Capstone Choose Your Own Project: Predict House Price

Using Linear Regression and Random Forest approach for Regression

Saúl Santillán Gutiérrez

October 25, 2022

Contents

General Instructions by HarvardX's Team

For this project, you will be applying machine learning techniques that go beyond standard linear regression. You will have the opportunity to use a publicly available dataset to solve the problem of your choice. **You are strongly discouraged from using well-known datasets**, particularly ones that **have been used as examples in previous courses** or are similar to them (such as **the iris**, **titanic**, **mnist**, or **movielens datasets**, **among others**) - this is your opportunity to branch out and explore some new data! The UCI Machine Learning Repository and Kaggle are good places to seek out a dataset. Kaggle also maintains a curated list of datasets that are cleaned and ready for machine learning analyses. Your dataset must be automatically downloaded in your code or included with your submission. **You may not submit the same project for both the MovieLens and Choose Your Own project submissions**, and your **Choose Your Own project submission may not simply be a variation of your MovieLens project**.

The ability to clearly communicate the process and insights gained from an analysis is an important skill for data scientists. You will submit a report that documents your analysis and presents your findings, with supporting statistics and figures. The report must be written in English and uploaded as both a PDF document and an Rmd file. Although the exact format is up to you, **the report should include the following at a minimum:**

- An **introduction/overview/executive summary section** that describes the dataset and variables, and summarizes the goal of the project and key steps that were performed.
- A **methods/analysis section** that explains the process and techniques used, including data cleaning, data exploration and visualization, any insights gained, and your modeling approach. **At least two different models or algorithms must be used, with at least one being more advanced than linear or logistic regression for prediction problems**.
- A **results section** that presents the modeling results and discusses the model performance.
- A **conclusion section** that gives a brief summary of the report, its potential impact, its limitations, and future work.

The submission for **the Choose Your Own project** will be **three files**: **a report** in the form of both a **PDF document** and **Rmd file** and the **R script that performs your machine learning task**. You must also provide access to **your dataset**, either through **automatic download in your script** or **inclusion in a GitHub repository**.

We recommend submitting a link to a GitHub repository with these three files and your dataset. Your grade for the project will be based on **your report** and **your script**.

Preliminary

The present report belongs to the **Choose Your Own project of the HarvardX's Data Science Professional Certificate** and it is composed of 4 chapters and by an extra section called Appendix, which are described as follows: Chapter 1 Overview describes the dataset, summarizes the goal of the project and key steps that were performed. In Chapter 2 Methods and Analysis explains the process and techniques used, for example, data science project preparation, data exploration and visualization, preparation and cleaning of the datasets, preparation of the training and testing datasets. Also, the insights gained during the course, and the modeling approach used. Chapter 3 Results presents the modeling results and discusses the model performance. In Chapter 4 Conclusion gives a brief summary of the report, its limitations and future work. And finally, in the extra section **Appendix**, that is divided into two parts, shows the information about the computer equipment used, its hardware specs, `R` session, the Operating Systems (OS) where this code was made and tested, and the loaded packages.

Chapter 1

Overview

Currently, the house prices have increased too much around the world, but especially during and after the Covid-19 pandemic, because with the pandemic our houses have become offices and schools at the same time, and in some cases we required one or more additional spaces to our houses, to carry out all those tasks, that originated the acquisition of a new house. It should be noted that, in addition, the house prices are among the main indicators that a country has a healthy economy along with the general condition of the market. From there, a problem arises, can we predict the price of a house according to our needs?

The answer to that problem is yes, and we can solve it using regression analysis. But, **what is a regression?** is *"a statistical method used to describe the nature of the relationship between variables, that is, positive or negative, linear or nonlinear"* defined by (Bluman, 2011, p. 534). And in a special way to solve this kind of problem, **multiple regression is used**, which consists as Bluman points out *"in a multiple relationship, called multiple regression, two or more independent variables are used to predict one dependent variable"* (2011, p. 535). It is necessary to mention that an **independent variable** can be called an **explanatory variable** or a **predictor variable**, and a **dependent variable** can be called a **response variable** or **outcome variable**.

On the other hand, we know that there are three types of regression most used: **Linear regression**, **Logistic regression** and **Polynomial regression**. Returning to our problem, here, the house prices is going to be our **response variable** and it is a **continuous variable**, for this type of problem we choose **linear regression**, because this regression model predicts a continuous dependent variable from two or more independent variables.

Additionally, we can also solve this type of problem by applying the random forest algorithm, because we know that we can use it or employ it either for classification or regression. But, **what is random forest** or what does it consist of? Bruce et al. give us their definition *"is based on applying bagging to decision trees, with one important extension: in addition to sampling the records, the algorithm also samples the variables"*, and they add that *"the random forest algorithm adds two more steps: the bagging […], and the bootstrap sampling of variables at each split"* (2020, p. 261).

For all the reasons stated above, in this report we predict the house price using Linear Regression and Random Forest approach for Regression in the **House Price prediction** dataset

from Kaggle applying our knowledge acquired in the lessons of the **HarvardX's Data Science Professional Certificate**.

1.1 About House Price Prediction Dataset

"Inside Kaggle you will find all the code and data you need to do your data science work. Use over 50,000 public datasets and 400,000 public notebooks to conquer any analysis in no time", that is what Kaggle says. But, **what is Kaggle?** this is how DataCamp explains it on its blog *"Kaggle is an online community platform for data scientists and machine learning enthusiasts. Kaggle allows users to collaborate with other users, find and publish datasets, use GPU integrated notebooks, and compete with other data scientists to solve data science challenges. The aim of this online platform (founded in 2010 by Anthony Goldbloom and Jeremy Howard and acquired by Google in 2017) is to help professionals and learners reach their goals in their data science journey with the powerful tools and resources it provides. As of today (2021), there are over 8 million registered users on Kaggle"* (2022).

The dataset of our interest can be found in this site House Price prediction. To get it, we need to be registered in Kaggle or if we are already registered we just need to sign in, click on the `Download` button and save the file. The file is save it with the extension `.xls`, for that, we must to change it to `.csv`. The dataset contains 4,600 prices of houses from 44 cities of the US country with different characteristics. This dataset was made by (Shree, 2018).

1.2 The Goal of the Project

In a linear regression model (linear model) the author Singh mentions that *"one of the most important tasks is to select an appropriate evaluation metric"* (2019). Definitely, this part of the process of building a linear model must be considered the most essential because there are various types of **evaluation metrics**, also known as **loss functions**, and we have to choose the most appropriate for each project.

The main functions used in linear models are: **MAE (Mean Absolute Error)**, **MSE (Mean Squared Error)** and **RMSE (Root Mean squared Error)**. Some authors also mention RMSLE (Root Mean squared Log Error), R-squared or Coefficient of determination and Adjusted R-squared.

MAE (Mean Absolute Error)
It symbolizes the average of the absolute difference between the predicted and actual (original) values in the dataset. That is to say, it measures the average of the residuals in the dataset and its formula is:

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|\hat{y}_i - y_i|$$

where N is the number of observations, $\hat{y}_i$ is the predicted value and y_i is the actual value.

IMPORTANT: MAE is **NOT sensitive** to **outliers**.

MSE (Mean Squared Error)
It symbolizes the average of the squared difference between the predicted and actual (original) values in the dataset. Namely, it measures the variance of the residuals in the dataset and is given by the formula:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

where N is the number of observations, $\hat{y}_i$ is the predicted value and y_i is the actual value.

IMPORTANT: MSE is **sensitive** to **outliers**, so be careful when the dataset has them because they can distort results.

RMSE (Root Mean Squared Error)
It is the square root of Mean Squared error. Namely, it measures the standard deviation of the residuals in the dataset and is represented by the formula:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}$$

where N is the number of observations, $\hat{y}_i$ is the predicted value and y_i is the actual value.

IMPORTANT: RMSE is **sensitive** to **outliers**, so be careful when the dataset has them because they can distort results.

For this project, we are going to focus on the **RMSE loss function**, because **RMSE** is *"a weighted measure of model accuracy given on the same scale as the prediction target"* that is how (Allwright, 2022) mentions it. In other words, the metric it produces is in terms of measurement units or the units being predicted, beside that, it is the typical evaluation metric preferred in the linear regression models using the `lm()` function to build this kind of models.

Remember the **RMSE** produces the metric or value in terms of measurement units, for that, sometimes we need to **normalize the value** of the **RMSE**, depending on the type of problem we want to solve, using one of the formulas suggested by (Otto, 2019), like this:

$$\text{NRMSE} = \frac{\text{RMSE}}{(y_{max} - y_{min})}$$

where y represents the actual observations and $(y_{max} - y_{min})$ is the difference between the maximum and minimum actual values (observations).

Our goal is to choose the best model that obtains **the lowest RMSE value** of four models to be evaluated. We fit two models, one performs a linear regression with the `lm()` function and the

other ones an random forest approach for regression with the `randomForest` function from the `randomForest` package, in a dataset that has outliers. And fit other two models, performing the same algorithms used in the first two models, in a dataset without them. Using the knowledge, concepts and techniques acquired in the lessons taught by the teacher (Irizarry, 2022) in the courses of **HarvardX's Data Science Professional Certificate**.

1.3 Key Steps Performed

A lot of data scientist follow a data science process or workflow, *"which is a structured framework used to complete a data science project"* this is how (Gupta, 2022) defines it and adds that *"there are many different frameworks, and some are better for business use cases, while others work best for research use cases"*.

The most popular and widely used data science process frameworks are **CRISP-DM (Cross Industry Standard Process for Data Mining)** and **OSEMN**, is made up of the capital letters of the following words: **O**btain data, **S**crub data, **E**xplore data, **M**odel data and i**N**terpret results, frameworks that is how (Gupta, 2022) and (Nantasenamat, 2022) point it out. In addition, **OSEMN** can be used on research, as (Gupta, 2022) states *"it's ideal for projects focusing on exploratory research, and is often used by research institutions and public health organizations"*. Another important point when selecting a framework is what was cited by (Nantasenamat, 2022) *"it should be noted that the flow among these processes is not linear and that in practice the flow can be non-linear and can re-iterate until satisfactory condition is met"* and that is fulfilled by **OSEMN**.

For the above, we utilize the **OSEMN framework**, it is composed of 5 steps which are described as follows:

1. **Obtain data**. In this first step we prepare the project, collect and get the datasets from different sources from surveys, databases, internet, files, etc. using different techniques as create new data, query databases, web scraping, downloading files, connecting to APIs, etc. We apply this step in the Section 2 Methods and Analysis, for more details go to Preparing the Data Science Project and the Dataset.

2. **Scrub data**. This step is considered, in data science, the most time-consuming depending on each project, that is why sometimes this step applies before Explore data or after. And it can be apply as many times as necessary throughout the project. It includes data cleaning, data pre-processing and handling missing values (this last process can be applied in the Explore data too, depending on each project). We employ this step in the Section 2 Methods and Analysis, for more details go to Preparing and Cleaning the Datasets.

3. **Explore data**. This procedure implements exploratory data analysis, where here, we must focus to understand and if it is possible, to find the relationship between the predict variable and possible predictor variables, additionally, detect missing values and outliers. Whereby, it involves the use of descriptive statistics and data visualizations. We apply this step in

the Section 2 Methods and Analysis, for more details go to Performing Data Exploration and Visualization to `data` dataset.

4. **Model data**. In this step, we build the model according to the results obtained or produced during the data exploration phase. We must also test and validate it. For more details about this procedure and how to we employ it, go to Section 3 Results.

5. **Interpret results**. This is the last phase and it is considered the most important, because we need to summarize all the results produced, such as the conclusion reached, what were its limitations and find out what the next course of action or future work would be, etc. of the created model in order to transmit them and that they can be understood or interpreted by people who have little or no knowledge of our subject. And to convey the results we must create a final report, a presentation or publish them in some type of media. We made this final report by creating an Rmarkdown document that generates a PDF document, here is the product.

Chapter 2

Methods and Analysis

2.1 Preparing the Data Science Project and the Dataset

This is the first step that belongs to **Obtain data**, here, we prepare the project and get the datasets from a specific internet site. The next code chunk install packages required if they do not exist, then load them.

