Property Value Prediction Using Machine Learning

Tatsiana Sokalava

September 2020

INTRODUCTION

This research focuses on analyzing a data set from the Property Appraiser's office in Orange County, Florida. The Property Appraiser (Assessor) office in Florida is a local government agency that determines the values of properties within a certain geographic area for taxation purposes. The goal of the office is to fairly and equitably determine the value of properties based on various characteristics and market conditions. The office uses mass appraisal techniques in determining property values. With the support of the computer-assisted mass appraisal system (CAMA), which allows to model and mass update data, the analysts at the property appraiser's office evaluate each property strata to ensure the assessments are within the acceptable standards defined by the Florida Department of Revenue (DOR) and the appraisal industry.

The data set used in this work lists qualified single-family residential properties sold in 2019. To be qualified, the sale of a property had to meet certain pre-defined criteria for its Sales Price to be an indication of the property's market value. Each property is identified by a unique parcel ID, situs address, various location characteristics, and property features. The goal of this project is to build an ensemble model that predicts the values of the properties using these characteristics and features. The performance of the model is measured by the value of the mean squared error (MSE), the square of the difference between the predictions and the targets. This is a widely used loss function for regression problems such as predicting property values. The key steps involved in the analysis are wrangling and exploring the data, feature engineering and preprocessing of the data, tunning model's parameters, fitting various models (including a deep learning model), and building the final ensemble model that yields the lowest MSE. Given that this model needs to meet the requirements defined by the DOR, we will also test it against the acceptable ratio standards.

We start by loading the necessary packages and libraries.

```
library(keras)
install_keras() # or install_keras(tensorflow = "gpu")
```

```
devtools::install_github("kassambara/ggpubr")
library(ggpubr)
```

```
library(tidyverse)
library(lubridate)
library(dslabs)
library(dplyr)
library(leaflet)
library(data.table)
library(matrixStats)
library(raster)
library(corrplot)
library(scales)
library(gam)
library(gbm)
library(xgboost)
```

```
library(Rborist)
library(reticulate)
library(caret)
library(RColorBrewer)
library(knitr)
```

Let's load and view the data set and its features.

```
str(sqtest, vec.len=2)
```

```
## tibble [28,085 x 27] (S3: tbl_df/tbl/data.frame)
##  $ Sale Date      : chr [1:28085] "12/31/2019" "12/31/2019" ...
##  $ Display Parcel : chr [1:28085] "03-20-27-8438-03-370" "03-20-27-8438-03-370" ...
##  $ Township       : chr [1:28085] "20" "20" ...
##  $ Range          : chr [1:28085] "27" "27" ...
##  $ Situs_Address  : chr [1:28085] "5946 RUTHERFORD RD" "5946 RUTHERFORD RD" ...
##  $ City           : chr [1:28085] "Mount Dora" "Mount Dora" ...
##  $ Land Sq Ft     : num [1:28085] 9003 9003 ...
##  $ Adj Sale Amt   : num [1:28085] 250500 250500 ...
##  $ City Code      : chr [1:28085] "ORG" "ORG" ...
##  $ HEATED_AREA    : num [1:28085] 1572 1572 ...
##  $ Actual Area    : num [1:28085] 2356 2356 ...
##  $ DOR_CODE       : chr [1:28085] "0103" "0103" ...
##  $ MKT_AREA_CODE  : chr [1:28085] "06" "06" ...
##  $ NBHD_DESC      : chr [1:28085] "STONEYBROOK NORTH" "STONEYBROOK NORTH" ...
##  $ LONGITUDE      : num [1:28085] -81.6 -81.6 ...
##  $ LATITUDE       : num [1:28085] 28.8 28.8 ...
##  $ Price Per Sq Ft: num [1:28085] 159 159 ...
##  $ BLDG_NUM       : num [1:28085] 1 1 1 1 1 ...
##  $ Sales Ratio    : num [1:28085] 76.2 76.2 ...
##  $ BEDS           : num [1:28085] 3 3 3 3 3 ...
##  $ BATHS          : num [1:28085] 2 2 1.5 1.5 1.5 ...
##  $ EYB            : chr [1:28085] "10/04/2007" "10/04/2007" ...
##  $ AYB            : chr [1:28085] "10/04/2007" "10/04/2007" ...
##  $ STORIES        : num [1:28085] 1 1 1 1 1 ...
##  $ Exterior Wall  : chr [1:28085] "CB.STUCCO" "CB.STUCCO" ...
##  $ ZIP Code       : chr [1:28085] "32757" "32757" ...
##  $ XFOB Desc      : chr [1:28085] "POOL 1" "SCRN ENC 2" ...
```

The data frame consists of 28,085 objects represented by 27 variables of numeric and character classes. To understand the data, we can group variables by its type:

- **Date and unique identifier:** Sale Date, Display Parcel
- **Location:** Township, Range, Situs_Address, City, City Code, MKT_AREA_CODE, NBHD_DESC, LONGITUDE, LATITUDE, ZIP Code
- **Qualitative:** DOR_CODE, Price Per Sq Ft, Sales Ratio, EYB, AYB, Exterior Wall, XFOB Desc
- **Quantitative:** Land Sq Ft, HEATED_AREA, Actual Area, BLDG_NUM, BEDS, BATHS, STORIES

```
head(sqtest, n=20)
```

```
## # A tibble: 20 x 27
##    `Sale Date` `Display Parcel` Township Range Situs_Address City  `Land Sq Ft`
##    <chr>       <chr>            <chr>    <chr> <chr>         <chr>        <dbl>
##  1 12/31/2019  03-20-27-8438-0~ 20       27    5946 RUTHERF~ Moun~        9003.
##  2 12/31/2019  03-20-27-8438-0~ 20       27    5946 RUTHERF~ Moun~        9003.
##  3 12/31/2019  22-20-27-6888-0~ 20       27    5112 JONES A~ Zell~       45330.
##  4 12/31/2019  22-20-27-6888-0~ 20       27    5112 JONES A~ Zell~       45330.
##  5 12/31/2019  22-20-27-6888-0~ 20       27    5112 JONES A~ Zell~       45330.
##  6 12/31/2019  22-20-27-6888-0~ 20       27    5112 JONES A~ Zell~       45330.
##  7 12/31/2019  09-23-27-5844-0~ 23       27    4319 OLD SYC~ Wint~       10725.
##  8 12/31/2019  09-23-27-5844-0~ 23       27    4319 OLD SYC~ Wint~       10725.
##  9 12/31/2019  19-23-27-5840-0~ 23       27    17396 LAKE I~ Wint~       39176.
## 10 12/31/2019  19-23-27-5840-0~ 23       27    17396 LAKE I~ Wint~       39176.
## 11 12/31/2019  19-23-27-5840-0~ 23       27    17396 LAKE I~ Wint~       39176.
## 12 12/31/2019  31-23-27-8851-0~ 23       27    16142 HAMPTO~ Wint~        7097.
## 13 12/31/2019  31-23-27-8851-0~ 23       27    16142 HAMPTO~ Wint~        7097.
## 14 12/31/2019  31-23-27-8851-0~ 23       27    16142 HAMPTO~ Wint~        7097.
## 15 12/31/2019  02-21-28-8308-0~ 21       28    835 VOTAW RD  Apop~      228737.
## 16 12/31/2019  02-21-28-8308-0~ 21       28    835 VOTAW RD  Apop~      228737.
## 17 12/31/2019  02-21-28-8308-0~ 21       28    835 VOTAW RD  Apop~      228737.
## 18 12/31/2019  02-21-28-8308-0~ 21       28    835 VOTAW RD  Apop~      228737.
## 19 12/31/2019  02-21-28-8308-0~ 21       28    835 VOTAW RD  Apop~      228737.
## 20 12/31/2019  02-21-28-8308-0~ 21       28    835 VOTAW RD  Apop~      228737.
## # ... with 20 more variables: `Adj Sale Amt` <dbl>, `City Code` <chr>,
## #   HEATED_AREA <dbl>, `Actual Area` <dbl>, DOR_CODE <chr>,
## #   MKT_AREA_CODE <chr>, NBHD_DESC <chr>, LONGITUDE <dbl>, LATITUDE <dbl>,
## #   `Price Per Sq Ft` <dbl>, BLDG_NUM <dbl>, `Sales Ratio` <dbl>, BEDS <dbl>,
## #   BATHS <dbl>, EYB <chr>, AYB <chr>, STORIES <dbl>, `Exterior Wall` <chr>,
## #   `ZIP Code` <chr>, `XFOB Desc` <chr>
```

The data is displayed in the long format and is not tidy. We note that some parcels are repeated and represented by various combinations of *Exterior Wall* and *XFOB Desc* observations. These stand for the quality of construction *(Exterior Wall)* and various extra features *(XFOB Desc)* such as pool, screen, patio, etc.

DATA WRANGLING

To make the data tidy, we will apply several data wrangling techniques to rename variables, convert variable types, remove repeated observations, and spread extra features (XFOB Desc) into a wide format.

```
sqtest<-sqtest%>%rename(c("Sale_Date"="Sale Date", "Parcel"="Display Parcel", "Township"=
                           "Township","Range"="Range", "Situs_Address"="Situs_Address",
                         "City"="City", "Land_Sq"="Land Sq Ft", "Sales_Price"=
                           "Adj Sale Amt", "City_Code" = "City Code", "HEATED_AREA"=
                           "HEATED_AREA", "AREA"="Actual Area",
                         "DOR_CODE"="DOR_CODE","MARKET"="MKT_AREA_CODE","NBHD_DESC"=
                           "NBHD_DESC","LONGITUDE"=
                           "LONGITUDE","LATITUDE"="LATITUDE","PPSF"="Price Per Sq Ft",
                         "BLDG_NUM"="BLDG_NUM","SR"="Sales Ratio","BEDS"="BEDS",
                         "BATHS"="BATHS","EYB"="EYB","AYB"="AYB","STORIES"="STORIES",
                         "EXT_W"= "Exterior Wall","ZIP_Code"="ZIP Code",
                         "XFOB"="XFOB Desc"))
sqtest<-sqtest%>%mutate(Sale_Date=as.Date(sqtest$Sale_Date, "%m/%d/%Y"),XFOB=
                          as.factor(XFOB), EXT_W=as.factor(EXT_W), ZIP_Code=
                          as.factor(ZIP_Code), Township=as.numeric(Township),Range=
                          as.numeric(Range),City_Code=as.factor(City_Code),
                        DOR_CODE=as.numeric(DOR_CODE),MARKET=as.numeric(MARKET),LONGITUDE=
                          as.numeric(LONGITUDE), LATITUDE=as.numeric(LATITUDE),
                        EYB=year(mdy(EYB)), AYB=year(mdy(AYB)))
sqtest<-sqtest%>%arrange(BLDG_NUM)%>%distinct(Sale_Date, Parcel,XFOB, .keep_all = TRUE)
sqtest<-sqtest%>%mutate(XFOB_NAME=
                        as.factor((str_extract(XFOB,"[:letter:]+[:space:]?[:letter:]+"))),
                        XFOB_Q=as.numeric(str_extract(XFOB, "[[:digit:]]+")))%>%
  mutate(XFOB_Q=ifelse(is.na(XFOB_Q),1,XFOB_Q))
sqtest<-sqtest%>%group_by(Sale_Date, Parcel,Township, Range,Situs_Address, City,Land_Sq,
                          Sales_Price,City_Code,HEATED_AREA,AREA, DOR_CODE,MARKET,
                          NBHD_DESC, LONGITUDE,LATITUDE,PPSF,BLDG_NUM,SR,BEDS, BATHS,
                          EYB,AYB,STORIES,EXT_W,ZIP_Code, XFOB_NAME)%>%
  summarize(XFOB_Q=sum(XFOB_Q))
sqtest<-sqtest%>%spread(key=XFOB_NAME, XFOB_Q,0)%>%ungroup()
sqtest<-rename(sqtest, c( "ACC_BLD"="ACC BLD","ACC_BLDG"="ACC BLDG","BOAT_COVER"=
                            "BOAT COVER","BOAT_DOCK"=
                            "BOAT DOCK","BOAT_HOUSE"="BOAT HOUSE","COVER_FG"="COVER FG",
                          "COVER_WD"="COVER WD" , "SMALL_SHED"="SMALL SHED",
                          "COVER_ALUM"="COVER ALUM","COVER_STL"="COVER STL",
                          "HORSE_STAB"="HORSE STAB","MOBILE_HM"="MOBILE HM",
                          "SUMMER_KITCHEN"="SUMMER KITCHEN", "PARKING_SPACE"=
                            "PARKING SPACE", "PO_BLDG"="PO BLDG", "PATIO_NO"="PATIO NO",
                          "PATIO_WD"="PATIO WD","PAV_CON"="PAV CON","POLE_BLDG"=
                            "POLE BLDG","RESIDENTIAL_EL"="RESIDENTIAL ELEVATOR",
                          "RM_ENCL"="RM ENCL","SCRN_ENC"="SCRN ENC", "SHED_N"="SHED N",
                          "SUMMER_KITCHEN"="SUMMER KITCHEN", "WALL_CB"="WALL CB",
                          "WALL_DEC"="WALL DEC", "WALL_NO"="WALL NO"))
```

Let's view the structure of our data set after wrangling. Now, it appears tidy, but it is not yet ready to be put through an ML model. Some variable features need to be further evaluated and re-engineered before we fit a model.

