Nationwide Modeling Exercise

QIWEI MEN

8/7/2020

```r
library(dplyr)

##
## 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

library(ggplot2)
library(stringr)
library(naniar)
library(finalfit)

library(pROC)

library(ggplot2)
library(ROCR)

library(randomForest)
```

Problem Understanding

From the instruction of the project, we know that there are 79 variables drawn from two different databases A and B, and every group has been split into training set and testing set. The response value y was labeled 0 and 1, the purpose of this project is building two efficient models to predict the response y in the testing set.

Based on this information, we can conclude that this is a typical classification problem of supervised machine learning. After checking my toolbox, my initial plan is building one logistic regression and one tree-based model (random forest or gradient boosting) and compare their performance.

Data Checking

```r
# set working directory
setwd("C:/Users/Dain/Desktop/Nationwide Modeling Exercise")

# import The data
a_learn <- read.csv("featuresGroupAlearning.csv", header = TRUE, na.strings=c("","NA"))
b_learn <- read.csv("featuresGroupBlearning.csv", header = TRUE, na.strings=c("","NA"))
a_predict <- read.csv("featuresGroupAprediction.csv", header = TRUE,
na.strings=c("","NA"))
b_predict <- read.csv("featuresGroupBprediction.csv", header = TRUE,
na.strings=c("","NA"))
r_learn <- read.csv("responseLearning.csv", header = TRUE)
```

Since the learning data was located in two datasets, we also need join the data firstly.

```r
## join tables
learn <- a_learn %>%
  inner_join(b_learn, by = "id") %>%
  inner_join(r_learn, by = "id")
```

Then take a look at the raw data.

```r
str(learn)

## 'data.frame':    12073 obs. of  81 variables:
##  $ id  : Factor w/ 12073 levels "1000-46","1001-29",..: 9641 8067 10394 43 7018 11423
3845 3797 1840 1232 ...
##  $ a00 : num  510 543 636 549 427 516 612 621 576 590 ...
##  $ a01 : num  -102.2 -58.3 18.5 128.8 -97.5 ...
##  $ a02 : num  13.7 NA 13 13.4 14.5 ...
##  $ a03 : num  -1.996 -4.086 0.775 -11.908 -11.562 ...
##  $ a04 : num  7.45 35.66 -42.79 -40.95 15.83 ...
##  $ a05 : num  -32.58 -16.96 24.49 -8.79 9.48 ...
##  $ a06 : num  -11.4 4.3 -10.5 -10.3 -14.8 ...
##  $ a07 : num  -1.83 -1.28 -1.52 NA -1.7 ...
##  $ a08 : num  5.06 3.61 -39.46 -41.56 3.61 ...
##  $ a09 : num  -22.83 -6.54 2.03 -8.88 -19.89 ...
##  $ a10 : num  -13 -10.4 -14 -17 -11 ...
##  $ a11 : num  -5.94 3.73 -20.14 -3.88 -10.86 ...
##  $ a12 : num  4 4 4 5 4 4 3 4 5 4 ...
##  $ a13 : num  -10.92 -5.28 3.03 -3.23 -5.7 ...
##  $ a14 : num  6 6 8 4 4 6 4 5 6 6 ...
##  $ a15 : num  665 740 369 598 666 654 765 657 735 848 ...
##  $ a16 : num  NA -17.08 27.72 43.38 -2.67 ...
##  $ a17 : num  -9.83 -23.26 -4.18 -11.28 -8.39 ...
##  $ code: int  1000001 1000003 1000008 1000005 1000008 1000002 1000009 1000009 1000009
1000009 ...
##  $ b00 : Factor w/ 12000 levels "-0.15","-0.16",..: 7323 8754 11045 8387 591 9517 3056
11671 9781 7767 ...
##  $ b01 : num  11.6 31.5 -15.3 -26.2 -16.9 ...
##  $ b02 : num  -19.4 33.2 46.7 18.2 32.2 ...
##  $ b03 : num  -65.53 -3.8 2.01 40.49 -85.12 ...
##  $ b04 : num  43.79 -2.62 3.44 -24.65 -12.81 ...
##  $ b05 : num  -9.65 -6.45 -12.79 -12.89 -14.74 ...
##  $ b06 : num  0.683 17.589 6.14 3.597 -4.468 ...
##  $ b07 : num  709 742 680 616 660 473 562 583 729 519 ...
##  $ b08 : num  4.32 -1.2 1.14 -9.1 -1.88 ...
##  $ b09 : num  14.29 8.68 13.7 3.93 14.15 ...
##  $ b10 : num  4.6 15.37 -8.71 4.61 4.49 ...
##  $ b11 : num  2.54 -2.64 4.07 5.15 5.95 ...
##  $ b12 : num  0.854 -10.472 45.572 5.704 25.42 ...
##  $ b13 : num  3 6 4 6 6 6 5 5 4 4 ...
##  $ b14 : num  6 6 7 5 6 5 6 6 5 7 ...
##  $ b15 : num  -27.73 6.64 8.73 16.92 -27.41 ...
##  $ b16 : num  5.8 7.39 -7.27 18.95 22.32 ...
##  $ b17 : num  11.66 18.06 18.13 4.41 21.11 ...
##  $ b18 : num  2.39 4.55 -25.67 -2.67 -14.74 ...
##  $ b19 : num  11.59 16.34 4.63 7.2 -6.27 ...
##  $ b20 : num  11 18.2 26 23 5.1 ...
##  $ b21 : Factor w/ 3 levels "Central","East",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ b22 : num  10.93 1.46 -17.57 -40.13 10.51 ...
##  $ b23 : num  -5.5 -5.75 -5.65 -1.67 -3.65 ...
##  $ b24 : num  4.167 0.584 5.084 6.312 4.757 ...
##  $ b25 : num  366 719 792 932 798 522 766 468 970 891 ...
##  $ b26 : num  -6.69 -10.54 -2.55 -7.81 -15 ...
##  $ b27 : num  12.99 10.57 10.97 9.72 12.92 ...
##  $ b28 : num  -13.87 -6.52 -4.48 -2.99 -11.88 ...
##  $ b29 : num  3.27 3.61 -2.74 4.96 5.6 ...
##  $ b30 : num  -14.2 -14.2 -24.4 -19.2 -25.3 ...
##  $ b31 : num  36.63 1.35 -0.969 -38.719 -2.301 ...
##  $ b32 : num  -7.272 -5.633 -11.266 0.449 -5.846 ...
##  $ b33 : num  6.2 25.6 50.8 44.4 31.7 ...
##  $ b34 : Factor w/ 31 levels "fri","Fri","fri.",..: 27 12 29 27 29 29 28 29 24 12 ...
##  $ b35 : Factor w/ 29 levels "-0.0","-0.0 %",..: 19 3 19 4 17 2 1 4 4 5 ...
##  $ b36 : num  30 14 29.7 26 20.6 ...
##  $ b37 : num  -22.66 -3.82 -36.39 16.53 -28.78 ...
##  $ b38 : num  -9.56 2 -5.2 1.69 -6.52 ...
##  $ b39 : num  4 6 7 6 8 7 5 5 6 7 ...
##  $ b40 : num  0.985 -1.439 -1.411 4.923 0.559 ...
##  $ b41 : num  7.78 -2.33 -23.1 -2.41 -11.31 ...
##  $ b42 : num  -10.1 -15.2 -13.8 -11 -13.4 ...
##  $ b43 : num  -7.63 -11.54 -20.45 -9.46 -3.93 ...
##  $ b44 : num  2.75 14.5 6.02 14.6 11.31 ...
##  $ b45 : num  7.68 9.77 -3.77 -1.82 12.85 ...
##  $ b46 : num  2.28 14.11 5.52 20.15 9.96 ...
##  $ b47 : num  -10.29 -6.37 -8.69 -13.85 7.59 ...
##  $ b48 : num  58.6 21.3 41.1 -87.5 36.8 ...
##  $ b49 : num  57.51 -5.09 -7.93 -43.38 12.66 ...
##  $ b50 : num  61.06 26.88 18.52 -5.47 -17.65 ...
##  $ b51 : num  0.805 11.189 -4.393 -11.552 1.836 ...
##  $ b52 : num  29.79 81.45 -22.49 37.74 9.11 ...
##  $ b53 : num  -9.9 -11.92 -6.34 -15.57 -20.8 ...
##  $ b54 : num  -40.2 -78.7 -49.9 -118.4 16.6 ...
##  $ b55 : num  -10.81 -9.88 -12.72 -11.58 -13.23 ...
##  $ b56 : num  2.714 2.796 -5.446 -2.114 -0.387 ...
##  $ b57 : num  2.53 4.76 1.62 6.59 8.35 ...
##  $ b58 : Factor w/ 61 levels "Apirl","apr",..: 32 29 31 32 27 8 41 49 14 27 ...
##  $ zip : int  83298 86333 92106 38978 42005 89139 0 99999 10071 87732 ...
##  $ y   : int  0 1 0 0 0 0 0 0 0 0 ...
```

From the summary of data, most variables are anaoymized, we don't know the actual meaning of these variables, so it's impossible to check if their ranges are reasonable. However, there are still some problem we need to fix in the data.

- code: should be a categorical variable but not int
- b00: some entries are with "$" and some are not, the format is not consistent, and it also should be numeric variable not factor.
- b34: this variable is "weekday", but there are spelling mistakes and inconsistencies. eg. "Mon" and "mon." all stands for Monday. And it should be a factor.
- b35: some entries are with "%" and some are not, the format is not consistent, and it also should be numeric variable not a factor.

