```r
#NYC=read.csv("/Users/zhongming/Desktop/DataFest/NYC.csv")
#NYC=data.frame(NYC[-1,])
#names(NYC)
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.5.1

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(tidyr)
# or library(tidyverse)
#NYC=NYC%>% group_by(Sub.Borough.Area) %>% fill(rent_asking_med)
#NYC=NYC%>% group_by(Sub.Borough.Area) %>% fill(Car.free.commute)
##NYC=NYC%>% group_by(Sub.Borough.Area) %>% fill(travel.time.to.wok)
#dd2010=filter(NYC,Year==2010)
##dd2011=filter(NYC,Year==2011)
#dd2012=filter(NYC,Year==2012)
#dd2013=filter(NYC,Year==2013)
#dd2014=filter(NYC,Year==2014)
#dd2015=filter(NYC,Year==2015)
#dd2016=filter(NYC,Year==2016)
#d2010=scale(dd2010[-c(1,2,3)])
#row.names(d2010)=dd2010$Sub.Borough.Area
#d2011=scale(dd2011[-c(1,2,3)])
#row.names(d2011)=dd2011$Sub.Borough.Area
#d2012=scale(dd2012[-c(1,2,3)])
#row.names(d2012)=dd2012$Sub.Borough.Area
#d2013=scale(dd2013[-c(1,2,3)])
#row.names(d2013)=dd2013$Sub.Borough.Area
#d2014=scale(dd2014[-c(1,2,3)])
#row.names(d2014)=dd2014$Sub.Borough.Area
#d2015=scale(dd2015[-c(1,2,3)])
#row.names(d2015)=dd2015$Sub.Borough.Area
#d2016=scale(dd2016[-c(1,2,3)])
#row.names(d2016)=dd2016$Sub.Borough.Area
```

```r
#library(cluster)
#library(factoextra)
#k2010 <- kmeans(d2010, centers = 4, nstart = 25)
#k2011 <- kmeans(d2011, centers = 4, nstart = 25)
#k2012 <- kmeans(d2012, centers = 4, nstart = 25)
#k2013 <- kmeans(d2013, centers = 4, nstart = 25)
#k2014 <- kmeans(d2014, centers = 4, nstart = 25)
#k2015 <- kmeans(d2015, centers = 4, nstart = 25)
#k2016 <- kmeans(d2016, centers = 4, nstart = 25)
#p2010 <- fviz_cluster(k2010, geom = "point", data =d2010 ) + ggtitle("2010")
#p2011 <- fviz_cluster(k2011, geom = "point", data =d2011 ) + ggtitle("2011")
```

```
#p2012 <- fviz_cluster(k2012, geom = "point", data =d2012 ) + ggtitle("2012")
#p2013 <- fviz_cluster(k2013, geom = "point", data =d2013 ) + ggtitle("2013")
#p2014 <- fviz_cluster(k2014, geom = "point", data =d2014 ) + ggtitle("2014")
#p2015 <- fviz_cluster(k2015, geom = "point", data =d2015 ) + ggtitle("2015")
#p2016 <- fviz_cluster(k2016, geom = "point", data =d2016 ) + ggtitle("2016")
#library(gridExtra)
#grid.arrange(p2010,p2011,p2012,p2013,p2014,p2015,p2016, nrow=4)
```

```
#c2010=dd2010[-c(1,2,3)] %>%
 # mutate(Cluster=k2010$cluster) %>%
  #group_by(Cluster) %>%
  #summarise_all("mean")
#c2011=dd2011[-c(1,2,3)] %>%
 # mutate(Cluster=k2011$cluster) %>%
  #group_by(Cluster) %>%
  #summarise_all("mean")
#c2012=dd2012[-c(1,2,3)] %>%
  #mutate(Cluster=k2012$cluster) %>%
  #group_by(Cluster) %>%
  #summarise_all("mean")
#c2013=dd2013[-c(1,2,3)] %>%
  #mutate(Cluster=k2013$cluster) %>%
  #group_by(Cluster) %>%
  #summarise_all("mean")
#c2014=dd2014[-c(1,2,3)] %>%
  #mutate(Cluster=k2014$cluster) %>%
  #group_by(Cluster) %>%
  #summarise_all("mean")
#c2015=dd2015[-c(1,2,3)] %>%
  #mutate(Cluster=k2015$cluster) %>%
  #group_by(Cluster) %>%
  #summarise_all("mean")
#c2016=dd2016[-c(1,2,3)] %>%
  #mutate(Cluster=k2016$cluster) %>%
  #group_by(Cluster) %>%
  #summarise_all("mean")
```

```
