Final Project

Cece Wang

11/25/2021

First, we eliminate certain text variables by hand through excel for the purpose of regression and Machine Learning. We saved the subset of data as *"data.csv"*. Then, we performed data cleaning in the following step (as shown in the code):

1. We transformed all the empty strings to 0 after making sure that each missing value equals a value of 0.
2. We transformed all variables into numeric values
3. We changed all the binary and multinomial varaibles into according numeric values
4. We dropped all of the missing values that we determined to be unreasonable to simply change to the value of 0

```
library(dplyr)
```

```
##
## 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
```

```
library(stringr)
library(tidyr)

data <- read.csv("data.csv")
data[data == ""] <- NA
data[is.na(data)] <- 0
data <- data %>%
  mutate(price = as.numeric(str_replace_all(price, "\\$",""))) %>%
  mutate(security_deposit = as.numeric(str_replace_all(security_deposit, "\\$",""))) %>%
  mutate(extra_people = as.numeric(str_replace_all(extra_people, "\\$",""))) %>%
  mutate(cleaning_fee = as.numeric(str_replace_all(cleaning_fee, "\\$",""))) %>%
  mutate(host_response_rate = as.numeric(str_replace_all(host_response_rate, "\\%",""))) %>%
  mutate(host_acceptance_rate = as.numeric(str_replace_all(host_acceptance_rate, "\\%",""))) %>%
  mutate(host_verifications = lengths(strsplit(data$host_verifications, "\\W+"))) %>%
  mutate(bed_type = ifelse(bed_type == "Real Bed",1,0)) %>%
  mutate(cancellation_policy = case_when(cancellation_policy == "moderate"~2,
                                       cancellation_policy == "strict"~1,
                                       cancellation_policy == "flexible"~3,
                                       cancellation_policy == "super_strict_30"~0)) %>%
  mutate(host_response_rate = as.numeric(str_replace_all(host_response_rate,"N/A",''))) %>%
  mutate(host_acceptance_rate = as.numeric(str_replace_all(host_acceptance_rate,"N/A",''))) %>%
  mutate(room_type = case_when(room_type == "Entire home/apt"~2,
                             room_type == "Private room"~1,
                             room_type == "Shared room"~0)) #%>%
data[data == ''] <- NA
data <- data %>% drop_na()
```

Then, we performed LASSO to do feature selection. The variables with positive coefficients are shown below.

