CYO - Credit Card Default

Julieta Peisino

Credit Card default payment

1- Introduction

Using the database of Credit cards users from UCI i'm going to identify what are the parameters that affect most in the probability of defaulting in the pay of the credit card next month. This understanding will be used to predict defaulters.

The database contains information about credit cards payments from April 2005 to September 2005. This information is very useful for banks to be able to predict the default ratio of their clients and improve their strategy to give credit cards to people that will be able to pay.

For this second project i wanted to use a binary variable to be able to use different algorithm than used in the previous one. Here I was able to understand better some of the concepts I learn in the courses

1.1 Download necessary packages

```
if(!require(gdata)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(tidyverse)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(ggplot2)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(knitr)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(gridExtra)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(corrplot)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(rpart)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(matrixStats)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(gam)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(randomForest)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(dplyr)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(gbm)) install.packages("data.table", repos = "http://cran.us.r-project.org")

library(gbm)
library(dplyr)
library(gdata)
library(tidyverse)
library(ggplot2)
library(knitr)
library(gridExtra)
library(corrplot)
library(caret)
library(data.table)
library(rpart)
library(matrixStats)
library(gam)
library(randomForest)
```

1.2 Data Cleaning

First step is to download data from UCI and rename columns to make it easier to understand

```
url <- "http://archive.ics.uci.edu/ml/machine-learning-databases/00350/default%20of%20credit%20card%
creditcard <- read.xls(url)
colnames<-creditcard[1,]
colnames[7]<-"PAY_1"
colnames[25]<-"def"
colnames(creditcard)<-colnames
creditcard<-creditcard[-1,]
```

2- Data analysis

The database is taken from UCI

This research employed a binary variable, def that is the probability of default in payment of next month credit card bill.

def default payment (Yes = 1, No = 0), as the response variable.

This study reviewed the literature and used the following 23 variables as explanatory variables:

Information About client

LIMIT_BAL: Amount of the given credit (NT dollar): it includes both the individual consumer credit and his/her family (supplementary) credit.

SEX: Gender (1 = male; 2 = female).

EDUCATION: Education (1 = graduate school; 2 = university; 3 = high school; 4 = others).

MARRIAGE: Marital status (1 = married; 2 = single; 3 = others).

AGE: Age (year).

Information of previous payments (from apr 2005 to sep 2005)

PAY_1 - PAY_6: History of past payment. We tracked the past monthly payment records (from April to September, 2005) as follows: PAY_1 = the repayment status in September, 2005; X7 = the repayment status in August, 2005; . . .;PAY_6 = the repayment status in April, 2005. The measurement scale for the repayment status is:-2= No consumption ,

-2= No movement

-1 = pay duly

0= The use of revolving credit

1 = payment delay for one month;

2 = payment delay for two months; . . .;

8 = payment delay for eight months;

9 = payment delay for nine months and above.

BILL_AMT1 - BILL_AMT6: Amount of bill statement (NT dollar). BILL_AMT = amount of bill statement

BILL_AMT6 = amount of bill statement in Sep, 2005;

....

BILL_AMT1 = amount of bill statement in April, 2005.

PAY_AMT1-PAY_AMT6: Amount of previous payment (NT dollar).

PAY_AMT1 = amount paid in September, 2005;

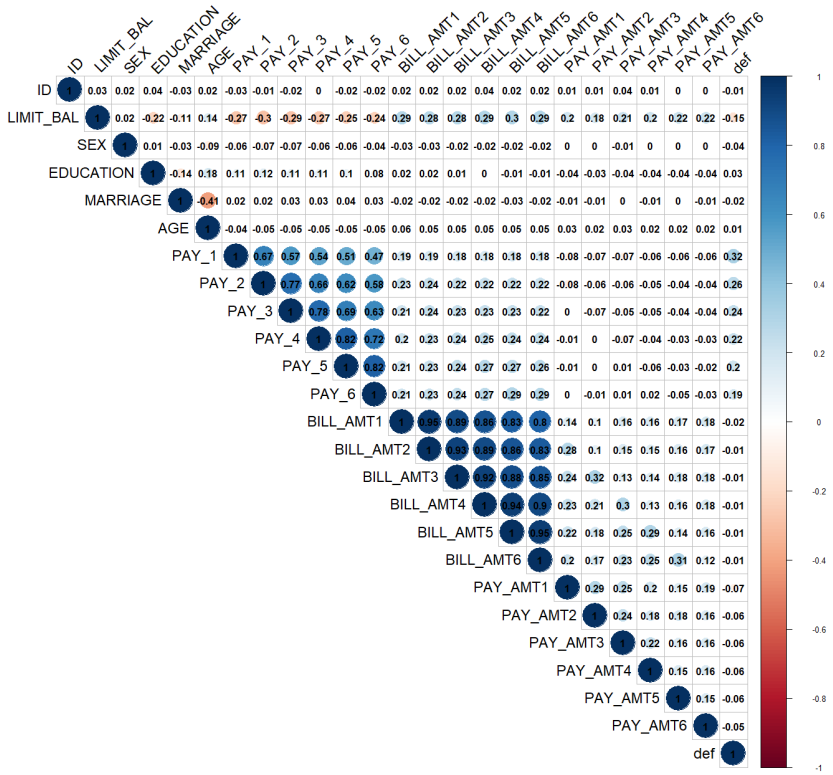
....

PAY_AMT6 = amount paid in April, 2005.

2.1 Correlation matrix

On this matrix we can see the correlation between different variables, on our case we are want to understand the correlation of the variable def= probability of default in next month payment and others.

```
creditcard<-lapply(creditcard, as.numeric)
creditcard<-as.data.frame(creditcard)
r<-cor(as.matrix(creditcard))
corrplot(r, method = "circle", type="upper",sig.level = 0.01,insig = "blank",
         tl.col = "black", tl.srt = 45, tl.cex=1.5 , addCoef.col = "black")
```



We can observe that there is correlation between variable PAY_N and def so it means that the probability of default is related to default in previous payments. Limit Balance and payment amount are also correlated.

Before analyzing I am going to change some variables to make charts easy to understand

```
##SEX data

creditcard$SEX[creditcard$SEX=="1"]<-"male"
creditcard$SEX[creditcard$SEX=="2"]<-"female"

## Education data
creditcard$EDUCATION[creditcard$EDUCATION=="1"]<-"graduate school"
creditcard$EDUCATION[creditcard$EDUCATION=="2"]<-"university"
creditcard$EDUCATION[creditcard$EDUCATION=="3"]<-"high school"
creditcard$EDUCATION[creditcard$EDUCATION=="4"]<-"others"

##Marital status
creditcard$MARRIAGE[creditcard$MARRIAGE=="1"]<-"married"
creditcard$MARRIAGE[creditcard$MARRIAGE=="2"]<-"single"
creditcard$MARRIAGE[creditcard$MARRIAGE=="3"]<-"others"

#Default
creditcard$def[creditcard$def=="1"]<-"yes"
creditcard$def[creditcard$def=="0"]<-"no"
```

