Kaggle Data Report

Yuxuan Peng

Agenda

I followed the standard order to go to this process: load data, clean data, model, prediction.
First, we load the data from Kaggle.

```
data = read.csv('analysisData.csv')
score = read.csv('scoringData.csv')
names(data)
score$price <- paste0(as.numeric(1))
names(score)
score<-score[,c(1:46,91,47:90)]
id<-score$id
names(score)
names(score)==names(data)
#bind
d <- rbind(data,score)
```

```
'data.frame':	9210 obs. of  794 variables:
 $ id                            : int  23136 37597 38982 42288 45375 47609 48740 48850 49205 49786 ...
 $ host_response_rate            : num  0.933 1.036 1.036 -1.004 -1.004 ...
 $ host_acceptance_rate          : num  0.00807 0.00807 0.00807 0.00807 0.00807 ...
 $ accommodates                  : num  0.0208 0.0208 1.0696 -0.5035 -0.5035 ...
 $ bathrooms                     : num  -0.336 3.327 -0.336 -0.336 -0.336 ...
 $ bedrooms                      : num  -0.251 3.767 -0.251 -0.251 -0.251 ...
 $ guests_included               : num  -0.494 0.348 2.032 -0.494 -0.494 ...
 $ extra_people                  : num  -0.6613 0.5741 -0.0436 -0.2495 0.0799 ...
 $ minimum_minimum_nights        : num  -0.2786 -0.2128 -0.2128 -0.147 -0.0812 ...
 $ maximum_minimum_nights        : num  -0.0961 -0.0793 -0.2793 -0.0635 -0.0477 ...
 $ maximum_maximum_nights        : num  -0.00474 -0.00474 -0.00477 -0.00477 -0.00481 ...
 $ availability_30               : num  -0.396 1.429 1.512 1.512 1.512 ...
 $ availability_365              : num  0.299 1.786 1.288 1.713 1.398 ...
 $ number_of_reviews             : num  4.69 0.89 1.19 4.76 3.22 ...
 $ number_of_reviews_ltm         : num  3.0215 0.1318 0.7466 0.0703 0.5622 ...
 $ review_scores_rating          : num  -0.4260 -0.3145 0.1352 -0.7642 0.0227 ...
 $ review_scores_accuracy        : num  -0.682 -0.682 0.465 -1.829 -0.682 ...
 $ review_scores_cleanliness     : num  -0.244 -0.244 0.662 -2.056 -0.244 ...
 $ review_scores_checkin         : num  0.362 0.362 0.362 -2.977 0.362 ...
 $ review_scores_communication   : num  0.345 -0.988 0.345 -0.988 0.345 ...
```

Now, we have 9210 Xs and 794 Ys. And all of the accounts as a numeric or NULL. We need to correct our variables types at first for data cleaning. I put Analysis data and scoring data together for future use since there is a missing variable in scoring data which is price.

Correct feature types to their correct types for analysis

```
# correct feature types  to their correct types for analysis
d
names(d)
d$host_name <- as.factor(d$host_name)
d$host_since <-as.factor(d$host_since)
d$host_location <- as.factor(d$host_location)
d$host_response_time <- as.factor(d$host_response_time)
d$host_response_rate <- as.numeric(d$host_response_rate)
d$host_acceptance_rate <- as.numeric(d$host_acceptance_rate)
d$host_is_superhost <- as.factor(d$host_is_superhost)
d$neighbourhood <-as.factor(d$neighbourhood)
d$host_verifications <- as.factor(d$host_verifications)
d$host_has_profile_pic <- as.factor(d$host_has_profile_pic)
d$host_identity_verified <- as.factor(d$host_identity_verified )
d$street <- as.factor(d$street)
d$neighbourhood <- as.factor(d$neighbourhood)
d$neighbourhood_cleansed <- as.factor(d$neighbourhood_cleansed)
```

```
d$neighbourhood_group_cleansed <- as.factor(d$neighbourhood_group_cleansed)
d$city <- as.factor(d$city)
d$state <- as.factor(d$state)
d$market <- as.factor(d$market)
d$smart_location <- as.factor(d$smart_location)
d$country_code <- as.factor(d$country_code)
d$is_location_exact <- as.factor(d$is_location_exact)
d$property_type <- as.factor(d$property_type)
d$room_type <- as.factor(d$room_type)
d$bathrooms <- as.numeric(d$bathrooms)
d$bed_type <- as.factor(d$bed_type)
d$amenities <- as.factor(d$amenities)
d$calendar_updated <- as.factor(d$calendar_updated)
d$requires_license <- as.factor(d$requires_license)
d$license <- as.factor(d$license)
d$jurisdiction_names <- as.factor(d$jurisdiction_names)
d$instant_bookable <- as.factor(d$instant_bookable)
d$is_business_travel_ready <- as.factor(d$is_business_travel_ready)
d$cancellation_policy <- as.factor(d$cancellation_policy)
d$require_guest_profile_picture <- as.factor(d$require_guest_profile_picture)
d$require_guest_phone_verification <- as.factor(d$require_guest_phone_verification)
d$reviews_per_month <- as.numeric(d$reviews_per_month)

str(d)
```