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

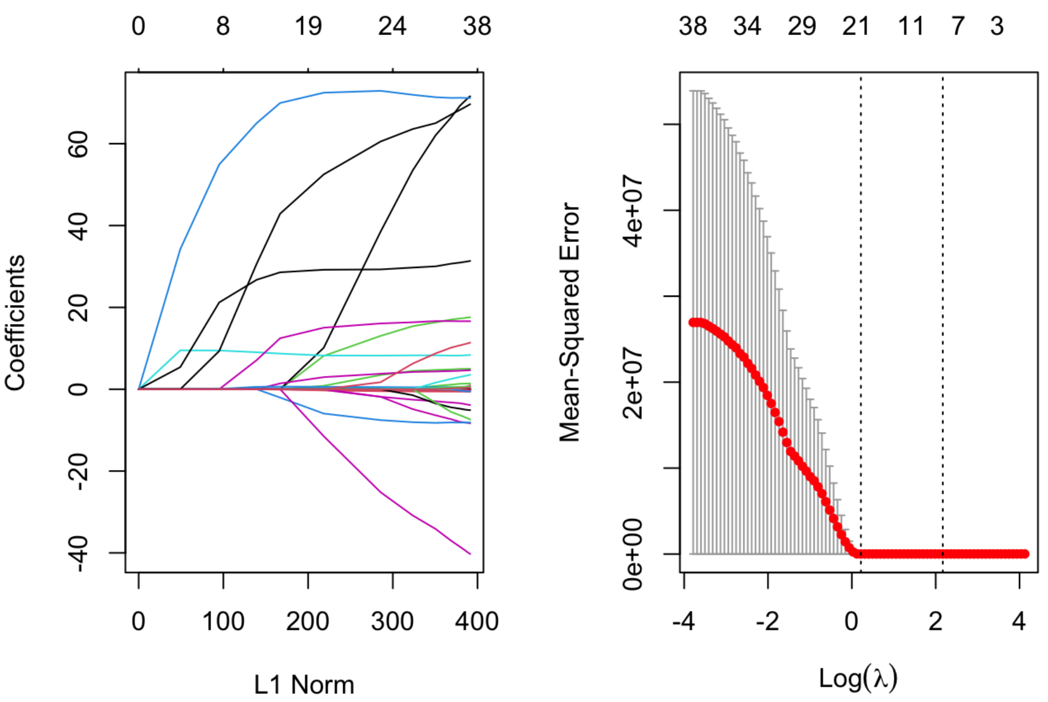
```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
## Loaded glmnet 4.1-3
```

```
require(Matrix)
x <- model.matrix(price ~ ., data = data)
y <- data$price
# names(x)<- c("gender","age","emotional","tangible","affect",
# "psi","psupport","esupport","supsources")
set.seed(2)
lambda_grid <- .5 ^ (-20:20)
lasso.mod = glmnet(x, y, alpha = 1, family = "gaussian", lambda = lambda_grid)
par(mfrow = c(1,2))
plot(lasso.mod)
```

```
## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values
```

```
cv.out <- cv.glmnet(x, y, alpha = 1)
lambda.min <- cv.out$lambda.min
plot(cv.out)
```




```
lasso_coe <- predict(lasso.mod, s = lambda.min, type = "coefficients")
coe <- as.matrix(lasso_coe)[,1]

coe[coe != 0]
```

```
##                       (Intercept)                   host_response_rate
##                       -68.30351990                        -0.08563676
##                  host_is_superhost                 host_verifications
##                       11.85776395                         -1.44097787
##               host_has_profile_pic                  is_location_exact
##                       31.68275561                          2.83454448
##                          room_type                       accommodates
##                       72.81226454                          8.22388484
##                          bathrooms                           bedrooms
##                       15.82584995                         29.26187910
##                   security_deposit                       cleaning_fee
##                        0.00885098                          0.10524630
##                    guests_included                     minimum_nights
##                        3.57848530                         -0.43445385
##                    availability_30                    availability_60
##                        0.54928709                          0.21593127
##                  number_of_reviews               review_scores_checkin
##                       -0.16654860                         -1.39402435
##             review_scores_location                    instant_bookable
##                        1.30160839                         -7.16095704
##    require_guest_profile_picture require_guest_phone_verification
##                      -21.87877966                         58.60108394
```

```
lasso_train <- data[names(coe[coe != 0])[-1]]
lasso_train$y <- y
```

Here, we sort the features by their importance determined by LASSO in order to better choose which ones should be included in our regression models.

```
library(vip)
```

```
##
## Attaching package: 'vip'
```

```
## The following object is masked from 'package:utils':
##
##     vi
```