- b58: this variable is “month”, like b34, there are also spelling mistakes and inconsistencies. eg.“apr.” and “Apr” all stands for April.And it should be a factor.
- There are some missing values in b21, b34 and b58 are filled with blank space, they should shown as “NA” to make the level number correct.
- The zip code looks wired, some have 4 digitals, some have 5, there are also fake numbers like “0” and “99999”.
- For the random forest model, we convert the response variable y into a factor

Data cleaning

We need to fix problems mentioned in the data checking section.

Drop or keep the “zip” variable

```
## check wrong zip code
sum(str_length(learn$zip)!=5)

## [1] 1450

sum(learn$zip==99999)

## [1] 770

length(unique(learn$zip))

## [1] 10214
```

we can see that about 18% zip data are wrong or problemic, and there are 10214 levels, we have resson to believe that it’s not a infoamative indicator for prediction, so drop it from the data.

The uniformity of b34(weekdays) and b58(monthes)

Though there are some automatic ways like applying string distance to deal with inconsistent text data, the performance is not good in this case, since there are limited levels, we decide to unify the spelling manually, and we will make this process as a function so we can also deal with data in testing set simply.

```
# build transformation function for weekdays
fix_weekdays <- function(x){

str_replace_all(x,c("mon."="Mon","mon"="Mon","Monday"="Mon","monday"="Mon","Monay"="Mon",
              "tue"="Tue","Tue."="Tue","Tuesday"="Tue","tuesday"="Tue","tue."="Tue",
"Tuey"="Tue","Tueay"="Tue","Tueday"="Tue", "Teusday"="Tue",

"Wednesday"="Wed","wednesday"="Wed","wed"="Wed","Wed."="Wed","Wendsday"="Wed","
wendsday"="Wed", "wendsday"="Wed",

"thu"="Thu","Thursday"="Thu","thu."="Thu","Thu."="Thu","thurday"="Thu","thursday"="Thu",
"Thuay"="Thu","Thuy"="Thu", "Thuday"="Thu",
              "Friday"="Fri","friday"="Fri","Fri."="Fri","fri."="Fri","fri"="Fri"))
}

# built transformation function
fix_monthes <- function(x){

str_replace_all(x,c("apr"="Apr","apr."="Apr","Apr."="Apr","april"="Apr","April"="Apr","Ap
irl"="Apr","Aprl"="Apr",

"aug"="Aug","aug."="Aug","Aug."="Aug","Augest"="Aug","august"="Aug","August"="Aug","Augst
"="Aug","Augt"="Aug",
              "dec"="Dec","dec."="Dec","december"="Dec","December"="Dec","Dec."="Dec",
              "feb"="Feb","feb."="Feb","Feb."="Feb","february"="Feb","February"="Feb",
"Febuary"="Feb","Febary"="Feb", "Febry"="Feb", "Feby" = "Feb",
              "jan"="Jan","jan."="Jan","Jan."="Jan","Janary"="Jan",
              "jun"="Jun","Jun."="Jun","june"="Jun","June"="Jun",
              "jul"="Jul",

"mar"="Mar","mar."="Mar","Mar."="Mar","march"="Mar","March"="Mar","Marhc"="Mar",
"Marc"="Mar","Marc"="Mar", "Marh"="Mar",
              "may"="May",
              "nov"="Nov","nov."="Nov","Nov."="Nov","november"="Nov","November"="Nov",
"Novmber"="Nov","Novber"="Nov", "Nover"="Nov", "Novr"="Nov",

"oct"="Oct","oct."="Oct","Oct."="Oct","october"="Oct","October"="Oct","Octber"="Oct","Oct
er"="Oct", "Octr"="Oct",

"sep"="Sep","sep."="Sep","Sep."="Sep","sept"="Sep","sept."="Sep","Sept."="Sep","september
"="Sep","Sep."="Sep","Sepember"="Sep","Sepber"="Sep","Sepr"="Sep","Sepmber"="Sep","Seper"
="Sep"))
}
```

Fix all the problems in a pipeline

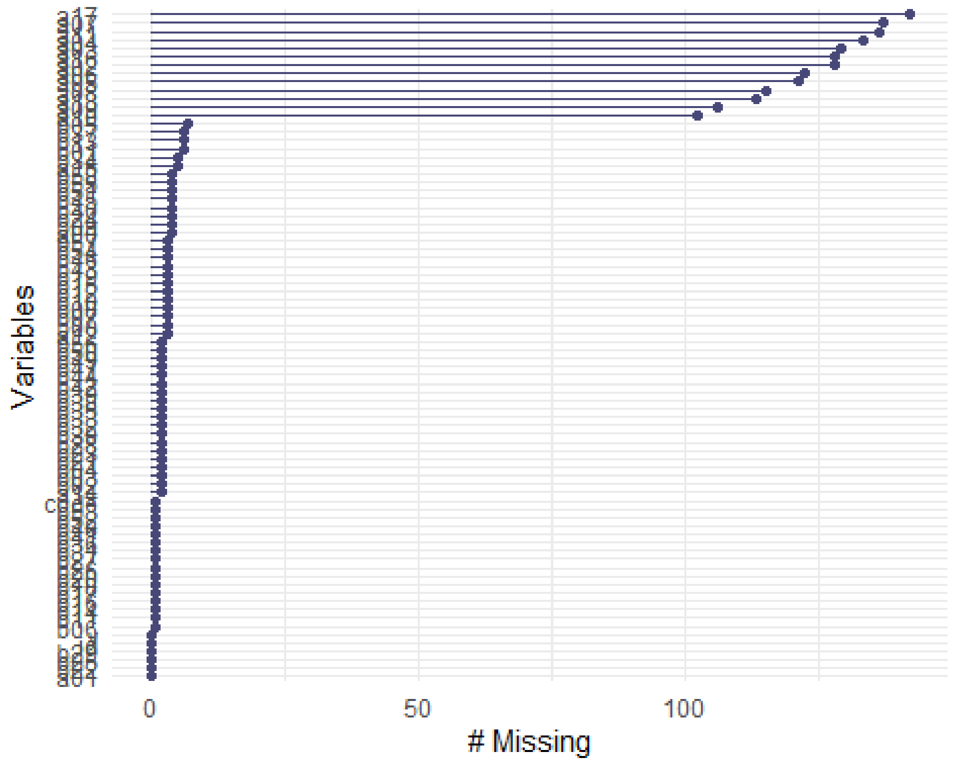
Apply dplyr package to build a pipeline to fix all the data cleaning problems.

```
learn_new <- learn %>%
  # convert code to factor
  mutate(code = as.factor(code)) %>%
  # remove "$" in b00 and convert to numeric
  mutate(b00 = as.numeric(gsub("\\$", "", b00))) %>%
  # remove "%" in b35 and convert to numeric
  mutate(b35 = as.numeric(gsub("\\%", "", b35))) %>%
  # fix weekdays and convert to factor
  mutate(b34 = as.factor(fix_weekdays(b34))) %>%
  # fix monthes and convert to factor
  mutate(b58 = as.factor(fix_monthes(b58))) %>%
  # convert y to factor
  mutate(y = as.factor(y)) %>%
  # drop the zip variable
  select(-zip)
```

Deal with missing data

```
# check the summary of missing data
miss_var_summary(learn_new)

## # A tibble: 80 x 3
##    variable n_miss pct_miss
##    <chr>     <int>    <dbl>
##  1 a17         142     1.18
##  2 a07         137     1.13
##  3 a11         136     1.13
##  4 a04         133     1.10
##   5 a03          129    1.07
##   6 a02          128    1.06
##   7 a16          128    1.06
##   8 a06          122    1.01
##   9 a05          121    1.00
##  10 a08          115    0.953
## # ... with 70 more rows
```

```
gg_miss_var(learn_new)
```



From the the plot, it looks that the missing occurs randomly, but we still need take a closer look.

```
## drop all rows with NA
learn_clean <- learn_new %>%
  na.omit()

dim(learn_clean)

## [1] 10426    80

learn_missing <- learn_new %>%
  anti_join(learn_clean, by = "id")

dim(learn_missing)

## [1] 1647   80

t.test(as.numeric(learn_clean$y),as.numeric(learn_missing$y))

##
##  Welch Two Sample t-test
##
## data:  as.numeric(learn_clean$y) and as.numeric(learn_missing$y)
## t = -0.27112, df = 2187.5, p-value = 0.7863
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.01826716  0.01382972
## sample estimates:
## mean of x mean of y 
##  1.104642  1.106861
```

the t-test shows that p-value = 0.7863, which means that the response value does not show any difference in complete set and missing set, we believe that the the data is Missing completely at random (MCAR), so simply drop rows with missing values. which is learn_clean.

```
## make id as rownames
learn_clean <- data.frame(learn_clean[,-1], row.names= learn_clean[,1])
```

For now, the data cleaning process is almost completed, we can check the data structure again and make sure all the problems has been fixed.

Split learning data into training and testing dataset

Though we do not know the value of response variable in the true testing dataset, we can still evaluate the performance of the model by splitting the learning data into training and testing part, first built som prototype models with training data, then test their performances with testing data, select the most powerful model, finally, use the learning data to produce the final model.

One thing needs to be noted is that random forest package has built in function to calculate the OOB error which can be taken as a metric to evaluate the performance of the model. However, in this project we need to compare the performance of two different models, creating a testing data set is necessary for comparison.

```
## creat train and test data
# Total number of rows in the data frame
n <- nrow(learn_clean)

# Number of rows for the training set (75% of the dataset)
n_train <- round(0.75 * n)

# Create a vector of indices which is an 75% random sample
set.seed(123)
train_indices <- sample(1:n, n_train)

# Subset the credit data frame to training indices only
learn_train <- learn_clean[train_indices, ]
learn_train2 <- learn_train %>%
  mutate(y = as.numeric(as.character(y)))

# Exclude the training indices to create the test set
learn_test <- learn_clean[-train_indices, ]
learn_test2 <- learn_test %>%
  mutate(y = as.numeric(as.character(y)))

dim(learn_train)

## [1] 7820   79

dim(learn_test)
```

```
## [1] 2606   79
```

We have 7820 observations in the training data and 2606 observations in the testing data.

