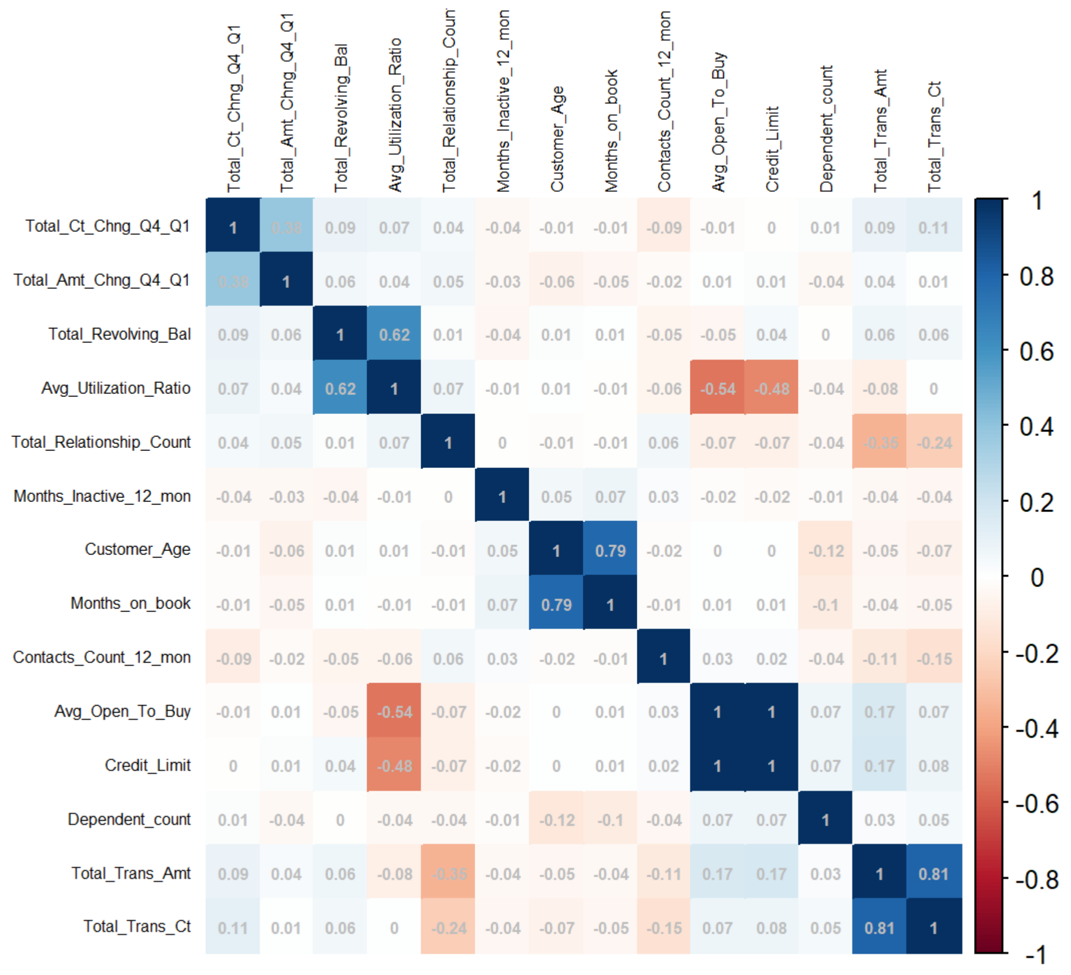
```
# read data
library(dplyr)
library(corrplot)
```

```
## Warning: package 'corrplot' was built under R version 4.1.2
```

```
## corrplot 0.92 loaded
```

```r
# remove 3 irrelevant variables
customers <- customer %>%
  select(-c(CLIENTNUM,
            Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_
count_Education_Level_Months_Inactive_12_mon_1,
            Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_
count_Education_Level_Months_Inactive_12_mon_2))

# change datatype
customers$Customer_Age = as.numeric(customers$Customer_Age)
customers$Dependent_count = as.numeric(customers$Dependent_count)
customers$Months_on_book = as.numeric(customers$Months_on_book)
customers$Total_Relationship_Count = as.numeric(customers$Total_Relationship_Count)
customers$Months_Inactive_12_mon = as.numeric(customers$Months_Inactive_12_mon)
customers$Contacts_Count_12_mon = as.numeric(customers$Contacts_Count_12_mon)
customers$Credit_Limit = as.numeric(customers$Credit_Limit)
customers$Total_Revolving_Bal = as.numeric(customers$Total_Revolving_Bal)
customers$Avg_Open_To_Buy = as.numeric(customers$Avg_Open_To_Buy)
customers$Total_Amt_Chng_Q4_Q1 = as.numeric(customers$Total_Amt_Chng_Q4_Q1)
customers$Total_Trans_Amt = as.numeric(customers$Total_Trans_Amt)
customers$Total_Trans_Ct = as.numeric(customers$Total_Trans_Ct)
customers$Total_Ct_Chng_Q4_Q1 = as.numeric(customers$Total_Ct_Chng_Q4_Q1)
customers$Avg_Utilization_Ratio = as.numeric(customers$Avg_Utilization_Ratio)
# customers
```

```r
# correlation plot
customers %>% select(where(is.numeric)) %>% as.matrix() %>%
cor() %>% corrplot(method = "color",shade.col = NA, tl.col ="black", tl.srt = 90, order = "AOE",
tl.cex=0.5,number.cex=0.5,addCoef.col = "grey")
```



```r
# remove 1 collinear variable
customers <- customers %>%
  select(-c(Avg_Open_To_Buy))
```

```r
# define order level of ordinal categorical variables
customers$Education_Level= factor(customers$Education_Level, levels=c("Uneducated", "High Schoo
l", "College","Graduate","Post-Graduate","Doctorate","Unknown"),order=T)

customers$Income_Category= factor(customers$Income_Category, levels=c("Less than $40K", "$40K -
 $60K", "$60K - $80K", "$80K - $120K", "$120K +","Unknown"),order=T)

customers$Card_Category= factor(customers$Card_Category, levels=c("Blue", "Silver", "Gold", "Pla
tinum"),order=T)

# define binary variable
customers$Gender = as.factor(ifelse(customers$Gender == "M",0,1))

# define response variable
customers$Attrition_Flag = as.factor(ifelse(customers$Attrition_Flag == "Existing Customer", 1,
0))
```

```r
# split into train & test set
library(recipes)
```

```
## Warning: package 'recipes' was built under R version 4.1.2
```

```
##
## Attaching package: 'recipes'
```

```
## The following object is masked from 'package:stats':
##
##     step
```

```
set.seed(123)

split  <- rsample::initial_split(customers, prop = 0.7)
train <- rsample::training(split)
test <- rsample::testing(split)
```

```
# feature engineering blueprint
blueprint<-recipe(Attrition_Flag~.,data=train)%>%
  step_nzv(all_nominal())%>%
  step_center(all_numeric(), -all_outcomes()) %>%
  step_scale(all_numeric(), -all_outcomes())%>%
  step_integer(matches("Education_Level|Income_Category|Card_Category")) %>%
  step_dummy(matches("Marital_Status"))
  prepare <- prep(blueprint, training = train)

customers.train <- bake(prepare, new_data = train)
customers.test <- bake(prepare, new_data = test)

customers.train$Gender<-as.factor(customers.train$Gender)
customers.train$Education_Level<-as.factor(customers.train$Education_Level)
customers.train$Income_Category<-as.factor(customers.train$Income_Category)
customers.train$Card_Category<-as.factor(customers.train$Card_Category)
customers.test$Gender<-as.factor(customers.test$Gender)
customers.test$Education_Level<-as.factor(customers.test$Education_Level)
customers.test$Income_Category<-as.factor(customers.test$Income_Category)
customers.test$Card_Category<-as.factor(customers.test$Card_Category)
```

```
#no. of existing(1)/attrited(0) customers before smote
table(customers.train$Attrition_Flag)
```

```
##
##    0    1
## 1149 5939
```

SMOTE

```
#SMOTE FOR ---tree based model---
#smote to deal with imbalanced data
library(performanceEstimation)
```

```
## Warning: package 'performanceEstimation' was built under R version 4.1.2
```

```
set.seed(123)
customers.train.smd1<-smote(Attrition_Flag~., customers.train, perc.over = 4, k=5, perc.under =
1.3)
customers.smoted1<-customers.train.smd1
```

```
#with 0 : 1 around 50% : 50%1
table(customers.smoted1$Attrition_Flag)
```

```
##
##    0    1
## 5745 5974
```

```
#SMOTE FOR ---SVM---
#smote to deal with imbalanced data
#computer crashed when we fit SVM with customers.smoted1 which has around 10,000 samples
#therefore, we created below customers.smoted2 in order to get result
library(performanceEstimation)
set.seed(123)
customers.train.smd2<-smote(Attrition_Flag~., customers.train, perc.over = 1, k=5, perc.under =
2)
customers.smoted2<-customers.train.smd2
```

```
#reduced sample size to 4596 for SVM
#with 0 : 1 around 50% : 50%
table(customers.smoted2$Attrition_Flag)
```

```
##
##    0    1
## 2298 2298
```

Logistic Regression

```
#Smoothing Spline for Logistic regression
library(gam) #load package
```

```
## Warning: package 'gam' was built under R version 4.1.2
```

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

```
## Warning: package 'foreach' was built under R version 4.1.2
```

```
## Loaded gam 1.20
```

```
#fit logistic regression model
fit_log <- glm(Attrition_Flag ~.,family = binomial(logit), data = customers.smoted1[,-c(1,17)])
summary(fit_log)
```

```
##
## Call:
## glm(formula = Attrition_Flag ~ ., family = binomial(logit), data = customers.smoted1[,
##     -c(1, 17)])
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.1385  -0.4268   0.0446   0.3694   3.5253
##
## Coefficients:
##                         Estimate Std. Error z value Pr(>|z|)
## (Intercept)              2.17910    0.18541  11.753  < 2e-16 ***
## Gender1                 -0.97014    0.11915  -8.142 3.88e-16 ***
## Dependent_count         -0.17474    0.03304  -5.289 1.23e-07 ***
## Education_Level2         0.02642    0.10709   0.247 0.805161
## Education_Level3        -0.19305    0.12412  -1.555 0.119855
## Education_Level4         0.08206    0.09739   0.843 0.399465
## Education_Level5        -0.30182    0.15325  -1.969 0.048903 *
## Education_Level6         0.05425    0.16422   0.330 0.741140
## Education_Level7         0.28273    0.11252   2.513 0.011983 *
## Income_Category2         0.24889    0.10073   2.471 0.013480 *
## Income_Category3        -0.05126    0.14947  -0.343 0.731622
## Income_Category4        -0.53793    0.14930  -3.603 0.000315 ***
## Income_Category5        -0.31561    0.18036  -1.750 0.080146 .
## Income_Category6         0.29547    0.11161   2.647 0.008111 **
## Card_Category2          -0.37302    0.15269  -2.443 0.014564 *
## Card_Category3          -0.58702    0.29388  -1.997 0.045774 *
## Card_Category4          -1.06724    0.62355  -1.712 0.086977 .
## Months_on_book           0.14359    0.03235   4.439 9.06e-06 ***
## Total_Relationship_Count  0.62251    0.03353  18.564  < 2e-16 ***
## Months_Inactive_12_mon  -0.66371    0.03427 -19.365  < 2e-16 ***
## Contacts_Count_12_mon   -0.68699    0.03471 -19.790  < 2e-16 ***
## Credit_Limit             0.09641    0.04504   2.141 0.032313 *
## Total_Revolving_Bal      0.77605    0.02824  27.483  < 2e-16 ***
## Total_Amt_Chng_Q4_Q1     0.15010    0.03570   4.204 2.62e-05 ***
## Total_Trans_Amt         -2.04320    0.06772 -30.169  < 2e-16 ***
## Total_Trans_Ct           3.37464    0.07813  43.191  < 2e-16 ***
## Total_Ct_Chng_Q4_Q1      0.79227    0.03948  20.069  < 2e-16 ***
## Marital_Status_Married   0.40474    0.12514   3.234 0.001219 **
## Marital_Status_Single   -0.27775    0.12597  -2.205 0.027466 *
## Marital_Status_Unknown   0.06632    0.15912   0.417 0.676860
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 16241.5  on 11718  degrees of freedom
## Residual deviance:  7192.6  on 11689  degrees of freedom
## AIC: 7252.6
##
## Number of Fisher Scoring iterations: 6
```

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.1.2
```

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 4.1.2
```

```
## Loading required package: lattice
```

```
#training accuracy
pred_glm_train<- predict(fit_log, newdata = customers.smoted1[,-c(1,17)],type='response')
pred_glm_train<-factor(ifelse(pred_glm_train<0.5,0,1))
confusionMatrix(pred_glm_train,customers.smoted1$Attrition_Flag)
```

```
#testing accuracy
pred_glm_test<- predict(fit_log, newdata = customers.test[,-c(1,17)],type='response')
pred_glm_test<-factor(ifelse(pred_glm_test<0.5,0,1))
confusionMatrix(pred_glm_test,customers.test$Attrition_Flag)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0  397  377
##          1   81 2184
##
##                Accuracy : 0.8493
##                  95% CI : (0.8361, 0.8618)
##     No Information Rate : 0.8427
##     P-Value [Acc > NIR] : 0.1657
##
##                   Kappa : 0.5459
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.8305
##             Specificity : 0.8528
##          Pos Pred Value : 0.5129
##          Neg Pred Value : 0.9642
##              Prevalence : 0.1573
##          Detection Rate : 0.1306
##    Detection Prevalence : 0.2547
##       Balanced Accuracy : 0.8417
##
##        'Positive' Class : 0
##
```

```
library(pROC)
```

```
## Warning: package 'pROC' was built under R version 4.1.2
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

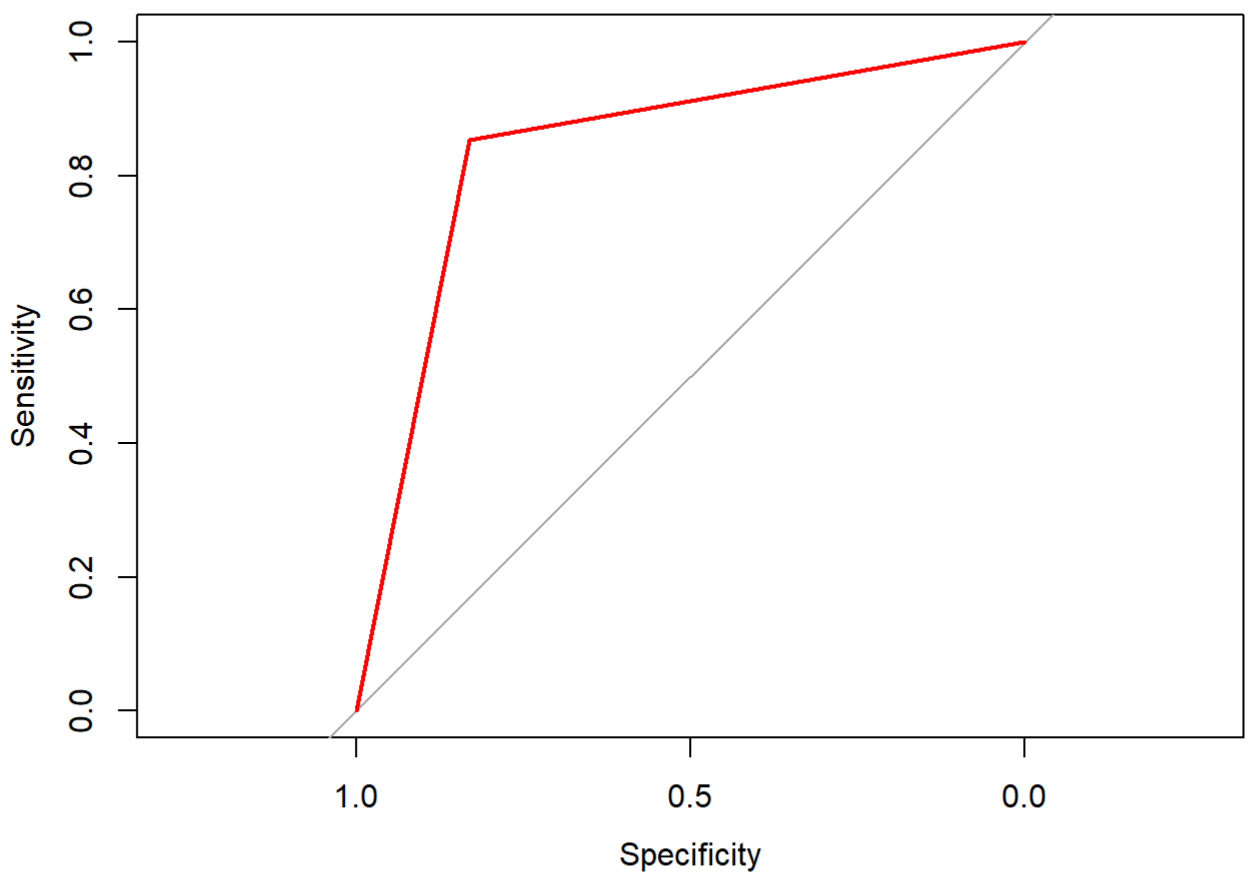
```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
roc_glm_test <- roc(as.numeric(customers.test$Attrition_Flag), as.numeric(pred_glm_test))
```