2.3 Data Structure

```
dim(creditcard)
```

```
## [1] 30000    25
```

```
str(creditcard)
```

```
## 'data.frame':    30000 obs. of  25 variables:
##  $ ID       : num  1 2 3 4 5 6 7 8 9 10 ...
##  $ LIMIT_BAL: num  20000 120000 90000 50000 50000 50000 500000 100000 140000 20000 ...
##  $ SEX      : chr  "female" "female" "female" "female" ...
##  $ EDUCATION: chr  "university" "university" "university" "university" ...
##  $ MARRIAGE : chr  "married" "single" "single" "married" ...
##  $ AGE      : num  24 26 34 37 57 37 29 23 28 35 ...
##  $ PAY_1    : num  2 -1 0 0 -1 0 0 0 0 -2 ...
##  $ PAY_2    : num  2 2 0 0 0 0 0 0 -1 0 -2 ...
##  $ PAY_3    : num  -1 0 0 0 -1 0 0 -1 2 -2 ...
##  $ PAY_4    : num  -1 0 0 0 0 0 0 0 0 -2 ...
##  $ PAY_5    : num  -2 0 0 0 0 0 0 0 0 -1 ...
##  $ PAY_6    : num  -2 2 0 0 0 0 0 -1 0 -1 ...
##  $ BILL_AMT1: num  3913 2682 29239 46990 8617 ...
##  $ BILL_AMT2: num  3102 1725 14027 48233 5670 ...
##  $ BILL_AMT3: num  689 2682 13559 49291 35835 ...
##  $ BILL_AMT4: num  0 3272 14331 28314 20940 ...
##  $ BILL_AMT5: num  0 3455 14948 28959 19146 ...
##  $ BILL_AMT6: num  0 3261 15549 29547 19131 ...
##  $ PAY_AMT1 : num  0 0 1518 2000 2000 ...
##  $ PAY_AMT2 : num  689 1000 1500 2019 36681 ...
##  $ PAY_AMT3 : num  0 1000 1000 1200 10000 657 38000 0 432 0 ...
##  $ PAY_AMT4 : num  0 1000 1000 1100 9000 ...
##  $ PAY_AMT5 : num  0 0 1000 1069 689 ...
##  $ PAY_AMT6 : num  0 2000 5000 1000 679 ...
##  $ def      : chr  "yes" "yes" "no" "no" ...
```

As mentioned before the database consists of 30000 observations of 23 categories Information about client= Genre, age, education, marital status Information about the product= Amount of the given credit card Amount of previous payments, status of previous payments and bill amount.

2.4 Age

```
##age range
range(creditcard$AGE)
```

```
## [1] 21 79
```

```
mean(creditcard$AGE)
```

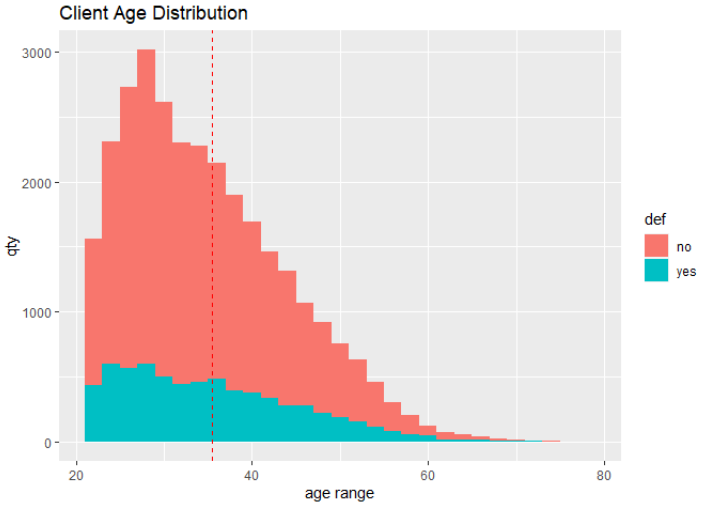
```
## [1] 35.4855
```

```
median(creditcard$AGE)
```

```
## [1] 34
```

Age of the clients is between 21 and 79 years and the mean age is 35 years. We can see that age starts on 21 because is the legal age to have credit card

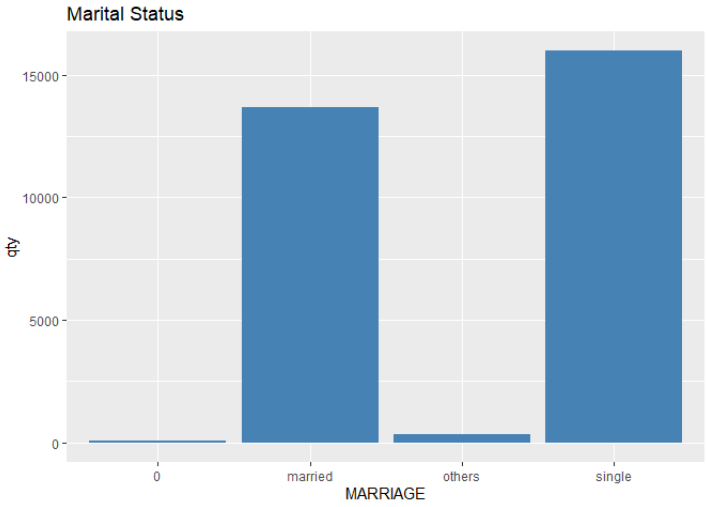
```
##age frec
creditcard%>%ggplot(aes(AGE,fill=def))+geom_histogram(bins = 30)+
  geom_vline(xintercept = mean(creditcard$AGE), lty = 2, color= "Red")+
  labs(title = "Client Age Distribution",x = "age range", y = "qty")
```



On this chart we can see that most of the clients are under 40 (and also defaulters)

2.5 Marital Status

```
##Marital Status count
creditcard%>%ggplot(aes(MARRIAGE))+geom_bar(fill="steelblue")+
  labs(title = "Marital Status", y = "qty")
```

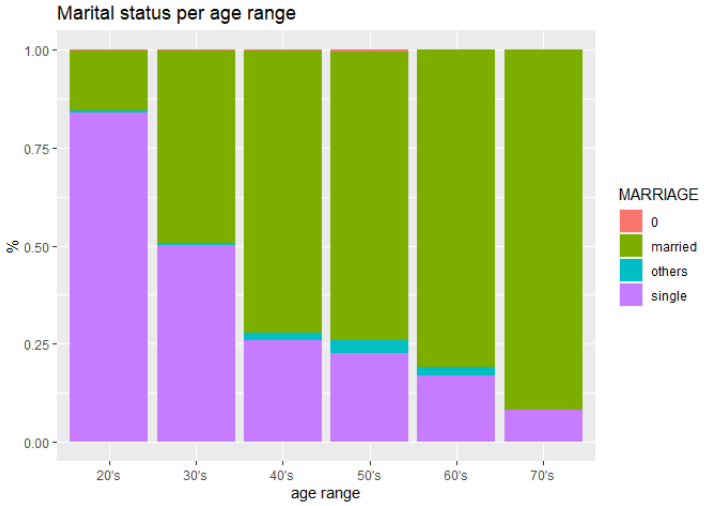



```
creditcard%>%group_by(MARRIAGE)%>%summarize(n=n(),prop=n/30000)%>%
  arrange(desc(prop))%>%kable()
```

MARRIAGE	n	prop
single	15964	0.5321333
married	13659	0.4553000
others	323	0.0107667
0	54	0.0018000

From this chart we can see that majority of clients ar single. And we can observe a category “0”, that was not specified in dataset, so i will assume it as unknown

```
creditcard%>%mutate(age_range=paste0(trunc(AGE/10)*10,"'s"),prop=n())%>%
  ggplot(aes(age_range,prop,fill=MARRIAGE))+
  geom_bar(position="fill", stat="identity")+
  labs(title = "Marital status per age range",x = "age range", y = "%")
```

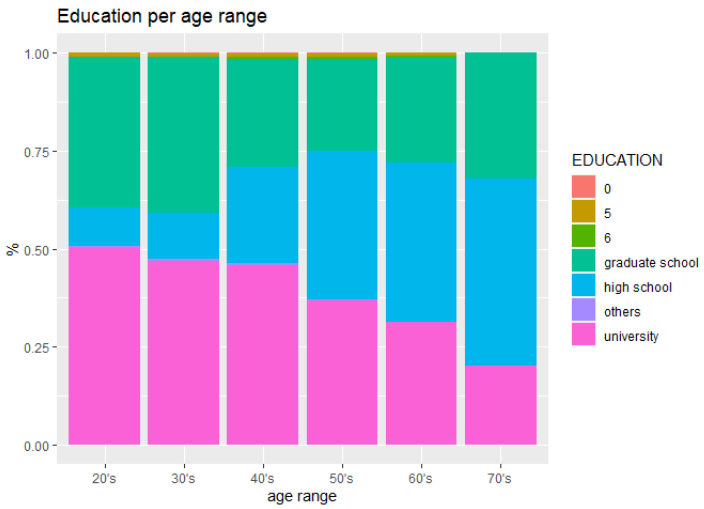


When age increases proportion of married people tend to increase. People in their 20's are majority single, 30's is around 50%/50% and 40's onwards are majority married