#label our data with the clustering label after take a closer look at the summry data of 3 clutsers eve

#for 2010 1=gentrify 2=not gentirfy 3=high income. we use this label for the rest of the year

#l2010=as.data.frame(k2010$cluster)
#l2011=as.data.frame(k2011$cluster)
#l2012=as.data.frame(k2012$cluster)
#l2013=as.data.frame(k2013$cluster)
##l2014=as.data.frame(k2014$cluster)
#l2015=as.data.frame(k2015$cluster)
#l2016=as.data.frame(k2016$cluster)


#gentrified="gentrify"
#potential="potential"
#notgentrified="not gentrify"
#highincome="high income"
```

```
#2010
#l2010[l2010==2]=gentrified
#l2010[l2010==3]=notgentrified
#l2010[l2010==1]=highincome
#l2010[l2010==4]=potential
#names(l2010)="label"
#l2010=data.frame(l2010)
#2011
#l2011[l2011==2]=gentrified
#l2011[l2011==3]=notgentrified
#l2011[l2011==1]=highincome
#l2011[l2011==4]=potential
#names(l2011)="label"
#l2011=data.frame(l2011)
#2012
#l2012[l2012==2]=gentrified
#l2012[l2012==3]=notgentrified
#l2012[l2012==4]=highincome
#l2012[l2012==1]=potential
#names(l2012)="label"
#l2012=data.frame(l2012)
#2013
# l2013[l2013==3]=gentrified
# l2013[l2013==2]=notgentrified
# l2013[l2013==1]=highincome
# l2013[l2013==4]=potential
# names(l2013)="label"
# l2013=data.frame(l2013)
# #2014
# l2014[l2014==1]=gentrified
# l2014[l2014==4]=notgentrified
# l2014[l2014==3]=highincome
# l2014[l2014==2]=potential
# names(l2014)="label"
# l2014=data.frame(l2014)
# #2015
# l2015[l2015==3]=gentrified
# l2015[l2015==1]=notgentrified
# l2015[l2015==4]=highincome
# l2015[l2015==2]=potential
# names(l2015)="label"
# l2015=data.frame(l2015)
# #2016
# l2016[l2016==2]=gentrified
# l2016[l2016==1]=notgentrified
# l2016[l2016==4]=highincome
# l2016[l2016==3]=potential
# names(l2016)="label"
# l2016=data.frame(l2016)
# #stacked those labels
# dd2010=data.frame(dd2010)
# dd2011=data.frame(dd2011)
# dd2012=data.frame(dd2012)
```

```r
# dd2013=data.frame(dd2013)
# dd2014=data.frame(dd2014)
# dd2015=data.frame(dd2015)
# dd2016=data.frame(dd2016)
# d2010=cbind(dd2010,l2010)
# d2011=cbind(dd2011,l2011)
# d2012=cbind(dd2012,l2012)
# d2013=cbind(dd2013,l2013)
# d2014=cbind(dd2014,l2014)
# d2015=cbind(dd2015,l2015)
# d2016=cbind(dd2016,l2016)
# labeldf=rbind(d2010,d2011,d2012,d2013,d2014,d2015,d2016)
labeldf=read.csv("/Users/zhongming/Desktop/labeldf_2010-2015.v1.csv")
labeldf$label=as.factor(labeldf$label)
train=labeldf[-c(1,2,3,4)]
```

```r
#get label dataset and then we start to train our model,leav the visulize to tableau
#xgboost to find out important features for our investor
# Create a training and validation sets
require(xgboost)
```

```
## Loading required package: xgboost

##
## Attaching package: 'xgboost'

## The following object is masked from 'package:dplyr':
##
##     slice
```

```r
require(Matrix)
```

```
## Loading required package: Matrix

##
## Attaching package: 'Matrix'

## The following object is masked from 'package:tidyr':
##
##     expand
```

```r
require(data.table)
```

```
## Loading required package: data.table

##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

