

# Predict LendingClub's Loan Data

- Synopsis
- Explore Data
- Problem Formulation
- Model Building Strategy
- Feature Engineering
- Model Training
  - Logistic Regression
  - SVM
  - RandomForest
  - Extreme Gradient Boosting
  - Averaging Ensemble
- Model Comparison
- Conclusion

## Synopsis

In this report, we attempt to predict the risk of the loan being default based on the past loan data. We obtained data from LendingClub's website(<https://www.lendingclub.com/info/download-data.action> (<https://www.lendingclub.com/info/download-data.action>)). We use loan data from year 2012-2014 as training and cross validation set and loan data from year 2015 as a testing set. We also compare our investment performance against the baseline algorithm. We found that, among multiple machine learning algorithms that we tried, Logistic Regression provided a reasonable trade-off performance, and a higher return than the naive loan picking strategy can be achieved.

## Explore Data

The loan dataset can be downloaded from <https://www.lendingclub.com/info/download-data.action> (<https://www.lendingclub.com/info/download-data.action>). We renamed the file of 2012-2013 loan data to `loan_2012_2013.csv`, 2014 loan data to `loan_2014.csv` and 2015 loan data to `loan_2015.csv`. We stored them on the same directory of the script. The data dictionary can be found at the same url.

There are 7 loan status:

`Charged Off`, `Current`, `Default`, `Fully Paid`, `In Grace Period`, `Late (16-30 days)`, `Late (31-120 days)`. We consider `Late (31-120 days)`, `Default`, `Charged Off` as a default loan and `Fully Paid` as a desirable loan and ignore everything else. We can create shell script to filter such loan instead of loading the entire files.

We create `extract.sh` to perform the task:

```
#!/bin/sh

input_file=$1
output_file=$2
sed '2!d' $input_file > $output_file
awk -F "','" 'BEGIN {OFS=","} { if (toupper($17) == "FULLY PAID" || toupper($17) == "LATE (31-120 DAYS)" || toupper($17) == "DEFAULT" || toupper($17) == "CHARGED OFF") print }' $input_file >> $output_file
```

Execute the following command in the command line to filter only target statuses:

```
sh extract.sh loan_2012_2013.csv loan_2012_2013.extract.csv
sh extract.sh loan_2014.csv loan_2014.extract.csv
sh extract.sh loan_2015.csv loan_2015.extract.csv
```

We can now use \*.extract.csv as our dataset

```
#load all required package
library(dplyr)
library(ggplot2)
library(caret)
library(rpart)
library(rpart.plot)
library(randomForest)
library(corrplot)
library(e1071)
library(xgboost)
library(stringr)
library(lubridate)
library(tm)
library(rms)
library(glmnet)
library(pROC)
library(doMC)
library(kernlab)
#make the analysis reproducible
set.seed(100)

setwd(".")
```

We firstly investigate overall data

```
data_a = read.csv("loan_2012_2013.extract.csv", stringsAsFactors=FALSE)
data_b = read.csv("loan_2014.extract.csv", stringsAsFactors=FALSE)
data = read.csv("loan_2015.extract.csv", stringsAsFactors=FALSE)
data = rbind(data, data_a, data_b)
rm(data_a, data_b)
```

We take a look at the number of each loan\_status so far

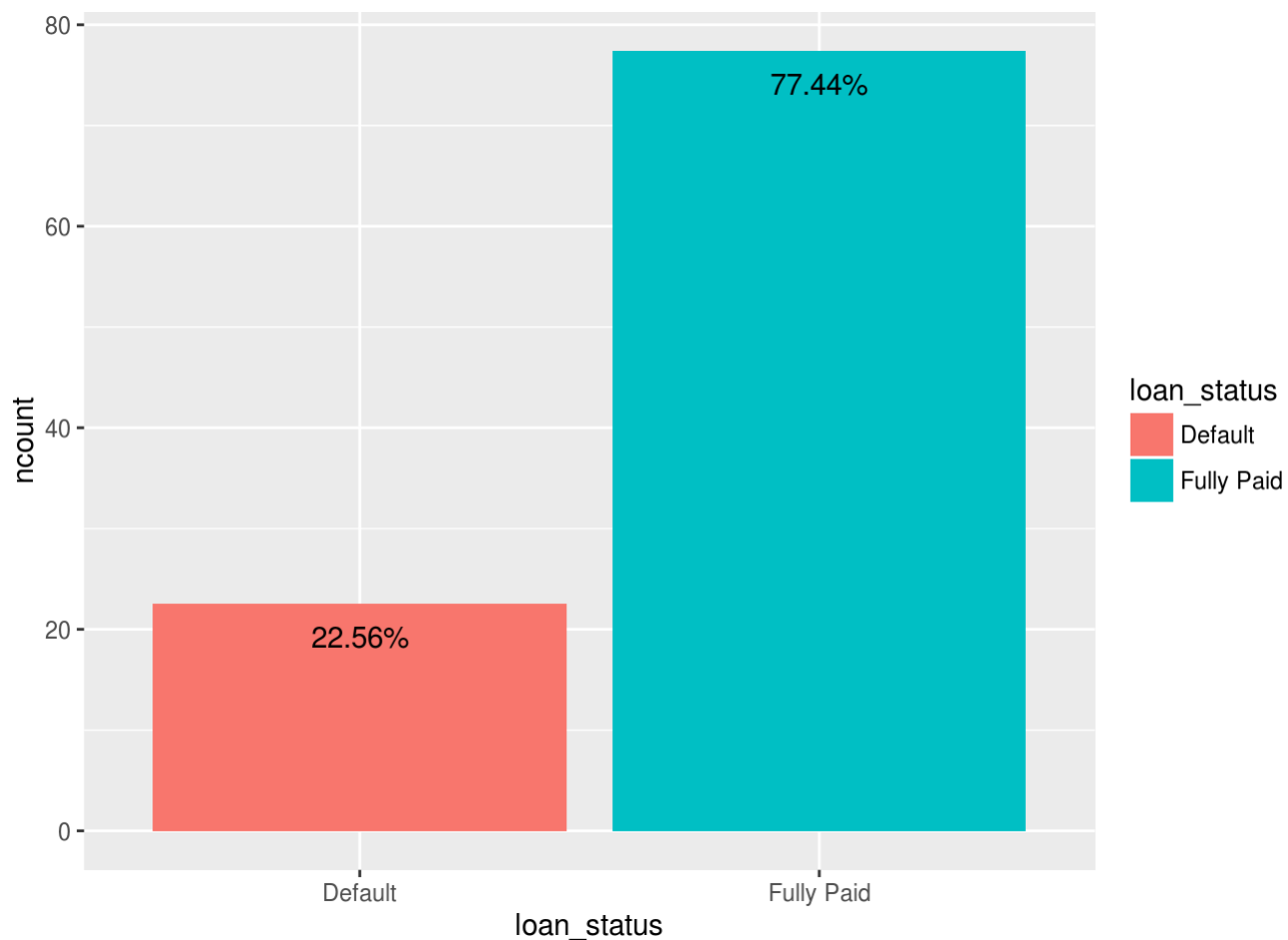
```
data %>% group_by(loan_status) %>% summarise(count = n())
```

```
## Source: local data frame [4 x 2]
##
##      loan_status  count
##      <chr>    <int>
## 1    Charged Off  57896
## 2      Default    203
## 3    Fully Paid  242869
## 4 Late (31-120 days) 12665
```

We categorize Charged Off , Default , and Late (31-120 days) to a single category: Default . The data is moderately imbalanced.

```
data$loan_status =
  ifelse(str_detect(data$loan_status,"Paid"),data$loan_status,"Default")

tmp = data %>% group_by(loan_status) %>% summarise(ncount = n())
tmp$ncount = 100 * tmp$ncount/nrow(data)
tmp$ncount_p = str_c(round(tmp$ncount,2),"%")
ggplot(tmp,aes(x=loan_status,y=ncount,fill=loan_status)) + geom_bar(stat="identity") +
  geom_text(aes(label=ncount_p),vjust = 2)
```



## Problem Formulation

The goal of this analysis is to produce a loan picking strategy that is superior to the naive one. Let's look at the return and default rate for each loan grade from 2012-2015.

```
data$int_rate = (as.numeric(gsub(pattern = "%",replacement = "",x = data$int_rate)))
#extract issued year
data$issue_y = as.numeric(sapply( data$issue_d ,function(x){str_split(x,"-")[[1]][2]}))

displayInterestByGrade <- function(dt){
  g1 = dt %>% filter(loan_status == "Default") %>% group_by(grade) %>% summarise(default_count = n())
  g2 = dt %>% group_by(grade) %>% summarise(count = n(),int_rate=mean(int_rate))
  g2 %>% left_join(g1) %>% mutate(default_rate = 100*default_count/count) %>% select(grade,count,default_count,int_rate,default_rate)
}

#year 2012
tmp0 = displayInterestByGrade(data %>% filter(issue_y==2012))
```

```
## Joining, by = "grade"
```

```
tmp0$year = 2012
#year 2013
tmp1 = displayInterestByGrade(data %>% filter(issue_y==2013))
```

```
## Joining, by = "grade"
```

```
tmp1$year = 2013
#year 2014
tmp2 = displayInterestByGrade(data %>% filter(issue_y==2014))
```

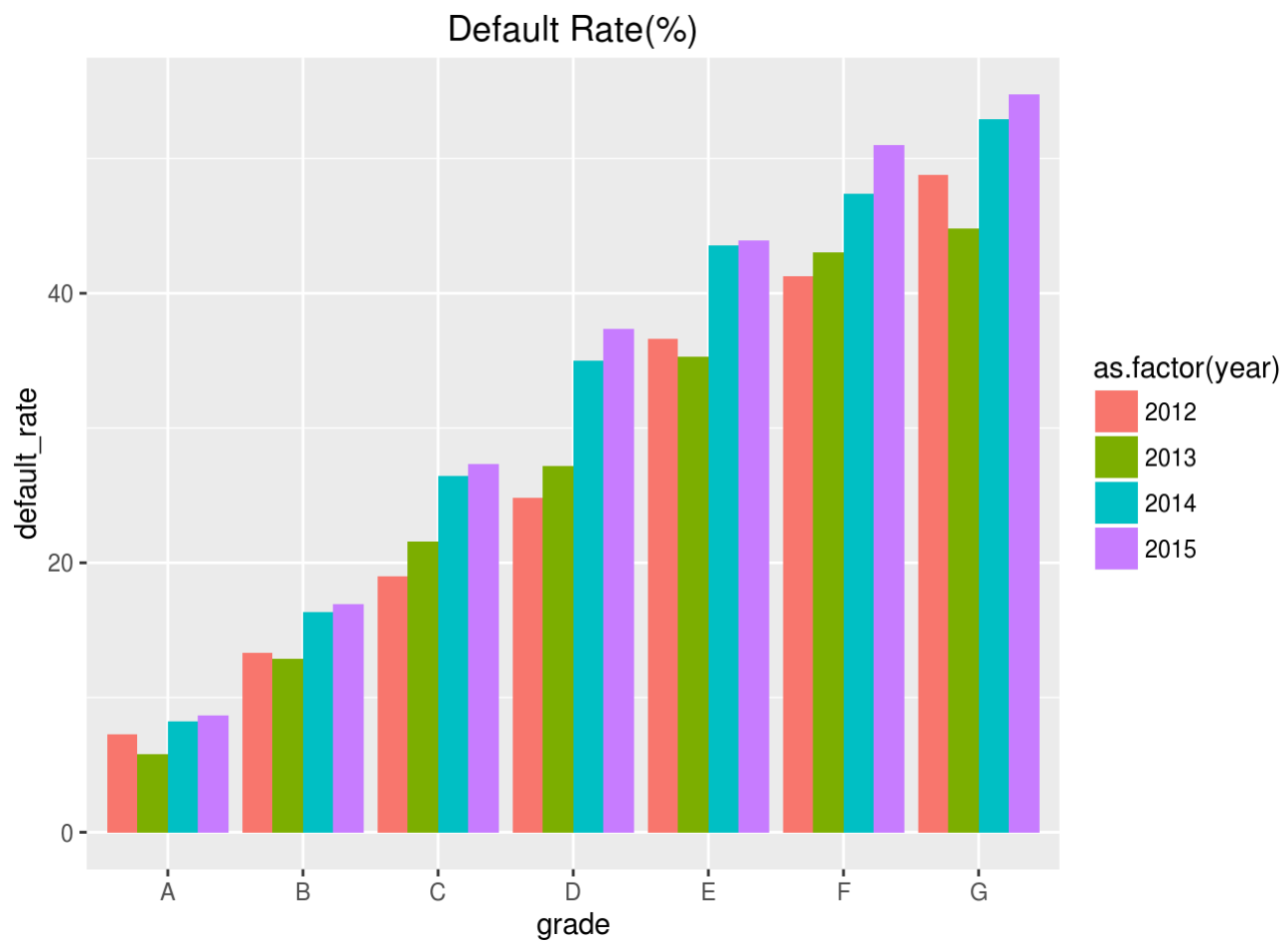
```
## Joining, by = "grade"
```

```
tmp2$year = 2014
#year 2015
tmp3 = displayInterestByGrade(data %>% filter(issue_y==2015))
```

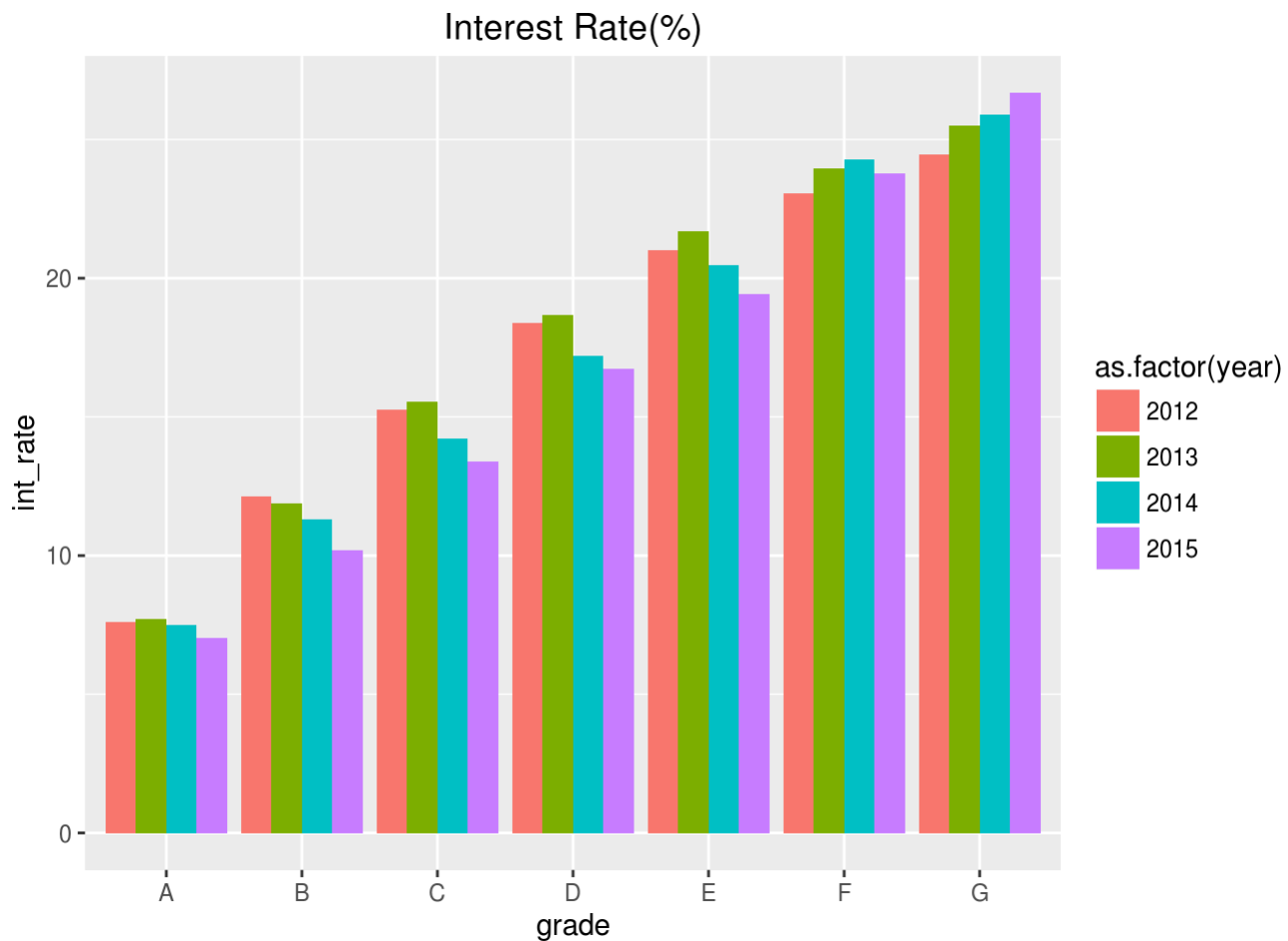
```
## Joining, by = "grade"
```

```
tmp3$year = 2015
tmp = rbind(tmp0,tmp1,tmp2,tmp3)

ggplot(tmp, aes(x=grade, y=default_rate,fill=as.factor(year))) + geom_bar(stat="identity",position="dodge") + ggtitle("Default Rate(%)")
```



```
ggplot(tmp, aes(x=grade, y=int_rate, fill=as.factor(year))) + geom_bar(stat="identity", position="dodge") + ggtitle("Interest Rate(%)")
```



```
rm(tmp, tmp0, tmp1, tmp2, tmp3)
```

As expected, riskier loans come with higher interest rates.

We can use `total_pymnt` (payment obtained from borrower) and `funded_amnt` (money lent) to calculate Return on Investment(ROI).

For year 2015, the ROI of our baseline investment strategy-Picking every loans-, is:

```
all_roi = sum((data %>% filter(issue_y==2015))$total_pymnt)/sum((data %>%
filter(issue_y==2015))$funded_amnt) - 1
all_roi
```

```
## [1] -0.1756042
```

or roughly -17.56%. Noted that we excluded `current` loans from our analysis(which a large amount of them typically ended up being paid-off), so negative ROI is as expected. Again, the higher the ROI is, the better.