Clean prediction dataset

```
## join tables
predict <- a_predict %>%
  inner_join(b_predict, by = "id")

# clean the data
predict_new <- predict %>%
  # convert code to factor
  mutate(code = as.factor(code)) %>%
  # remove "$" in b00 and convert to numeric
  mutate(b00 = as.numeric(gsub("\\$", "", b00))) %>%
  # remove "%" in b35 and convert to numeric
  mutate(b35 = as.numeric(gsub("\\%", "", b35))) %>%
  # fix weekdays and convert to factor
  mutate(b34 = as.factor(fix_weekdays(b34))) %>%
  # fix monthes and convert to factor
  mutate(b58 = as.factor(fix_monthes(b58))) %>%
  # drop the zip variable
  select(-zip)
```

Fix NAs in the predict data

Instead of removing the missing data, we impute the NAs as means.

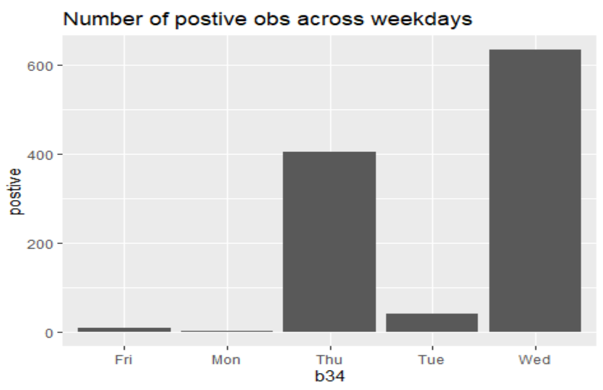
```
predict_clean <- na.roughfix(predict_new)
## make id as rownames
predict_clean <- data.frame(predict_clean[,-1], row.names= predict_clean[,1])
```

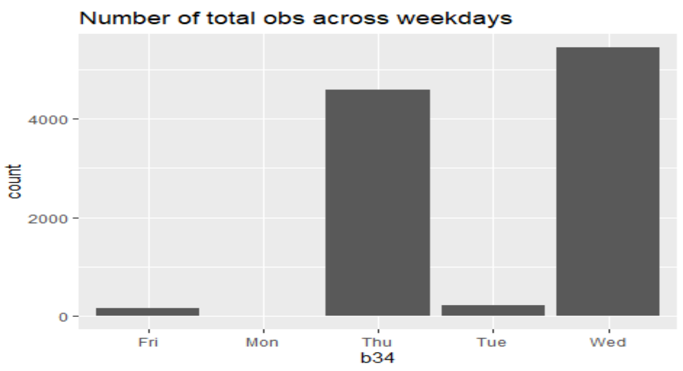
Data exploration

Check the distribution of "1"s in character variables.

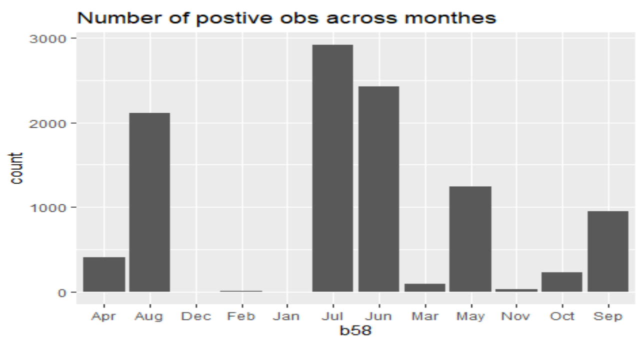
In this data set, we only know the meaning of 4 character variables, which are weekdays, months, regions and codes, we want to check if "1"s are randomly distributed in these variables or they concentrated in some ones.

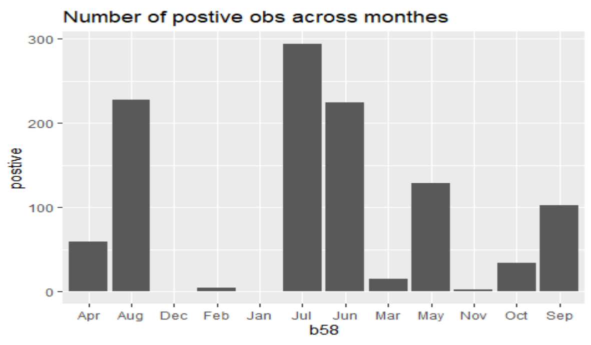
```
# Define by_code
by_weekdays <- learn_clean %>%
  group_by(b34) %>%
  summarize(postive = n()* mean(y == 1))

# Create bar plot
ggplot(by_weekdays, aes(x = b34, y = postive, group = 1)) +
  geom_col() +
  labs(title="Number of postive obs across weekdays")
```





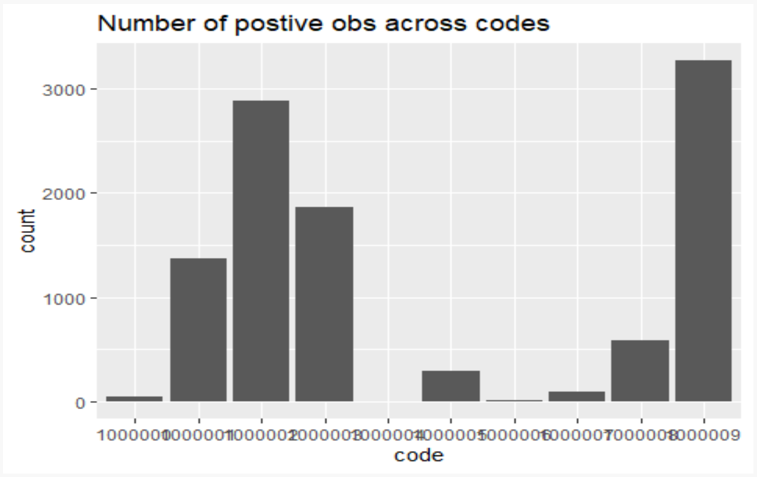
```
ggplot(learn_clean, aes(x = b34)) +
  geom_bar() +
  labs(title="Number of total obs across weekdays")

# Define by_code
by_monthes <- learn_clean %>%
  group_by(b58) %>%
  summarize(postive = n()* mean(y == 1))

# Create bar plot
ggplot(by_monthes, aes(x = b58, y = postive, group = 1)) +
  geom_col() +
  labs(title="Number of postive obs across monthes")

ggplot(learn_clean, aes(x = b58)) +
  geom_bar()+
  labs(title="Number of postive obs across monthes")
```

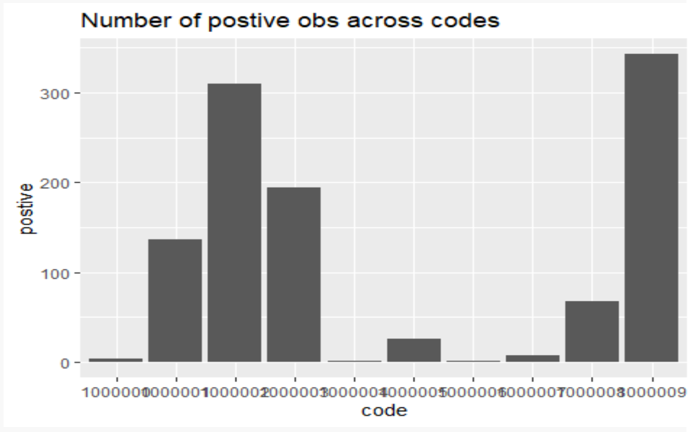




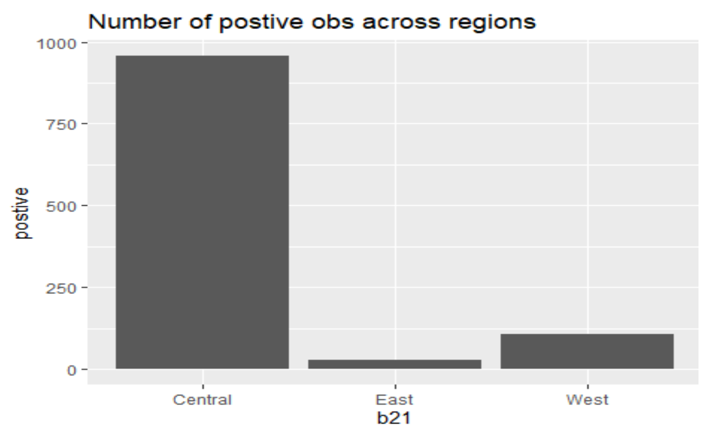
```
# Define by_code
by_code <- learn_clean %>%
  group_by(code) %>%
  summarize(postive = n()* mean(y == 1))

# Create bar plot
ggplot(by_code, aes(x = code, y = postive, group = 1)) +
  geom_col() +
  labs(title="Number of postive obs across codes")
```

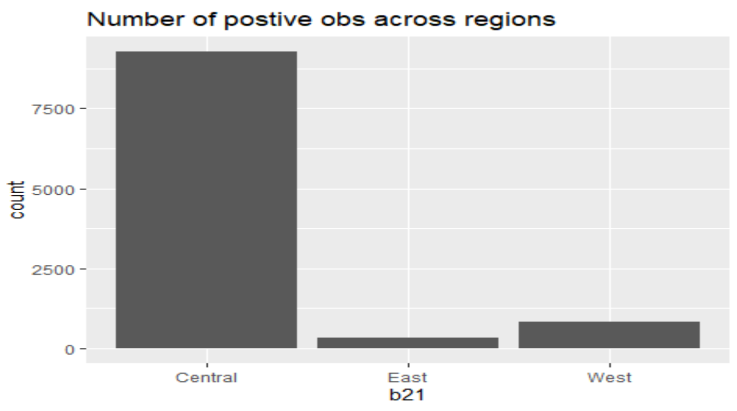
```
ggplot(learn_clean, aes(x = code)) +
  geom_bar() +
  labs(title="Number of postive obs across codes")
```





```
# Define by_region
by_region <- learn_clean %>%
  group_by(b21) %>%
  summarize(postive = n()* mean(y == 1))


# Create bar plot
ggplot(by_region, aes(x = b21, y = postive, group = 1)) +
  geom_col()+
  labs(title="Number of postive obs across regions")
```