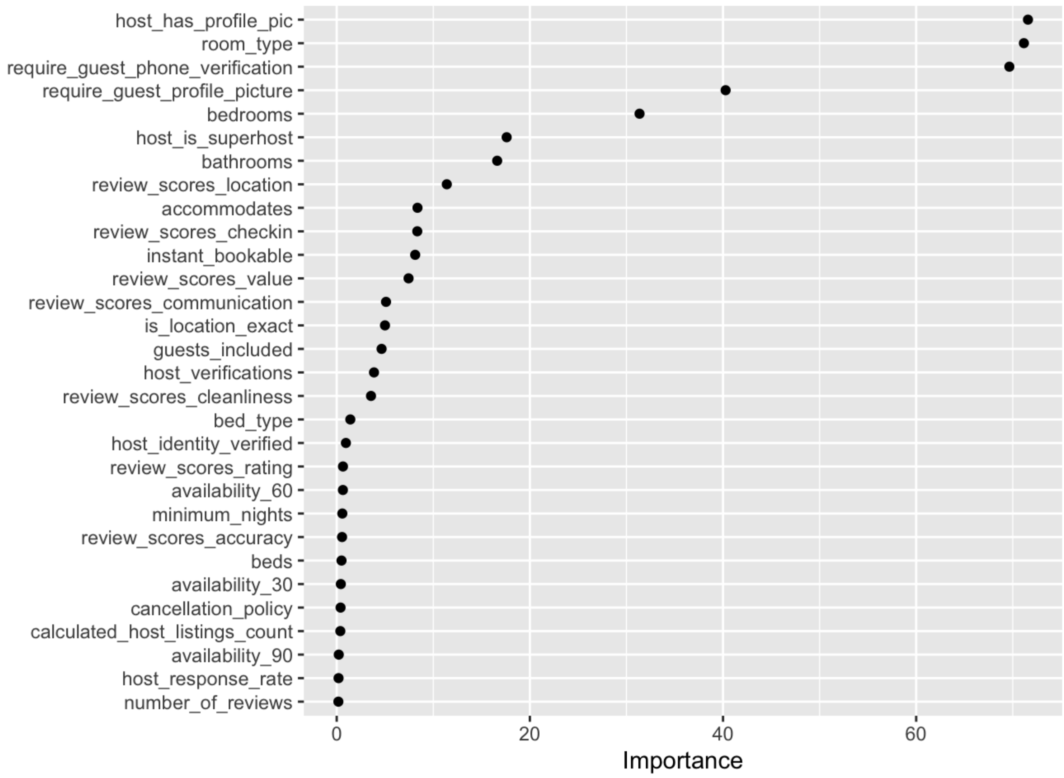
```
library(tidyverse)
```

```
## ── Attaching packages ─────────────────────────────────────── tidyverse 1.3.1 ──
```

```
## ✓ ggplot2 3.3.5     ✓ purrr   0.3.4
## ✓ tibble  3.1.6     ✓ forcats 0.5.1
## ✓ readr   2.0.2
```

```
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## x Matrix::expand() masks tidyr::expand()
## x dplyr::filter()  masks stats::filter()
## x dplyr::lag()     masks stats::lag()
## x Matrix::pack()   masks tidyr::pack()
## x Matrix::unpack() masks tidyr::unpack()
```

```
library(ggpubr)
vip(lasso.mod, num_features=30, geom="point")
```




```
coef.min = coef(cv.out, s="lambda.min")
active.min = which(coef.min !=0)
dimnames(coef.min)[[1]][which(coef.min !=0)]
```

```
##  [1] "(Intercept)"                      "host_response_rate"
##  [3] "host_is_superhost"                "host_verifications"
##  [5] "host_has_profile_pic"             "is_location_exact"
##  [7] "room_type"                        "accommodates"
##  [9] "bathrooms"                        "bedrooms"
## [11] "security_deposit"                 "cleaning_fee"
## [13] "guests_included"                  "minimum_nights"
## [15] "availability_30"                  "availability_60"
## [17] "number_of_reviews"                "review_scores_checkin"
## [19] "instant_bookable"                 "require_guest_profile_picture"
## [21] "require_guest_phone_verification"
```

Regression

First, we put all variables with positive coefficients after the LASSO regularization into the regression model.