We can look at ROI pertaining to each loan grade:

```

data$prediction = "Fully Paid"
createPerformanceTable <- function(dt){

  dt_pick = dt %>% filter(prediction == "Fully Paid")
  all_roi = sum(dt_pick$total_pymnt)/sum(dt_pick$funded_amnt) - 1

  temp_table = data.frame(grade=character(0),roi=numeric(0),percent_pick=numeric(0))
  for(g in c("A","B","C","D","E","F","G")){
    data_pick_grade = dt_pick %>% filter(grade==g)
    if(nrow(data_pick_grade)==0){
      temp_table = rbind(temp_table,data.frame(grade=g,roi=0,percent_pick=0))
    }
    else
    {
      data_grade = dt %>% filter(grade==g)
      roi = sum(data_pick_grade$total_pymnt)/sum(data_pick_grade$funded_amnt) - 1
      temp_table = rbind(temp_table,data.frame(grade=g,roi=roi,percent_pick=100 *
nrow(data_pick_grade)/nrow(data_grade)))
    }
  }

  temp_table = rbind(temp_table,data.frame(grade="ALL",roi=all_roi,percent_pick=100 *
nrow(dt_pick)/nrow(dt) ))

  return(temp_table)
}

baseline_table = createPerformanceTable(data %>% filter(issue_y==2015))
baseline_table

```

```

##   grade      roi percent_pick
## 1    A -0.02860544          100
## 2    B -0.08447862          100
## 3    C -0.16181754          100
## 4    D -0.23565662          100
## 5    E -0.28144831          100
## 6    F -0.34315720          100
## 7    G -0.35786380          100
## 8   ALL -0.17560423          100

```

The `baseline_table` served as a baseline performance when we evaluate our model

## Model Building Strategy

2012-2014 data will served as training & cross validation set and 2015 as a test set. We will put more emphasis on avoiding **default** loans since they can wreak havoc on our overall investment. We also take a note of percentage of loan picked as picking a small number of loans doesn't make a sizable investment.

## Feature Engineering

We explore some features in the dataset. We will create transformation model for the 2012-2014 data. We will later apply it to 2015 data.

```
data1 = data %>% filter(issue_y != 2015)
#look at number of row and column
dim(data1)
```

```
## [1] 246073    113
```

```
rm(data)
```

Looking at the data dictionary, we can identify and drop irrelevant features(features that do not appear at the issued time, or id field). `emp_title` maybe useful but it need to be effectively categorized.

```
discard_column = c("collection_recovery_fee", "emp_title",
                   "funded_amnt_inv", "id",
                   "installment", "last_credit_pull_d",
                   "last_fico_range_high", "last_fico_range_low",
                   "last_pymnt_amnt", "last_pymnt_d",
                   "loan_amnt", "member_id",
                   "next_pymnt_d", "num_tl_120dpd_2m",
                   "num_tl_30dpd", "out_prncp",
                   "out_prncp_inv", "recoveries",
                   "total_pymnt", "total_pymnt_inv",
                   "total_rec_int", "total_rec_late_fee",
                   "total_rec_prncp", "url",
                   "zip_code"
                   )

data1 = (data1[,!(names(data1) %in% discard_column)])
```

We also drop `grade` featrue as such data also carried in `sub_grade`

```
data1$grade = NULL
```

We drop features that contain too many NA values. We can list the raito of NA value for each feature:

```
tmp = sort(sapply(data1, function(x) sum(length(which(is.na(x)))))/nrow(data1),decreasin
g = TRUE)
```

Drop features that have more than 50% missing and take a look at what features that are still missing:

```
discard_column = names(tmp[tmp>0.5])
data1 = (data1[,!(names(data1) %in% discard_column)])

tmp = sort(sapply(data1, function(x) sum(length(which(is.na(x)))))/nrow(data1),decreasin
g = TRUE)
tmp[tmp>0]
```



```

##      mo_sin_old_il_acct      mths_since_recent_inq
##      0.13758925      0.11994815
##      pct_tl_nvr_dlq      avg_cur_bal
##      0.10749656      0.10719177
##      mo_sin_old_rev_tl_op      mo_sin_rcnt_rev_tl_op
##      0.10715113      0.10715113
##      tot_coll_amt      tot_cur_bal
##      0.10714707      0.10714707
##      total_rev_hi_lim      mo_sin_rcnt_tl
##      0.10714707      0.10714707
##      num_accts_ever_120_pd      num_actv_bc_tl
##      0.10714707      0.10714707
##      num_actv_rev_tl      num_bc_tl
##      0.10714707      0.10714707
##      num_il_tl      num_op_rev_tl
##      0.10714707      0.10714707
##      num_rev_accts      num_rev_tl_bal_gt_0
##      0.10714707      0.10714707
##      num_tl_90g_dpd_24m      num_tl_op_past_12m
##      0.10714707      0.10714707
##      tot_hi_cred_lim      total_il_high_credit_limit
##      0.10714707      0.10714707
##      num_bc_sats      num_sats
##      0.06212384      0.06212384
##      bc_util      percent_bc_gt_75
##      0.03882994      0.03843168
##      bc_open_to_buy      mths_since_recent_bc
##      0.03821630      0.03720441
##      acc_open_past_24mths      mort_acc
##      0.02899952      0.02899952
##      total_bal_ex_mort      total_bc_limit
##      0.02899952      0.02899952

```

```

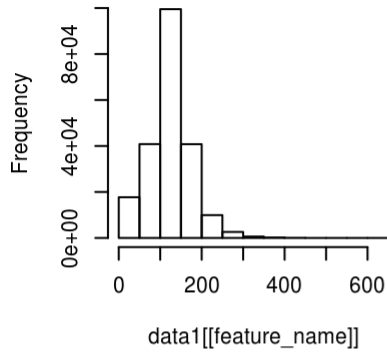
tmp = tmp[tmp>0]

par(mfrow=c(2,3))
for(feature_name in names(tmp)){
  hist(data1[[feature_name]],main = str_c(feature_name,"(missing=",100* round(as.numeric(tmp[feature_name]),2) ,"%") )
}

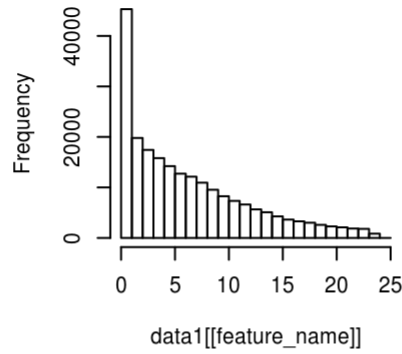
```



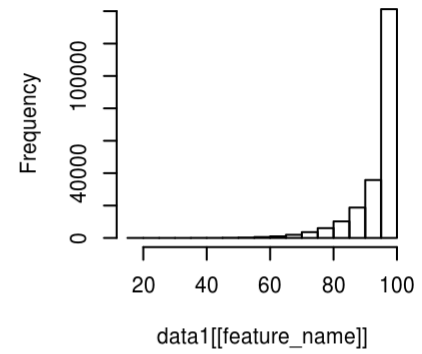
**mo\_sin\_old\_il\_acct(missing=14%)**



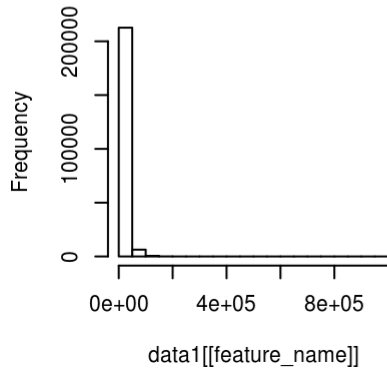
**mths\_since\_recent\_inq(missing=12%)**



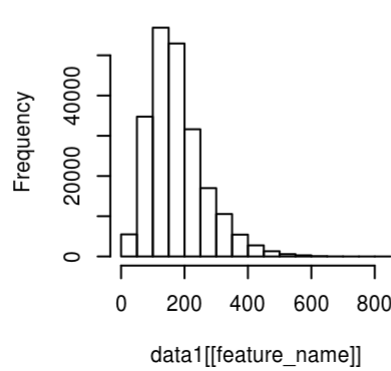
**pct\_tl\_nvr\_dlq(missing=11%)**



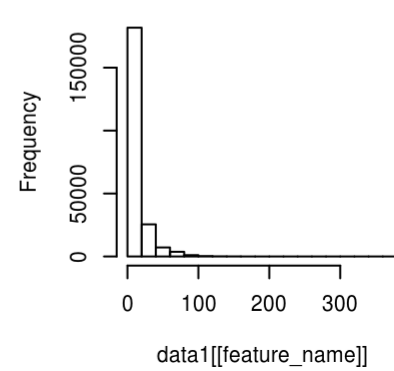
**avg\_cur\_bal(missing=11%)**



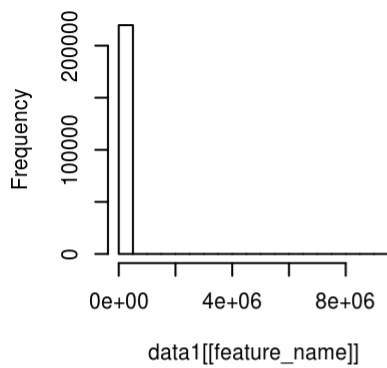
**mo\_sin\_old\_rev\_tl\_op(missing=11%)**



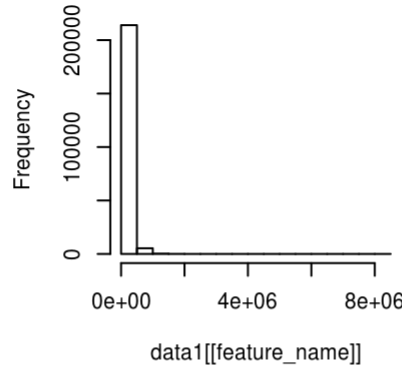
**mo\_sin\_rcnt\_rev\_tl\_op(missing=11%)**



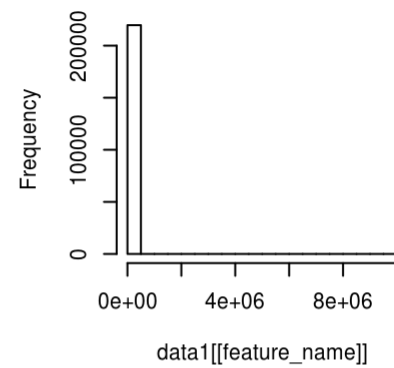
**tot\_coll\_amt(missing=11%)**



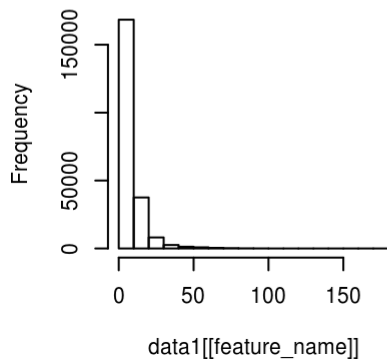
**tot\_cur\_bal(missing=11%)**



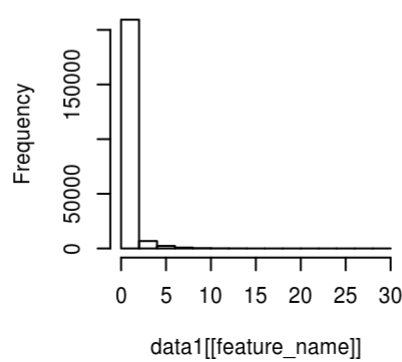
**total\_rev\_hi\_lim(missing=11%)**



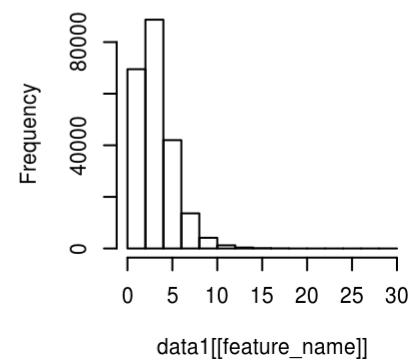
**mo\_sin\_rcnt\_tl(missing=11%)**



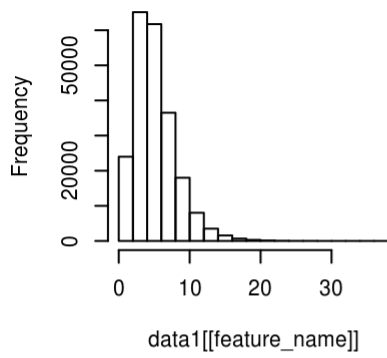
**num\_accts\_ever\_120\_pd(missing=11%)**



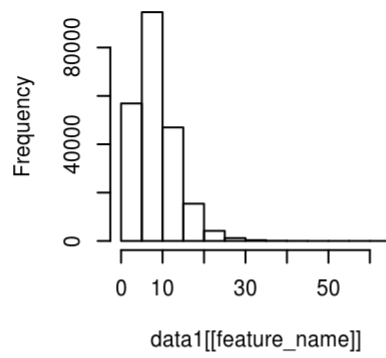
**num\_actv\_bc\_tl(missing=11%)**



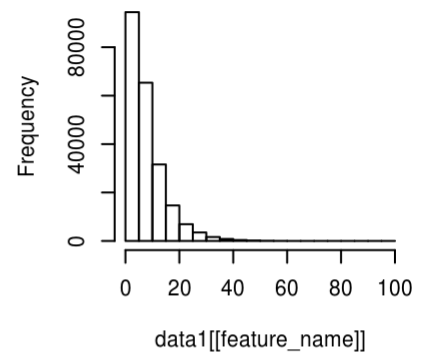
**num\_actv\_rev\_tl(missing=11%)**



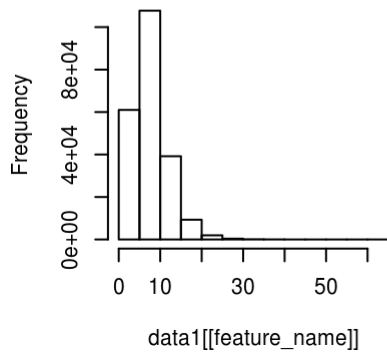
**num\_bc\_tl(missing=11%)**



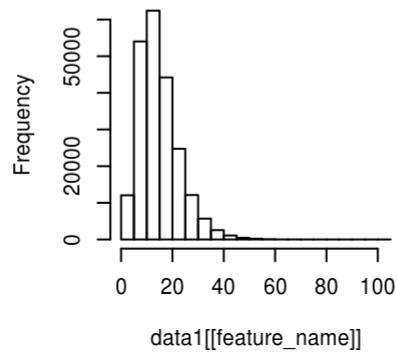
**num\_il\_tl(missing=11%)**



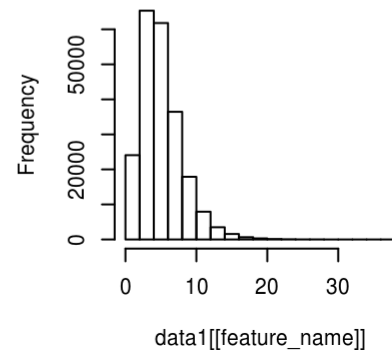
**num\_op\_rev\_tl(missing=11%)**



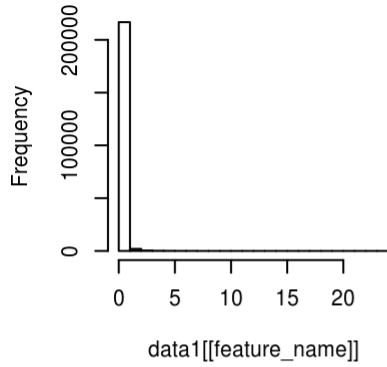
**num\_rev\_accts(missing=11%)**



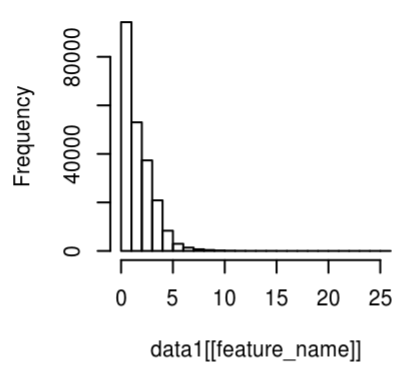
**num\_rev\_tl\_bal\_gt\_0(missing=11%)**



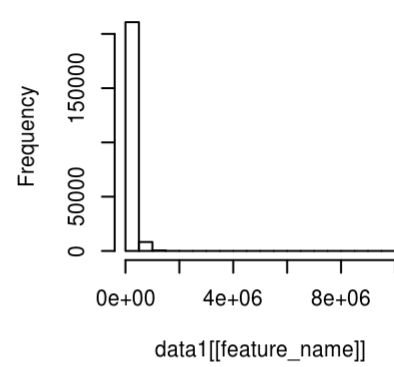
**num\_tl\_90g\_dpd\_24m(missing=11%**



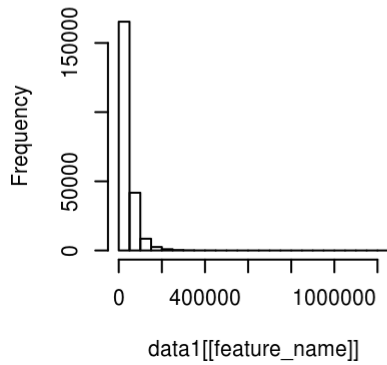
**num\_tl\_op\_past\_12m(missing=11%**



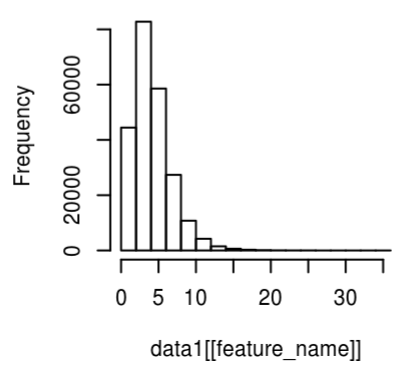
**tot\_hi\_cred\_lim(missing=11%)**



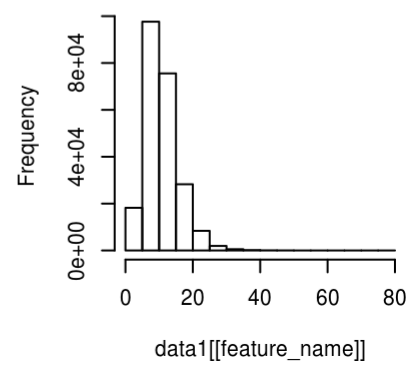
**total\_il\_high\_credit\_limit(missing=11%**