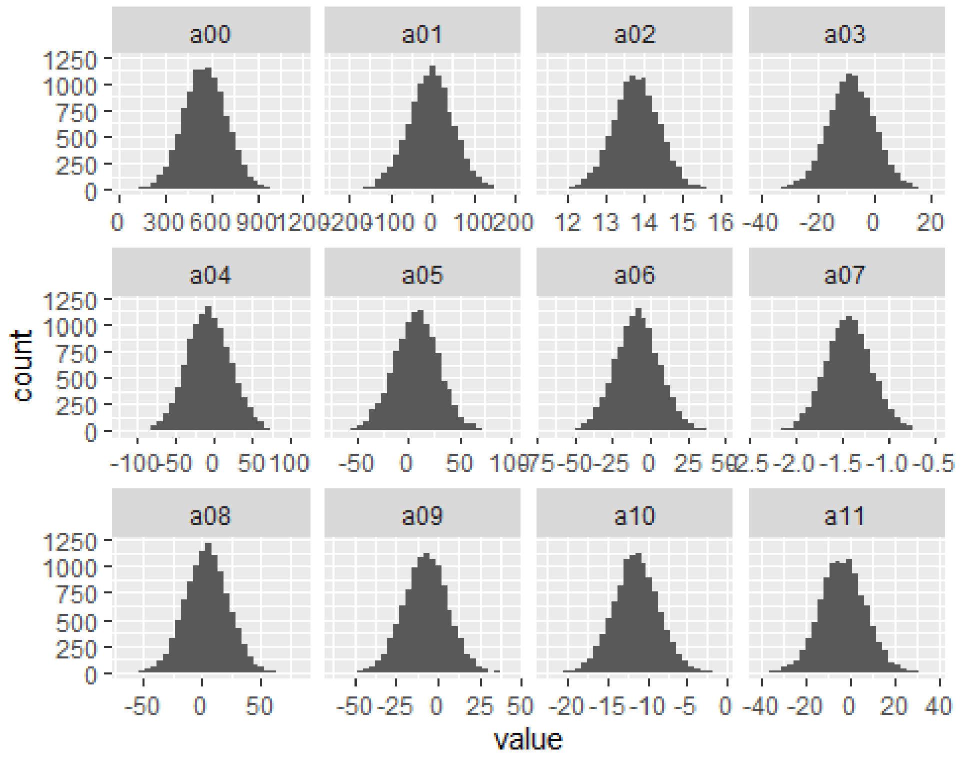


```
ggplot(learn_clean, aes(x = b21)) +
  geom_bar()+
  labs(title="Number of postive obs across regions")
```

From the plot we can say that positive obs are randomly and proportionally distributed across character variables.

Check the distribution of numeric variables

```
library(ggfortify)
ggplot(tidyr::gather(learn_clean[1:12]), aes(value)) +
    geom_histogram(bins = 30) +
    facet_wrap(~key, scales = 'free_x')
```



we just show the first 12 variables(a00-a11) here, as the plot shows, all the numeric variables are normally distributed.

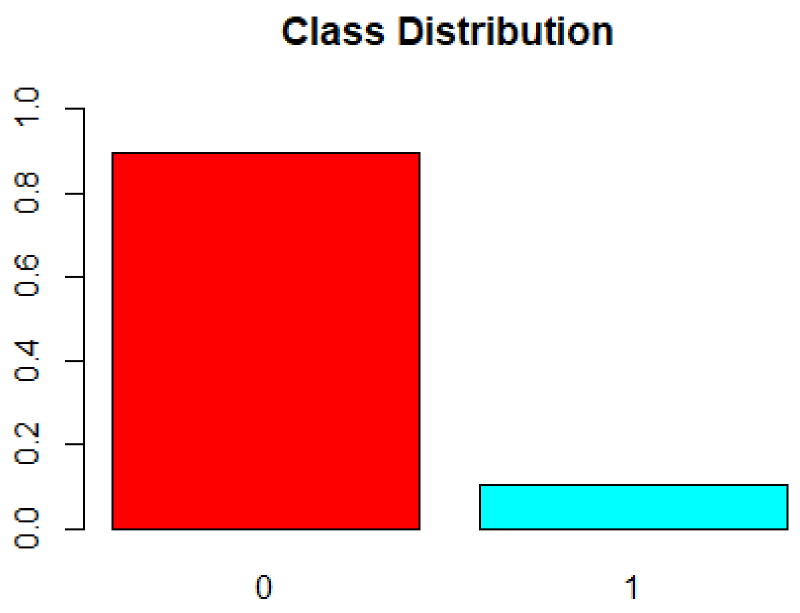
Deal with inbalance data

Imbalanced classes of y

When checking and cleaning the data, we had already noticed that the class of response variable y is not balance, from the plot, the ratio of “0” and “1” is about 9:1.

Imbalanced class is a common problem in machine learning classification where there is a disproportionate ratio of observations in each class. Class imbalance can be found in many different areas including medical diagnosis, spam filtering, and fraud detection.

```
barplot(prop.table(table(learn_clean$y)),
        col = rainbow(2),
        ylim = c(0, 1),
        main = "Class Distribution")
```



Fix the data with inbalanced class

- **Choose the proper metrics for evaluatiing the model**

  We know that standered metrics like accuravy and error do not perform well in evaluatiing model performance at imbllance data. There are some candidates like AUC/ROC, F1 score, Recal and Precision can be used for this purpose. One important reference is the purpose of the project, that's something that we are not clear here, is the positive class more important? Or both classes are important? Since the instruction says the assessment metric for these models is ROC/AUC, We will take AUC as main matric, but we will also consider other matrics like F1 score and Precision.

- **Apply oversampling techniques balance the data**

  Oversampling can be defined as adding more copies of the minority class. We will use the `ROSE` package in R to randomly replicate samples from the minority class. there are 4 methods for this resample procedure, "over", "under", "both" and ROSE, we will try all of them and see which one give the best performance

```
library(ROSE)

over <- ovun.sample(y~., data = learn_train, method = "over", seed = 101)$data
table(over$y)

##
##    0    1
## 7003 6969

under <- ovun.sample(y~., data = learn_train, method = "under", seed = 102)$data
table(under$y)

##
##   0   1
## 792 817
```

```
both <- ovun.sample(y~., data=learn_train, method = "both", p = 0.5,seed = 103)$data
table(both$y)

##
##    0    1
## 3873 3947

rose <- ROSE(y~., data = learn_train,seed = 104)$data
table(rose$y)

##
##    0    1
## 3906 3914
```

Modeling

Part1: tree-based model

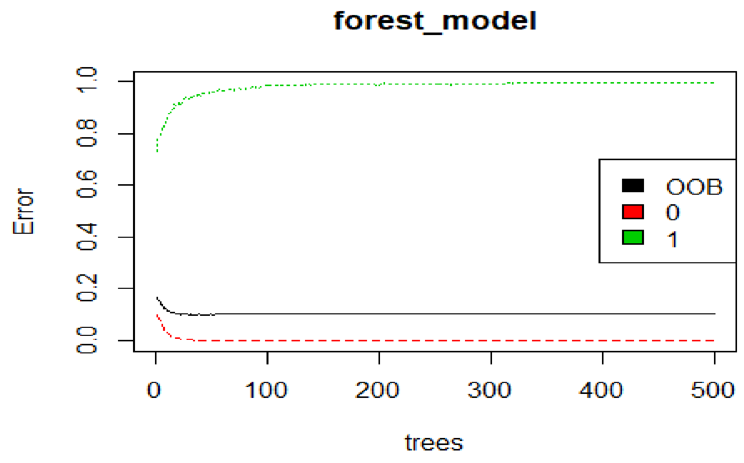
Tree-based model is always a good choice for solving classification problem, here we firstly use the `random Forest package in R to build a random forest model,we also fix the model with different resampled datasets and see which give the best performance.

```
# Load the randomForest package
library(randomForest)

## randomForest model with row training data
forest_model <- randomForest(y ~ ., data = learn_train, ntree = 500)

## randomForest model with over sampled data
forest_over <- randomForest(y ~ ., data = over,ntree = 500)

## randomForest model with under sampled data
forest_under <- randomForest(y ~ ., data = under,ntree = 1000, mtry = 12, nodesize =
1,sampsize = 1300)

## randomForest model with both sampled data
forest_both <- randomForest(y ~ ., data = both,ntree = 500)

## randomForest model with ROSE sampled data
forest_rose <- randomForest(y ~ ., data = rose,ntree = 1000)
```

check the plot of diferent models and see how OOB eror changes across number of trees

```
print(forest_model)

##
## Call:
##  randomForest(formula = y ~ ., data = learn_train, ntree = 500)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 8
##
##         OOB estimate of  error rate: 10.4%
## Confusion matrix:
##      0 1 class.error
## 0 7003 0     0.000000
## 1  813 4     0.995104
```

