COMP 4442 Final Project

Ben Karabinus

11/8/2021

knitr::opts\_chunk$set(echo = TRUE)  
# load necessary libraries  
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(tidyverse)

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

## ✓ ggplot2 3.3.5 ✓ purrr 0.3.4  
## ✓ tibble 3.1.4 ✓ stringr 1.4.0  
## ✓ tidyr 1.1.3 ✓ forcats 0.5.1  
## ✓ readr 2.0.1

## ── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
## x dplyr::filter() masks stats::filter()  
## x dplyr::lag() masks stats::lag()

library(stringr)  
library(ggplot2)  
library(ggeasy)  
if(!require(GGally)){ install.packages("GGally"); library(GGally)}

## Loading required package: GGally

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

if(!require(skimr)){ install.packages("skimr"); library(skimr)}

## Loading required package: skimr

if(!require(xgboost)){ install.packages("xgboost"); library(xgboost)}

## Loading required package: xgboost

##   
## Attaching package: 'xgboost'

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

if(!require(caret)){ install.packages("caret"); library(caret)}

## Loading required package: caret

## Loading required package: lattice

##   
## Attaching package: 'caret'

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

if(!require(recipes)){ install.packages("recipes"); library(recipes)}

## Loading required package: recipes

##   
## Attaching package: 'recipes'

## The following object is masked from 'package:stringr':  
##   
## fixed

## The following object is masked from 'package:stats':  
##   
## step

if(!require(rsample)){ install.packages("rsample"); library(rsample)}

## Loading required package: rsample

if(!require(parsnip)){ install.packages("parsnip"); library(parsnip)}

## Loading required package: parsnip

if(!require(dials)){ install.packages("dials"); library(dials)}

## Loading required package: dials

## Loading required package: scales

##   
## Attaching package: 'scales'

## The following object is masked from 'package:purrr':  
##   
## discard

## The following object is masked from 'package:readr':  
##   
## col\_factor

if(!require(workflows)){ install.packages("workflows"); library(workflows)}

## Loading required package: workflows

if(!require(yardstick)){ install.packages("yardstick"); library(yardstick)}

## Loading required package: yardstick

## For binary classification, the first factor level is assumed to be the event.  
## Use the argument `event\_level = "second"` to alter this as needed.

##   
## Attaching package: 'yardstick'

## The following objects are masked from 'package:caret':  
##   
## precision, recall, sensitivity, specificity

## The following object is masked from 'package:readr':  
##   
## spec

if(!require(tune)){ install.packages("tune"); library(tune)}

## Loading required package: tune

## Registered S3 method overwritten by 'tune':  
## method from   
## required\_pkgs.model\_spec parsnip

if(!require(vip)){ install.packages("vip"); library(vip)}

## Loading required package: vip

##   
## Attaching package: 'vip'

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

if(!require(doParallel)){ install.packages("doParallel");library(doParallel)}

## Loading required package: doParallel

## Loading required package: foreach

##   
## Attaching package: 'foreach'

## The following objects are masked from 'package:purrr':  
##   
## accumulate, when

## Loading required package: iterators

## Loading required package: parallel

all\_cores <- parallel::detectCores(logical = FALSE)  
registerDoParallel(cores = all\_cores)  
options(scipen = 9999)

# load data   
condos <- read.csv("Metro\_zhvi\_condos and co-ops.csv", header=TRUE, sep=",")  
fiveBed <- read.csv("Metro\_zhvi\_Five\_Plus\_Bedroom.csv", header=TRUE, sep=",")  
fourBed <- read.csv("Metro\_zhvi\_Four\_Bedroom.csv", header=TRUE, sep=",")  
threeBed <- read.csv("Metro\_zhvi\_Three\_Bedroom.csv", header=TRUE, sep=",")  
twoBed <- read.csv("Metro\_zhvi\_Two\_Bedroom.csv", header=TRUE, sep=",")  
oneBed <- read.csv("Metro\_zhvi\_One\_Bedroom.csv", header=TRUE, sep=",")  
popIncome <- read.csv("PopulationandPerCapitaIncomeMSA2019.csv", header=TRUE, sep=",")  
Rpp <- read.csv("RegionalPriceParitiesMSA2019.csv", header=TRUE, sep=",")  
unEmp <- read.csv("UnemploymentMSA2019.csv", header=TRUE, sep=",")

