Report

Introduction:

In all banks, people get loans all the times. We have young people who need the loan. We have old people who need the loan. We have people getting loan for $1000, $10000, or any other amount of loan. Some people take the loan for terms of 7 days, some take it for 14 days, some for a month and so on. The point is there are many kinds of people with different characteristics taking the loan. There is different type of loans with different loan amounts. People don't always pay loans on time, and this is bad for the bank business. If there is a way, we can predict whether a customer will pay loan on time or not or find out what kind of characteristics of people who don't pay loan on time, it could be very valuable for banks.

Problems:

- Can we build a machine learning model with a good enough accuracy to predict whether a person will pay his/her loan on time?
- Can we use clustering algorithms and/or association rule mining technique to learn what kind of people and what type of loan are likely not to be paid off on time?

Data:

The data set being used for this project comes from Kaggle. Each row of the data is a customer who took a loan. The columns are the characteristics of the customers and the loan information.

The customer characteristics include age, education, and gender. The loan information includes loan id, loan status indication whether the loan is paid off, in collection, new customer yet to paid

off, or paid off after collection, the principal loan amount, terms for loan payment, effective date, due date, paid off time, and past due days.

The columns I used for modeling are not everything from the original data. I only used loan status, principal loan amount, terms, age, education, and gender, because the other columns don't provide much value based on my intuition. For the loan status column I transform it into binary class label, which is called pay_on_time taking on values of 'N' or 'Y'. I also turned all columns to factors. I turned age into groups of 10 ages, because it makes more sense this way. I also turn terms and principal amount columns to factors, because they are discrete, instead of continuous.

Approach:

The classification machine learning algorithms used were Naïve Bayes classification, Random Forest classification, and Kernel SVM with RBF. The clustering technique used was K mean clustering. Last, the association rule mining technique was also used.

For classification models, I tried to first fit the training data into the "train()" function from caret package with the parameter "tuneLength" of about 10. After the initial fitting, I was able to see the approximate range of optimal values for the hyper parameters. I then train the models again using the "tuneGrid" parameter to search for the best parameters combination based on the results from initial training. I then store the model information like testing accuracy to compare models at the end.

Example-

```
##   laplace  usekernel  adjust  Accuracy   Kappa
##   0        FALSE      1       0.5349528  0.08807129
##   0        FALSE      2       0.5349528  0.08807129
##   0        FALSE      3       0.5349528  0.08807129
##   0         TRUE      1       0.5398972  0.04173839
##   0         TRUE      2       0.5398972  0.04173839
##   0         TRUE      3       0.5398972  0.04173839
```

```
...
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were laplace = 0, usekernel = TRUE
##  and adjust = 1.
```

For K means clustering algorithm, I run a for loop to fit several K means clustering models. The number of clusters used and the total sum of square error for each loop were recorded. I then created an elbow plot to find a good clusters number to use.