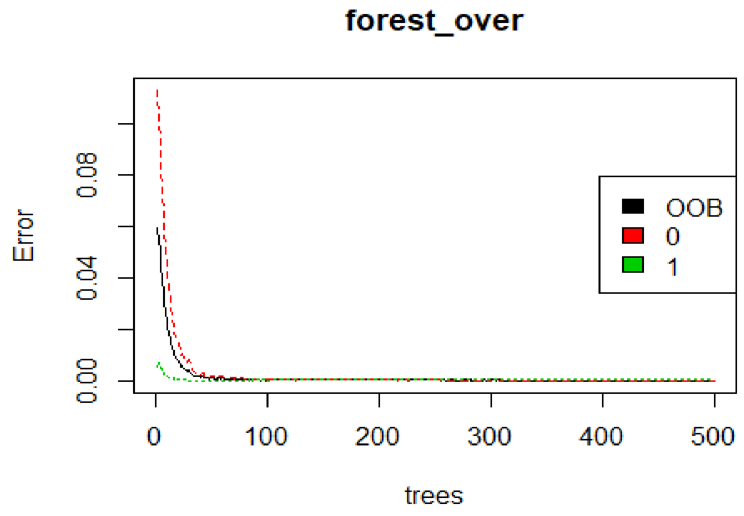
```
plot(forest_model)
legend(x = "right",legend = c( "OOB","0","1" ), fill = 1:3)
```



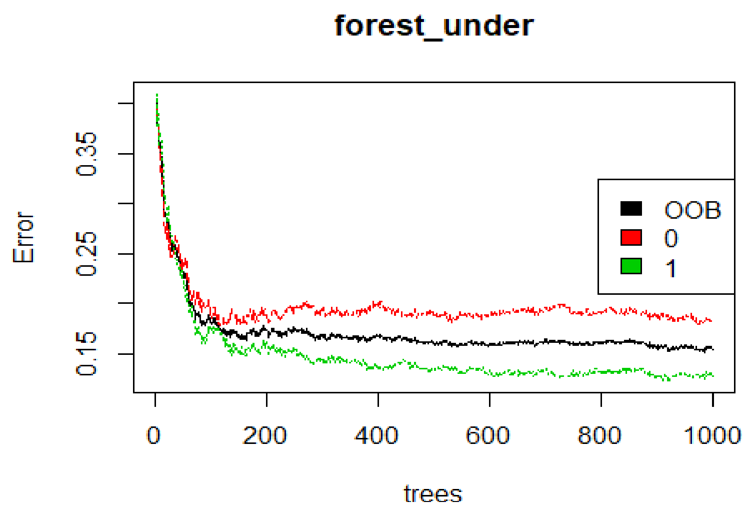
```
print(forest_over)
```

```
##
## Call:
##  randomForest(formula = y ~ ., data = over, ntree = 500)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 8
##
##         OOB estimate of  error rate: 0.02%
## Confusion matrix:
##      0    1  class.error
## 0 7003    0 0.0000000000
## 1    3 6966 0.0004304778
```

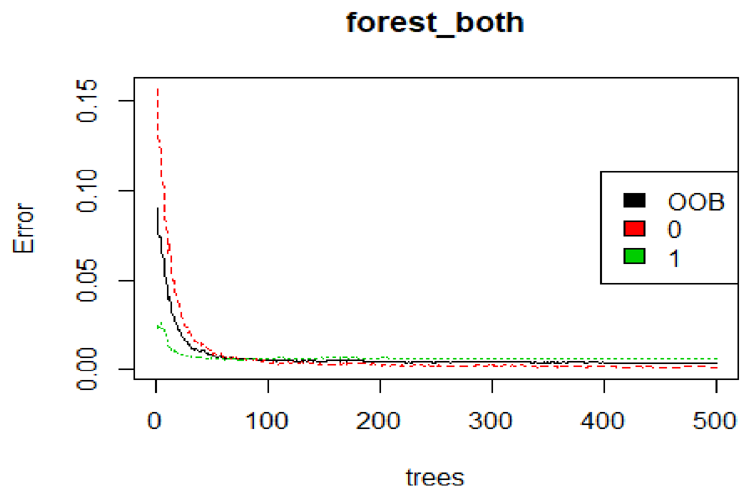
```
plot(forest_over)
legend(x = "right",legend = c( "OOB","0","1" ),fill = 1:3)
```



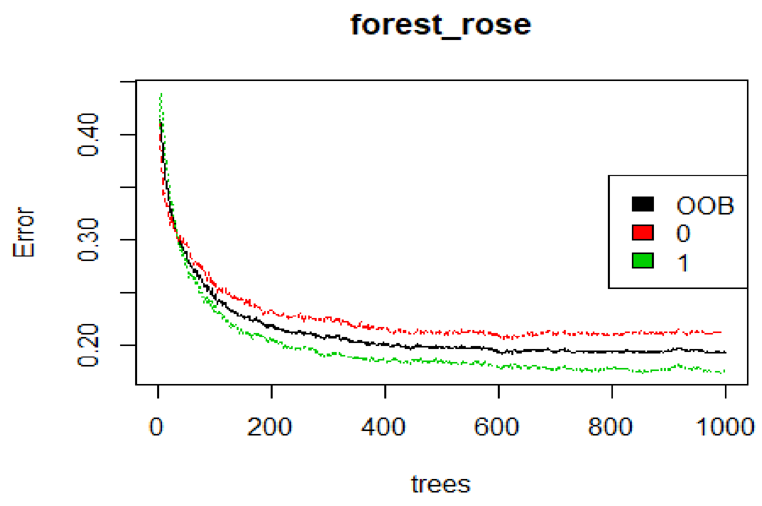
```
print(forest_under)

##
## Call:
##  randomForest(formula = y ~ ., data = under, ntree = 1000, mtry = 12,      nodesize = 
1, sampsize = 1300)
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 12
##
##         OOB estimate of  error rate: 15.48%
## Confusion matrix:
##     0   1 class.error
## 0 648 144   0.1818182
## 1 105 712   0.1285190

plot(forest_under)
legend(x = "right",legend = c( "OOB","0","1" ),fill = 1:3)
```



```
print(forest_both)

##
## Call:
##  randomForest(formula = y ~ ., data = both, ntree = 500)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 8
##
##         OOB estimate of  error rate: 0.36%
## Confusion matrix:
##      0    1 class.error
## 0 3868    5 0.001290989
## 1   23 3924 0.005827211

plot(forest_both)
legend(x = "right",legend = c( "OOB","0","1" ),fill = 1:3)
```



```
print(forest_rose)

##
## Call:
##  randomForest(formula = y ~ ., data = rose, ntree = 1000)
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 8
##
##         OOB estimate of  error rate: 19.39%
## Confusion matrix:
##      0    1 class.error
## 0 3080  826   0.2114695
## 1  690 3224   0.1762902

plot(forest_rose)
legend(x = "right",
       legend = c( "OOB","0","1" ),fill = 1:3)
```



use the r package caret to creat the confusion Matrix of these models.

```
# load caret package
library(caret)

## Loading required package: lattice

# Confusion Matrix and Statistics of forest_model
confusionMatrix(predict(forest_model, learn_test), learn_test$y, positive = '1')

## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2332  273
##          1    0    1
##
##                Accuracy : 0.8952
##                  95% CI : (0.8828, 0.9067)
##     No Information Rate : 0.8949
##     P-Value [Acc > NIR] : 0.4906
##
##                   Kappa : 0.0065
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.0036496
##             Specificity : 1.0000000
##          Pos Pred Value : 1.0000000
##          Neg Pred Value : 0.8952015
##              Prevalence : 0.1051420
##          Detection Rate : 0.0003837
##    Detection Prevalence : 0.0003837
##       Balanced Accuracy : 0.5018248
##
##        'Positive' Class : 1
##
```

This model fitted with raw training data has very high Specificity but very low Sensitivity, since the data is severely imbalanced, this result is not surprising， but if we were interested in predicting, “1”s, this model is not useful, though the total accuracy is 95%, the balanced accuracy is only 0.510949, this is an example of how imbalanced data influence the model evaluation.

```
# Confusion Matrix and Statistics of forest_over
confusionMatrix(predict(forest_over, learn_test), learn_test$y, positive = '1')

## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2332  270
##          1    0    4
##
##                Accuracy : 0.8964
##                  95% CI : (0.8841, 0.9078)
##     No Information Rate : 0.8949
##     P-Value [Acc > NIR] : 0.4147
##
##                   Kappa : 0.0258
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.014599
##             Specificity : 1.000000
##          Pos Pred Value : 1.000000
##          Neg Pred Value : 0.896234
##              Prevalence : 0.105142
##          Detection Rate : 0.001535
##    Detection Prevalence : 0.001535
##       Balanced Accuracy : 0.507299
##
##        'Positive' Class : 1
##
```

The Sensitivity is also low in this model, it looks like oversampling make the model over fitted. Balanced accuracy is only 0.510949.

```
# Confusion Matrix and Statistics of forest_under
confusionMatrix(predict(forest_under, learn_test), learn_test$y, positive = '1')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1944   28
##          1  388  246
##
##                Accuracy : 0.8404
##                  95% CI : (0.8257, 0.8542)
##     No Information Rate : 0.8949
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.463
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.8978
##             Specificity : 0.8336
##          Pos Pred Value : 0.3880
##          Neg Pred Value : 0.9858
##              Prevalence : 0.1051
##          Detection Rate : 0.0944
##    Detection Prevalence : 0.2433
##       Balanced Accuracy : 0.8657
##
##        'Positive' Class : 1
##
```

Sensitivity and specificity are both improved, Balanced Accuracy is 0.886, it seems this model performs better than previous ones.

```
confusionMatrix(predict(forest_both, learn_test), learn_test$y, positive = '1')

## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2326  216
##          1    6   58
##
##                Accuracy : 0.9148
##                  95% CI : (0.9034, 0.9253)
##     No Information Rate : 0.8949
##     P-Value [Acc > NIR] : 0.0003633
##
##                   Kappa : 0.316
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.21168
##             Specificity : 0.99743
##          Pos Pred Value : 0.90625
##          Neg Pred Value : 0.91503
##              Prevalence : 0.10514
##          Detection Rate : 0.02226
##    Detection Prevalence : 0.02456
##       Balanced Accuracy : 0.60455
##
##        'Positive' Class : 1
##
```

Balanced Accuracy is only 0.60681, which is lower than the under moedl.

```
confusionMatrix(predict(forest_rose, learn_test), learn_test$y, positive = '1')

## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1974   32
##          1  358  242
##
##                Accuracy : 0.8503
##                  95% CI : (0.8361, 0.8638)
##     No Information Rate : 0.8949
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.4785
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.88321
##             Specificity : 0.84648
##          Pos Pred Value : 0.40333
##          Neg Pred Value : 0.98405
##              Prevalence : 0.10514
##          Detection Rate : 0.09286
##    Detection Prevalence : 0.23024
##       Balanced Accuracy : 0.86485
##
##        'Positive' Class : 1
##
```

the Sensitivity and Specificity are fairly high, and the Balanced Accuracy is 0.86, almost the same as under.