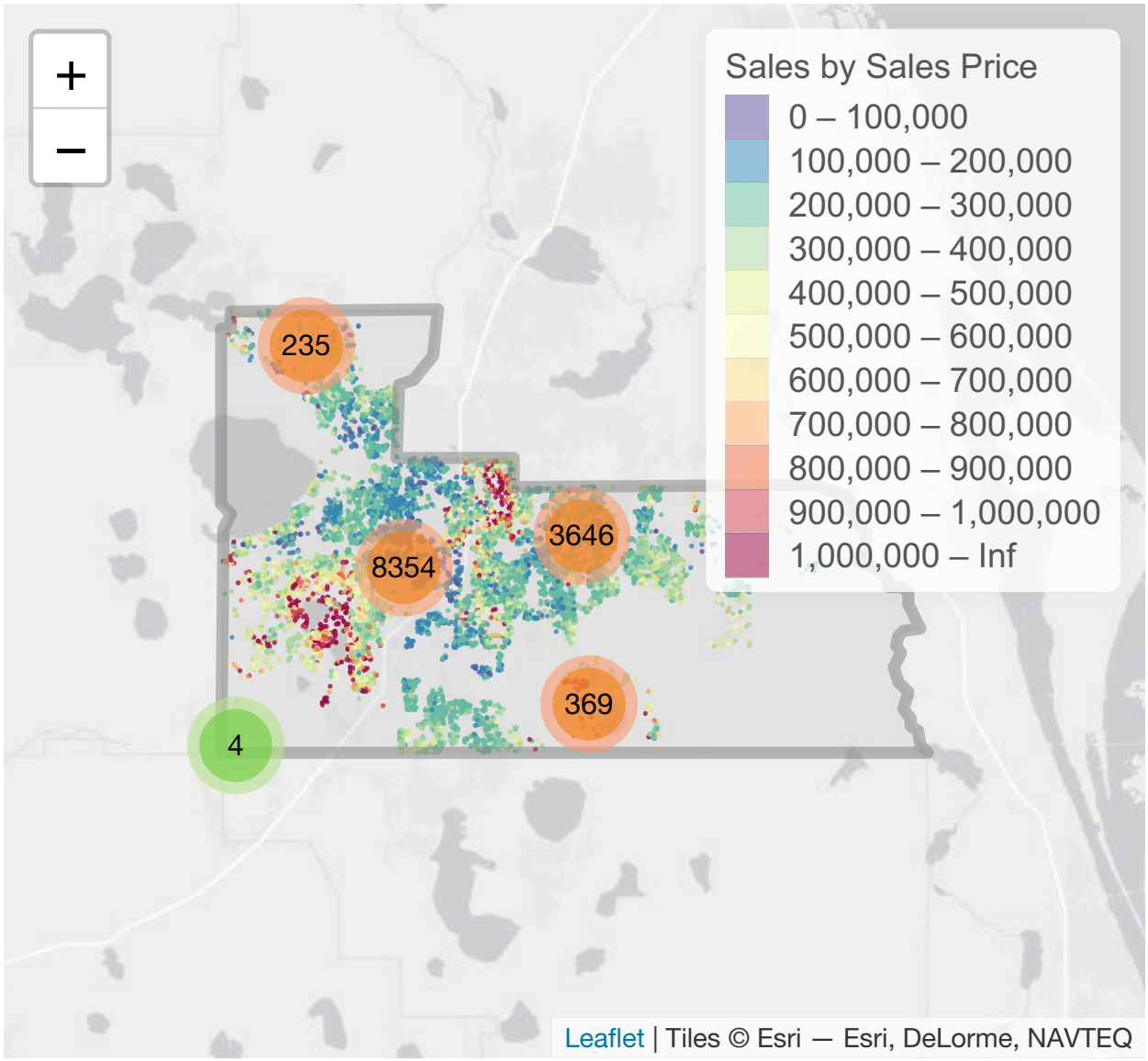
```r
str(sqtest, vec.len=2, list.len=50)
```

```
## tibble [12,608 x 68] (S3: tbl_df/tbl/data.frame)
##  $ Sale_Date     : Date[1:12608], format: "2019-01-01" "2019-01-01" ...
##  $ Parcel        : chr [1:12608] "02-22-31-7840-00-350" "18-24-31-9163-05-730" ...
##  $ Township      : num [1:12608] 22 24 23 23 22 ...
##  $ Range         : num [1:12608] 31 31 32 29 28 ...
##  $ Situs_Address : chr [1:12608] "4220 FOREST ISLAND DR" "11971 LAZIO LN" ...
##  $ City          : chr [1:12608] "Orlando" "Orlando" ...
##  $ Land_Sq       : num [1:12608] 5251 8043 ...
##  $ Sales_Price   : num [1:12608] 257000 400000 ...
##  $ City_Code     : Factor w/ 12 levels "APK","BI","EDG",..: 8 9 8 9 7 ...
##  $ HEATED_AREA   : num [1:12608] 1907 2023 ...
##  $ AREA          : num [1:12608] 2327 2707 ...
##  $ DOR_CODE      : num [1:12608] 103 103 103 103 102 ...
##  $ MARKET        : num [1:12608] 2 12 4 28 9 ...
##  $ NBHD_DESC     : chr [1:12608] "SANCTUARY" "VILLAGE WALK AT LAKE NONA" ...
##  $ LONGITUDE     : num [1:12608] -81.2 -81.3 ...
##  $ LATITUDE      : num [1:12608] 28.6 28.4 ...
##  $ PPSF          : num [1:12608] 135 198 ...
##  $ BLDG_NUM      : num [1:12608] 1 1 1 1 1 ...
##  $ SR            : num [1:12608] 84.7 78.4 ...
##  $ BEDS          : num [1:12608] 4 3 3 3 4 ...
##  $ BATHS         : num [1:12608] 2 2 2 1.5 3 ...
##  $ EYB           : num [1:12608] 1999 2008 ...
##  $ AYB           : num [1:12608] 1999 2008 ...
##  $ STORIES       : num [1:12608] 1 1 1 1 1 ...
##  $ EXT_W         : Factor w/ 27 levels "ABOVE.AVG","ALUM/VYLSD",..: 6 6 6 6 6 ...
##  $ ZIP_Code      : Factor w/ 43 levels "32703","32709",..: 29 30 34 18 41 ...
##  $ ACC_BLD       : num [1:12608] 0 0 0 0 0 ...
##  $ ACC_BLDG      : num [1:12608] 0 0 0 0 0 ...
##  $ BARN          : num [1:12608] 0 0 0 0 0 ...
##  $ BOAT_COVER    : num [1:12608] 0 0 0 0 0 ...
##  $ BOAT_DOCK     : num [1:12608] 0 0 0 0 0 ...
##  $ BOAT_HOUSE    : num [1:12608] 0 0 0 0 0 ...
##  $ CARPORT       : num [1:12608] 0 0 0 0 0 ...
##  $ COVER_ALUM    : num [1:12608] 0 0 0 0 0 ...
##  $ COVER_FG      : num [1:12608] 0 0 0 0 0 ...
##  $ COVER_STL     : num [1:12608] 0 0 0 0 0 ...
##  $ COVER_WD      : num [1:12608] 0 0 0 0 0 ...
##  $ CT            : num [1:12608] 0 0 0 0 0 ...
##  $ ELEVATOR      : num [1:12608] 0 0 0 0 0 ...
##  $ FOUNTAIN      : num [1:12608] 0 0 0 0 0 ...
##  $ FPLACE        : num [1:12608] 0 0 0 0 0 ...
##  $ GAZEBO        : num [1:12608] 0 0 0 0 0 ...
##  $ GRNHS         : num [1:12608] 0 0 0 0 0 ...
##  $ HORSE_STAB    : num [1:12608] 0 0 0 0 0 ...
##  $ HORSE STABLE  : num [1:12608] 0 0 0 0 0 ...
##  $ MOBILE_HM     : num [1:12608] 0 0 0 0 0 ...
##  $ PARKING_SPACE : num [1:12608] 0 0 0 0 0 ...
##  $ PATIO         : num [1:12608] 0 0 0 0 0 ...
##  $ PATIO_NO      : num [1:12608] 1 1 0 0 0 ...
##  $ PATIO_WD      : num [1:12608] 0 0 0 0 0 ...
##   [list output truncated]
```

VISUALIZATION AND RATIO STANDARDS

Let's visualize our tidy data set on the Orange County, Florida map using *Leaflet* package.

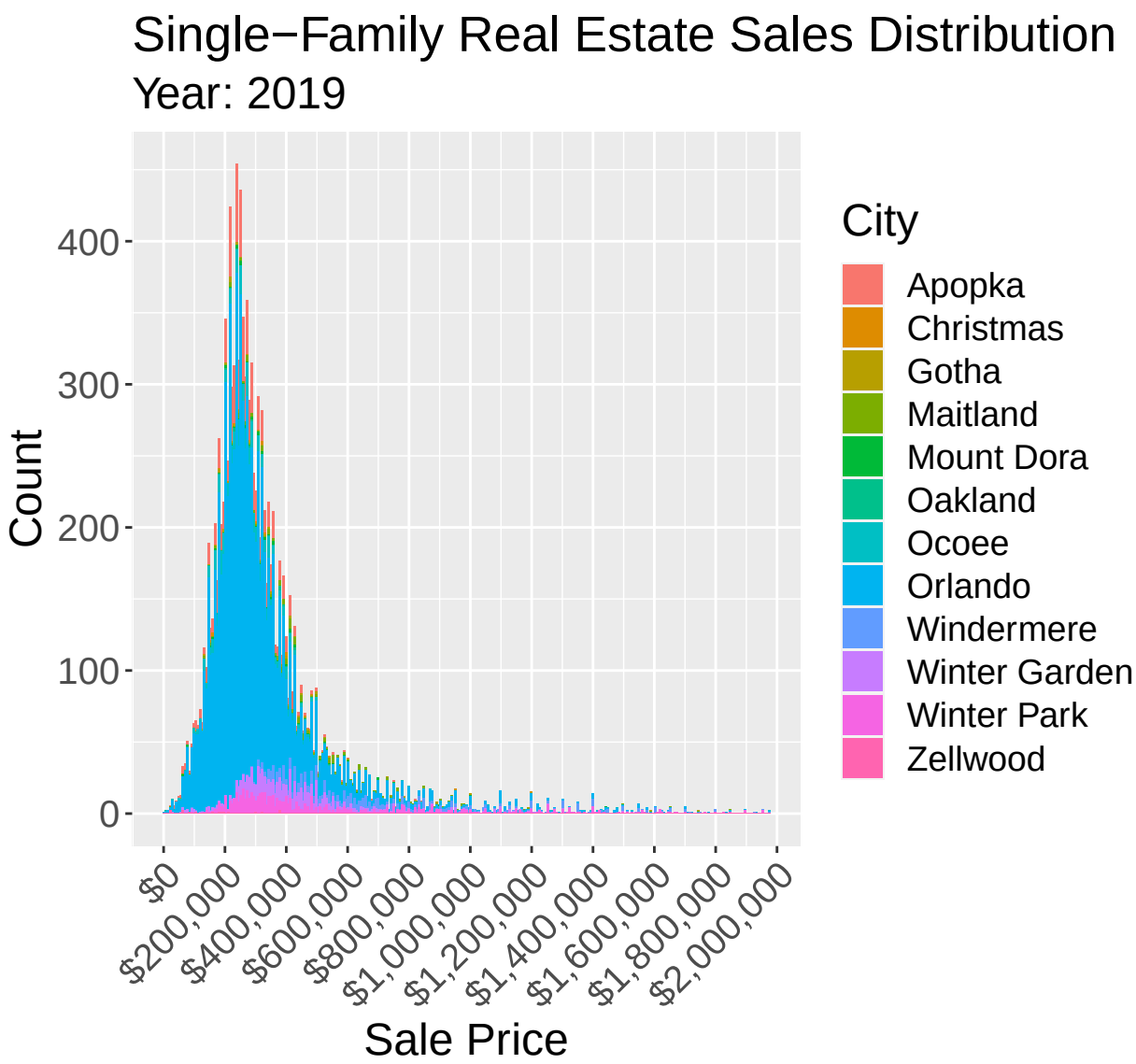
```
county <-subset(raster::getData("GADM", country="usa", level=2),
                NAME_1=="Florida"&NAME_2=="Orange")
mybins<-c(0,100000,200000,300000,400000,500000,600000,700000,800000,900000,1000000,Inf)
pal<-colorBin(palette="Spectral",domain=sqtest$Sales_Price, bins=mybins,reverse=T)

map<-sqtest%>%leaflet()%>%addProviderTiles(providers$Esri.WorldGrayCanvas)%>%
  addCircles(radius=50, color=pal(sqtest$Sales_Price), weight = 3, stroke = FALSE,
             fillOpacity = 0.8)%>%addCircleMarkers(color=pal(sqtest$Sales_Price),
                                                   clusterOptions = markerClusterOptions(),
                                                    labelOptions =
                                                     labelOptions( opacity=0.9,
                                                                   color="grey",textsize =
                                                                     "10px",
                                                                   direction="right"))%>%
  addPolygons(data = county,fillOpacity = 0.1, color = "grey", stroke = TRUE,
              weight = 5,layerId = county@data$NAME_2)%>%
  leaflet::addLegend("topright", pal=pal, values=sqtest$Sales_Price,
                     title = "Sales by Sales Price")
map
```



It is evident that data points are clustered around certain areas. In the central area of the county, there are 8,354 property sales. The majority of sales over $800,000 (marked in orange/red) are clustered around the lakes.