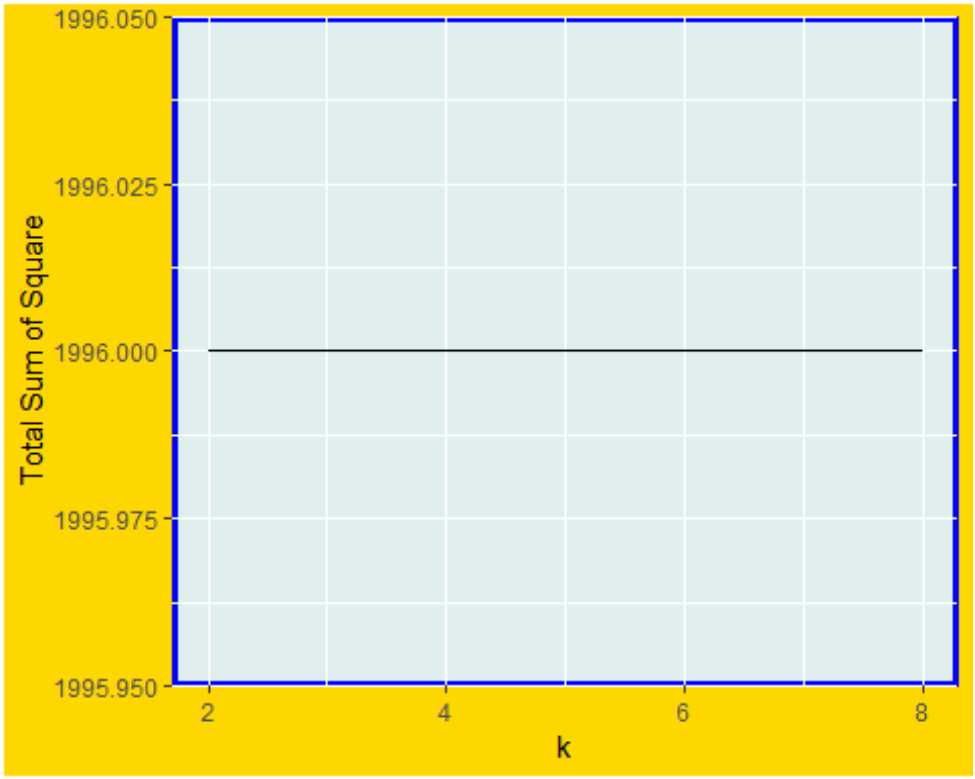
For association rules mining, I look for the rules with support of 0.01 and the confidence of 0.6. I set the right-hand side to pay_on_time = 'N' therefore I can investigate what kind of people and/or loans are not likely to be paid off on time.

Results:

```
##           Model Test.Accuracy
## 1   Naive Bayes     0.5033557
## 2 Random Forest     0.5704698
## 3    Kernel SVM     0.5973154
```

```
##   paid_on_time     n     p
##   <fct>        <int> <dbl>
## 1 N               60 0.403
## 2 Y               89 0.597
```

Looking at the classification models results and the true values of testing data set, it's clear that the models do not provide any values at predicting whether a person will pay loan on time or not.



Looking at K means elbow plot, it seems like no matter how many clusters we choose, the total sum of square would be the same.

```
##     lhs                      rhs              support confidence     lift c
ount
## [1] {terms=30,

##      age=(20,30],

##      education=Bechalor} => {paid_on_time=N}   0.014  0.6363636 1.582994
   7
## [2] {Principal=1000,

##      terms=30,

##      age=(20,30],

##      education=Bechalor} => {paid_on_time=N}   0.014  0.6363636 1.582994
   7
## [3] {terms=30,

##      age=(20,30],

##      education=Bechalor,
```

```
##      Gender=male}        => {paid_on_time=N}   0.014  0.6363636 1.582994
   7
## [4] {Principal=1000,

##      terms=30,

##      age=(20,30],

##      education=Bechalor,

##      Gender=male}        => {paid_on_time=N}   0.014  0.6363636 1.582994
   7
## [5] {terms=15,

##      age=(40,52],

##      education=college}  => {paid_on_time=N}   0.010  0.6250000 1.554726
   5
## [6] {age=(20,30],

##      education=Bechalor,

##      Gender=male}        => {paid_on_time=N}   0.028  0.6086957 1.514168
  14
```

Looking at the association rules, they are not strong based on the confidence. However, they are still worth mentioning for some degrees. We can see that people who has highest education degree of college or bachelor's degree, tend to no pay loan on time. This is probably because they don't have a good degree to find a good paying job. They need loans, but they don't have enough money to pay on time. Young people tend to not pay loan on time as well. The reason for this is probably like the first one. They might not have a good education, not much experience for professional works, so they don't earn good money. Because their budgets are tight all the time, they are not likely to pay loan on time. We also see people who take the loans with longest terms the monthly terms are not likely to pay on time. They are most likely people with low to medium income, and they know they need time to be able to pay off. The monthly term is the longest one to take, so they take it. They don't pay off on time, because they simply do not have money. We also see people who borrows the highest amount of loan tend to not pay

off on time. The reason again is not much different than others. These people do not have a good income, so they need as much loan as they could get. The $1000 loan is the highest they can get, so they took it. However, they have tight budgets, so they probably just do not have money to pay the loan on time.