# ZHVI data structure represents each month in a year as a column  
# keep only 2019  
condos <- subset(condos, select = -c(1:2, 4, 18:38))  
fiveBed <- subset(fiveBed, select = -c(1:2, 4, 18:38))  
fourBed <- subset(fourBed, select = -c(1:2, 4, 18:38))  
threeBed <- subset(threeBed, select = -c(1:2, 4, 18:38))  
twoBed <- subset(twoBed, select = -c(1:2, 4, 18:38))  
oneBed <- subset(oneBed, select = -c(1:2, 4, 18:38))

# create aggregate of home value for year 2019 (median)  
condos$value <- apply(condos[,3:14], 1, median)  
fiveBed$value <- apply(fiveBed[,3:14], 1, median)  
fourBed$value <- fourBed$value <- apply(fourBed[,3:14], 1, median)  
threeBed$value <- threeBed$value <- apply(threeBed[,3:14], 1, median)  
twoBed$value <- twoBed$value <- apply(twoBed[,3:14], 1, median)  
oneBed$value <- oneBed$value <- apply(oneBed[,3:14], 1, median)

# drop unecessary columns  
condos <- subset(condos, select = -c(3:14))  
fiveBed <- subset(fiveBed, select = -c(3:14))  
fourBed <- subset(fourBed, select = -c(3:14))  
threeBed <- subset(threeBed, select = -c(3:14))  
twoBed <- subset(twoBed, select = -c(3:14))  
oneBed <- subset(oneBed, select = -c(3:14))  
# create categorical variable for home size  
condos[, "Size"] <- "condo"  
fiveBed[, "Size"] <- "five"  
fourBed[, "Size"] <- "four"  
threeBed[, "Size"] <- "three"  
twoBed[, "Size"] <- "two"  
oneBed[, 'Size'] <- "one"  
# combine ZHVI data  
homeValue <- rbind(condos,oneBed, twoBed, threeBed, fourBed, fiveBed)  
# drop aggregate (United States)  
homeValue <- subset(homeValue, RegionName != "United States")  
# rename value to Value  
colnames(homeValue)[3] <- "Value"

# clean unemployment data  
# remove extra strings in MSA column  
unEmp <- unEmp %>% filter(!grepl("Metropolitan NECTA", MSA))  
unEmp$MSA <- str\_remove\_all(unEmp$MSA,   
 "Metropolitan Statistical Area")  
# remove unwanted characters from column MSA  
unEmp$State <- sub(".\*,", "",unEmp$MSA)  
unEmp$MSA <- sub("-.\*", "",unEmp$MSA)  
unEmp$State <- sub("-.\*", "", unEmp$State)  
unEmp$MSA <- sub(",.\*","",unEmp$MSA)  
unEmp$MSA <- str\_trim(unEmp$MSA, side = 'both')  
unEmp$State <- str\_trim(unEmp$State, side = 'both')  
# create clean column of MSA's rename to Region  
unEmp$Region <- str\_c(unEmp$MSA, ", ", unEmp$State)  
# drop unneeded columns  
unEmp <- subset(unEmp, select = -c(1, 3))  
# rename column for unemployment rate  
colnames(unEmp)[1] <- "UnEmpRate"

# Drop column GeoFips  
Rpp <- subset(Rpp, select = -c(1))  
# rename column X2019 and GeoName  
colnames(Rpp)[4] <- c("RPP")  
colnames(Rpp)[1] <- c("MSA")  
# keep aggregate of regional price parities only  
Rpp <- subset(Rpp, LineCode == 1)   
# remove rows containing United States  
Rpp <- Rpp %>% filter(!grepl("United States", MSA))  
# clean the MSA column  
Rpp$MSA <- str\_remove\_all(Rpp$MSA, "\\s\*\\([^\\)]+\\)")  
Rpp$MSA <- str\_remove\_all(Rpp$MSA, "2/")  
Rpp$MSA <- str\_trim(Rpp$MSA, side = 'both')  
# remove unecessary columns  
Rpp <- subset(Rpp, select = -c(2, 3))