IMPORTAT NOTICE: We advise you to install all the packages required for this project manually and individually before running the code. Because during the installation process they can cause errors due to the lack of libraries either in the operating system, in `R` or in both.

```
########  Install and Load the packages required +++++
#
# Install the libraries if they do not exist and load them
if(!require(this.path)) install.packages("this.path",
                                         repos = "http://cran.us.r-project.org")
if(!require(tidyverse)) install.packages("tidyverse",
                                         repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret",
                                     repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table",
                                          repos = "http://cran.us.r-project.org")
if(!require(ggthemes)) install.packages("ggthemes",
                                        repos = "http://cran.us.r-project.org")
if(!require(scales)) install.packages("scales",
                                      repos = "http://cran.us.r-project.org")
if(!require(Hmisc)) install.packages("Hmisc",
                                     repos = "http://cran.us.r-project.org")
if(!require(huxtable)) install.packages("huxtable",
                                        repos = "http://cran.us.r-project.org")
if(!require(GGally)) install.packages("GGally",
                                      repos = "http://cran.us.r-project.org")
if(!require(gridExtra)) install.packages("gridExtra",
```

```
                                          repos = "http://cran.us.r-project.org")
if(!require(randomForest)) install.packages("randomForest",
                                            repos = "http://cran.us.r-project.org")

library(this.path)
library(tidyverse)
library(caret)
library(data.table)
library(ggthemes)
library(scales)
library(Hmisc)
library(huxtable)
library(GGally)
library(gridExtra)
library(randomForest)
```

Now, we proceed to construct the structure of our project creating the subdirectories, we do this to facilitate and speed up collaborative work, share publications or reproducible research. It is well known that the structure of a project is:

- **our_project_dir**/
  - **rdas**/
  - **raw_data**/
  - **figs**/
- README.md
- final_report.Rmd
- download-data.R
- wrangle-data.R
- analysis.R

For this specific project, the structure of its directories is:

- **our_project_dir**/
  - **rdas**/
    - cp_house_price.rda
  - **raw_data**/
    - data.csv
- README.md
- report_house_price_proj.Rmd
- code_house_price_proj.R
- report_house_price_proj.pdf

NOTE: We can download the project from this repository on GitHub.

This code chunk get the current path, where this `.Rmd` file is running with the `this.dir` function from the `this.path` package, to set it like working directory. Check if the folders `raw_data` and `rdas` exist in the actual directory, if not create the `raw_data` and `rdas` directories. Get the version of `R`, assign it to the variable `v` that will serve us to set the seed, verify if the `data.csv` file exists in the `raw_data` directory, if it exists it prints a message that it already exists and this process is finished. If it does not exist, then, a message is printed that it does not exist, it is downloaded from https://gitlab.com/saulcol/house-price-data/-/raw/main/raw_data/data.csv to `raw_data` directory and load the `data.csv` file.

```
### set the working directory and create a subdirectory
#
# Get the current path with the `this.dir` function
wd <- this.dir(default = getwd())
# Set the working directory
setwd(wd)

# check if the folder "raw_data" exists in the current directory,
# if not creates a "raw_data" directory
ifelse(!dir.exists("raw_data"), dir.create("raw_data"), "Folder raw_data exists already")
```

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

```
# check if the folder "rdas" exists in the current directory,
# if not creates a "rdas" directory
ifelse(!dir.exists("rdas"), dir.create("rdas"), "Folder rdas exists already")
```

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

```
# ++++++++++++ Get the version of R ++++++++++++
v <- R.Version() # It is a List


# Verify if the "data.csv" file exists in the "raw_data/" directory
if(file.exists(paste0(wd, "/raw_data/data.csv"))==TRUE){
  print("File data.csv exists already")
}else{
  print("File data.csv does NOT exist...downloading")
  # Download the file from gitlab to "raw_data/" directory
  download.file("https://gitlab.com/saulcol/house-price-data/-/raw/main/raw_data/data.csv",
                paste0(wd, "/raw_data/data.csv"))
}
```

```
## [1] "File data.csv does NOT exist...downloading"
```

```r
# load the dataset
data <- read.csv("./raw_data/data.csv")
```

2.2 Performing Data Exploration and Visualization to `data` dataset

It is part of **Explore data**, where, we use descriptive statistics and data visualizations to understand and if it is possible, to find the relationship between the predict variable and possible predictor variables, additionally, detect missing values and outliers.

2.2.1 Overall Exploration in the Dataset

Let us start by performing a general exploration on the `data` dataframe by running the following functions to see its structure and how it is composed.

```r
### ++++++++++ Overall Exploration in the Dataset ++++++++++++++
#
#
# Knowing its structure and how it is composed the `data` dataframe
glimpse(data)
```

```
## Rows: 4,600
## Columns: 18
## $ date          <chr> "2014-05-02 00:00:00", "2014-05-02 00:00:00", "2014-05-0~
## $ price         <dbl> 313000, 2384000, 342000, 420000, 550000, 490000, 335000,~
## $ bedrooms      <dbl> 3, 5, 3, 3, 4, 2, 2, 4, 3, 4, 3, 4, 3, 3, 5, 3, 3, 4, 3,~
## $ bathrooms     <dbl> 1.50, 2.50, 2.00, 2.25, 2.50, 1.00, 2.00, 2.50, 2.50, 2.~
## $ sqft_living   <int> 1340, 3650, 1930, 2000, 1940, 880, 1350, 2710, 2430, 152~
## $ sqft_lot      <int> 7912, 9050, 11947, 8030, 10500, 6380, 2560, 35868, 88426~
## $ floors        <dbl> 1.5, 2.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0, 1.0, 1.5, 1.0, 1~
## $ waterfront    <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ view          <int> 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ condition     <int> 3, 5, 4, 4, 4, 3, 3, 3, 4, 3, 3, 5, 3, 4, 3, 4, 4, 3, 3,~
## $ sqft_above    <int> 1340, 3370, 1930, 1000, 1140, 880, 1350, 2710, 1570, 152~
## $ sqft_basement <int> 0, 280, 0, 1000, 800, 0, 0, 0, 860, 0, 0, 1010, 360, 0, ~
## $ yr_built      <int> 1955, 1921, 1966, 1963, 1976, 1938, 1976, 1989, 1985, 19~
## $ yr_renovated  <int> 2005, 0, 0, 0, 1992, 1994, 0, 0, 0, 2010, 1994, 1988, 0,~
## $ street        <chr> "18810 Densmore Ave N", "709 W Blaine St", "26206-26214 ~
## $ city          <chr> "Shoreline", "Seattle", "Kent", "Bellevue", "Redmond", "~
## $ statezip      <chr> "WA 98133", "WA 98119", "WA 98042", "WA 98008", "WA 9805~
## $ country       <chr> "USA", "USA", "USA", "USA", "USA", "USA", "USA", "USA", ~
```

```r
str(data)
```

```
## 'data.frame':    4600 obs. of  18 variables:
##  $ date          : chr  "2014-05-02 00:00:00" "2014-05-02 00:00:00" "2014-05-02 00:00:00" "2014
##  $ price         : num  313000 2384000 342000 420000 550000 ...
##  $ bedrooms      : num  3 5 3 3 4 2 2 4 3 4 ...
##  $ bathrooms     : num  1.5 2.5 2 2.25 2.5 1 2 2.5 2.5 2 ...
##  $ sqft_living   : int  1340 3650 1930 2000 1940 880 1350 2710 2430 1520 ...
##  $ sqft_lot      : int  7912 9050 11947 8030 10500 6380 2560 35868 88426 6200 ...
##  $ floors        : num  1.5 2 1 1 1 1 1 2 1 1.5 ...
##  $ waterfront    : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ view          : int  0 4 0 0 0 0 0 0 0 0 ...
##  $ condition     : int  3 5 4 4 4 3 3 3 4 3 ...
##  $ sqft_above    : int  1340 3370 1930 1000 1140 880 1350 2710 1570 1520 ...
##  $ sqft_basement: int  0 280 0 1000 800 0 0 0 860 0 ...
##  $ yr_built      : int  1955 1921 1966 1963 1976 1938 1976 1989 1985 1945 ...
##  $ yr_renovated : int  2005 0 0 0 1992 1994 0 0 0 2010 ...
##  $ street        : chr  "18810 Densmore Ave N" "709 W Blaine St" "26206-26214 143rd Ave SE" "85
##  $ city          : chr  "Shoreline" "Seattle" "Kent" "Bellevue" ...
##  $ statezip      : chr  "WA 98133" "WA 98119" "WA 98042" "WA 98008" ...
##  $ country       : chr  "USA" "USA" "USA" "USA" ...
```

```r
# How many rows and columns are there in the dataframe?
cat('The dataframe called data has', dim(data)[1], 'rows and', dim(data)[2], 'columns.')
```

```
## The dataframe called data has 4600 rows and 18 columns.
```

By executing the above code, we know that the `data` dataframe is composed by **4,600 observations** (`rows`) and **18 variables** (`columns`). As well, its eighteen `columns` have these characteristics:

```
date          : character
price         : numeric <dbl>
bedrooms      : numeric <dbl>
bathrooms     : numeric <dbl>
sqft_living   : integer
sqft_lot      : integer
floors        : numeric <dbl>
waterfront    : integer
view          : integer
condition     : integer
sqft_above    : integer
sqft_basement: integer
yr_built      : integer
```

```
yr_renovated : integer
street       : character
city         : character
statezip     : character
country      : character
```

How many times each class occurs in this dataframe?

```
as.data.frame(table(unlist(
  lapply(data,
         function(x) paste(class(x), collapse = ","))))) %>%
  separate(col = "Var1", into = c("Var1", "Var2"), sep = ",", fill = "right") %>%
  rename(Class = Var1, OtherClass = Var2) %>%
  as_hux() %>%
  set_font_size(9) %>%
  set_tb_padding(2) %>%
  set_col_width(c(.1, .1, .1)) %>%
  set_latex_float("h!") %>%
  theme_basic()
```

Class	OtherClass	Freq
character		5
integer		9
numeric		4

The dataframe called `data` has 5 `character`, 9 `integer` and 4 `numeric` variables.

We continue now observing its statistical summary with the `summary()` function.

```
# Get a summary statistics
summary(data)
```

```
##      date               price            bedrooms       bathrooms
##  Length:4600        Min.   :       0   Min.   :0.000   Min.   :0.000
##  Class :character   1st Qu.:  322875   1st Qu.:3.000   1st Qu.:1.750
##  Mode  :character   Median :  460943   Median :3.000   Median :2.250
##                     Mean   :  551963   Mean   :3.401   Mean   :2.161
##                     3rd Qu.:  654962   3rd Qu.:4.000   3rd Qu.:2.500
##                     Max.   :26590000   Max.   :9.000   Max.   :8.000
##   sqft_living       sqft_lot           floors        waterfront
##  Min.   :  370   Min.   :    638   Min.   :1.000   Min.   :0.000000
##  1st Qu.: 1460   1st Qu.:   5001   1st Qu.:1.000   1st Qu.:0.000000
##  Median : 1980   Median :   7683   Median :1.500   Median :0.000000
##  Mean   : 2139   Mean   :  14852   Mean   :1.512   Mean   :0.007174
```

```
##  3rd Qu.: 2620   3rd Qu.:  11001   3rd Qu.:2.000   3rd Qu.:0.000000
##  Max.   :13540   Max.   :1074218   Max.   :3.500   Max.   :1.000000
##       view          condition       sqft_above   sqft_basement
##  Min.   :0.0000   Min.   :1.000   Min.   : 370   Min.   :   0.0
##  1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:1190   1st Qu.:   0.0
##  Median :0.0000   Median :3.000   Median :1590   Median :   0.0
##  Mean   :0.2407   Mean   :3.452   Mean   :1827   Mean   : 312.1
##  3rd Qu.:0.0000   3rd Qu.:4.000   3rd Qu.:2300   3rd Qu.: 610.0
##  Max.   :4.0000   Max.   :5.000   Max.   :9410   Max.   :4820.0
##     yr_built     yr_renovated       street              city
##  Min.   :1900   Min.   :   0.0   Length:4600        Length:4600
##  1st Qu.:1951   1st Qu.:   0.0   Class :character   Class :character
##  Median :1976   Median :   0.0   Mode  :character   Mode  :character
##  Mean   :1971   Mean   : 808.6
##  3rd Qu.:1997   3rd Qu.:1999.0
##  Max.   :2014   Max.   :2014.0
##    statezip           country
##  Length:4600        Length:4600
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

Other way, to obtain a better summary statistics is with the function `describe()` of the package `Hmisc`.