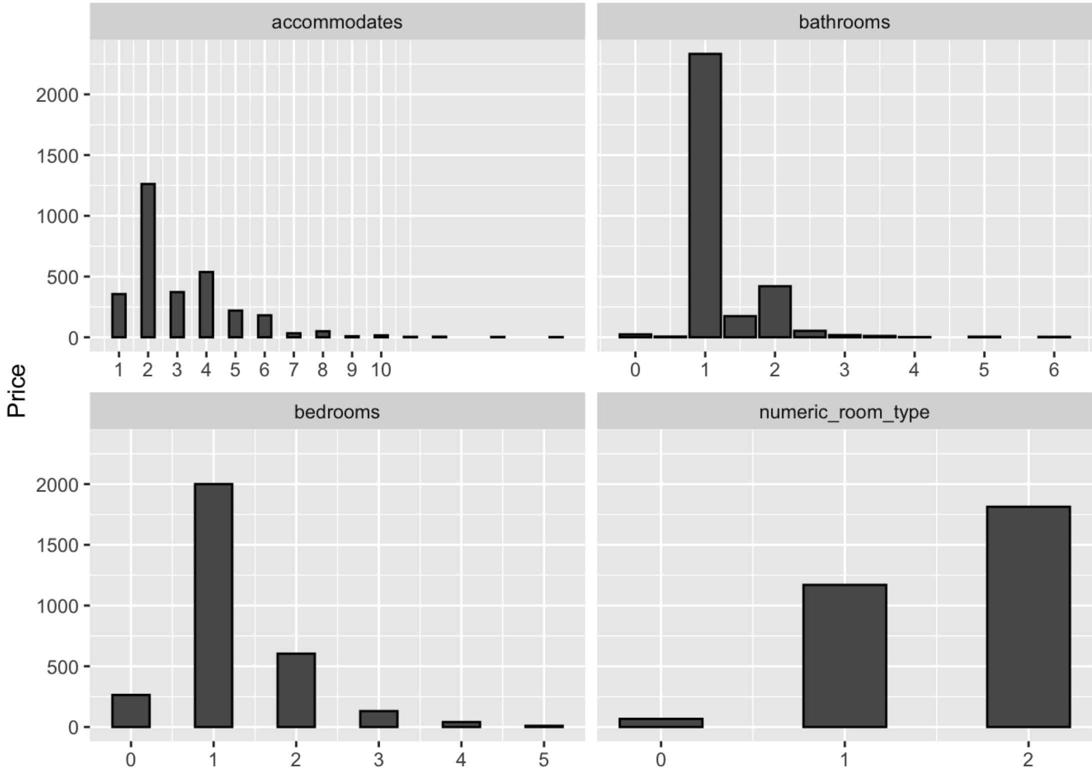
```
selected_variables <- dimnames(coef.min)[[1]][which(coef.min !=0)]
selected_variables <- selected_variables[2:length(selected_variables)]
sv <- paste0(selected_variables,collapse = " + ")
linear1.mod <- lm(paste("price ~", sv), data = data)
summary(linear1.mod)
```

```
##
## Call:
## lm(formula = paste("price ~", sv), data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -234.20  -40.06   -9.32   29.71  861.56
##
## Coefficients:
##                                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)                     -92.135913  39.522235  -2.331 0.019806 *
## host_response_rate               -0.177891   0.109515  -1.624 0.104406
## host_is_superhost                18.044994   4.268256   4.228 2.43e-05 ***
## host_verifications               -3.603485   1.620495  -2.224 0.026243 *
## host_has_profile_pic             66.283215  37.172053   1.783 0.074663 .
## is_location_exact                 6.370527   3.910000   1.629 0.103355
## room_type                        74.475612   3.242862  22.966  < 2e-16 ***
## accommodates                      8.101232   1.352477   5.990 2.35e-09 ***
## bathrooms                        17.322206   2.918269   5.936 3.26e-09 ***
## bedrooms                         28.982463   2.781775  10.419  < 2e-16 ***
## security_deposit                  0.017696   0.008926   1.983 0.047513 *
## cleaning_fee                      0.099739   0.032241   3.094 0.001996 **
## guests_included                   4.642418   1.473304   3.151 0.001643 **
## minimum_nights                   -0.702967   0.211739  -3.320 0.000911 ***
## availability_30                   0.507344   0.377181   1.345 0.178696
## availability_60                   0.268199   0.180878   1.483 0.138243
## number_of_reviews                -0.192434   0.039720  -4.845 1.33e-06 ***
## review_scores_checkin            -0.430607   0.384584  -1.120 0.262944
## instant_bookable                 -9.475878   3.635331  -2.607 0.009189 **
## require_guest_profile_picture   -39.813301  11.002134  -3.619 0.000301 ***
## require_guest_phone_verification 69.401747   6.293269  11.028  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 74.01 on 3029 degrees of freedom
## Multiple R-squared:  0.5256, Adjusted R-squared:  0.5225
## F-statistic: 167.8 on 20 and 3029 DF,  p-value: < 2.2e-16
```

However, it is obvious that this model includes too many variables impractically. Therefore, from these variables we select variables that are both statistically and realistically insignificant based on our own experience when it comes to apartment hunting.

Recalling our own experience of finding AirBnbs, room type, accommodates, number of bathrooms and number of bedrooms are definitely important factors that affect our decisions. Therefore, we plot these four variables out to further investigate below. From the histograms, we can see the most common accommodation is for two people, and only a few houses can accommodate more than 6 people. What's more, most houses only have one bathroom and one or two bedrooms. There are three room types in our data: entire house/apt (labeled as 2), private room (1) and shared room (0). The first two account for the vast majority of the data.

```
data %>%
  mutate(numeric_room_type = as.numeric(room_type)) %>%
  gather(predictor, value, c(numeric_room_type, accommodates, bathrooms, bedrooms)) %>%
  ggplot(aes(x = value)) +
  geom_bar(color = "black") +
  facet_wrap(~ predictor, scales = 'free_x') +
  scale_x_continuous( breaks  = c(0:10)) +
  xlab(NULL) + ylab("Price")
```