Let's review the distribution of the Sales Prices. For better representation, we filtered the Sales Prices under $2 million. The majority of sales records are from the Orlando area. The distribution is right-skewed with a median somewhere between $200,000 to 400,000.

```
sqtest%>%filter(Sales_Price<2000000)%>%ggplot(aes(Sales_Price, fill=City))+
  geom_histogram(binwidth=7000)+
  scale_x_continuous(labels=dollar, breaks = scales::pretty_breaks(n = 10))+
  theme(axis.text.x = element_text(angle=45, hjust=1, size=15),
        plot.title = element_text(size=20),plot.subtitle = element_text(size=18),
        legend.title = element_text(size=18), legend.text = element_text(size=14),
        axis.title = element_text(size=18), axis.text.y = element_text(size=15))+
  labs(title = 'Single-Family Real Estate Sales Distribution',
       subtitle= 'Year: 2019', x = 'Sale Price', y = 'Count')
```



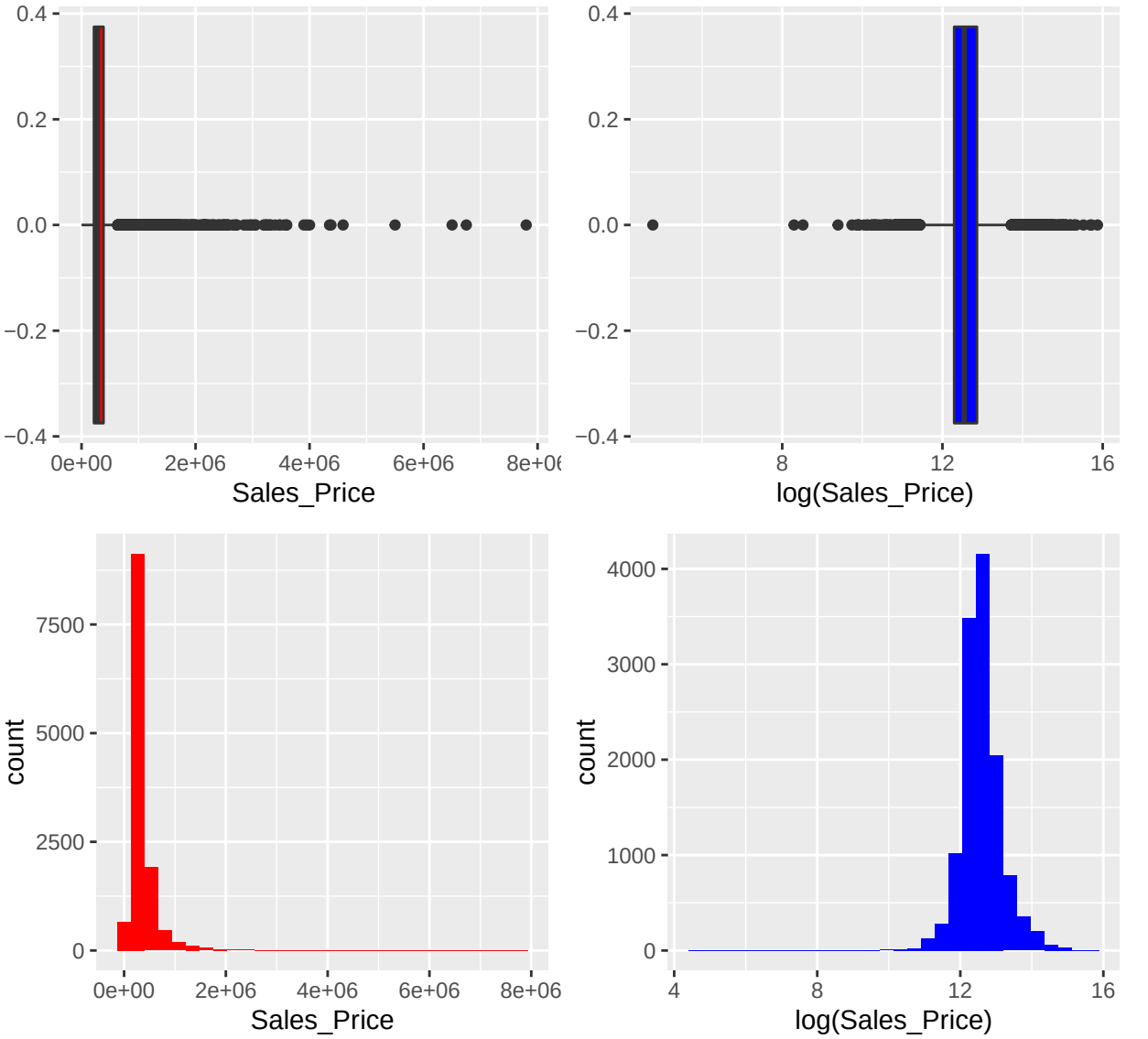
Let's calculate the Mean and the Median Sales Price in 2019 for single-family homes.

```
sqtest%>%summarize(MEAN=mean(Sales_Price), MEDIAN=median(Sales_Price))%>%data.frame()
```

```
##       MEAN MEDIAN
## 1 355988.2 280000
```

The Mean is $355,988, when the Median is $280,000. That signals that the data is skewed. Let's review the boxplot and histograms of the sales data. Since the data is right-skewed, we test the effect of a logarithmic transformation on the *Sales Price*.

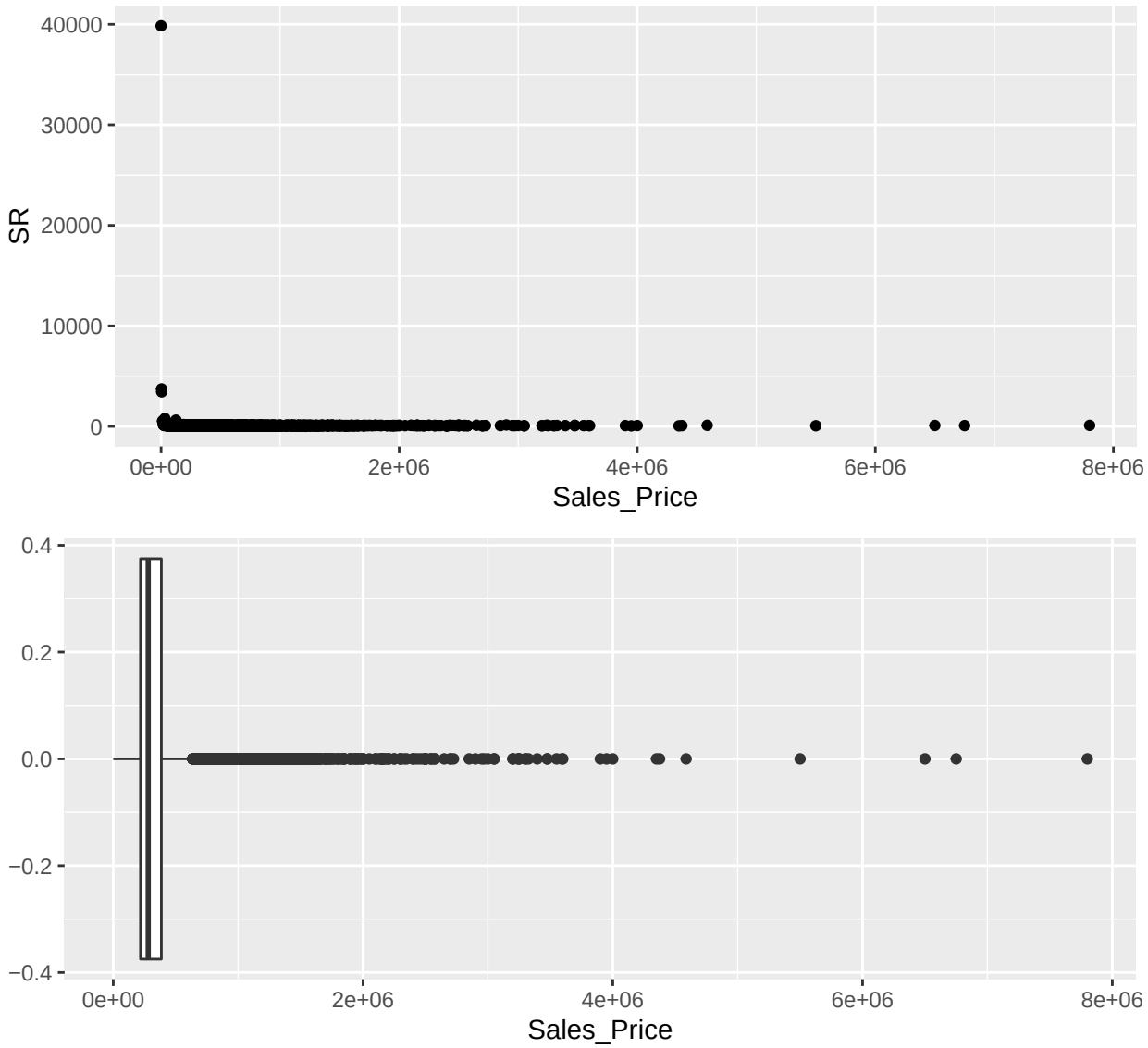
```
a<-ggplot(sqtest)+geom_boxplot(aes(Sales_Price), fill="red")
b<-ggplot(sqtest)+geom_boxplot(aes(log(Sales_Price)), fill="blue")
c<-ggplot(sqtest)+geom_histogram(aes(Sales_Price), fill="red")
d<-ggplot(sqtest)+geom_histogram(aes(log(Sales_Price)), fill="blue")
ggpubr::ggarrange(a,b,c,d,ncol = 2, nrow = 2)
```



Let's now review the *SR* feature in the data set that stands for sales ratio. This ratio is calculated for each observation by dividing the Appraised (or Assessed) value of the property by the Sale Price.

It measures the quality of the assessments and is used by the DOR as a metric to determine the annual performance of the assessment roll produced by the assessors' office.

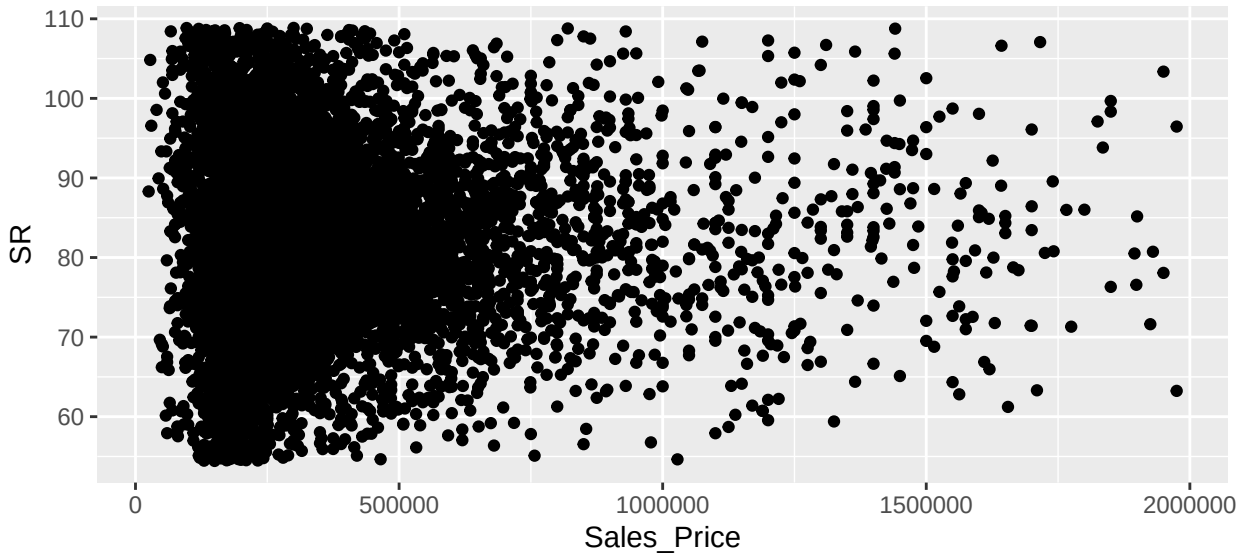
```
e<-ggplot(sqtest)+geom_point(aes(Sales_Price, SR))
f<-ggplot(sqtest)+geom_boxplot(aes(Sales_Price))
ggpubr::ggarrange(e,f,ncol = 1, nrow = 2)
```

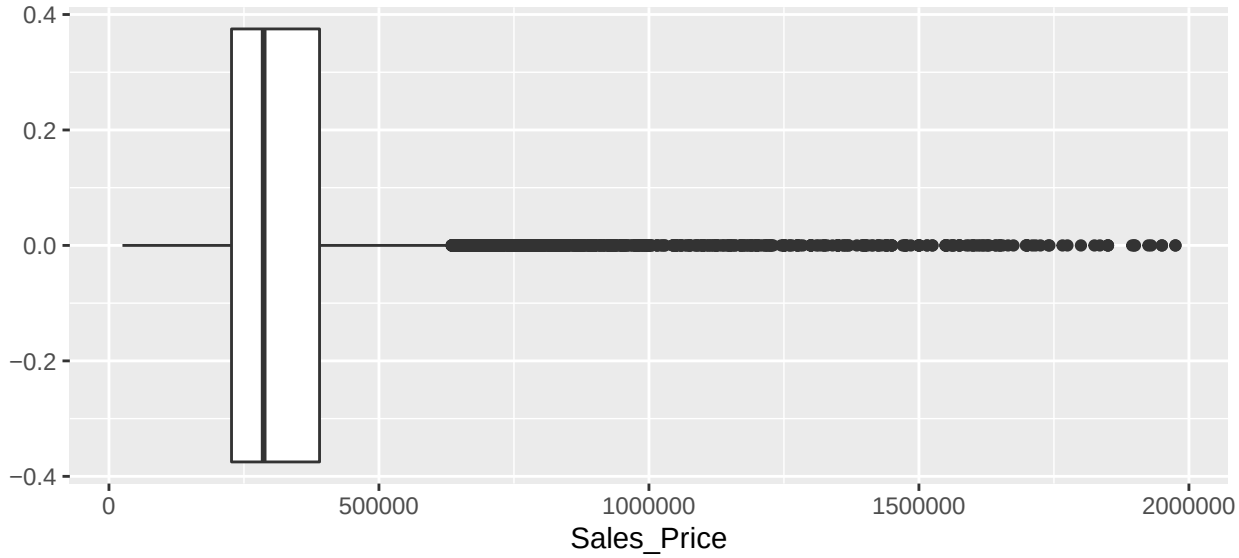


We observe from the plots that (1) the data set is affected by the sales ratio outliers (which often signal bias in the assessment or sales price); (2) the majority of properties are below $2 million range.