```
## Setting levels: control = 1, case = 2
```

```
## Setting direction: controls < cases
```

```
plot(roc_glm_test,col = "red")#draw the roc plots of the three model
```



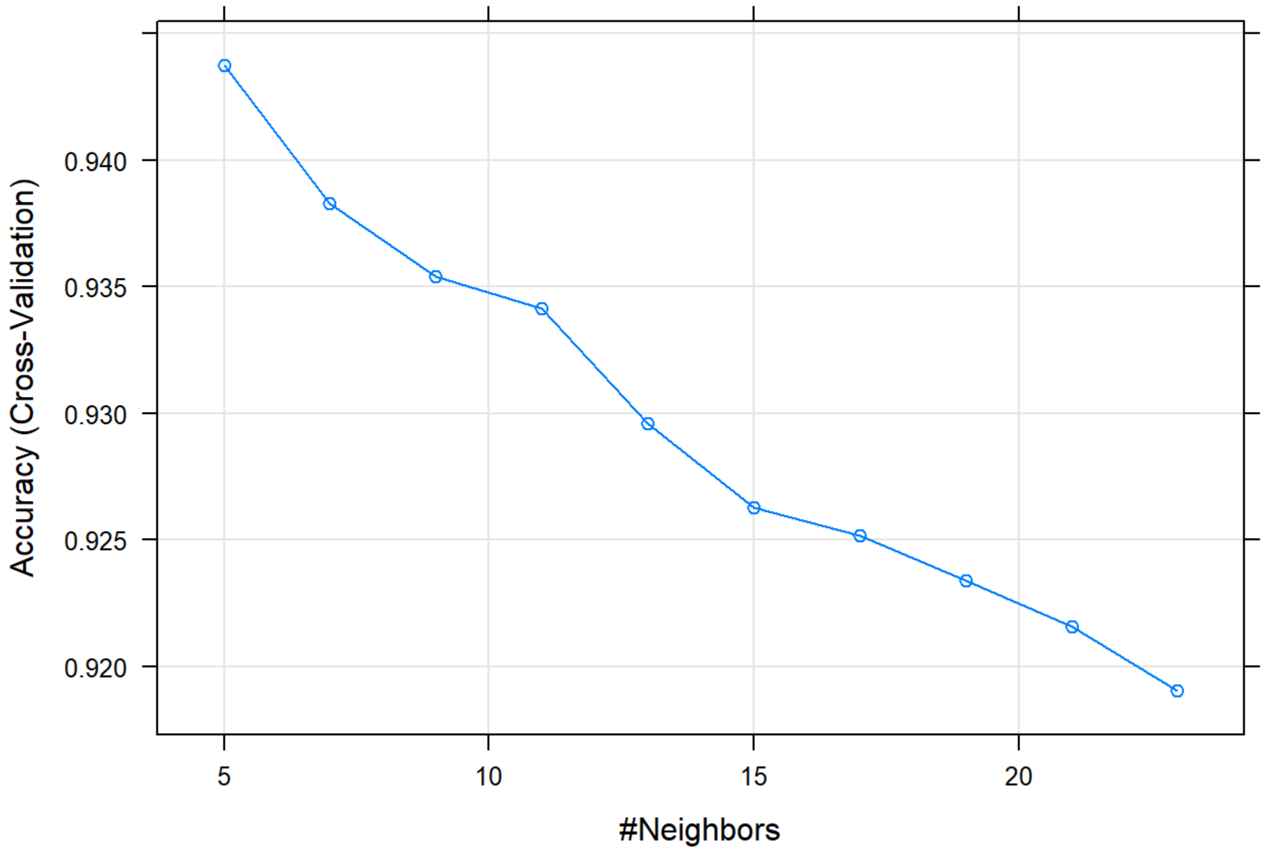
```
roc_glm_test$auc
```

```
## Area under the curve: 0.8417
```

KNN

```
library(caret)
set.seed(123)
knn_fit <- train(
  Attrition_Flag~., data = customers.smoted1, method = "knn",
  trControl = trainControl("cv", number = 10),
  tuneLength = 10
  )
```

```
# Plot model error RMSE vs different values of k
plot(knn_fit)
```



```
# Best tuning parameter k that minimize the RMSE
knn_fit$bestTune
```

	k <int>
1	5

1 row

```
#training accuracy
pred_knn_train <- knn_fit %>% predict(customers.smoted1)
table(true=customers.smoted1$Attrition_Flag,predict=pred_knn_train)
```

```
##     predict
## true    0    1
##    0 5726   19
##    1  389 5585
```

```
mean(pred_knn_train!= customers.smoted1$Attrition_Flag)
```

```
## [1] 0.03481526
```

```
1-mean(pred_knn_train!= customers.smoted1$Attrition_Flag)
```

```
## [1] 0.9651847
```

```
# make predictions, show confusion matrix and testing accuracy
pred_knn_test <- knn_fit %>% predict(customers.test)
table(true=customers.test$Attrition_Flag,predict=pred_knn_test)
```

```
##     predict
## true    0    1
##    0  410   68
##    1  358 2203
```

```
mean(pred_knn_test!= customers.test$Attrition_Flag)
```

```
## [1] 0.1401777
```

```
1-mean(pred_knn_test!= customers.test$Attrition_Flag)
```

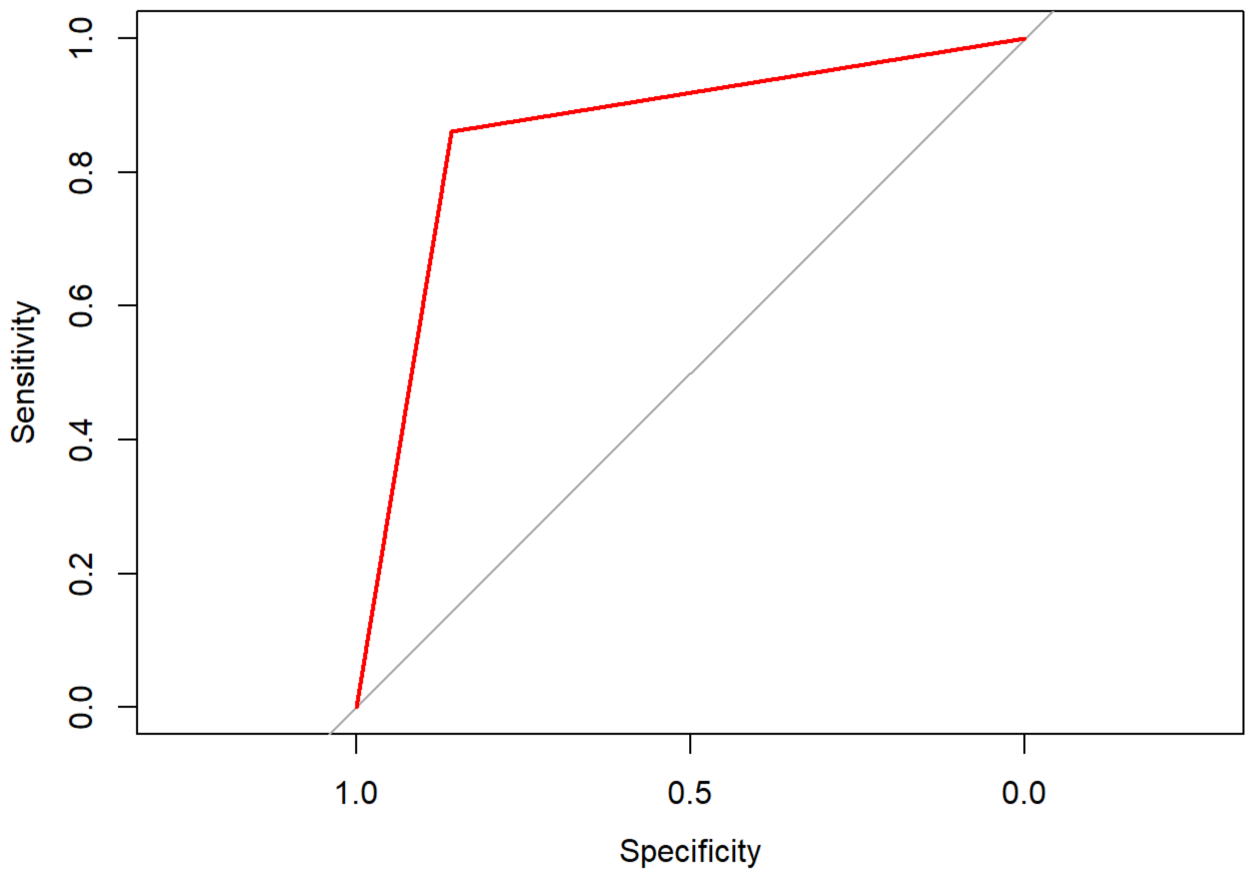
```
## [1] 0.8598223
```

```
library(pROC)
roc_knn_test <- roc(as.numeric(customers.test$Attrition_Flag), as.numeric(pred_knn_test))
```

```
## Setting levels: control = 1, case = 2
```

```
## Setting direction: controls < cases
```

```
plot(roc_knn_test,col = "red")#draw the roc plots of the three model
```



```r
roc_knn_test$auc
```

```
## Area under the curve: 0.859
```

SVM

```r
#tune the parameter of linear kernel
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 4.1.2
```

```r
set.seed(999)
tune.out<-tune(svm,Attrition_Flag~.,data=customers.smoted2,kernel='linear',ranges=list(cost=c(0.
01,0.1,1,10)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.1379521
##
## - Detailed performance results:
##    cost     error dispersion
## 1  0.01 0.1446983 0.01319688
## 2  0.10 0.1390395 0.01486667
## 3  1.00 0.1379525 0.01500811
## 4 10.00 0.1379521 0.01413705
```

```
svm_fit1<-svm(Attrition_Flag~.,data=customers.smoted2,kernel='linear',cost=10)
#make predictions, show confusion matrix and accuracy
pred_test1<-predict(svm_fit1,customers.test)
table(true=customers.test$Attrition_Flag,predict=pred_test1)
```

```
##     predict
## true    0    1
##    0  408   70
##    1  425 2136
```

```
1-mean(pred_test1!= customers.test$Attrition_Flag)
```

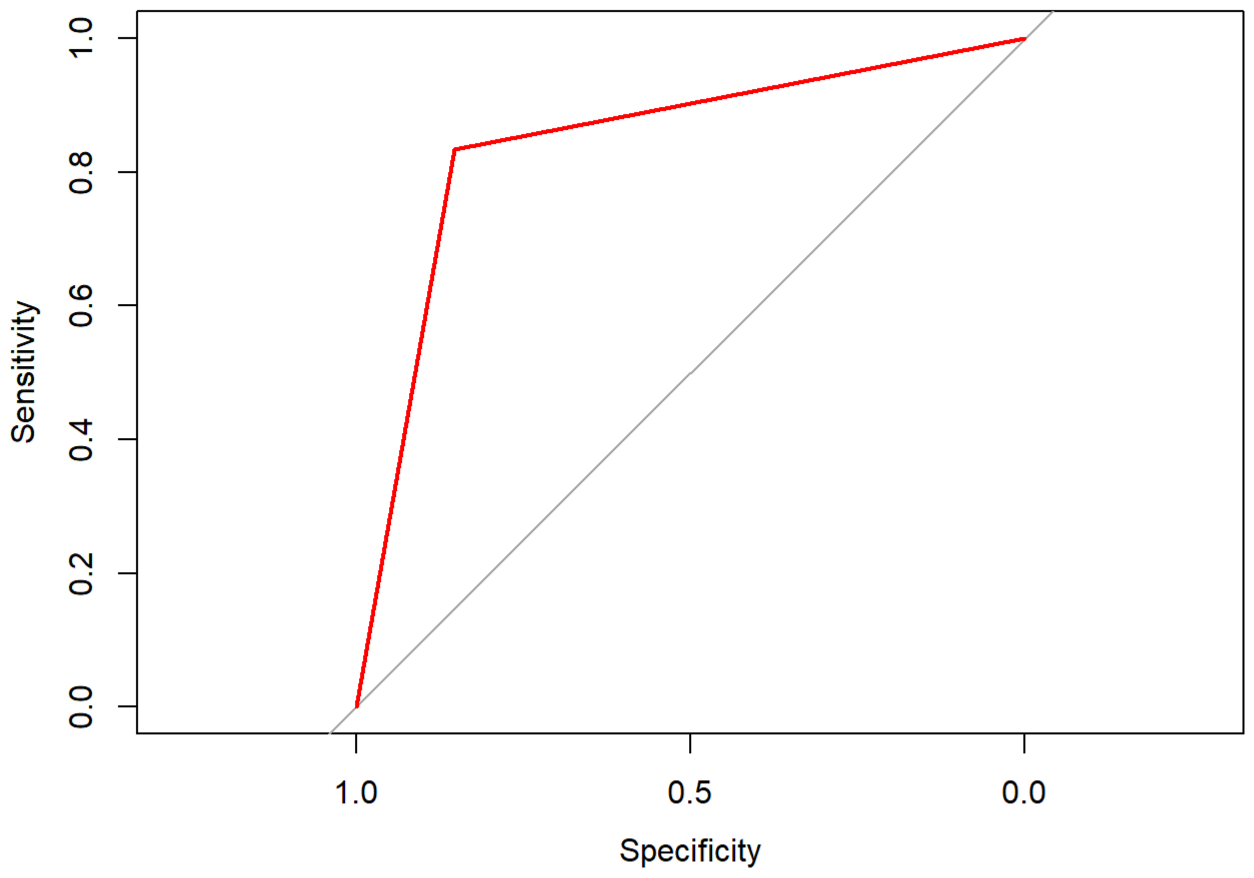
```
## [1] 0.8371175
```

```
library(pROC)
roc_svm1_test <- roc(as.numeric(customers.test$Attrition_Flag), as.numeric(pred_test1))
```

```
## Setting levels: control = 1, case = 2
```

```
## Setting direction: controls < cases
```

```
plot(roc_svm1_test,col = "red")#draw the roc plots of the three model
```



```
roc_svm1_test$auc
```

```
## Area under the curve: 0.8438
```

```
#tune the parameter of radial kernel
library(e1071)
set.seed(123)
tune.out2<-tune(svm,Attrition_Flag~.,data=customers.smoted2,kernel='radial',ranges=list(cost=c(
0.01, 0.1, 1, 10),gamma=c(0.01,0.1,0.5,1,2)))
summary(tune.out2)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##    10   0.1
##
## - best performance: 0.04787061
##
## - Detailed performance results:
##     cost gamma      error  dispersion
## 1   0.01  0.01 0.19951122 0.020951843
## 2   0.10  0.01 0.14947191 0.017568541
## 3   1.00  0.01 0.10639055 0.015827039
## 4  10.00  0.01 0.08289381 0.008569583
## 5   0.01  0.10 0.23040779 0.018572319
## 6   0.10  0.10 0.10878990 0.013766474
## 7   1.00  0.10 0.05852894 0.008670647
## 8  10.00  0.10 0.04787061 0.008418990
## 9   0.01  0.50 0.51457990 0.011840396
## 10  0.10  0.50 0.35730700 0.136516131
## 11  1.00  0.50 0.07811594 0.011832773
## 12 10.00  0.50 0.07594061 0.012651707
## 13  0.01  1.00 0.51457990 0.011840396
## 14  0.10  1.00 0.51457990 0.011840396
## 15  1.00  1.00 0.13142181 0.012970980
## 16 10.00  1.00 0.12358861 0.012282683
## 17  0.01  2.00 0.51457990 0.011840396
## 18  0.10  2.00 0.51457990 0.011840396
## 19  1.00  2.00 0.16123851 0.040339802
## 20 10.00  2.00 0.15231647 0.034885353
```

```
svm_fit2<-svm(Attrition_Flag~.,data=customers.smoted2,kernel='radial',gamma=0.1,cost=10)
#make predictions, show confusion matrix and accuracy
pred_test2<-predict(svm_fit2,customers.test)
table(true=customers.test$Attrition_Flag,predict=pred_test2)
```