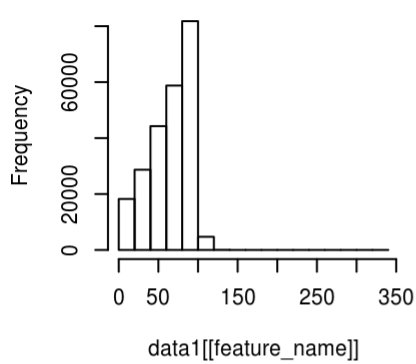
**num\_bc\_sats(missing=6%)**



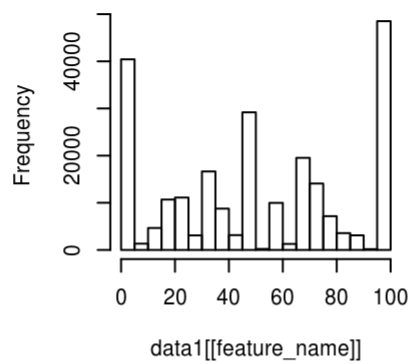
**num\_sats(missing=6%)**



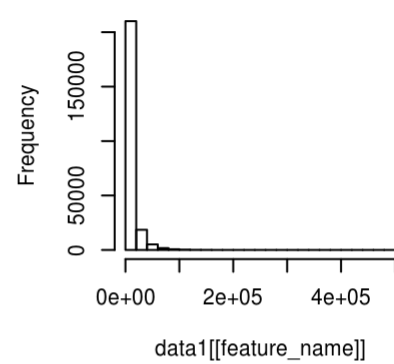
**bc\_util(missing=4%)**



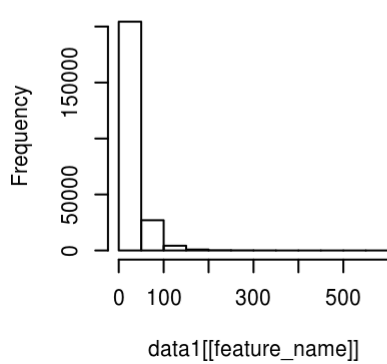
**percent\_bc\_gt\_75(missing=4%)**



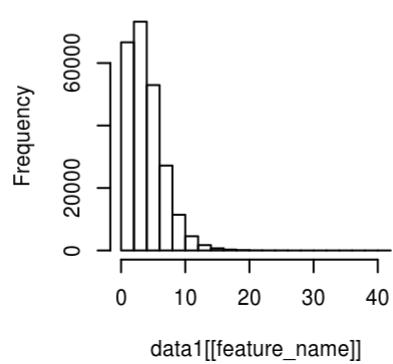
**bc\_open\_to\_buy(missing=4%)**



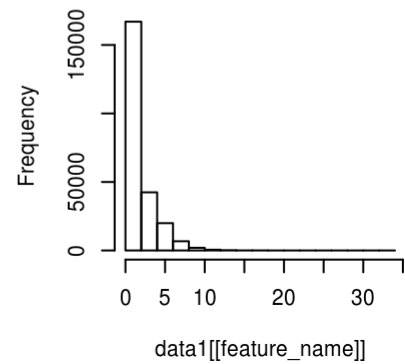
**mths\_since\_recent\_bc(missing=4%**

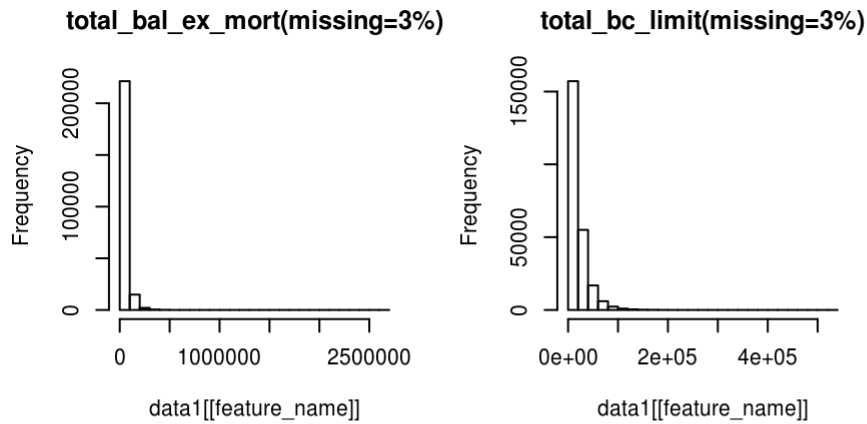


**acc\_open\_past\_24mths(missing=3%**



**mort\_acc(missing=3%)**





```
par(mfrow=c(1,1))
```

From the look of distributions, it is reasonable to impute missing values with the median value. We store `median_impute_model` for further use.

```
median_impute_model = preProcess(data1[names(tmp)],method="medianImpute")
data1 = predict(median_impute_model,data1)
```

Let's re-check the columns left, there should be no feature with NA value

```
sort(sapply(data1, function(x) sum(length(which(is.na(x)))))/nrow(data1),decreasing = TRUE)
```

##	funded_amnt	term
##	0	0
##	int_rate	sub_grade
##	0	0
##	emp_length	home_ownership
##	0	0
##	annual_inc	verification_status
##	0	0
##	issue_d	loan_status
##	0	0
##	pymnt_plan	desc
##	0	0
##	purpose	title
##	0	0
##	addr_state	dti
##	0	0
##	delinq_2yrs	earliest_cr_line
##	0	0
##	inq_last_6mths	open_acc
##	0	0
##	pub_rec	revol_bal
##	0	0
##	revol_util	total_acc
##	0	0
##	initial_list_status	collections_12_mths_ex_med
##	0	0
##	policy_code	application_type
##	0	0
##	acc_now_delinq	tot_coll_amt
##	0	0
##	tot_cur_bal	total_rev_hi_lim
##	0	0
##	acc_open_past_24mths	avg_cur_bal
##	0	0
##	bc_open_to_buy	bc_util
##	0	0
##	chargeoff_within_12_mths	delinq_amnt
##	0	0
##	mo_sin_old_il_acct	mo_sin_old_rev_tl_op
##	0	0
##	mo_sin_rcnt_rev_tl_op	mo_sin_rcnt_tl
##	0	0
##	mort_acc	mths_since_recent_bc
##	0	0
##	mths_since_recent_inq	num_accts_ever_120_pd
##	0	0
##	num_actv_bc_tl	num_actv_rev_tl
##	0	0
##	num_bc_sats	num_bc_tl
##	0	0
##	num_il_tl	num_op_rev_tl
##	0	0
##	num_rev_accts	num_rev_tl_bal_gt_0

```
##                                0                                0
##                num_sats                num_tl_90g_dpd_24m
##                                0                                0
##                num_tl_op_past_12m                pct_tl_nvr_dlq
##                                0                                0
##                percent_bc_gt_75                pub_rec_bankruptcies
##                                0                                0
##                tax_liens                tot_hi_cred_lim
##                                0                                0
##                total_bal_ex_mort                total_bc_limit
##                                0                                0
## total_il_high_credit_limit                issue_y
##                                0                                0
##                prediction
##                                0
```

Let's take a look at all features left

```
str(data1)
```



```

## 'data.frame':    246073 obs. of  67 variables:
## $ funded_amnt      : int  20800 3000 12000 12000 28000 11100 8000 24000 150
00 12000 ...
## $ term             : chr   " 36 months" " 36 months" " 36 months" " 36 month
s" ...
## $ int_rate         : num  13.53 12.85 13.53 7.62 7.62 ...
## $ sub_grade        : chr   "B5" "B4" "B5" "A3" ...
## $ emp_length       : chr   "10+ years" "10+ years" "10+ years" "3 years" ...
## $ home_ownership   : chr   "RENT" "RENT" "RENT" "MORTGAGE" ...
## $ annual_inc       : num  81500 25000 40000 96500 325000 90000 33000 100000
98000 60000 ...
## $ verification_status : chr   "Verified" "Verified" "Source Verified" "Not Veri
fied" ...
## $ issue_d          : chr   "Dec-2013" "Dec-2013" "Dec-2013" "Dec-2013" ...
## $ loan_status      : chr   "Fully Paid" "Fully Paid" "Fully Paid" "Fully Pai
d" ...
## $ pymnt_plan       : chr   "n" "n" "n" "n" ...
## $ desc             : chr   " Borrower added on 12/31/13 > My goal is to pur
chase a home. I am consolidating my debt to lower interest rate to pay off debt"| __trun
cated__ "" "" " Borrower added on 12/31/13 > Bought a new house, furniture, water softe
ner, a second car, etc. Got our lives started and now "| __truncated__ ...
## $ purpose          : chr   "debt_consolidation" "debt_consolidation" "debt_c
onsolidation" "debt_consolidation" ...
## $ title            : chr   "Reducing Debt to Purchase Home" "debt" "Debt con
solidation" "Debt Consolidation and Credit Transfer" ...
## $ addr_state       : chr   "NY" "FL" "NM" "TX" ...
## $ dti              : num  16.7 24.7 16.9 12.6 18.6 ...
## $ delinq_2yrs      : int   0 0 0 0 0 1 0 0 0 0 ...
## $ earliest_cr_line  : chr   "Jun-1998" "May-1991" "Oct-1998" "Sep-2003" ...
## $ inq_last_6mths    : int   2 0 0 0 1 0 1 0 2 1 ...
## $ open_acc         : int   29 5 7 17 15 9 9 14 16 15 ...
## $ pub_rec          : int   0 2 2 0 0 0 1 0 0 0 ...
## $ revol_bal        : int  23473 2875 5572 13248 29581 6619 7203 21617 5749
7137 ...
## $ revol_util       : chr   "54.5%" "54.2%" "68.8%" "55.7%" ...
## $ total_acc        : int   41 26 32 30 31 12 16 39 16 18 ...
## $ initial_list_status : chr   "f" "f" "w" "f" ...
## $ collections_12_mths_ex_med : int  0 0 0 0 0 0 0 0 0 0 ...
## $ policy_code      : int   1 1 1 1 1 1 1 1 1 1 ...
## $ application_type  : chr   "INDIVIDUAL" "INDIVIDUAL" "INDIVIDUAL" "INDIVIDUA
L" ...
## $ acc_now_delinq    : int   0 0 0 0 0 0 0 0 0 0 ...
## $ tot_coll_amt     : num   0 154 15386 0 0 ...
## $ tot_cur_bal      : num  23473 19530 13605 200314 799592 ...
## $ total_rev_hi_lim  : num  43100 5300 8100 23800 54200 10000 20800 28200 258
00 29700 ...
## $ acc_open_past_24mths : num  9 3 4 4 6 2 2 7 6 8 ...
## $ avg_cur_bal      : num  869 3906 2268 11783 53306 ...
## $ bc_open_to_buy    : num  6811 2050 1428 2441 13901 ...
## $ bc_util          : num  54.6 52.3 79.6 83.5 67.1 74.6 72.5 77.6 27.6 15.9
...
## $ chargeoff_within_12_mths : int  0 0 0 0 0 0 0 0 0 0 ...
## $ delinq_amnt      : int  0 0 0 0 0 0 0 0 0 0 ...

```

```
## $ mo_sin_old_il_acct      : num  115 164 124 123 125 128 129 179 2 128 ...
## $ mo_sin_old_rev_tl_op   : num  186 271 182 118 229 150 269 299 257 48 ...
## $ mo_sin_rcnt_rev_tl_op  : num   0  7  1 10  5 11 14 18  7  1 ...
## $ mo_sin_rcnt_tl         : num   0  7  1  9  2 11 14  7  2  1 ...
## $ mort_acc               : num   0  6  0  1  5  1  0  3  0  0 ...
## $ mths_since_recent_bc   : num   0 14 11 10  5 11 18 18  7  1 ...
## $ mths_since_recent_inq  : num   0  8 17 10  3 11  6  7  2  3 ...
## $ num_accts_ever_120_pd  : num   1  1  6  0  0  1  0  0  0  0 ...
## $ num_actv_bc_tl         : num   8  2  2  4  4  4  3  3  8  4 ...
## $ num_actv_rev_tl        : num  24  3  2  5  5  8  5  5  8  7 ...
## $ num_bc_sats            : num  11  3  3  4  6  4  4  5 13  8 ...
## $ num_bc_tl              : num  17  6 14 10  8  4  7 10 13 10 ...
## $ num_il_tl              : num   1 11  8 15 11  0  2 17  1  0 ...
## $ num_op_rev_tl          : num  29  4  6  8  9  8  8  8 15 15 ...
## $ num_rev_accts          : num  40  9 24 14 15 11 14 19 15 18 ...
## $ num_rev_tl_bal_gt_0    : num  24  3  2  5  5  8  5  5  8  7 ...
## $ num_sats               : num  29  5  7 17 15  9  9 14 16 15 ...
## $ num_tl_90g_dpd_24m    : num   0  0  0  0  0  1  0  0  0  0 ...
## $ num_tl_op_past_12m    : num   3  1  2  3  5  1  0  2  2  4 ...
## $ pct_tl_nvr_dlq         : num  90.2 91.3 81.2 100 100 75 100 100 100 100 ...
## $ percent_bc_gt_75       : num  50 66.7 33.3 100 16.7 50 75 75 7.7 0 ...
## $ pub_rec_bankruptcies   : int   0  2  0  0  0  0  1  0  0  0 ...
## $ tax_liens              : int   0  0  0  0  0  0  0  0  0  0 ...
## $ tot_hi_cred_lim         : num 43100 32082 18130 233004 850886 ...
## $ total_bal_ex_mort       : num 23473 19530 13605 46738 199739 ...
## $ total_bc_limit         : num 15000 4300 7000 14800 42200 4000 8200 21500 20800
18100 ...
## $ total_il_high_credit_limit: num   0 26782 10030 53404 196686 ...
## $ issue_y                : num  2013 2013 2013 2013 2013 ...
## $ prediction              : chr  "Fully Paid" "Fully Paid" "Fully Paid" "Fully Paid" ...
```

We parse `revol_util` to numeric

```
data1$revol_util = (as.numeric(gsub(pattern = "%",replacement = "",x = data1$int_rate)))
```

`earliest_cr_line` is transformed to the number of days before the loan is issued

```
data1$earliest_cr_line = parse_date_time(str_c("01",data1$issue_d),"dmy" ) - parse_date_time(str_c("01",data1$earliest_cr_line),"dmy" )
data1$earliest_cr_line = as.numeric(data1$earliest_cr_line,units = "days")
```

We can see that the default rate doesn't vary much by the month it issued. We will drop `issue_d`

```
#extract issued month
data1$issue_m = sapply( data1$issue_d ,function(x){str_split(x,"-")[[1]][1]})

tmp = data1 %>% filter(loan_status=="Default") %>% group_by(issue_m) %>% summarise(default_count = n())
tmp2 = data1 %>% group_by(issue_m) %>% summarise(count = n())
tmp2 %>% left_join(tmp) %>% mutate(default_rate = default_count/count)
```

```
## Joining, by = "issue_m"
```

```
## Source: local data frame [12 x 4]
##
##   issue_m count default_count default_rate
##   <chr> <int>      <int>          <dbl>
## 1     Apr  20828         4134      0.1984828
## 2     Aug  20604         4592      0.2228693
## 3     Dec  17151         3452      0.2012711
## 4     Feb  17129         3391      0.1979684
## 5     Jan  17315         3356      0.1938204
## 6     Jul  24245         5663      0.2335739
## 7     Jun  20575         4230      0.2055893
## 8     Mar  18570         3599      0.1938072
## 9     May  21288         4357      0.2046693
## 10    Nov  22413         5007      0.2233971
## 11    Oct  28244         6553      0.2320139
## 12    Sep  17711         3714      0.2097002
```

```
data1$issue_m = NULL
data1$issue_d = NULL

rm(tmp, tmp2)
```

Let's see if there are any features that have zero variance:

```
#define some functions to be used later on
getNumericColumns<-function(t){
  tn = sapply(t,function(x){is.numeric(x)})
  return(names(tn)[which(tn)])
}

getCharColumns<-function(t){
  tn = sapply(t,function(x){is.character(x)})
  return(names(tn)[which(tn)])
}

getFactorColumns<-function(t){
  tn = sapply(t,function(x){is.factor(x)})
  return(names(tn)[which(tn)])
}

getIndexsOfColumns <- function(t,column_names){
  return(match(column_names,colnames(t)))
}
```

```
tmp = apply(data1[getCharColumns(data1)], 2, function(x){length(unique(x))})
tmp = tmp[tmp==1]

tmp2 = apply(data1[getNumericColumns(data1)], 2, function(x){(sd(x))})
tmp2 = tmp2[tmp2==0]

discard_column = c(names(tmp), names(tmp2))
discard_column
```

```
## [1] "application_type" "prediction"          "policy_code"
```

We then proceed to drop the zero variance features

```
data1 = (data1[,!(names(data1) %in% discard_column)])
```

Next, we investigate whether `desc` (description) can be useful in determining Default and Fully Paid loan. However, noted that our test dataset(2015 loans) has only **9** loans with non-empty description's length so `desc` is quite useless for current 2015 data. Also noted that information in `title` is replicated in `purpose` so we will drop them.

```
data1$desc = NULL
data1$title = NULL
```