We first build a linear regression model with only the four variables mentioned above. The model is

$$\hat{Price_i} = -58.306 + 82.776 RoomType_i + 9.066 Accommodates_i + 21.166 Bathrooms + 33.683 Bedrooms$$

. The result is in line with common sense; rent prices will increase with more accommodates, bedrooms, bathrooms, and better privacy (represented by room type). According to this model, we can simply judge whether the house has exceeded its common price based on these easy-collected variables when choosing rooms.

```
linear2.mod <- lm(price ~ room_type + accommodates + bathrooms + bedrooms , data = data)
summary(linear2.mod)
```

```
##
## Call:
## lm(formula = price ~ room_type + accommodates + bathrooms + bedrooms,
##     data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -249.52  -42.45  -11.23   30.51  910.61
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -58.306      5.504 -10.594  < 2e-16 ***
## room_type      82.776      3.165  26.156  < 2e-16 ***
## accommodates    9.066      1.330   6.818 1.11e-11 ***
## bathrooms      21.166      3.035   6.973 3.79e-12 ***
## bedrooms       33.683      2.886  11.669  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 78.12 on 3045 degrees of freedom
## Multiple R-squared:  0.4687, Adjusted R-squared:  0.468
## F-statistic: 671.5 on 4 and 3045 DF,  p-value: < 2.2e-16
```

```
lm2 <- summary(linear2.mod)

#### MSE
mean(lm2$residuals ^ 2)
```

```
## [1] 6093.219
```

```
library(car)
```

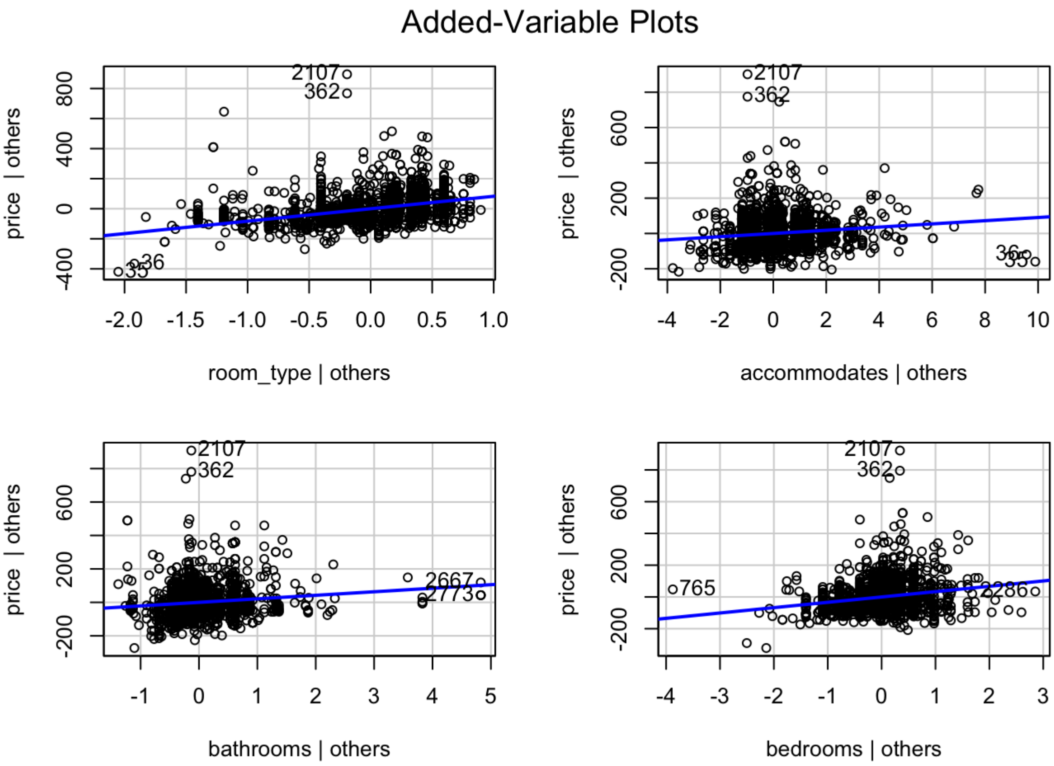
```
## Loading required package: carData
```

```
##
## Attaching package: 'car'
```

```
## The following object is masked from 'package:purrr':
##
##     some
```

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

```
avPlots(linear2.mod)
```