Therefore, to improve our model's performance, it is reasonable to remove the sales ratio (SR) outliers from the data set and sales observations over $2 million, which represent less than 1% of the data.

```
sqtest<-sqtest%>%filter(Sales_Price<2000000, !SR%in%boxplot(SR, plot=FALSE)$out)%>%
  mutate(Sale_Date=year(Sale_Date))
e<-ggplot(sqtest)+geom_point(aes(Sales_Price, SR))
f<-ggplot(sqtest)+geom_boxplot(aes(Sales_Price))
ggpubr::ggarrange(e,f,ncol = 1, nrow = 2)
```





For single-family residential properties (excluding condos), the acceptable Sales Ratio *(SR)* should be close to 85% due to a state requirement to adjust the sales value by 15% (referred to as 8th criteria).

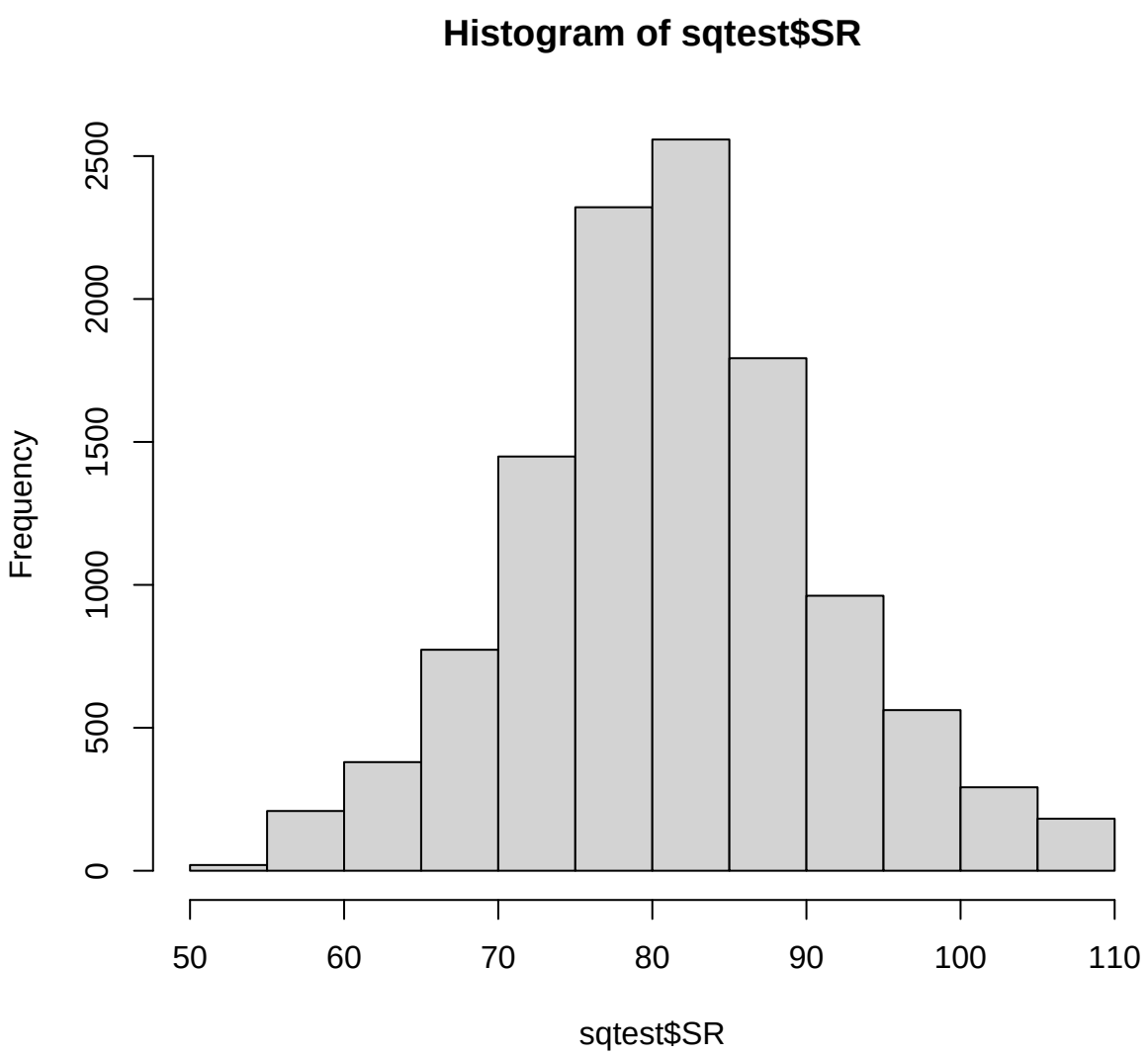
Additionally, assessment jurisdictions do an annual ratio study to determine the uniformity, variability, and equality of the assessment roll.

The most generally useful measure of variability or uniformity is the COD. It relates to "horizontal," or random, dispersion among the ratios in a stratum, regardless of the value of individual parcels. It is calculated by subtracting the median from each ratio, taking the absolute value of the calculated differences, summing the absolute differences, dividing by the number of ratios to obtain the average absolute deviation, dividing by the median, and multiplying by a 100. The acceptable COD level for the single-family type of properties is 5.0-15.0.

Another form of inequity relates to differences in the appraisal of low- and high-value properties, termed "vertical" inequities. An index for measuring vertical equity is the PRD, which is calculated by dividing the mean ratio by the weighted mean ratio. This statistic should be close to 1.00, but the acceptable level is 0.98-1.03.

So, let's see how these metrics hold on our data set:

```r
stat<-sqtest%>%summarise(MEAN=mean(SR), MEDIAN=median(SR))
hist(sqtest$SR)
```




```r
# We find confidence intervals using the following formula:
ci<-stat$MEAN+c(-qnorm(0.975),qnorm(0.975))*sd(sqtest$SR)

# Let's calculate the COD ratio:
COD<-sqtest%>%summarize(COD=mean(abs(SR-as.numeric(stat$MEDIAN)))/
                          as.numeric(stat$MEDIAN)*100)

# Let's also calculate the PRD ratio:
PRD<-sqtest%>%summarize(PRD=as.numeric(stat$MEAN)/(sum(sqtest$SR*sqtest$Sales_Price)/
                                                     sum(sqtest$Sales_Price)))
```

With that, we will use the following ratios to benchmark if the final model passes the assessment industry ratio standards:

```
original<-knitr::kable(cbind(MEAN=stat[1], MEDIAN=stat[2], CONF_LOW=ci[1],
                             CONF_HIGH=ci[2], COD=COD, PRD=PRD), "simple",
                       caption = "Original CAMA Ratios")
original
```

Table 1: Original CAMA Ratios

MEAN	MEDIAN	CONF_LOW	CONF_HIGH	COD	PRD
81.29171	81.14531	61.78642	100.797	9.518778	0.9939223

FEATURE ENGINEERING

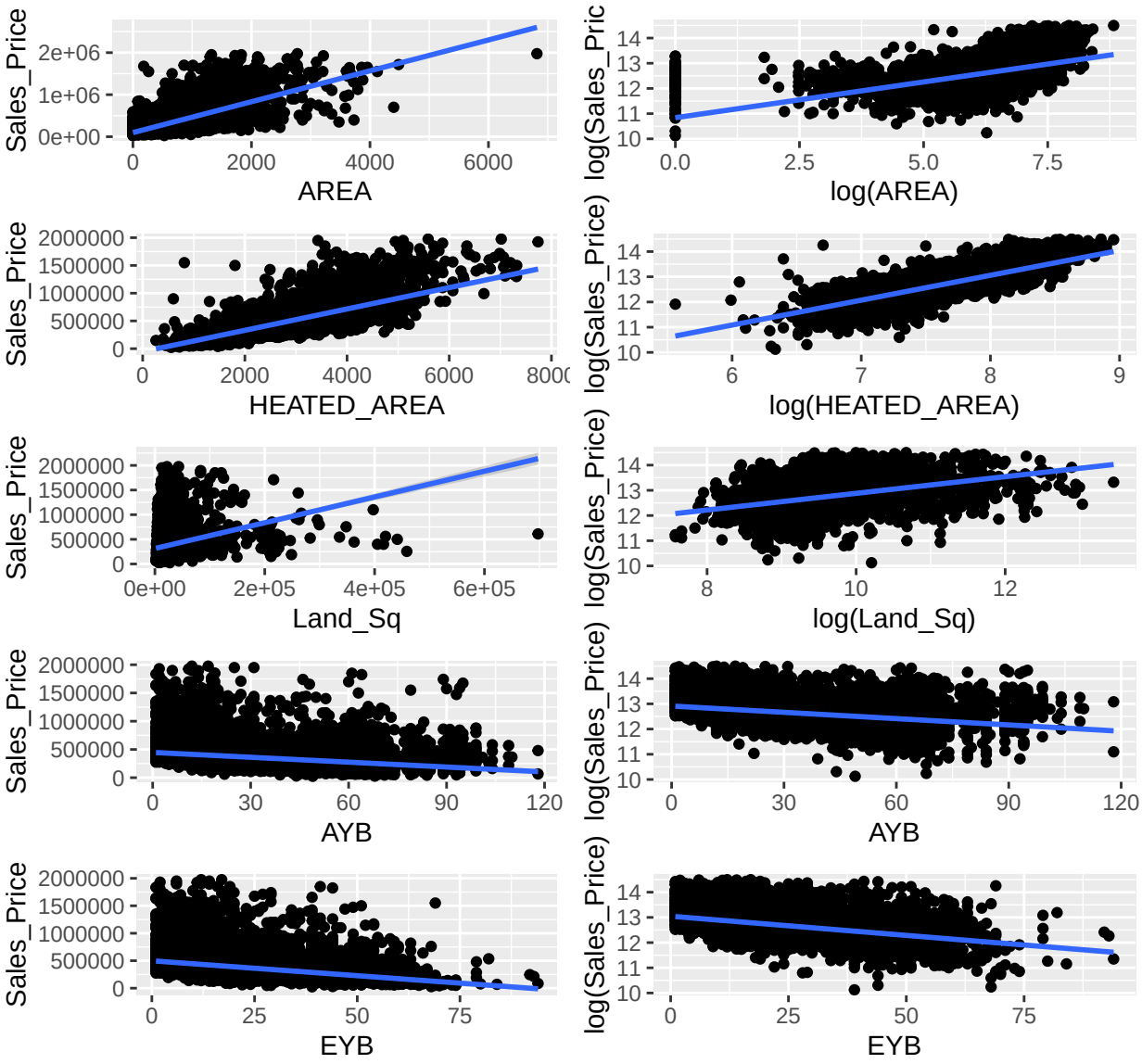
Features such as *EYB* and *AYB* stand for effective and actual year build. If a property had no renovations throughout its lifetime, these parameters will be the same. If a property had a significant renovation, the difference in *EYB* will be 10 years or more. Therefore, we will calculate the age of property by substructing it from a sales date.

HEATED AREA is one of the most important features when it comes to explaining variations in Sales Price. However, the total property base area *(AREA)* also has an effect. With that, we will adjust the feature by subtracting the heated area for each observation.

```
sqtest<-sqtest%>%mutate(EYB=Sale_Date-EYB, AYB=Sale_Date-AYB, AREA=AREA-HEATED_AREA)
```

Now, let's review the relationship between features and the outcome. Since a logarithmic transformation better explains variations in the data, we will apply and evaluate it on several features:

```
area<-sqtest%>%ggplot(aes(AREA,Sales_Price)) + geom_point() +
  geom_smooth(method="lm")
area_log2<-sqtest%>%mutate(AREA=ifelse(AREA==0, 1,AREA))%>%
  ggplot(aes(log(AREA),log(Sales_Price))) + geom_point() +
  geom_smooth(method="lm")
harea<-sqtest%>%ggplot(aes(HEATED_AREA,Sales_Price)) + geom_point() +
  geom_smooth(method="lm")
harea_log2<-sqtest%>%ggplot(aes(log(HEATED_AREA),log(Sales_Price))) + geom_point() +
  geom_smooth(method="lm")
land<-sqtest%>%ggplot(aes(Land_Sq,Sales_Price)) + geom_point() +
  geom_smooth(method="lm")
land_log2<-sqtest%>%ggplot(aes(log(Land_Sq),log(Sales_Price))) + geom_point() +
  geom_smooth(method="lm")
ayb<-sqtest%>%ggplot(aes(AYB,Sales_Price)) + geom_point() +
  geom_smooth(method="lm")
ayb_log<-sqtest%>%ggplot(aes(AYB,log(Sales_Price))) + geom_point() +
  geom_smooth(method="lm")
eyb<-sqtest%>%ggplot(aes(EYB,Sales_Price)) + geom_point() +
  geom_smooth(method="lm")
eyb_log<-sqtest%>%ggplot(aes(EYB,log(Sales_Price))) + geom_point() +
  geom_smooth(method="lm")
ggpubr::ggarrange(area, area_log2, harea, harea_log2, land, land_log2,
                  ayb,ayb_log, eyb,eyb_log,ncol = 2, nrow = 5)
```



Again, we see that the logarithmic transformation for sales price is reasonable as it creates a closer to a linear relationship between features such as Heated Area, Age, and Sales Price.