```
# Other way, to obtain a better summary statistics
Hmisc::describe(data)
```

```
## data
##
##  18  Variables      4600  Observations
## ---------------------------------------------------------------------------
## date
##        n  missing distinct
##     4600        0       70
##
## lowest : 2014-05-02 00:00:00 2014-05-03 00:00:00 2014-05-04 00:00:00 2014-05-05 00:00:00 2014-
## highest: 2014-07-06 00:00:00 2014-07-07 00:00:00 2014-07-08 00:00:00 2014-07-09 00:00:00 2014-
## ---------------------------------------------------------------------------
## price
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##     4600        0     1741        1   551963   356066   200000   239950
##      .25      .50      .75      .90      .95
##   322875   460943   654963   900000  1184050
##
```

```
## lowest :        0     7800    80000    83000    83300
## highest:  4489000  4668000  7062500 12899000 26590000
## ---------------------------------------------------------------------------
## bedrooms
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##     4600        0       10    0.875    3.401   0.9486        2        2
##      .25      .50      .75      .90      .95
##        3        3        4        4        5
##
## lowest : 0 1 2 3 4, highest: 5 6 7 8 9
##
## Value          0     1     2     3     4     5     6     7     8     9
## Frequency      2    38   566  2032  1531   353    61    14     2     1
## Proportion 0.000 0.008 0.123 0.442 0.333 0.077 0.013 0.003 0.000 0.000
## ---------------------------------------------------------------------------
## bathrooms
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##     4600        0       26    0.974    2.161   0.8521     1.00     1.00
##      .25      .50      .75      .90      .95
##     1.75     2.25     2.50     3.00     3.50
##
## lowest : 0.00 0.75 1.00 1.25 1.50, highest: 5.75 6.25 6.50 6.75 8.00
## ---------------------------------------------------------------------------
## sqft_living
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##     4600        0      566        1     2139     1015      950     1110
##      .25      .50      .75      .90      .95
##     1460     1980     2620     3340     3870
##
## lowest :   370   380   420   430   490, highest:  8020  8670  9640 10040 13540
## ---------------------------------------------------------------------------
## sqft_lot
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##     4600        0     3113        1    14853    17219     1691     3300
##      .25      .50      .75      .90      .95
##     5001     7683    11001    24302    43560
##
## lowest :     638     681     704     746     747
## highest:  423838  435600  478288  641203 1074218
## ---------------------------------------------------------------------------
## floors
##        n  missing distinct     Info     Mean      Gmd
##     4600        0        6    0.832    1.512   0.5583
##
## lowest : 1.0 1.5 2.0 2.5 3.0, highest: 1.5 2.0 2.5 3.0 3.5
##
## Value        1.0   1.5   2.0   2.5   3.0   3.5
```

```
## Frequency   2174   444  1811    41   128     2
## Proportion 0.473 0.097 0.394 0.009 0.028 0.000
## ---------------------------------------------------------------------------
## waterfront
##        n  missing distinct     Info      Sum     Mean      Gmd
##     4600        0        2    0.021       33 0.007174  0.01425
##
## ---------------------------------------------------------------------------
## view
##        n  missing distinct     Info     Mean      Gmd
##     4600        0        5    0.271   0.2407   0.4432
##
## lowest : 0 1 2 3 4, highest: 0 1 2 3 4
##
## Value          0     1     2     3     4
## Frequency   4140    69   205   116    70
## Proportion 0.900 0.015 0.045 0.025 0.015
## ---------------------------------------------------------------------------
## condition
##        n  missing distinct     Info     Mean      Gmd
##     4600        0        5    0.735    3.452   0.6549
##
## lowest : 1 2 3 4 5, highest: 1 2 3 4 5
##
## Value          1     2     3     4     5
## Frequency      6    32  2875  1252   435
## Proportion 0.001 0.007 0.625 0.272 0.095
## ---------------------------------------------------------------------------
## sqft_above
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##     4600        0      511        1     1827    912.2      860      970
##      .25      .50      .75      .90      .95
##     1190     1590     2300     3030     3440
##
## lowest :  370  380  420  430  490, highest: 6640 7320 7680 8020 9410
## ---------------------------------------------------------------------------
## sqft_basement
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##     4600        0      207    0.787    312.1    445.8        0        0
##      .25      .50      .75      .90      .95
##        0        0      610     1000     1210
##
## lowest :    0   20   50   60   65, highest: 2550 2730 2850 4130 4820
## ---------------------------------------------------------------------------
## yr_built
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##     4600        0      115        1     1971    33.74     1913     1925
```

```
##      .25      .50      .75      .90      .95
##     1951     1976     1997     2006     2009
##
## lowest : 1900 1901 1902 1903 1904, highest: 2010 2011 2012 2013 2014
## ---------------------------------------------------------------------------
## yr_renovated
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##     4600        0       60     0.79    808.6    964.9        0        0
##      .25      .50      .75      .90      .95
##        0        0     1999     2006     2011
##
## lowest :    0 1912 1913 1923 1934, highest: 2010 2011 2012 2013 2014
##
## Value          0  1910  1915  1925  1935  1945  1950  1955  1960  1965  1970
## Frequency   2735    33     1    57     6     7     1    26     6    13    51
## Proportion 0.595 0.007 0.000 0.012 0.001 0.002 0.000 0.006 0.001 0.003 0.011
##
## Value       1975  1980  1985  1990  1995  2000  2005  2010  2015
## Frequency      8    60    86   147   148   401   398   283   133
## Proportion 0.002 0.013 0.019 0.032 0.032 0.087 0.087 0.062 0.029
##
## For the frequency table, variable is rounded to the nearest 5
## ---------------------------------------------------------------------------
## street
##        n  missing distinct
##     4600        0     4525
##
## lowest : 1 View Ln NE        10 W Etruria St     100 20th Ave E      100 24th Ave E      100 M
## highest: Shangri-La Way NW   Sunrise Loop Trail  Tolt Pipeline Trail Trossachs Blvd SE   Valle
## ---------------------------------------------------------------------------
## city
##        n  missing distinct
##     4600        0       44
##
## lowest : Algona             Auburn             Beaux Arts Village Bellevue           Black Dia
## highest: Snoqualmie Pass    Tukwila            Vashon             Woodinville        Yarrow Po
## ---------------------------------------------------------------------------
## statezip
##        n  missing distinct
##     4600        0       77
##
## lowest : WA 98001 WA 98002 WA 98003 WA 98004 WA 98005
## highest: WA 98188 WA 98198 WA 98199 WA 98288 WA 98354
## ---------------------------------------------------------------------------
## country
##        n  missing distinct    value
##     4600        0        1      USA
```

```
##
## Value       USA
## Frequency  4600
## Proportion    1
## --------------------------------------------------------------------------------
```

In general, it can be observed the mentioned dataframe **does not have missing values**. The `date` column has a range from 2014-05-02 00:00:00 to 2014-07-10 00:00:00 and has 70 different dates, it is candidate to **be eliminate**. The `price` column has a range from 0 to 26590000 and has 1741 different prices. The `bedrooms` column has 10 different bedrooms that are in a range from 0 to 9 with increments by 1. The `bathrooms` variable has a range from 0 to 8 and has 26 different bathrooms. The `sqft_living` column has a range from 370 to 13540 and has 566 different square foot livings. The `sqft_lot` variable has a range from 638 to 1074218 and has 3113 different square foot lots. The `floors` column has 6 different floors that are in a range from 1.0 to 3.5 with increments by 0.5. The `waterfront` does not have too much information, it is candidate to **be eliminate**. The `view` column has 5 different views that are in a range from 0 to 4 with increments by 1, 0 value is present in 4140 observations that represent the 90%, it is candidate to be **eliminate**. The `condition` variable has 5 different conditions that are in a range from 1 to 5 with increments by 1. The `sqft_above` column has a range from 370 to 9410 and has 511 different square foot aboves. The `sqft_basement` column has a range from 0 to 4820 and has 207 different square foot basements. The `yr_built` variable has a range from 1900 to 2014 and has 115 different years built. The `yr_renovated` variable has a range from 0 to 2014 and has 60 different years renovated. The `street` column has 4525 different streets that is all, it is candidate to **be eliminate**. The `city` column has 44 different cities, it is candidate to be **eliminate** (maybe). The `statezip` variable has 77 different states zip, it is candidate to be **eliminate** (perhaps). And the last one column `country` has 1 different country, it is candidate to be **eliminate**.

Recall, our **response** or **dependent variable** is the `price` column.

2.2.2 Are there missing values?

Earlier when we used the `describe` function we saw that the eighteen columns do not have missing values. We can corroborate it with this code.

```
###++++++++++++ Are there missing values? ++++++++++++++++++
#
# Count total missing values in each column of data frame
colSums(is.na(data))
```

```
##          date         price      bedrooms     bathrooms   sqft_living
##             0             0             0             0             0
##      sqft_lot        floors    waterfront          view     condition
##             0             0             0             0             0
```

```
##    sqft_above sqft_basement      yr_built  yr_renovated        street
##             0             0             0             0             0
##          city      statezip       country
##             0             0             0
```

In effect, the `data` dataframe **does not have missing values**. Now, we proceed to perform a data analysis, first of all on the numerical variables of this dataset. But, we need to make a few changes to those variables.

2.2.3 Data Analysis in numerical columns to detect outliers

First, we know which columns are integers.

```
# Know which columns are integers
names(data)[sapply(data, is.integer)]
```

```
## [1] "sqft_living"   "sqft_lot"      "waterfront"    "view"
## [5] "condition"     "sqft_above"    "sqft_basement" "yr_built"
## [9] "yr_renovated"
```

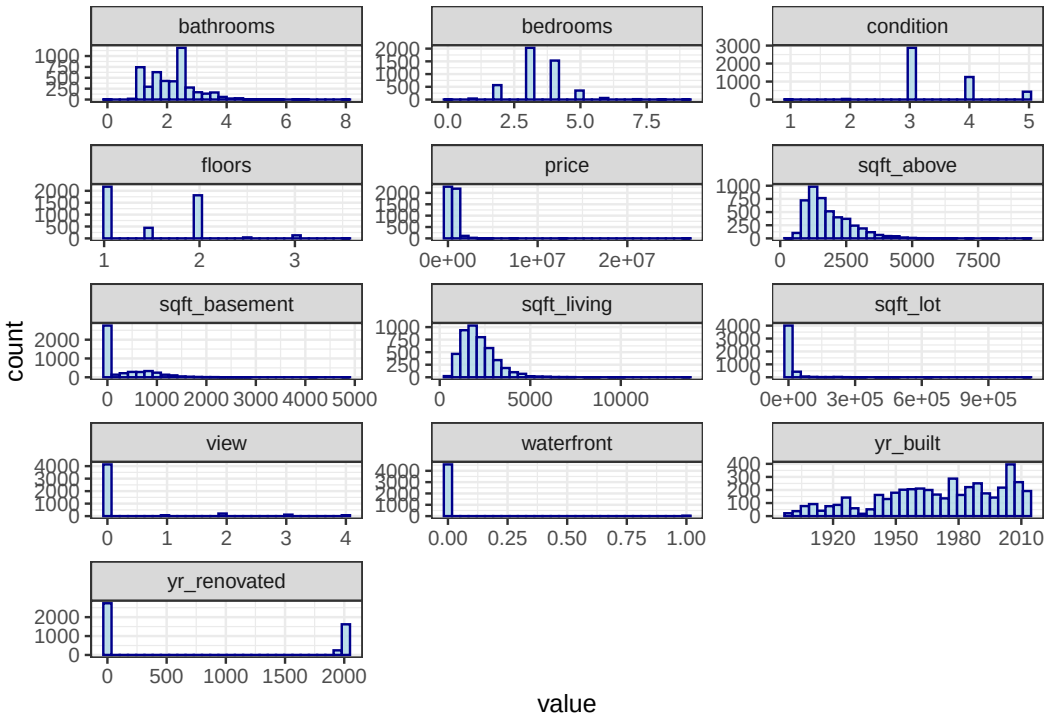
Make a copy of the dataframe, and then, we convert these integer columns to numeric.

```
# Make a copy of the dataframe
data2 <- data

# Convert these integer columns to numeric
data2$sqft_living <- as.numeric(data$sqft_living)
data2$sqft_lot <- as.numeric(data$sqft_lot)
data2$view <- as.numeric(data$view)
data2$waterfront <- as.numeric(data$waterfront)
data2$condition <- as.numeric(data$condition)
data2$sqft_above <- as.numeric(data$sqft_above)
data2$sqft_basement <- as.numeric(data$sqft_basement)
```

We do a multiple histogram about all numeric columns.

```
data2 %>%
  pivot_longer(cols = c(2:14),
               names_to = 'variables',
               values_to = 'value') %>%
  ggplot(aes(y = value)) +
  geom_histogram(color = "darkblue", alpha = 0.8, fill = "lightblue") +
```