```r
trainObs <- sample(nrow(train), .8 * nrow(train), replace = FALSE)
valObs <- sample(nrow(train), .2* nrow(train), replace = FALSE)

train_dat <- train[trainObs,]
val_dat <- train[valObs,]

# Create numeric labels with one-hot encoding
```

```r
train_labs <- as.numeric(train_dat$label) - 1
val_labs <- as.numeric(val_dat$label) - 1

new_train <- model.matrix(~ . + 0, data = train_dat[, -34])
new_val <- model.matrix(~ . + 0, data = train[valObs, -34])

# Prepare matrices
xgb_train <- xgb.DMatrix(data = new_train, label = train_labs,missing=NA)
xgb_val <- xgb.DMatrix(data = new_val, label = val_labs,missing=NA)
```

```r
# Set parameters(default)
params <- list(booster = "gbtree", objective = "multi:softprob", num_class = 4, eval_metric = "mlogloss

# Calculate # of folds for cross-validation
xgbcv <- xgb.cv(params = params, data = xgb_train, nrounds = 100, nfold = 5, showsd = TRUE, stratified =
```

```
## Warning: 'print.every.n' is deprecated.
## Use 'print_every_n' instead.
## See help("Deprecated") and help("xgboost-deprecated").
```

```
## [1]  train-mlogloss:0.927471+0.005111    test-mlogloss:0.971083+0.020232
## [11] train-mlogloss:0.086325+0.002587    test-mlogloss:0.334667+0.098897
## [21] train-mlogloss:0.028125+0.001709    test-mlogloss:0.346275+0.131827
## [31] train-mlogloss:0.018109+0.000914    test-mlogloss:0.361580+0.147571
## [41] train-mlogloss:0.014798+0.000892    test-mlogloss:0.367508+0.156868
## [51] train-mlogloss:0.013106+0.000801    test-mlogloss:0.375408+0.164018
## [61] train-mlogloss:0.011976+0.000753    test-mlogloss:0.379224+0.166699
## [71] train-mlogloss:0.011204+0.000664    test-mlogloss:0.381149+0.168067
## [81] train-mlogloss:0.010641+0.000602    test-mlogloss:0.383162+0.169348
## [91] train-mlogloss:0.010218+0.000566    test-mlogloss:0.383431+0.171921
## [100]    train-mlogloss:0.009874+0.000542    test-mlogloss:0.383627+0.173474
```

```r
# Function to compute classification error
classification_error <- function(conf_mat) {
  conf_mat = as.matrix(conf_mat)

  error = 1 - sum(diag(conf_mat)) / sum(conf_mat)

  return (error)
}

# Mutate xgb output to deliver hard predictions
xgb_train_preds <- data.frame(xgbcv$pred) %>% mutate(max = max.col(., ties.method = "last"), label = tra

# Examine output
head(xgb_train_preds)
```

```
##            X1           X2           X3           X4 max label
## 1 0.008677742 0.9656572938 0.0030785108 0.0225864816   2     2
## 2 0.630852818 0.0053585265 0.0020993860 0.3616892993   1     1
## 3 0.998762846 0.0004014414 0.0006278768 0.0002078750   1     1
## 4 0.999249518 0.0002643213 0.0002655583 0.0002206264   1     1
## 5 0.001860823 0.0019360061 0.0014864443 0.9947167039   4     4
## 6 0.997641444 0.0006038005 0.0013791418 0.0003756297   1     1
```

```r
# Confustion Matrix
xgb_conf_mat <- table(true = train_labs + 1, pred = xgb_train_preds$max)