2.6 Education

If we do the same analysis for education

```
##Education
creditcard%>%mutate(age_range=paste0(trunc(AGE/10)*10,"'s"),prop=n())%>%
  ggplot(aes(age_range,prop,fill=EDUCATION))+
  geom_bar(position="fill", stat="identity")+
  labs(title = "Education per age range",x = "age range", y = "%")
```



we can see that young people tend to have higher level education

From the total list of customers we can see that more of 80% of the clients have high level education (university/graduate school). It also appear a category 0/5/6 that was not defined in database so we will assume it as unknown

```
creditcard%>%group_by(EDUCATION)%>%summarize(n=n(),prop=n/30000)%>%arrange(desc(prop))%>%kable()
```

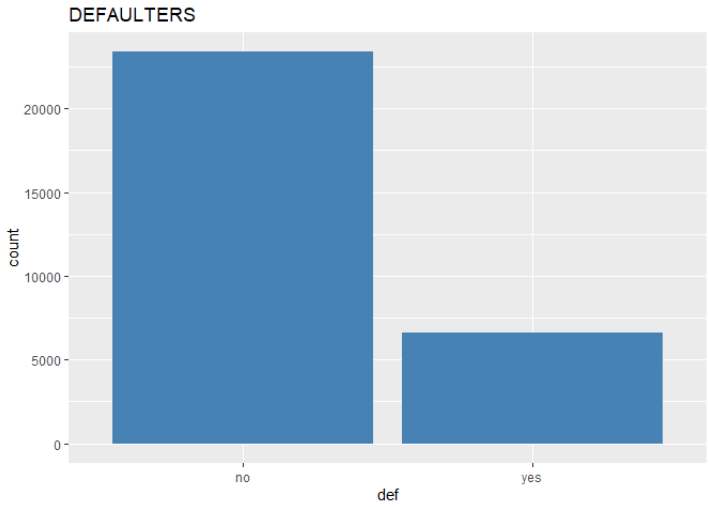
EDUCATION	n	prop
university	14030	0.4676667
graduate school	10585	0.3528333
high school	4917	0.1639000
5	280	0.0093333
others	123	0.0041000
6	51	0.0017000
0	14	0.0004667

So majority of clients are young people with high level education

If we analyze de quantity of defaulters:

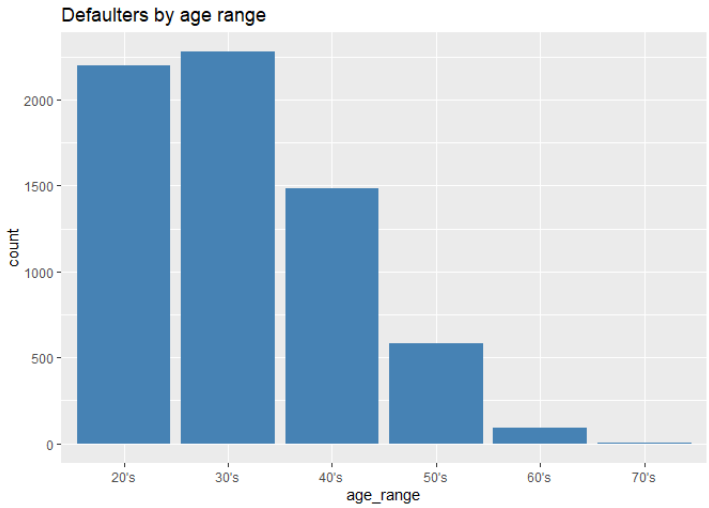
2.6 Defaulters

```
creditcard%>%ggplot(aes(def))+geom_bar(fill="steelblue")+labs(title = "DEFAULTERS")
```



Of the 30000 observations only 6636 person are not going to pay next month, this represents 22.12% of the observation. It looks a high percentage and it is a good idea for the bank to try to take some measures to reduce this.

```
creditcard%>%mutate(age_range=paste0(trunc(AGE/10)*10,"'s"),prop=n())%>%
  filter(def=="yes")%>%ggplot(aes(age_range))+geom_bar(fill="steelblue")+
  labs(title = "Defaulters by age range")
```




```
totaldefault<-length(creditcard$ID[creditcard$def=="yes"])
creditcard%>%mutate(age_range=paste0(trunc(AGE/10)*10,"'s"))%>%
  filter(def=="yes")%>%group_by(age_range)%>%summarize(def_pay=n(),
  prop=paste0(round(def_pay/totaldefault*100,2),"%"))%>%
  arrange(desc(def_pay))%>%kable()
```

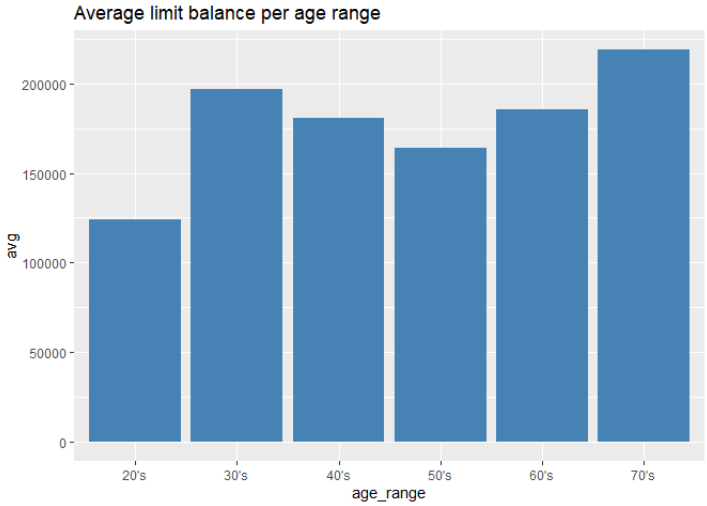
age_range	def_pay	prop
30's	2276	34.3%
20's	2197	33.11%
40's	1485	22.38%
50's	582	8.77%
60's	89	1.34%
70's	7	0.11%

most of the defaults are registered on people under 50.

2.7 Limit Balance

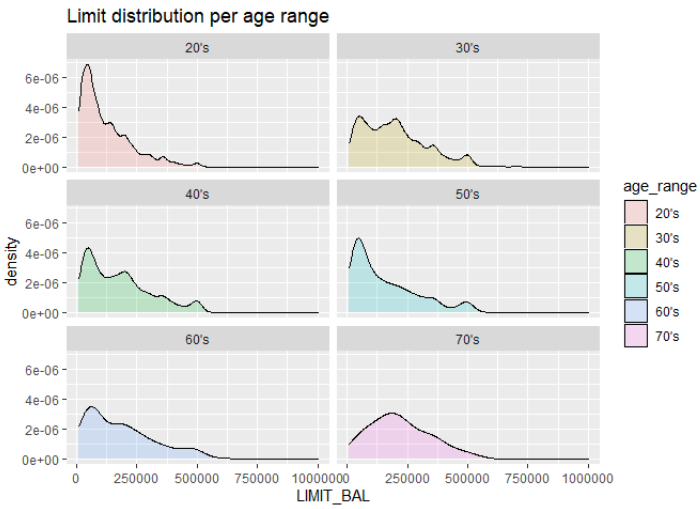
if we analyze limit by age range

```
creditcard%>%mutate(age_range=paste0(trunc(AGE/10)*10,"'s"))%>%
  group_by(age_range)%>%summarize(avg=mean(LIMIT_BAL))%>%
  ggplot(aes(age_range,avg))+geom_bar(stat="identity",fill="steelblue")+
  labs(title = "Average limit balance per age range")
```