```
  facet_wrap(~variables, ncol = 3, scales = 'free') +
  coord_flip() +
  theme_bw()
```



From the previous graph, we must put emphasis on the following variables: `price`, `sqft_lot`, `view`, `waterfront`, `condition`, `sqft_basement` and `yr_renovated`.

We create a plot with multiple histograms with different X-axis scale from our response variable that is `price` with the use of the function `grid.arrange` of the `gridExtra` package and control the formatting of the `labels = comma` on `scale_x_continuous` with the `scales` package.

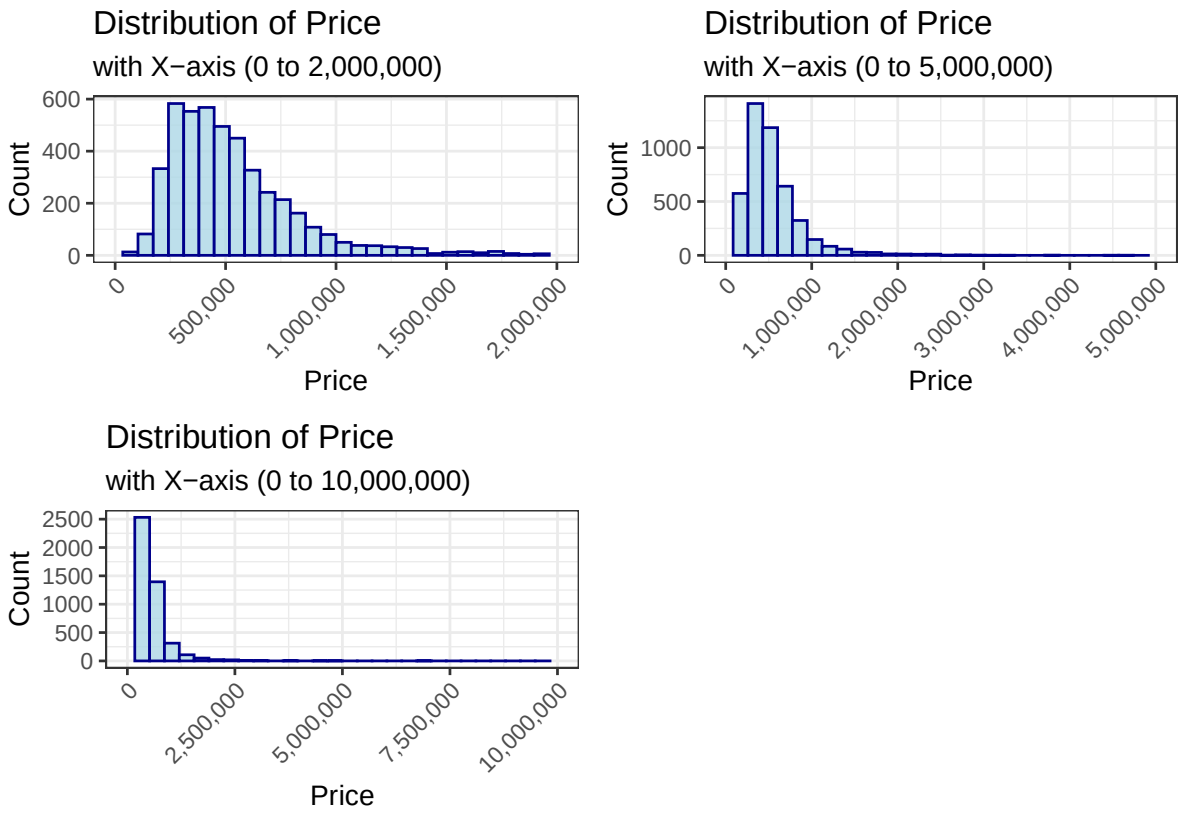
```
# Create a plot with multiple histograms with different X-axis scale from
# our response variable that is `price` with the use of the function
# `grid.arrange` of the `gridExtra` package and control the formatting
# of the `labels = comma` on `scale_x_continuous` with the `scales` package
p1 <- data2 %>%
  ggplot(aes(x = price)) +
  geom_histogram(color = "darkblue", alpha = 0.8, fill = "lightblue") +
  ggtitle("Distribution of Price",
          subtitle = "with X-axis (0 to 2,000,000)") +
  xlab("Price") +
  ylab("Count") +
  scale_x_continuous(limits = c(0,2e6), labels = comma) +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

```r
p2 <- data2 %>%
  ggplot(aes(x = price)) +
  geom_histogram(color = "darkblue", alpha = 0.8, fill = "lightblue") +
  ggtitle("Distribution of Price",
          subtitle = "with X-axis (0 to 5,000,000)") +
  xlab("Price") +
  ylab("Count") +
  scale_x_continuous(limits = c(0,5e6), labels = comma) +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

p3 <- data2 %>%
  ggplot(aes(x = price)) +
  geom_histogram(color = "darkblue", alpha = 0.8, fill = "lightblue") +
  ggtitle("Distribution of Price",
          subtitle = "with X-axis (0 to 10,000,000)") +
  xlab("Price") +
  ylab("Count") +
  scale_x_continuous(limits = c(0,10e6), labels = comma) +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

p4 <- data2 %>%
  ggplot(aes(x = price)) +
  geom_histogram(color = "darkblue", alpha = 0.8, fill = "lightblue") +
  ggtitle("Distribution of Price",
          subtitle = "with X-axis (0 to 15,000,000)") +
  xlab("Price") +
  ylab("Count") +
  scale_x_continuous(limits = c(0,15e6), labels = comma) +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

p5 <- data2 %>%
  ggplot(aes(x = price)) +
  geom_histogram(color = "darkblue", alpha = 0.8, fill = "lightblue") +
  ggtitle("Distribution of Price",
          subtitle = "with X-axis (0 to 20,000,000)") +
  xlab("Price") +
  ylab("Count") +
  scale_x_continuous(limits = c(0,20e6), labels = comma) +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

p6 <- data2 %>%
  ggplot(aes(x = price)) +
```

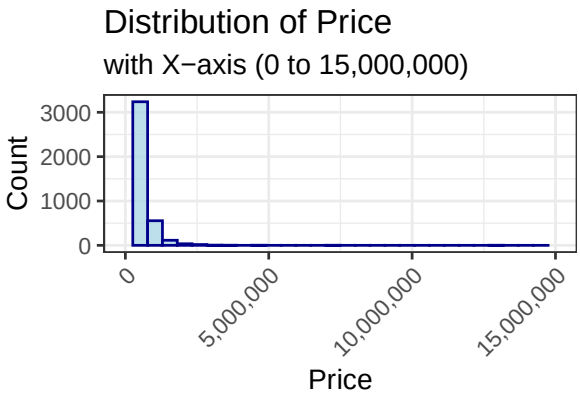
```
  geom_histogram(color = "darkblue", alpha = 0.8, fill = "lightblue") +
  ggtitle("Distribution of Price",
          subtitle = "with X-axis (0 to 27,000,000)") +
  xlab("Price") +
  ylab("Count") +
  scale_x_continuous(limits = c(0,27e6), labels = comma) +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

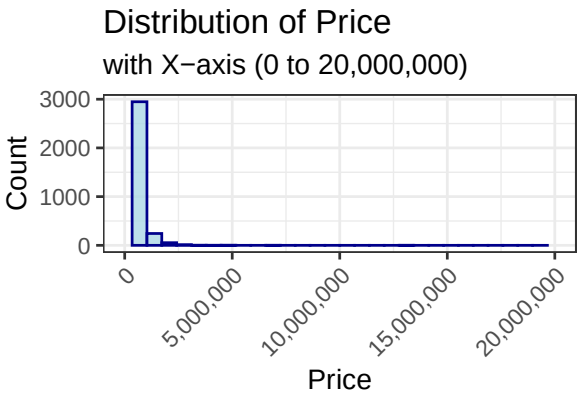
We separate in two grids.

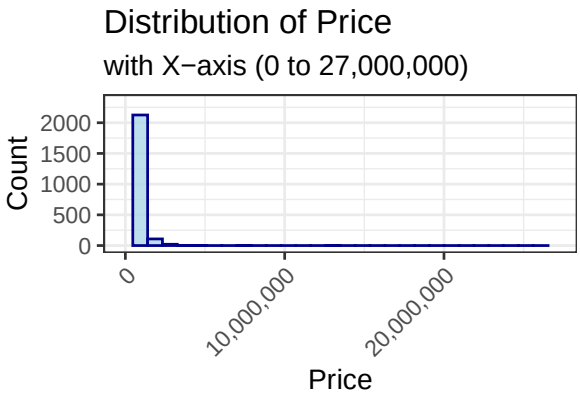
```
grid.arrange(p1,p2,p3, ncol = 2)
```




```
grid.arrange(p4,p5,p6, ncol = 2)
```







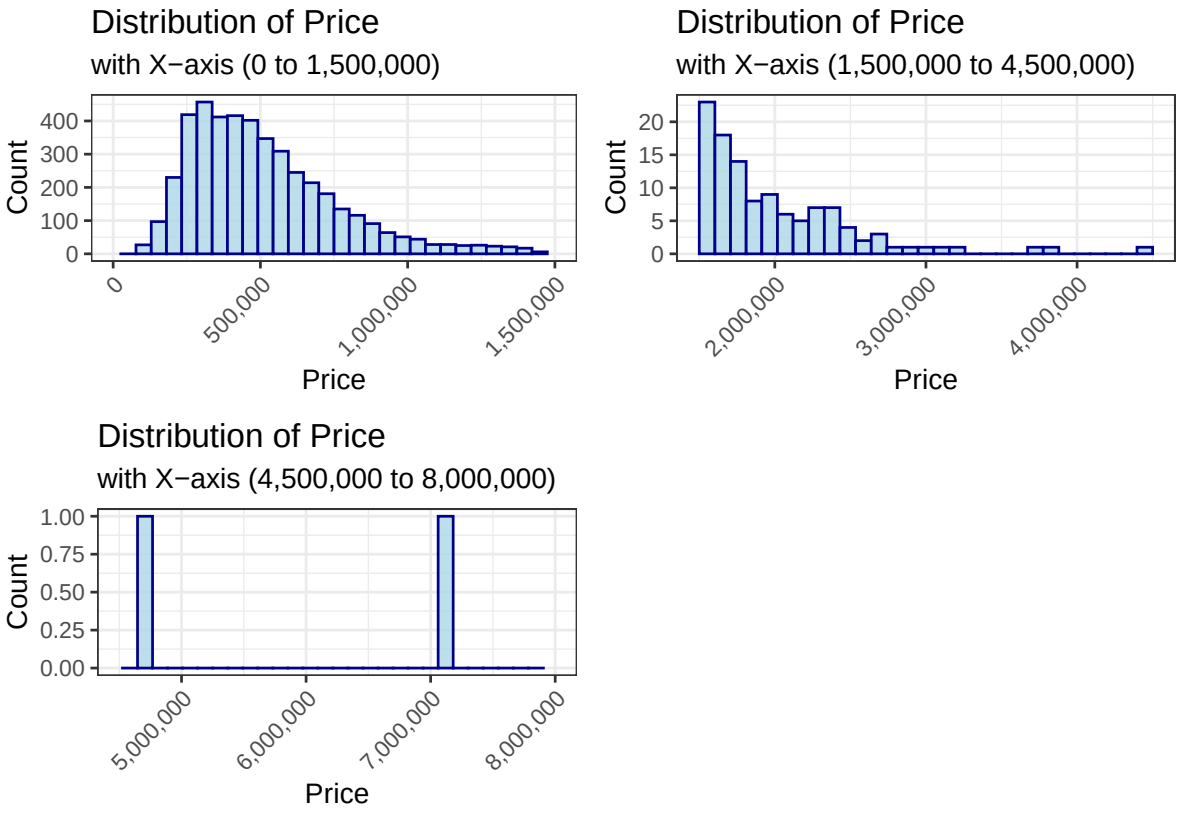
We proceed to make a plot with multiple histograms in certain ranges (bins) in the `price` variable.

```
p1 <- data2 %>%
  ggplot(aes(x = price)) +
  geom_histogram(color = "darkblue", alpha = 0.8, fill = "lightblue") +
  ggtitle("Distribution of Price",
          subtitle = "with X-axis (0 to 1,500,000)") +
  xlab("Price") +
  ylab("Count") +
  scale_x_continuous(limits = c(0,15e5), labels = comma) +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

p2 <- data2 %>%
  ggplot(aes(x = price)) +
  geom_histogram(color = "darkblue", alpha = 0.8, fill = "lightblue") +
  ggtitle("Distribution of Price",
          subtitle = "with X-axis (1,500,000 to 4,500,000)") +
  xlab("Price") +
  ylab("Count") +
  scale_x_continuous(limits = c(15e5,45e5), labels = comma) +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