Now I have corrected features type of variables, I'm ready for data cleaning.

Data Cleaning

1. I identify the variables that have many missing value

```
dim(d[!complete.cases(d),])[[1]]/nrow(d)*100
```

```
> dim(d[!complete.cases(d),])[[1]]/nrow(d)*100
[1] 99.99349
```

I have almost complete data set in Analysis data and Scoring data, but I still need to find them.

I used my professor code in my undergraduate school that helps me find the variables that have missing value.

```
DataQualityReport = function(dataSetName) {

  n = dim(dataSetName)[[1]]          # Number of observations/records
  p = dim(dataSetName)[[2]]          # Number of attributes
  Attributes = names(dataSetName)    # Attribute names
  Type = c(1:p)                      # Attributes data type
  NumberMissing = c(1:p)             # Number of missing values
  PercentComplete = c(1:p)           # Percent of missing values
  Min = c(1:p)                       # Min value for numeric attributes
  Avg = c(1:p)                       # Average value for numeric attributes
  Median = c(1:p)                    # Median value for numeric attributes
  Max = c(1:p)                       # Max value for numeric attributes
  NumberLevels = c(1:p)      # Number of Levels for factor attributes
  x = data.frame(Attributes, Type, NumberMissing, PercentComplete, Min, Avg, Median, Max, NumberLevels) # Dataframe of attribute's statistics
```

```
source('DataQualityReport.R')
DataQualityReport(d)
```

```
# I observe there are some features that have many missing values.
#46                 square_feet numeric         36483          0.97     0      685.87
#48                weekly_price numeric         33285          9.65   100      575.99
#49               monthly_price numeric         36617          0.60   500      851.97
#50            security_deposit numeric         12864         65.08     0      168.2
```

I found out that there are 4 variables has the missing value that is not accepted in this project include: square_feet, weekly_price, monthly_price, and security_deposit. Their percent of completion are 0.97%,9.65%,0.60% and 65.08%.

```
d$square_feet <- NULL
d$weekly_price <- NULL
d$monthly_price <- NULL
d$security_deposit <- NULL
```

I eliminate these variables since there is too much information are missing. I need to eliminate other variables that have too much information to my memory, it will beyond the calculation ability of my computer. I will identify them, not relevant variables, by human decision.

```
d$name <- NULL
d$host_name <- NULL
d$host_since <- NULL
d$host_location <- NULL
d$host_response_time <- NULL
d$host_neighbourhood <- NULL
d$host_verifications <- NULL
d$street <- NULL
d$neighbourhood <- NULL
d$market <- NULL
d$country <- NULL
d$amenities <- NULL
d$calendar_updated <- NULL
d$summary <-NULL
d$space <-NULL
d$description<- NULL
d$neighborhood_overview <- NULL
d$notes <- NULL
d$transit <- NULL
d$access <-NULL
d$interaction <-NULL
d$house_rules <-NULL
d$host_about <-NULL
```