```
##     predict
## true    0    1
##    0  412   66
##    1  207 2354
```

```
1-mean(pred_test2!= customers.test$Attrition_Flag)
```

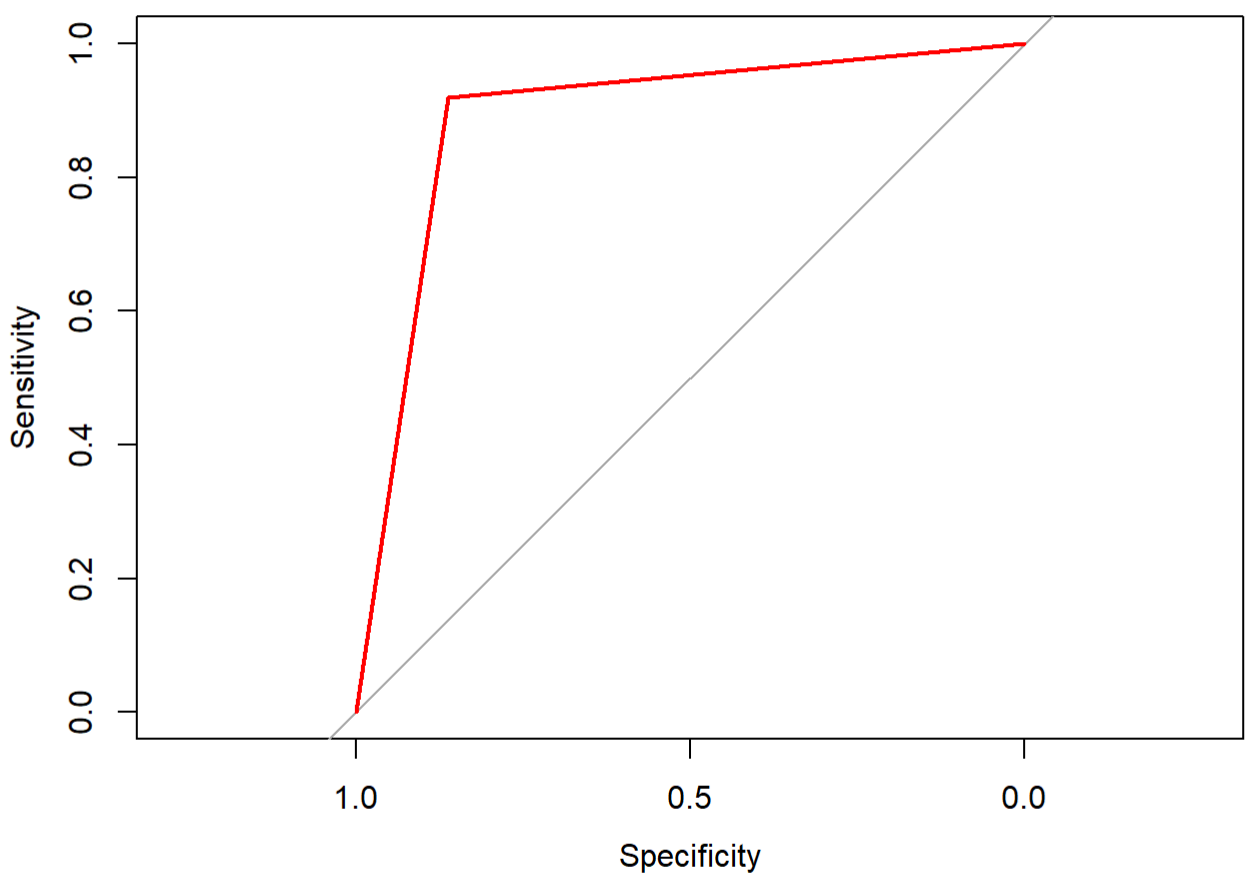
```
## [1] 0.9101678
```

```r
library(pROC)
roc_svm2_test <- roc(as.numeric(customers.test$Attrition_Flag), as.numeric(pred_test2))
```

```
## Setting levels: control = 1, case = 2
```

```
## Setting direction: controls < cases
```

```r
plot(roc_svm2_test,col = "red")#draw the roc plots of the three model
```



```r
roc_svm2_test$auc
```

```
## Area under the curve: 0.8905
```

```r
#tune the parameter of polynomial kernel
library(e1071)
set.seed(123)
tune.out3<-tune(svm,Attrition_Flag~.,data=customers.smoted2, kernel='polynomial',ranges=list(cos
t=c(0.01, 0.1, 1, 10, 100),degree=c(1,2,3,4,5,6)))
summary(tune.out3)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost degree
##   100      5
##
## - best performance: 0.05918348
##
## - Detailed performance results:
##     cost degree      error  dispersion
## 1  1e-02      1 0.18471488 0.023039775
## 2  1e-01      1 0.14990575 0.018611416
## 3  1e+00      1 0.14120583 0.021154742
## 4  1e+01      1 0.13728900 0.017564247
## 5  1e+02      1 0.13685469 0.017568854
## 6  1e-02      2 0.44362129 0.112910097
## 7  1e-01      2 0.14837738 0.021206811
## 8  1e+00      2 0.10334991 0.010763925
## 9  1e+01      2 0.09181444 0.008517750
## 10 1e+02      2 0.08332386 0.013511852
## 11 1e-02      3 0.42864355 0.091762329
## 12 1e-01      3 0.18210666 0.031175908
## 13 1e+00      3 0.09725585 0.009540156
## 14 1e+01      3 0.06875107 0.008182124
## 15 1e+02      3 0.06679454 0.010691271
## 16 1e-02      4 0.50369234 0.028380752
## 17 1e-01      4 0.43080184 0.025062924
## 18 1e+00      4 0.16035190 0.021488454
## 19 1e+01      4 0.06745145 0.010224208
## 20 1e+02      4 0.06396704 0.012010377
## 21 1e-02      5 0.50739699 0.018151557
## 22 1e-01      5 0.43841622 0.026577608
## 23 1e+00      5 0.25412617 0.028026608
## 24 1e+01      5 0.07811310 0.012842656
## 25 1e+02      5 0.05918348 0.011377770
## 26 1e-02      6 0.51000900 0.015471129
## 27 1e-01      6 0.46039358 0.026642424
## 28 1e+00      6 0.37966847 0.021224896
## 29 1e+01      6 0.19778015 0.020609944
## 30 1e+02      6 0.06027233 0.009482694
```

```
svm_fit3<-svm(Attrition_Flag~.,data=customers.smoted2,kernel='polynomial',cost=100,degree=5)
#make predictions, show confusion matrix and accuracy
pred_test3<-predict(svm_fit3,customers.test)
table(true=customers.test$Attrition_Flag,predict=pred_test3)
```

```
##      predict
## true    0    1
##    0  415   63
##    1  265 2296
```

```
1-mean(pred_test3!= customers.test$Attrition_Flag)
```

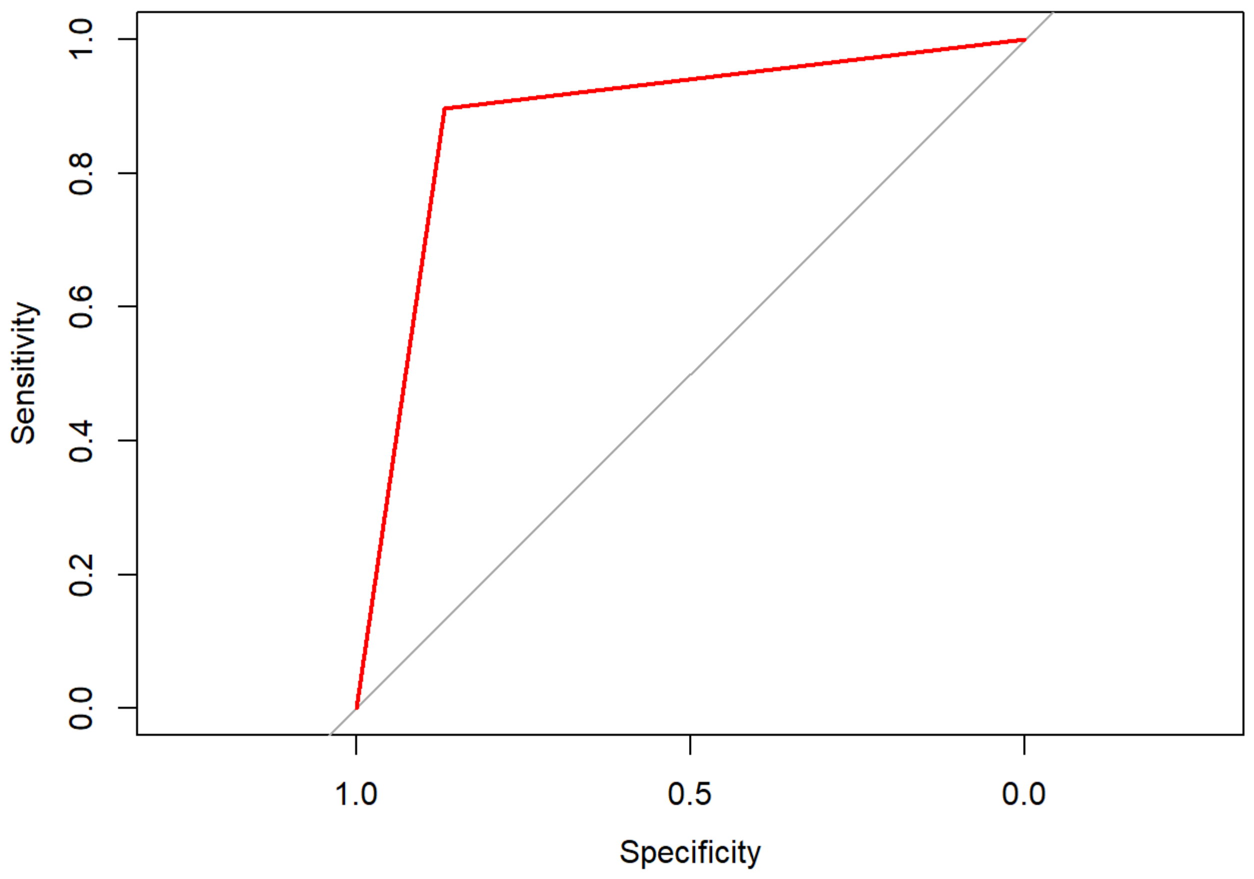
```
## [1] 0.8920698
```

```
library(pROC)
roc_svm3_test <- roc(as.numeric(customers.test$Attrition_Flag), as.numeric(pred_test3))
```

```
## Setting levels: control = 1, case = 2
```

```
## Setting direction: controls < cases
```

```
plot(roc_svm3_test,col = "red")#draw the roc plots of the three model
```



```
roc_svm3_test$auc
```

```
## Area under the curve: 0.8824
```

Per above result, radial SVM with gemma = 0.1 and cost = has the highest test accuracy of 91%, and highest test AUC 0.8905.

Random Forest

```
#RF
library(ranger)
```

```
## Warning: package 'ranger' was built under R version 4.1.2
```

```
n_features <- length(setdiff(names(customers.smoted1),"Attrition_Flag"))
hyper_grid <- expand.grid(
  mtry = floor(n_features * c(.05,.15,.25,.333,.4)),
  min.node.size = c(1,3,5,10),
  replace=c(TRUE,FALSE),
  sample.fraction = c(.5,.63,.8),
  rmse = NA
)
#excecute full cartesian grid search
for(i in seq_len(nrow(hyper_grid))) {
  #fit model for ith hyperparameter combination
  fit <- ranger(
    formula = Attrition_Flag ~.,
    data = customers.smoted1,
    num.trees = n_features * 10,
    mtry = hyper_grid$mtry[i],
    min.node.size = hyper_grid$min.node.size[i],
    replace = hyper_grid$replace[i],
    sample.fraction = hyper_grid$sample.fraction[i],
    verbose = FALSE,
    seed = 233,
    respect.unordered.factors = "order",
  )
  #export OOB error
  hyper_grid$rmse[i] <- sqrt(fit$prediction.error)
}
```

```
#model without tuning as baseline
ames_rf1 <- ranger(
  Attrition_Flag ~.,
  data = customers.smoted1,
  mtry = floor(n_features / 3),  #default set rule
  respect.unordered.factors = "order",
  seed = 123
)
#get OOB RMSE is 0.1067394
(default_rmse <- sqrt(ames_rf1$prediction.error))
```

```
## [1] 0.1104645
```

```
#assess top 10 models
hyper_grid %>%
  arrange(rmse) %>%
  mutate(perc_gain = (default_rmse - rmse) / default_rmse * 100) %>%
  head(10)
```

	mtry <dbl>	min.node.size <dbl>	replace <lgl>	sample.fraction <dbl>	rmse <dbl>	perc_gain <dbl>
1	3	1	FALSE	0.63	0.1069319	3.1979888
2	3	3	FALSE	0.80	0.1085161	1.7638068
3	6	1	FALSE	0.80	0.1096893	0.7017631
4	3	1	FALSE	0.80	0.1108501	-0.3490412
5	5	5	FALSE	0.80	0.1112343	-0.6968725
6	5	1	TRUE	0.80	0.1116172	-1.0435065
7	5	3	FALSE	0.80	0.1116172	-1.0435065
8	5	3	FALSE	0.63	0.1119988	-1.3889554
9	3	1	TRUE	0.80	0.1123791	-1.7332313
10	3	5	FALSE	0.80	0.1123791	-1.7332313

1-10 of 10 rows

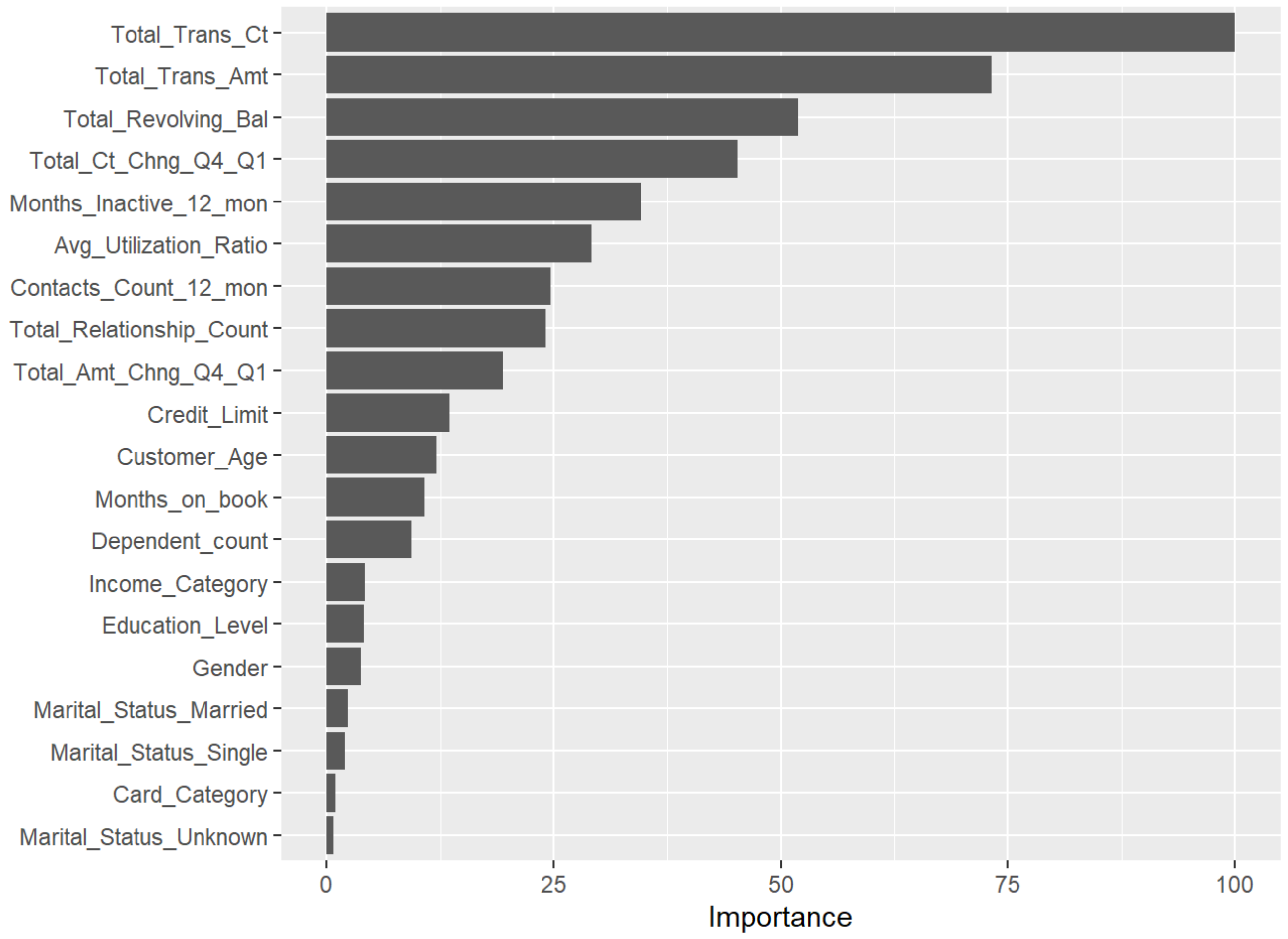
```
#so the best tuning parameter is mtry = 3(near the number of features/3) & min.node.size = 1 (th
e less deep tree performs better)& replace = FALSE & sample.fraction = 0.63

#feature interpretetion
#rerun model with impurity-based variable importance
library(vip)
```