We take a look at default rate for each state from year 2012-2014. We filter out states that have too small number of loans(less than 1000):

```
tmp = data1 %>% filter(loan_status=="Default") %>% group_by(addr_state, issue_y) %>% summarise(default_count = n())
tmp2 = data1 %>% group_by(addr_state, issue_y) %>% summarise(count = n())
tmp3 = tmp2 %>% left_join(tmp) %>% mutate(default_rate = default_count/count)
```

```
## Joining, by = c("addr_state", "issue_y")
```

```

#order by highest default rate
a0 = (tmp3 %>% filter(issue_y == 2012 & count > 1000) %>% arrange(desc(default_rate)))
[1:10,"addr_state"]$addr_state
a1 = (tmp3 %>% filter(issue_y == 2013 & count > 1000) %>% arrange(desc(default_rate)))
[1:10,"addr_state"]$addr_state
a2 = (tmp3 %>% filter(issue_y == 2014 & count > 1000) %>% arrange(desc(default_rate)))
[1:10,"addr_state"]$addr_state
high_default = intersect(intersect(a0,a1),a2)

#order by lowest default rate
a0 = (tmp3 %>% filter(issue_y == 2012 & count > 1000) %>% arrange((default_rate)))
[1:10,"addr_state"]$addr_state
a1 = (tmp3 %>% filter(issue_y == 2013 & count > 1000) %>% arrange((default_rate)))
[1:10,"addr_state"]$addr_state
a2 = (tmp3 %>% filter(issue_y == 2014 & count > 1000) %>% arrange((default_rate)))
[1:10,"addr_state"]$addr_state
low_default = intersect(intersect(a0,a1),a2)

```

We noticed Florida and New York consistently constitute in top 10 - highest default rate from year 2012 - 2014

```
high_default
```

```
## [1] "FL" "NY"
```

While Illinois, Texas, California, Georgia have lowest default rate

```
low_default
```

```
## [1] "IL" "TX" "CA" "GA"
```

We then create binary variable for 6 states and discard the rest

```

data1$sis_fl = ifelse(data1$addr_state=="FL",1,0)
data1$sis_ny = ifelse(data1$addr_state=="NY",1,0)

data1$sis_il = ifelse(data1$addr_state=="IL",1,0)
data1$sis_tx = ifelse(data1$addr_state=="TX",1,0)
data1$sis_ca = ifelse(data1$addr_state=="CA",1,0)
data1$sis_ga = ifelse(data1$addr_state=="GA",1,0)

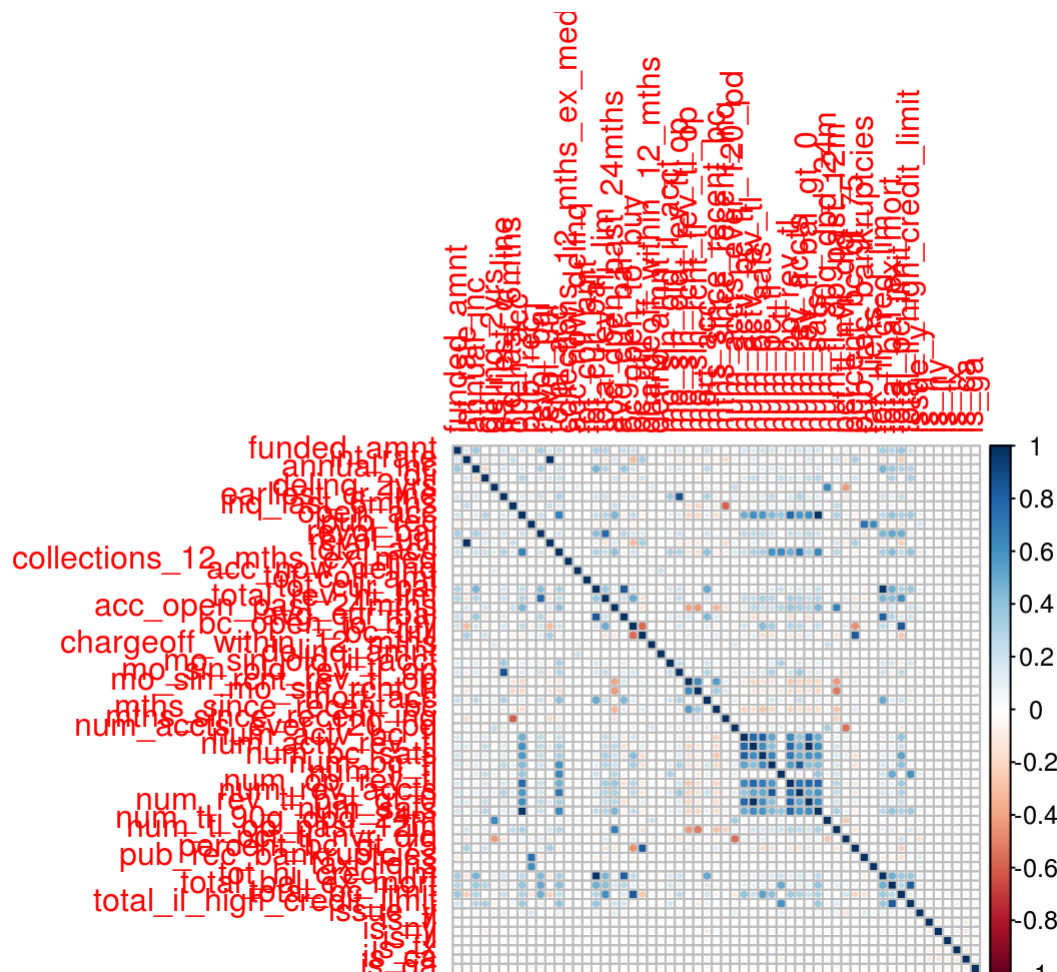
data1$addr_state = NULL

rm(tmp,tmp2,tmp3,a0,a1,a2,high_default,low_default)

```

We will investigate if there are any correlation among features

```
corrplot(cor(data1[getNumericColumns(data1)],use="na.or.complete"))
```



We found some features are quite correlated, we can remove correlated features with `findCorrelation` function. The function will find all correlated pairs that have correlation exceed a specified threshold and try to remove one of them in such a way that overall correlation is reduced.

```
high_corr <- findCorrelation(cor(data1[getNumericColumns(data1)]), cutoff = .75)
high_corr = getNumericColumns(data1)[high_corr]
high_corr
```

```
## [1] "num_sats"          "open_acc"          "num_op_rev_tl"
## [4] "num_rev_accts"     "num_bc_sats"       "total_bc_limit"
## [7] "num_rev_tl_bal_gt_0" "num_actv_rev_tl"   "tot_hi_cred_lim"
## [10] "total_rev_hi_lim"  "tot_cur_bal"       "total_bal_ex_mort"
## [13] "earliest_cr_line" "int_rate"          "bc_util"
```

```
data1 = (data1[,!(names(data1) %in% high_corr)])
```

Let's look at all numeric features we have left.

```
str(data1[getNumericColumns(data1)])
```

```
## 'data.frame':    246073 obs. of  42 variables:
## $ funded_amnt      : int  20800 3000 12000 12000 28000 11100 8000 24000 150
00 12000 ...
## $ annual_inc       : num  81500 25000 40000 96500 325000 90000 33000 100000
98000 60000 ...
## $ dti              : num  16.7 24.7 16.9 12.6 18.6 ...
## $ delinq_2yrs      : int  0 0 0 0 0 1 0 0 0 0 ...
## $ inq_last_6mths   : int  2 0 0 0 1 0 1 0 2 1 ...
## $ pub_rec          : int  0 2 2 0 0 0 1 0 0 0 ...
## $ revol_bal        : int  23473 2875 5572 13248 29581 6619 7203 21617 5749
7137 ...
## $ revol_util       : num  13.53 12.85 13.53 7.62 7.62 ...
## $ total_acc        : int  41 26 32 30 31 12 16 39 16 18 ...
## $ collections_12_mths_ex_med: int  0 0 0 0 0 0 0 0 0 0 ...
## $ acc_now_delinq   : int  0 0 0 0 0 0 0 0 0 0 ...
## $ tot_coll_amt     : num  0 154 15386 0 0 ...
## $ acc_open_past_24mths : num  9 3 4 4 6 2 2 7 6 8 ...
## $ avg_cur_bal      : num  869 3906 2268 11783 53306 ...
## $ bc_open_to_buy   : num  6811 2050 1428 2441 13901 ...
## $ chargeoff_within_12_mths : int  0 0 0 0 0 0 0 0 0 0 ...
## $ delinq_amnt      : int  0 0 0 0 0 0 0 0 0 0 ...
## $ mo_sin_old_il_acct : num  115 164 124 123 125 128 129 179 2 128 ...
## $ mo_sin_old_rev_tl_op : num  186 271 182 118 229 150 269 299 257 48 ...
## $ mo_sin_rcnt_rev_tl_op : num  0 7 1 10 5 11 14 18 7 1 ...
## $ mo_sin_rcnt_tl    : num  0 7 1 9 2 11 14 7 2 1 ...
## $ mort_acc         : num  0 6 0 1 5 1 0 3 0 0 ...
## $ mths_since_recent_bc : num  0 14 11 10 5 11 18 18 7 1 ...
## $ mths_since_recent_inq : num  0 8 17 10 3 11 6 7 2 3 ...
## $ num_accts_ever_120_pd : num  1 1 6 0 0 1 0 0 0 0 ...
## $ num_actv_bc_tl    : num  8 2 2 4 4 4 3 3 8 4 ...
## $ num_bc_tl         : num  17 6 14 10 8 4 7 10 13 10 ...
## $ num_il_tl         : num  1 11 8 15 11 0 2 17 1 0 ...
## $ num_tl_90g_dpd_24m : num  0 0 0 0 0 1 0 0 0 0 ...
## $ num_tl_op_past_12m : num  3 1 2 3 5 1 0 2 2 4 ...
## $ pct_tl_nvr_dlq    : num  90.2 91.3 81.2 100 100 75 100 100 100 100 ...
## $ percent_bc_gt_75  : num  50 66.7 33.3 100 16.7 50 75 75 7.7 0 ...
## $ pub_rec_bankruptcies : int  0 2 0 0 0 0 1 0 0 0 ...
## $ tax_liens        : int  0 0 0 0 0 0 0 0 0 0 ...
## $ total_il_high_credit_limit: num  0 26782 10030 53404 196686 ...
## $ issue_y          : num  2013 2013 2013 2013 2013 ...
## $ is_fl            : num  0 1 0 0 0 0 0 0 0 0 ...
## $ is_ny            : num  1 0 0 0 0 1 0 0 1 0 ...
## $ is_il            : num  0 0 0 0 0 0 0 0 0 0 ...
## $ is_tx            : num  0 0 0 1 0 0 0 0 0 0 ...
## $ is_ca            : num  0 0 0 0 1 0 0 0 0 0 ...
## $ is_ga            : num  0 0 0 0 0 0 0 0 0 0 ...
```

We transform `annual_inc`, `revol_bal`, `avg_cur_bal`, `bc_open_to_buy`, `total_il_high_credit_limit` by deviding them by `funded_amnt` (amount of loan)

```

data1$annual_inc = data1$annual_inc/data1$funded_amnt
data1$revol_bal = data1$revol_bal/data1$funded_amnt
data1$avg_cur_bal = data1$avg_cur_bal/data1$funded_amnt
data1$bc_open_to_buy = data1$bc_open_to_buy/data1$funded_amnt
data1$total_il_high_credit_limit = data1$total_il_high_credit_limit/data1$funded_amnt

data1$funded_amnt = NULL

```

Let's look at all character features we have left.

```
str(data1[getCharColumns(data1)])
```

```

## 'data.frame':    246073 obs. of  9 variables:
## $ term           : chr  " 36 months" " 36 months" " 36 months" " 36 months" ...
## $ sub_grade      : chr  "B5" "B4" "B5" "A3" ...
## $ emp_length     : chr  "10+ years" "10+ years" "10+ years" "3 years" ...
## $ home_ownership : chr  "RENT" "RENT" "RENT" "MORTGAGE" ...
## $ verification_status: chr  "Verified" "Verified" "Source Verified" "Not Verified"
## ...
## $ loan_status     : chr  "Fully Paid" "Fully Paid" "Fully Paid" "Fully Paid" ...
## $ pymnt_plan      : chr  "n" "n" "n" "n" ...
## $ purpose         : chr  "debt_consolidation" "debt_consolidation" "debt_consolidation" "debt_consolidation" ...
## $ initial_list_status: chr  "f" "f" "w" "f" ...

```

We look at `home_ownership` and filter only observations that have value “MORTGAGE”, “OWN”, or “RENT” as these are only values that appear in 2015 data

```
table(data1$home_ownership)
```

```

##
##      ANY MORTGAGE      NONE      OTHER      OWN      RENT
##      1    123973      40        43    21228    100788

```

```
data1 = data1 %>% filter(home_ownership == "MORTGAGE" | home_ownership == "OWN" | home_ownership == "RENT")
```

We see there are 3 loans with `pymnt_plan="y"`, all ended in being default

```
data1 %>% filter(pymnt_plan=="y") %>% select(loan_status)
```

```

##   loan_status
## 1    Default
## 2    Default
## 3    Default

```

But 3 loans is too small to make the conclusive finding. We will just remove `pymnt_plan` feature.



```
data1$pymnt_plan = NULL
```

remove issue\_y as it has no further use

```
## Warning in rm(all_roi, c, discard_column, feature_name, high_corr,  
## numeric_column): object 'c' not found
```

```
## Warning in rm(all_roi, c, discard_column, feature_name, high_corr,  
## numeric_column): object 'numeric_column' not found
```

We will transform all character predictor features to binary dummy variables

```
loan_status = data1$loan_status  
dummy_model = dummyVars(loan_status ~ ., data1, fullRank = TRUE)  
data1 = as.data.frame(predict(dummy_model, data1))  
data1$loan_status = loan_status  
rm(loan_status)  
  
#set loan with status 'Fully Paid' as a positive sample  
data1$loan_status = ifelse(data1$loan_status == "Fully Paid", "Fully.Paid", data1$loan_status)  
data1$loan_status = factor(data1$loan_status, levels = c("Default", "Fully.Paid"))
```

The data is centered and scaled. We can try to remove the number of dimension further by fitting Logistic regression and investigate p-value of the coefficients. The null hypothesis is that each feature makes no contribution to the predictive model(its coefficient is zero). We then discard each feature that fails to reject the hypothesis.

```
trans_model = preProcess(data1, method=c("center", "scale"))  
data1 = predict(trans_model, data1)  
  
model = lrm(loan_status ~ ., data1)  
model
```

```
##
## Logistic Regression Model
##
## lrm(formula = loan_status ~ ., data = data1)
##
##           Model Likelihood      Discrimination      Rank Discrim.
##           Ratio Test           Indexes           Indexes
## Obs           245989      LR chi2    24375.03      R2           0.147      C           0.714
## Default        52032      d.f.           103      g           0.968      Dxy          0.428
## Fully.Paid 193957      Pr(> chi2) <0.0001      gr          2.632      gamma        0.430
## max |deriv|  7e-11                                     gp          0.142      tau-a        0.143
##
##                                     Brier          0.150
##
##                                     Coef      S.E.    Wald Z Pr(>|Z|)
## Intercept                                     1.5087 0.0059 255.30 <0.0001
## term 60 months                                -0.2062 0.0061 -34.04 <0.0001
## sub_gradeA2                                -0.0644 0.0142  -4.53 <0.0001
## sub_gradeA3                                -0.1097 0.0145  -7.57 <0.0001
## sub_gradeA4                                -0.1864 0.0163 -11.45 <0.0001
## sub_gradeA5                                -0.2593 0.0173 -15.00 <0.0001
## sub_gradeB1                                -0.3310 0.0185 -17.89 <0.0001
## sub_gradeB2                                -0.4134 0.0210 -19.70 <0.0001
## sub_gradeB3                                -0.5064 0.0235 -21.55 <0.0001
## sub_gradeB4                                -0.5522 0.0237 -23.32 <0.0001
## sub_gradeB5                                -0.5411 0.0218 -24.83 <0.0001
## sub_gradeC1                                -0.5974 0.0233 -25.67 <0.0001
## sub_gradeC2                                -0.6219 0.0235 -26.51 <0.0001
## sub_gradeC3                                -0.6453 0.0233 -27.69 <0.0001
## sub_gradeC4                                -0.6569 0.0234 -28.06 <0.0001
## sub_gradeC5                                -0.6666 0.0233 -28.59 <0.0001
## sub_gradeD1                                -0.6597 0.0230 -28.67 <0.0001
## sub_gradeD2                                -0.6445 0.0223 -28.91 <0.0001
## sub_gradeD3                                -0.6108 0.0214 -28.58 <0.0001
## sub_gradeD4                                -0.6266 0.0213 -29.47 <0.0001
## sub_gradeD5                                -0.5803 0.0200 -29.07 <0.0001
## sub_gradeE1                                -0.5178 0.0177 -29.29 <0.0001
## sub_gradeE2                                -0.5465 0.0183 -29.83 <0.0001
## sub_gradeE3                                -0.4929 0.0170 -29.07 <0.0001
## sub_gradeE4                                -0.4856 0.0166 -29.30 <0.0001
## sub_gradeE5                                -0.4608 0.0157 -29.36 <0.0001
## sub_gradeF1                                -0.4137 0.0146 -28.42 <0.0001
## sub_gradeF2                                -0.3745 0.0133 -28.10 <0.0001
## sub_gradeF3                                -0.3769 0.0132 -28.64 <0.0001
## sub_gradeF4                                -0.3256 0.0116 -27.99 <0.0001
## sub_gradeF5                                -0.2906 0.0106 -27.35 <0.0001
## sub_gradeG1                                -0.2538 0.0094 -27.15 <0.0001
## sub_gradeG2                                -0.2050 0.0082 -25.04 <0.0001
## sub_gradeG3                                -0.1826 0.0073 -24.98 <0.0001
## sub_gradeG4                                -0.1331 0.0063 -21.06 <0.0001
## sub_gradeG5                                -0.1389 0.0062 -22.29 <0.0001
## emp_length< 1 year                        -0.0273 0.0057  -4.83 <0.0001
## emp_length1 year                           0.0015 0.0057   0.26 0.7911
## emp_length2 years                          0.0002 0.0058   0.03 0.9749
## emp_length3 years                        -0.0016 0.0057  -0.28 0.7826
```