2. Now I have 64 variables that are useful and have accepted the missing value. I'm ready to impute value since there are some variable still have missing value. These variables are host_listings_count, host_total_listings_count, beds, cleaning_fee and reviews_per_month. I put the variables separately by variables that need impute value and variables not need to impute value. The reason why I make theses variables separately is the same as I want to put Analysis data and Scoring data together.

```
id<-d$id
d$id<-NULL
DataQualityReport(d)
str(d)
#doesn't need impute goes d1
d1<-d[,c(1:3,6:20,22:23,25:62)]
#need impute goes d2
d2<-d[,c(“host_listings_count","host_total_listings_count","beds","cleaning_fee","reviews_per_month")]
imputedValues <- mice(data=d2, m=3, method="cart", seed=2019)
d2 <- complete(imputedValues,1)
DataQualityReport(d2)
#把d1,d2合并了
d <- cbind(d1,d2)
```

```
#still missing value,    NULL it
DataQualityReport(d)
library(dplyr)
d$host_total_listings_count <- NULL
d$price<-as.numeric(d$price)
```

	Attributes	Type	NumberMissing	PercentComplete	Min	Avg	Median	Max	NumberLevels
1	host_listings_count	numeric	0	100.00	0.20	5.28	1.00	1483.00	-
2	host_total_listings_count	numeric	3	99.99	0.20	5.28	1.00	1483.00	-
3	beds	numeric	0	100.00	0.20	1.58	1.00	21.00	-
4	cleaning_fee	numeric	0	100.00	0.20	61.44	50.00	602.00	-
5	reviews_per_month	numeric	0	100.00	0.01	1.49	0.96	19.98	-

After imputing value, there is a variable still has missing value and I don’t know why, so I eliminated it. And then we put d1 and d2 together for the next step.

Data clean up

In this step, I found the variable which name ‘price’, I switch ‘price’ to ‘y’ and put it in the first column of data set for easy coding.

```
names(d)
names(d)[20] <- 'y'
names(d)
d<-d[,c(20,1:19,21:62)]
names(d)
```

Creating Dummy Variables

Now I have 56 variables that are useful and have no missing value. The data set d is ready for dummy variables since expecting numeric variables, we have a lot of factor variables.

First, we need to remove the factor variable which has only one level, it can not be dummy variables. Also, we need to remove the factor variable which has too many levels since it doesn’t represent anything.

```
str(d)
d$country_code <- NULL
d$has_availability  <- NULL
d$requires_license <- NULL
d$is_business_travel_ready<- NULL
d$first_review<- NULL
d$last_review <-NULL
DataQualityReport(d)
```

Second, I dummy the variables left. I did the same split the variables which need dummy variables to t2 and the variables are numeric goes t1.

```
t1<-d[,c(2:3,16:18,20:42,49:56)]
#need dummy goes d2
t2<-d[,c(‘y','host_is_superhost','host_has_profile_pic','host_identity_verified','neighbourhood_cleansed',
```

```
'neighbourhood_group_cleansed','city' ,'state','zipcode','smart_location', 'is_location_exact',
'property_type','room_type' ,'bed_type' ,'license' ,'jurisdiction_names'  ,'instant_bookable',
'cancellation_policy', 'require_guest_profile_picture','require_guest_phone_verification')]


dummies <- dummyVars(y~host_is_superhost+host_has_profile_pic+host_identity_verified+neighbourhood_cleansed+
                 neighbourhood_group_cleansed+city +state+zipcode+smart_location+ is_location_exact +
                 property_type+room_type +bed_type +license +jurisdiction_names  +instant_bookable+
                 cancellation_policy+ require_guest_profile_picture+require_guest_phone_verification,
                data = t2)            #create dumyes for Xs


ex <- data.frame(predict(dummies, newdata = t2))  # actually creates the dummies
names(ex) <- gsub("\\.", "", names(ex))         # removes dots from col names
t2 <- cbind(d$y, ex)                          # combine your target variable with Xs
```

After I actually created the dummies, I need to remove dots from column names, and then I combined them to t2. I combined t1 and t2 together after that.