We also notice that the double log transformation (outcome and feature transformation) is useful for features related to the size of a property such as Area, Heated Area, and Land Size.

As a result, we do the transformations after removing zeros and NAs:

```
sqtest<-sqtest%>%mutate(AREA=ifelse(AREA==0, 1, AREA))%>%
  mutate(STORIES=ifelse(is.na(STORIES),1,STORIES),
         BEDS=ifelse(is.na(BEDS),1,BEDS), BATHS=ifelse(is.na(BATHS),1,BATHS))%>%
  mutate(Sales_Price=log(Sales_Price), AREA=log(AREA),
         HEATED_AREA=log(HEATED_AREA), Land_Sq=log(Land_Sq))
```

There are several other features listed in the columns 27 through 68. Having too many features increases the computation time of a model often without adding much impact on the outcome. Let's review the correlation to Sales Price and variability of these features. We will remove those with little or no correlation (less than $|x|<0.25$) and near-zero variability.

```
# Correlation:
correlation<-cor(sqtest[,8],sqtest[,27:ncol(sqtest)])
correlation
```

```
##                  ACC_BLD    ACC_BLDG       BARN BOAT_COVER BOAT_DOCK BOAT_HOUSE
## Sales_Price -0.003399965 -0.07948267 0.01406299  0.2609139 0.2673032 0.04492928
##                 CARPORT COVER_ALUM     COVER_FG   COVER_STL     COVER_WD
## Sales_Price -0.05741641 -0.1029199 -0.005141086 -0.02925084 -0.009511034
##                      CT   ELEVATOR  FOUNTAIN    FPLACE     GAZEBO        GRNHS
## Sales_Price 0.04477023 0.02994146 0.1661293 0.4200706 0.05005982 -0.005484353
##             HORSE_STAB HORSE STABLE    MOBILE_HM PARKING_SPACE      PATIO
## Sales_Price 0.01107023 -0.001586453 -0.005779447    0.01506275 0.06501222
##                PATIO_NO    PATIO_WD     PAV_CON      PO_BLDG   POLE_BLDG
## Sales_Price -0.04441993 -0.02043141 -0.02641678 0.0008084378 0.002550693
##                   POOL RESIDENTIAL_EL     RM_ENCL       SAUNA  SCRN_ENC
## Sales_Price 0.5366617     0.05278357 -0.08087024 0.009684062 0.2268007
##                SEAWALL SHADEHOUSE       SHED     SHED_N  SMALL_SHED       SPA
## Sales_Price 0.01635317 -0.0105572 -0.1926425 -0.2050584 -0.01741571 0.3218369
##             SUMMER_KITCHEN    WALL_CB  WALL_DEC    WALL_NO
## Sales_Price       0.322415 0.04867913 0.2080337 0.07839063
```

```
nocor<-which(correlation>-0.25&correlation<0.25|is.na(correlation))
X1<-colnames(sqtest[,27:ncol(sqtest)][,nocor])

# Variability:
subset<-data.matrix(sqtest)
mean<-apply(subset,2,mean)
std<-apply(subset,2,sd)
subset1<-scale(subset, center=mean,scale=std)
nd<-nearZeroVar(subset1)
X2<-colnames(subset[,nd])
```

We will subset our feature variables using the intersect function between X1 - no correlation and X2 - no variability:

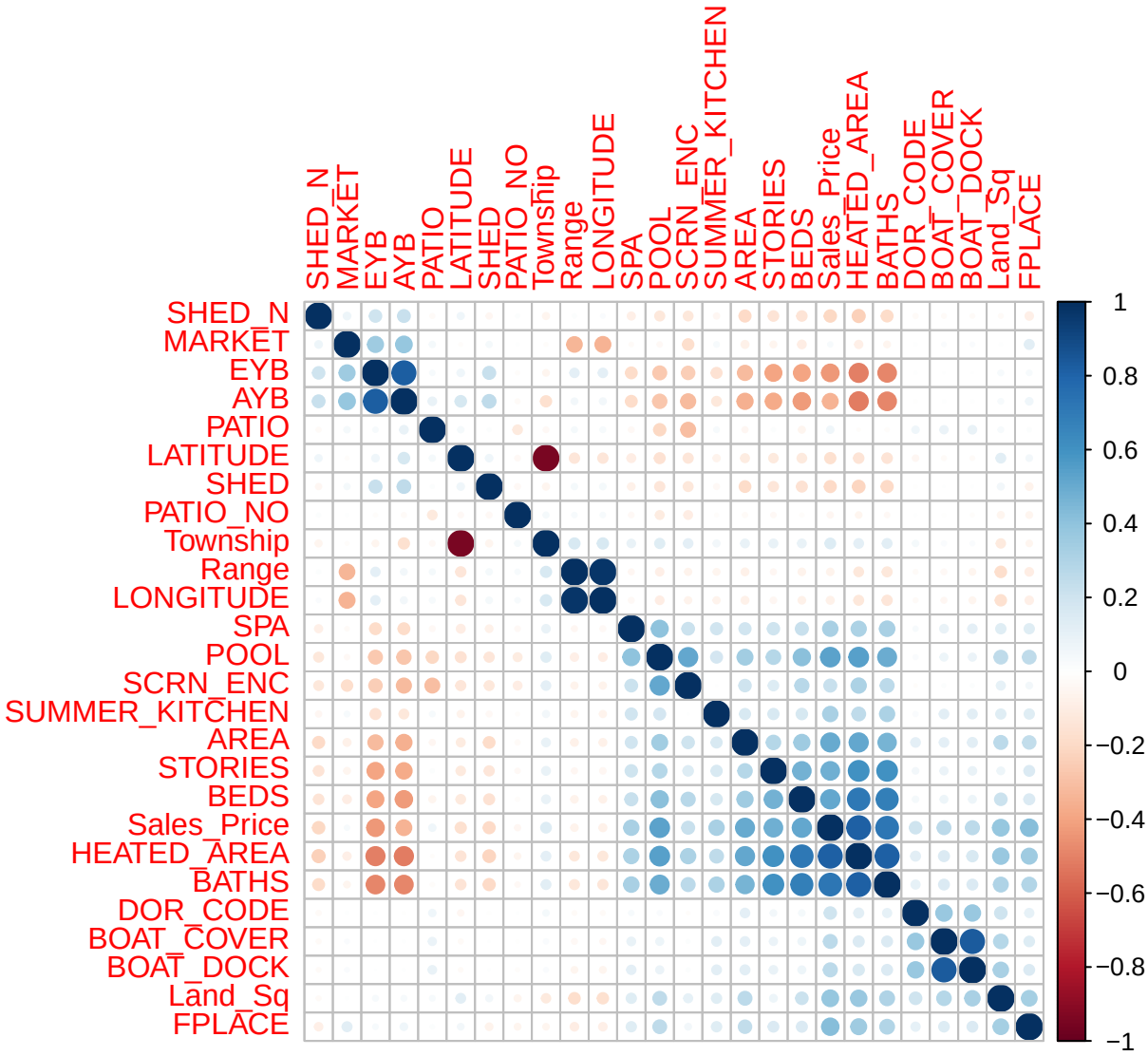
```
feature_remove<-intersect(X1,X2)
feature_remove
```

```
##  [1] "ACC_BLD"        "ACC_BLDG"       "BARN"           "BOAT_HOUSE"
##  [5] "CARPORT"        "COVER_ALUM"     "COVER_FG"       "COVER_STL"
##  [9] "COVER_WD"       "CT"             "ELEVATOR"       "FOUNTAIN"
## [13] "GAZEBO"         "GRNHS"          "HORSE_STAB"     "HORSE STABLE"
## [17] "MOBILE_HM"      "PARKING_SPACE"  "PATIO_WD"       "PAV_CON"
## [21] "PO_BLDG"        "POLE_BLDG"      "RESIDENTIAL_EL" "RM_ENCL"
## [25] "SAUNA"          "SEAWALL"        "SHADEHOUSE"     "SMALL_SHED"
## [29] "WALL_CB"        "WALL_DEC"       "WALL_NO"
```

```
sqtest<-sqtest[,-which(colnames(sqtest)%in%feature_remove)]
```

Let's now review how some of the remaining features correlated:

```
m<-cor(sqtest[,c(3,4,7,8,10:13,15,16,20:24,27:ncol(sqtest))])
corrplot::corrplot(m, method="circle", order="hclust")
```



To improve our model, it makes sense to remove *TOWNSHIP* and *RANGE* features since these are highly correlated with *LONGITUDE* and *LATITUDE*.

Also, we will remove the following features: *Sale_Date*, *Parcel*, *Situs_Address*, *City*, *NBHG_DESC*, and *ZipCode* as these don't add much information and will likely increase the complexity of our model. Additionally, features such as *SR* and *PPSF* will not be available in the future applications and will be removed from a model's input.

Therefore, our set is reduced to 26 variables and we are ready to begin constructing the ML algorithms.

ML ALGORITHMS

First, let's separate our outcome data into a vector y of Sales Prices and create a matrix x containing the features:

```
y<-data.matrix(sqtest[,2])
x<-data.matrix(sqtest[,-2])
```

Lets partition the data using createDataPartition function into 80/20 sets. While there are other ways to partition the data, we use the 80/20 rule or Pareto Principle, which states in general that in most cases, 80% of effects come from 20% of causes. Potentially, we could have partitioned using scaling (Guyon) law, which determines splits by how many unique features and complexity of these features in the data set with the goal to prevent overtraining.

```
set.seed(1)
index<-createDataPartition(y,times=1,p=0.8,list=FALSE)
train_x<-x[index,]
test_x<-x[-index,]
train_y<-y[index]
test_y<-y[-index]
```

It is problematic to fit the data into an ML algorithm that takes a variety of different ranges. For this reason, a feature-wise normalization is used. For each feature, the mean is subtracted and divided by the standard deviation, so the features are centered around 0 and have a unit standard deviation. The quantities used for normalization of the test data are computed on the train data only.

```
mean<-apply(train_x,2,mean)
std<-apply(train_x,2,sd)
train_x<-scale(train_x, center=mean,scale=std)
test_x<-scale(test_x, center=mean, scale=std)
```

There are a variety of loss functions available to measure the performance of ML algorithms. We will use MSE (mean squared error) as our loss function through this work. For results interpretation, we will also define a function for MAE (mean absolute error) as a relative metric of loss. One important distinction between MAE & MSE is that minimizing the squared error over a set of numbers results in finding its mean and minimizing the absolute error results in finding its median. This is the reason why MAE is robust to outliers whereas MSE is not. We will also define its reverse functions considering the original logarithmic transformations to evaluate end results:

```
MSE <- function(true_price, predicted_price){
  mean((true_price -  predicted_price)^2)
}

MAE<-function(true_price, predicted_price){
  mean(abs(true_price -  predicted_price))
}

MSEE<-function(true_price, predicted_price){
  mean((exp(true_price) -  exp(predicted_price))^2)
}
MAEE<-function(true_price, predicted_price){
  mean(abs(exp(true_price)-exp(predicted_price)))
}
```

Now, we are going to fit several different ML models:

Model 1: GLM

Generalized Linear Model (GLM), perhaps most commonly used model in data analysis, represents our outcome variable as a linear function of input features. To improve model's performance, we will define trControl function that controls the resampling scheme to 10-fold cross-validation:

```
Control<-trainControl(method="cv", number=10)
set.seed(1)
train_glm<-train(train_x,train_y, method="glm", trControl=Control)
glm_predict<-predict(train_glm, test_x)

results1<-data_frame(MODEL="GLM", MSE=MSE(test_y, glm_predict))
knitr::kable(results1, "simple")
```

MODEL	MSE
GLM	0.0495869

Model 2: LOESS

Loess model is a local polynomial regression that uses local weighted regression to fit a smooth curve through data points. We use 5-fold cross-validation for this model. In Loess, we can tune the parameters, namely span, using tuneGrid function. *Note: Tuning is omitted in the code below to improve the computational time.*

```
Control<-trainControl(method="cv", number=5)
set.seed(1)
#tuneGrid = expand.grid(span = seq(0.20, 0.50, len = 10), degree=2)
train_loess<-train(train_x,train_y, method="gamLoess", trControl=Control)
loess_predict<-predict(train_loess, test_x)

results2<-bind_rows(results1, data_frame(MODEL="LOESS",
                                         MSE=MSE(test_y, loess_predict)) )
knitr::kable(results2, "simple")
```

MODEL	MSE
GLM	0.0495869
LOESS	0.0347857

The MSE results have improved from 0.0495869 to 0.0347857.

Model 3: svmLinear

A support-vector machine (SVM) model classifies data by determining the optimal hyperplane that separates observations according to their class labels. While commonly used for classification problems, it can also be applied to regression tasks. Let's see how well it will perform on our data. We use 5-fold cross-validation without additional grid tuning, which substantially increases the model's training time.

```
Control<-trainControl(method="cv", number=5)
# grid <- expand.grid(span = seq(0.15, 0.65, len = 10), degree = 1) - optional tuning
set.seed(1)
train_svm<-train(train_x,train_y, method="svmLinear", trControl=Control, tuneLength = 8)
```

```
svm_predict<-predict(train_svm, test_x)
```

```
results3<-bind_rows(results2, data_frame(MODEL="SVM", MSE=MSE(test_y, svm_predict)) )
knitr::kable(results3, "simple")
```

MODEL	MSE
GLM	0.0495869
LOESS	0.0347857
SVM	0.0506544

The SVM model showed an increase in MSE when compared to GLM or Loess. The varImp function showed that *HEATED_AREA*, *BATHS*, *POOL* are among the top three variables, while *SHED*, *DOR_CODE*, and *SHED_N* being the least important.

```
# Variables importance:
varImp(train_svm)
```

```
## loess r-squared variable importance
##
##   only 20 most important variables shown (out of 25)
##
##                Overall
## HEATED_AREA    100.000
## BATHS           79.195
## POOL            42.719
## BEDS            39.844
## AREA            38.916
## STORIES         34.507
## EYB             28.068
## FPLACE          26.132
## Land_Sq         25.909
## AYB             18.089
## LONGITUDE       17.119
## SUMMER_KITCHEN  15.692
## SPA             15.566
## BOAT_DOCK       10.335
## BOAT_COVER       9.338
## LATITUDE         8.070
## SCRN_ENC         6.938
## SHED_N           6.382
## DOR_CODE         5.810
## SHED             5.505
```

Let's fit K Nearest Neighbors (KNN) model next.