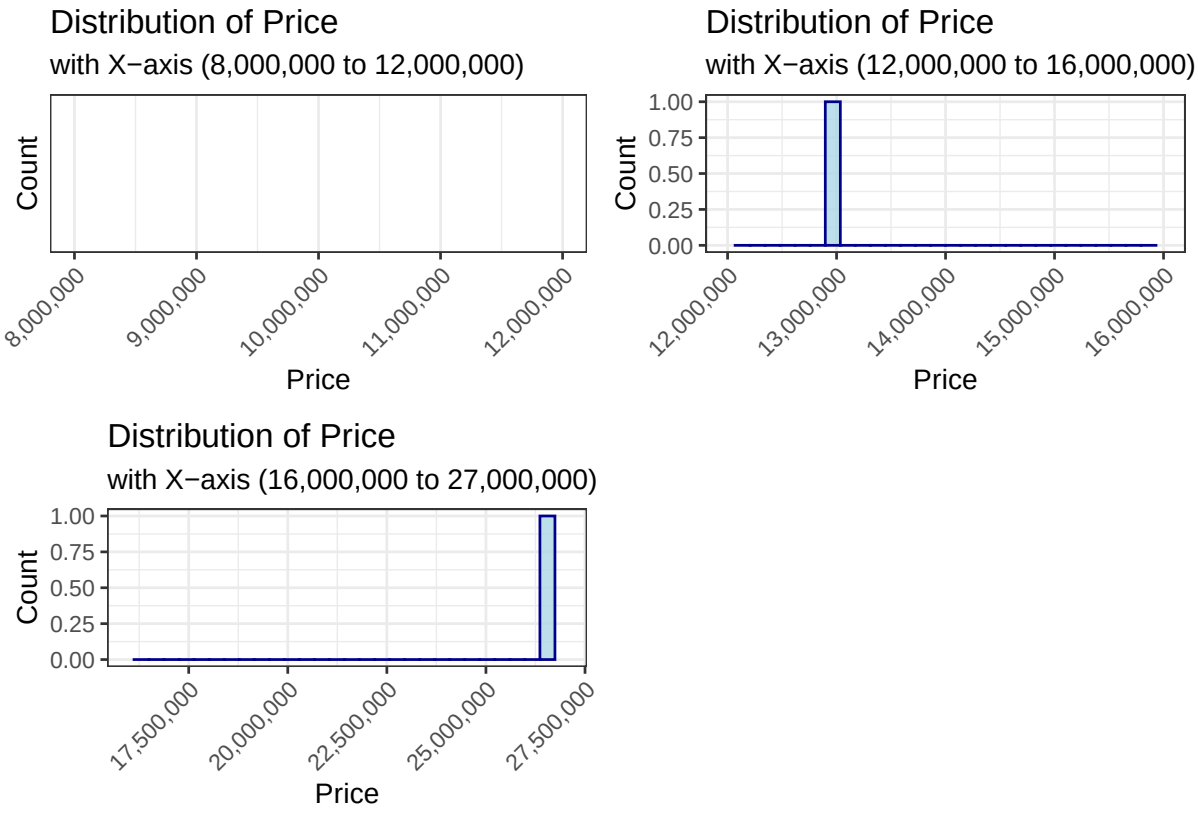
```r
p3 <- data2 %>%
  ggplot(aes(x = price)) +
  geom_histogram(color = "darkblue", alpha = 0.8, fill = "lightblue") +
  ggtitle("Distribution of Price",
          subtitle = "with X-axis (4,500,000 to 8,000,000)") +
  xlab("Price") +
  ylab("Count") +
  scale_x_continuous(limits = c(45e5,8e6), labels = comma) +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

p4 <- data2 %>%
  ggplot(aes(x = price)) +
  geom_histogram(color = "darkblue", alpha = 0.8, fill = "lightblue") +
  ggtitle("Distribution of Price",
          subtitle = "with X-axis (8,000,000 to 12,000,000)") +
  xlab("Price") +
  ylab("Count") +
  scale_x_continuous(limits = c(8e6,12e6), labels = comma) +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

p5 <- data2 %>%
  ggplot(aes(x = price)) +
  geom_histogram(color = "darkblue", alpha = 0.8, fill = "lightblue") +
  ggtitle("Distribution of Price",
          subtitle = "with X-axis (12,000,000 to 16,000,000)") +
  xlab("Price") +
  ylab("Count") +
  scale_x_continuous(limits = c(12e6,16e6), labels = comma) +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

p6 <- data2 %>%
  ggplot(aes(x = price)) +
  geom_histogram(color = "darkblue", alpha = 0.8, fill = "lightblue") +
  ggtitle("Distribution of Price",
          subtitle = "with X-axis (16,000,000 to 27,000,000)") +
  xlab("Price") +
  ylab("Count") +
  scale_x_continuous(limits = c(16e6,27e6), labels = comma) +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

Again, we divide in two grids.

```
grid.arrange(p1,p2,p3, ncol = 2)
```




```
grid.arrange(p4,p5,p6, ncol = 2)
```



Compute the `min()`, `mean()`, `median()`, `max()`, `sd()` and the coefficients of variation, they are obtained by dividing `sd()/mean()`, across the `price`, `sqft_lot`, `view`, `waterfront`, `condition`, `sqft_basement` and `yr_renovated` columns. With descending order in `Coefficients_Variation`.

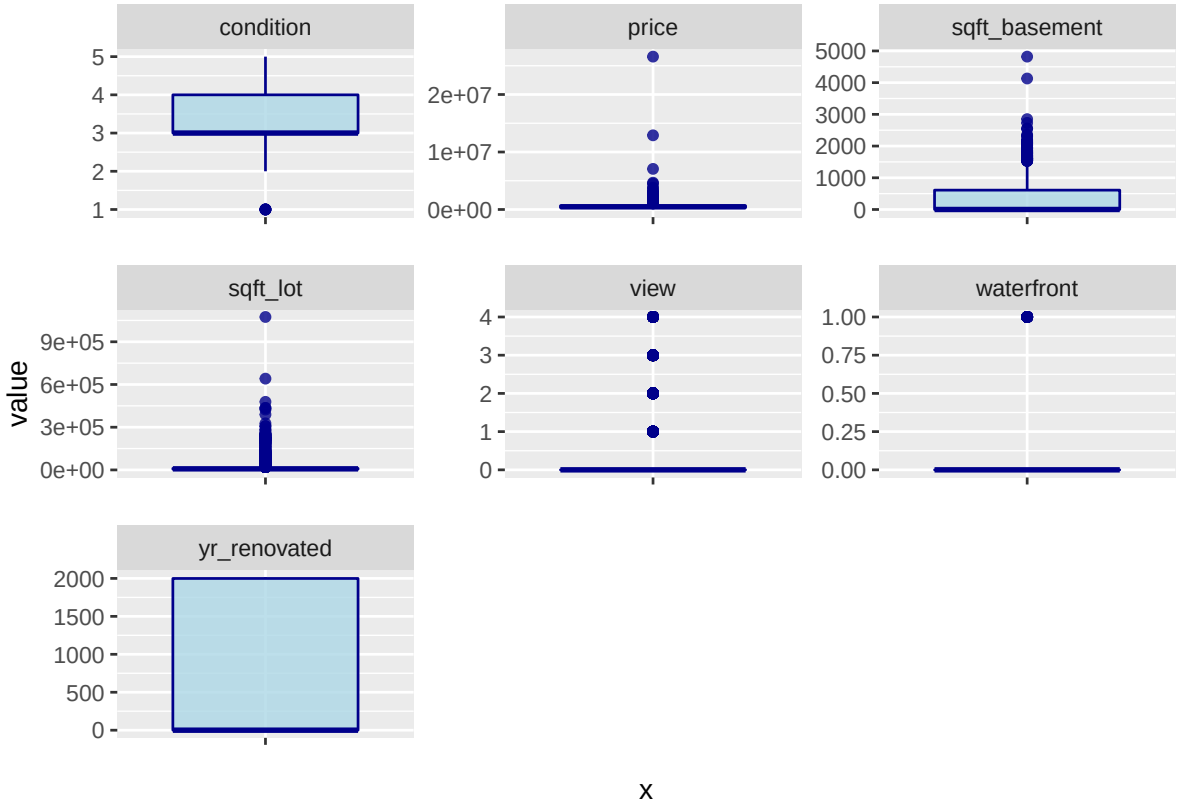
```
data2 %>%
  pivot_longer(cols = c(2, 6, 8:10, 12, 14),
               names_to = 'Variables',
               values_to = 'value') %>%
  group_by(Variables) %>%
  summarise(Min = min(value),
            Mean = mean(value),
            Median = median(value),
            Max = max(value),
            Coefficients_Variation = sd(value)/mean(value)) %>%
  arrange(desc(Coefficients_Variation)) %>%
  as_hux() %>%
  set_font_size(9) %>%
  set_tb_padding(2) %>%
  set_col_width(c(.1, .1, .1, .1, .1, .2)) %>%
  set_number_format(everywhere, 2:6, 3) %>%
  set_latex_float("h!") %>%
  theme_basic()
```

Variables	Min	Mean	Median	Max	Coefficients_Variation
waterfront	0.000	0.007	0.000	1.000	11.765
view	0.000	0.241	0.000	4.000	3.235
sqft_lot	638.000	14852.516	7683.000	1074218.000	2.416
sqft_basement	0.000	312.082	0.000	4820.000	1.487
yr_renovated	0.000	808.608	0.000	2014.000	1.211
price	0.000	551962.988	460943.462	26590000.000	1.022
condition	1.000	3.452	3.000	5.000	0.196

From the above table, we see that the variable `condition` has a low standard deviation, in the case of the variables `price`, `sqft_basement` and `yr_renovated`, they have a high standard deviation. The `sqft_lot` and `view` variables have a somewhat high standard deviation. And the `waterfront` variable has too high standard deviation. Also, in the `price` variable the maximum value is 57 times than the median, and `sqft_lot` variable the maximum value is 139 times higher than the median. Which tells us that the variables `price`, `sqft_basement`, `yr_renovated`, `sqft_lot`, `view` and `waterfront` have the presence of outliers.

We can corroborate those statements by plotting the following multiple boxplots:

```
data2 %>%
  pivot_longer(cols = c(2, 6, 8:10, 12, 14),
               names_to = 'variables',
               values_to = 'value') %>%
  ggplot(aes(x = "", y = value)) +
  geom_boxplot(color = "darkblue", alpha = 0.8, fill = "lightblue") +
  facet_wrap(~variables, scales = 'free')
```

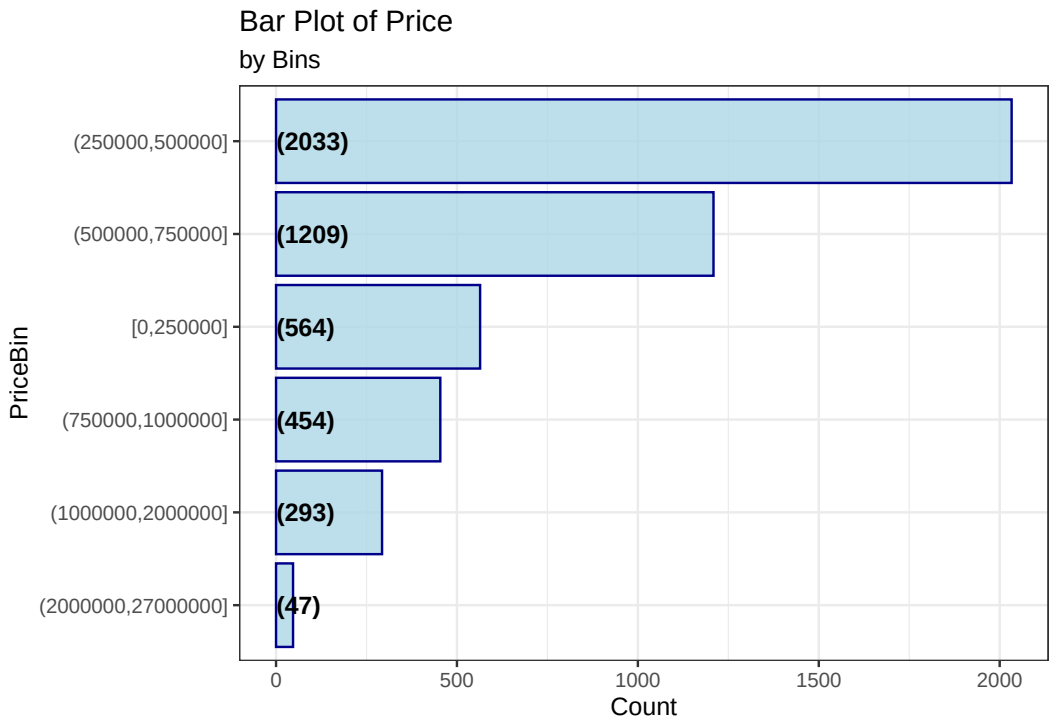


In effect, those variables have outliers.

2.2.4 How the predictor variables affect the `price` variable

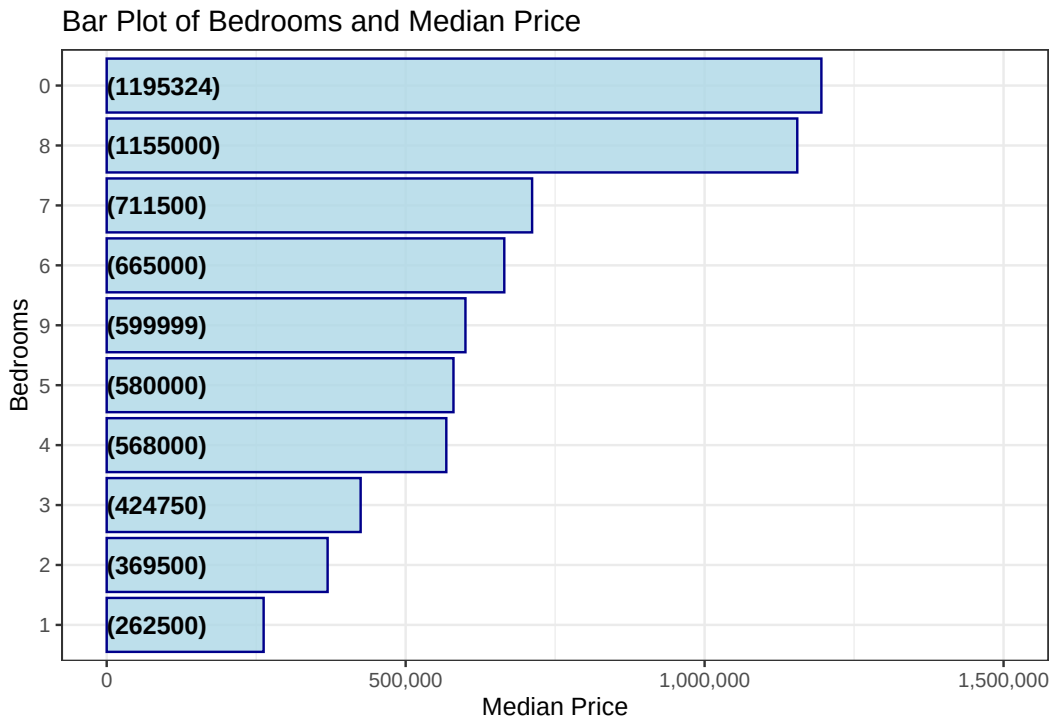
As we know, the `price` column is our outcome, the first thing we will do in this variable is we are going to create a new column called `pricebin` to categorize the `price` variable to generate some bins, example, (0 to 250000), (250000 to 500000), (500000 to 750000), (750000 to 1000000), (1000000 to 2000000) and (2000000 to 27000000). Then, visualize a bar plot of `price` by bins.