Possible reasons for poor performance:

The reason for poor performance could be that, there are just not much difference for the characteristics of people, and loan types for whether the loan is paid on time. Another reason could be that, the data attributes available are just not good predictors. It could also be that, more data is needed to capture the true pattern.

Conclusion:

Young people, people with college or bachelor's degrees, people who takes long terms of the loan, and people who takes the highest loan are not likely to pay on time. The banks should pay extra attention when a customer with these characteristics apply for a loan.

```
library(tidyverse)
library(caret)
library(lubridate)
```

Load data

```
df0=read_csv("C:\\Users\\Arnold\\OneDrive\\R_Python_working_directory\\Loan_payments_data.csv")
```

Look at the structure

```
str(df0)
```

```
## Classes 'spec_tbl_df', 'tbl_df', 'tbl' and 'data.frame': 500 obs. of  11 variables:
##  $ Loan_ID       : chr  "xqd20166231" "xqd20168902" "xqd20160003" "xqd20160004" ...
##  $ loan_status   : chr  "PAIDOFF" "PAIDOFF" "PAIDOFF" "PAIDOFF" ...
##  $ Principal     : num  1000 1000 1000 1000 1000 300 1000 1000 1000 800 ...
##  $ terms         : num  30 30 30 15 30 7 30 30 30 15 ...
##  $ effective_date: chr  "9/8/2016" "9/8/2016" "9/8/2016" "9/8/2016" ...
##  $ due_date      : chr  "10/7/2016" "10/7/2016" "10/7/2016" "9/22/2016" ...
##  $ paid_off_time : chr  "9/14/2016 19:31" "10/7/2016 9:00" "9/25/2016 16:58" "9/22/2016 20:00" ...
##  $ past_due_days : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ age           : num  45 50 33 27 28 35 29 36 28 26 ...
##  $ education     : chr  "High School or Below" "Bechalor" "Bechalor" "college" ...
##  $ Gender        : chr  "male" "female" "female" "male" ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   Loan_ID = col_character(),
##   ..   loan_status = col_character(),
##   ..   Principal = col_double(),
##   ..   terms = col_double(),
##   ..   effective_date = col_character(),
##   ..   due_date = col_character(),
##   ..   paid_off_time = col_character(),
##   ..   past_due_days = col_double(),
##   ..   age = col_double(),
##   ..   education = col_character(),
##   ..   Gender = col_character()
##   .. )
```

Drop columns I won't use

```
df=df0 %>% select(-Loan_ID,-effective_date,-loan_status,-past_due_days)
str(df)
```

```
## Classes 'spec_tbl_df', 'tbl_df', 'tbl' and 'data.frame': 500 obs. of  7
variables:
##  $ Principal    : num  1000 1000 1000 1000 1000 300 1000 1000 1000 800 ...
##  $ terms        : num  30 30 30 15 30 7 30 30 30 15 ...
##  $ due_date     : chr  "10/7/2016" "10/7/2016" "10/7/2016" "9/22/2016" ...
##  $ paid_off_time: chr  "9/14/2016 19:31" "10/7/2016 9:00" "9/25/2016
16:58" "9/22/2016 20:00" ...
##  $ age          : num  45 50 33 27 28 35 29 36 28 26 ...
##  $ education    : chr  "High School or Below" "Bechalor" "Bechalor"
"college" ...
##  $ Gender       : chr  "male" "female" "female" "male" ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   Loan_ID = col_character(),
##   ..   loan_status = col_character(),
##   ..   Principal = col_double(),
##   ..   terms = col_double(),
##   ..   effective_date = col_character(),
##   ..   due_date = col_character(),
##   ..   paid_off_time = col_character(),
##   ..   past_due_days = col_double(),
##   ..   age = col_double(),
##   ..   education = col_character(),
##   ..   Gender = col_character()
##   .. )
```