# Error
cat("XGB Training Classification Error Rate:", classification_error(xgb_conf_mat), "\n")
```

```
## XGB Training Classification Error Rate: 0.08794788
```

```r
# Automated confusion matrix using "caret"
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```r
xgb_conf_mat_2 <- confusionMatrix(factor(xgb_train_preds$label),
                                  factor(xgb_train_preds$max),
                                  mode = "everything")

print(xgb_conf_mat_2)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2   3   4
##          1 160   1   2   2
##          2   0  29   0   0
##          3   6   0  56   5
##          4   5   1   5  35
##
## Overall Statistics
##
##                Accuracy : 0.9121
##                  95% CI : (0.8746, 0.9412)
##     No Information Rate : 0.557
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8595
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4
## Sensitivity            0.9357  0.93548   0.8889   0.8333
## Specificity            0.9632  1.00000   0.9549   0.9585
## Pos Pred Value         0.9697  1.00000   0.8358   0.7609
## Neg Pred Value         0.9225  0.99281   0.9708   0.9732
## Precision              0.9697  1.00000   0.8358   0.7609
## Recall                 0.9357  0.93548   0.8889   0.8333
## F1                     0.9524  0.96667   0.8615   0.7955
## Prevalence             0.5570  0.10098   0.2052   0.1368
## Detection Rate         0.5212  0.09446   0.1824   0.1140
## Detection Prevalence   0.5375  0.09446   0.2182   0.1498
## Balanced Accuracy      0.9495  0.96774   0.9219   0.8959
```

```r
# Create the model
xgb_model <- xgb.train(params = params, data = xgb_train, nrounds = 100)

# Predict for validation set
xgb_val_preds <- predict(xgb_model, newdata = xgb_val)

xgb_val_out <- matrix(xgb_val_preds, nrow = 4, ncol = length(xgb_val_preds) / 4) %>%
               t() %>%
               data.frame() %>%
               mutate(max = max.col(., ties.method = "last"), label = val_labs + 1)

# Confustion Matrix
xgb_val_conf <- table(true = val_labs + 1, pred = xgb_val_out$max)

cat("XGB Validation Classification Error Rate:", classification_error(xgb_val_conf), "\n")
```

```
## XGB Validation Classification Error Rate: 0.02631579
```

```r
# Automated confusion matrix using "caret"
xgb_val_conf2 <- confusionMatrix(factor(xgb_val_out$label),
                                 factor(xgb_val_out$max),
                                 mode = "everything")

print(xgb_val_conf2)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4
##          1 36  0  0  0
##          2  0  9  0  0
##          3  1  0 17  0
##          4  1  0  0 12
##
## Overall Statistics
##
##                Accuracy : 0.9737
##                  95% CI : (0.9082, 0.9968)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9607
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4
## Sensitivity            0.9474   1.0000   1.0000   1.0000
## Specificity            1.0000   1.0000   0.9831   0.9844
## Pos Pred Value         1.0000   1.0000   0.9444   0.9231
## Neg Pred Value         0.9500   1.0000   1.0000   1.0000
## Precision              1.0000   1.0000   0.9444   0.9231
## Recall                 0.9474   1.0000   1.0000   1.0000
## F1                     0.9730   1.0000   0.9714   0.9600
## Prevalence             0.5000   0.1184   0.2237   0.1579
```

```
## Detection Rate         0.4737   0.1184   0.2237   0.1579
## Detection Prevalence   0.4737   0.1184   0.2368   0.1711
## Balanced Accuracy      0.9737   1.0000   0.9915   0.9922
```

```r
#feature importance
importance <- xgb.importance(feature_names = colnames(train_dat[, -34]), model = xgb_model)
head(importance)
```

```
##                    Feature       Gain      Cover  Frequency
## 1:          assessed_value 0.22963947 0.10215164 0.05509642
## 2:    Hh_inc_rent_med_adj 0.16485188 0.09014317 0.03948577
## 3:         bachelor_degree 0.13104264 0.05325311 0.02938476
## 4:       Homeownership.rate 0.10380600 0.09212188 0.03489440
## 5: singlepersonhouseholds 0.07831679 0.06425895 0.02662994
## 6:       gross_rent_2_3beds 0.07339637 0.03181776 0.02662994
```