## emp_length4 years	0.0013	0.0056	0.22	0.8229
## emp_length5 years	0.0023	0.0057	0.40	0.6917
## emp_length6 years	-0.0118	0.0056	-2.11	0.0345
## emp_length7 years	-0.0004	0.0056	-0.07	0.9425
## emp_length8 years	-0.0121	0.0055	-2.22	0.0265
## emp_length9 years	-0.0187	0.0053	-3.51	0.0004
## emp_lengthn/a	-0.1086	0.0051	-21.22	<0.0001
## home_ownershipOWN	-0.0256	0.0056	-4.61	<0.0001
## home_ownershipRENT	-0.0949	0.0067	-14.11	<0.0001
## annual_inc	0.1356	0.0118	11.54	<0.0001
## verification_statusSource Verified	-0.0386	0.0065	-5.91	<0.0001
## verification_statusVerified	0.0074	0.0070	1.07	0.2850
## purposecredit_card	0.0161	0.0253	0.64	0.5244
## purposedebt_consolidation	-0.0110	0.0295	-0.37	0.7090
## purposehome_improvement	-0.0316	0.0147	-2.15	0.0316
## purposehouse	-0.0001	0.0068	-0.01	0.9889
## purposemajor_purchase	-0.0178	0.0098	-1.82	0.0695
## purposemedical	-0.0246	0.0077	-3.20	0.0014
## purposemoving	-0.0198	0.0067	-2.93	0.0033
## purposeother	-0.0365	0.0135	-2.69	0.0071
## purposerenewable_energy	-0.0006	0.0055	-0.11	0.9111
## purposesmall_business	-0.0679	0.0084	-8.12	<0.0001
## purposevacation	-0.0167	0.0067	-2.49	0.0129
## purposewedding	0.0199	0.0072	2.77	0.0055
## dti	-0.2184	0.0064	-34.26	<0.0001
## delinq_2yrs	-0.0920	0.0071	-12.93	<0.0001
## inq_last_6mths	-0.0438	0.0066	-6.68	<0.0001
## pub_rec	-0.0337	0.0148	-2.28	0.0226
## revol_bal	-0.0166	0.0081	-2.06	0.0399
## revol_util	0.6408	0.0336	19.06	<0.0001
## total_acc	0.1322	0.0131	10.11	<0.0001
## initial_list_statusw	-0.0022	0.0054	-0.41	0.6791
## collections_12_mths_ex_med	-0.0110	0.0048	-2.29	0.0217
## acc_now_delinq	-0.0035	0.0049	-0.72	0.4739
## tot_coll_amt	0.0030	0.0109	0.27	0.7854
## acc_open_past_24mths	-0.1285	0.0077	-16.72	<0.0001
## avg_cur_bal	0.0353	0.0100	3.52	0.0004
## bc_open_to_buy	0.0351	0.0096	3.64	0.0003
## chargeoff_within_12_mths	0.0121	0.0056	2.19	0.0288
## delinq_amnt	0.0022	0.0058	0.37	0.7084
## mo_sin_old_il_acct	-0.0186	0.0057	-3.25	0.0011
## mo_sin_old_rev_tl_op	0.0175	0.0061	2.84	0.0044
## mo_sin_rcnt_rev_tl_op	-0.0161	0.0081	-1.97	0.0483
## mo_sin_rcnt_tl	0.0221	0.0079	2.81	0.0050
## mort_acc	0.0618	0.0075	8.26	<0.0001
## mths_since_recent_bc	0.0555	0.0070	7.89	<0.0001
## mths_since_recent_inq	-0.0064	0.0067	-0.96	0.3394
## num_accts_ever_120_pd	-0.0285	0.0067	-4.26	<0.0001
## num_actv_bc_tl	-0.0526	0.0066	-8.03	<0.0001
## num_bc_tl	0.0279	0.0097	2.88	0.0040
## num_il_tl	0.0168	0.0099	1.70	0.0894
## num_tl_90g_dpd_24m	0.0175	0.0067	2.62	0.0088
## num_tl_op_past_12m	-0.0088	0.0076	-1.16	0.2462
## pct_tl_nvr_dlq	-0.0228	0.0070	-3.24	0.0012

```
## percent_bc_gt_75          -0.0766 0.0065 -11.73 <0.0001
## pub_rec_bankruptcies      0.0391 0.0118  3.32 0.0009
## tax_liens                 0.0090 0.0097  0.92 0.3550
## total_il_high_credit_limit 0.0721 0.0098  7.36 <0.0001
## is_fl                    -0.0178 0.0052 -3.42 0.0006
## is_ny                    -0.0284 0.0053 -5.34 <0.0001
## is_il                    0.0130 0.0054  2.39 0.0166
## is_tx                    0.0375 0.0055  6.77 <0.0001
## is_ca                    0.0402 0.0057  7.09 <0.0001
## is_ga                    0.0181 0.0054  3.34 0.0008
```

We set our two-tailed p-value cutoff at 0.01, we discard features with p-value exceed this threshold.

```
tmp = as.data.frame(anova(model))
tmp$feature = rownames(tmp)
tmp = tmp %>% filter(P > 0.01) %>% select(feature,P)
tmp
```

```
##           feature          P
## 1      emp_length1 year 0.79109580
## 2      emp_length2 years 0.97486100
## 3      emp_length3 years 0.78255500
## 4      emp_length4 years 0.82290650
## 5      emp_length5 years 0.69167559
## 6      emp_length6 years 0.03454150
## 7      emp_length7 years 0.94254322
## 8      emp_length8 years 0.02648818
## 9 verification_statusVerified 0.28497122
## 10      purposecredit_card 0.52436187
## 11      purposedebt_consolidation 0.70902630
## 12      purposehome_improvement 0.03161482
## 13      purposehouse 0.98894832
## 14      purposemajor_purchase 0.06951881
## 15      purposerenewable_energy 0.91112091
## 16      purposevacation 0.01287300
## 17      pub_rec 0.02260737
## 18      revol_bal 0.03987652
## 19      initial_list_statusw 0.67910967
## 20      collections_12_mths_ex_med 0.02174068
## 21      acc_now_delinq 0.47394870
## 22      tot_coll_amt 0.78543120
## 23      chargeoff_within_12_mths 0.02876742
## 24      delinq_amnt 0.70839057
## 25      mo_sin_rcnt_rev_tl_op 0.04832643
## 26      mths_since_recent_inq 0.33939555
## 27      num_il_tl 0.08944578
## 28      num_tl_op_past_12m 0.24616264
## 29      tax_liens 0.35499506
## 30      is_il 0.01664836
```

```
data1 = (data1[,!(names(data1) %in% tmp$feature)])
```

```
rm(model,tmp)
```

Some feature names are invalid, replace invalid characters with “\_”

```
colnames(data1) = str_replace_all(colnames(data1)," ","_")
```

```
colnames(data1) = str_replace_all(colnames(data1),"<","_")
```

```
colnames(data1) = str_replace_all(colnames(data1),"/","_")
```

We then create function to apply all of the above feature engineering:

```

#derived from str_c("",paste(colnames(data1),collapse="",""),"")
#kept_column = colnames(data1)
kept_column =
c('term_60_months','sub_gradeA2','sub_gradeA3','sub_gradeA4','sub_gradeA5','sub_gradeB1','sub_gradeB2','sub_gradeB3','sub_gradeB4','sub_gradeB5','sub_gradeC1','sub_gradeC2','sub_gradeC3','sub_gradeC4','sub_gradeC5','sub_gradeD1','sub_gradeD2','sub_gradeD3','sub_gradeD4','sub_gradeD5','sub_gradeE1','sub_gradeE2','sub_gradeE3','sub_gradeE4','sub_gradeE5','sub_gradeF1','sub_gradeF2','sub_gradeF3','sub_gradeF4','sub_gradeF5','sub_gradeG1','sub_gradeG2','sub_gradeG3','sub_gradeG4','sub_gradeG5','emp_length_1_year','emp_length9_year','emp_lengthn_a','home_ownershipOWN','home_ownershipRENT','annual_inc','verification_statusSource_Verified','purposemedical','purposemoving','purposeother','purposesmall_business','purposewedding','dti','delinq_2yrs','inq_last_6mths','revol_util','total_acc','acc_open_past_24mths','avg_cur_bal','bc_open_to_buy','mo_sin_old_il_acct','mo_sin_old_rev_tl_op','mo_sin_rcnt_tl','mort_acc','mths_since_recent_bc','num_accts_ever_120_pd','num_actv_bc_tl','num_bc_tl','num_tl_90g_dpd_24m','pct_tl_nvr_dlq','percent_bc_gt_75','pub_rec_bankruptcies','total_il_high_credit_limit','is_fl','is_ny','is_tx','is_ca','is_ga','loan_status')

```

```

applyFeatureEngineering <- function(dt,use_kept_column = kept_column,use_median_impute_model=median_impute_model, use_dummy_model=dummy_model,use_trans_model=trans_model){

```

```

  #consolidate loan status
  dt$loan_status = ifelse(str_detect(dt$loan_status,"Paid"),dt$loan_status,"Default")
  #parse int_rate
  dt$int_rate = (as.numeric(gsub(pattern = "%",replacement = "",x = dt$int_rate)))
  #impute median
  dt = predict(median_impute_model,dt)
  #parse revol_util
  dt$revol_util = (as.numeric(gsub(pattern = "%",replacement = "",x = dt$int_rate)))
  #binary variables for addr_state
  dt$is_fl = ifelse(dt$addr_state=="FL",1,0)
  dt$is_ny = ifelse(dt$addr_state=="NY",1,0)

  dt$is_il = ifelse(dt$addr_state=="IL",1,0)
  dt$is_tx = ifelse(dt$addr_state=="TX",1,0)
  dt$is_ca = ifelse(dt$addr_state=="CA",1,0)
  dt$is_ga = ifelse(dt$addr_state=="GA",1,0)
  #transform transactions
  dt$annual_inc = dt$annual_inc/dt$funded_amnt
  dt$revol_bal = dt$revol_bal/dt$funded_amnt
  dt$avg_cur_bal = dt$avg_cur_bal/dt$funded_amnt
  dt$bc_open_to_buy = dt$bc_open_to_buy/dt$funded_amnt
  dt$total_il_high_credit_limit = dt$total_il_high_credit_limit/dt$funded_amnt
  #if purpose falling outside of recognized values
  all_purpose = c('debt_consolidation','small_business','other','credit_card','major_purchase','moving','home_improvement','house','car','medical','renewable_energy','vacation','wedding')
  dt$purpose = ifelse(dt$purpose %in% all_purpose,dt$purpose,"other")
  #create dummy variables
  loan_status = dt$loan_status
  dt = as.data.frame(predict(use_dummy_model,dt))
  dt$loan_status = loan_status
  #center, scale data

```

```

dt = predict(use_trans_model, dt)
#remove all unused features
colnames(dt) = str_replace_all(colnames(dt), " ", "_")
colnames(dt) = str_replace_all(colnames(dt), "<", "_")
colnames(dt) = str_replace_all(colnames(dt), "/", "_")
dt = dt[use_kept_column]

#set loan with status 'Fully Paid' as a positive sample
dt$loan_status = ifelse(dt$loan_status == "Fully Paid", "Fully.Paid", dt$loan_status)
dt$loan_status = factor(dt$loan_status, levels = c("Default", "Fully.Paid"))

return(dt)
}

```

## Model Training

We need to determine which performance metric we want to focus on. We load the 2015 test dataset and apply feature engineering procedure.

```

test_data = read.csv("loan_2015.extract.csv", stringsAsFactors=FALSE)
#later used for evaluate investment
test_data_grade = test_data$grade
test_data_funded_amnt = test_data$funded_amnt
test_data_total_pymnt = test_data$total_pymnt

test_data = applyFeatureEngineering(test_data)

```

The loan data typically have higher proportion of good loans. We can achieve high accuracy just by labelling all loans as Fully Paid

```
100*nrow(test_data %>% filter(loan_status=="Fully.Paid"))/nrow(test_data)
```

```
## [1] 72.29722
```

For our test data, we gain 72.3% accuracy by just following the above strategy. Recall that we yet to include the outcome of current loans. In a real situation, the ratio of Fully Paid loans is usually much higher so accuracy metric is not our main concern here. We instead focus on a trade-off in identifying a default loan as an expense of mislabelling some good loans. We will look at ROC curve and pay particular focus on AUC when we train our models.

Because there is a disproportion of target variable, we downsample the Fully Paid loans to be equal to Default loans. This method tends to work well and run faster than upsampling or cost-sensitive training.

Noted that at the end, we aim to stack the results of various learning models(Logistic Regression, SVM, Random Forest, and XGB). Since the downside of downsampling is that information of majority class is discarded, we will continue to make a new downsampling data when we feed it to each model along the way. We anticipate that better result can be obtained by stacking all 4 models since it gets more information from the majority class.

# Logistic Regression

We tune Logistic Regression to our training dataset. We use Elastic Net regularization, which comprised of Ridge and Lasso regularization, with cross validation to prevent overfitting. Our goal is maximizing AUC.

Due to limited computation resource, we run model tuning on small data and fixed lambda parameter. We use small fold: 3-fold cross validation. We then refit the best model with the whole data.

(Noted:we put the final tuning result here instead of running through the whole process. We disable the execution of tuning code although readers can enable it back by setting `eval = TRUE` )

```
set.seed(100)
samp = downSample(data1[-getIndxsOfColumns(data1, c( "loan_status" )],data1$loan_status,
yname="loan_status")
#choose small data for tuning
train_index = createDataPartition(samp$loan_status,p = 0.05,list=FALSE,times=1)
```

```
#run tuning in parallel using available computing cores(you may need to change this)
registerDoMC(cores = 4)

ctrl <- trainControl(method = "cv",
  summaryFunction = twoClassSummary,
  classProbs = TRUE,
  number = 3
)

glmnetGrid = expand.grid(.alpha = seq(0, 1, length = 10), .lambda = 0.01)

glmnetTuned = train(samp[train_index,-getIndxsOfColumns(samp,"loan_status")],y = samp[train_index,"loan_status"],method = "glmnet",tuneGrid = glmnetGrid,metric = "ROC",trControl = ctrl)
```

```
plot(glmnetTuned)
glmnetTuned
```

The best penalty parameter is  $\alpha = 0.7777778$ (more weight on Ridge) with fixed shrinking  $\lambda = 0.01$ . We use this parameter to retrain the whole sample.

```
model = glmnet(
  x = as.matrix(samp[-getIndxsOfColumns(samp,"loan_status")]),
  y=samp$loan_status,
  alpha = 0.7777778,
  lambda = 0.01,
  family = "binomial",
  standardize = FALSE)
```

The finalized Logistic Regression model is applied to the 2015 loan data. We look at ROC graph and AUC. We also set probability prediction cutoff at 50%(noted that the higher this value is, the more likely the loan is Fully Paid ) and collect some performance metrics for a later comparison.



```

#data frame for collect model's performance
table_perf = data.frame(model=character(0),
                        auc=numeric(0),
                        accuracy=numeric(0),
                        sensitivity=numeric(0),
                        specificity=numeric(0),
                        kappa=numeric(0),
                        stringsAsFactors = FALSE
                        )

predict_loan_status_logit = predict(model,newx = as.matrix(test_data[-
getIndexesOfColumns(test_data,"loan_status")] ),type="response")

rocCurve_logit = roc(response = test_data$loan_status,
                    predictor = predict_loan_status_logit)

```

```

## Warning in roc.default(response = test_data$loan_status, predictor
## = predict_loan_status_logit): Deprecated use a matrix as predictor.
## Unexpected results may be produced, please pass a numeric vector.

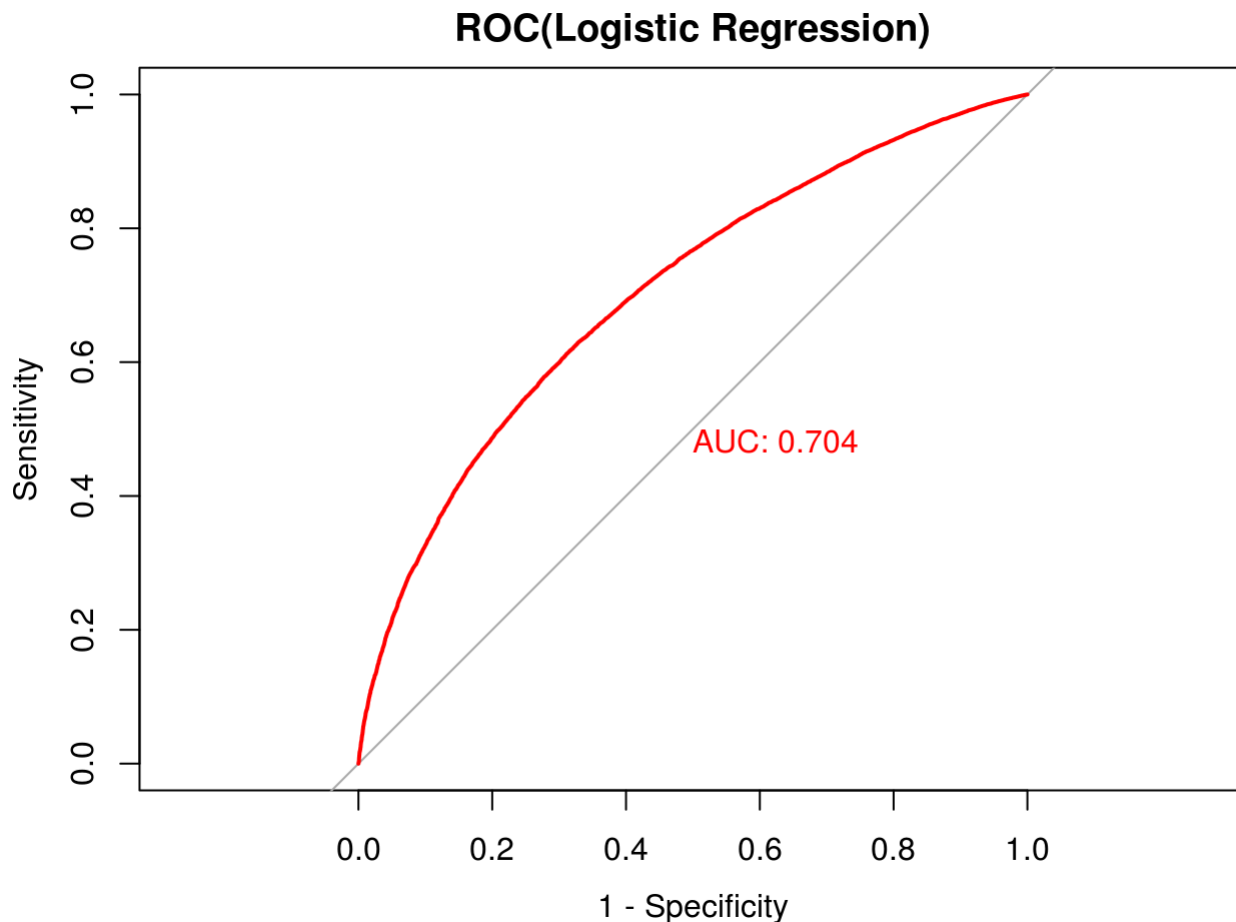
```

```

auc_curve = auc(rocCurve_logit)

plot(rocCurve_logit,legacy.axes = TRUE,print.auc = TRUE,col="red",main="ROC(Logistic Reg
ression)")