```
## Warning: package 'vip' was built under R version 4.1.2
```

```
##
## Attaching package: 'vip'
```

```
## The following object is masked from 'package:utils':
##
##     vi
```

```
library(ranger)
rf_impurity <- ranger(
    formula = Attrition_Flag ~.,
    data = customers.smoted1,
    num.trees = n_features * 10,
    mtry = 3,
    min.node.size = 1,
    replace = FALSE,
    importance = "impurity", #use impurity approach to estimate importance
    sample.fraction = 0.63,
    verbose = FALSE,
    seed = 233,
    respect.unordered.factors = "order",
  )
p_rf <- vip::vip(rf_impurity, num_features = n_features, scale=TRUE)
gridExtra::grid.arrange(p_rf)
```



```
#according to the importance, the most important predictors are Total_Trans_Amt.......
```

```
library(caret)
library(vip)
library(ranger)
rf_impurity <- ranger(
    formula = Attrition_Flag ~.,
    data = customers.smoted1,
    num.trees = n_features * 10,
    mtry = 3,
    min.node.size = 1,
    replace = FALSE,
    importance = "impurity", #use impurity approach to estimate importance
    sample.fraction = 0.63,
    verbose = FALSE,
    seed = 233,
    respect.unordered.factors = "order",
  )
```

```
#training accuracy
pred_rf_train<- predict(rf_impurity, data = customers.smoted1,type='response', verbose = FALSE)
$predictions
confusionMatrix(pred_rf_train,customers.smoted1$Attrition_Flag)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    0    1
##          0 5745    0
##          1    0 5974
##                                       
##                Accuracy : 1           
##                  95% CI : (0.9997, 1) 
##     No Information Rate : 0.5098      
##     P-Value [Acc > NIR] : < 2.2e-16   
##                                       
##                   Kappa : 1           
##                                       
##  Mcnemar's Test P-Value : NA          
##                                       
##             Sensitivity : 1.0000      
##             Specificity : 1.0000      
##          Pos Pred Value : 1.0000      
##          Neg Pred Value : 1.0000      
##              Prevalence : 0.4902      
##          Detection Rate : 0.4902      
##    Detection Prevalence : 0.4902      
##       Balanced Accuracy : 1.0000      
##                                       
##        'Positive' Class : 0           
## 
```

```
#testing accuracy
pred_rf_test<- predict(rf_impurity, data = customers.test,type='response', verbose = FALSE)$pred
ictions
confusionMatrix(pred_rf_test,customers.test$Attrition_Flag)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    0    1
##          0  429   94
##          1   49 2467
##                                           
##                Accuracy : 0.9529          
##                  95% CI : (0.9448, 0.9602)
##     No Information Rate : 0.8427          
##     P-Value [Acc > NIR] : < 2.2e-16       
##                                           
##                   Kappa : 0.829           
##                                           
##  Mcnemar's Test P-Value : 0.0002337       
##                                           
##             Sensitivity : 0.8975          
##             Specificity : 0.9633          
##          Pos Pred Value : 0.8203          
##          Neg Pred Value : 0.9805          
##              Prevalence : 0.1573          
##          Detection Rate : 0.1412          
##    Detection Prevalence : 0.1721          
##       Balanced Accuracy : 0.9304          
##                                           
##        'Positive' Class : 0               
## 
```

Basic GBM

```
#Basic GBM
library(gbm)
```

```
## Warning: package 'gbm' was built under R version 4.1.2
```

```
## Loaded gbm 2.1.8
```

```
#tuning parameters
#create grid search to seek the optimal learning rate or shrinkage.
hyper_grid2 <- expand.grid(
  learning_rate = c(0.3,0.1,0.05,0.01,0.005),
  RMSE= NA,
  trees=NA
)
#execute grid search
for (i in seq_len(nrow(hyper_grid2))){
  #fix gbm
  set.seed(233)
  m <- gbm(
    formula = as.character(Attrition_Flag) ~.,
    data = customers.smoted1,
    distribution = "bernoulli",
    n.trees = 5000,
    shrinkage = hyper_grid2$learning_rate[i],
    interaction.depth = 3,
    n.minobsinnode = 10,
    cv.folds = 10
  )

  #add SSE, trees
  hyper_grid2$RMSE[i] <- sqrt(min(m$cv.error))
  hyper_grid2$trees[i] <- which.min(m$cv.error)
}
```

```
#results and choose optimal learning rate parameters
arrange(hyper_grid2,RMSE)
```

learning_rate <dbl>	RMSE <dbl>	trees <int>
0.050	0.2543645	4876
0.100	0.2591907	2908
0.300	0.2702669	1078
0.010	0.3369709	5000
0.005	0.3997746	5000

5 rows

```
#so, the optimal learning rate is 0.05. And requires 4876 trees.
```

```
#under this learning rate, create grid search to seek the optimal number of trees, interaction d
epth and number of minimum node.
hyper_grid3 <- expand.grid(
  n.trees = 4876,
  shrinkage = 0.05,
  interaction.depth = c(3,5,7),
  n.minobsinnode = c(5,10,15)
)
```

```
#create model fit function, take 4 GBM parameters input and return RMSE
model_fit <- function(n.trees, shrinkage, interaction.depth, n.minobsinnode){
  set.seed(233)
  m<-gbm(
    formula = as.character(Attrition_Flag) ~.,
    data = customers.smoted1,
    distribution = "bernoulli",
    n.trees = n.trees,
    shrinkage = shrinkage,
    interaction.depth = interaction.depth,
    n.minobsinnode = n.minobsinnode,
    cv.folds = 10
  )
  #compute RMSE
  sqrt(min(m$cv.error))
}
#perform search grid with functional programming
hyper_grid3$rmse <- purrr::pmap_dbl(
  hyper_grid3,
  ~model_fit(
    n.trees = ..1,
    shrinkage = ..2,
    interaction.depth = ..3,
    n.minobsinnode = ..4
  )
)

#display the results and choose the optimal depth and minnode parameters.
arrange(hyper_grid3,rmse)
```

n.trees <dbl>	shrinkage <dbl>	interaction.depth <dbl>	n.minobsinnode <dbl>	rmse <dbl>
4876	0.05	7	15	0.2429540
4876	0.05	7	10	0.2475343
4876	0.05	5	15	0.2480404
4876	0.05	7	5	0.2498449
4876	0.05	5	10	0.2506239

n.trees	shrinkage	interaction.depth	n.minobsinnode	rmse
<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
4876	0.05	5	5	0.2512152
4876	0.05	3	10	0.2543645
4876	0.05	3	15	0.2545519
4876	0.05	3	5	0.2564295

9 rows

```
#so the optimal interaction.depth = 7, n.minobsinnode = 15, n.trees = 4876,rmse =0.2441435
```

```
library(gbm)
bgbm <- gbm(
    formula = as.character(Attrition_Flag) ~.,
    data = customers.smoted1,
    distribution = "bernoulli",
    n.trees = 4876,
    shrinkage = 0.05,
    interaction.depth = 7,
    n.minobsinnode = 15,
    cv.folds = 10
  )
```

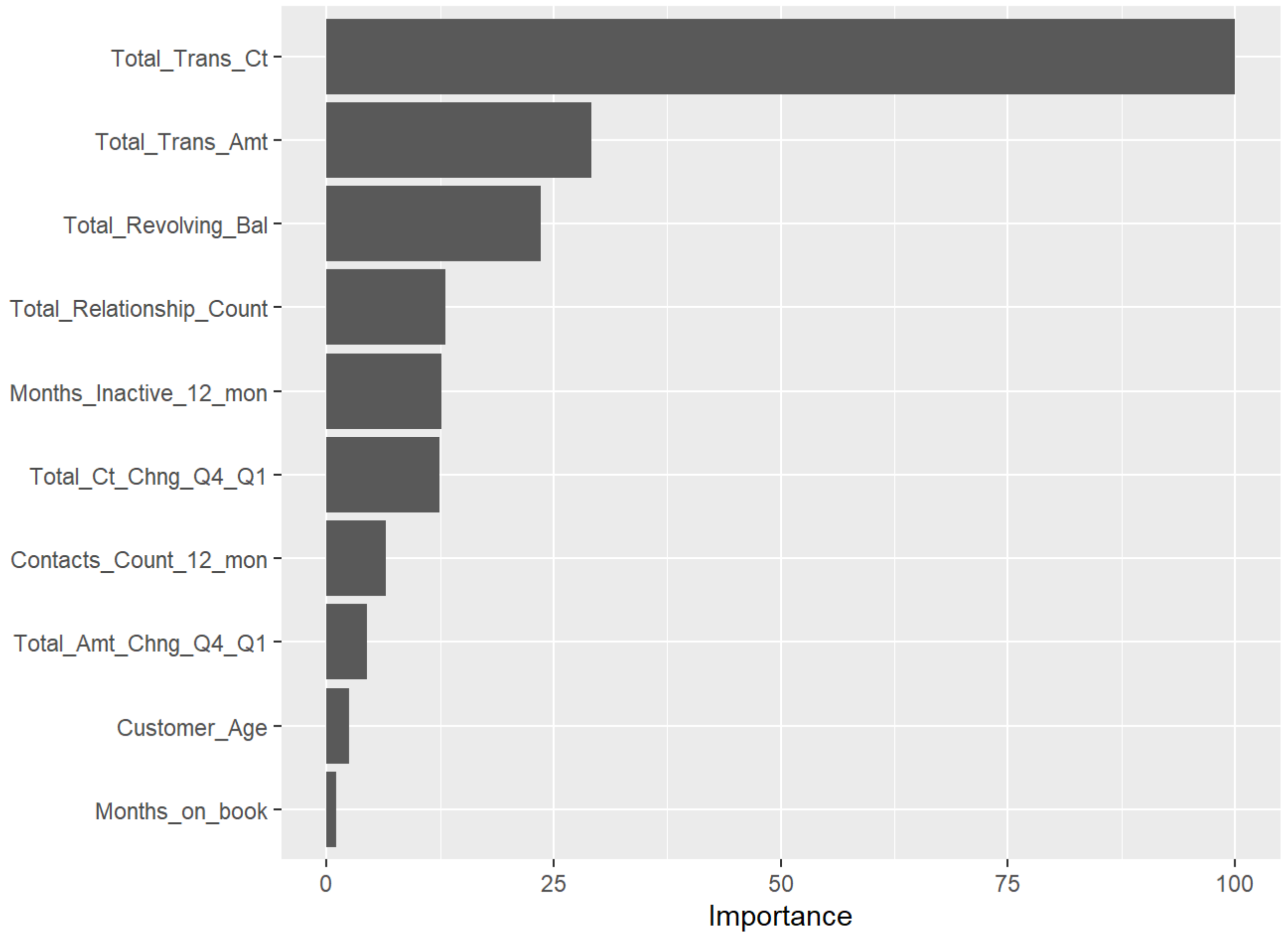
```
#training accuracy
pred_bgbm_train<- predict(bgbm, newdata = customers.smoted1,verbose = FALSE)
```

```
## Using 2662 trees...
```

```
pred_bgbm_train<-ifelse(pred_bgbm_train<0.5,0,1)
library(caret)
confusionMatrix(factor(pred_bgbm_train),customers.smoted1$Attrition_Flag)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    0    1
##          0 5745    0
##          1    0 5974
##                                      
##                Accuracy : 1          
##                  95% CI : (0.9997, 1)
##     No Information Rate : 0.5098     
##     P-Value [Acc > NIR] : < 2.2e-16  
##                                      
##                   Kappa : 1          
##                                      
##  Mcnemar's Test P-Value : NA         
##                                      
##             Sensitivity : 1.0000     
##             Specificity : 1.0000     
##          Pos Pred Value : 1.0000     
##          Neg Pred Value : 1.0000     
##              Prevalence : 0.4902     
##          Detection Rate : 0.4902     
##    Detection Prevalence : 0.4902     
##       Balanced Accuracy : 1.0000     
##                                      
##        'Positive' Class : 0          
## 
```

```
p_bgbm <- vip::vip(bgbm, scale=TRUE)
gridExtra::grid.arrange(p_bgbm)
```



```
#most important : Total_Trans_Ct , Total_Trans_Amt, Total_Revolving_Bal...
```

```
#testing gbm
pred_bgbm_test<- predict(bgbm, newdata = customers.test,verbose = FALSE)
```

```
## Using 2662 trees...
```

```
pred_bgbm_test<-ifelse(pred_bgbm_test<0.5,0,1)
library(caret)
confusionMatrix(factor(pred_bgbm_test),customers.test$Attrition_Flag)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0  451   67
##          1   27 2494
##
##                Accuracy : 0.9691
##                  95% CI : (0.9623, 0.9749)
##     No Information Rate : 0.8427
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8872
##
##  Mcnemar's Test P-Value : 5.757e-05
##
##             Sensitivity : 0.9435
##             Specificity : 0.9738
##          Pos Pred Value : 0.8707
##          Neg Pred Value : 0.9893
##              Prevalence : 0.1573
##          Detection Rate : 0.1484
##    Detection Prevalence : 0.1705
##       Balanced Accuracy : 0.9587
##
##        'Positive' Class : 0
##
```

Stochastic GBM

```
library(gbm)
library(caret)
library(purrr)
```

```
## Warning: package 'purrr' was built under R version 4.1.2
```

```
##
## Attaching package: 'purrr'
```

```
## The following object is masked from 'package:caret':
##
##     lift
```

```
## The following objects are masked from 'package:foreach':
##
##     accumulate, when
```

```
#Stochastic GBM
#bag fraction ranging from 0.5-0.8
hyper_grid_sto <- expand.grid(
  n.trees = 4876,
  shrinkage = 0.05,
  bag.fraction = c(0.5,0.6,0.7,0.8),
  interaction.depth = 7,
  n.minobsinnode = 15
)

sto_fit <- function(n.trees, shrinkage, bag.fraction,interaction.depth,n.minobsinnode){
  set.seed(233)
  gbm_sto<-gbm(
    formula = as.character(Attrition_Flag) ~.,
    data = customers.smoted1,
    distribution = "bernoulli",
    n.trees = n.trees,
    shrinkage = shrinkage,
    bag.fraction = bag.fraction,
    interaction.depth = interaction.depth,
    n.minobsinnode = n.minobsinnode,
    cv.folds = 10
  )
  #compute RMSE
  sqrt(min(gbm_sto$cv.error))
}


#perform search grid with functional programming
hyper_grid_sto$rmse <- purrr::pmap_dbl(
  hyper_grid_sto,
  ~sto_fit(
    n.trees = ..1,
    shrinkage = ..2,
    bag.fraction = ..3,
    interaction.depth = ..4,
    n.minobsinnode = ..5
  )
)

#display the results and choose the optimal depth and minnode parameters.
arrange(hyper_grid_sto,rmse)
```

n.trees <dbl>	shrinkage <dbl>	bag.fraction <dbl>	interaction.depth <dbl>	n.minobsinnode <dbl>	rmse <dbl>
4876	0.05	0.5	7	15	0.2429540
4876	0.05	0.7	7	15	0.2451082
4876	0.05	0.8	7	15	0.2451941

n.trees <dbl>	shrinkage <dbl>	bag.fraction <dbl>	interaction.depth <dbl>	n.minobsinnode <dbl>	rmse <dbl>
4876	0.05	0.6	7	15	0.2452240

4 rows

```
library(gbm)
library(caret)
library(purrr)
sgbm <- gbm(
    formula = as.character(Attrition_Flag) ~.,
    data = customers.smoted1,
    distribution = "bernoulli",
    n.trees = 4876,
    shrinkage = 0.05,
    interaction.depth = 7,
    n.minobsinnode = 15,
    cv.folds = 10,
    bag.fraction = 0.5
  )
```

```
#training accuracy
pred_sgbm_train<- predict(sgbm, newdata = customers.smoted1,type='response',verbose = FALSE)
```

```
## Using 2662 trees...
```

```
pred_sgbm_train<-ifelse(pred_sgbm_train<0.5,0,1)
library(caret)
confusionMatrix(factor(pred_sgbm_train),customers.smoted1$Attrition_Flag)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    0    1
##          0 5745    0
##          1    0 5974
##                                      
##                Accuracy : 1          
##                  95% CI : (0.9997, 1)
##     No Information Rate : 0.5098     
##     P-Value [Acc > NIR] : < 2.2e-16  
##                                      
##                   Kappa : 1          
##                                      
##  Mcnemar's Test P-Value : NA         
##                                      
##             Sensitivity : 1.0000     
##             Specificity : 1.0000     
##          Pos Pred Value : 1.0000     
##          Neg Pred Value : 1.0000     
##              Prevalence : 0.4902     
##          Detection Rate : 0.4902     
##    Detection Prevalence : 0.4902     
##       Balanced Accuracy : 1.0000     
##                                      
##        'Positive' Class : 0          
## 
```

```
#testing accuracy
pred_sgbm_test<- predict(sgbm, newdata = customers.test,type='response',verbose = FALSE)
```