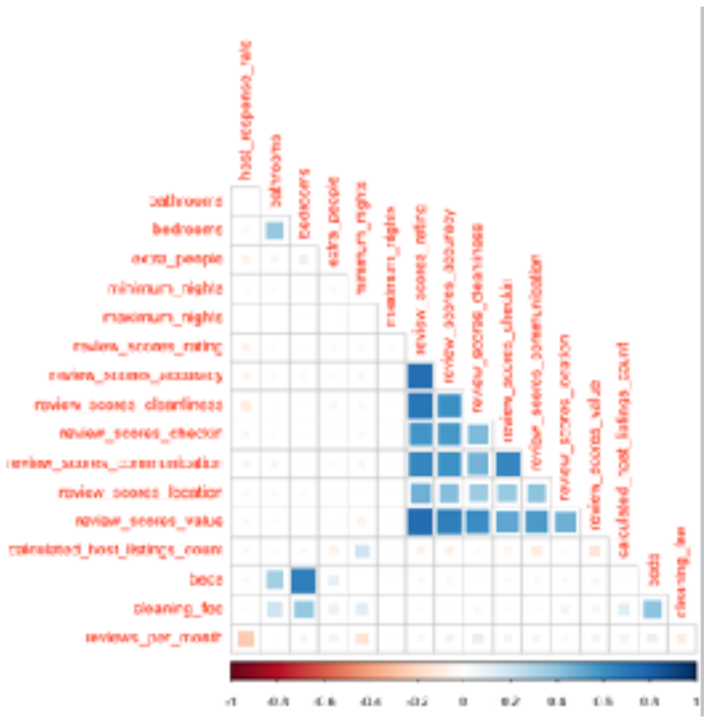
```
d3 <- cbind(t2,t1)
d <- d3

DataQualityReport(d)
names(d)
str(d)
names(d)[1] <- "y"                          # make target variable called 'y'
names(d)
rm(dummies, ex,d3,t1,t2)
```

Remove Zero- and Near Zero-Variance Predictors

In this step, I supposed to remove zero or near-zero variance predictors. After a lot of attempts, I failed to remove these predictors since I don't know what is wrong in the data set.

Identify Correlated Predictors and remove them



I identified these numeric variables correlation before dummy variables since after dummy variables there are too many variables in the plot. It makes sense in this plot. For example, review_scores_rating has a high correlation with review_ scores_accuracy and review_scores value.

Now I have 1181 variables, I want to calculate the correlation matrix using Pearson's correlation formula. I choose the rows(Xs) which correlation is over 85%.

```
descrCor <-  cor(d[,2:ncol(d)])                    # correlation matrix
highCorr <- sum(abs(descrCor[upper.tri(descrCor)]) > .85) # number of Xs having a corr > some value
summary(descrCor[upper.tri(descrCor)])                 # summarize the correlations
```

```
> summary(descrCor[upper.tri(descrCor)])                    # summarize the correlations
      Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
-1.0000000 -0.0005355 -0.0001408  0.0006458 -0.0000376  1.0000000
>
```

I removed those specific columns that have an over 85% correlation between them. Now I summarized those correlations to see if all features are now within my range.

```
highlyCorDescr <- findCorrelation(descrCor, cutoff = 0.85)
filteredDescr <- d[,2:ncol(d)][,-highlyCorDescr] # remove those specific columns from your dataset
descrCor2 <- cor(filteredDescr)                # calculate a new correlation matrix

summary(descrCor2[upper.tri(descrCor2)])
```

```
> summary(descrCor2[upper.tri(descrCor2)])
      Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
-0.7136564 -0.0008469 -0.0002228  0.0001106 -0.0000575  0.8486265
```

And then, I updated my d dataset by removing those filtered variables that were highly correlated.