Then, we added the variable with feasibility of immediate booking because it is also a very intuitive and easy-to-obtain parameter in real life – landlords don't want their house to be empty! Thus, it may be helpful to better predict the value of the rent The new model is

$$\hat{Price_i} = -53.844 + 81.535 RoomType_i + 9.420 Accommodates_i + 20.890 Bathrooms + 33.706 Bedrooms - 18.416 InstantBookabl$$

. Comparing the two models we build, we can see that the adjusted R^2 of this new regression model with the instant_bookable term is greater than the first one (0.4721 > 0.468). The MSE is also smaller than before (6044 < 6093). Thus, we think it is appropriate to add the term of whether the house can be booked immediately into linear regression model.

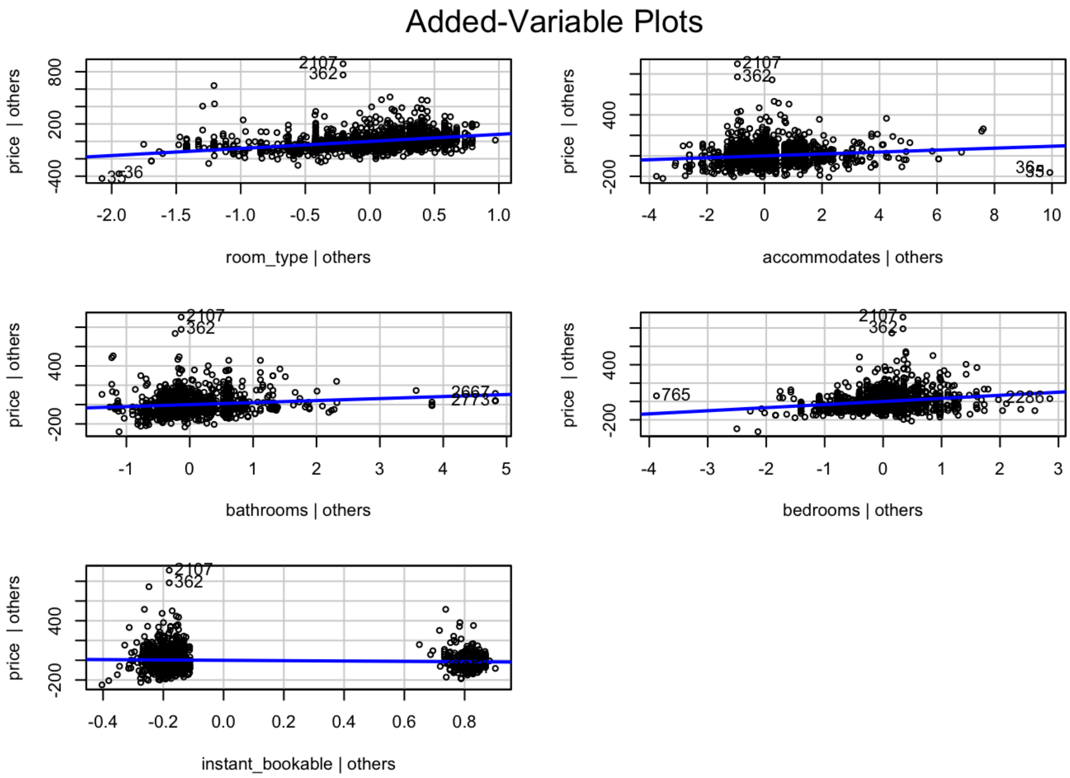
```
linear3.mod <- lm(price ~ room_type + accommodates + bathrooms + bedrooms + instant_bookable, data = data)
summary(linear3.mod)
```

```
##
## Call:
## lm(formula = price ~ room_type + accommodates + bathrooms + bedrooms +
##     instant_bookable, data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -256.96  -42.07  -11.13   31.88  907.29
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)       -53.844      5.555  -9.694  < 2e-16 ***
## room_type          81.535      3.162  25.784  < 2e-16 ***
## accommodates        9.420      1.327   7.101 1.53e-12 ***
## bathrooms          20.890      3.024   6.908 5.97e-12 ***
## bedrooms           33.706      2.875  11.723  < 2e-16 ***
## instant_bookable  -18.416      3.693  -4.987 6.47e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 77.82 on 3044 degrees of freedom
## Multiple R-squared:  0.473,  Adjusted R-squared:  0.4721
## F-statistic: 546.4 on 5 and 3044 DF,  p-value: < 2.2e-16
```

```
lm3 <- summary(linear3.mod)
mean(lm3$residuals ^ 2)
```

```
## [1] 6043.836
```

```
avPlots(linear3.mod)
```