Get data in shape

```
df = df %>%
mutate(Principal=ordered(Principal),due_date=mdy(due_date),terms=ordered(
terms),paid_date=date(mdy_hm(paid_off_time)),education=factor(education,order
ed =
T,levels = c("High School or Below","college","Bechalor","Master or Above")),
Gender=factor(Gender)) %>%
mutate(paid_on_time=factor(ifelse(paid_date>due_date | is.na(
paid_date),'N',"Y"))) %>% select(-due_date,-paid_date,-paid_off_time)
```

Look at summary

```
summary(df)
```

```
##  Principal  terms         age                        education
##  300 :  6   7 : 21   Min.   :18.00   High School or Below:209
##  500 :  3   15:207   1st Qu.:27.00   college              :220
##  700 :  1   30:272   Median :30.00   Bechalor             : 67
```

```
##  800 :111            Mean   :31.12   Master or Above     :  4
##  900 :  2            3rd Qu.:35.00
##  1000:377            Max.   :51.00
##     Gender    paid_on_time
##  female: 77   N:201
##  male  :423   Y:299
##
##
##
##
```

It makes more sense to cut age into 10 years intervals

```
df=df %>% mutate(age=cut(age,breaks = c(1:4,5.2)*10))
```

Get row numbers for training data

```
trnidx=createDataPartition(df$paid_on_time,p = .7,list = F)
trndf=df[trnidx,]
ttdf=df[-trnidx,]
```

Classification model to predict whether a person will pay on time or not.

Naive Bayes model

```
f=formula(paid_on_time~.)
tg=expand.grid(laplace=0:3,usekernel=c(T,F),adjust=1:3)
ctr=trainControl(method = 'cv',number  = 10)
(nb=train(form=f,data=trndf,tuneGrid=tg,method='naive_bayes'))
```

```
## Naive Bayes
##
## 351 samples
##   5 predictor
##   2 classes: 'N', 'Y'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 351, 351, 351, 351, 351, 351, ...
## Resampling results across tuning parameters:
##
##   laplace  usekernel  adjust  Accuracy   Kappa
##   0        FALSE      1       0.5349528  0.08807129
##   0        FALSE      2       0.5349528  0.08807129
##   0        FALSE      3       0.5349528  0.08807129
##   0         TRUE      1       0.5398972  0.04173839
##   0         TRUE      2       0.5398972  0.04173839
```

```
##   0         TRUE      3       0.5398972  0.04173839
##   1        FALSE      1       0.5349528  0.08807129
##   1        FALSE      2       0.5349528  0.08807129
##   1        FALSE      3       0.5349528  0.08807129
##   1         TRUE      1       0.5398972  0.04173839
##   1         TRUE      2       0.5398972  0.04173839
##   1         TRUE      3       0.5398972  0.04173839
##   2        FALSE      1       0.5349528  0.08807129
##   2        FALSE      2       0.5349528  0.08807129
##   2        FALSE      3       0.5349528  0.08807129
##   2         TRUE      1       0.5398972  0.04173839
##   2         TRUE      2       0.5398972  0.04173839
##   2         TRUE      3       0.5398972  0.04173839
##   3        FALSE      1       0.5349528  0.08807129
##   3        FALSE      2       0.5349528  0.08807129
##   3        FALSE      3       0.5349528  0.08807129
##   3         TRUE      1       0.5398972  0.04173839
##   3         TRUE      2       0.5398972  0.04173839
##   3         TRUE      3       0.5398972  0.04173839
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were laplace = 0, usekernel = TRUE
##  and adjust = 1.
```

```
report=data.frame(Model='Naive Bayes','Test Accuracy'=postResample(pred  =
predict(
nb,ttdf),obs = ttdf$paid_on_time)[[1]])
```