```
data2$pricebin <- cut(data2$price,
                      c(0, 250000, 500000, 750000, 1000000, 2000000, 27000000),
                      dig.lab = 10, include.lowest = TRUE)

# Then, visualize a bar plot of `price` by bins
data2 %>%
  mutate(pricebin = as.factor(pricebin)) %>%
  group_by(pricebin) %>%
  summarise(Count = n()) %>%
  ungroup() %>%
  mutate(pricebin = reorder(pricebin, Count)) %>%
  arrange(desc(Count)) %>%
  ggplot(aes(x = pricebin, y = Count)) +
  geom_bar(stat='identity', color = "darkblue", alpha = 0.8, fill = "lightblue") +
  geom_text(aes(x = pricebin, y = 1, label = paste0("(",Count,")",sep = "")),
            hjust = 0, vjust = .5, size = 4, colour = 'black',
            fontface = 'bold') +
  ggtitle("Bar Plot of Price",
          subtitle = "by Bins") +
  xlab("PriceBin") +
  ylab("Count") +
  coord_flip() +
  theme_bw()
```



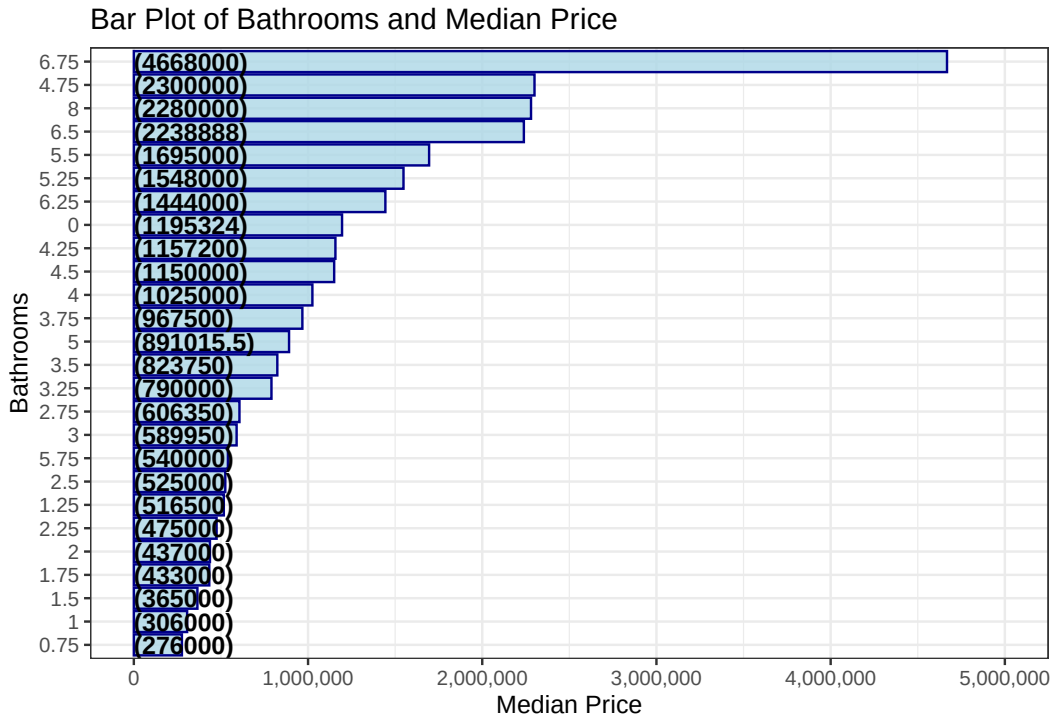
Next, we are going to show four bar plots of the `bedrooms`, `bathrooms`, `yr_built` and `yr_renovated` variables versus median `price`. The first bar plot of `bedrooms` and median `price` in descending order.

```
data2 %>%
  group_by(bedrooms) %>%
  summarise(PriceMedian = median(price, na.rm = TRUE)) %>%
  ungroup() %>%
  mutate(bedrooms = reorder(bedrooms,PriceMedian)) %>%
  arrange(desc(PriceMedian)) %>%
  ggplot(aes(x = bedrooms,y = PriceMedian)) +
  geom_bar(stat='identity', color = "darkblue", alpha = 0.8, fill = "lightblue") +
  geom_text(aes(x = bedrooms, y = 1, label = paste0("(",PriceMedian,")",sep="")),
            hjust=0, vjust=.5, size = 4, colour = 'black',
            fontface = 'bold') +
  ggtitle("Bar Plot of Bedrooms and Median Price") +
  xlab("Bedrooms") +
  ylab("Median Price") +
  scale_y_continuous(limits = c(0,150e4), labels = comma) +
  coord_flip() +
  theme_bw()
```



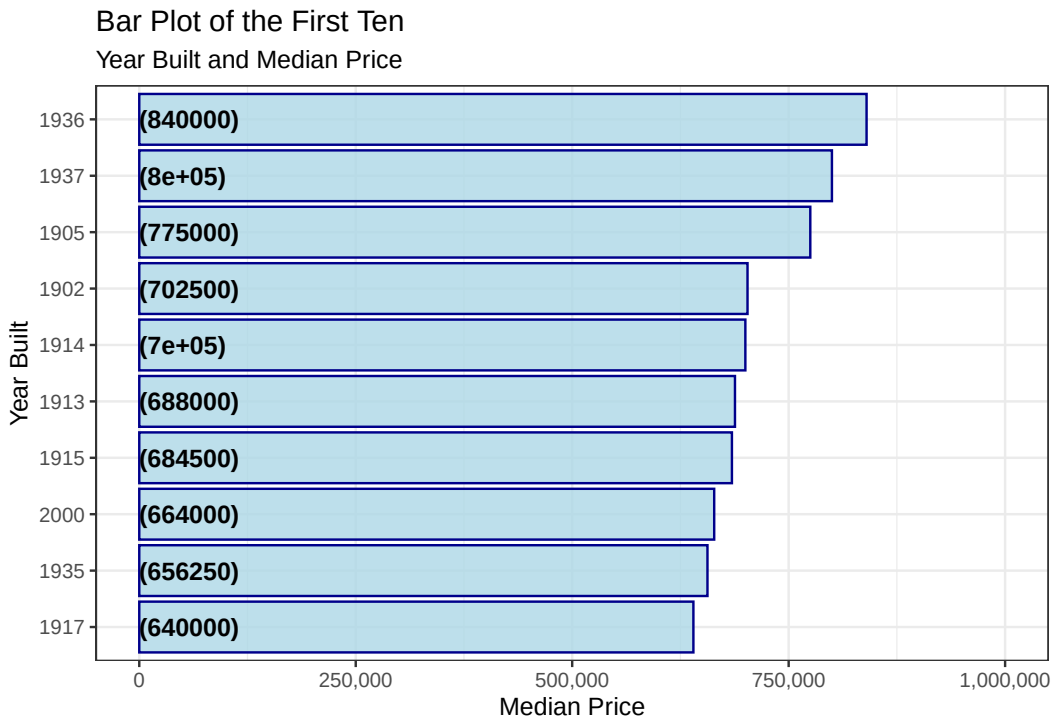
Second bar plot of `bathrooms` and median `price` in descending order.

```
data2 %>%
  group_by(bathrooms) %>%
  summarise(PriceMedian = median(price, na.rm = TRUE)) %>%
  ungroup() %>%
  mutate(bathrooms = reorder(bathrooms,PriceMedian)) %>%
  arrange(desc(PriceMedian)) %>%
  ggplot(aes(x = bathrooms,y = PriceMedian)) +
  geom_bar(stat='identity', color = "darkblue", alpha = 0.8, fill = "lightblue") +
  geom_text(aes(x = bathrooms, y = 1, label = paste0("(",PriceMedian,")",sep="")),
            hjust=0, vjust=.5, size = 4, colour = 'black',
            fontface = 'bold') +
  ggtitle("Bar Plot of Bathrooms and Median Price") +
  xlab("Bathrooms") +
  ylab("Median Price") +
  scale_y_continuous(limits = c(0,5e6), labels = comma) +
  coord_flip() +
  theme_bw()
```



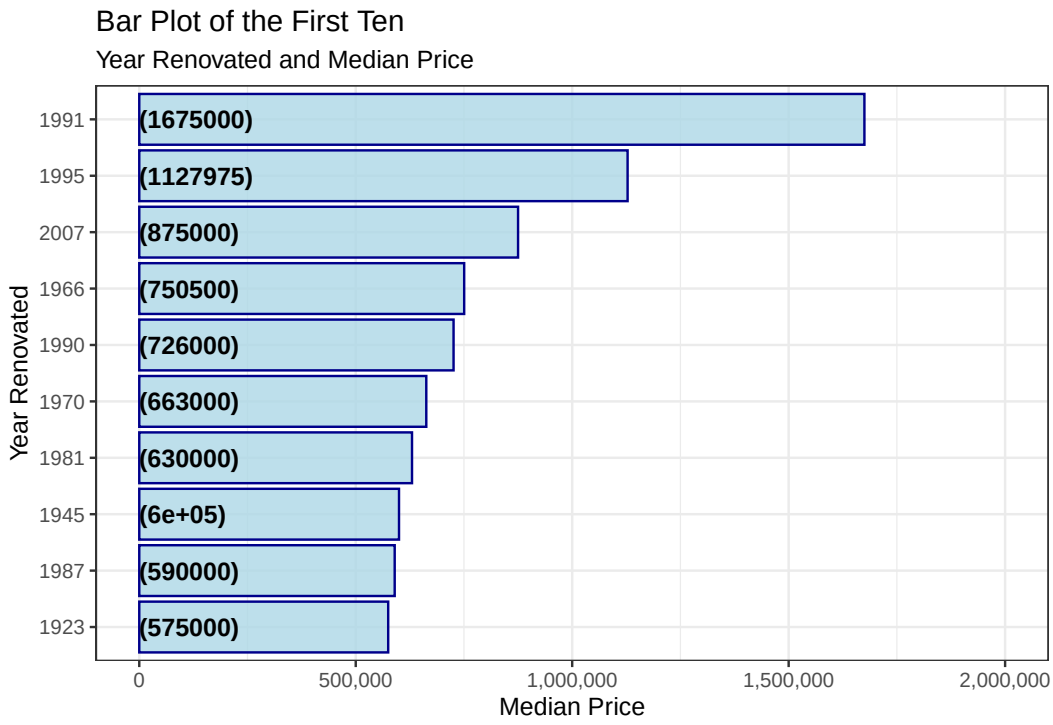
Third bar plot of the first ten `yr_built` and median `price` in descending order.