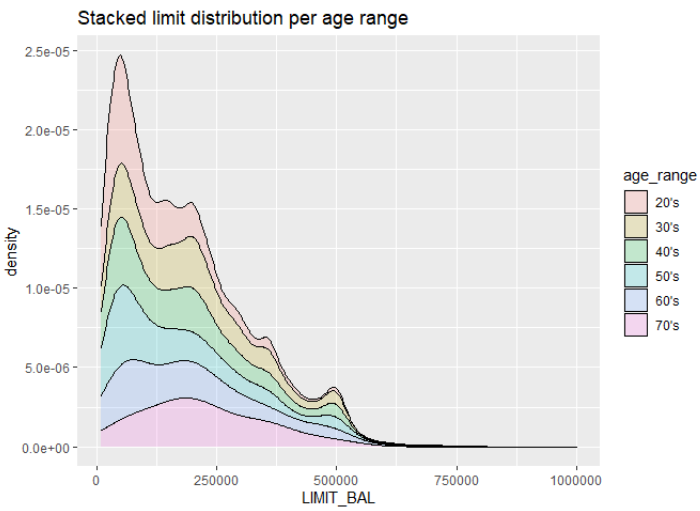


We can see that the smallest average Limit is for people in their 20's and highest for people in their 70's.

```
creditcard%>%mutate(age_range=paste0(trunc(AGE/10)*10,"'s"))%>%
  ggplot(aes(LIMIT_BAL,fill=age_range))+
  geom_density(alpha = 0.2)+
  facet_wrap(age_range ~ .,nrow=3,ncol=2)+
  labs(title = "Limit distribution per age range")
```




```
creditcard%>%mutate(age_range=paste0(trunc(AGE/10)*10,"'s"))%>%
  ggplot(aes(LIMIT_BAL,fill=age_range))+
  geom_density(alpha = 0.2, position = "stack")+
  labs(title = "Stacked limit distribution per age range")
```

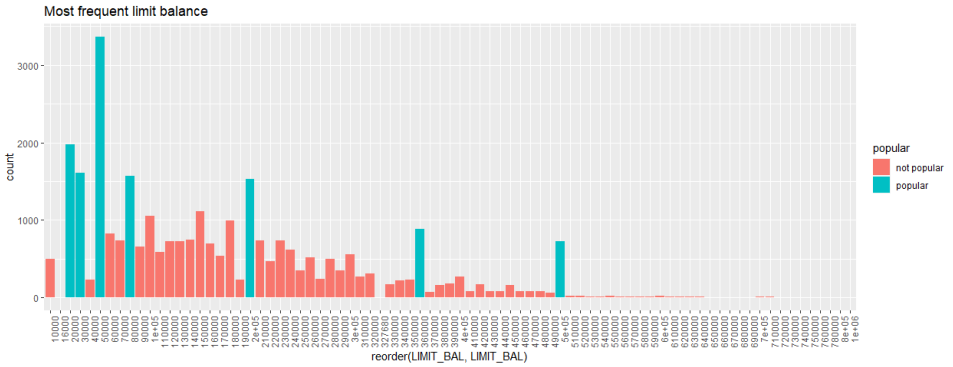


Analyzing the curves we can see that there are some "popular" limits, that we see as bumps in the curve that tend to have more quantity that values in the same range like 50K, 80K, 360K and 500K.

it can be seen more clearly in histogram

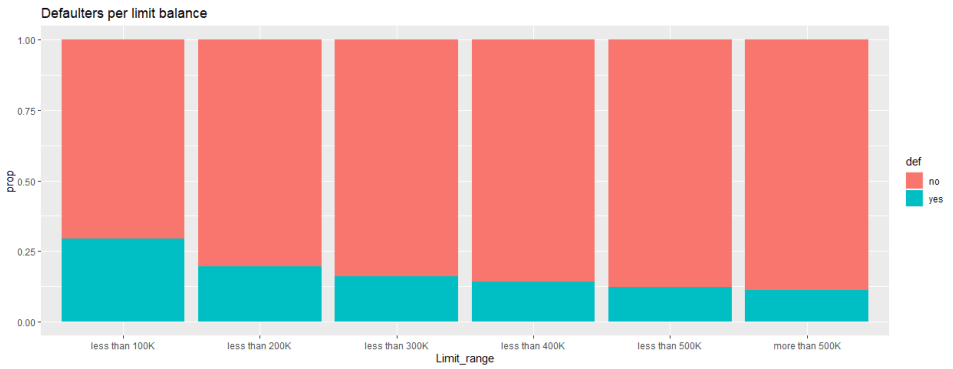
```
creditcard%>%
 mutate(popular=ifelse(LIMIT_BAL%in%c(20000,30000,50000,80000,200000,360000,500000),
                       "popular","not popular"))%>%
 ggplot(aes(x=reorder(LIMIT_BAL,LIMIT_BAL),fill=popular)) +
 geom_bar() +
```

```
  theme(axis.text.x = element_text(angle = 90, hjust = 1))+
  geom_vline(xintercept = 500000, color= "Red")+
  labs(title = "Most frequent limit balance")
```



If we analyze defaults in limit ranges

```
creditcard%>%mutate(Limit_range=
ifelse(LIMIT_BAL<=100000,"less than 100K",
ifelse(LIMIT_BAL<=200000,"less than 200K",
ifelse(LIMIT_BAL<=300000,"less than 300K",
ifelse(LIMIT_BAL<=400000,"less than 400K",
ifelse(LIMIT_BAL<=500000,"less than 500K","more than 500K" ))))),
prop=n())%>%
  ggplot(aes(Limit_range,prop,fill=def))+
  geom_bar(position="fill", stat="identity")+
  labs(title = "Defaulters per limit balance")
```



The default risk is higher for lower rates

```
creditcard%>%mutate(Limit_range=
ifelse(LIMIT_BAL<=100000,"less than 100K",
ifelse(LIMIT_BAL<=200000,"less than 200K",
ifelse(LIMIT_BAL<=300000,"less than 300K",
ifelse(LIMIT_BAL<=400000,"less than 400K",
ifelse(LIMIT_BAL<=500000,"less than 500K","more than 500K" ))))),prop=n())%>%
  group_by(Limit_range)%>%
  summarize(
probability_of_def_on_category=paste0(round(sum(def=="yes")/n()*100,2),"%"),
Proportion_of_total_defaults=paste0(round(sum(def=="yes")/totaldefault*100,2),"%"),
n=n())%>%
  kable()
```

Limit_range	probability_of_def_on_category	Proportion_of_total_defaults	n
less than 100K	29.48%	55.52%	12498
less than 200K	19.48%	23.13%	7880
less than 300K	16.05%	12.24%	5059
less than 400K	14.06%	5.85%	2759
less than 500K	12.14%	2.92%	1598
more than 500K	11.17%	0.35%	206

The default risk is higher for lower rates. For limits under 100K we have 12498 defaults that represent 55.52% of total defaults (more than half of the default are for limits under 100K). And 29.48% of the clients that have limits under 100K are not going to pay their credit cards.

2.8 PAY_1 - PAY_6 (status of payment in previous 6 months)

In the columns PAY_1 to PAY_6 we have the information about past payments PAY_1 corresponds to sep 2005 and PAY_6 to april 2005

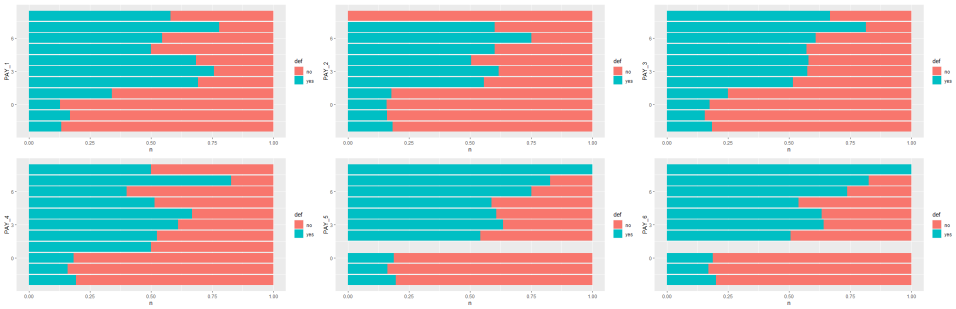
-1 means payed on time

1 delayed 1 month

2 delayed 2 month

etcetera

```
p1<-creditcard%>%mutate(n=n())%>%ggplot(aes(PAY_1,n,fill=def))+
  geom_bar(position="fill", stat="identity")+coord_flip()
p2<-creditcard%>%mutate(n=n())%>%ggplot(aes(PAY_2,n,fill=def))+
  geom_bar(position="fill", stat="identity")+coord_flip()
p3<-creditcard%>%mutate(n=n())%>%ggplot(aes(PAY_3,n,fill=def))+
  geom_bar(position="fill", stat="identity")+coord_flip()
p4<-creditcard%>%mutate(n=n())%>%ggplot(aes(PAY_4,n,fill=def))+
  geom_bar(position="fill", stat="identity")+coord_flip()
p5<-creditcard%>%mutate(n=n())%>%ggplot(aes(PAY_5,n,fill=def))+
  geom_bar(position="fill", stat="identity")+coord_flip()
p6<-creditcard%>%mutate(n=n())%>%ggplot(aes(PAY_6,n,fill=def))+
  geom_bar(position="fill", stat="identity")+coord_flip()
grid.arrange(p1,p2,p3,p4,p5,p6, ncol = 3)
```