Model 4: KNN

K nearest neighbors (KNN) is an algorithm that classifies new cases based on a similarity (distance). It can be used to solve both classification and regression problems. We use 5-fold cross-validation and tune with a sequence of k parameters.

```
tuningknn<-data.frame(k=seq(3,13,2))
Control<-trainControl(method="cv", number=5)
```

```
set.seed(1)
train_knn<-train(train_x, train_y, method="knn",tuneGrid=tuningknn,
                 trControl=Control,tuneLength = 10)
knn_predict<-predict(train_knn, test_x)

results4<-bind_rows(results3, data_frame(MODEL="KNN", MSE=MSE(test_y, knn_predict)) )
knitr::kable(results4, "simple")
```

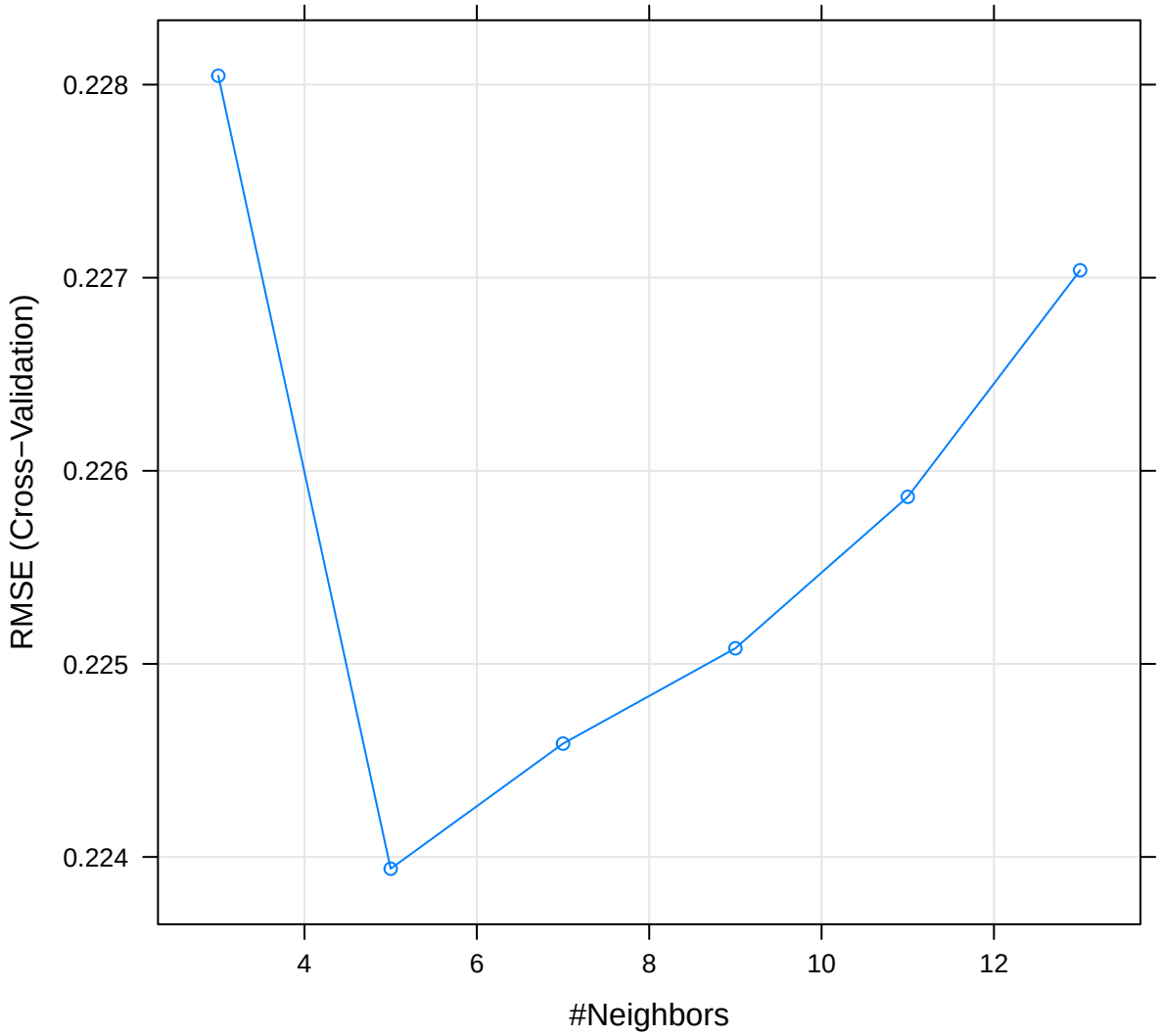
MODEL	MSE
GLM	0.0495869
LOESS	0.0347857
SVM	0.0506544
KNN	0.0470616

The MSE result was 0.0470616, which makes it the second best performing model so far. We can also check for variable importance and determine the best K parameter:

```
varImp(train_knn)
```

```
## loess r-squared variable importance
##
##   only 20 most important variables shown (out of 25)
##
##                Overall
## HEATED_AREA    100.000
## BATHS           79.195
## POOL            42.719
## BEDS            39.844
## AREA            38.916
## STORIES         34.507
## EYB             28.068
## FPLACE          26.132
## Land_Sq         25.909
## AYB             18.089
## LONGITUDE       17.119
## SUMMER_KITCHEN  15.692
## SPA             15.566
## BOAT_DOCK       10.335
## BOAT_COVER       9.338
## LATITUDE         8.070
## SCRN_ENC         6.938
## SHED_N           6.382
## DOR_CODE         5.810
## SHED             5.505
```

```
plot(train_knn)
```




```
train_knn$bestTune
```

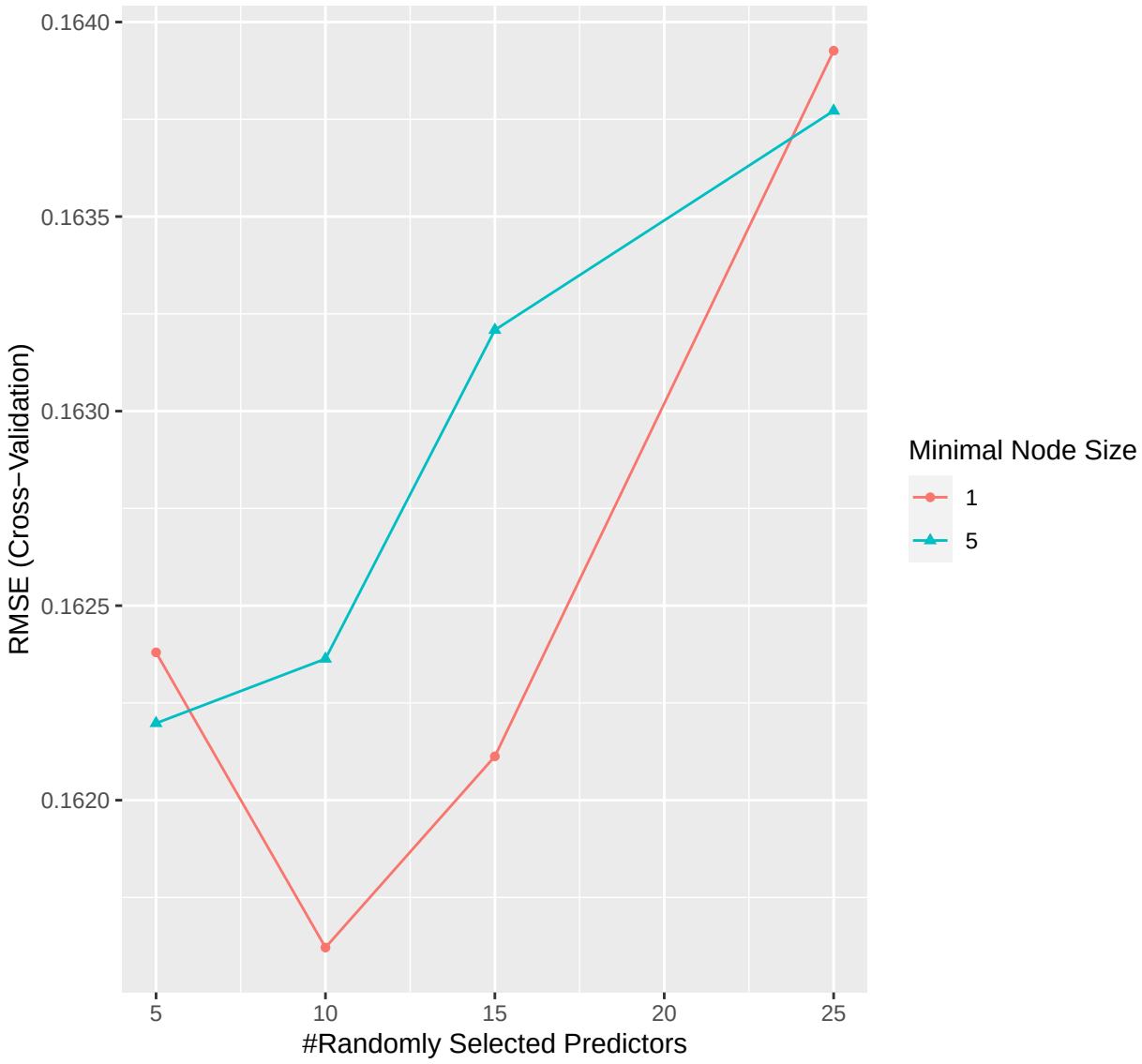
```
##   k
## 2 5
```

Similarly to the svmLinear model, *HEATED_AREA* is the important feature followed by *BATHS*. By plotting the model, we see that the best number of neighbors parameter is 5.

Model 5: Random Forest with Rborist

Random Forest (RF) operates by constructing a multitude of decision trees and outputting mean prediction of the trees. We use 5-fold cross-validation for the model. Let's first run a model on 50 trees to define the best tuning parameters for the model:

```
control<-trainControl(method="cv", number = 5, p=0.8)
grid<-expand.grid(minNode=c(1,5),predFixed=c(5,10,15,25))
set.seed(1)
train_rf<-train(train_x, train_y, method="Rborist", nTree=50,trControl=control,
                tuneGrid = grid, nSamp=5000)
# Let's plot the result
ggplot(train_rf)
```




```
# Let's see the best tunning parameters
train_rf$bestTune
```

```
##   predFixed minNode
## 2        10       1
```

Since predFixed=10 and minNode=1 are the best parameters, we apply 500 trees and the Rborist model, which is a faster implementation of Random Forest:

```
grid<-expand.grid(minNode=1, predFixed=10)
set.seed(1)
train_rf<-Rborist(train_x, train_y, nTree=500, tuneGrid=grid) #- (less then 1 min)

rf_predict<-predict(train_rf, test_x)%>%.$yPred

results5<-bind_rows(results4, data_frame(MODEL="RF", MSE=MSE(test_y, rf_predict)) )
knitr::kable(results5, "simple")
```

MODEL	MSE
GLM	0.0495869
LOESS	0.0347857
SVM	0.0506544
KNN	0.0470616
RF	0.0252417

The Random Forrest model yields the lowest MSE of 0.252417.

Model 6: GBM

Gradient Boosting Model (GBM) is considered a gradient descent algorithm. Whereas random forests build an ensemble of deep independent trees, GBMs build an ensemble of shallow and weak successive trees with each tree learning and improving on the previous. We use 1000 trees to fit the model. GBMs often require many trees; however, unlike random forests, GBMs can overfit so the goal is to find the optimal number of trees that minimizes the loss function of interest with cross-validation.