# drop NA's  
popIncome <- popIncome %>% drop\_na()  
# drop unnecessary columns  
popIncome <- subset(popIncome, select = -c(1))  
# drop rows containing "United States" (aggregates)  
popIncome <- popIncome %>% filter(!grepl("United States", GeoName))  
# rename columns   
colnames(popIncome)[1] <- c("MSA")  
# split population and per capita income into seperate columns  
Pop <- popIncome[popIncome$LineCode == 2, ]  
Income <- popIncome[popIncome$LineCode == 3, ]  
# rename columns  
colnames(Pop)[4] <- c("POP")  
colnames(Income)[4] <- c("Income")  
# drop unecessary columns  
Pop <- subset(Pop, select = -c(2, 3))  
Income <- subset(Income, select = -c(2,3))  
# create columns for population and income  
dat.temp <- inner\_join(Pop, Income, by = "MSA")  
# clean the MSA column   
dat.temp$MSA = str\_remove\_all(dat.temp$MSA, "\\\*")  
dat.temp$MSA <- str\_remove\_all(dat.temp$MSA, "\\s\*\\([^\\)]+\\)")  
dat.temp$MSA <- str\_trim(dat.temp$MSA, side = 'both')  
# join RPP and popIncome data  
combined <- inner\_join(dat.temp, Rpp, by = "MSA")

# clean MSA column combined dataframe and rename Region  
combined$State <- sub(".\*,", "",combined$MSA)  
combined$MSA <- sub("-.\*", "",combined$MSA)  
combined$State <- sub("-.\*", "", combined$State)  
combined$MSA <- sub("/.\*", "",combined$MSA)  
combined$MSA <- sub(",.\*","",combined$MSA)  
combined$MSA <- str\_trim(combined$MSA, side = 'both')  
combined$State <- str\_trim(combined$State, side = 'both')  
combined$Region <- str\_c(combined$MSA, ", ", combined$State)  
# clean homeValue RegionName and rename Region  
homeValue$RegionName <- sub(",.\*", "",homeValue$RegionName)  
homeValue$RegionName <- sub("-.\*", "", homeValue$RegionName)  
homeValue$RegionName <- str\_trim(homeValue$RegionName, side = 'both')  
homeValue$StateName <- str\_trim(homeValue$StateName, side = 'both')  
homeValue$Region <- str\_c(homeValue$RegionName, ", ", homeValue$StateName)  
  
# join homeValue and combined rename dat.XG  
dat.XG <- left\_join(homeValue, combined, by = "Region")  
dat.XG <- dat.XG %>% drop\_na()  
# join unEmp and dat.XG  
dat.XG <- left\_join(unEmp, dat.XG, by = "Region")  
# drop unnecessary rows and rearrange columns   
dat.XG <- subset(dat.XG, select = -c(3, 4, 7, 11))  
dat.XG <- dat.XG[c("Region", "POP", "Size", "Income",   
 "RPP","UnEmpRate", "Value")]  
# transform variable types  
dat.XG$Region <- as.factor(dat.XG$Region)  
dat.XG$Size <- as.factor(dat.XG$Size)  
# check for NA's in each column  
colSums((is.na(dat.XG)))

## Region POP Size Income RPP UnEmpRate Value   
## 0 11 11 11 11 0 11

# examine NA's  
which(is.na(dat.XG), arr.ind = T)

## row col  
## 726 726 2  
## 888 888 2  
## 1101 1101 2  
## 1144 1144 2  
## 1157 1157 2  
## 1428 1428 2  
## 1441 1441 2  
## 1682 1682 2  
## 1718 1718 2  
## 2001 2001 2  
## 2008 2008 2  
## 726 726 3  
## 888 888 3  
## 1101 1101 3  
## 1144 1144 3  
## 1157 1157 3  
## 1428 1428 3  
## 1441 1441 3  
## 1682 1682 3  
## 1718 1718 3  
## 2001 2001 3  
## 2008 2008 3  
## 726 726 4  
## 888 888 4  
## 1101 1101 4  
## 1144 1144 4  
## 1157 1157 4  
## 1428 1428 4  
## 1441 1441 4  
## 1682 1682 4  
## 1718 1718 4  
## 2001 2001 4  
## 2008 2008 4  
## 726 726 5  
## 888 888 5  
## 1101 1101 5  
## 1144 1144 5  
## 1157 1157 5  
## 1428 1428 5  
## 1441 1441 5  
## 1682 1682 5  
## 1718 1718 5  
## 2001 2001 5  
## 2008 2008 5  
## 726 726 7  
## 888 888 7  
## 1101 1101 7  
## 1144 1144 7  
## 1157 1157 7  
## 1428 1428 7  
## 1441 1441 7  
## 1682 1682 7  
## 1718 1718 7  
## 2001 2001 7  
## 2008 2008 7