```r
importanceRaw <- xgb.importance(feature_names = colnames(train_dat[, -34]), model = xgb_model, data =
```

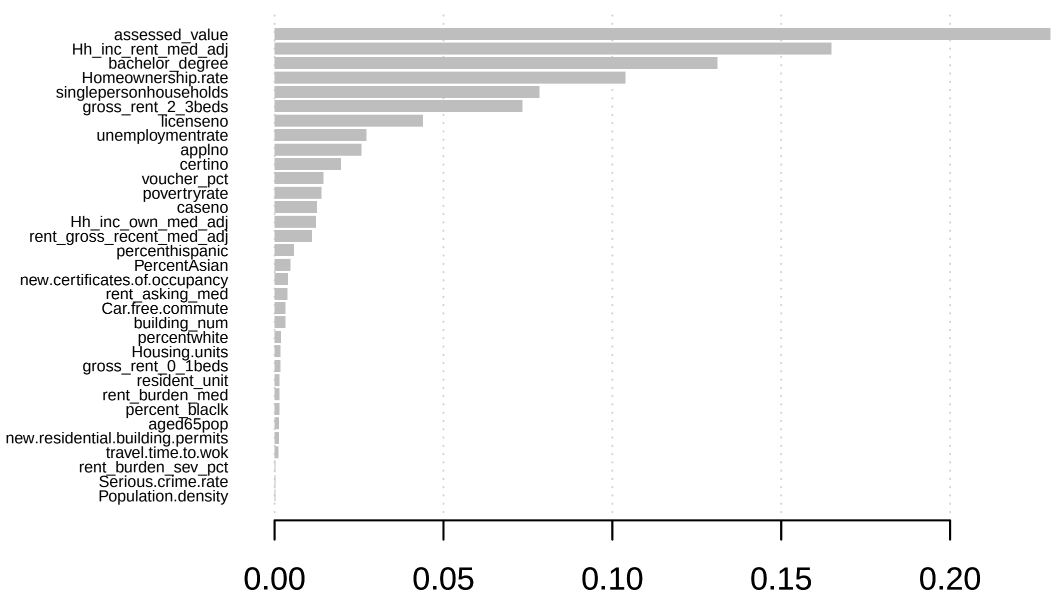
```
## Warning in xgb.importance(feature_names = colnames(train_dat[, -34]), model
## = xgb_model, : xgb.importance: parameters 'data', 'label' and 'target' are
## deprecated
```

```r
# Cleaning for better display
importanceClean <- importanceRaw[,`:=`(Cover=NULL, Frequency=NULL)]

head(importanceClean)
```

```
##                    Feature       Gain
## 1:          assessed_value 0.22963947
## 2:    Hh_inc_rent_med_adj 0.16485188
## 3:         bachelor_degree 0.13104264
## 4:       Homeownership.rate 0.10380600
## 5: singlepersonhouseholds 0.07831679
## 6:       gross_rent_2_3beds 0.07339637
```

```r
xgb.plot.importance(importance_matrix = importance)
```



```r
chisq.test(train_dat$gross_rent_2_3beds,train_labs)
```

```
## Warning in chisq.test(train_dat$gross_rent_2_3beds, train_labs): Chi-
## squared approximation may be incorrect
```

```
##
##  Pearson's Chi-squared test
##
## data:  train_dat$gross_rent_2_3beds and train_labs
## X-squared = 793.53, df = 273, p-value < 2.2e-16
```

```r
chisq.test(train_dat$bachelor_degree,train_labs)
```

```
## Warning in chisq.test(train_dat$bachelor_degree, train_labs): Chi-squared
## approximation may be incorrect
```

```
##
##  Pearson's Chi-squared test
##
## data:  train_dat$bachelor_degree and train_labs
## X-squared = 828.93, df = 294, p-value < 2.2e-16
```

```r
chisq.test(train_dat$rent_gross_recent_med_adj,train_labs)
```

```
## Warning in chisq.test(train_dat$rent_gross_recent_med_adj, train_labs):
## Chi-squared approximation may be incorrect
```

```
##
##  Pearson's Chi-squared test
##
## data:  train_dat$rent_gross_recent_med_adj and train_labs
## X-squared = 791.3, df = 273, p-value < 2.2e-16
```