Identifying linear dependencies and remove

I want to find if any linear combinations exist in my dataset and which combos they are. I added a vector of 1s at the beginning of the dataset and it will help ensure the same features are identified and removed.

```
# first save response
y <- d$y

# create a column of 1s. This will help identify all the right linear combos
```

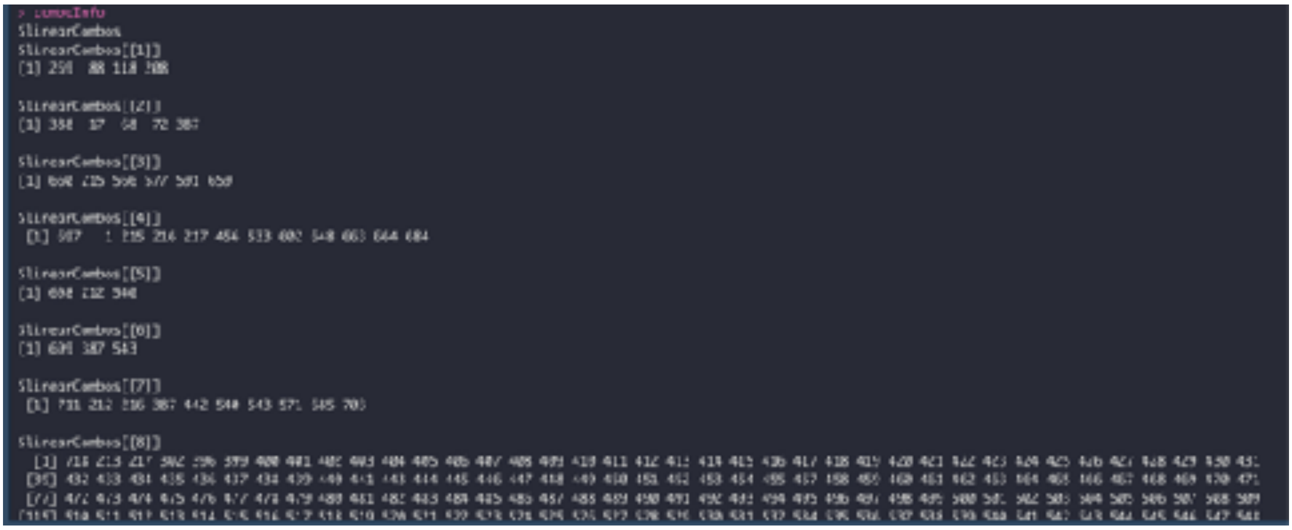
```
d <- cbind(rep(1, nrow(d)), d[2:ncol(d)])
names(d)[1] <- "ones"

# identify the columns that are linear combos
comboInfo <- findLinearCombos(d)
comboInfo

# remove columns identified that led to linear combos
d <- d[, -comboInfo$remove]

# remove the "ones" column in the first column
d <- d[, c(2:ncol(d))]

# Add the target variable back to our data.frame
d <- cbind(y, d)
names(d)
rm(y, comboInfo)  # clean up
```



After I know the column has linear combos, I removed it and removed the 1s column that I added before.

Standardize input features.

Our dataset is pretty clean now and ready for the final step of cleaning data. I standardize the input features using the preProcess() function by performing a typical Z-score standardization. It made all numeric features centered at 0 and have a standard deviation of 1. To make sure I didn't standardize the dummy variables, I created dCats and dNums that represent contains the 0/1 variables and the numeric features.

```
numcols <- apply(X=d, MARGIN=2, function(c) sum(c==0 | c==1)) != nrow(d)
catcols <- apply(X=d, MARGIN=2, function(c) sum(c==0 | c==1)) == nrow(d)
dNums <- d[,numcols]
dCats <- d[,catcols]

str(d)


preProcValues <- preProcess(dNums[,2:ncol(dNums)], method = c("center","scale"))
preProcValues <- preProcess(dNums[,2:ncol(dNums)], method = c("range","YeoJohnson"))

dNums <- predict(preProcValues, dNums)
```

```
# combine the standardized numeric features with the dummy vars
d <- cbind(dNums, dCats)

rm(preProcValues, numcols, catcols, dNums, dCats)  # clean up
```

Split the data

Now I split the data by Analysis data and Scoring data since all variables went through the same process.

```
AnalysisData <- d[1:36839,]
ScoringData<-d[36840:46049,]
ScoringData$Id<-id
ScoringData<-ScoringData[,c(795,2:794)]
str(ScoringData)
```

1.Backward selection

I choose the backward selection for a couple of reasons.