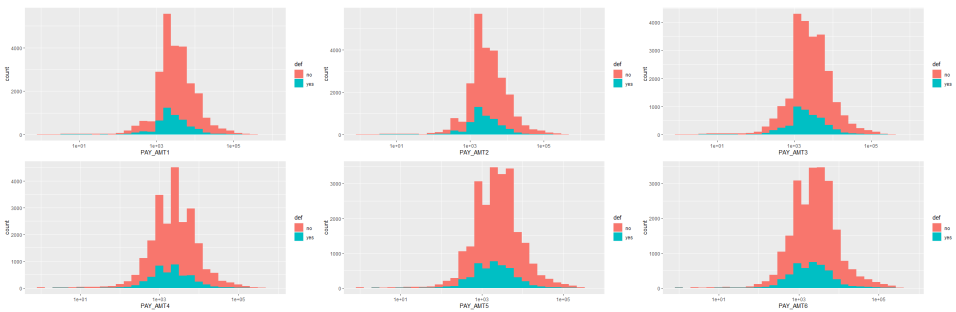


On this set of charts is very clear that when pay>=1 (delayed clients) the proportion of default pays on next month increase.

2.9 PAY_AMT1 - PAY_AMT6 amount payed in previous 6 months

Payment amount for the previous month has a wide range of values (there is not an evident relation between default and amount payed)

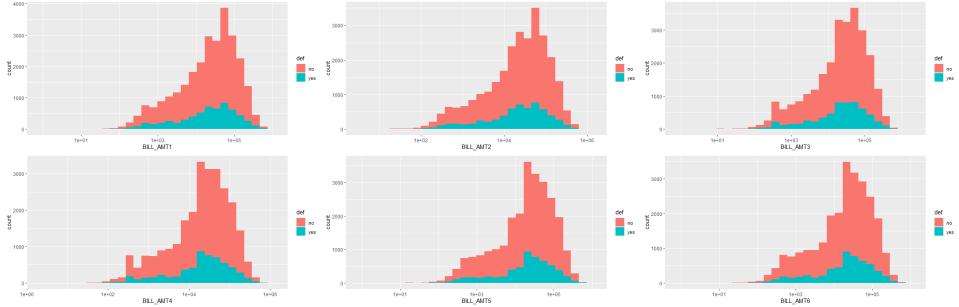
```
am1<-creditcard%>%ggplot(aes(PAY_AMT1,fill=def))+
  geom_histogram()+scale_x_log10()
am2<-creditcard%>%ggplot(aes(PAY_AMT2,fill=def))+
  geom_histogram()+scale_x_log10()
am3<-creditcard%>%ggplot(aes(PAY_AMT3,fill=def))+
  geom_histogram()+scale_x_log10()
am4<-creditcard%>%ggplot(aes(PAY_AMT4,fill=def))+
  geom_histogram()+scale_x_log10()
am5<-creditcard%>%ggplot(aes(PAY_AMT5,fill=def))+
  geom_histogram()+scale_x_log10()
am6<-creditcard%>%ggplot(aes(PAY_AMT6,fill=def))+
  geom_histogram()+scale_x_log10()
grid.arrange(am1,am2,am3,am4,am5,am6, ncol = 3)
```



2.10 BILL_AMT1-BILL_AMT6 (bill of previous 6 months)

Bills amount for the previous month has a wide range of values (there is not an evident relation between default and amount payed)

```
bam1<-creditcard%>%ggplot(aes(BILL_AMT1,fill=def))+
  geom_histogram()+scale_x_log10()
bam2<-creditcard%>%ggplot(aes(BILL_AMT2,fill=def))+
  geom_histogram()+scale_x_log10()
bam3<-creditcard%>%ggplot(aes(BILL_AMT3,fill=def))+
  geom_histogram()+scale_x_log10()
bam4<-creditcard%>%ggplot(aes(BILL_AMT4,fill=def))+
  geom_histogram()+scale_x_log10()
bam5<-creditcard%>%ggplot(aes(BILL_AMT5,fill=def))+
  geom_histogram()+scale_x_log10()
bam6<-creditcard%>%ggplot(aes(BILL_AMT6,fill=def))+
  geom_histogram()+scale_x_log10()
grid.arrange(bam1,bam2,bam3,bam4,bam5,bam6, ncol = 3)
```



3-Models

I am going to use 8 of the models learned in the course.

3.1 Logistic regression

The regression approach can be extended to categorical data. For binary data, one can simply assign numeric values of 0 and 1 to the outcomes y, and apply regression as if the data were continuous.

$p(x) = Pr(Y = 1 | X = x) = \beta 0 + \beta 1x$

Once we have estimates β0 and β1, we can obtain an actual prediction. But the function can take any value including negatives and values larger than 1 and we are estimating a probability: $Pr(Y = 1 \mid X = x)$ which is constrained between 0 and 1. The idea of generalized linear models (GLM) is:

1. define a distribution of Y that is consistent with it's possible outcomes.
2. find a function g so that $g(Pr(Y = 1 \mid X = x))$ can be modeled as a linear combination of predictors. Logistic regression is the most commonly used GLM. It is an extension of linear regression that assures that the estimate of $Pr(Y = 1 \mid X = x)$ is between 0 and 1.

$g(p) = \log (p/(1 - p))$

3.2 Classification Decition Trees

A tree is basically a flow chart of yes or no questions. The general idea of the methods we are describing is to define an algorithm that uses data to create these trees with predictions at the ends, referred to as nodes. Regression and decision trees operate by predicting an outcome variable Y by partitioning the predictors.

Classification trees, or decision trees, are used in prediction problems where the outcome is categorical. We use the same partitioning principle with some differences to account for the fact that we are now working with a categorical outcome.The first difference is that we form predictions by calculating which class is the most common among the training set observations within the partition, rather than taking the average in each partition (as we can't take the average of categories).

The second is that we can no longer use RSS to choose the partition. While we could use the naive approach of looking for partitions that minimize training error, better performing approaches use more sophisticated metrics. Two of the more popular ones are the Gini Index and Entropy.

For this case we have a tuning parameter. CP (complexity parameter) is the minimum improvement in the model needed at each node.

3.3 Random Forest

Random forests are a very popular machine learning approach that addresses the shortcomings of decision trees using a clever idea. The goal is to improve prediction performance and reduce instability by averaging multiple decision trees (a forest of trees constructed with randomness). It has two features that help accomplish this. The first step is bootstrap aggregation or bagging. The general idea is to generate many predictors, each using regression or classification trees, and then forming a final prediction based on the average prediction of all these trees. To assure that the individual trees are not the same, we use the bootstrap to induce randomness. These two features combined explain the name: the bootstrap makes the individual trees randomly different, and the combination of trees is the forest.

there is one tuning paramenter mtry :Number of variables available for splitting at each tree node. In the random forests literature, this is referred to as the mtry parameter

3.4 LDA Linear discriminant Analysis

A relatively simple solution to the problem of having too many parameters is to assume that the correlation structure is the same for all classes, which reduces the number of parameters we need to estimate. In this case, we would compute just one pair of standard deviations and one correlation. When we force this assumption, we can show mathematically that the boundary is a line, just as with logistic regression. For this reason, we call the method linear discriminant analysis (LDA)

3.5 Generalized Additive Model using LOESS (GamLoess)

The gam model is fit using the local scoring algorithm, which iteratively fits weighted additive models by backfitting. The backfitting algorithm is a Gauss-Seidel method for fitting additive models, by iteratively smoothing partial residuals. The algorithm separates the parametric from the nonparametric part of the fit, and fits the parametric part using weighted linear least squares within the backfitting algorithm. This version of gam remains faithful to the philosophy of GAM models as outlined in the references below.

we have 2 tunning parameters span and degree but we will use default because calculation is too long.