Random Forest model

```
(rf=train(form=f,data=trndf,tuneLength=5,method='rf'))
```

```
## Random Forest
##
## 351 samples
##   5 predictor
##   2 classes: 'N', 'Y'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 351, 351, 351, 351, 351, 351, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.5715284  0.03129396
##    5    0.5347346  0.02167812
##    8    0.5362160  0.03670507
##   11    0.5344969  0.03210383
##   14    0.5362947  0.03864316
##
```

```
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.

report=rbind(report,data.frame(Model='Random Forest','Test
Accuracy'=postResample(pred =
predict(rf,ttdf),obs = ttdf$paid_on_time)[[1]]))
```

Kernal SVM model

```
(svm=train(form=f,data=trndf,tuneGrid=expand.grid(C=seq(0.001,2.5,length.out
= 5),
sigma=seq(0,1,length.out = 5)),method='svmRadial'))

## Support Vector Machines with Radial Basis Function Kernel
##
## 351 samples
##   5 predictor
##   2 classes: 'N', 'Y'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 351, 351, 351, 351, 351, 351, ...
## Resampling results across tuning parameters:
##
##   C        sigma  Accuracy   Kappa
##   0.00100  0.00   0.6023388  0.00000000
##   0.00100  0.25   0.6023388  0.00000000
##   0.00100  0.50   0.6023388  0.00000000
##   0.00100  0.75   0.6023388  0.00000000
##   0.00100  1.00   0.6023388  0.00000000
##   0.62575  0.00   0.6023388  0.00000000
##   0.62575  0.25   0.5565033  0.02703921
##   0.62575  0.50   0.5546694  0.02803827
##   0.62575  0.75   0.5541802  0.02869712
##   0.62575  1.00   0.5532779  0.02752703
##   1.25050  0.00   0.6023388  0.00000000
##   1.25050  0.25   0.5494909  0.02247418
##   1.25050  0.50   0.5504347  0.02409250
##   1.25050  0.75   0.5504347  0.02409250
##   1.25050  1.00   0.5504347  0.02409250
##   1.87525  0.00   0.6023388  0.00000000
##   1.87525  0.25   0.5482107  0.02079658
##   1.87525  0.50   0.5504347  0.02409250
##   1.87525  0.75   0.5504347  0.02409250
##   1.87525  1.00   0.5504347  0.02409250
##   2.50000  0.00   0.6023388  0.00000000
##   2.50000  0.25   0.5482107  0.02079658
##   2.50000  0.50   0.5504347  0.02409250
##   2.50000  0.75   0.5504347  0.02409250
##   2.50000  1.00   0.5504347  0.02409250
##
```

```
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 1 and C = 0.001.

report=rbind(report,data.frame(Model='Kernal SVM','Test
Accuracy'=postResample(pred =
predict(svm,ttdf),obs  = ttdf$paid_on_time)[[1]]))
```

Look at the proportion of people pay on time and not on time

```
ttdf %>% group_by(paid_on_time) %>% tally() %>% mutate(p=n/sum(n))

## # A tibble: 2 x 3
##   paid_on_time     n     p
##   <fct>        <int> <dbl>
## 1 N               60 0.403
## 2 Y               89 0.597
```