#drop NA's  
dat.XG <- dat.XG %>% drop\_na()

##############Exploratory data analysis###########

# quick view of summary statistics for data set  
skim(dat.XG)

Data summary

|  |  |
| --- | --- |
| Name | dat.XG |
| Number of rows | 2135 |
| Number of columns | 7 |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  |
| Column type frequency: |  |
| factor | 2 |
| numeric | 5 |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  |
| Group variables | None |

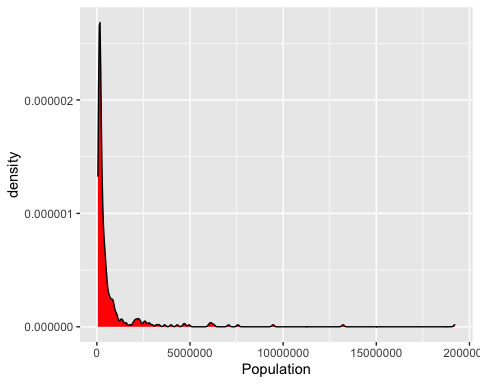
**Variable type: factor**

| skim\_variable | n\_missing | complete\_rate | ordered | n\_unique | top\_counts |
| --- | --- | --- | --- | --- | --- |
| Region | 0 | 1 | FALSE | 357 | Abi: 6, Akr: 6, Alb: 6, Alb: 6 |
| Size | 0 | 1 | FALSE | 6 | fiv: 357, one: 357, fou: 356, thr: 356 |

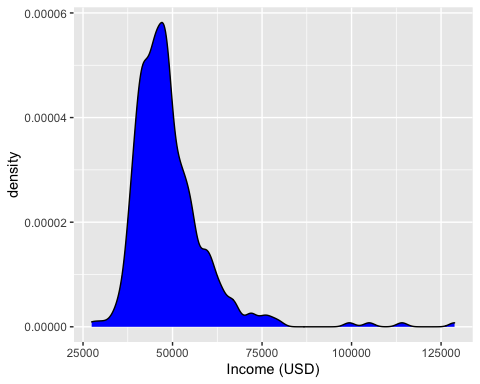
**Variable type: numeric**

| skim\_variable | n\_missing | complete\_rate | mean | sd | p0 | p25 | p50 | p75 | p100 | hist |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| POP | 0 | 1 | 739773.60 | 1677914.09 | 55916.0 | 144088.5 | 233002.0 | 550525.5 | 19216182.0 | ▇▁▁▁▁ |
| Income | 0 | 1 | 49072.59 | 10639.51 | 27415.0 | 42687.0 | 47418.0 | 52828.5 | 128766.0 | ▇▆▁▁▁ |
| RPP | 0 | 1 | 93.05 | 8.20 | 77.8 | 87.8 | 91.2 | 96.4 | 134.5 | ▆▇▂▁▁ |
| UnEmpRate | 0 | 1 | 3.90 | 1.59 | 1.8 | 3.0 | 3.6 | 4.3 | 20.9 | ▇▁▁▁▁ |
| Value | 0 | 1 | 228186.49 | 161634.00 | 20795.5 | 126895.2 | 187187.5 | 282629.5 | 1876608.5 | ▇▁▁▁▁ |

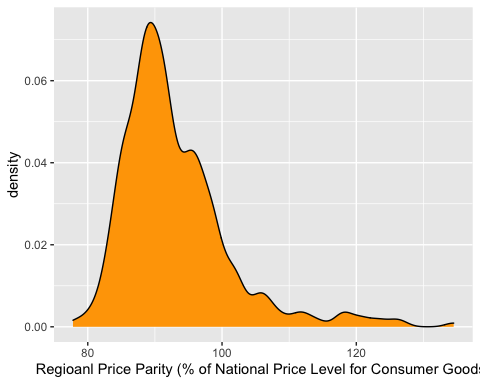
# plot the distribution on continous variables  
ggplot(data = dat.XG, aes(x=POP))+geom\_density(fill = 'red')+  
 xlab("Population")



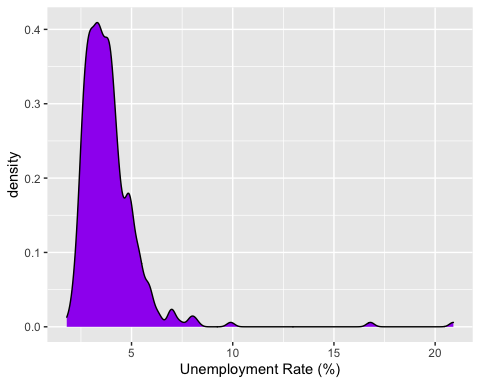
ggplot(data = dat.XG, aes(x=Income))+geom\_density(fill = 'blue')+  
 xlab('Income (USD)')