3.6 K nearest neighbors

We define the distance between all observations based on the features. Then, for any point (x1, x2) for which we want an estimate of p(x1, x2), we look for the k nearest points to (x1, x2) and then take an average of the 0s and 1s associated with these points. We refer to the set of points used to compute the average as the neighborhood. Due to the connection we described earlier between conditional expectations and conditional probabilities, this gives us a ˆp(x1, x2), just like the bin smoother gave us an estimate of a trend. As with bin smoothers, we can control the flexibility of our estimate, in this case through the k parameter: larger ks result in smoother estimates, while smaller ks result in more flexible and more wiggly estimates.

Tuning parameter is number of neighbors K.

3.7 Stochastic Gradient Boosting

Boosting is a method of converting weak learners into strong learners. In boosting, each new tree is a fit on a modified version of the original data set. The gradient boosting algorithm (gbm) can be most easily explained by first introducing the AdaBoost Algorithm.The AdaBoost Algorithm begins by training a decision tree in which each observation is assigned an equal weight. After evaluating the first tree, we increase the weights of those observations that are difficult to classify and lower the weights for those that are easy to classify. The second tree is therefore grown on this weighted data. Here, the idea is to improve upon the predictions of the first tree. Our new model is therefore Tree 1 + Tree 2. We then compute the classification error from this new 2-tree ensemble model and grow a third tree to predict the revised residuals. We repeat this process for a specified number of iterations. Subsequent trees help us to classify observations that are not well classified by the previous trees. Predictions of the final ensemble model is therefore the weighted sum of the predictions made by the previous tree models. Gradient Boosting trains many models in a gradual, additive and sequential manner. The major difference between AdaBoost and Gradient Boosting Algorithm is how the two algorithms identify the shortcomings of weak learners (eg. decision trees). While the AdaBoost model identifies the shortcomings by using high weight data points, gradient boosting performs the same by using gradients in the loss function (y=ax+b+e , e needs a special mention as it is the error term). The loss function is a measure indicating how good are model's coefficients are at fitting the underlying data. A logical understanding of loss function would depend on what we are trying to optimize.

tunning parameters, depth, number of trees, shrinkage and n.minobsinnode.

3.8 Ensamble

We take the results of previous methods and if majority of the methods predict "defaulter" ensamble will predict "defaulter" if majority is "not defaulter" it will predict "non defaulter".

4-Results

Before starting we are going to divide dataset in training set, test set and scale the matrix of predictors and to simplify calculations we will use only variables with more correlation with the probability of default next month

```
##Convert as factor output variable
creditcard$def[creditcard$def=="yes"]<-1
creditcard$def[creditcard$def=="no"]<-0
creditcard$def<-as.factor(creditcard$def)

### To Simplify calculations I will take just most relevant variables.

#PAY_1,PAY_2,PAY_3,PAY_4,PAY_5,PAY_6,
#PAY_AMT1,PAY_AMT2,PAY_AMT3,PAY_AMT4,PAY_AMT5,PAY_AMT6 and LIMIT_BAL

filtered<-creditcard%>%
  select(PAY_1,PAY_2,PAY_3,PAY_4,PAY_5,PAY_6,PAY_AMT1,PAY_AMT2,PAY_AMT3,
         PAY_AMT4,PAY_AMT5,PAY_AMT6,LIMIT_BAL,def)

##Scale matrix of predictors

Mean_x<- sweep(as.matrix(filtered[,-14]), 2, colMeans(as.matrix(filtered[,-14])))
x_scaled <- sweep(Mean_x, 2, colSds(as.matrix(filtered[,-14])), FUN = "/")
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = filtered$def, times = 1, p = 0.25, list = FALSE)
##Divide dataset in training and test sets
train_x<-x_scaled[-test_index,]
train_y<-filtered$def[-test_index]
test_x<-x_scaled[test_index,]
test_y<-filtered$def[test_index]
levels(test_y)
```

```
## [1] "0" "1"
```

MODEL 0: Random guessing

```
##To use as reference we are going to predict defaulter randomly
set.seed(1974,sample.kind="Rounding")
random_preds<-as.factor(sample(c(0,1),7500,replace=TRUE))
##Calculate confusion matrix and f1 Score
cm_random<-confusionMatrix(random_preds, test_y)
f1_random<-F_meas(data = random_preds, reference = factor(test_y))
##Save parameters
model_0<-data.frame(Method= "random",
                    Accuracy= cm_random$overall[["Accuracy"]],
                    F1 = f1_random,
                    Sensitivity=cm_random$byClass[[1]],
```

```
                    Specificity=cm_random$byClass[[2]])
model_0%>%kable()
```

Method	Accuracy	F1	Sensitivity	Specificity
random	0.5101333	0.6209245	0.5151515	0.4924653

MODEL 1: Logistic regression

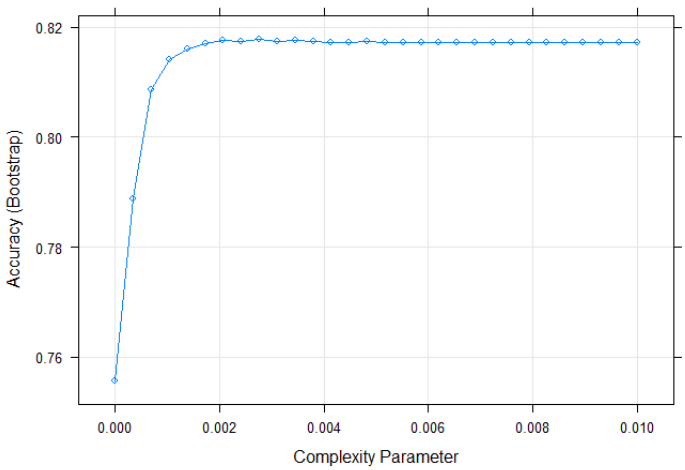
```
##train model
train_glm<-train(train_x,train_y,method = "glm")
##predict outputs
glm_preds<-predict(train_glm, test_x, type = "raw")
##Calculate confusion matrix and f1 Score
cm_glm<-confusionMatrix(glm_preds, test_y)
f1_glm<-F_meas(data = glm_preds, reference = factor(test_y))
##Save parameters
model_1<-data.frame(Method= "glm",
                    Accuracy= cm_glm$overall[["Accuracy"]],
                    F1 = f1_glm,
                    Sensitivity=cm_glm$byClass[[1]],
                    Specificity=cm_glm$byClass[[2]])
RESULTS<-bind_rows(model_0,model_1)

RESULTS%>%kable()
```

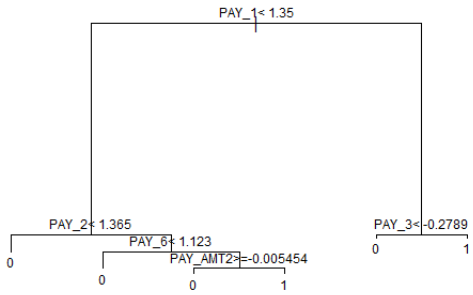
Method	Accuracy	F1	Sensitivity	Specificity
random	0.5101333	0.6209245	0.5151515	0.4924653
glm	0.8128000	0.8907393	0.9797980	0.2248342

MODEL 2: Classification (decision) trees

```
##train model
set.seed(1974, sample.kind="Rounding")
train_rpart <- train(train_x,train_y,method = "rpart",
tuneGrid = data.frame(cp = seq(0.0, 0.01, len = 30)))
##Plot tuning CP to see where is optimum
plot(train_rpart)
```




```
##Plot the tree
plot(train_rpart$finalModel,margin=0.1)
text(train_rpart$finalModel,cex=0.75)
```