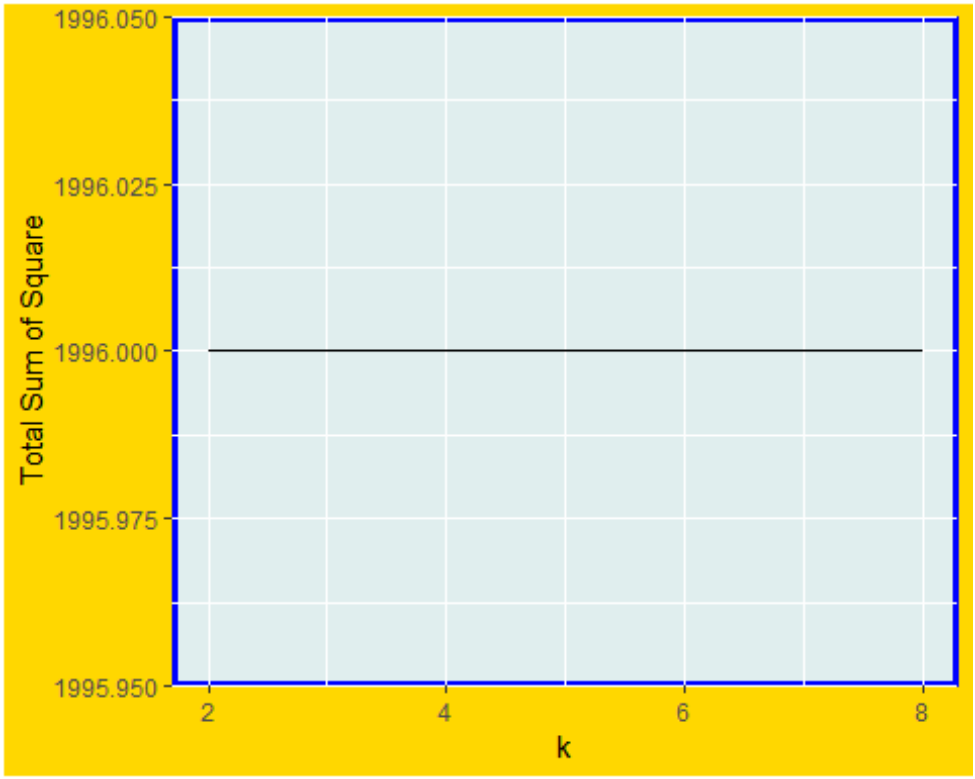
Model comparison

```
report

##           Model Test.Accuracy
## 1   Naive Bayes     0.5033557
## 2 Random Forest     0.5704698
## 3    Kernal SVM     0.5973154
```

K means clustering

```
kmdf=df0 %>% mutate(education=ordered(education,levels=c("High School or
Below",
"college","Bechalor","Master or Above")) %>% as.numeric()) %>% select(
Principal,terms,age,education)
pp = preProcess(kmdf, method = c("center", "scale"))
kmdf=predict(pp,kmdf)
tss=c()
km=c()
for (i in 2:8){
  mod=kmeans(kmdf,i)
  km = c(km,mod)
  tss= c(tss,mod$totss)
}
data.frame(k=2:8,tss) %>% ggplot(aes(k,tss)) + geom_line() + ylab('Total Sum
of Square')
```



Association rules minning

```
library(arules)
rules = apriori(df, parameter = list(supp = 0.01, conf = 0.6),appearance =
list(
  rhs="paid_on_time=N")) %>% sort(by="confidence", decreasing=TRUE)
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.6    0.1    1 none FALSE            TRUE       5    0.01      1
##  maxlen target   ext
##      10  rules FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 5
##
## set item appearances ...[1 item(s)] done [0.00s].
## set transactions ...[21 item(s), 500 transaction(s)] done [0.00s].
## sorting and recoding items ... [17 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5 6 done [0.00s].
```

```
## writing ... [6 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
inspect(rules)
```

```
##     lhs                     rhs              support confidence     lift
count
## [1] {terms=30,
##      age=(20,30],
##      education=Bechalor} => {paid_on_time=N}   0.014  0.6363636 1.582994
7
## [2] {Principal=1000,
##      terms=30,
##      age=(20,30],
##      education=Bechalor} => {paid_on_time=N}   0.014  0.6363636 1.582994
7
## [3] {terms=30,
##      age=(20,30],
##      education=Bechalor,
##      Gender=male}        => {paid_on_time=N}   0.014  0.6363636 1.582994
7
## [4] {Principal=1000,
##      terms=30,
##      age=(20,30],
##      education=Bechalor,
##      Gender=male}        => {paid_on_time=N}   0.014  0.6363636 1.582994
7
## [5] {terms=15,
##      age=(40,52],
##      education=college}  => {paid_on_time=N}   0.010  0.6250000 1.554726
5
## [6] {age=(20,30],
##      education=Bechalor,
##      Gender=male}        => {paid_on_time=N}   0.028  0.6086957 1.514168
14
```