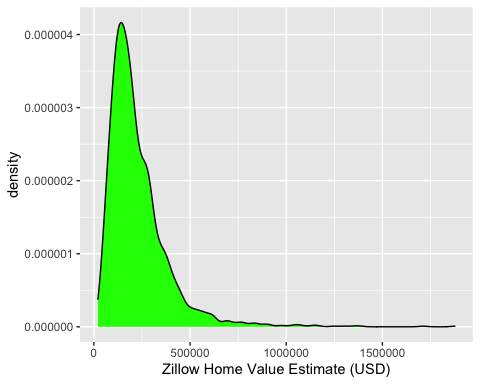
ggplot(data = dat.XG, aes(x=RPP))+geom\_density(fill = 'orange')+  
 xlab('Regioanl Price Parity (% of National Price Level for Consumer Goods)')



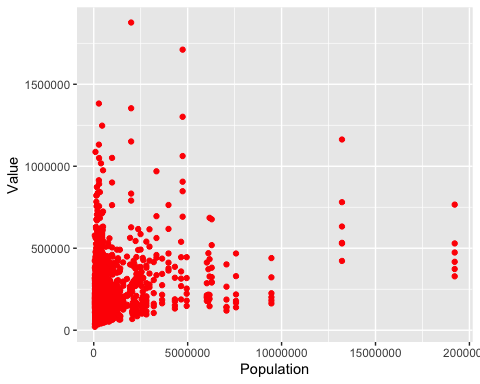
ggplot(data = dat.XG, aes(x=UnEmpRate))+geom\_density(fill = 'purple')+  
 xlab('Unemployment Rate (%)')



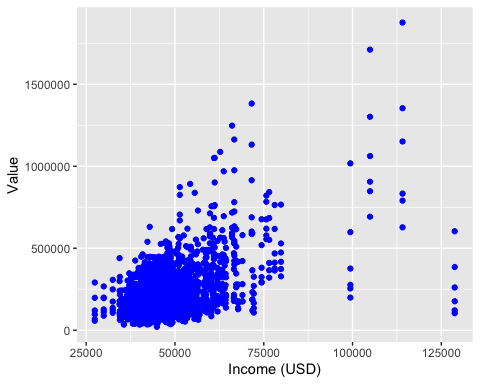
ggplot(data = dat.XG, aes(x=Value))+geom\_density(fill = 'green')+  
 xlab('Zillow Home Value Estimate (USD)')



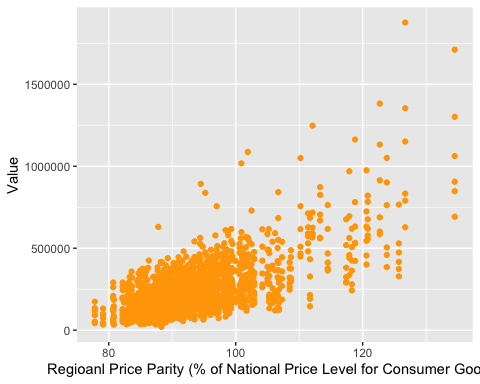
# scatter plots of "Value" as function of continuous predictors  
ggplot(data = dat.XG, aes(x=POP, y=Value))+geom\_point(color = 'red')+  
 xlab("Population")



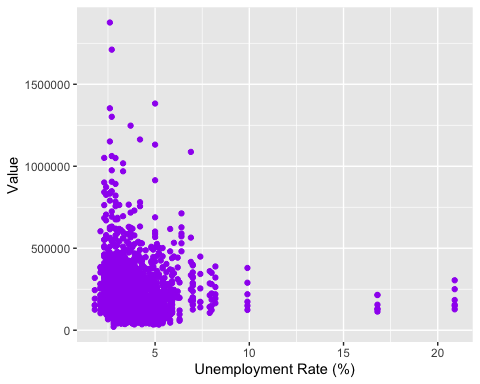
ggplot(data = dat.XG, aes(x=Income, y= Value))+geom\_point(color = 'blue')+  
 xlab('Income (USD)')