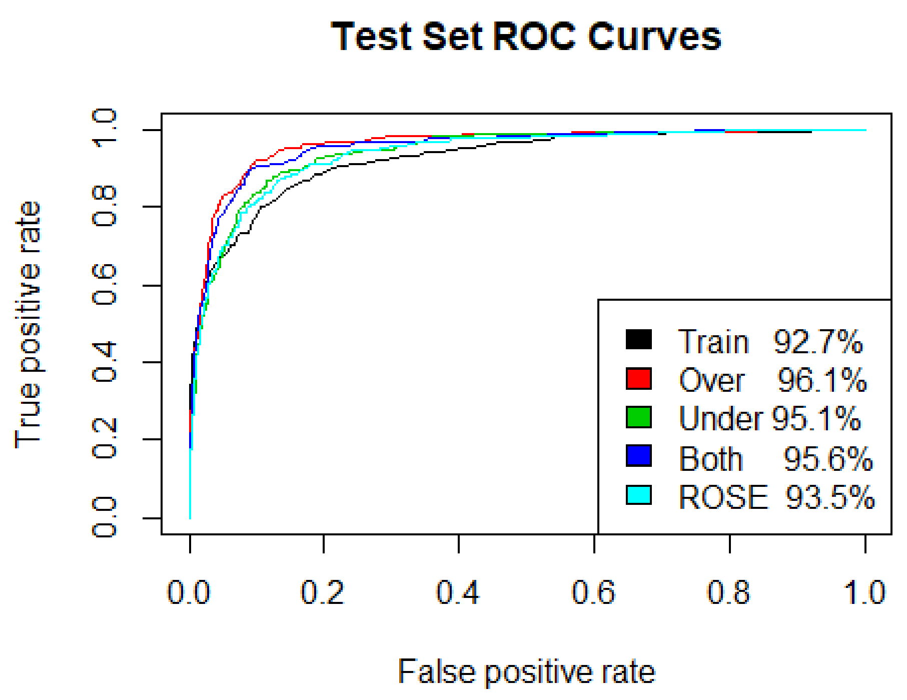
Plot the ROC curve of different models.

Predict the class of y in testing model

```
pred_model <- predict(object = forest_model, newdata = learn_test, n.trees = 1000,
                  type = "prob")

pred_over <-  predict(object = forest_over,  newdata = learn_test, n.trees = 1000,

                  type = "prob")

pred_under <- predict(object = forest_under, newdata = learn_test, n.trees = 1000,
                  type = "prob")

pred_both <-  predict(object = forest_both, newdata = learn_test,  n.trees = 1000,
                  type = "prob")

pred_rose <- predict(object = forest_rose,  newdata = learn_test,  n.trees = 1000,
                  type = "prob")
```

Use R package ROCR to plot ROC curve in one graph and compare their performance.

```
library(ROCR)
# List of predictions
preds_list <- list(pred_model[,2], pred_over[,2], pred_under[,2],pred_both[,2],
pred_rose[,2])

# List of actual values (same for all)
m <- length(preds_list)
actuals_list <- rep(list(learn_test2$y), m)

# Plot the ROC curves
pred <- prediction(preds_list, actuals_list)
rocs <- performance(pred, "tpr", "fpr")

plot(rocs, col = as.list(1:m), main = "Test Set ROC Curves",lwd = 1)
legend(x = "bottomright",
       legend = c("Train   92.7%", "Over    96.1%", "Under 95.1%", "Both     95.6%","ROSE
93.5%"),
       fill = 1:m)
```



Based on the confusion matrix and AUC, we can see all the resample strategies result in improved performance, three models with AUC over 95% are “over”, “under” and “both,” “over” and “both” shows extremely good performance in the training data but do not perform so well in the testing data， especially at the performance in predicting, “1”s correctly. Here we make a assumption that we are more interested in predicting “1”s, given that “under” model has the highest balanced accuracy and fairly high AUC, we will adopt under sampling for the data.

Build model with true learning data

```
## take under sample data
rf_data <- ovun.sample(y~., data = learn_clean, method = "under", seed = 357)$data

## fit the random forest data
rf_model <- randomForest(y ~ ., data = rf_data, ntree = 1000)

## print the model
print(rf_model)

##
## Call:
##  randomForest(formula = y ~ ., data = rf_data, ntree = 1000)
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 8
##
##         OOB estimate of  error rate: 13.85%
## Confusion matrix:
##     0   1 class.error
## 0 964 154   0.1377460
## 1 152 939   0.1393217
```

Make predictions with the random forest model

```
## make predictions
predict_clean$prediction <- predict(object = rf_model,
                   newdata = predict_clean,
                   n.trees = 1000,
                   type = "prob") [,2]

## perduce the final result
rf_final <- predict_clean %>%
  tibble::rownames_to_column("id") %>%
  select("id","prediction")
  #write.csv("predictions_QiweiMen_model1.csv",row.names = FALSE)
```

Part2: Logistic Regression Model

The second model I want to build is logistic regression model, which is a sample but powerful tool for binary classification problem.

feature selection

There are 78 variables in this dataset, some of them are useless for prediction, the first step is proper features for the model, here we use the `stepwise` function to select features.

fit the model with the raw data

```
# build a null model
fit0 <- glm(y~1, family = binomial(), data = learn_train)

# build a full model include all features
fitfull <- glm(y~., family = binomial(),data = learn_train)

# conduct feature selection
#fit_aic <- step(fit0,scope=list(upper=fitfull),data = learn_train, direction = "both",k
= 2,trace = 0)
fit_aic <- glm(formula = y ~ b49 + b00 + b53 + b12 + a01 + b01 + b18 + a00 + b45 + a16 +
b41 + b48 + b05 + b19 + a10 + b28 + a04 + b58 + b22 + b52 + a02 + a06 + b33 + b38 + b20 +
b50 + b25 + b03 +
a15 + b34 + b35 + b42 + b51 + b40 + b15 + b10 + b37 + b07 + b17 + b44 + a14 + b54 + b55,
family = binomial(), data = learn_train)
```