```
## Using 2662 trees...
```

```
pred_sgbm_test<-ifelse(pred_sgbm_test<0.5,0,1)
library(caret)
confusionMatrix(factor(pred_sgbm_test),customers.test$Attrition_Flag)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    0    1
##          0  445   54
##          1   33 2507
##                                           
##                Accuracy : 0.9714          
##                  95% CI : (0.9648, 0.977)
##     No Information Rate : 0.8427          
##     P-Value [Acc > NIR] : < 2e-16         
##                                           
##                   Kappa : 0.8939          
##                                           
##  Mcnemar's Test P-Value : 0.03201         
##                                           
##             Sensitivity : 0.9310          
##             Specificity : 0.9789          
##          Pos Pred Value : 0.8918          
##          Neg Pred Value : 0.9870          
##              Prevalence : 0.1573          
##          Detection Rate : 0.1464          
##    Detection Prevalence : 0.1642          
##       Balanced Accuracy : 0.9549          
##                                           
##        'Positive' Class : 0               
## 
```

```
library("RColorBrewer")
```

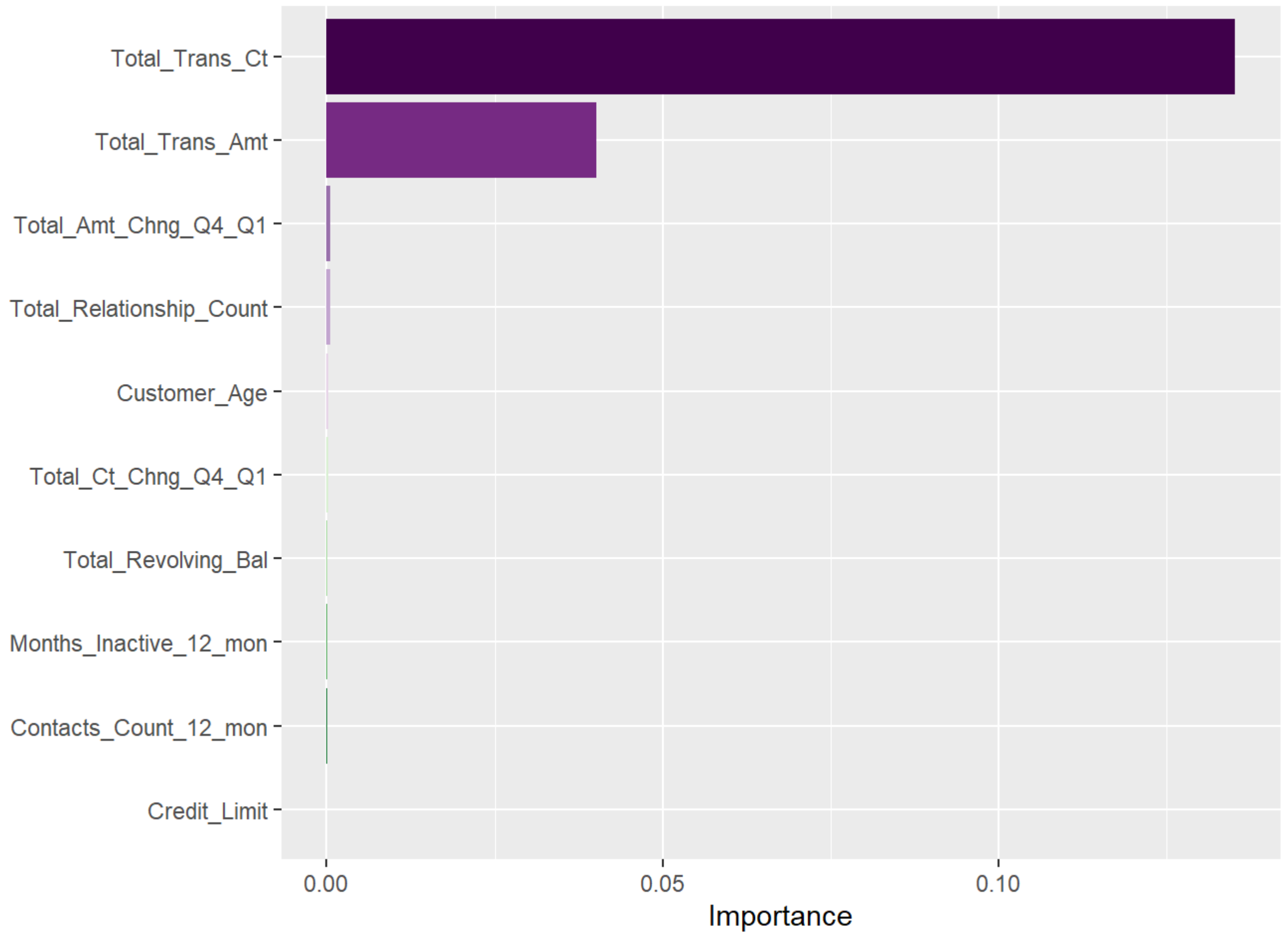
```
## Warning: package 'RColorBrewer' was built under R version 4.1.1
```

```
set.seed(233)
pred_fun = function(X.model, newdata) {
  predict(X.model, newdata)
}
p_sgbm <- vip::vip(object = sgbm, method = "permute", target = "Attrition_Flag", metric = "auc",
pred_wrapper = pred_fun, newdata=data.matrix(customers.smoted1[,-18]),reference_class = 1, aesth
etics = list(fill = brewer.pal(n=10,name = "PRGn")[1:10],border=NA) ,all_permutations = TRUE, ji
tter = TRUE)
```

```
## Using 2662 trees...
##
## Using 2662 trees...
##
## Using 2662 trees...
##
## Using 2662 trees...
##
## Using 2662 trees...
##
## Using 2662 trees...
##
## Using 2662 trees...
##
## Using 2662 trees...
##
## Using 2662 trees...
##
## Using 2662 trees...
##
## Using 2662 trees...
##
## Using 2662 trees...
##
## Using 2662 trees...
##
## Using 2662 trees...
##
## Using 2662 trees...
##
## Using 2662 trees...
##
## Using 2662 trees...
##
## Using 2662 trees...
##
## Using 2662 trees...
##
## Using 2662 trees...
##
## Using 2662 trees...
```

```
## Warning: Ignoring unknown parameters: border
```

```
gridExtra::grid.arrange(p_sgbm)
```



```
#pdp
library(dplyr)
library(pdp)
```

```
## Warning: package 'pdp' was built under R version 4.1.2
```

```
##
## Attaching package: 'pdp'
```

```
## The following object is masked from 'package:purrr':
##
##     partial
```

```
library(ggplot2)
partial(sgbm,pred.var = "Total_Trans_Ct", train = customers.smoted1[,-18],n.trees=4876) %>% auto
plot(size = 0.8) + geom_point(colour = brewer.pal(n=20,"PRGn")[3],size = 2.5)+scale_x_continuous
(breaks=seq(-2.5,3.5,0.5))+ theme_bw()+ geom_smooth()
```

```
## Warning in brewer.pal(n = 20, "PRGn"): n too large, allowed maximum for palette PRGn is 11
## Returning the palette you asked for with that many colors
## Warning: Use of `object[[1L]]` is discouraged. Use `.data[[1L]]` instead.
```

```
## Warning: Use of `object[["yhat"]]` is discouraged. Use `.data[["yhat"]]`
## instead.
```

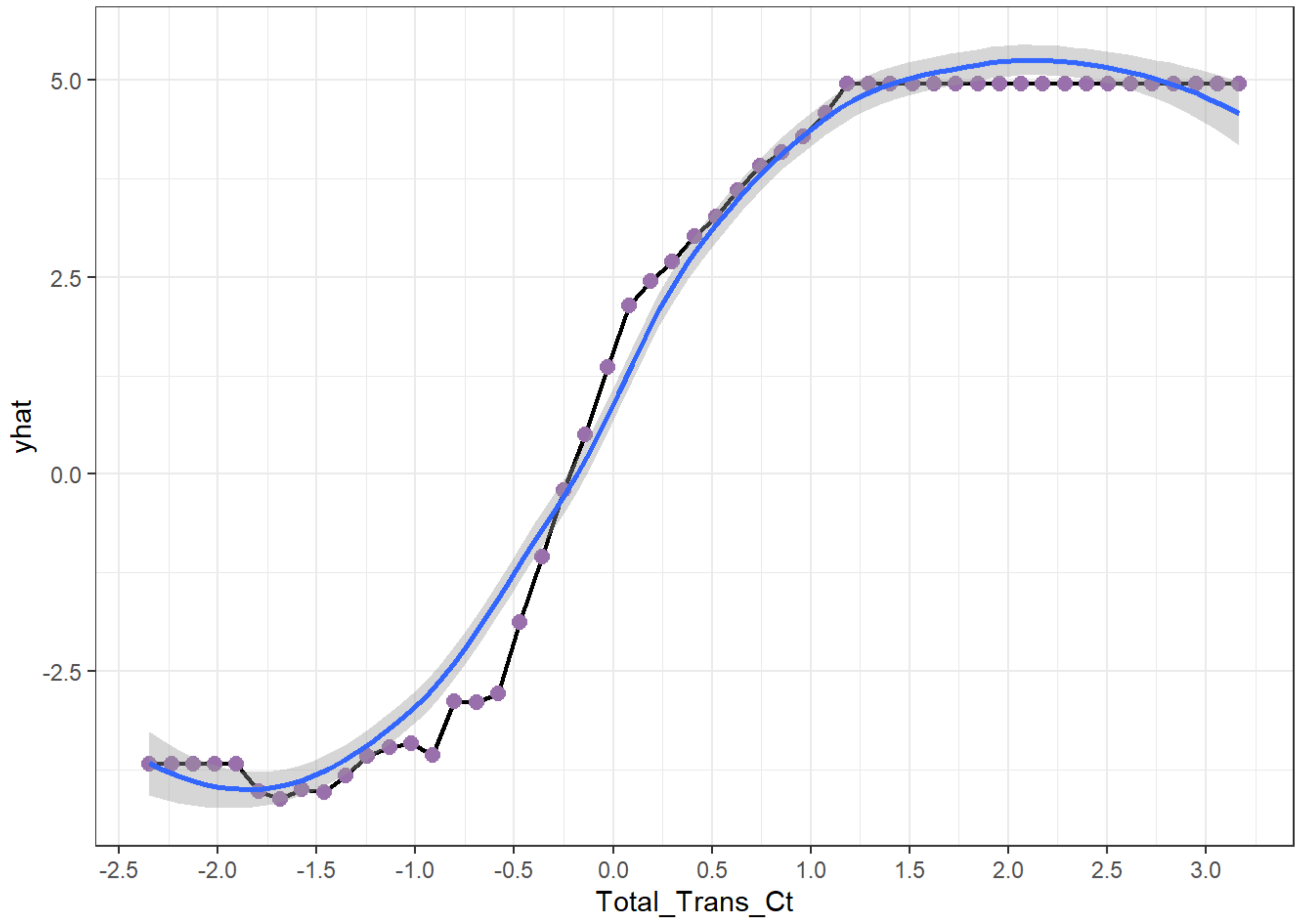
```
## Warning: Use of `object[[1L]]` is discouraged. Use `.data[[1L]]` instead.
```

```
## Warning: Use of `object[["yhat"]]` is discouraged. Use `.data[["yhat"]]`
## instead.
```

```
## Warning: Use of `object[[1L]]` is discouraged. Use `.data[[1L]]` instead.
```

```
## Warning: Use of `object[["yhat"]]` is discouraged. Use `.data[["yhat"]]`
## instead.
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



```
partial(sgbm,pred.var = "Total_Trans_Amt", train = customers.smoted1[,-18],n.trees=4876) %>% aut
oplot(size = 0.8)+ geom_point(colour = brewer.pal(n=20,"PRGn")[3],size = 2.5)+scale_x_continuous
(breaks=seq(-1.5,4,0.5))+theme_bw()+ geom_smooth()
```

```
## Warning in brewer.pal(n = 20, "PRGn"): n too large, allowed maximum for palette PRGn is 11
## Returning the palette you asked for with that many colors
```

```
## Warning: Use of `object[[1L]]` is discouraged. Use `.data[[1L]]` instead.
```

```
## Warning: Use of `object[["yhat"]]` is discouraged. Use `.data[["yhat"]]`
## instead.
```

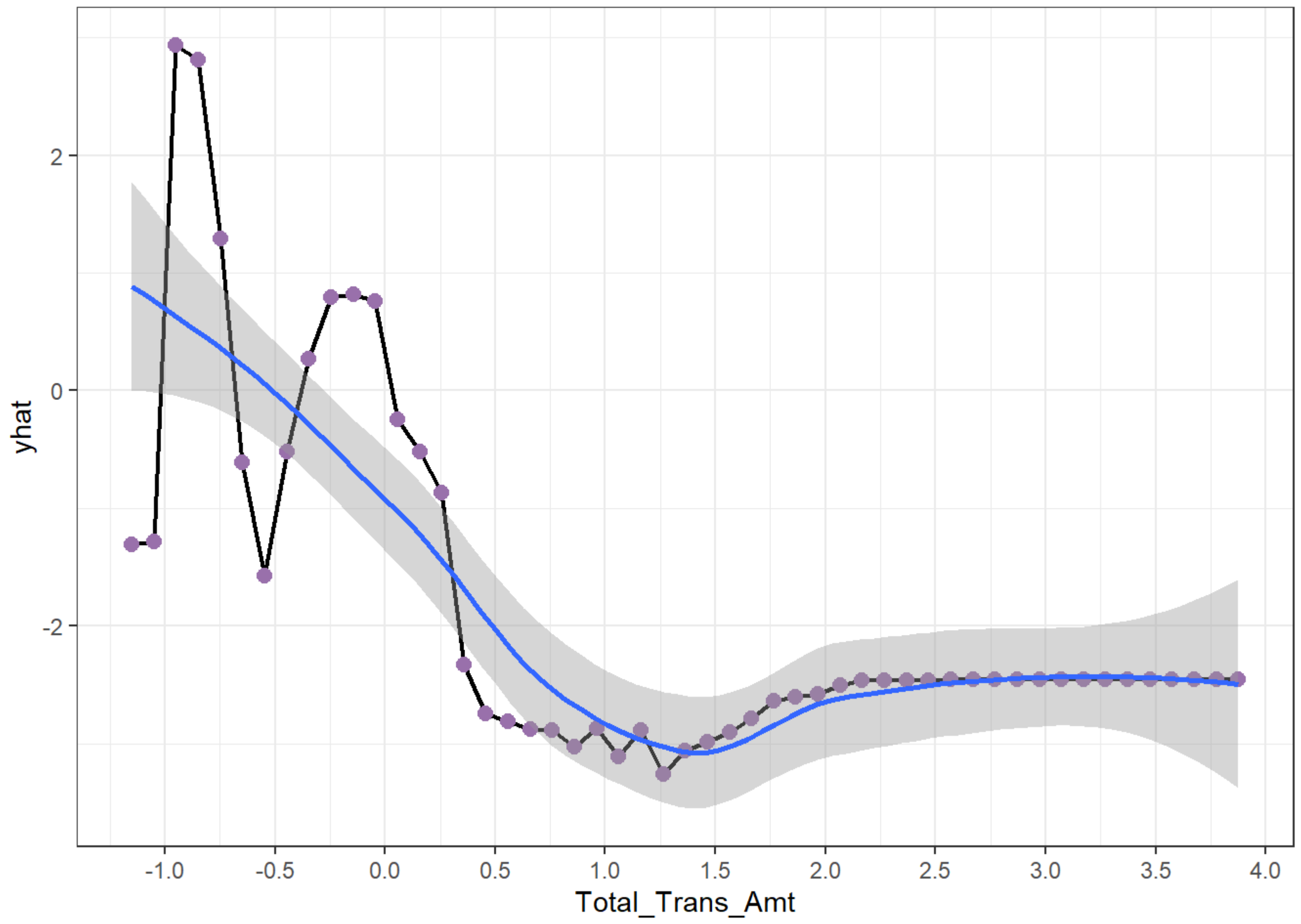
```
## Warning: Use of `object[[1L]]` is discouraged. Use `.data[[1L]]` instead.
```

```
## Warning: Use of `object[["yhat"]]` is discouraged. Use `.data[["yhat"]]`
## instead.
```

```
## Warning: Use of `object[[1L]]` is discouraged. Use `.data[[1L]]` instead.
```

```
## Warning: Use of `object[["yhat"]]` is discouraged. Use `.data[["yhat"]]`
## instead.
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



XGBOOST

```
#XGBoost (based on basic GBM)
library(xgboost)
```

```
## Warning: package 'xgboost' was built under R version 4.1.2
```

```
##
## Attaching package: 'xgboost'
```

```
## The following object is masked from 'package:dplyr':
##
##     slice
```

```
x <- data.matrix(customers.smoted1[setdiff(names(customers.smoted1),"Attrition_Flag")])
y <- data.matrix(customers.smoted1$Attrition_Flag)
dtrain <- xgb.DMatrix(x,label = y)
```