Machine Learning

```
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
```

```
library(caret)
```

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

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(tree)
```

```
## Registered S3 method overwritten by 'tree':
##   method     from
##   print.tree cli
```

```
library(MASS)
```

```
##
## Attaching package: 'MASS'
```

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

```
library(GGally)
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```
library(dslabs)
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

We build two Machine Learning models. In the Random Forest model, `room_type`, `bedrooms`, `accommodates`, `cleaning_fee`, `bathrooms` and `beds` are top 6 variables for rent price prediction ability.

```r
####data_rf <- data %>% dplyr::select(price, room_type , accommodates , bathrooms , bedrooms , require_guest_profile_picture ,
 require_guest_phone_verification) %>% mutate(room_type = factor(room_type))

data_rf <- data

set.seed(1)
price_index_train = createDataPartition(y = data_rf$price,               ### train:test = 1:1
                                  times = 1, p = 0.5, list = FALSE)
price_train_set = slice(data_rf, price_index_train)
price_test_set = slice(data_rf, -price_index_train)

set.seed(64)
fit_rf = randomForest(price ~ ., data = price_train_set)
fit_rf
```

```
##
## Call:
##  randomForest(formula = price ~ ., data = price_train_set)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 13
##
##           Mean of squared residuals: 4018.984
##                     % Var explained: 62.36
```

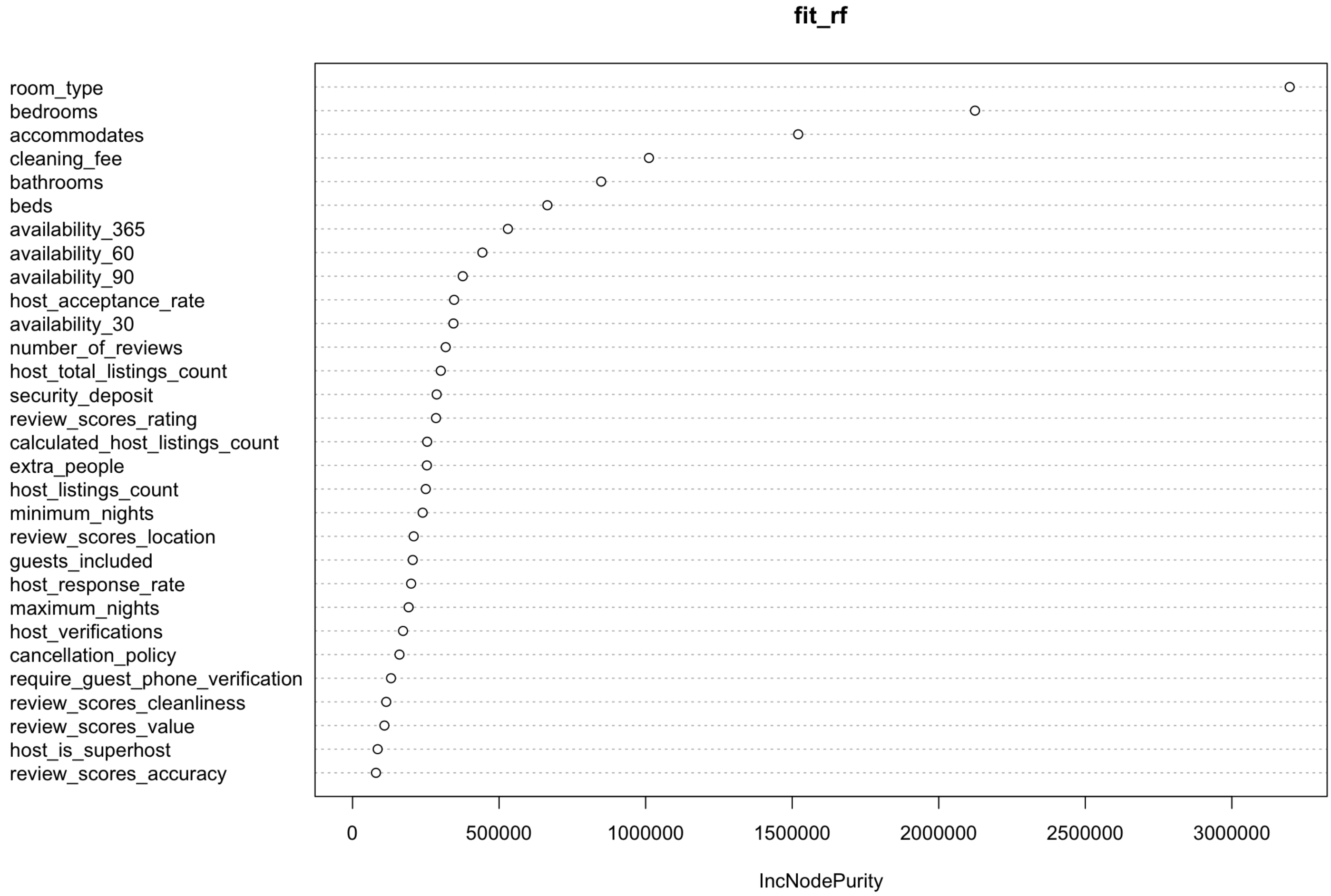
```r
preds_fit_rf = predict(fit_rf, newdata = price_test_set)

library(knitr)
variable_importance <- importance(fit_rf)
tmp <- data_frame(feature = rownames(variable_importance),
                  Gini = variable_importance[,1]) %>%
                  arrange(desc(Gini))
```

```
## Warning: `data_frame()` was deprecated in tibble 1.1.0.
## Please use `tibble()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.
```

```r
kable(tmp[1:6,])
```