fit the model with resampled data

```
# build a null model
fit0_rose <- glm(y~1, family = binomial(), data = rose)

# build a full model include all features
fitfull_rose <- glm(y~., family = binomial(),data = rose)

# select feature
#fit_rose <- step(fit0_rose,scope=list(upper=fitfull_rose),data = rose, direction =
"both",k = 2,trace = 0)

fit_rose <- glm(formula = y ~ b49 + b53 + b00 + b45 + b12 + a00 + b01 + b18 + a05 + b04 +
b58 + b42 + b41 + b37 + a16 + b19 + b34 + a04 + b05 + a01 + b35 + a15 + b20 + b15 + a08 +
code + b10 + b21 + b28 + a10 + b46 + b38 + b16 + b30 + b50 + b26 + b44 + b54 + a11 + b09
+ a14 + b32 + a12 + a02 + a07 + b27 + a13 + b25 + a03 + b29 + b56 + b52, family =
binomial(), data = rose)

summary(fit_aic)

##
## Call:
## glm(formula = y ~ b49 + b00 + b53 + b12 + a01 + b01 + b18 + a00 +
##     b45 + a16 + b41 + b48 + b05 + b19 + a10 + b28 + a04 + b58 +
##     b22 + b52 + a02 + a06 + b33 + b38 + b20 + b50 + b25 + b03 +
##     a15 + b34 + b35 + b42 + b51 + b40 + b15 + b10 + b37 + b07 +
##     b17 + b44 + a14 + b54 + b55, family = binomial(), data = learn_train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.6807  -0.3769  -0.2122  -0.1051   4.0016
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.291e+01  2.079e+00   6.213 5.20e-10 ***
## b49         -6.557e-02  3.595e-03 -18.240  < 2e-16 ***
## b00         -6.078e-04  6.991e-05  -8.695  < 2e-16 ***
## b53          8.011e-02  6.949e-03  11.529  < 2e-16 ***
## b12         -1.337e-02  2.314e-03  -5.778 7.54e-09 ***
## a01         -4.976e-03  1.056e-03  -4.714 2.43e-06 ***
## b01          4.297e-02  2.927e-03  14.679  < 2e-16 ***
## b18          5.309e-02  4.423e-03  12.004  < 2e-16 ***
## a00         -1.325e-02  8.332e-04 -15.908  < 2e-16 ***
## b45         -4.216e-02  3.185e-03 -13.236  < 2e-16 ***
## a16          2.119e-02  2.068e-03  10.245  < 2e-16 ***
## b41         -3.401e-02  3.884e-03  -8.757  < 2e-16 ***
## b48          1.123e-02  1.177e-03   9.543  < 2e-16 ***
## b05         -1.005e-01  1.170e-02  -8.586  < 2e-16 ***
## b19          4.190e-02  4.284e-03   9.782  < 2e-16 ***
## a10          1.555e-01  1.981e-02   7.850 4.15e-15 ***
## b28          5.373e-02  7.188e-03   7.476 7.68e-14 ***
## a04          7.909e-03  2.067e-03   3.825 0.000131 ***
## b58Aug       1.420e+00  2.617e-01   5.426 5.77e-08 ***
## b58Dec      -8.964e+00  8.219e+02  -0.011 0.991298
## b58Feb       1.418e+00  9.546e-01   1.485 0.137548
## b58Jan      -1.479e+01  9.571e+02  -0.015 0.987673
## b58Jul       1.015e+00  2.469e-01   4.111 3.94e-05 ***
## b58Jun       5.207e-01  2.438e-01   2.136 0.032695 *
## b58Mar      -5.553e-01  5.565e-01  -0.998 0.318385
## b58May       1.589e-01  2.568e-01   0.619 0.536187
## b58Nov      -1.077e+01  2.594e+02  -0.042 0.966885
## b58Oct       2.714e+00  3.843e-01   7.062 1.64e-12 ***
## b58Sep       1.945e+00  2.922e-01   6.656 2.81e-11 ***
## b22         -1.715e-02  1.708e-03 -10.042  < 2e-16 ***
## b52          9.592e-03  1.352e-03   7.093 1.31e-12 ***
## a02         -2.077e-01  9.944e-02  -2.088 0.036773 *
## a06          3.565e-02  5.118e-03   6.967 3.25e-12 ***
## b33         -8.001e-03  1.336e-03  -5.990 2.10e-09 ***
## b38         -9.763e-03  3.600e-03  -2.712 0.006696 **
## b20          5.517e-03  2.924e-03   1.887 0.059177 .
## b50         -1.227e-02  1.680e-03  -7.302 2.83e-13 ***
## b25         -6.755e-03  8.398e-04  -8.043 8.75e-16 ***
## b03          1.449e-02  1.902e-03   7.619 2.56e-14 ***
## a15          1.979e-03  6.876e-04   2.878 0.004000 **
## b34Mon       3.161e+00  1.278e+00   2.473 0.013396 *
## b34Thu       7.780e-01  5.118e-01   1.520 0.128502
## b34Tue       2.027e+00  6.101e-01   3.322 0.000894 ***
## b34Wed       1.326e+00  5.264e-01   2.518 0.011803 *
## b35         -1.164e+01  6.477e+00  -1.797 0.072329 .
## b42         -9.612e-02  2.571e-02  -3.739 0.000185 ***
## b51         -2.707e-02  6.325e-03  -4.280 1.87e-05 ***
## b40          2.821e-02  1.326e-02   2.128 0.033365 *
## b15         -6.284e-03  3.633e-03  -1.730 0.083648 .
## b10          3.669e-02  9.097e-03   4.033 5.50e-05 ***
## b37         -9.536e-03  2.478e-03  -3.848 0.000119 ***
## b07         -2.710e-03  8.550e-04  -3.169 0.001528 **
## b17          1.176e-02  3.770e-03   3.120 0.001807 **
## b44         -1.313e-02  7.691e-03  -1.707 0.087831 .
## a14         -6.472e-02  3.910e-02  -1.655 0.097833 .
## b54          3.451e-03  1.759e-03   1.962 0.049767 *
## b55          2.397e-02  1.684e-02   1.423 0.154619
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 5236.4  on 7819  degrees of freedom
## Residual deviance: 3391.5  on 7763  degrees of freedom
## AIC: 3505.5
##
## Number of Fisher Scoring iterations: 14
```

```
summary(fit_rose)
```

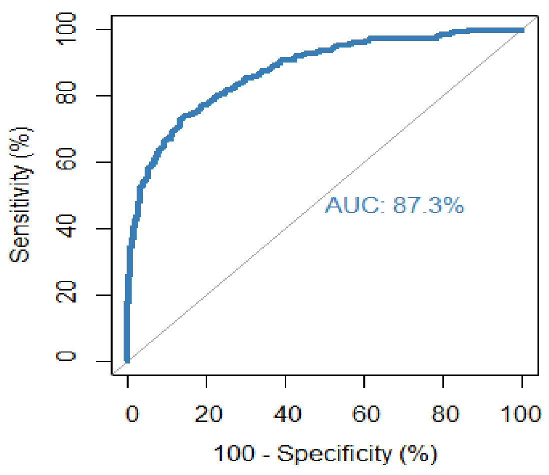
```
##
## Call:
## glm(formula = y ~ b49 + b53 + b00 + b45 + b12 + a00 + b01 + b18 +
##     a05 + b04 + b58 + b42 + b41 + b37 + a16 + b19 + b34 + a04 +
##     b05 + a01 + b35 + a15 + b20 + b15 + a08 + code + b10 + b21 +
##     b28 + a10 + b46 + b38 + b16 + b30 + b50 + b26 + b44 + b54 +
##     a11 + b09 + a14 + b32 + a12 + a02 + a07 + b27 + a13 + b25 +
##     a03 + b29 + b56 + b52, family = binomial(), data = rose)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.6033  -0.9119   0.1408   0.9113   2.4535
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.781e-01  7.752e-01  -0.359 0.719813
## b49         -2.143e-02  1.156e-03 -18.543  < 2e-16 ***
## b53          4.120e-02  2.643e-03  15.585  < 2e-16 ***
## b00         -2.342e-04  2.144e-05 -10.921  < 2e-16 ***
## b45         -1.012e-02  1.149e-03  -8.810  < 2e-16 ***
## b12         -8.867e-03  8.512e-04 -10.417  < 2e-16 ***
## a00          -1.465e-03  1.711e-04  -8.562  < 2e-16 ***
## b01           8.977e-03  9.119e-04   9.845  < 2e-16 ***
## b18           1.013e-02  1.237e-03   8.188 2.66e-16 ***
## a05          -8.925e-03  9.782e-04  -9.123  < 2e-16 ***
## b04           8.234e-03  9.633e-04   8.548  < 2e-16 ***
## b58Aug        2.687e-01  1.427e-01   1.883 0.059685 .
## b58Dec       -1.389e+01  8.827e+02  -0.016 0.987447
## b58Feb        2.438e+00  8.281e-01   2.945 0.003234 **
## b58Jan       -1.404e+01  6.176e+02  -0.023 0.981860
## b58Jul        1.145e-01  1.376e-01   0.832 0.405640
## b58Jun       -1.572e-01  1.387e-01  -1.133 0.257105
## b58Mar       -5.510e-02  2.885e-01  -0.191 0.848543
## b58May       -4.630e-02  1.484e-01  -0.312 0.755016
## b58Nov       -1.430e+01  1.801e+02  -0.079 0.936700
## b58Oct        6.169e-01  2.149e-01   2.871 0.004096 **
## b58Sep        1.959e-01  1.573e-01   1.245 0.213125
## b42          -6.295e-02  8.275e-03  -7.607 2.79e-14 ***
## b41          -1.194e-02  1.431e-03  -8.344  < 2e-16 ***
## b37           4.115e-03  8.242e-04   4.993 5.95e-07 ***
## a16           3.864e-03  6.557e-04   5.893 3.79e-09 ***
## b19           9.794e-03  1.432e-03   6.841 7.84e-12 ***
## b34Mon        1.686e+00  7.776e-01   2.169 0.030108 *
## b34Thu        4.106e-01  2.764e-01   1.486 0.137386
## b34Tue        1.425e+00  3.215e-01   4.433 9.31e-06 ***
## b34Wed        7.480e-01  2.777e-01   2.693 0.007071 **
## a04           4.558e-03  7.248e-04   6.289 3.20e-10 ***
## b05          -2.411e-02  3.924e-03  -6.145 8.00e-10 ***
## a01          -2.093e-03  3.964e-04  -5.281 1.29e-07 ***
## b35          -1.135e+01  1.984e+00  -5.720 1.06e-08 ***
## a15           8.904e-04  1.325e-04   6.722 1.80e-11 ***
## b20           5.238e-03  8.327e-04   6.291 3.15e-10 ***
## b15           6.891e-03  1.367e-03   5.040 4.66e-07 ***
## a08          -5.296e-03  1.153e-03  -4.593 4.37e-06 ***
## code1000001 -1.559e-02  4.142e-01  -0.038 0.969974
## code1000002  3.232e-01  4.099e-01   0.788 0.430435
## code1000003  1.379e-01  4.114e-01   0.335 0.737429
## code1000005 -2.366e-01  4.428e-01  -0.534 0.593026
## code1000006 -5.371e-01  8.209e-01  -0.654 0.512884
## code1000007  5.715e-02  4.892e-01   0.117 0.907002
## code1000008  6.056e-01  4.225e-01   1.433 0.151763
## code1000009  1.496e-01  4.098e-01   0.365 0.715070
## b10           1.015e-02  2.658e-03   3.819 0.000134 ***
## b21East      -5.420e-01  1.713e-01  -3.164 0.001555 **
## b21West       1.990e-01  9.678e-02   2.057 0.039730 *
## b28           8.160e-03  2.337e-03   3.492 0.000480 ***
## a10           2.986e-02  7.238e-03   4.125 3.71e-05 ***
## b46           1.175e-02  3.552e-03   3.307 0.000944 ***
## b38          -3.955e-03  1.179e-03  -3.355 0.000795 ***
## b16          -2.917e-03  1.045e-03  -2.790 0.005273 **
## b30           6.312e-03  2.365e-03   2.669 0.007614 **
## b50          -1.273e-03  6.002e-04  -2.121 0.033899 *
## b26           1.289e-02  5.432e-03   2.374 0.017617 *
## b44          -7.372e-03  3.406e-03  -2.164 0.030441 *
## b54          -1.174e-03  4.636e-04  -2.532 0.011342 *
## a11          -4.703e-03  1.949e-03  -2.413 0.015840 *
## b09         -9.870e-03  4.787e-03  -2.062 0.039211 *
## a14         -3.735e-02  1.649e-02  -2.265 0.023500 *
## b32          7.120e-03  3.688e-03   1.931 0.053506 .
## a12          4.324e-02  2.257e-02   1.916 0.055403 .
## a02         -7.746e-02  3.448e-02  -2.246 0.024695 *
## a07         -1.468e-01  7.891e-02  -1.861 0.062764 .
## b27         -2.211e-02  1.258e-02  -1.757 0.078882 .
## a13          5.439e-03  3.366e-03   1.616 0.106102
## b25          2.339e-04  1.368e-04   1.709 0.087433 .
## a03         -3.745e-03  2.520e-03  -1.486 0.137316
## b29         -8.693e-03  5.552e-03  -1.566 0.117455
## b56          1.161e-02  7.554e-03   1.537 0.124390
## b52          6.656e-04  4.573e-04   1.456 0.145522
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 10840.8  on 7819  degrees of freedom
## Residual deviance:  8576.9  on 7746  degrees of freedom
## AIC: 8724.9
##
## Number of Fisher Scoring iterations: 13
```