1. Using a lot of features will increase the training time.
2. The more features you have, the more likely you will overfit to the train data, and thus have poor performance on the test set.

```
mf <- lm(y ~ ., data=train)
summary(mf)

# automatic backward selection
library(leaps)
mb <- regsubsets(y ~ ., data=train
              , nbest=1
              , intercept=T
              , method='backward'
              , really.big=T
)
vars2keep <- data.frame(summary(mb)$which[which.max(summary(mb)$adjr2),])
names(vars2keep) <- c("keep")
head(vars2keep)
library(data.table)
vars2keep <- setDT(vars2keep, keep.rownames=T)[]
vars2keep <- c(vars2keep[which(vars2keep$keep==T & vars2keep$rn!="(Intercept)"),"rn"])[[1]]

# here are the final features found to be statistically significant
vars2keep
#[1] "accommodates"             "bathrooms"                "bedrooms"
#[4] "cleaning_fee"             "neighbourhood_cleansedHarlem" "cityNewYork.1"
#[7] "property_typeResort"      "room_typePrivateroom"     "room_typeSharedroom"

modelFormula <- paste("y ~ accommodates + bathrooms + bedrooms + cleaning_fee + neighbourhood_cleansedHarlem +
                  + cityNewYork.1 + property_typeResort + room_typePrivateroom +
                  room_typeSharedroom")
mb <- lm(modelFormula, data=train)
summary(mb)
predback = predict(mb,newdata=test)

rmseback = sqrt(mean((predback-test$y)^2)); rmseback
rm(rmsebcak,rmseCV,vars2keep,modelFormula,submissionFile,mf,larsBetas)
```

There are 9 features, including accommodates, bathrooms, bedrooms, cleaning_fee, neighbourhood_cleansedHarlem, cityNewYork.1, property_typeResort, room_typePrivateroom, and room_typeSharedroom, that I found to be statistically significant. The RMSE for backward is around 75.

```
> summary(mb)

Call:
lm(formula = modelformula, data = train)

Residuals:
    Min      1Q  Median      3Q     Max
-411.62  -35.23   -5.14   23.17  875.96

Coefficients:
                              Estimate Std. Error t value Pr(>|t|)
(Intercept)                   139.2676     0.7545  184.59   <2e-16 ***
accommodates                   23.2651     0.6382   36.45   <2e-16 ***
bathrooms                      14.6679     0.4463   32.86   <2e-16 ***
bedrooms                       17.9782     0.5844   30.77   <2e-16 ***
cleaning_fee                   17.8143     0.4991   35.69   <2e-16 ***
neighbourhood_cleansedHarlem  -43.5712     1.8227  -23.91   <2e-16 ***
cityNewYork.1                  55.7667     0.8791   63.43   <2e-16 ***
property_typeResort           311.4289    11.5287   27.01   <2e-16 ***
room_typePrivateroom          -53.2277     1.8304  -51.66   <2e-16 ***
room_typeSharedroom           -76.1402     2.7456  -27.73   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 71.71 on 31305 degrees of freedom
Multiple R-squared:  0.5525,    Adjusted R-squared:  0.5523
F-statistic:  4294 on 9 and 31305 DF,  p-value: < 2.2e-16

> predback = predict(mb,newdata=test)
> rmseback = sqrt(mean((predback-test$y)^2)); rmseback
[1] 75.08831
> 
```

2. Lasso

I trained a LASSO model using 3-fold cross-validation with k-fold. We also tune it using 15 different values from s=0.05 to s=1.

```
library(caret)
ctrl <- trainControl(method="cv",     # cross-validation set approach to use
                     number=3,        # k number of times to do k-fold
                     classProbs = F,
                     summaryFunction = defaultSummary,
                     allowParallel=T)
# train a LASSO
lassofit <- train(y ~ .,
                  data = train,
                  method = "lars",
                  trControl = ctrl,
                  #preProcess=c("center","scale"), # not needed; already transformed
                  tuneLength = 15,              # this specifies various s values
                  metric = "RMSE")
lassofit

# optimal s
lassofit$bestTune[[1]]


plot(x=lassofit$results$fraction, y=lassofit$results$RMSE
     , col="blue", pch=19
     , main="RMSE vs s from caret runs", xlab="S", ylab="RMSE")

predLasso = predict(RF,newdata=ScoringData)
```

The RMSE of last is around 69.