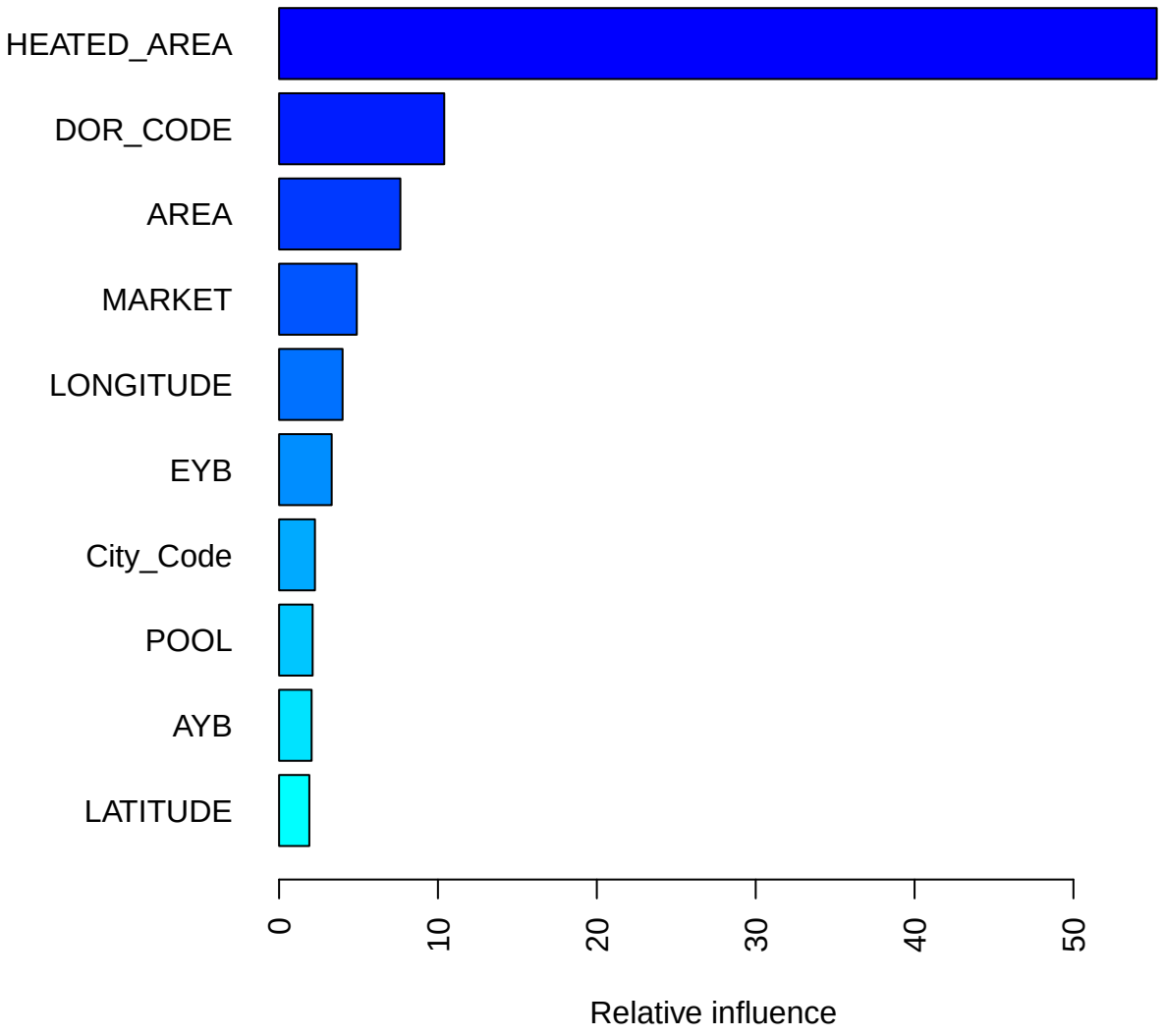
Please note the parameters below were tuned using (1) shrinkage = c(.01, .1, .3) - learning rate that controls how quickly the algorithm proceeds down the gradient descent; (2) interaction.depth = c(1, 3, 5) - the number of splits in each tree, which controls the complexity of the boosted ensemble; (3) n.minobsinnode = c(5, 10, 15) - the minimum number of observations allowed in the trees terminal nodes. The best tuning parameters are picked in the code below.

```
TrainControl <- trainControl( method = "repeatedcv", number = 5, repeats = 4)
tunninggbm<-expand.grid(interaction.depth = 5, n.trees = 1000, shrinkage = .1,
                        n.minobsinnode = 5)
set.seed(1)
train_gbm<-train(train_x, train_y, method="gbm", tuneGrid=tunninggbm,
                 trControl=TrainControl, verbose=FALSE)
gbm_predict<-predict(train_gbm, test_x)

results6<-bind_rows(results5, data_frame(MODEL="GBM", MSE=MSE(test_y, gbm_predict)) )
knitr::kable(results6, "simple")
```

MODEL	MSE
GLM	0.0495869
LOESS	0.0347857
SVM	0.0506544
KNN	0.0470616
RF	0.0252417
GBM	0.0192465

By plotting the most influential parameters, we see that *HEATED_AREA* appears to have the most relative influence followed by *DOR_CODE* and *AREA*. On the contrary, *SPA*, *SHED_N*, *PATIO_NO* are the bottom three that can potentially be removed from the model.

```
par(mar = c(5, 8, 1, 1))
summary(
  train_gbm,
  cBars = 10,
  las = 2)
```




```
##                          var     rel.inf
## HEATED_AREA       HEATED_AREA 55.24145089
## DOR_CODE             DOR_CODE 10.39614707
## AREA                     AREA  7.63295220
## MARKET                 MARKET  4.89207963
## LONGITUDE           LONGITUDE  3.99652915
## EYB                       EYB  3.31129841
## City_Code           City_Code  2.25658618
## POOL                     POOL  2.11188702
## AYB                       AYB  2.04807405
## LATITUDE             LATITUDE  1.90276892
## Land_Sq               Land_Sq  1.70685921
## BATHS                   BATHS  1.44749397
## FPLACE                 FPLACE  0.78923922
## BOAT_DOCK           BOAT_DOCK  0.49154311
## EXT_W                   EXT_W  0.37473452
## BEDS                     BEDS  0.34176574
## BOAT_COVER         BOAT_COVER  0.29602053
## PATIO                   PATIO  0.20116767
## SUMMER_KITCHEN SUMMER_KITCHEN  0.20058856
```

```
## STORIES               STORIES  0.10440179
## SHED                     SHED  0.09775366
## SCRN_ENC             SCRN_ENC  0.08782589
## SPA                       SPA  0.02926075
## SHED_N                 SHED_N  0.02412780
## PATIO_NO             PATIO_NO  0.01744405
```

The GBM model has the lowest MSE out of 6 models of 0.0192465.

Let's now compile the models in the ensemble with equal weight for each model.

Ensemble 1:

```
ensemble_predict1<-apply(cbind(glm_predict, loess_predict, svm_predict,
                               knn_predict, rf_predict, gbm_predict),1,mean)
results7<-bind_rows(results6, data_frame(MODEL="ENS1",
                                         MSE=MSE(test_y, ensemble_predict1)) )
knitr::kable(results7, "simple")
```

MODEL	MSE
GLM	0.0495869
LOESS	0.0347857
SVM	0.0506544
KNN	0.0470616
RF	0.0252417
GBM	0.0192465
ENS1	0.0279754

The MSE increased substantially from the GBM's lowest result.

Let's create a second ensemble that uses two best performing models: Random Forest and GBM.

Ensemble 2:

Let's begin with assigning equal weights to each model:

```
ensemble_predict2<-apply(cbind(rf_predict, gbm_predict),1,mean)
MSE(test_y,ensemble_predict2)
```

```
## [1] 0.02046332
```

The MSE has improved. But, can we do better knowing that GBM performed stronger than RF? Let's try assigning different weights to each model:

```
MSE_test<-NULL
for (i in 1:9){
  ensemble_predict2<-cbind(rf_predict,gbm_predict)%>%data.frame()%>%
    mutate(weighted=rf_predict*i/10+gbm_predict*(1-i/10))%>%.$weighted
  MSE_test=rbind(MSE_test,MSE(test_y, ensemble_predict2))
}
MSE_test[which.min(MSE_test),]
```

```
## [1] 0.0192049
```

```
which.min(MSE_test)
```

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

The best result was achieved with i equal to 1. Let's update the ensemble 2 model:

```
ensemble_predict2<-cbind(rf_predict,gbm_predict)%>%data.frame()%>%
  mutate(weighted=rf_predict*(which.min(MSE_test)/10)+gbm_predict*
           (1-which.min(MSE_test)/10))%>%.$weighted
results8<-bind_rows(results7, data_frame(MODEL="ENS2",
                                          MSE=MSE(test_y, ensemble_predict2)) )
knitr::kable(results8, "simple")
```

MODEL	MSE
GLM	0.0495869
LOESS	0.0347857
SVM	0.0506544
KNN	0.0470616
RF	0.0252417
GBM	0.0192465
ENS1	0.0279754
ENS2	0.0192049

So far, this is the best model that yields MSE of 0.0192049. Let's see if we can take it a step further and improve the MSE parameter by running an experiment with a deep learning model.

Model: Deep Learning

Deep learning in nature is a subset of machine learning that puts an emphasis on models called neural networks, structured in literal layers stacked on top of each other.

Using the powerful *Keras* package, we create a network with five dense layers with the following units: 256, 1024, 512, 256, 1 (output layer) and a dropout rate of 0.1 that helps to mitigate overfitting. The network ends with a single unit to predict a single continuous value of Sales Prices.

The process begins with building a function for a model. We compile a model and define an optimizer ("adam"), a loss function ("mse"), and a metric to monitor ("mae").

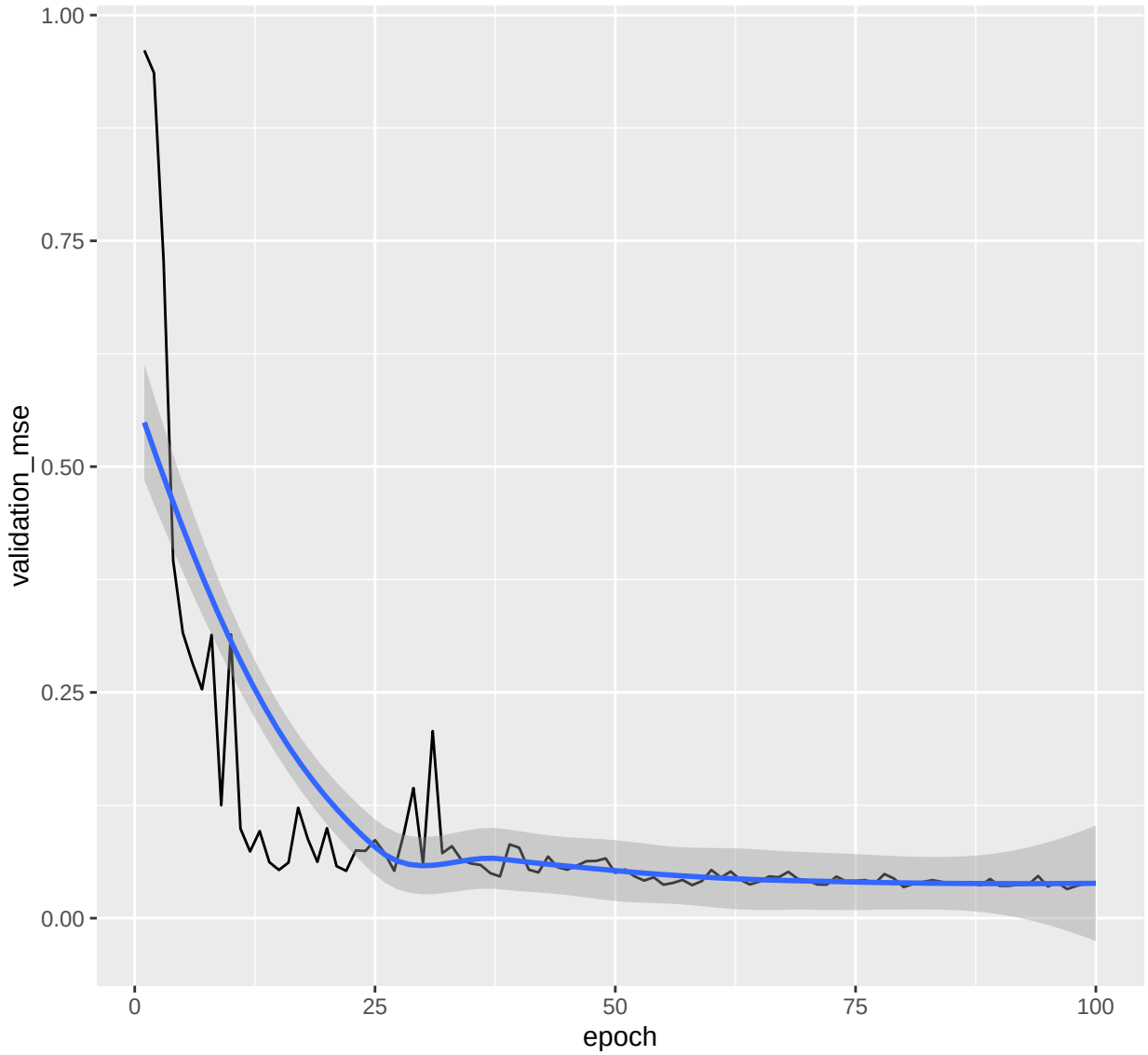
```
build_model<-function(){
  model <- keras_model_sequential() %>%
    layer_dense(units = 256, activation = "relu",input_shape = dim(train_x)[[2]]) %>%
    layer_dropout(rate=0.1)%>%layer_dense(units=1024, activation = "relu")%>%
    layer_dropout(rate=0.1)%>%
    layer_dense(units=512, activation="relu")%>%layer_dropout(rate=0.1)%>%
    layer_dense(units = 256, activation = "relu")%>%layer_dropout(rate=0.1)%>%
    layer_dense(units = 1)

  model %>% compile(optimizer = "adam",loss = "mse",metrics = c("mae"))
}
```

To evaluate our network and adjust the parameters, we use 4-fold validation which consists of splitting the available data into 4 partitions and training on each partition split while evaluating on the remaining partitions.

```
k_clear_session()
k<-4
indices<-sample(1:nrow(train_x))
folds<-cut(1:length(indices), breaks=k, labels=FALSE)
num_epochs<-100
all_scores<-NULL

for (i in 1:k){
  cat("processing fold #", i, "\n")
  #prepare the validation data from partition #k
  val_indices<-which(folds==i, arr.ind=TRUE)
  val_data<-train_x[val_indices,]
  val_targets<-train_y[val_indices]
  #prepare the training data from all other partitions
  partial_train_x<-train_x[-val_indices,]
  partial_train_y<-train_y[-val_indices]
  #build the model already compiled above
  model<-build_model()
  #train the model in silent mode (verbose=0)
  history<-model%>%fit(partial_train_x,partial_train_y,validation_data=
                         list(val_data, val_targets),epochs=num_epochs,
                       batch_size=20, verbose=0)
  mse_history<-history$metrics$val_loss
  all_scores<-rbind(all_scores,mse_history)

}
```

```
## processing fold # 1
## processing fold # 2
## processing fold # 3
## processing fold # 4
```

```
# Now, lets compute the average of the MSE scores for all folds:
average_mse_history<-data.frame(epoch=seq(1:ncol(all_scores)),
                                validation_mse=apply(all_scores,2,mean))
# Lets plot it:
ggplot(average_mse_history, aes(x=epoch, y=validation_mse))+geom_line()+geom_smooth()
```



According to the plot, MSE stops improving after about 70 epochs. Past that, we start overfilling. While not included in this work, the size and number of hidden layers can also be tuned to improve the model's performance.