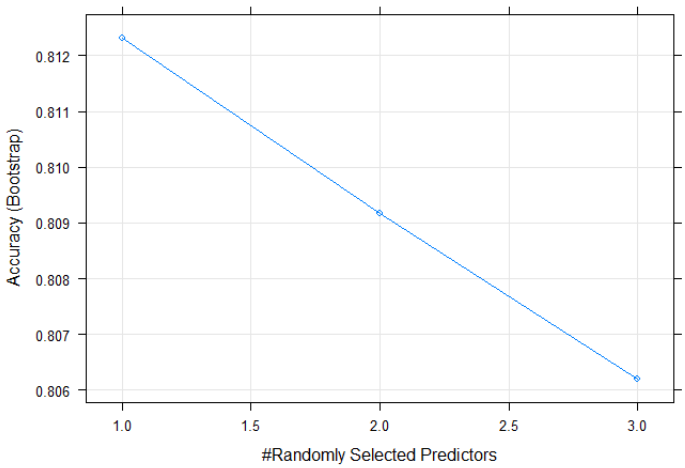



```
##predict output
rpart_preds <- predict(train_rpart, test_x)
##Calculate confusion matrix and f1 Score
cm_rpart<-confusionMatrix(rpart_preds,test_y)
f1_rpart<-F_meas(data = rpart_preds, reference = factor(test_y))
##Save parameters
model_2<-data.frame(Method= "rpart",
                    Accuracy= cm_rpart$overall[["Accuracy"]],
                    F1 = f1_rpart,
                    Sensitivity=cm_rpart$byClass[[1]],
                    Specificity=cm_rpart$byClass[[2]])
RESULTS<-bind_rows(model_0,model_1,model_2)
RESULTS%>%kable()
```

Method	Accuracy	F1	Sensitivity	Specificity
random	0.5101333	0.6209245	0.5151515	0.4924653
glm	0.8128000	0.8907393	0.9797980	0.2248342
rpart	0.8242667	0.8943741	0.9553159	0.3628692

MODEL 3 Random Forest

```
##train model
set.seed(19, sample.kind="Rounding")
train_rf <- train(train_x,train_y,method = "rf",ntree=20,
                  importance=TRUE,tuneGrid = (data.frame(mtry = c(1,2,3))))
##Plot mtry to find optimum
plot(train_rf)
```




```
##Variable importance
print(varImp(train_rf,scale=T))
```

```
## rf variable importance
##
##           Importance
## PAY_1        100.000
## PAY_2         28.534
## PAY_4         16.802
## PAY_5         10.787
## PAY_3         10.369
## PAY_AMT2       8.062
```

```
## PAY_AMT5       7.381
## PAY_AMT3       6.889
## PAY_AMT4       6.581
## PAY_AMT6       6.244
## PAY_AMT1       5.967
## PAY_6          4.169
## LIMIT_BAL      0.000
```

```
#Predict output
rf_preds <- predict(train_rf$finalModel, test_x)
##Calculate confusion matrix and f1 Score
cm_rf<-confusionMatrix(rf_preds,test_y)
f1_rf<-F_meas(data = rf_preds, reference = factor(test_y))
##Save parameters
model_3<-data.frame(Method= "rf",
                    Accuracy= cm_rf$overall[["Accuracy"]],
                    F1 = f1_rf,
                    Sensitivity=cm_rf$byClass[[1]],
                    Specificity=cm_rf$byClass[[2]])
RESULTS<-bind_rows(model_0,model_1,model_2,model_3)
RESULTS%>%kable()
```

Method	Accuracy	F1	Sensitivity	Specificity
random	0.5101333	0.6209245	0.5151515	0.4924653
glm	0.8128000	0.8907393	0.9797980	0.2248342
rpart	0.8242667	0.8943741	0.9553159	0.3628692
rf	0.8192000	0.8919694	0.9583975	0.3291139

In variable importance we can see that default in previous payments (specially in previous month) are key to estimate defaulters.

MODEL 4 LDA

```
##train model
train_lda<-train(train_x,train_y, method = "lda")
#Predict output
lda_preds <- predict(train_lda, test_x)
##Calculate confusion matrix and f1 Score
cm_lda<-confusionMatrix(lda_preds,test_y)
f1_lda<-F_meas(data = lda_preds, reference = factor(test_y))
##Save parameters
model_4<-data.frame(Method= "lda",
                    Accuracy= cm_lda$overall[["Accuracy"]],
                    F1 = f1_lda,
                    Sensitivity=cm_lda$byClass[[1]],
                    Specificity=cm_lda$byClass[[2]])
RESULTS<-bind_rows(model_0,model_1,model_2,model_3,model_4)
RESULTS%>%kable()
```

Method	Accuracy	F1	Sensitivity	Specificity
random	0.5101333	0.6209245	0.5151515	0.4924653
glm	0.8128000	0.8907393	0.9797980	0.2248342
rpart	0.8242667	0.8943741	0.9553159	0.3628692
rf	0.8192000	0.8919694	0.9583975	0.3291139
lda	0.8136000	0.8911215	0.9794556	0.2296564

MODEL 5 GamLoess

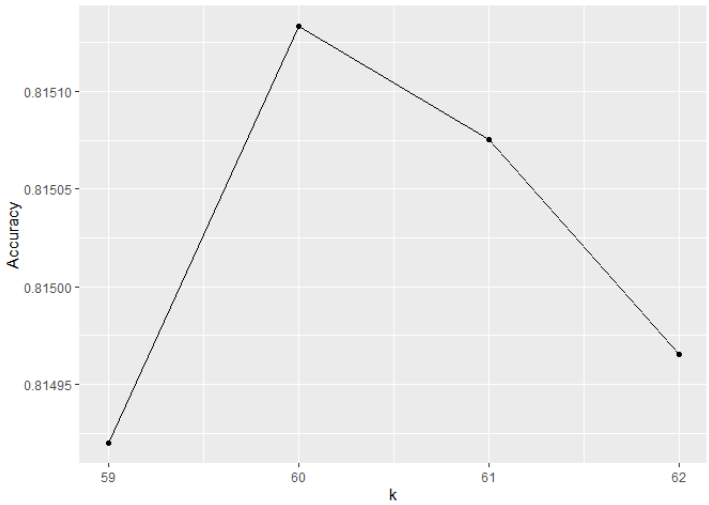
```
##train model
train_gam<-train(train_x,train_y, method = "gamLoess")
#Predict output
gam_preds <- predict(train_gam, test_x, type = "raw")
##Calculate confusion matrix and f1 Score
cm_gam<-confusionMatrix(gam_preds,test_y)
f1_gam<-F_meas(data = gam_preds, reference = factor(test_y))
##Save parameters
model_5<-data.frame(Method= "gamLoess",
                    Accuracy= cm_gam$overall[["Accuracy"]],
                    F1 = f1_gam,
                    Sensitivity=cm_gam$byClass[[1]],
                    Specificity=cm_gam$byClass[[2]])
RESULTS<-bind_rows(model_0,model_1,model_2,model_3,model_4,model_5)
RESULTS%>%kable()
```

Method	Accuracy	F1	Sensitivity	Specificity
random	0.5101333	0.6209245	0.5151515	0.4924653

Method	Accuracy	F1	Sensitivity	Specificity
glm	0.8128000	0.8907393	0.9797980	0.2248342
rpart	0.8242667	0.8943741	0.9553159	0.3628692
rf	0.8192000	0.8919694	0.9583975	0.3291139
lda	0.8136000	0.8911215	0.9794556	0.2296564
gamLoess	0.8193333	0.8918336	0.9563431	0.3369500

MODEL 6 KNN

```
set.seed(85, sample.kind="Rounding")
##train model
train_knn <- train(train_x, train_y,method = "knn",
                   tuneGrid = data.frame(k = seq(59, 62, 1)))
##plot accuracy vs K to see optimum numbers of neighbors
train_knn$results %>%
  ggplot(aes(x = k, y = Accuracy)) +
  geom_line() +
  geom_point()
```




```
#Predict output
knn_preds <- predict(train_knn, test_x)
##Calculate confusion matrix and f1 Score
cm_knn<-confusionMatrix(knn_preds,test_y)
f1_knn<-F_meas(data = knn_preds, reference = factor(test_y))
##Save parameters
model_6<-data.frame(Method= "knn",
                    Accuracy= cm_knn$overall[["Accuracy"]],
                    F1 = f1_knn,
                    Sensitivity=cm_knn$byClass[[1]],
                    Specificity=cm_knn$byClass[[2]])
RESULTS<-bind_rows(model_0,model_1,model_2,model_3,model_4,model_5,model_6)
RESULTS%>%kable()
```