```
set.seed(233)
ames_xgb <- xgb.cv(
  data = dtrain,
  nrounds = 4876,
  objective = "binary:logistic",
  early_stopping_rounds = 50,
  nfold = 10,
  params = list(
    eta = 0.05,
    max_depth = 7,
    min_child_weight = 15,
    subsample = 0.5,
    colsample_bytree = 0.5),
  verbose = 0
)
```

```
## [20:12:21] WARNING: amalgamation/../src/learner.cc:1115: Starting in XGBoost 1.3.0, the defau
lt evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logl
oss'. Explicitly set eval_metric if you'd like to restore the old behavior.
## [20:12:21] WARNING: amalgamation/../src/learner.cc:1115: Starting in XGBoost 1.3.0, the defau
lt evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logl
oss'. Explicitly set eval_metric if you'd like to restore the old behavior.
## [20:12:21] WARNING: amalgamation/../src/learner.cc:1115: Starting in XGBoost 1.3.0, the defau
lt evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logl
oss'. Explicitly set eval_metric if you'd like to restore the old behavior.
## [20:12:21] WARNING: amalgamation/../src/learner.cc:1115: Starting in XGBoost 1.3.0, the defau
lt evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logl
oss'. Explicitly set eval_metric if you'd like to restore the old behavior.
## [20:12:21] WARNING: amalgamation/../src/learner.cc:1115: Starting in XGBoost 1.3.0, the defau
lt evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logl
oss'. Explicitly set eval_metric if you'd like to restore the old behavior.
## [20:12:21] WARNING: amalgamation/../src/learner.cc:1115: Starting in XGBoost 1.3.0, the defau
lt evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logl
oss'. Explicitly set eval_metric if you'd like to restore the old behavior.
## [20:12:21] WARNING: amalgamation/../src/learner.cc:1115: Starting in XGBoost 1.3.0, the defau
lt evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logl
oss'. Explicitly set eval_metric if you'd like to restore the old behavior.
## [20:12:21] WARNING: amalgamation/../src/learner.cc:1115: Starting in XGBoost 1.3.0, the defau
lt evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logl
oss'. Explicitly set eval_metric if you'd like to restore the old behavior.
## [20:12:21] WARNING: amalgamation/../src/learner.cc:1115: Starting in XGBoost 1.3.0, the defau
lt evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logl
oss'. Explicitly set eval_metric if you'd like to restore the old behavior.
## [20:12:21] WARNING: amalgamation/../src/learner.cc:1115: Starting in XGBoost 1.3.0, the defau
lt evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logl
oss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

```
#minimum test cv RMSE
min(ames_xgb$evaluation_log$test_logloss_mean)
```

```
## [1] 0.0455307
```

```
#hyperparameter grid
hyper_grid4 <- expand.grid(
  eta = 0.05,
  max_depth = 7,
  min_child_weight = 15,
  subsample = 0.5,
  colsample_bytree = 0.5,
  gamma = c(0,1,10,100),
  lambda = c(0,1e-2,0.1,1,100),
  alpha = c(0,1e-2,0.1,1,100),
  rmse = 0,
  trees = 0
)
```

```
#grid search
for(i in seq_len(nrow(hyper_grid4))) {
  set.seed(233)
  m <- xgb.cv(
    data = x,
    label = y,
    nrounds = 4876,
    objective = "binary:logistic",
    early_stopping_rounds = 50,
    nfold = 10,
    verbose = 0,
    params = list(
      eta = hyper_grid4$eta[i],
      max_depth = hyper_grid4$max_depth[i],
      min_child_weight = hyper_grid4$min_child_weight[i],
      subsample = hyper_grid4$subsample[i],
      colsample_bytree = hyper_grid4$colsample_bytree[i],
      gamma = hyper_grid4$gamma[i],
      lambda = hyper_grid4$lambda[i],
      alpha = hyper_grid4$alpha[i]
  )
  )
  hyper_grid4$rmse[i] <- min(m$evaluation_log$test_logloss_mean)
  hyper_grid4$trees[i] <- m$best_iteration
}
```

```
## [22:20:51] WARNING: amalgamation/../src/learner.cc:1115: Starting in XGBoost 1.3.0, the defau
lt evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logl
oss'. Explicitly set eval_metric if you'd like to restore the old behavior.
## [22:20:51] WARNING: amalgamation/../src/learner.cc:1115: Starting in XGBoost 1.3.0, the defau
lt evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logl
oss'. Explicitly set eval_metric if you'd like to restore the old behavior.
## [22:22:15] WARNING: amalgamation/../src/learner.cc:1115: Starting in XGBoost 1.3.0, the defau
lt evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logl
oss'. Explicitly set eval_metric if you'd like to restore the old behavior.
## [22:22:15] WARNING: amalgamation/../src/learner.cc:1115: Starting in XGBoost 1.3.0, the defau
lt evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logl
oss'. Explicitly set eval_metric if you'd like to restore the old behavior.
## [22:22:15] WARNING: amalgamation/../src/learner.cc:1115: Starting in XGBoost 1.3.0, the defau
lt evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logl
oss'. Explicitly set eval_metric if you'd like to restore the old behavior.
## [22:22:15] WARNING: amalgamation/../src/learner.cc:1115: Starting in XGBoost 1.3.0, the defau
lt evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logl
oss'. Explicitly set eval_metric if you'd like to restore the old behavior.
## [22:22:15] WARNING: amalgamation/../src/learner.cc:1115: Starting in XGBoost 1.3.0, the defau
lt evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logl
oss'. Explicitly set eval_metric if you'd like to restore the old behavior.
## [22:22:15] WARNING: amalgamation/../src/learner.cc:1115: Starting in XGBoost 1.3.0, the defau
lt evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logl
oss'. Explicitly set eval_metric if you'd like to restore the old behavior.
## [22:22:15] WARNING: amalgamation/../src/learner.cc:1115: Starting in XGBoost 1.3.0, the defau
lt evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logl
oss'. Explicitly set eval_metric if you'd like to restore the old behavior.
## [22:22:15] WARNING: amalgamation/../src/learner.cc:1115: Starting in XGBoost 1.3.0, the defau
lt evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logl
oss'. Explicitly set eval_metric if you'd like to restore the old behavior.
## [22:22:15] WARNING: amalgamation/../src/learner.cc:1115: Starting in XGBoost 1.3.0, the defau
lt evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logl
oss'. Explicitly set eval_metric if you'd like to restore the old behavior.
## [22:22:15] WARNING: amalgamation/../src/learner.cc:1115: Starting in XGBoost 1.3.0, the defau
lt evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logl
oss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

```
#display the results and choose the optimal depth and minnode parameters.
arrange(hyper_grid4,rmse)
```

eta	max_de…	min_child_weight	subsam…	colsample_bytree	ga…	lam…	alp…	rms
<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl
0.05	7	15	0.5	0.5	0	1e-02	0e+00	0.044645
0.05	7	15	0.5	0.5	0	1e-01	0e+00	0.044817
0.05	7	15	0.5	0.5	0	1e+00	1e-02	0.044877
0.05	7	15	0.5	0.5	0	1e-01	1e-01	0.045038

eta <dbl>	max_de… <dbl>	min_child_weight <dbl>	subsam… <dbl>	colsample_bytree <dbl>	ga… <dbl>	lam… <dbl>	alp… <dbl>	rms <dbl
0.05	7	15	0.5	0.5	0	1e-02	1e-02	0.045111
0.05	7	15	0.5	0.5	0	1e-02	1e-01	0.0454120
0.05	7	15	0.5	0.5	0	1e-01	1e+00	0.045438
0.05	7	15	0.5	0.5	0	0e+00	0e+00	0.045460
0.05	7	15	0.5	0.5	0	1e+00	0e+00	0.045530
0.05	7	15	0.5	0.5	0	1e+00	1e-01	0.045573

1-10 of 100 rows Previous **1** 2 3 4 5 6 ... 10 Next

```
#so the optimal gamma = 0, lambda = 1e+0, alpha = 0e+0, trees = 788, eta = 0.1, max_depth = 7,mi
n_child_weight=5
```

```
library(xgboost)
xgb1 <- xgboost(
    data = x,
    label = y,
    nrounds = 2584,
    objective = "binary:logistic",
    early_stopping_rounds = 50,
    nfold = 10,
    verbose = 0,
    params = list(
      eta = 0.05,
      max_depth = 7,
      min_child_weight = 15,
      subsample = 0.5,
      colsample_bytree = 0.5,
      gamma = 0,
      lambda = 1e-02,
      alpha = 0e+00))
```

```
## [22:22:30] WARNING: amalgamation/../src/learner.cc:576: 
## Parameters: { "nfold" } might not be used.
## 
##   This could be a false alarm, with some parameters getting used by language bindings but
##   then being mistakenly passed down to XGBoost core, or some parameter actually being used
##   but getting flagged wrongly here. Please open an issue if you find any such cases.
## 
## 
## [22:22:30] WARNING: amalgamation/../src/learner.cc:1115: Starting in XGBoost 1.3.0, the defau
lt evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logl
oss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

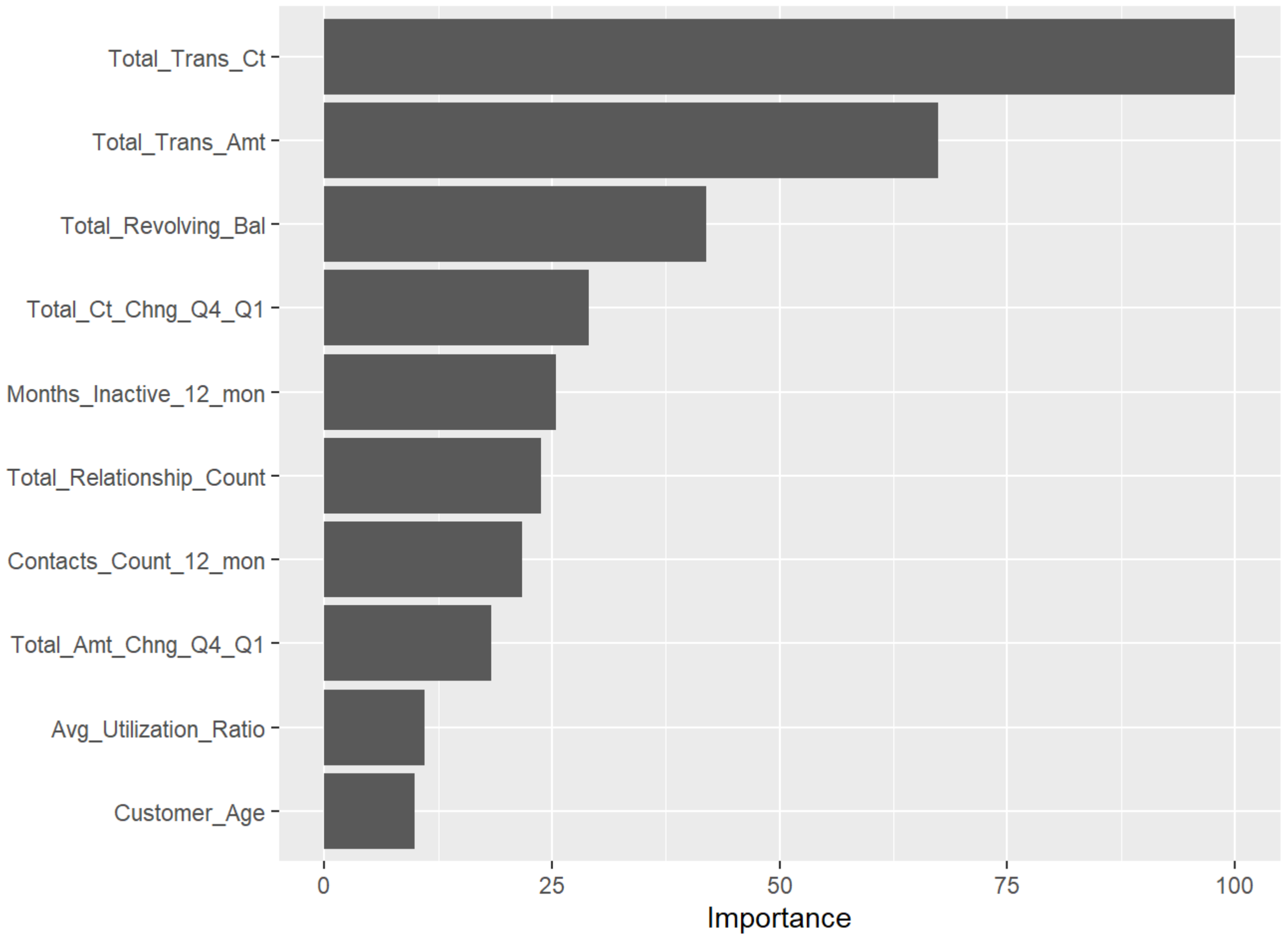
```
#training accuracy
pred_xgb_train<-predict(xgb1, newdata = data.matrix(customers.smoted1[,-18]),type='response',ver
bose = FALSE)
pred_xgb_train<-ifelse(pred_xgb_train<0.5,0,1)
confusionMatrix(factor(pred_xgb_train),customers.smoted1$Attrition_Flag)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    0    1
##          0 5727   11
##          1   18 5963
##                                           
##                Accuracy : 0.9975          
##                  95% CI : (0.9964, 0.9983)
##     No Information Rate : 0.5098          
##     P-Value [Acc > NIR] : <2e-16          
##                                           
##                   Kappa : 0.995           
##                                           
##  Mcnemar's Test P-Value : 0.2652          
##                                           
##             Sensitivity : 0.9969          
##             Specificity : 0.9982          
##          Pos Pred Value : 0.9981          
##          Neg Pred Value : 0.9970          
##              Prevalence : 0.4902          
##          Detection Rate : 0.4887          
##    Detection Prevalence : 0.4896          
##       Balanced Accuracy : 0.9975          
##                                           
##        'Positive' Class : 0               
## 
```

```
#testing accuracy
pred_xgb_test<-predict(xgb1, newdata = data.matrix(customers.test[,-18]),type='response',verbose
= FALSE)
pred_xgb_test<-ifelse(pred_xgb_test<0.5,0,1)
confusionMatrix(factor(pred_xgb_test),customers.test$Attrition_Flag)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0  423   54
##          1   55 2507
##
##                Accuracy : 0.9641
##                  95% CI : (0.9569, 0.9705)
##     No Information Rate : 0.8427
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8646
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.8849
##             Specificity : 0.9789
##          Pos Pred Value : 0.8868
##          Neg Pred Value : 0.9785
##              Prevalence : 0.1573
##          Detection Rate : 0.1392
##    Detection Prevalence : 0.1570
##       Balanced Accuracy : 0.9319
##
##        'Positive' Class : 0
##
```

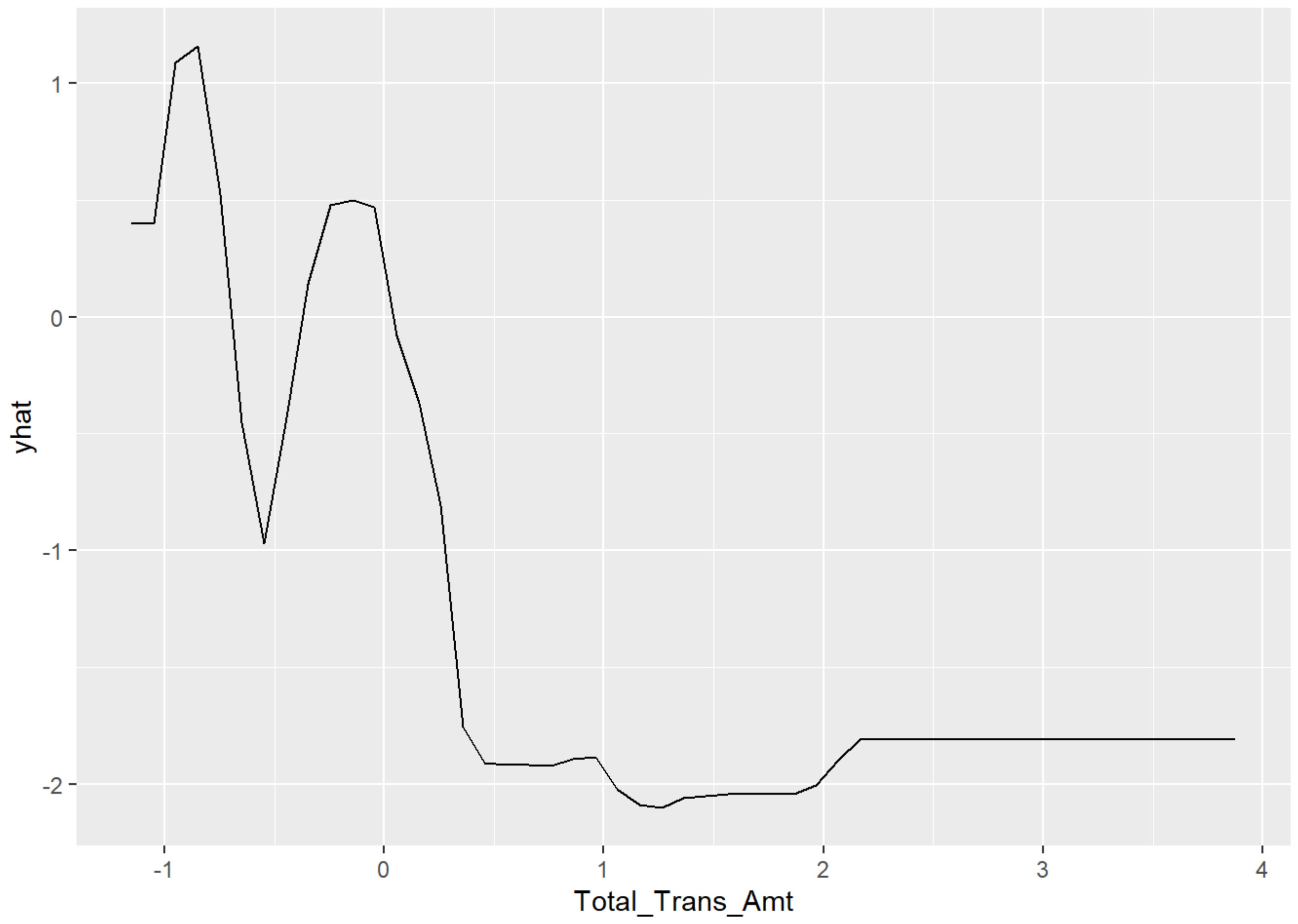
```
#important features
library(caret)
p_xgb <- vip::vip(xgb1, scale=TRUE)
gridExtra::grid.arrange(p_xgb)
```



```
#pdps for xgboost
library(dplyr)
library(pdp)
library(ggplot2)
partial(xgb1,pred.var = "Total_Trans_Amt", train = customers.smoted1[,-18]) %>% autoplot()
```