3.Random Forests

I attempt a lot of time on the Random forest model with cross-validation since it cost 7-10 hours on my computer.

```
library(caret)
ctrl <- trainControl(method="cv",     # cross-validation set approach to use
                number=3,       # k number of times to do k-fold
                classProbs = F,
                summaryFunction = defaultSummary,
                allowParallel=T)

# train a random forest.
rf <- train(y ~ .,
          data = AnalysisData,
          method = "rf",
          importance=T,    # we add this in or it varImp cannot be computed
          trControl = ctrl,
          tuneLength = 10,
          metric = "RMSE")
rf
library(mgcv) # used to save models

varImp(rf)
plot(varImp(rf))
```

I failed to run this model since it cost over 15 hours.

At second time, I removed cross validation from previous model.

```
tuneGrid = expand.grid(mtry=1:5)
set.seed(2019)

RF = train(y~.,data=train,
          method="rf",ntree=1000,trControl=ctrl,tuneGrid=tuneGrid )

predRF = predict(RF,newdata=test)
rmseRF = sqrt(mean((predRF-test$y)^2)); rmseRF
```

I have RMSE at 58.8 when ntree = 600, it is the smallest RMSE I have for my all attempts under the computer performance limitation.

4 - Boosting with cross validation

I failed to run this model since it cost over 15 hours.

```
library(caret)
library(gbm)

set.seed(2019)
tuneGrid=  expand.grid(n.trees = 1000, interaction.depth = c(1,2),
                shrinkage = (1:100)*0.001,n.minobsinnode=5)
garbage = capture.output(cvBoost <- train(y~.,data=train,method="gbm",
                        trControl=ctrl, tuneGrid=tuneGrid))
boostCV = gbm(y~.,data=train,distribution="gaussian",
          n.trees=cvBoost$bestTune$n.trees,
          interaction.depth=cvBoost$bestTune$interaction.depth,
          shrinkage=cvBoost$bestTune$shrinkage,
```

```
        n.minobsinnode = cvBoost$bestTune$n.minobsinnode)
predBoostCV = predict(boostCV,test,n.trees=1000)
rmseBoostCV = sqrt(mean((predBoostCV-test$y)^2)); rmseBoostCV
```

5 - xgboost

Xgboost is a flexible model since I can adjust a lot of different parameters in this model. It is not included in our class, but it is still worth to try.

```
train = AnalysisData
test = ScoringData[-1]

library(xgboost)
X = data.matrix(train[-1])
library(Matrix)
X = Matrix(X, sparse = TRUE)
Y = data.matrix(train[1])
traindata = list(data = X, label = Y)
train_xgb = xgb.DMatrix(data = traindata$data, label = traindata$label)
testdata = data.matrix(test)
test_xgb = xgb.DMatrix(data = testdata)

xgb_cv = xgb.cv(data = train_xgb,
         nrounds = 30,
         nthread = 5,
         nfold = 10,
         metrics = 'rmse',
         subsample = 0.8,
         max_depth = 10,
         eta = 0.2,
         seed = 1)

fit_xgb = xgboost(train_xgb,
          max_depth = 15,
          nrounds = 25,
          subsample = 0.8,
          eta = 0.1,
          seed = 1,
          nthread = 3)


predxgb = predict(RF,newdata=testdata)
rmsexgb = sqrt(mean((predRF-test$y)^2)); rmseRF
```

My best try with RMSE at 56 in xgboost, I have 88 RMSE at the kaggle which worse than the result in the random forest model.
Random forest model with ntree at 600 is the best prediction under the limitation of my computer performance.