Method	Accuracy	F1	Sensitivity	Specificity
random	0.5101333	0.6209245	0.5151515	0.4924653
glm	0.8128000	0.8907393	0.9797980	0.2248342
rpart	0.8242667	0.8943741	0.9553159	0.3628692
rf	0.8192000	0.8919694	0.9583975	0.3291139
lda	0.8136000	0.8911215	0.9794556	0.2296564
gamLoess	0.8193333	0.8918336	0.9563431	0.3369500
knn	0.8213333	0.8923176	0.9505222	0.3664858

MODEL 7 gbm

```
set.seed(85, sample.kind="Rounding")
##train model
#I tried
#caretGrid <- expand.grid(interaction.depth=c(1, 3, 5),
#                         n.trees = c(100,150),
#                        shrinkage=c(0,001,0,01,0.025, 0.05, 0.1),
#                         n.minobsinnode=10)
##   n.trees interaction.depth shrinkage n.minobsinnode
## 6     200                 5     0.025             10

#i will use best tune i got to reduce calculation time
```

```
caretGrid <- expand.grid(interaction.depth=5,
                         n.trees = 150,
                        shrinkage=0.025,
                        n.minobsinnode=10)



train_gbm <- train(train_x, train_y,method = "gbm",tuneGrid=caretGrid)

#Predict output
gbm_preds <- predict(train_gbm, test_x)
##Calculate confusion matrix and f1 Score
cm_gbm<-confusionMatrix(gbm_preds,test_y)
f1_gbm<-F_meas(data = gbm_preds, reference = factor(test_y))
##Save parameters
model_7<-data.frame(Method= "gbm",
                    Accuracy= cm_gbm$overall[["Accuracy"]],
                    F1 = f1_gbm,
                    Sensitivity=cm_gbm$byClass[[1]],
                    Specificity=cm_gbm$byClass[[2]])
```

```
RESULTS<-bind_rows(model_0,model_1,model_2,model_3,model_4,model_5,model_6,model_7)
RESULTS%>%kable()
```

Method	Accuracy	F1	Sensitivity	Specificity
random	0.5101333	0.6209245	0.5151515	0.4924653
glm	0.8128000	0.8907393	0.9797980	0.2248342
rpart	0.8242667	0.8943741	0.9553159	0.3628692
rf	0.8192000	0.8919694	0.9583975	0.3291139
lda	0.8136000	0.8911215	0.9794556	0.2296564
gamLoess	0.8193333	0.8918336	0.9563431	0.3369500
knn	0.8213333	0.8923176	0.9505222	0.3664858
gbm	0.8249333	0.8944788	0.9527478	0.3749247

MODEL 8 Ensamble

```
##Create ensamble matrix with predictions from 1 to 7
ensamble<-data.frame(glm_preds,rpart_preds,rf_preds,
                     lda_preds,gam_preds,knn_preds,gbm_preds)
##Predict with ensamble
ens_preds<-test_y
for (i in 1:length(ens_preds)){
  ens_preds[i]<-ifelse(sum(ensamble[i,]==1)>=4,1,0)
}
##Calculate confusion matrix and f1 Score
cm_ens<-confusionMatrix(ens_preds,test_y)
f1_ens<-F_meas(data = ens_preds, reference = factor(test_y))
##Save parameters
model_8<-data.frame(Method= "ens",
                    Accuracy= cm_ens$overall[["Accuracy"]],
                    F1 = f1_ens,
                    Sensitivity=cm_ens$byClass[[1]],
                    Specificity=cm_ens$byClass[[2]])
RESULTS<-bind_rows(model_0,model_1,model_2,model_3,
model_4,model_5,model_6,model_7,model_8)
RESULTS%>%kable()
```

Method	Accuracy	F1	Sensitivity	Specificity
random	0.5101333	0.6209245	0.5151515	0.4924653
glm	0.8128000	0.8907393	0.9797980	0.2248342
rpart	0.8242667	0.8943741	0.9553159	0.3628692
rf	0.8192000	0.8919694	0.9583975	0.3291139
lda	0.8136000	0.8911215	0.9794556	0.2296564
gamLoess	0.8193333	0.8918336	0.9563431	0.3369500
knn	0.8213333	0.8923176	0.9505222	0.3664858
gbm	0.8249333	0.8944788	0.9527478	0.3749247
ens	0.8225333	0.8941045	0.9619928	0.3315250

5-Conclusion

Of the 8 models analyzed gbm is the one with higher accuracy and F1 score.

```
RESULTS$Method[which.max(RESULTS$Accuracy)]
```

```
## [1] "gbm"
```

```
max(RESULTS$Accuracy)
```

```
## [1] 0.8249333
```

```
RESULTS$Method[which.max(RESULTS$F1)]
```

```
## [1] "gbm"
```

```
max(RESULTS$F1)
```

```
## [1] 0.8944788
```

```
print(varImp(train_gbm,scale=T))
```

```
## gbm variable importance
##
##             Overall
## PAY_1     100.00000
## PAY_2      12.96249
## PAY_3       7.77422
## LIMIT_BAL   3.83609
## PAY_AMT1    3.78431
## PAY_6       3.50170
## PAY_5       3.36311
## PAY_4       3.15986
## PAY_AMT3    3.09873
## PAY_AMT2    1.35475
## PAY_AMT4    1.10106
## PAY_AMT6    0.09157
## PAY_AMT5    0.00000
```

As we can see most important variables are status of payment of previous 3 periods (jul2005,aug2005 and sep2005)

sensitivity is defined as the ability of an algorithm to predict a positive outcome when the actual outcome is positive. In this case positive outcome is “non defaulter”. So the probability of our model guessing “non defaulter” given that the client is “non defaulter” is high.

On the other side **specificity** is defined as the ability of an algorithm to predict a negative outcome when the actual outcome is negative. In this case negative outcome is “defaulter”. So the probability of our model guessing “Defaulter” when client is actually a defaulter is not very high.

we have a relatively high accuracy because the prevalence of the “non defaulters” is ~ 80%.

If we analyze confusion matrix we can observe:

```
cm_gbm$table
```

```
##          Reference
## Prediction    0    1
##          0 5565 1037
##          1  276  622
```

In our test set we had 7500 observation with 1659 defaulters of which we were able to predict correctly only 622, this is 37,49% of defaulters were correctly identified by our model, it seems too low for the purpose of this project.

If we analyze the random model we have higher specificity, but now sensitivity is considerably lower (we are going to predict a lot of non defaulters as defaulters)

```
cm_random$table
```

```
##          Reference
## Prediction    0    1
##          0 3009  842
##          1 2832  817
```

The idea of this project was to to be able to identify defaulters to take actions.

With this low specificity we are going to get several “FALSE negatives” meaning that we will predict incorrectly defaulters as non defaulters, our model need to be improved getting more predictors that helps us to improve specificity, like income, savings, etc.

There are plenty of algorithms to try for classification, like neural networks, also is possible to add more trees to try to improve prediction in gbm. I had several limitations with my computer because some calculations consumed too much time (like in gamLoess thah i was not able to tune parameters and i had to use default ones)

6- References

Introduction to Data Science - Rafael A. Irizarry

http://archive.ics.uci.edu/ml/machine-learning-databases/00350/default%20of%20credit%20card%20clients.xls

http://topepo.github.io/caret/available-models.html

https://towardsdatascience.com/understanding-gradient-boosting-machines-9be756fe76ab#:~:text=The%20gradient%20boosting%20algorithm%20(gbm,is%20assigned%20an%20equal%20weight.&text=The%20second%20t

http://archive.ics.uci.edu/ml/machine-learning-databases/00350/default%20of%20credit%20card%20clients.xls

https://rafalab.github.io/dsbook/

https://rdrr.io/cran/gam/man/gam.html