```



```
##
## Call:
## roc.default(response = test_data$loan_status, predictor = predict_loan_status_logit)
##
## Data: predict_loan_status_logit in 18716 controls (test_data$loan_status Default) < 4
8844 cases (test_data$loan_status Fully.Paid).
## Area under the curve: 0.7042
```

```
#make a prediction on 50% cutoff
predict_loan_status_label = ifelse(predict_loan_status_logit<0.5,"Default","Fully.Paid")
c = confusionMatrix(predict_loan_status_label,test_data$loan_status,positive="Fully.Paid")

table_perf[1,] = c("logistic regression",
  round(auc_curve,3),
  as.numeric(round(c$overall["Accuracy"],3)),
  as.numeric(round(c$byClass["Sensitivity"],3)),
  as.numeric(round(c$byClass["Specificity"],3)),
  as.numeric(round(c$overall["Kappa"],3))
)
rm(samp,train_index)
```

The model's performance is as follow

```
tail(table_perf,1)
```

```
##
## 1 logistic regression 0.704 0.643 0.636 0.662 0.251
```

## SVM

For SVM, we use Radial Basis as a kernel function. Due to limited computation reason, we use 5% of downsampling data for tuning parameter and 10% of downsampling data for training.

```
set.seed(200)
#down sampling again so than we get more info when stacking
samp = downSample(data1[-getIndexesOfColumns(data1, c( "loan_status" )],data1$loan_status,
yname="loan_status")
#choose small data for tuning
train_index_tuning = createDataPartition(samp$loan_status,p = 0.05,list=FALSE,times=1)
#choose small data for re-train
train_index_training = createDataPartition(samp$loan_status,p = 0.1,list=FALSE,times=1)
```

```

svmGrid = expand.grid(
  .sigma = as.numeric(sigest(loan_status ~., data =
samp[train_index_tuning,], scaled=FALSE)),
  .C = c(0.1,1,10)
)

svmTuned = train(
  samp[train_index_tuning, -getIndxsOfColumns(samp, "loan_status")],
  y = samp[train_index_tuning, "loan_status"],
  method = "svmRadial",
  tuneGrid = svmGrid,
  metric = "ROC",
  trControl = ctrl,
  preProcess = NULL,
  scaled = FALSE,
  fit = FALSE)

plot(svmTuned)

svmTuned

```

The best parameter for the model is `sigma = 0.003909534`, and `c = 0.1`. We use this values to fit the 10% of downsampling data and collect its performance based on test set.

```

svm_model = ksvm(loan_status ~ .,
  data = samp[train_index_training,],
  kernel = "rbfdot",
  kpar = list(sigma=0.003909534),
  C = 0.1,
  prob.model = TRUE,
  scaled = FALSE)

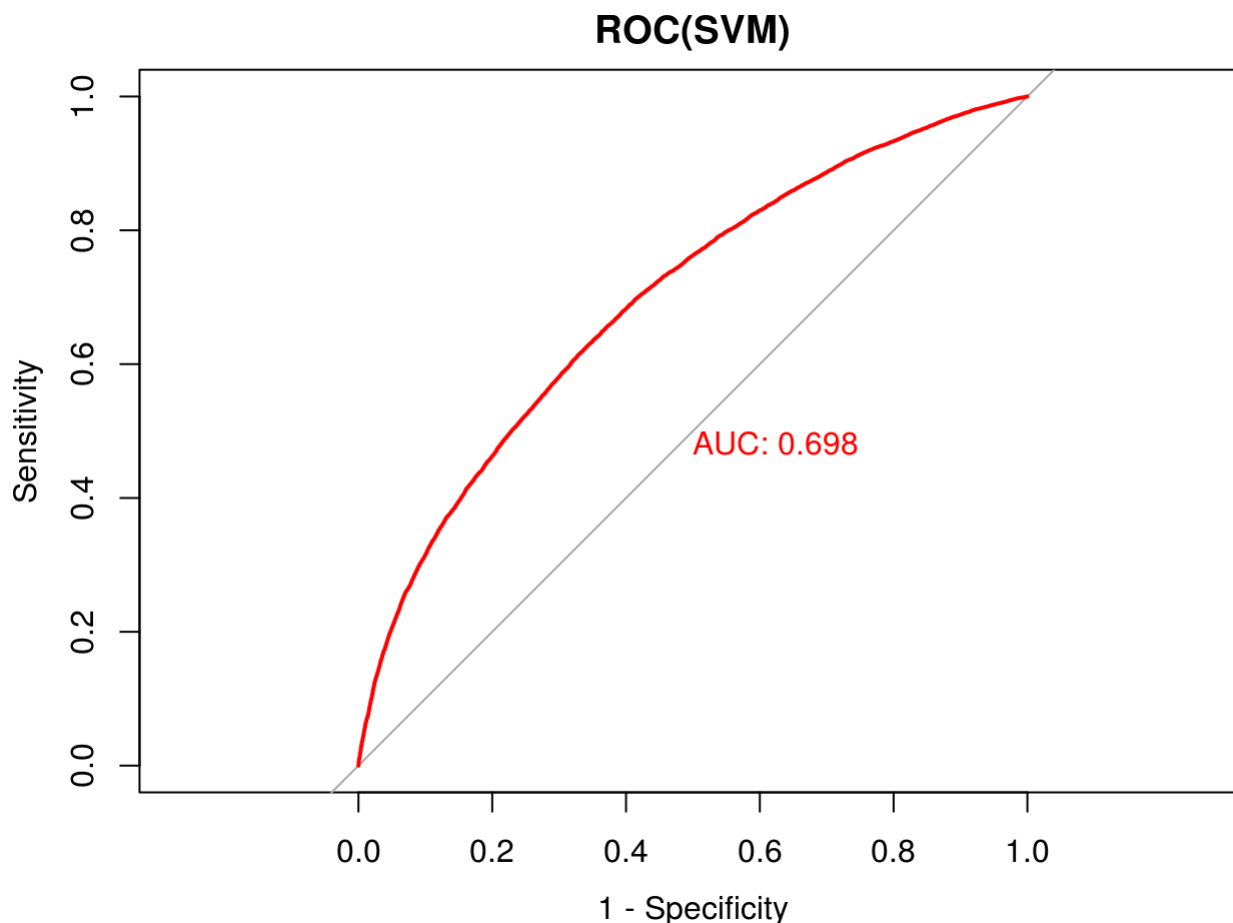
predict_loan_status_svm = predict(svm_model, test_data, type="probabilities")
predict_loan_status_svm = as.data.frame(predict_loan_status_svm)$Fully.Paid

rocCurve_svm = roc(response = test_data$loan_status,
  predictor = predict_loan_status_svm)

auc_curve = auc(rocCurve_svm)

plot(rocCurve_svm, legacy.axes = TRUE, print.auc = TRUE, col="red", main="ROC(SVM)")

```



```
##
## Call:
## roc.default(response = test_data$loan_status, predictor = predict_loan_status_svm)
##
## Data: predict_loan_status_svm in 18716 controls (test_data$loan_status Default) < 488
44 cases (test_data$loan_status Fully.Paid).
## Area under the curve: 0.6976
```

```
predict_loan_status_label = ifelse(predict_loan_status_svm<0.5,"Default","Fully.Paid")
c = confusionMatrix(predict_loan_status_label,test_data$loan_status,positive="Fully.Paid")
```

This is the summary of model's performance

```
table_perf[2,] = c("SVM",
  round(auc_curve,3),
  as.numeric(round(c$overall["Accuracy"],3)),
  as.numeric(round(c$byClass["Sensitivity"],3)),
  as.numeric(round(c$byClass["Specificity"],3)),
  as.numeric(round(c$overall["Kappa"],3))
)

tail(table_perf,1)
```

```
##   model   auc accuracy sensitivity specificity kappa
## 2    SVM 0.698    0.592      0.536      0.739 0.213
```

## RandomForest

Now, we tune RandomForest model. Like SVM, we tune parameter based on 5% downsampling data.

```
set.seed(300)
#down sampling again so than we get more info when stacking
samp = downSample(data1[-getIndxsOfColumns(data1, c( "loan_status" )],data1$loan_status,
yname="loan_status")
#choose small data for tuning
train_index_tuning = createDataPartition(samp$loan_status,p = 0.05,list=FALSE,times=1)
#choose small data for re-train
train_index_training = createDataPartition(samp$loan_status,p = 0.1,list=FALSE,times=1)
```

```
rfGrid = expand.grid(
  .mtry = as.integer(seq(2,ncol(samp), (ncol(samp) - 2)/4))
)

rfTuned = train(
  samp[train_index_tuning,-getIndxsOfColumns(samp,"loan_status")],
  y = samp[train_index_tuning,"loan_status"],
  method = "rf",
  tuneGrid = rfGrid,
  metric = "ROC",
  trControl = ctrl,
  preProcess = NULL,
  ntree = 100
)

plot(rfTuned)

rfTuned
```

The best parameter is `mtry` (number of predictors) = 2. Like SVM, we fit 10% of downsampling data with this value.

```
rf_model = randomForest(loan_status ~ . ,data = samp[train_index_training,],mtry = 2,ntree=400)

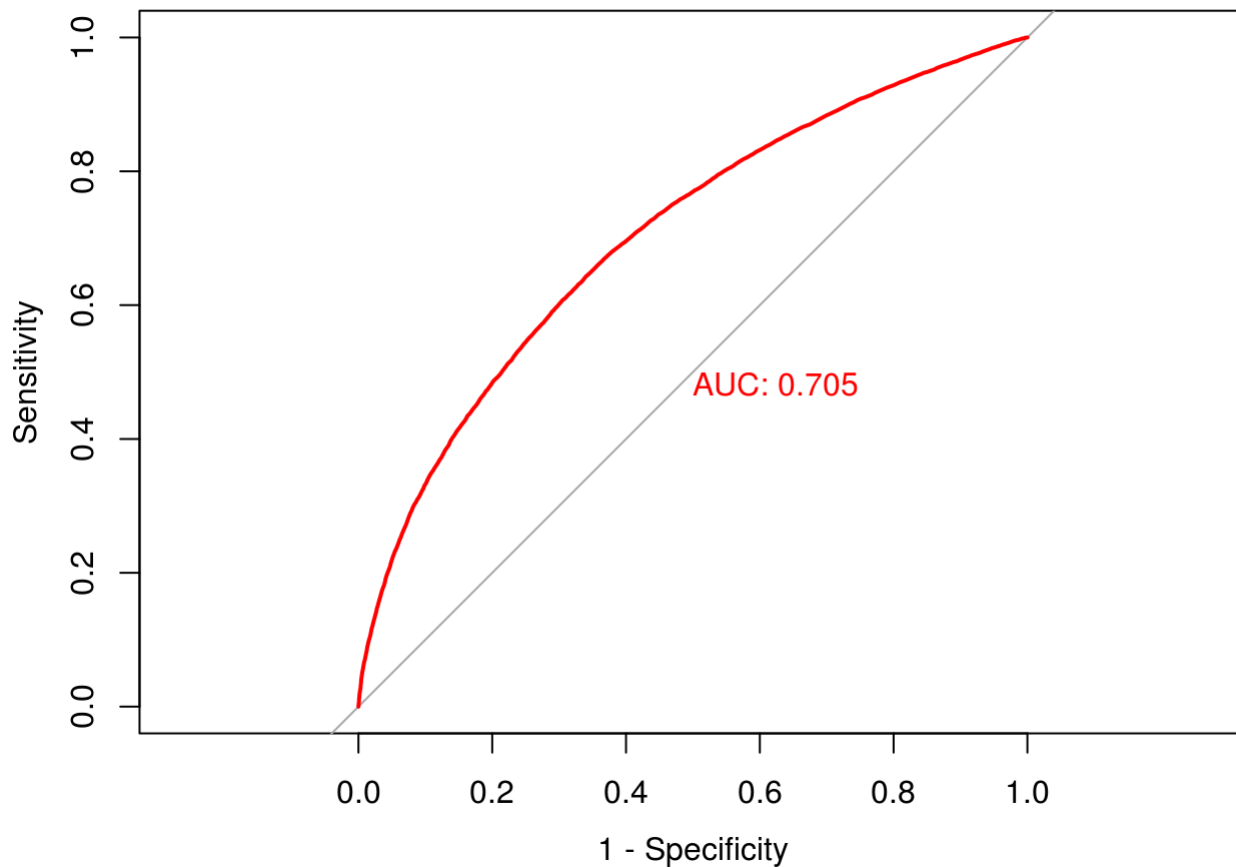
predict_loan_status_rf = predict(rf_model,test_data,"prob")
predict_loan_status_rf = as.data.frame(predict_loan_status_rf)$Fully.Paid

rocCurve_rf = roc(response = test_data$loan_status,
  predictor = predict_loan_status_rf)

auc_curve = auc(rocCurve_rf)

plot(rocCurve_rf,legacy.axes = TRUE,print.auc = TRUE,col="red",main="ROC(RandomForest)")
```

## ROC(RandomForest)



```
##  
## Call:  
## roc.default(response = test_data$loan_status, predictor = predict_loan_status_rf)  
##  
## Data: predict_loan_status_rf in 18716 controls (test_data$loan_status Default) < 4884  
## cases (test_data$loan_status Fully.Paid).  
## Area under the curve: 0.7046
```

```
predict_loan_status_label = ifelse(predict_loan_status_rf<0.5, "Default", "Fully.Paid")  
c = confusionMatrix(predict_loan_status_label, test_data$loan_status, positive="Fully.Paid")  
  
table_perf[3,] = c("RandomForest",  
  round(auc_curve,3),  
  as.numeric(round(c$overall["Accuracy"],3)),  
  as.numeric(round(c$byClass["Sensitivity"],3)),  
  as.numeric(round(c$byClass["Specificity"],3)),  
  as.numeric(round(c$overall["Kappa"],3))  
)
```

The model's performance is as follow

```
tail(table_perf,1)
```

```
##           model   auc accuracy sensitivity specificity kappa
## 3 RandomForest 0.705    0.638      0.622      0.679  0.25
```

# Extreme Gradient Boosting

Extreme Gradient Boosting has a very efficient implementation. Unlike SVM and RandomForest, we can tune parameter using the whole downsampling set. We focus on varying Reidge & Lasso regularization and learning rate. We use 10% of data for validating tuning parameter.

```
set.seed(400)
#down sampling again so than we get more info when stacking
samp = downSample(data1[-getIndxsOfColumns(data1, c( "loan_status" )],data1$loan_status,
yname="loan_status")
#choose small data for validating
train_index_tuning= createDataPartition(samp$loan_status,p = 0.1,list=FALSE,times=1)
```

```
etas = c(0.1,0.3)
alphas = c(0,0.5,1)
lambdas = c(0,0.5,1)

test_watchlist = list(
  test = xgb.DMatrix(
    data = as.matrix(samp[train_index_tuning,][getNumericColumns(samp)]),
    label = as.numeric(samp[train_index_tuning,"loan_status"])-1
  )
)

gbm_perf = data.frame(eta=numeric(0),alpha=numeric(0),lambda=numeric(0),auc=numeric(0))
for(eta in etas){
  for(alpha in alphas){
    for(lambda in lambdas){
      model = xgb.train(
        data= xgb.DMatrix(
          data = as.matrix(samp[-train_index_tuning,]
[getNumericColumns(samp)]),
          label = as.numeric(samp[-train_index_tuning,"loan_status"])-1
        ),
        objective = "binary:logistic",
        nrounds = 350,
        watchlist = test_watchlist,
        eval_metric = "auc",
        early.stop.round = 10,
        alpha = alpha,
        lambda = lambda,
        eta = eta)
      gbm_perf[nrow(gbm_perf)+1,] = c(eta,alpha,lambda,model$bestScore)
    }
  }
}

gbm_perf %>% arrange(desc(auc))
```