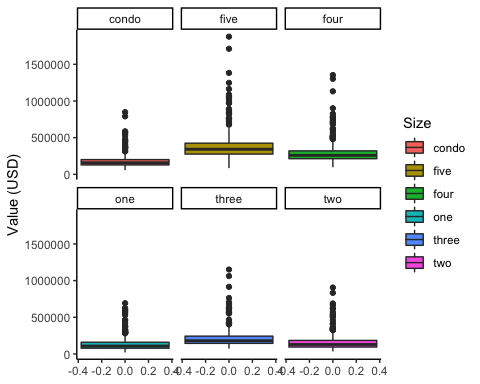
ggplot(data = dat.XG, aes(x=RPP, y=Value))+geom\_point(color = 'orange')+  
 xlab('Regioanl Price Parity (% of National Price Level for Consumer Goods)')



ggplot(data = dat.XG, aes(x=UnEmpRate, y=Value))+geom\_point(color = 'purple')+  
 xlab('Unemployment Rate (%)')



# distribution of home values by size (histogram)  
ggplot(data = dat.XG, aes(y=Value, group = Size, fill = Size))+  
 geom\_boxplot()+  
 theme\_classic()+  
 ylab("Value (USD)")+  
 facet\_wrap(~Size)



# quantitative assessment of home values by size  
dat.XG %>% group\_by(Size)%>% summarize(mean(Value), median(Value))

## # A tibble: 6 × 3  
## Size `mean(Value)` `median(Value)`  
## <fct> <dbl> <dbl>  
## 1 condo 180784. 158256.  
## 2 five 383533. 342322.  
## 3 four 293933. 259084.  
## 4 one 135075. 108036   
## 5 three 215871. 180835.  
## 6 two 159348. 129430.

# split data into training and test sets  
n <- nrow(dat.XG)  
# shuffle data  
dat.XG <- dat.XG[sample(1:n),]  
# set seed for reproducability  
set.seed(123456)  
# create random sample of int 0-1 from n, proportions .70, .30  
tv.split <- sample(rep(0:1,c(round(n\*.3),n-round(n\*.3))),n)  
# print table of counts (0, 1) for verification   
table(tv.split)

## tv.split  
## 0 1   
## 640 1495

# training set (1495)  
xg.train <- dat.XG[tv.split==1,]  
# test set (640)  
xg.test <- dat.XG[tv.split == 0,]

###### Alternaitve Method for Data Prep ######  
# create matrix of predictors  
#X.train <- data.matrix(xg.train[, -7])  
#X.test <- data.matrix(xg.test[, -7])  
# create vector of outcomes   
#Y.train <- data.matrix(xg.train[, 7])  
#Y.test <- data.matrix(xg.test[, 7])  
# create xgb.DMatrix for training and test  
#xgb\_train <- xgb.DMatrix(data = X.train, label = Y.train)  
#xgb\_test <- xgb.DMatrix(data = X.test, label = Y.test)

# define pre-processing steps for the data "recipe"  
recipe <-  
 # provide data and formula  
 recipes::recipe(Value ~ ., data = xg.train) %>%  
 # convert categorical variables to factors  
 recipes::step\_string2factor(all\_nominal()) %>%   
 prep()

# set seed for reproducability  
set.seed(123456)  
# create folds for cross validation  
cv\_folds <-   
 # treat data using recipe  
 recipes::bake(recipe,new\_data = xg.train) %>%  
 # specify num\_folds  
 rsample::vfold\_cv(v = 5)

# XGBoost model specification using parsnip package, boost\_tree function  
model.XG <-   
 parsnip::boost\_tree(  
 mode = "regression",  
 trees = tune(), # number of trees to build during model fitting  
 min\_n = tune(), # min number of data points required for split in tree node  
 tree\_depth = tune(), # max splits for trees  
 learn\_rate = tune(), # rate at which model minimizes loss function  
 loss\_reduction = tune(), # min reduction in loss function required for split  
 sample\_size = 0.5,  
 stop\_iter = 10 # stop tuning if no improvement after 10 iterations  
 ) %>%  
 # specify XGBoost algorithm as engine   
 set\_engine("xgboost", objective = "reg:squarederror")

# specify parameters for grid search  
xgboost\_params <-   
 dials::parameters(  
 min\_n(),  
 tree\_depth(),  
 learn\_rate(),  
 loss\_reduction(),  
 trees()  
 )  
# set grid space  
xgboost\_grid <-   
 dials::grid\_max\_entropy(  
 xgboost\_params,   
 size = 100 # (5 CV folds) \* (size = 100), 500 parameter combinations  
 )  
# print head for search grid  
head(xgboost\_grid)