```r
chisq.test(train_dat$certino,train_labs)
```

```
## Warning in chisq.test(train_dat$certino, train_labs): Chi-squared
## approximation may be incorrect
```

```
##
##  Pearson's Chi-squared test
##
## data:  train_dat$certino and train_labs
## X-squared = 192.15, df = 159, p-value = 0.03753
```

```r
#random forest
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```r
rf=randomForest(label~.,data=train_dat[-c(1,2)],ntree = 100, mtry = 6, importance = TRUE)
rf
```

```
##
## Call:
##  randomForest(formula = label ~ ., data = train_dat[-c(1, 2)],      ntree = 100, mtry = 6, importanc
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 6
##
##         OOB estimate of  error rate: 7.17%
## Confusion matrix:
##              gentrify high income not gentrify potential class.error
## gentrify          164           1            0         0 0.006060606
## high income         0          29            0         0 0.000000000
## not gentrify        6           0           54         7 0.194029851
## potential           5           1            2        38 0.173913043
```

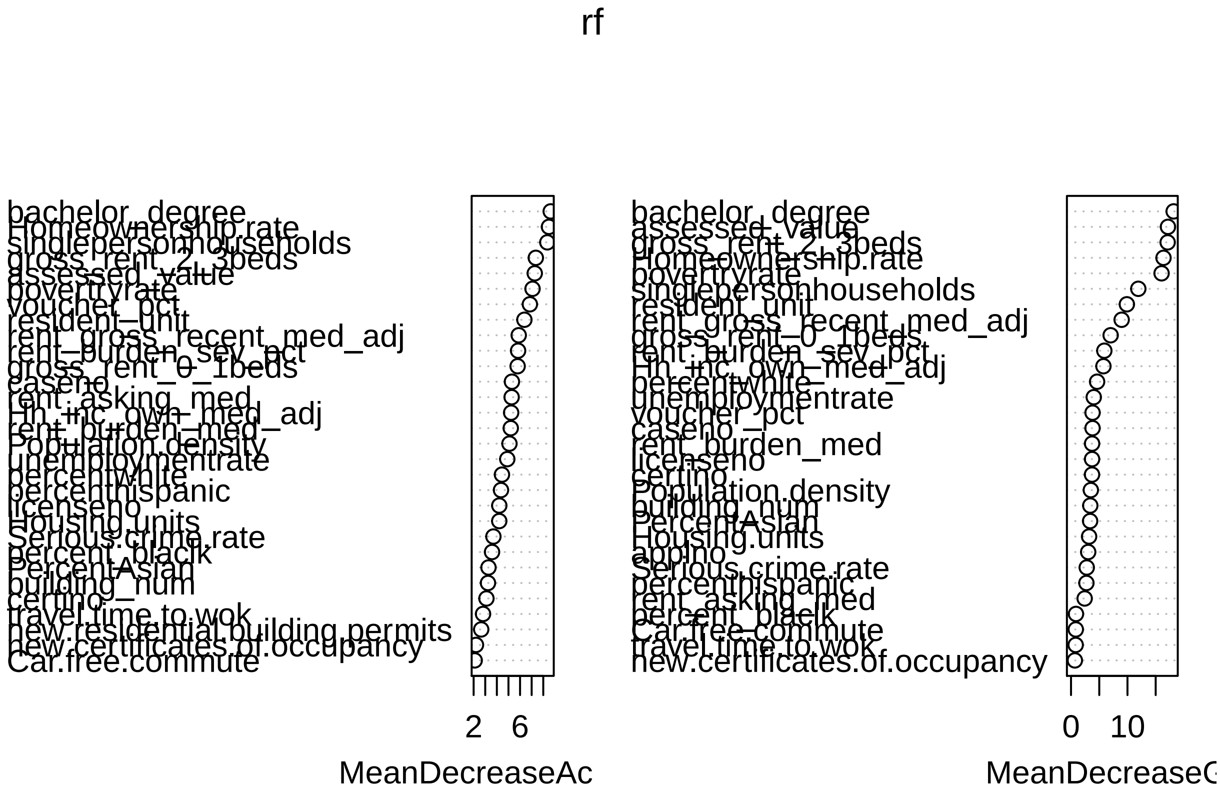
```r
predValid <- predict(rf, val_dat[-c(1,2,36)], type = "class")
predTrain <- predict(rf, train_dat[-c(1,2,36)], type = "class")
mean(predValid == val_dat$label)
```