```
data2 %>%
  group_by(yr_built) %>%
  summarise(PriceMedian = median(price, na.rm = TRUE)) %>%
  ungroup() %>%
  mutate(yr_built = reorder(yr_built,PriceMedian)) %>%
  arrange(desc(PriceMedian)) %>%
  head(10) %>%
  ggplot(aes(x = yr_built,y = PriceMedian)) +
  geom_bar(stat='identity', color = "darkblue", alpha = 0.8, fill = "lightblue") +
  geom_text(aes(x = yr_built, y = 1, label = paste0("(",PriceMedian,")",sep="")),
            hjust=0, vjust=.5, size = 4, colour = 'black',
            fontface = 'bold') +
  ggtitle("Bar Plot of the First Ten",
          subtitle = "Year Built and Median Price") +
  xlab("Year Built") +
  ylab("Median Price") +
  scale_y_continuous(limits = c(0,1e6), labels = comma) +
  coord_flip() +
  theme_bw()
```



And the fourth bar plot of the first ten `yr_renovated` and median `price` in descending order.

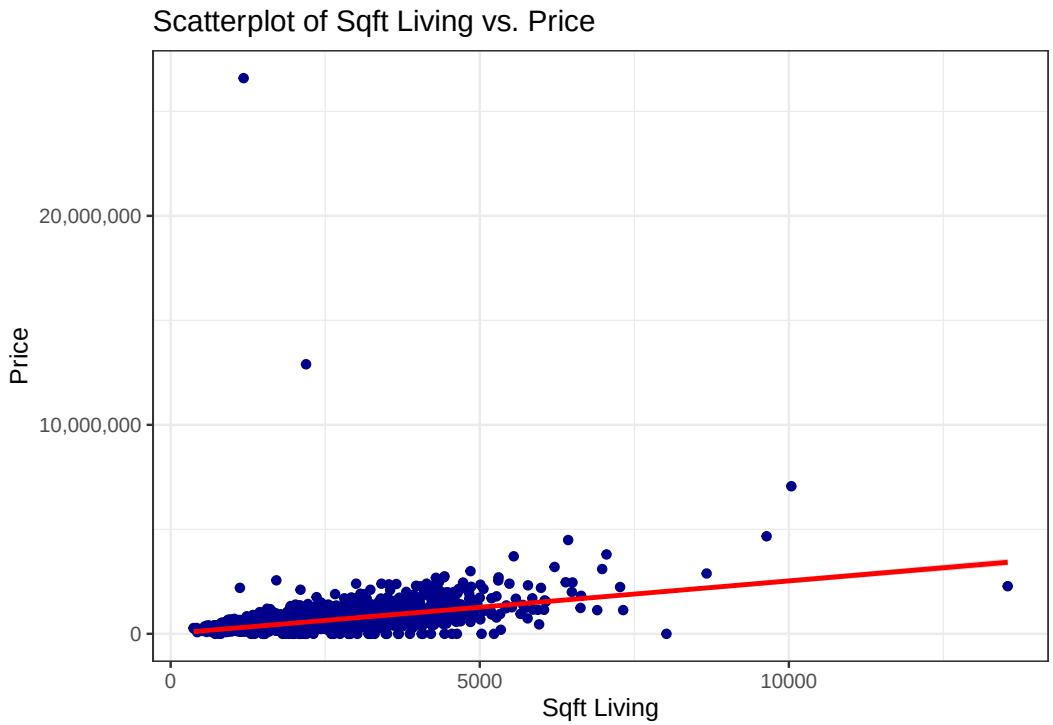
```
data2 %>%
  group_by(yr_renovated) %>%
  summarise(PriceMedian = median(price, na.rm = TRUE)) %>%
  ungroup() %>%
  mutate(yr_renovated = reorder(yr_renovated, PriceMedian)) %>%
  arrange(desc(PriceMedian)) %>%
  head(10) %>%
  ggplot(aes(x = yr_renovated, y = PriceMedian)) +
  geom_bar(stat='identity', color = "darkblue", alpha = 0.8, fill = "lightblue") +
  geom_text(aes(x = yr_renovated, y = 1, label = paste0("(",PriceMedian,")", sep="")),
            hjust = 0, vjust = .5, size = 4, colour = 'black',
            fontface = 'bold') +
  ggtitle("Bar Plot of the First Ten",
          subtitle = "Year Renovated and Median Price") +
  xlab("Year Renovated") +
  ylab("Median Price") +
  scale_y_continuous(limits = c(0,2e6), labels = comma) +
  coord_flip() +
  theme_bw()
```


Bar Plot of the First Ten
Year Renovated and Median Price
Year Renovated
1991
1995
2007
1966
1990
1970
1981
1945
1987
1923
(1675000)
(1127975)
(875000)
(750500)
(726000)
(663000)
(630000)
(6e+05)
(590000)
(575000)
0
500,000
1,000,000
1,500,000
2,000,000
Median Price

Now, we are going to visualize four scatterplots about the `sqft_living`, `sqft_lot`, `sqft_above` and `sqft_basement` variables against `price`.

Scatterplot `sqft_living` against `price`.

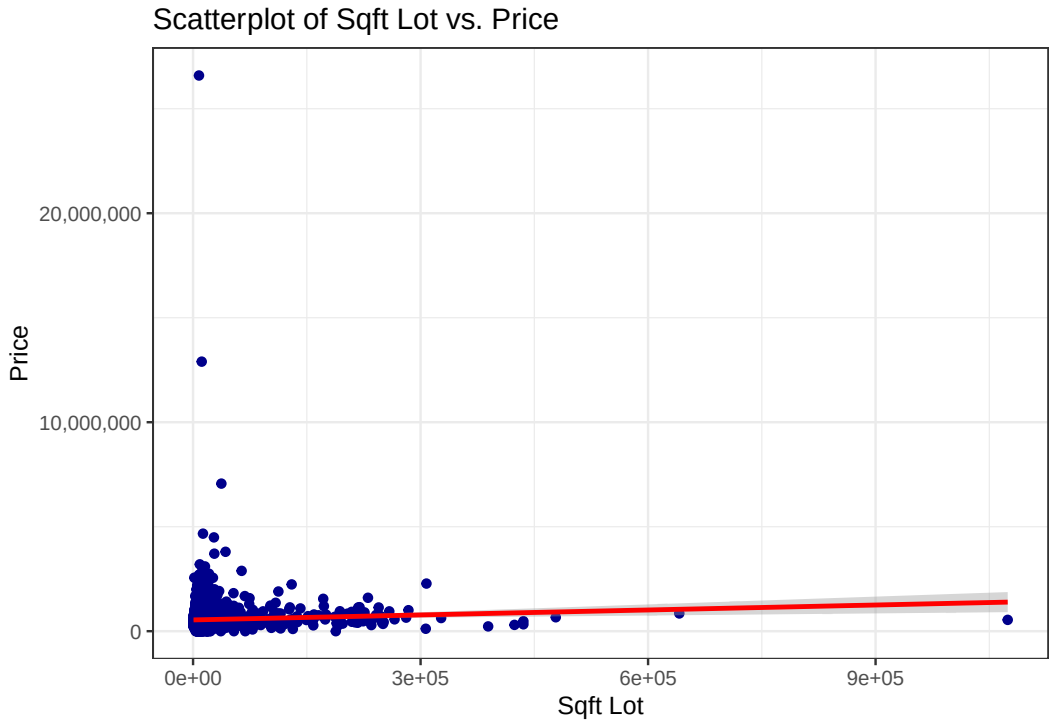
```
data2 %>%
  filter(!is.na(price)) %>%
  filter(!is.na(sqft_living)) %>%
  ggplot(aes(x = sqft_living, y = price)) +
  geom_point(color = "darkblue") +
  stat_smooth(aes(x = sqft_living, y = price), method = "lm", color = "red") +
  ggtitle("Scatterplot of Sqft Living vs. Price") +
  xlab("Sqft Living") +
  ylab("Price") +
  scale_y_continuous(labels = comma) +
  theme_bw()
```



Scatterplot `sqft_lot` vs. `price`.

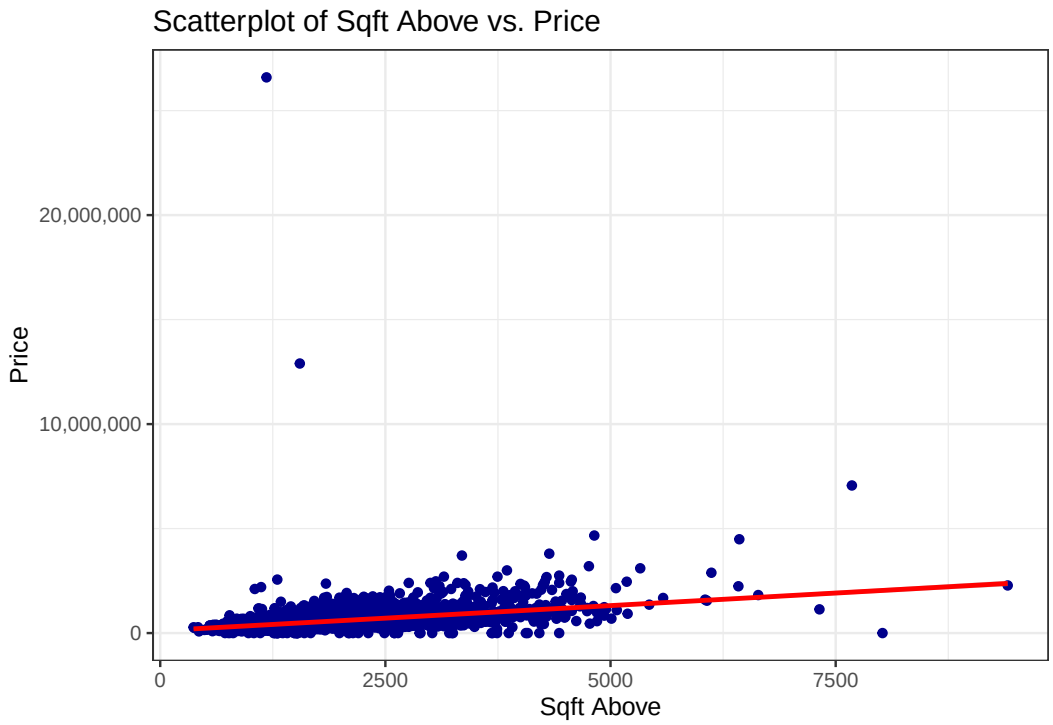
```
data2 %>%
  filter(!is.na(price)) %>%
  filter(!is.na(sqft_lot)) %>%
  ggplot(aes(x = sqft_lot, y = price)) +
  geom_point(color = "darkblue") +
```

```
  stat_smooth(aes(x = sqft_lot, y = price), method = "lm", color = "red") +
  ggtitle("Scatterplot of Sqft Lot vs. Price") +
  xlab("Sqft Lot") +
  ylab("Price") +
  scale_y_continuous(labels = comma) +
  theme_bw()
```



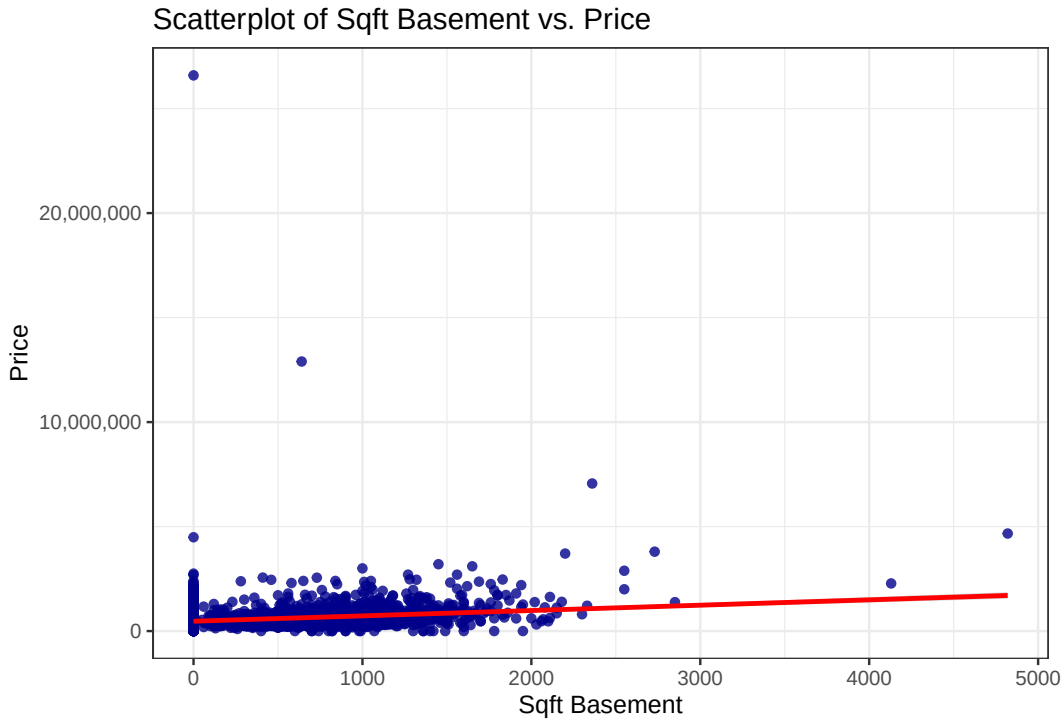
Scatterplot `sqft_above` against `price`.

```
data2 %>%
  filter(!is.na(price)) %>%
  filter(!is.na(sqft_above)) %>%
  ggplot(aes(x = sqft_above, y = price))+
  geom_point(color = "darkblue")+
  stat_smooth(aes(x = sqft_above, y = price), method = "lm", color = "red")+
  ggtitle("Scatterplot of Sqft Above vs. Price") +
  xlab("Sqft Above") +
  ylab("Price") +
  scale_y_continuous(labels = comma) +
  theme_bw()
```



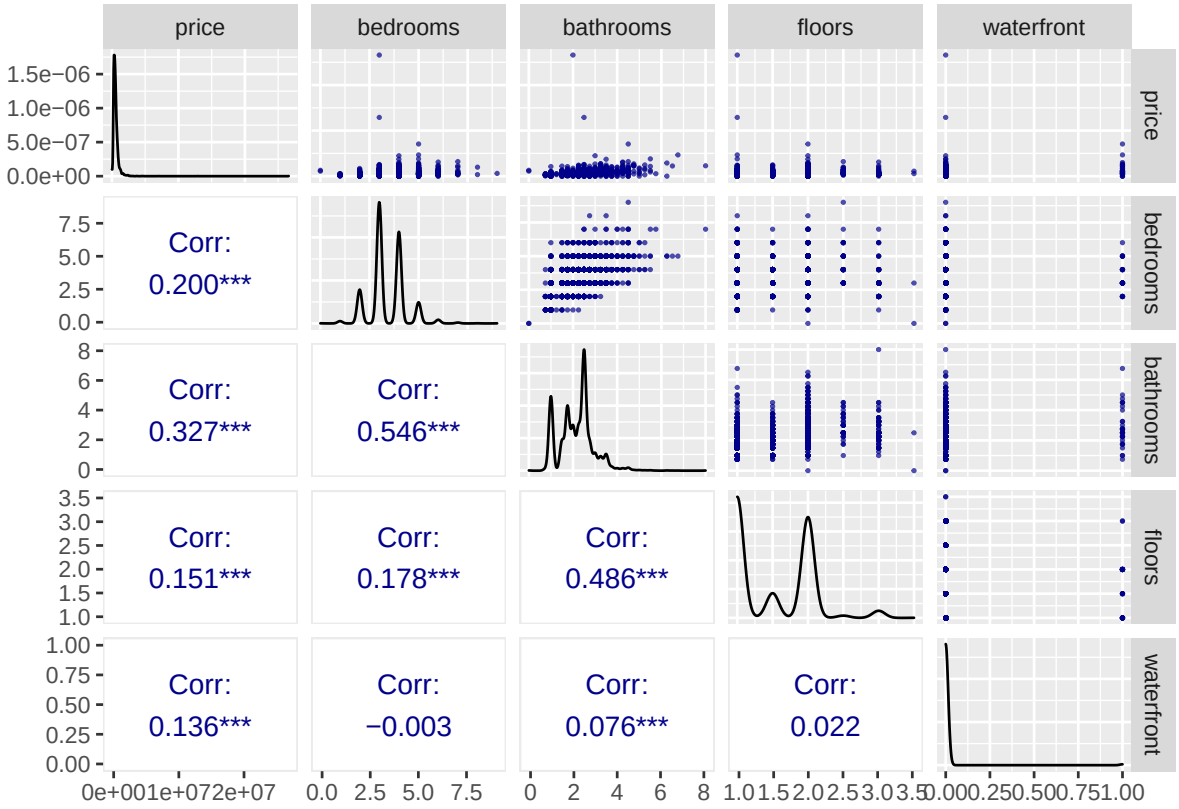
Scatterplot `sqft_basement` vs. `price`.