The best tuning parameter is  $\eta = 0.1$ ,  $\alpha = 0.5$ , and  $\lambda = 1.0$ . We retrain it again here in case readers didn't run the tuning code. We collect its performance.

```
set.seed(400)
test_watchlist = list(
  test = xgb.DMatrix(
    data = as.matrix(samp[train_index_tuning,][getNumericColumns(samp)]),
    label = as.numeric(samp[train_index_tuning,"loan_status"])-1
  )
)

xgb_model = xgb.train(
  data= xgb.DMatrix(
    data = as.matrix(samp[-train_index_tuning,]
[getNumericColumns(samp)]),
    label = as.numeric(samp[-train_index_tuning,"loan_status"])-1
  ),
  objective = "binary:logistic",
  nrounds = 350,
  watchlist = test_watchlist,
  eval_metric = "auc",
  early.stop.round = 10,
  alpha = 0.5,
  lambda = 1.0,
  eta = 0.1)
```



```
## [0] test-auc:0.692064
## [1] test-auc:0.693999
## [2] test-auc:0.697318
## [3] test-auc:0.699014
## [4] test-auc:0.699192
## [5] test-auc:0.700705
## [6] test-auc:0.701450
## [7] test-auc:0.702239
## [8] test-auc:0.702641
## [9] test-auc:0.703252
## [10] test-auc:0.703859
## [11] test-auc:0.704341
## [12] test-auc:0.704865
## [13] test-auc:0.705212
## [14] test-auc:0.705281
## [15] test-auc:0.705811
## [16] test-auc:0.706393
## [17] test-auc:0.706762
## [18] test-auc:0.707394
## [19] test-auc:0.707741
## [20] test-auc:0.708143
## [21] test-auc:0.708453
## [22] test-auc:0.708823
## [23] test-auc:0.709150
## [24] test-auc:0.709570
## [25] test-auc:0.709920
## [26] test-auc:0.710222
## [27] test-auc:0.710539
## [28] test-auc:0.710843
## [29] test-auc:0.711221
## [30] test-auc:0.711444
## [31] test-auc:0.711497
## [32] test-auc:0.711852
## [33] test-auc:0.712029
## [34] test-auc:0.712396
## [35] test-auc:0.712686
## [36] test-auc:0.712853
## [37] test-auc:0.712997
## [38] test-auc:0.713061
## [39] test-auc:0.713231
## [40] test-auc:0.713370
## [41] test-auc:0.713491
## [42] test-auc:0.713599
## [43] test-auc:0.713893
## [44] test-auc:0.713867
## [45] test-auc:0.714141
## [46] test-auc:0.714303
## [47] test-auc:0.714376
## [48] test-auc:0.714483
## [49] test-auc:0.714808
## [50] test-auc:0.715020
## [51] test-auc:0.715061
## [52] test-auc:0.715091
```

```
## [53] test-auc:0.715098
## [54] test-auc:0.715198
## [55] test-auc:0.715272
## [56] test-auc:0.715473
## [57] test-auc:0.715698
## [58] test-auc:0.715782
## [59] test-auc:0.716132
## [60] test-auc:0.716180
## [61] test-auc:0.716102
## [62] test-auc:0.716098
## [63] test-auc:0.716191
## [64] test-auc:0.716454
## [65] test-auc:0.716429
## [66] test-auc:0.716613
## [67] test-auc:0.716584
## [68] test-auc:0.716636
## [69] test-auc:0.716661
## [70] test-auc:0.716861
## [71] test-auc:0.716958
## [72] test-auc:0.717042
## [73] test-auc:0.717406
## [74] test-auc:0.717539
## [75] test-auc:0.717514
## [76] test-auc:0.717457
## [77] test-auc:0.717645
## [78] test-auc:0.717704
## [79] test-auc:0.717659
## [80] test-auc:0.717606
## [81] test-auc:0.717711
## [82] test-auc:0.717821
## [83] test-auc:0.717667
## [84] test-auc:0.717834
## [85] test-auc:0.717749
## [86] test-auc:0.717893
## [87] test-auc:0.717920
## [88] test-auc:0.717964
## [89] test-auc:0.718027
## [90] test-auc:0.718159
## [91] test-auc:0.718124
## [92] test-auc:0.718170
## [93] test-auc:0.718174
## [94] test-auc:0.718334
## [95] test-auc:0.718440
## [96] test-auc:0.718514
## [97] test-auc:0.718610
## [98] test-auc:0.718661
## [99] test-auc:0.718525
## [100] test-auc:0.718644
## [101] test-auc:0.718732
## [102] test-auc:0.718867
## [103] test-auc:0.718806
## [104] test-auc:0.718848
## [105] test-auc:0.718819
## [106] test-auc:0.718930
```

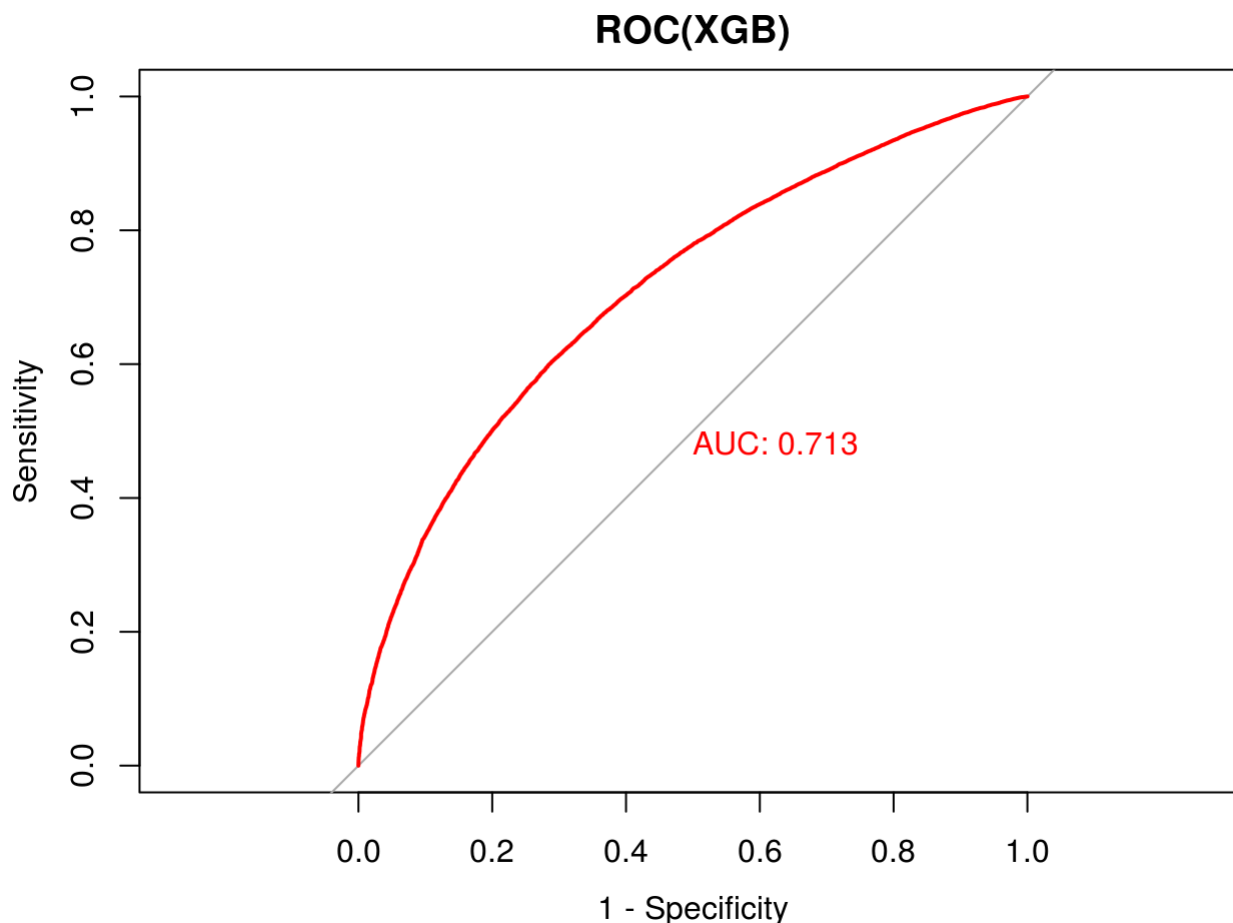
```
## [107] test-auc:0.718979
## [108] test-auc:0.719016
## [109] test-auc:0.719053
## [110] test-auc:0.719173
## [111] test-auc:0.719311
## [112] test-auc:0.719291
## [113] test-auc:0.719311
## [114] test-auc:0.719341
## [115] test-auc:0.719551
## [116] test-auc:0.719563
## [117] test-auc:0.719583
## [118] test-auc:0.719598
## [119] test-auc:0.719521
## [120] test-auc:0.719502
## [121] test-auc:0.719677
## [122] test-auc:0.719646
## [123] test-auc:0.719615
## [124] test-auc:0.719596
## [125] test-auc:0.719609
## [126] test-auc:0.719571
## [127] test-auc:0.719693
## [128] test-auc:0.719664
## [129] test-auc:0.719737
## [130] test-auc:0.719680
## [131] test-auc:0.719655
## [132] test-auc:0.719715
## [133] test-auc:0.719762
## [134] test-auc:0.719814
## [135] test-auc:0.719833
## [136] test-auc:0.719678
## [137] test-auc:0.719681
## [138] test-auc:0.719712
## [139] test-auc:0.719671
## [140] test-auc:0.719693
## [141] test-auc:0.719686
## [142] test-auc:0.719705
## [143] test-auc:0.719756
## [144] test-auc:0.719792
## [145] test-auc:0.719727
## Stopping. Best iteration: 136
```

```
predict_loan_status_xgb = predict(xgb_model,as.matrix(test_data[getNumericColumns(test_data)]))

rocCurve_xgb = roc(response = test_data$loan_status,
                    predictor = predict_loan_status_xgb)

auc_curve = auc(rocCurve_xgb)

plot(rocCurve_xgb,legacy.axes = TRUE,print.auc = TRUE,col="red",main="ROC(XGB)")
```



```
##
## Call:
## roc.default(response = test_data$loan_status, predictor = predict_loan_status_xgb)
##
## Data: predict_loan_status_xgb in 18716 controls (test_data$loan_status Default) < 488
44 cases (test_data$loan_status Fully.Paid).
## Area under the curve: 0.7128
```

```
predict_loan_status_label = ifelse(predict_loan_status_xgb<0.5,"Default","Fully.Paid")
c = confusionMatrix(predict_loan_status_label,test_data$loan_status,positive="Fully.Paid")

table_perf[4,] = c("XGB",
  round(auc_curve,3),
  as.numeric(round(c$overall["Accuracy"],3)),
  as.numeric(round(c$byClass["Sensitivity"],3)),
  as.numeric(round(c$byClass["Specificity"],3)),
  as.numeric(round(c$overall["Kappa"],3))
)
```

The model's performance is as follow

```
tail(table_perf,1)
```

```
##   model   auc accuracy sensitivity specificity kappa
## 4   XGB 0.713    0.607      0.549      0.759 0.239
```

## Averaging Ensemble

Our final model is to combine the result of previous machine learning models and provide a single prediction by averaging probabilities from all previous models.

```
predict_loan_status_ensemble = predict_loan_status_logit +
                                predict_loan_status_svm +
                                predict_loan_status_rf +
                                predict_loan_status_xgb

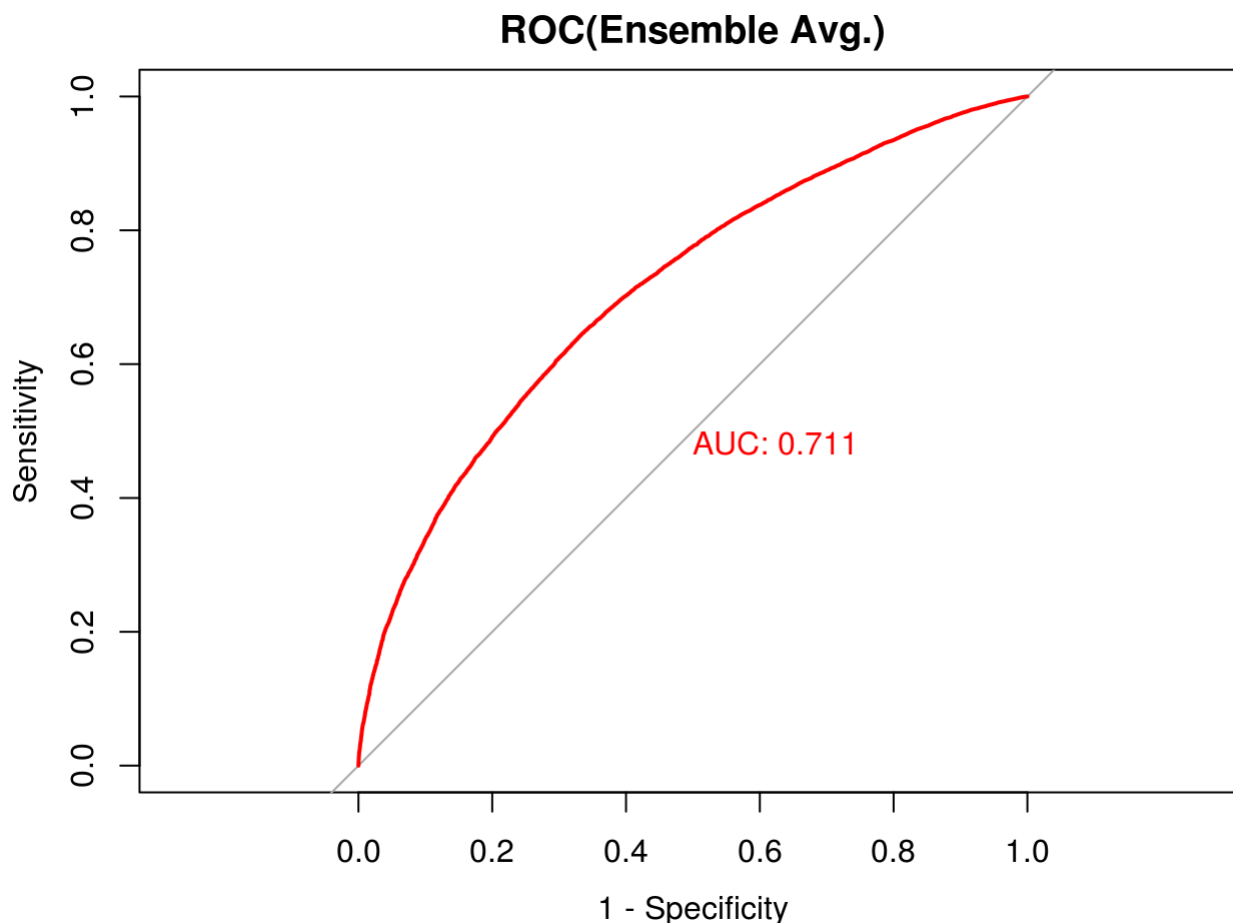
predict_loan_status_ensemble = predict_loan_status_ensemble / 4

rocCurve_ensemble = roc(response = test_data$loan_status,
                        predictor = predict_loan_status_ensemble)
```

```
## Warning in roc.default(response = test_data$loan_status, predictor =
## predict_loan_status_ensemble): Deprecated use a matrix as predictor.
## Unexpected results may be produced, please pass a numeric vector.
```

```
auc_curve = auc(rocCurve_ensemble)

plot(rocCurve_ensemble, legacy.axes = TRUE, print.auc = TRUE, col="red", main="ROC(Ensemble
Avg.)")
```



```
##
## Call:
## roc.default(response = test_data$loan_status, predictor = predict_loan_status_ensembl
## e)
##
## Data: predict_loan_status_ensemble in 18716 controls (test_data$loan_status Default)
## < 48844 cases (test_data$loan_status Fully.Paid).
## Area under the curve: 0.7113
```

```
predict_loan_status_label = ifelse(predict_loan_status_ensemble<0.5,"Default","Fully.Pai
d")
c = confusionMatrix(predict_loan_status_label,test_data$loan_status,positive="Fully.Pai
d")

table_perf[5,] = c("Ensemble",
  round(auc_curve,3),
  as.numeric(round(c$overall["Accuracy"],3)),
  as.numeric(round(c$byClass["Sensitivity"],3)),
  as.numeric(round(c$byClass["Specificity"],3)),
  as.numeric(round(c$overall["Kappa"],3))
)
```

We get the following performance

```
tail(table_perf,1)
```

```
##      model   auc accuracy sensitivity specificity kappa
## 5 Ensemble 0.711    0.623      0.586      0.72 0.246
```

# Model Comparison

AUC for each model and their performance when we set probability cutoff at 50% is summarised below

```
table_perf
```

```
##           model   auc accuracy sensitivity specificity kappa
## 1 logistic regression 0.704    0.643      0.636      0.662 0.251
## 2           SVM 0.698    0.592      0.536      0.739 0.213
## 3      RandomForest 0.705    0.638      0.622      0.679 0.25
## 4           XGB 0.713    0.607      0.549      0.759 0.239
## 5           Ensemble 0.711    0.623      0.586      0.72 0.246
```

```
plot(rocCurve_logit,legacy.axes = TRUE,col="red",main="ROC compare")
```

```
##
## Call:
## roc.default(response = test_data$loan_status, predictor = predict_loan_status_logit)
##
## Data: predict_loan_status_logit in 18716 controls (test_data$loan_status Default) < 4
8844 cases (test_data$loan_status Fully.Paid).
## Area under the curve: 0.7042
```

```
plot(rocCurve_svm,legacy.axes = TRUE,col="blue",add=TRUE)
```

```
##
## Call:
## roc.default(response = test_data$loan_status, predictor = predict_loan_status_svm)
##
## Data: predict_loan_status_svm in 18716 controls (test_data$loan_status Default) < 488
44 cases (test_data$loan_status Fully.Paid).
## Area under the curve: 0.6976
```

```
plot(rocCurve_rf,legacy.axes = TRUE,col="green",add=TRUE)
```

```
##  
## Call:  
## roc.default(response = test_data$loan_status, predictor = predict_loan_status_rf)  
##  
## Data: predict_loan_status_rf in 18716 controls (test_data$loan_status Default) < 4884  
4 cases (test_data$loan_status Fully.Paid).  
## Area under the curve: 0.7046
```

```
plot(rocCurve_xgb, legacy.axes = TRUE, col="orange", add=TRUE)
```

```
##  
## Call:  
## roc.default(response = test_data$loan_status, predictor = predict_loan_status_xgb)  
##  
## Data: predict_loan_status_xgb in 18716 controls (test_data$loan_status Default) < 488  
44 cases (test_data$loan_status Fully.Paid).  
## Area under the curve: 0.7128
```