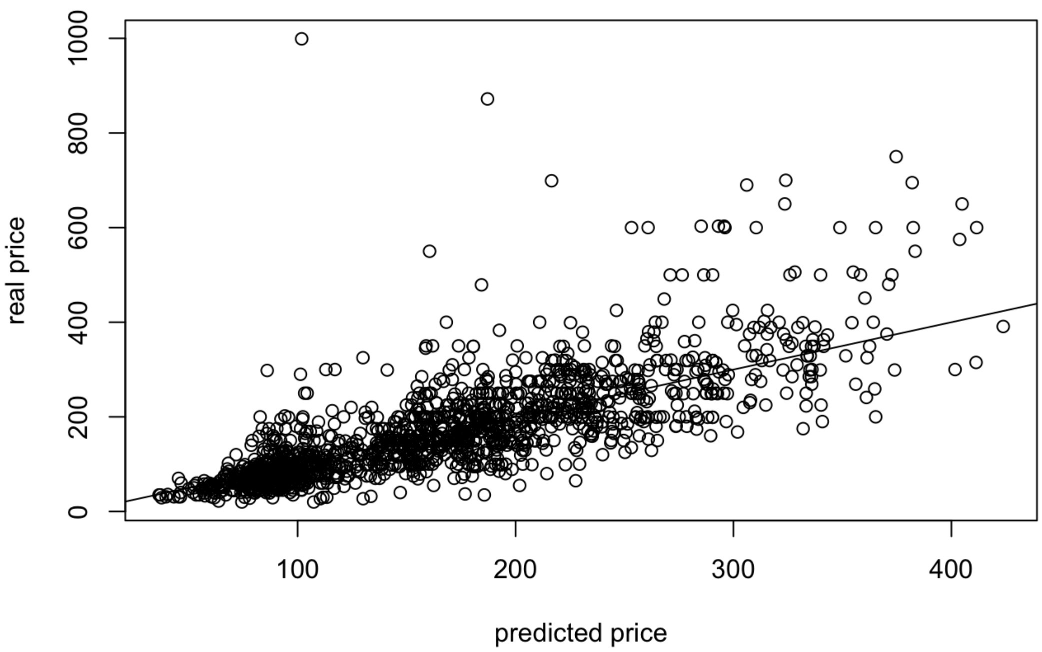
feature	Gini
room_type	3197321.2
bedrooms	2123707.4
accommodates	1520471.9
cleaning_fee	1011333.1
bathrooms	848396.6
beds	664715.7

```r
varImpPlot(fit_rf)
```



We can see that the Random Forest Model has a much smaller MSE than linear regression model.

```
plot(preds_fit_rf, price_test_set$price, xlab = "predicted price", ylab="real price")
abline(0,1)
```




```
mean((preds_fit_rf - price_test_set$price) ^2 )  ###MSE
```

```
## [1] 5047.702
```