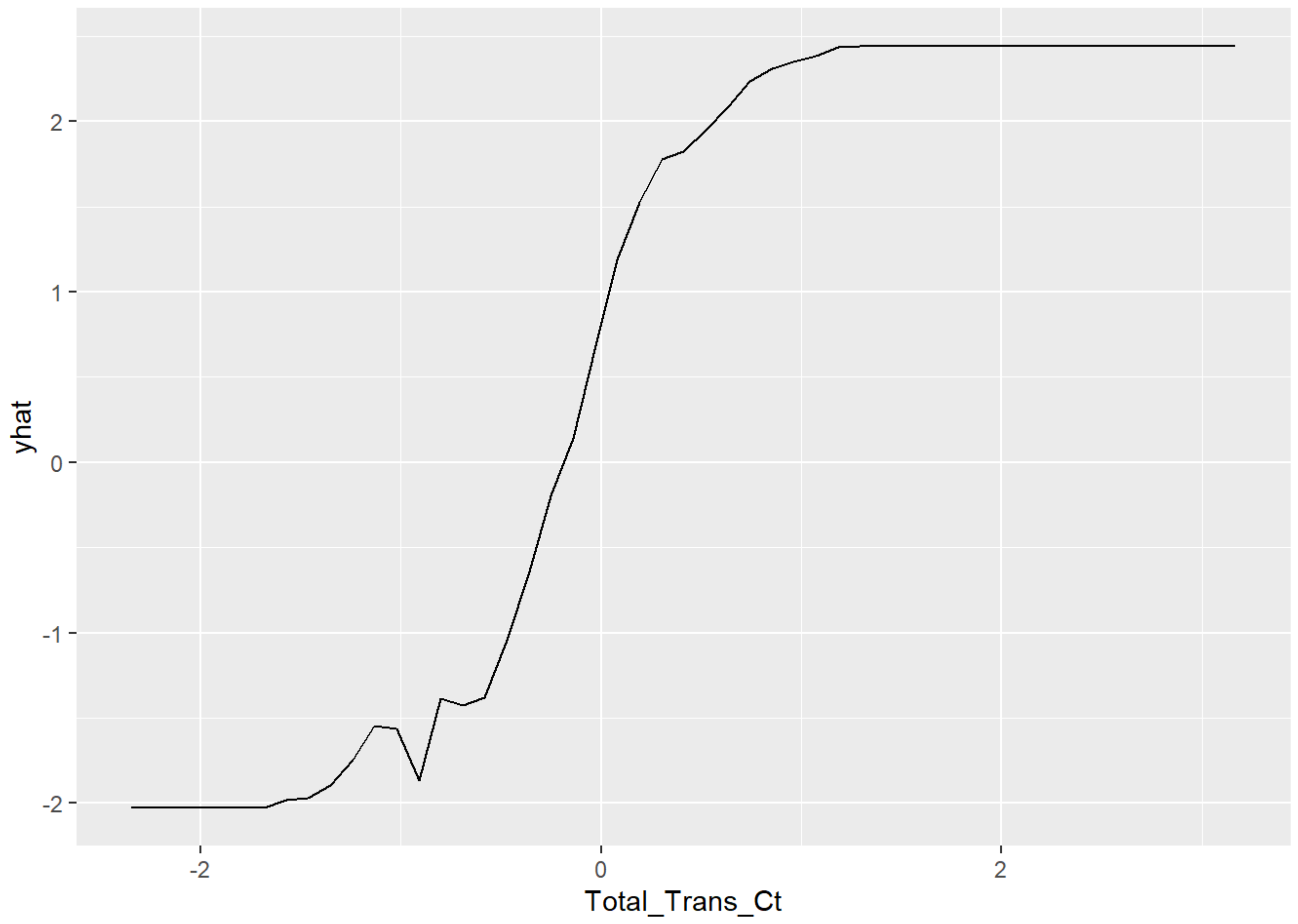
```
## Warning: Use of `object[[1L]]` is discouraged. Use `.data[[1L]]` instead.
```

```
## Warning: Use of `object[["yhat"]]` is discouraged. Use `.data[["yhat"]]`
## instead.
```



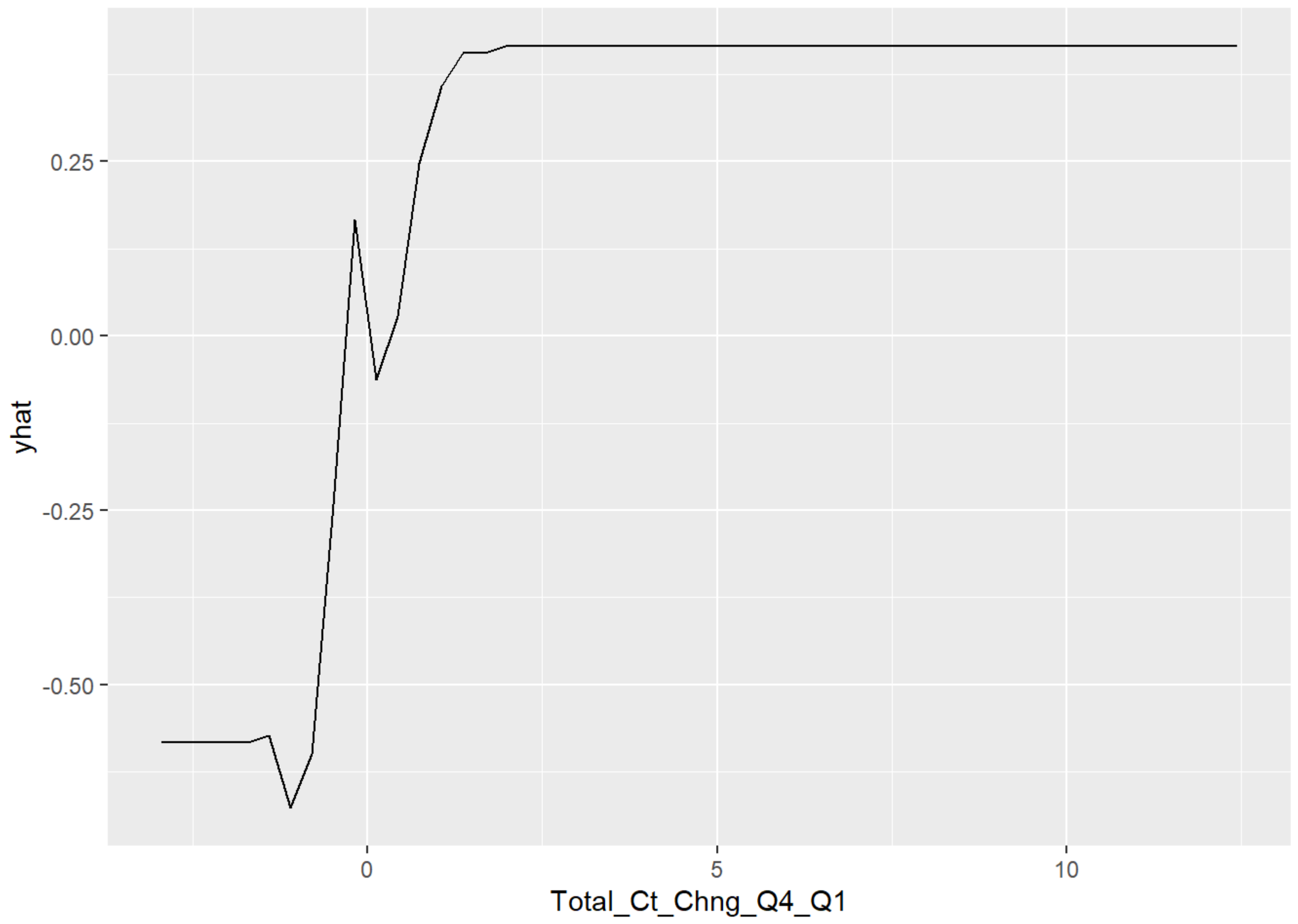
```
partial(xgb1,pred.var = "Total_Trans_Ct", train = customers.smoted1[,-18]) %>% autoplot()
```

```
## Warning: Use of `object[[1L]]` is discouraged. Use `.data[[1L]]` instead.

## Warning: Use of `object[["yhat"]]` is discouraged. Use `.data[["yhat"]]`
## instead.
```



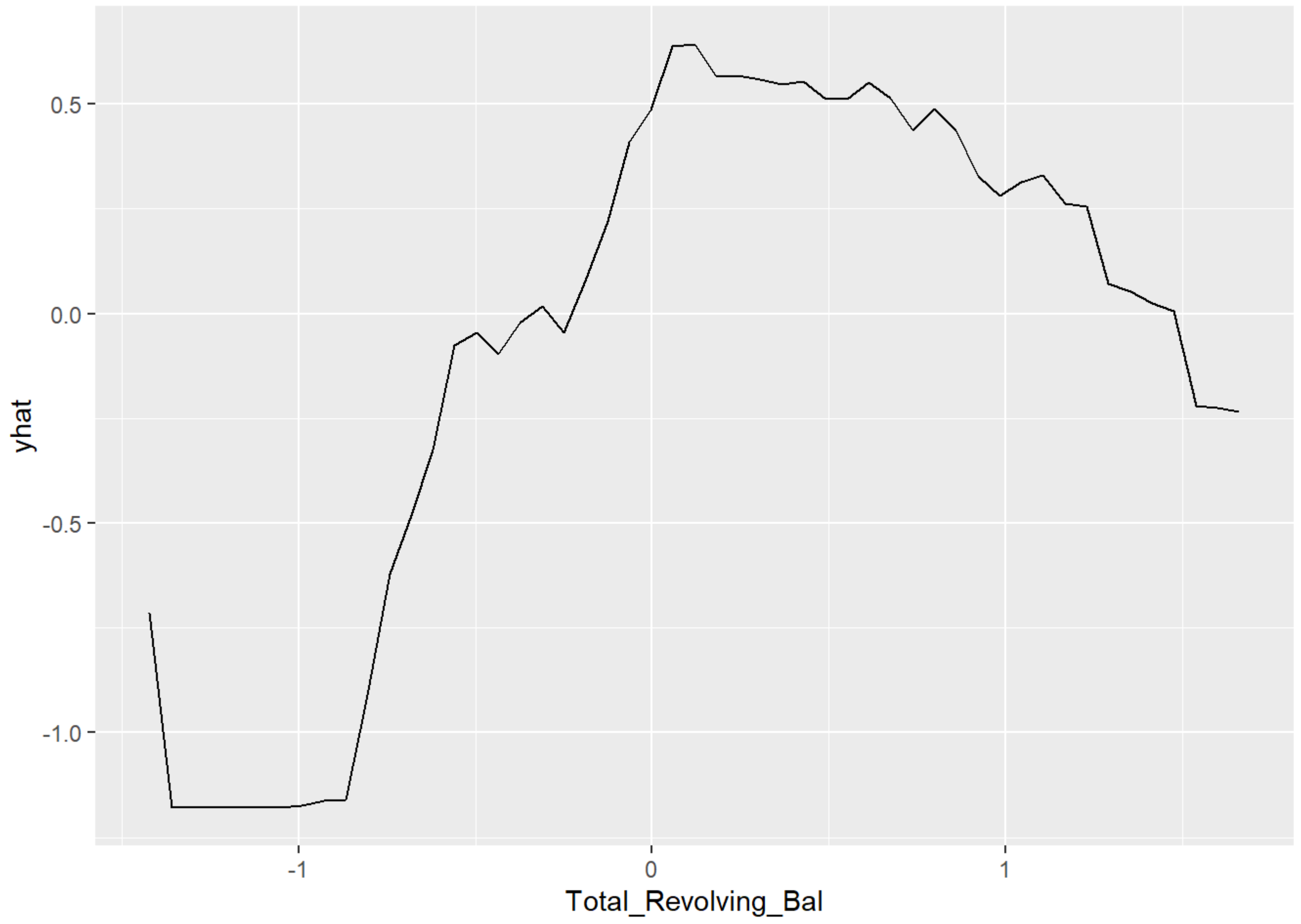
```
partial(xgb1,pred.var = "Total_Ct_Chng_Q4_Q1", train = customers.smoted1[,-18]) %>% autoplot()
```

```
## Warning: Use of `object[[1L]]` is discouraged. Use `.data[[1L]]` instead.

## Warning: Use of `object[["yhat"]]` is discouraged. Use `.data[["yhat"]]`
## instead.
```



```
partial(xgb1,pred.var = "Total_Revolving_Bal", train = customers.smoted1[,-18]) %>% autoplot()
```

```
## Warning: Use of `object[[1L]]` is discouraged. Use `.data[[1L]]` instead.

## Warning: Use of `object[["yhat"]]` is discouraged. Use `.data[["yhat"]]`
## instead.
```



```
#ROC for training
library(pROC)
roc_rf_train <- roc(as.numeric(customers.smoted1$Attrition_Flag), as.numeric(pred_rf_train))
```

```
## Setting levels: control = 1, case = 2
```

```
## Setting direction: controls < cases
```

```
roc_bgbm_train <- roc(as.numeric(customers.smoted1$Attrition_Flag), as.numeric(pred_bgbm_train))
```

```
## Setting levels: control = 1, case = 2
## Setting direction: controls < cases
```