```
## [1] 0.9868421
```

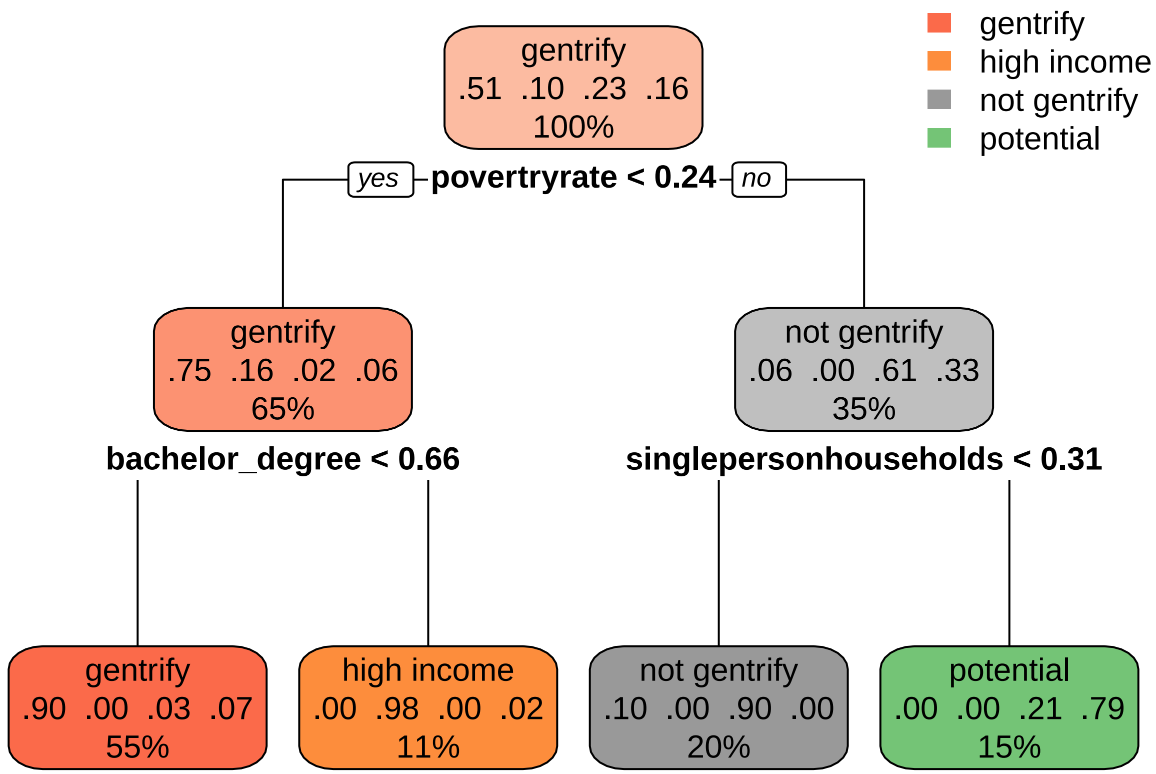
```r
table(predValid,val_dat$label)
```

```
##
## predValid      gentrify high income not gentrify potential
##   gentrify           36           0            0         1
##   high income         0           9            0         0
##   not gentrify        0           0           18         0
##   potential           0           0            0        12
```

```r
varImpPlot(rf)
```




```r
#decision tree
library(rpart)
library(rpart.plot)
tree=rpart(train[-c(1,2)]$label~.,train[-c(1,2)],control=rpart.control(maxdepth = 2))
rpart.plot(tree)
```




```
#amazing! we successfully predict the gentrification situation of NYC perfectly in 2016
#next we visulize the trend and final predication of our result in Tableau.
```