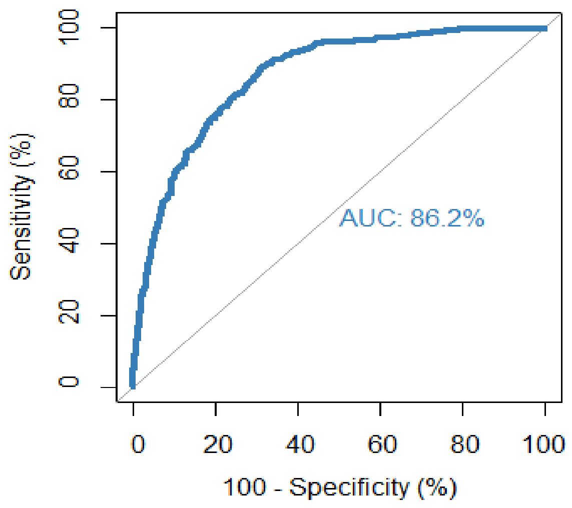
make prediction on the testing data with both models

```
learn_test3 <- learn_test2 %>%
  filter(code != 1000004)
lr_predict <- predict(fit_aic, learn_test3, type = "response")
lr_predict_rose <-  predict(fit_rose, learn_test3, type = "response")
```

Plot ROC and AUC

```
par(pty = "s")
ROC_lr <- roc(learn_test3$y, lr_predict, plot = TRUE,legacy.axes=TRUE ,percent =
TRUE,print.auc = TRUE, col="#377eb8", lwd = 4)

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases
```





```
ROC_lr_rose <- roc(learn_test3$y, lr_predict_rose, plot = TRUE,legacy.axes=TRUE ,percent
= TRUE,print.auc = TRUE, col="#377eb8", lwd = 4)

## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

The AUC of these two models are very close, which means that the resampling doesn't improve the overall performance of the model too much, However,the resampling model gives a more balanced performance in sensitivity and specificity, this is also indicated in the confusion matrix, the resampled model has s slightly higher Balanced Accuracy,so we decided to go with the second method.

```
# Confusion Matrix
yhat_lr <- as.factor(ifelse(lr_predict > 0.5,1,0))
confusionMatrix(yhat_lr, as.factor(learn_test3$y), positive = '1')

## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2296  166
##          1   36  107
##
##                Accuracy : 0.9225
##                  95% CI : (0.9115, 0.9324)
##     No Information Rate : 0.8952
##     P-Value [Acc > NIR] : 1.335e-06
##
##                   Kappa : 0.4767
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.39194
##             Specificity : 0.98456
##          Pos Pred Value : 0.74825
##          Neg Pred Value : 0.93258
##              Prevalence : 0.10480
##          Detection Rate : 0.04107
##    Detection Prevalence : 0.05489
##       Balanced Accuracy : 0.68825
##
##        'Positive' Class : 1
##

# Confusion Matrix
yhat_rose <- as.factor(ifelse(lr_predict_rose > 0.5,1,0))
confusionMatrix(yhat_rose, as.factor(learn_test3$y), positive = '1')

## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1763   54
##          1  569  219
##
##                Accuracy : 0.7608
##                  95% CI : (0.744, 0.7771)
##     No Information Rate : 0.8952
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.3046
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.80220
##             Specificity : 0.75600
##          Pos Pred Value : 0.27792
##          Neg Pred Value : 0.97028
##              Prevalence : 0.10480
##          Detection Rate : 0.08407
##    Detection Prevalence : 0.30250
##       Balanced Accuracy : 0.77910
##
##        'Positive' Class : 1
##
```

Build model with learning data

```
predict_rose <- ROSE(y~., data = learn_clean,seed = 105)$data

# build a null model
lr_model_final <- glm(formula =
    y ~ b49 + b53 + b00 + b45 + b12 + a00 + b01 + b18 +
    a05 + b04 + b58 + b42 + b41 + b37 + a16 + b19 + b34 + a04 +
    b05 + a01 + b35 + a15 + b20 + b15 + a08 + code + b10 + b21 +
    b28 + a10 + b46 + b38 + b16 + b30 + b50 + b26 + b44 + b54 +
    a11 + b09 + a14 + b32 + a12 + a02 + a07 + b27 + a13 + b25 +
    a03 + b29 + b56 + b52, family = binomial(), data = predict_rose)

predict_clean2 <- subset(predict_clean, select = -prediction )
## make predictions
predict_clean2$prediction <- predict(object = lr_model_final,
                 newdata = predict_clean2,
                 type = "response")

## perduce the final result
lr_final <- predict_clean2 %>%
  tibble::rownames_to_column("id") %>%
  select("id","prediction")
  #write.csv("predictions_QiweiMen_model2.csv",row.names = FALSE)
```

Summarize of questions mentioned in the instruction:

- **How did you handle the class imbalance?**

  Firstly, I Choose a proper metrics like AUC and Balanced Accuracy to evaluate the performance of the data.

  Secondly, I apply some resampleing techniques like over-samples, under-sample and Randomly Over-Sample to fix the imbalanced data.

- **Had a metric not been imposed, what metric(s) would you have considered using for this model? Why?**

  I would like to choose the Balanced Accuracy as the metric, since we don't know the purpose of this project, maybe we interested in predicting 0s, or we interested in predicting 1s, though sometimes predicting one class successfully is more important, a balanced model can give a more stable and reliable model. For Example, the 1s in our data is the label of person who has certain diseases, our procedure is a diagnose procedure, a balance model with fairly highly sensitive and specificity means it has low Misdiagnosis rates and low missed diagnosis rate. So when I have to choose a model, if they have similar AUC, I would choose the one with higher Balanced Accuracy.

- **What modeling choices were impacted by scale? Put otherwise, if the learning set's sample size had been (i) 20 times greater or (ii) 20 times smaller, briefly describe high-level changes in approach you might have considered.**

  In my opinion, the performance of tree-based model like random forest will be improved gradually with the growth of sample size, since with more observations, we can build more complex trees and find subtle patterns.

  The situation for logistic regression is more complicated, on the one hand, a sample has to be large enough to cover the variability of features within the study area, and to yield stable and reproducible results; on the other hand, the sample must not be too large, because a large sample is likely to violate the assumption of independent observations, also as the growth of sample size, the accuracy of the model will not improve too much.

  If I have a 20 times greater dataset, I would not consider methods like logistic regression, because the sample is too large and with 78 variables, the feature selection will be tedious. A tree-based model will be more proper.

  If I have a 20 times smaller data(about 600), random forest will not be a good choice, since the small sample size will limit the power of random forest to find patterns. As I mentioned, Sample size calculation for logistic regression is a complex problem, but based on the work of Peduzzi et al. (1996). Let p be the smallest of the proportions of negative or positive cases in the population and k the number of covariates (the number of independent variables), then the minimum number of cases to include is: N = 10 k / p. I our case, we have 78 covariates to include in the model and the proportion of positive cases in the population is 0.1 . The minimum number of cases required are

  N = 10 x 78 / 0.20 = 7800

  so logistic regression is not a good choice, we can try other methods like one decision tree, simple Bayesian or KNN.

- **If data augmentation had been permitted for this project, what data might you have considered incorporating?**

  I don't know the actual purpose of this model, and I don't know what features have already been included in the data, so my answer is I don't know, maybe after I know more about this project, I can figure out some potential helpful features.

- **Given another week to work on this modeling exercise, what would you want to try next towards better understanding this dataset and improving predictive performance?**

  I will try model tuning of the random forest data, conduct a grid search on the combination of parameters like ntrees, mtry, nod.size, sample.size.

Find a more elegant way to deal with missing data, like conduct imputation using with algorithms.

Try other methods like bagging, boosting, simple Bayesian and KNN, compare their performance.

Tune the logistic regression in the step selection step, for example, see if the performance will be improved if we add some interaction or quadratic terms.

Try to do this project on with Python, because Python have more tools for classification problems.

• **Compare and contrast both submitted models with regards to strengths, weaknesses, and performance**.

In summary, the random forest model has a better performance between these two models, AUC is about 95%, both sensitivity and specificity are over 85% of the test. The logistic regression model has a lower AUC of 86% which is also a good performance.

The strength of random forest model is that it has a very good performance in classification problems, over 95% AUC and fairly high balanced accuracy is amazing. And it's also a very simple model, not too many parameters for tuning, and very friendly for beginners.

The weakness of random forest model is that it usually needs large sample size, if we do not have so many obs, it maybe not a good choice. The other weakness is that it's more sensitive to the imbalanced data, if we don't resample the data, the sensitivity is extremely low.