## # A tibble: 6 × 5  
## min\_n tree\_depth learn\_rate loss\_reduction trees  
## <int> <int> <dbl> <dbl> <int>  
## 1 6 4 3.08e- 4 5.41e-10 266  
## 2 27 9 2.96e- 9 1.70e- 1 1843  
## 3 3 8 5.92e-10 1.42e+ 1 1839  
## 4 9 12 2.60e- 8 4.65e- 4 430  
## 5 25 15 2.01e- 9 1.29e-10 125  
## 6 24 4 9.36e- 2 3.23e- 6 1598

# define the workflow  
xgboost\_wf <-   
 workflows::workflow() %>%  
 add\_model(model.XG) %>% # add the XGBoost model defined with parsnip  
 add\_formula(Value ~ .) # specify formula

# tune the model using tidymodels tune()  
xgboost\_tuned <- tune::tune\_grid(  
 object = xgboost\_wf, # specified workflow  
 resamples = cv\_folds, # specified cross-validation  
 grid = xgboost\_grid, # specified search grid  
 metrics = yardstick::metric\_set(rmse), # metric for tuning rmse   
 control = tune::control\_grid(verbose = TRUE)  
)

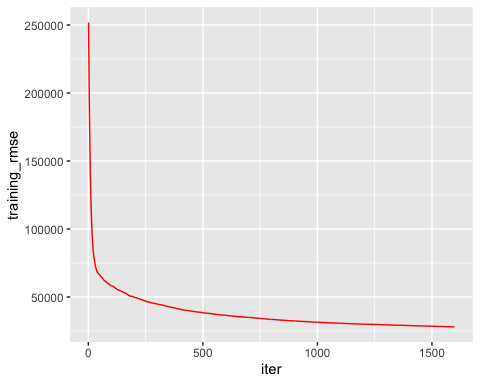
# select and view best parameters from model tuning  
best\_params <- xgboost\_tuned %>%  
 tune::select\_best("rmse")  
# print best params  
best\_params

## # A tibble: 1 × 6  
## trees min\_n tree\_depth learn\_rate loss\_reduction .config   
## <int> <int> <int> <dbl> <dbl> <chr>   
## 1 1598 24 4 0.0936 0.00000323 Preprocessor1\_Model056

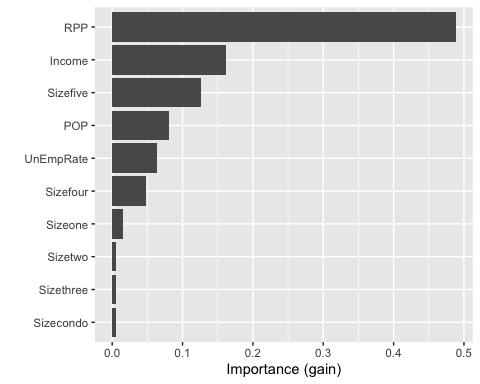
# finalize the model  
model.best <- model.XG %>% finalize\_model(best\_params)

# fit the best model on the training data  
train <- bake(recipe, new\_data = xg.train)  
model.best.fit <- model.best %>% fit(formula = Value ~ ., data = train)

# vislual assessment of model.best.fit  
ggplot(model.best.fit$fit$evaluation\_log) +  
 geom\_line(aes(iter, training\_rmse), color = "red")



# plot variable importance  
vip(model.best.fit, include\_type= TRUE)



# make prediction on training data  
predict(model.best.fit, train)%>%  
bind\_cols(xg.train)%>%  
yardstick::metrics(Value, .pred)

## # A tibble: 3 × 3  
## .metric .estimator .estimate  
## <chr> <chr> <dbl>  
## 1 rmse standard 28091.   
## 2 rsq standard 0.968  
## 3 mae standard 18844.

# make a prediction on the test data  
test <- bake(recipe, new\_data = xg.test)  
predict(model.best.fit, test)%>%  
bind\_cols(xg.test)%>%  
yardstick::metrics(Value, .pred)

## # A tibble: 3 × 3  
## .metric .estimator .estimate  
## <chr> <chr> <dbl>  
## 1 rmse standard 59197.   
## 2 rsq standard 0.888  
## 3 mae standard 32768.