Now let's train the optimized deep learning model with the best parameters (in this case, the number of epochs is 70). Please note that when fitting on GPU, the model yields a better result. The result discussed in this study was produced using Tesla V100 GPU on ubuntu-18.04-LTS server. GPU use is recommended.

```
model<-build_model()
model%>%fit(train_x,train_y,epochs=70, batch_size=20, verbose=0)
```

Obtaining a consistent and reproducible model that yields the same MSE is challenging with *Keras.* In order to maintain a consistent result, the original model produced ("my_dl_model") is made available in the repository.

```
model%>%evaluate(test_x, test_y)
```

```
##               loss mean_absolute_error
##         0.02793818          0.11761925
```

```
dl_predict<-model%>%predict(test_x)
# MSE is generated by the deep learning model is:
MSE(test_y,dl_predict)
```

```
## [1] 0.02793818
```

The MSE result is relatively good, but it is not ranked first.

Let's compile a third ensemble including the deep learning model.

Ensemble 3:

First, we assign the same weights for each model.

```
ensemble_predict3<-apply(cbind( rf_predict, gbm_predict, dl_predict),1,mean)
MSE(test_y, ensemble_predict3)
```

```
## [1] 0.02022693
```

The result is significantly better. Let's try leveraging weights with emphasis on GBM:

```
# Let's create a sample matrix of weights and apply to predictions of each model:
s<-t(matrix(c(0.1, 0.8, 0.1, 0.2, 0.7, 0.1, 0.1, 0.7, 0.2, 0.1, 0.6, 0.3, 0.2, 0.6,
              0.2, 0.3, 0.6, 0.1), nrow = 3, ncol = 6))
s
```

```
##      [,1] [,2] [,3]
## [1,]  0.1  0.8  0.1
## [2,]  0.2  0.7  0.1
## [3,]  0.1  0.7  0.2
## [4,]  0.1  0.6  0.3
## [5,]  0.2  0.6  0.2
## [6,]  0.3  0.6  0.1
```

```
MSE_test<-NULL
for (i in 1:6){
  ensemble_predict3<-cbind(rf_predict,gbm_predict,dl_predict)%>%
    data.frame()%>%mutate(weighted=rf_predict*s[i,1]+gbm_predict*s[i,2]+
                            dl_predict*s[i,3])%>%.$weighted
  MSE_test=rbind(MSE_test,MSE(test_y, ensemble_predict3))
}
MSE_test
```

```
##            [,1]
## [1,] 0.01892670
## [2,] 0.01908555
## [3,] 0.01891634
## [4,] 0.01917384
## [5,] 0.01913314
## [6,] 0.01938687
```

We pick the weights attributed to the lowest MSE and apply to the ensemble 3 model:

```
s<-s[which.min(MSE_test),]
ensemble_predict3<-apply(cbind( rf_predict*s[1], gbm_predict*s[2],
                                dl_predict*s[3]),1,sum)

results9<-bind_rows(results8, data_frame(MODEL="ENS3",
```

```
                                    MSE=MSE(test_y, ensemble_predict3)) )
knitr::kable(results9, "simple")
```

MODEL	MSE
GLM	0.0495869
LOESS	0.0347857
SVM	0.0506544
KNN	0.0470616
RF	0.0252417
GBM	0.0192465
ENS1	0.0279754
ENS2	0.0192049
ENS3	0.0189163

The weighted ensemble 3, has the best MSE among all the models.

So what does it mean? Let's calculate relative measures using pre-logarithmic RMSE and MAE to understand the model's performance:

```
sqrt(MSEE(test_y,ensemble_predict3))
```

```
## [1] 54235.41
```

```
MAEE(test_y,ensemble_predict3)
```

```
## [1] 33230.26
```

That means that the model's results deviate by $54,235 on average with $33,230 being the median deviation. Considering that the Mean Sales Price is $355,988 and the Median is $280,000, there is still room for improvement.

INDUSTRY STANDARDS

Now, let's evaluate if this model passes the assessment industry ratio standards. First, we will apply exponential transformation to the predicted and actual results; then, apply 85% state requirement to the Predicted (Assessment) values before calculating the hypothetical Sales Ratio:

```
pred<-cbind(Assessed=exp(ensemble_predict3)*.85,Sales_Price=exp(test_y))%>%data.frame()%>%
  mutate(SR_pred=Assessed/Sales_Price)
head(pred)
```

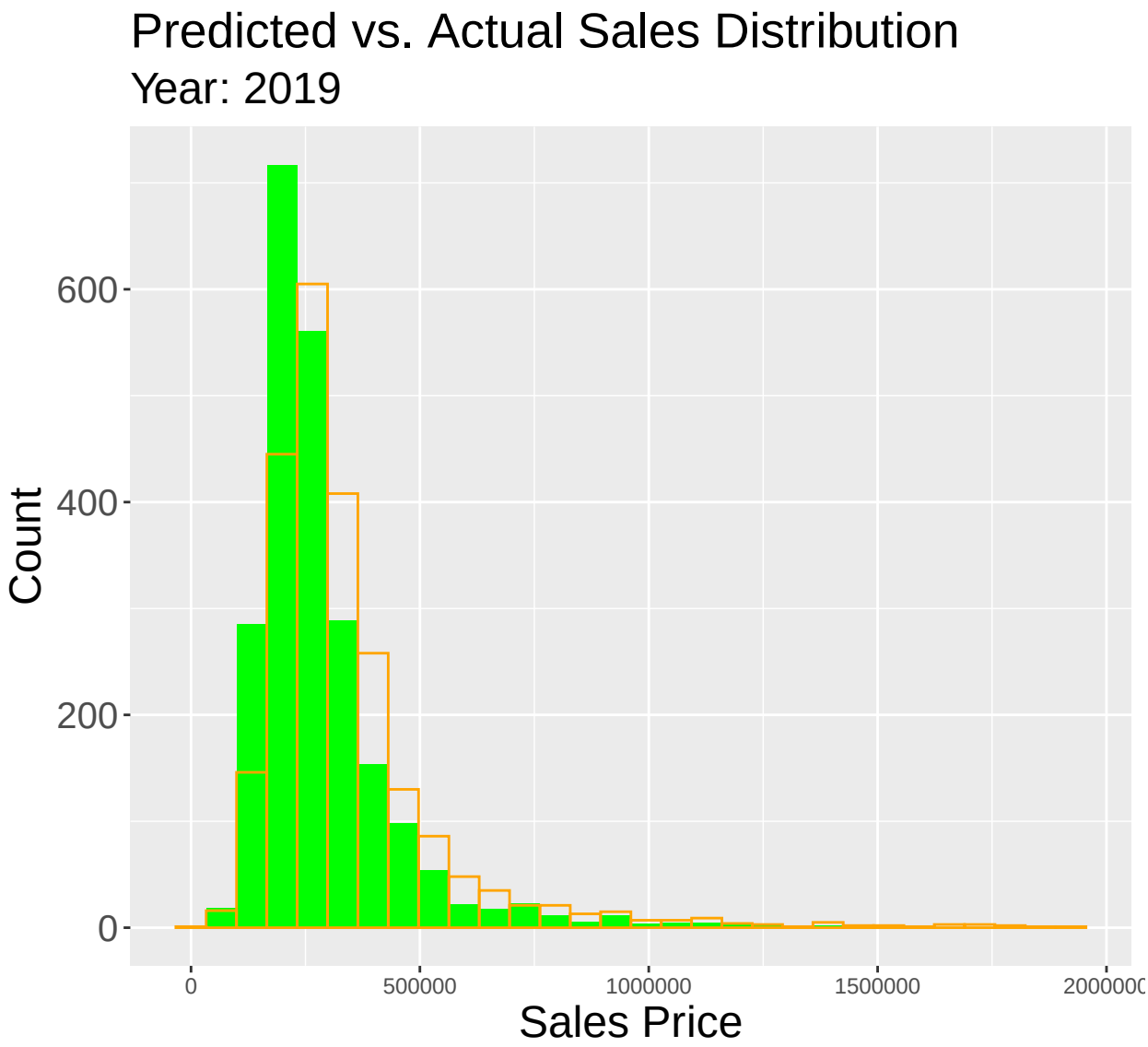
```
##   Assessed Sales_Price   SR_pred
## 1 323590.2      360000 0.8988616
## 2 199084.4      238900 0.8333378
## 3 147604.6      180000 0.8200254
## 4 649488.8      567000 1.1454829
## 5 829677.9      930000 0.8921268
## 6 248980.9      260000 0.9576189
```

Let's plot it and see the distributions of Predicted (Assessed) vs. Actual (Test Data) Sales Prices:

```
pred%>%ggplot()+geom_histogram(aes(Assessed), fill="green")+
  geom_histogram(aes(Sales_Price), colour="orange", fill=NA)+
  theme(plot.title = element_text(size=20),plot.subtitle = element_text(size=18),
        legend.title = element_text(size=18), legend.text = element_text(size=14),
```

```
       axis.title = element_text(size=18), axis.text.y = element_text(size=15))+
  labs(title = 'Predicted vs. Actual Sales Distribution',
       subtitle= 'Year: 2019', x = 'Sales Price', y = 'Count')
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



We can see from the plot, the distribution is fairly close to the original Sales Prices distribution considering the 15% adjustment of the Predicted (Assessed) values identified in green.

Let's calculate the required statistics:

```
stat<-pred%>%summarise(MEAN=mean(SR_pred), MEDIAN=median(SR_pred))

# 95% confidence interval for the mean
sd<-sqrt(sum((pred$SR_pred-stat$MEAN)^2)/(length(pred$SR_pred)-1))
ci<-(mean(pred$SR_pred)+c(-qnorm(0.975),qnorm(0.975))*sd)*100

COD<-mean(abs(pred$SR_pred-stat$MEDIAN))/stat$MEDIAN*100
```

```
PRD<-stat$MEAN/(sum(pred$Assessed)/sum(pred$Sales_Price))

# Table with data

final<-knitr::kable(cbind(MEDIAN=stat$MEDIAN*100, MEAN=stat$MEAN*100,
                          CONF_LOW=ci[1], CONF_HIGH=ci[2],COD=COD, PRD=PRD),
                    "simple", caption = "Final Model Ratios")
final
```

Table 11: Final Model Ratios

MEDIAN	MEAN	CONF_LOW	CONF_HIGH	COD	PRD
84.88738	86.08646	61.30132	110.8716	9.919129	1.019258

Now, we compare the performance of the final model ratios to the original data ratios.

```
original
```

Table 12: Original CAMA Ratios

MEAN	MEDIAN	CONF_LOW	CONF_HIGH	COD	PRD
81.29171	81.14531	61.78642	100.797	9.518778	0.9939223

Notably, all ratios have either improved or within acceptable limits. The MEDIAN and MEAN of the final model are closer to 85%. The COD and PRD of the data are within acceptable industry ranges. 95% of the observations are within the 61 to 110 range, which is a bit above the original values due to sampling. We can conclude that the model passes the industry ratio standards test.

CONCLUSION AND FUTURE WORK

The application of Machine Learning algorithms for predicting property values is reasonable in a mass appraisal setting. It offers flexibility and scalability necessary when valuing a large number of properties on an annual basis. The limitations of this methodology in determining property values include its complexity in application considering required data pre-processing and feature engineering, model tuning and configuring. While Ensemble models offer better performance when compared to stand-alone ML models, they could be difficult to explain.

The best model produced through this work is a weighted ensemble model based on Random Forest, Gradient Boosting, and Deep Learning (DL). This model yields the lowest MSE of 0.0189163 with results deviating by $54,235 on average and $33,230 using mean. There is a wide array of opportunities to continue tuning parameters and changing the structure of the Deep Learning model to improve the ensemble performance. The biggest challenge using the DL model is its lack of interpretability for assessors working in the highly regulated industry.

The future work includes feature engineering - the process of evaluating and adjusting variables inputted into a model. This may include adding new features that can explain the variability of an outcome and removing those that add noise to the model. For example, adding features like area school grades and average household characteristics can positively impact the model's performance. Also, using a prior year assessment data of properties will help to better explain the outcome, especially if used on a large data set to predict the values of properties that do not have any recent sales. Further filtering and transforming the data, removing the outliers may have a positive effect on the MSE. Another area of future work is parameter tuning of the ML models including configuring the number of nodes and layers in the Deep Learning model. Furthermore, to accommodate the scaling of the Deep Learning model and adding more data to process, such as aerial imagery of properties, warrants investment in computational resources such as GPU clusters. Finally out of a library of hundreds of available ML models, we fitted 6 stand-alone and created 3 ensemble models. It is prudent to further explore fitting additional models or creating different ensembles to yield the best model's performance.