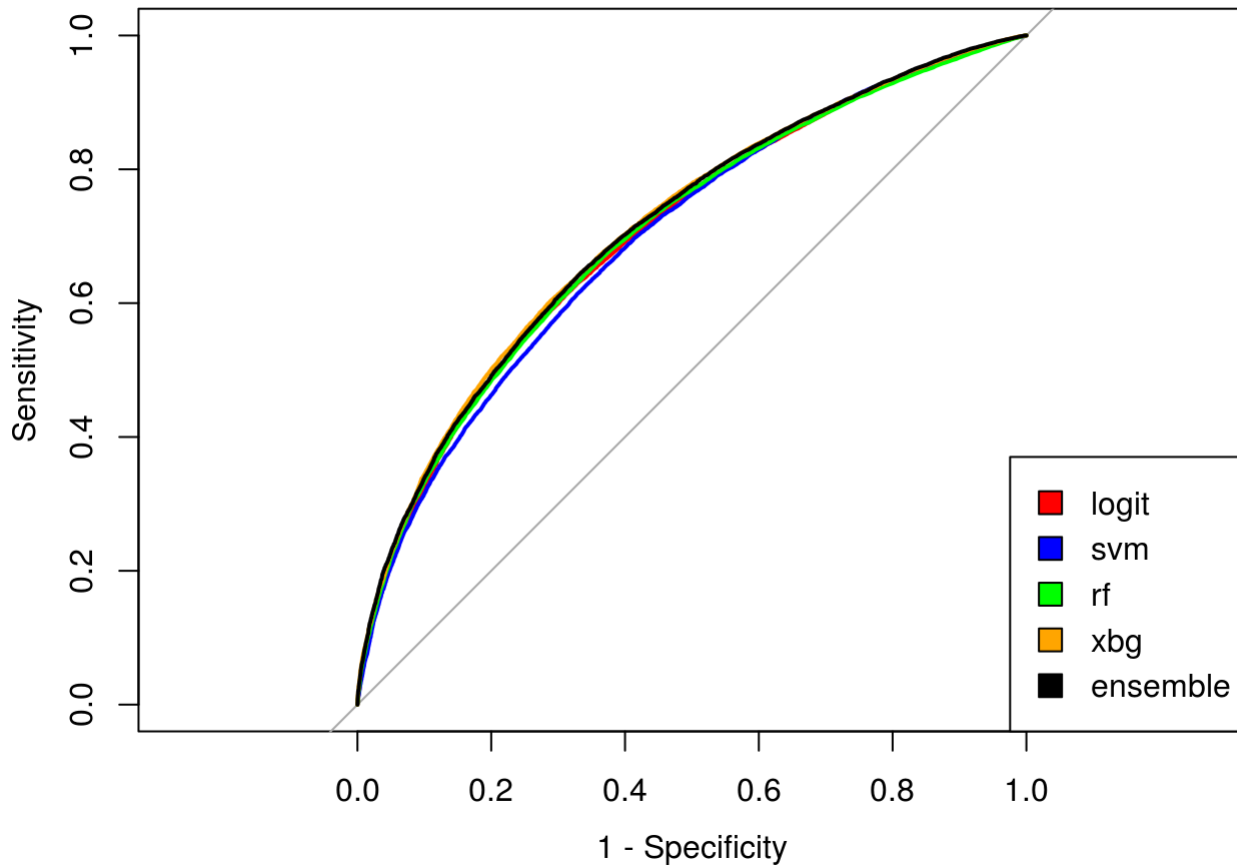
```
plot(rocCurve_ensemble, legacy.axes = TRUE, col="black", add=TRUE)
```

```
##  
## Call:  
## roc.default(response = test_data$loan_status, predictor = predict_loan_status_ensembl  
e)  
##  
## Data: predict_loan_status_ensemble in 18716 controls (test_data$loan_status Default)  
< 48844 cases (test_data$loan_status Fully.Paid).  
## Area under the curve: 0.7113
```

```
legend("bottomright", legend=c("logit", "svm", "rf", "xgb", "ensemble"), fill=c("red", "blue", "gr  
en", "orange", "black"))
```



## ROC compare



Kappa statistics from all models exceed 20% by just small amount, which indicate that they perform moderately better than chance. XGB takes advantage of receiving all downsampling data and provides highest AUC. Comparing performance across models may not be valid, though, because we use different downsampling data for each model. Ensemble model doesn't improve AUC as we expected.

We are surprised to find that Logistic regression does provide a very competitive performance. At 50% cutoff, it yields reasonable compromise between percentage of correctly identified good loans (Sensitivity) and bad loans (Specificity) while not sacrificing Accuracy too much (recall that the naive strategy yields 72.3% accuracy). SVM with RBF kernel has lowest AUC. We can train it with only some portion of data as time complexity of the model rapidly jump up. RandomForest yields comparable result to Logistic Regression. XGB sacrifices Sensitivity rate for Specificity (ability to recall bad loans). It maybe suitable if we really want to avoid default loans. Ensemble model does tune up XGB a little bit. Given the simplicity of Logistic Regression model, and ROC graph are, over all, not significantly difference, we recommend it as a model of choice for predicting LendingClub dataset.

We can calculate investment performance assumed that we follow each model predictions based on 50% cutoff:

```

#assign the features back for evaluation
test_data$grade = test_data_grade
test_data$funded_amnt = test_data_funded_amnt
test_data$total_pymnt = test_data_total_pymnt

#logistic regression
test_data$prediction = ifelse(predict_loan_status_logit<0.5,"Default","Fully Paid")
logit_table = createPerformanceTable(test_data)

#SVM
test_data$prediction = ifelse(predict_loan_status_svm<0.5,"Default","Fully Paid")
svm_table = createPerformanceTable(test_data)

#rf
test_data$prediction = ifelse(predict_loan_status_rf<0.5,"Default","Fully Paid")
rf_table = createPerformanceTable(test_data)

#XGB
test_data$prediction = ifelse(predict_loan_status_xgb<0.5,"Default","Fully Paid")
xgb_table = createPerformanceTable(test_data)

#ensemble
test_data$prediction = ifelse(predict_loan_status_ensemble<0.5,"Default","Fully Paid")
ensemble_table = createPerformanceTable(test_data)

```

For Logistic Regression:

```
logit_table
```

```

##      grade      roi percent_pick
## 1      A -0.02868136  99.9407513
## 2      B -0.07712652  94.1537468
## 3      C -0.11730945  60.4015616
## 4      D -0.14398647  18.4491764
## 5      E -0.12421653   2.1814007
## 6      F -0.16091886   0.1059322
## 7      G  0.00000000   0.0000000
## 8     ALL -0.08203923  55.3463588

```

For SVM:

```
svm_table
```

##	grade	roi	percent_pick
## 1	A	-0.02842189	99.43121223
## 2	B	-0.08390121	99.15374677
## 3	C	-0.11470667	36.73883283
## 4	D	0.01494777	0.77139413
## 5	E	0.09743927	0.05102692
## 6	F	0.00000000	0.00000000
## 7	G	0.00000000	0.00000000
## 8	ALL	-0.07477126	46.01243339

For RandomForest:

rf\_table

##	grade	roi	percent_pick
## 1	A	-0.02860544	100.0000000
## 2	B	-0.08057145	97.6227390
## 3	C	-0.11395171	59.3570958
## 4	D	-0.09056090	8.0112495
## 5	E	-0.13019485	1.6456181
## 6	F	-0.28616990	0.6002825
## 7	G	0.06104822	0.3740648
## 8	ALL	-0.07832291	53.8839550

For XGB:

xgb\_table

##	grade	roi	percent_pick
## 1	A	-0.02755103	99.0046214
## 2	B	-0.06683685	84.5994832
## 3	C	-0.08522953	38.6807281
## 4	D	-0.10742489	14.2868622
## 5	E	-0.11417002	5.2940426
## 6	F	-0.10204509	1.5889831
## 7	G	-0.09492406	0.6234414
## 8	ALL	-0.06275065	46.3632327

For Ensemble model:

ensemble\_table

```
##   grade      roi percent_pick
## 1    A -0.02857455 99.97630051
## 2    B -0.07905501 96.93798450
## 3    C -0.09891564 47.92881407
## 4    D -0.07386266  7.11128967
## 5    E -0.06045371  0.98226815
## 6    F -0.42870486  0.07062147
## 7    G  0.00000000  0.00000000
## 8   ALL -0.07069706 50.11841326
```

Our baseline strategy:

```
baseline_table
```

```
##   grade      roi percent_pick
## 1    A -0.02860544      100
## 2    B -0.08447862      100
## 3    C -0.16181754      100
## 4    D -0.23565662      100
## 5    E -0.28144831      100
## 6    F -0.34315720      100
## 7    G -0.35786380      100
## 8   ALL -0.17560423      100
```

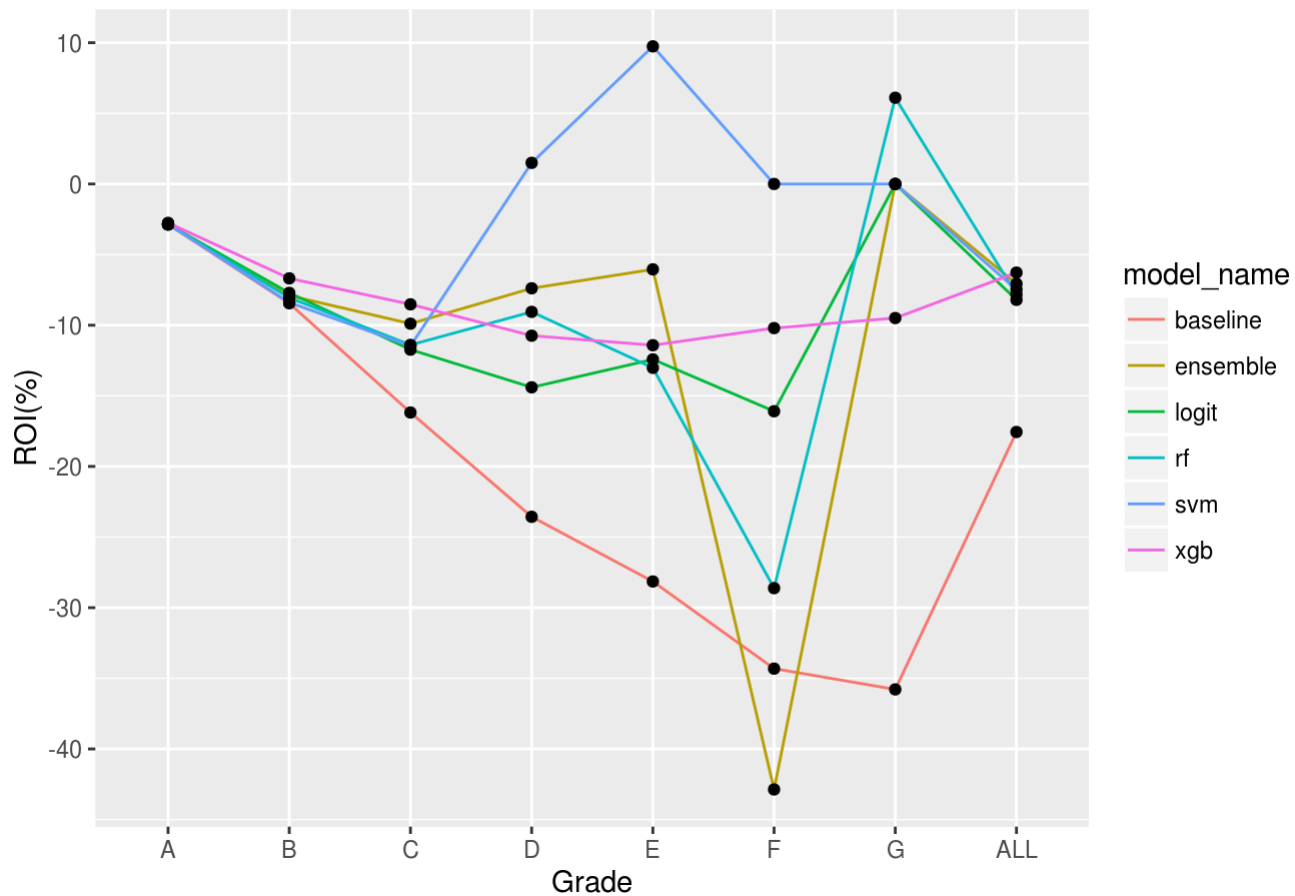
We can visualize returns for each model:

```
logit_table$model_name = "logit"
svm_table$model_name = "svm"
rf_table$model_name = "rf"
xgb_table$model_name = "xgb"
ensemble_table$model_name = "ensemble"
baseline_table$model_name = "baseline"

full_table = rbind(logit_table,svm_table,rf_table,xgb_table,ensemble_table,baseline_table)

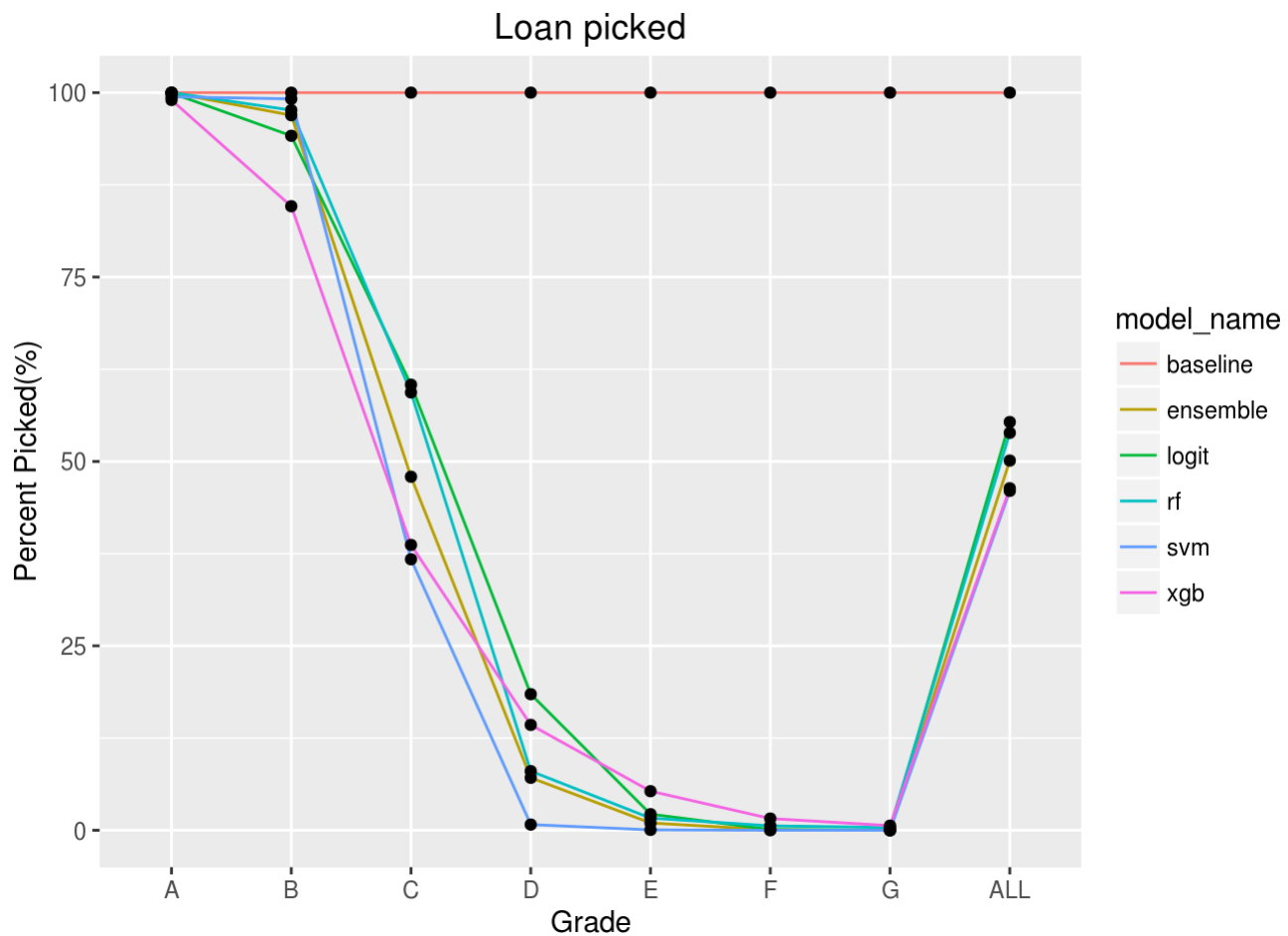
ggplot(full_table,aes(x=grade,y=roi*100,group=model_name)) +
  geom_line(aes(colour = model_name)) +
  geom_point() +
  xlab("Grade") +
  ylab("ROI(%)") +
  labs(title = "Return on identified 2015 loans")
```

Return on identified 2015 loans



and percentage of loans for each grade they picked:

```
ggplot(full_table, aes(x=grade, y=percent_pick, group=model_name)) +
  geom_line(aes(colour = model_name)) +
  geom_point() +
  xlab("Grade") +
  ylab("Percent Picked(%)") +
  labs(title = "Loan picked")
```



All 5 prediction models beat the benchmark strategy in nearly very aspects. Though they done so for low-grade loans by selecting virtually none. Percentage of loan picked by SVM taper off very fast for D, E, F, G grades. It has high Specificity(ability to recall default) so it might be best to use against riskier loans. Many models do shine on predicting middle rate- B, C, D grade - loans, as their ROI is significantly higher than the baseline's, and still have sizable investment.

## Conclusion

Determining the loan outcome, like many financial predictions, is clearly not an easy task. Our models barely go above 20% Kappa Statistics which indicates that chance still play a large role here. They also penalized low grade loans so much that they rather not select anything, and thus prevent us from reaping benefits of higher interest rate. Our finding nevertheless show a promising venue in loan prediction, as higher-than-average ROI can still be gained on certain loan quality. Further analysis may incorporate some discarded features, such as `emp_title`, or external information, like relevant economic status, into the model.