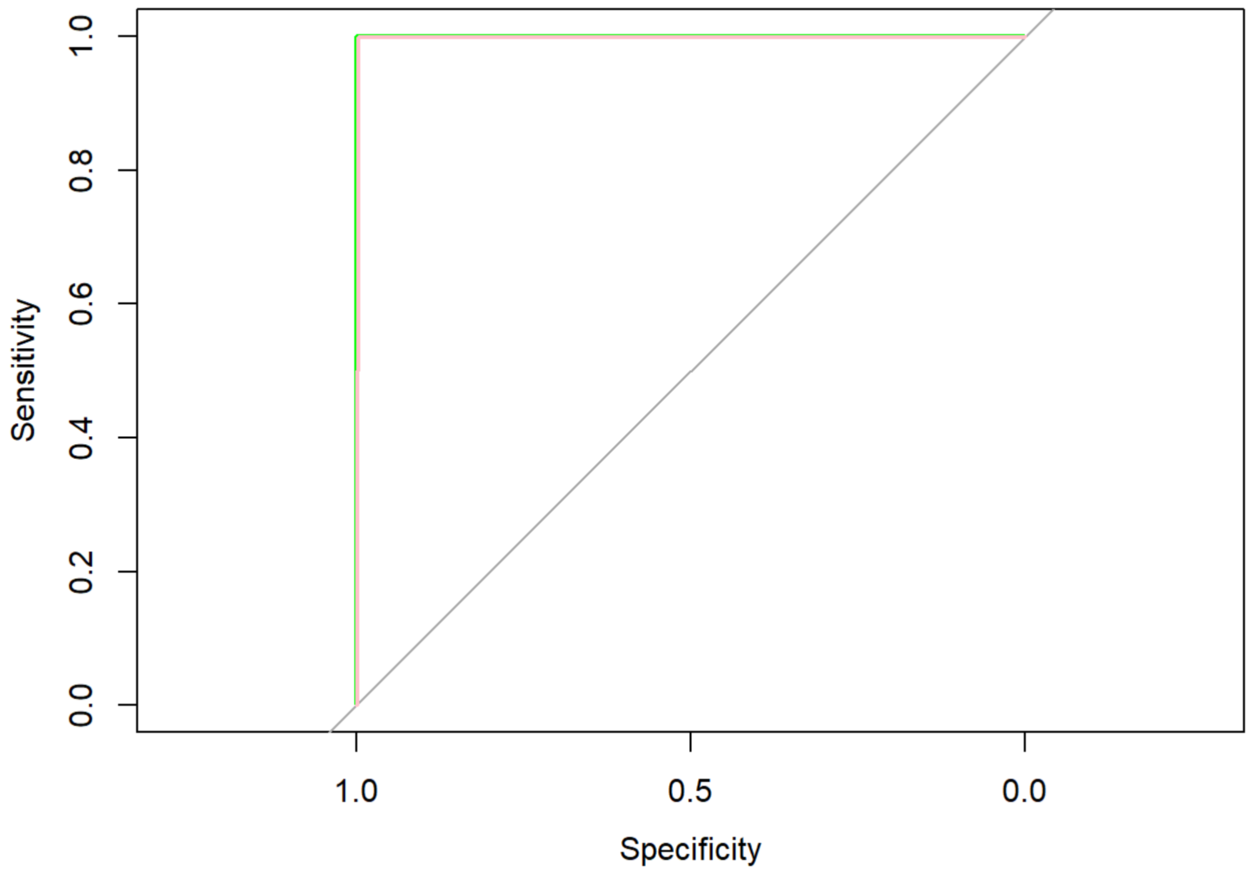
```
roc_sgbm_train <- roc(as.numeric(customers.smoted1$Attrition_Flag), as.numeric(pred_sgbm_train))
```

```
## Setting levels: control = 1, case = 2
## Setting direction: controls < cases
```

```
roc_xgb_train <- roc(as.numeric(customers.smoted1$Attrition_Flag), as.numeric(pred_xgb_train))
```

```
## Setting levels: control = 1, case = 2
## Setting direction: controls < cases
```

```
plot(roc_rf_train,col = "red")#draw the roc plots of the three model
plot.roc(roc_bgbm_train, add = TRUE,col = "blue")
plot.roc(roc_sgbm_train, add = TRUE, col = "green")
plot.roc(roc_xgb_train, add = TRUE, col = "pink")
```



```
roc_rf_train$auc#get the auc score of the three models
```

```
## Area under the curve: 1
```

```
roc_bgbm_train$auc
```

```
## Area under the curve: 1
```

```
roc_sgbm_train$auc
```

```
## Area under the curve: 1
```

```
roc_xgb_train$auc
```

```
## Area under the curve: 0.9975
```

```
#ROC for testing
library(pROC)
roc_rf_test <- roc(as.numeric(customers.test$Attrition_Flag), as.numeric(pred_rf_test))
```

```
## Setting levels: control = 1, case = 2
```

```
## Setting direction: controls < cases
```

```
roc_bgbm_test <- roc(as.numeric(customers.test$Attrition_Flag), as.numeric(pred_bgbm_test))
```

```
## Setting levels: control = 1, case = 2
## Setting direction: controls < cases
```

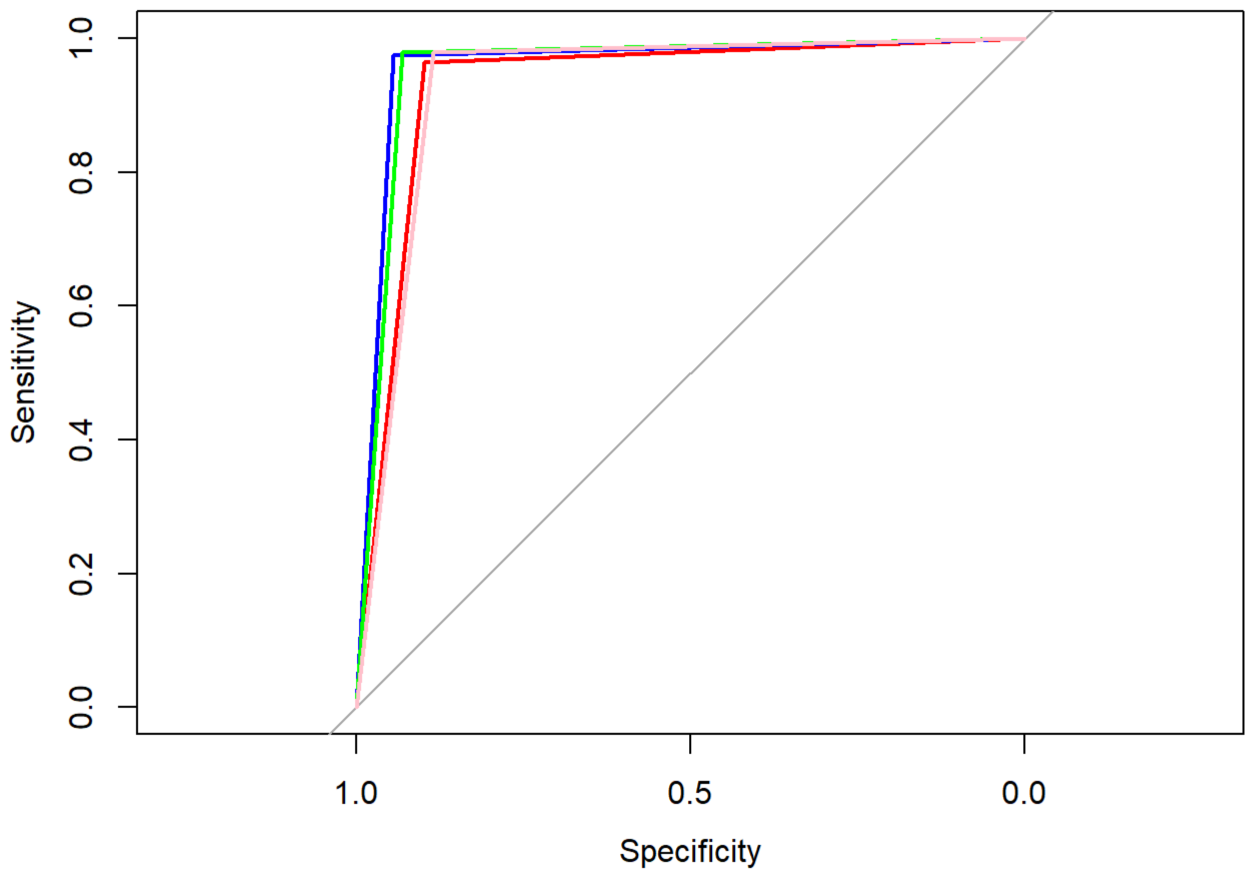
```
roc_sgbm_test <- roc(as.numeric(customers.test$Attrition_Flag), as.numeric(pred_sgbm_test))
```

```
## Setting levels: control = 1, case = 2
## Setting direction: controls < cases
```

```
roc_xgb_test <- roc(as.numeric(customers.test$Attrition_Flag), as.numeric(pred_xgb_test))
```

```
## Setting levels: control = 1, case = 2
## Setting direction: controls < cases
```

```
plot(roc_rf_test,col = "red")#draw the roc plots of the three model
plot.roc(roc_bgbm_test, add = TRUE,col = "blue")
plot.roc(roc_sgbm_test, add = TRUE, col = "green")
plot.roc(roc_xgb_test, add = TRUE, col = "pink")
```



```
roc_rf_test$auc#get the auc score of the three models
```

```
## Area under the curve: 0.9304
```

```
roc_bgbm_test$auc
```

```
## Area under the curve: 0.9587
```

```
roc_sgbm_test$auc
```

```
## Area under the curve: 0.9549
```

```
roc_xgb_test$auc
```

```
## Area under the curve: 0.9319
```

```
#of all the models, stochasitic GBM has the highest test AUC
```

Kmeans

```r
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 4.1.2
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v tibble  3.1.6     v stringr 1.4.0
## v tidyr   1.1.4     v forcats 0.5.1
## v readr   2.1.1
```

```
## Warning: package 'tibble' was built under R version 4.1.2
```

```
## Warning: package 'tidyr' was built under R version 4.1.2
```

```
## Warning: package 'readr' was built under R version 4.1.2
```

```
## Warning: package 'stringr' was built under R version 4.1.2
```

```
## Warning: package 'forcats' was built under R version 4.1.2
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x purrr::accumulate() masks foreach::accumulate()
## x dplyr::filter()     masks stats::filter()
## x stringr::fixed()    masks recipes::fixed()
## x dplyr::lag()        masks stats::lag()
## x purrr::lift()       masks caret::lift()
## x pdp::partial()      masks purrr::partial()
## x xgboost::slice()    masks dplyr::slice()
## x purrr::when()       masks foreach::when()
```

```
# use kmeans to do clustering
# determine optimal number of clusters

# drop response and categorical columns
customers.train.k3<-customers.smoted1[-c(18,2,4,5,6,19,20,21)]

set.seed(1234)

# function to compute total within-cluster sum of square
wss <- function(k) {
  kmeans(customers.train.k3, k, nstart = 10 )$tot.withinss
}

# Compute and plot wss for k = 1 to k = 15
k.values <- 1:15

# extract wss for 2-15 clusters
wss_values <- map_dbl(k.values, wss)
```

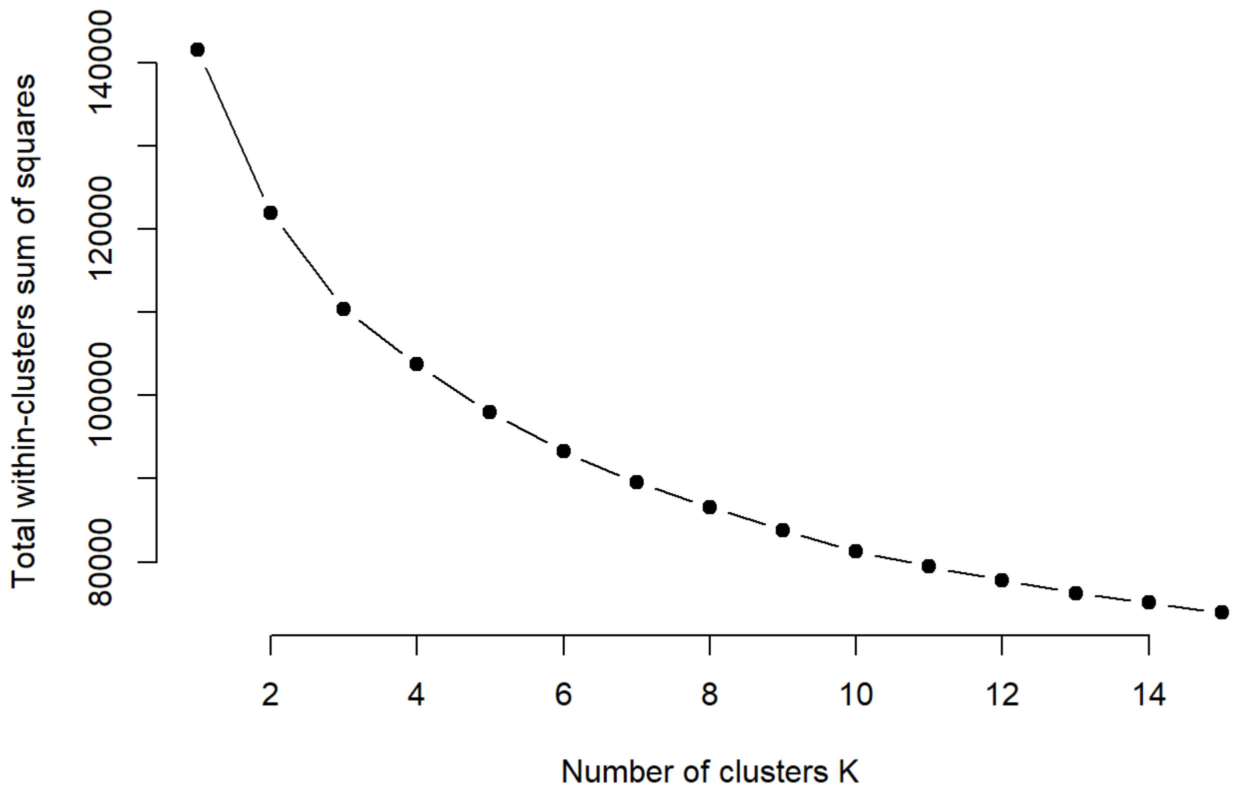
```
## Warning: did not converge in 10 iterations
```

```
## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations
```

```
## Warning: Quick-TRANSfer stage steps exceeded maximum (= 585950)
```

```
## Warning: did not converge in 10 iterations
```

```
plot(k.values, wss_values,
       type="b", pch = 19, frame = FALSE,
       xlab="Number of clusters K",
       ylab="Total within-clusters sum of squares")
```



```
# choose 6 as the optimal number of clusters.
# use numeric columns to do kmeans clustering
set.seed(123)
res.km3 <- kmeans(customers.train.k3,centers = 6, nstart = 25)
```

```
## Warning: did not converge in 10 iterations
```

```
# label each cluster
customers.train.k3$cluster<-as.factor(res.km3$cluster)
customers.train.k3$Attrition_Flag<-as.factor(customers.smoted1$Attrition_Flag)

# add back categorical columns
customers.train.k3$Gender<-customers.smoted1$Gender
customers.train.k3$Education_Level<-customers.smoted1$Education_Level
customers.train.k3$Income_Category<-customers.smoted1$Income_Category
customers.train.k3$Card_Category<-customers.smoted1$Card_Category
customers.train.k3$Marital_Status_Married<-customers.smoted1$Marital_Status_Married
customers.train.k3$Marital_Status_Single<-customers.smoted1$Marital_Status_Single
customers.train.k3$Marital_Status_Unknown<-customers.smoted1$Marital_Status_Unknown
```

```
library(cbar)
```

```
## Warning: package 'cbar' was built under R version 4.1.2
```

```
# destandardize numeric columns for better interpretation
customers.train.k3$Customer_Age<-destandardized(customers.train.k3$Customer_Age,mean(train$Customer_Age),sd(train$Customer_Age))
customers.train.k3$Dependent_count<-destandardized(customers.train.k3$Dependent_count,mean(train$Dependent_count),sd(train$Dependent_count))
customers.train.k3$Months_on_book<-destandardized(customers.train.k3$Months_on_book,mean(train$Months_on_book),sd(train$Months_on_book))
customers.train.k3$Total_Relationship_Count<-destandardized(customers.train.k3$Total_Relationship_Count,mean(train$Total_Relationship_Count),sd(train$Total_Relationship_Count))
customers.train.k3$Months_Inactive_12_mon<-destandardized(customers.train.k3$Months_Inactive_12_mon,mean(train$Months_Inactive_12_mon),sd(train$Months_Inactive_12_mon))
customers.train.k3$Contacts_Count_12_mon<-destandardized(customers.train.k3$Contacts_Count_12_mon,mean(train$Contacts_Count_12_mon),sd(train$Contacts_Count_12_mon))
customers.train.k3$Credit_Limit<-destandardized(customers.train.k3$Credit_Limit,mean(train$Credit_Limit),sd(train$Credit_Limit))
customers.train.k3$Total_Revolving_Bal<-destandardized(customers.train.k3$Total_Revolving_Bal,mean(train$Total_Revolving_Bal),sd(train$Total_Revolving_Bal))
customers.train.k3$Total_Amt_Chng_Q4_Q1<-destandardized(customers.train.k3$Total_Amt_Chng_Q4_Q1,mean(train$Total_Amt_Chng_Q4_Q1),sd(train$Total_Amt_Chng_Q4_Q1))
customers.train.k3$Total_Trans_Amt<-destandardized(customers.train.k3$Total_Trans_Amt,mean(train$Total_Trans_Amt),sd(train$Total_Trans_Amt))
customers.train.k3$Total_Trans_Ct<-destandardized(customers.train.k3$Total_Trans_Ct,mean(train$Total_Trans_Ct),sd(train$Total_Trans_Ct))
customers.train.k3$Total_Ct_Chng_Q4_Q1<-destandardized(customers.train.k3$Total_Ct_Chng_Q4_Q1,mean(train$Total_Ct_Chng_Q4_Q1),sd(train$Total_Ct_Chng_Q4_Q1))
customers.train.k3$Avg_Utilization_Ratio<-destandardized(customers.train.k3$Avg_Utilization_Ratio,mean(train$Avg_Utilization_Ratio),sd(train$Avg_Utilization_Ratio))
```

```
#write cluster to csv
#write.csv(customers.train.k3,"C:/Users/AniS/Desktop/MSBA/7027/pj/cluster1215.3.csv", row.names=FALSE)
```