```
data2 %>%
  filter(!is.na(price)) %>%
  filter(!is.na(sqft_basement)) %>%
  ggplot(aes(x = sqft_basement, y = price))+
  geom_point(color = "darkblue", alpha = 0.8)+
  stat_smooth(aes(x = sqft_basement,  y = price), method = "lm", color = "red")+
  ggtitle("Scatterplot of Sqft Basement vs. Price") +
  xlab("Sqft Basement") +
  ylab("Price") +
  scale_y_continuous(labels = comma) +
  theme_bw()
```



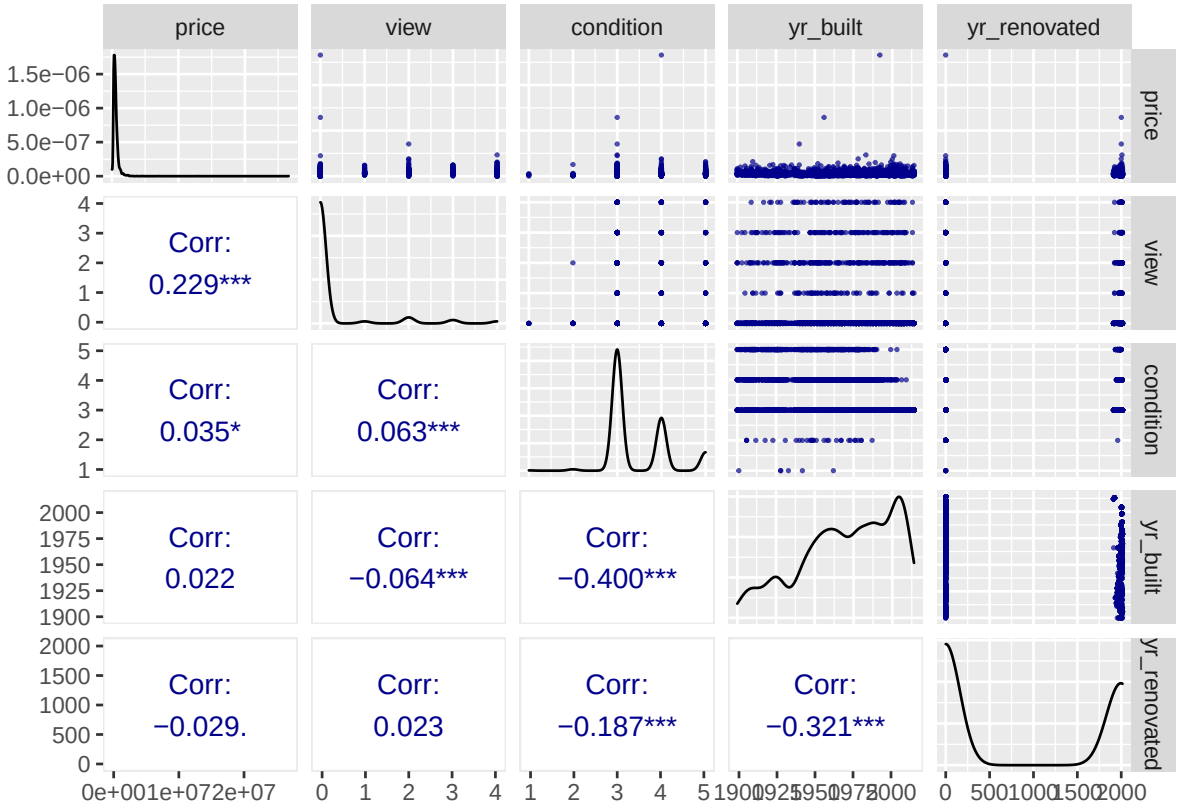
Finally, we plot the correlations from `c(2,3,4,7,8)` columns with the function `ggpairs` of the `GGally` package.

```
data2 %>%
  select(c(2:4, 7, 8)) %>%
  ggpairs(.,
          lower = list(continuous = wrap("cor", size = 4,
                                         alignPercent = 0.8, colour = "darkblue")),
          upper = list(continuous = wrap("points", alpha = 0.7,
                                         size = 0.3, colour = "darkblue"))
          )
```



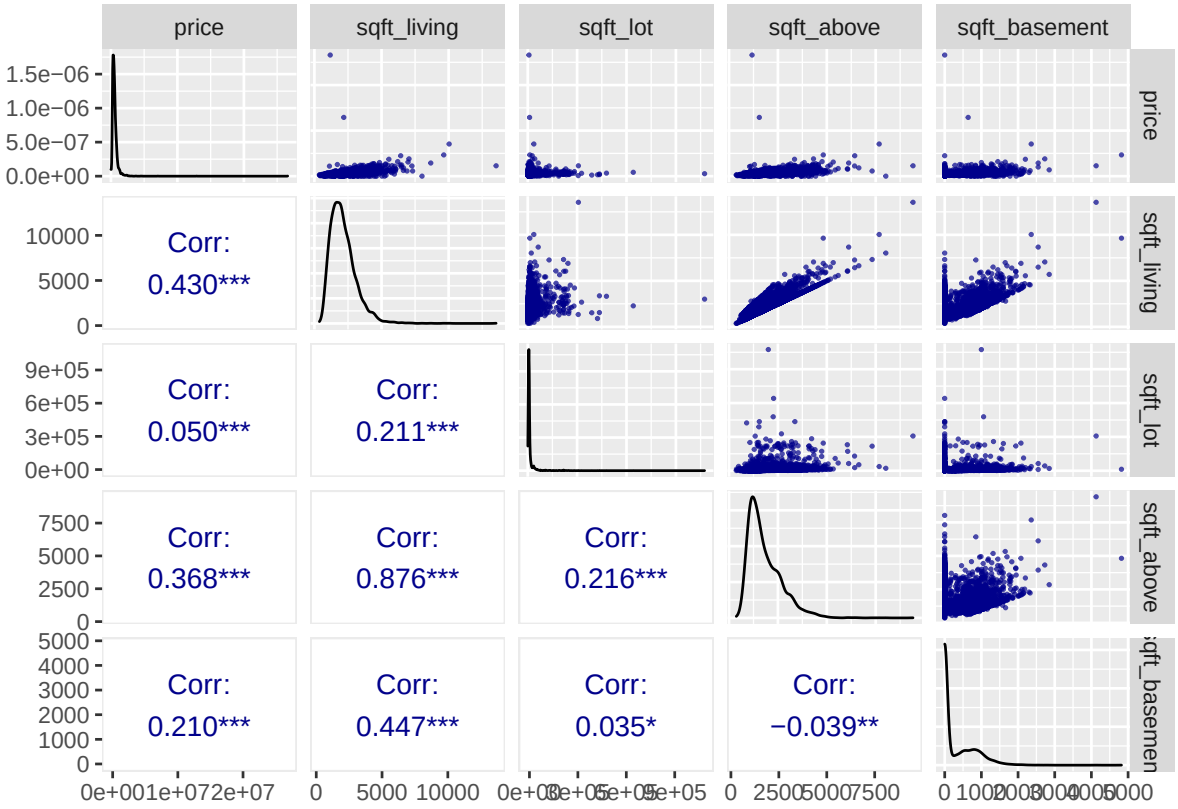
Now, we plot the correlations from `c(2,9,10,13,14)` columns with the function `ggpairs` of the `GGally` package.

```
data2 %>%
  select(c(2, 9:10, 13:14)) %>%
  ggpairs(.,
          lower = list(continuous = wrap("cor", size = 4,
                                         alignPercent = 0.8, colour = "darkblue")),
          upper = list(continuous = wrap("points", alpha = 0.7,
                                         size = 0.3, colour = "darkblue"))
          )
```



And plot the correlations from `c(2,5,6,11,12)` columns with `ggpairs`.

```
data2 %>%
  select(c(2,5,6,11,12)) %>%
  ggpairs(.,
          lower = list(continuous = wrap("cor", size = 4,
                                         alignPercent = 0.8, colour = "darkblue")),
          upper = list(continuous = wrap("points", alpha = 0.7,
                                         size = 0.3, colour = "darkblue"))
          )
```


price
sqft_living
sqft_lot
sqft_above
sqft_basement
Corr:
0.430***
Corr:
0.050***
Corr:
0.211***
Corr:
0.368***
Corr:
0.876***
Corr:
0.216***
Corr:
0.210***
Corr:
0.447***
Corr:
0.035*
Corr:
−0.039**

2.3 Preparing and Cleaning the Datasets

This phase is **Scrub data**, but, before we start building our model, we need to create two new datasets one with outliers and other without them.

2.3.1 Cleaning and Creating the Datasets

We proceed to create the `outliers` dataset with make a copy from `data2` by removing these columns `date`, `waterfront`, `view`, `street`, `city`, `statezip`, `country` and `pricebin`.

```
#++++++++ Create the outliers set ++++++++++
#
# remove these columns `date`, `waterfront`, `view`,
# `street`, `city`, `statezip`, `country` and `pricebin`
outliers <- data2 %>%
  select(-c("date", "waterfront", "view", "street", "city", "statezip", "country", "pricebin"))
```

The number of observations and columns that have the dataframe `outliers`.

```
cat('The dataframe called outliers has',
    dim(outliers)[1], 'rows and', dim(outliers)[2], 'columns.')
```

```
## The dataframe called outliers has 4600 rows and 11 columns.
```

Next, create the `no_outliers` dataset from `data2` by removing these columns `date`, `waterfront`, `view`, `street`, `city`, `statezip`, `country` and `pricebin`. Create the `detect_outlier` and `remove_outlier` functions to remove outliers from multiple columns with the Interquartile range method suggets by (GeeksforGeeks, 2022). Then, assign the names of the columns with outliers `price`, `sqft_lot`, `sqft_basement` and `yr_renovated` to the `col_num` variable, apply the function `remove_outlier` and remove these objects from the Global Environment `data`, `data2`, `p1`, `p2`, `p3`, `p4`, `p5`, and `p6`.

```
#++++++++ Create the `no_outliers` set ++++++++++
#
# remove these columns `date`, `waterfront`, `view`,
# `street`, `city`, `statezip`, `country` and `pricebin`
no_outliers <- data2 %>%
  select(-c("date", "waterfront", "view", "street", "city", "statezip", "country", "pricebin"))

#++ Remove Outliers from Multiple Columns with the Interquartile range method ++
#
# create detect_outlier function
detect_outlier <- function(x) {
```

```
  # calculate first quantile
  Quantile1 <- quantile(x, probs=0.25)

  # calculate third quantile
  Quantile3 <- quantile(x, probs=0.75)

  # calculate inter quartile range
  iqr <- Quantile3-Quantile1

  # get minimums values
  min_values <- Quantile1 - (iqr*1.5)

  # get maximums values
  max_values <- Quantile3 + (iqr*1.5)

  # return true or false
  x > max_values | x < min_values
}

# create remove_outlier function
remove_outlier <- function(dataframe,
                           columns) {

  # for loop to traverse in columns vector
  for (col in columns) {

    # remove observation if it satisfies outlier function
    dataframe <- dataframe[!detect_outlier(dataframe[[col]]), ]
  }

  dataframe
}


# Assign the names of the columns with outliers `price`,
# `sqft_lot`, `sqft_basement` and `yr_renovated`
col_num <- c("price", "sqft_lot", "sqft_basement", "yr_renovated")

# apply the function `remove_outlier`
no_outliers <- remove_outlier(no_outliers, col_num)

# Remove these objects from the Global Environment
rm(data, data2, p1, p2, p3, p4, p5, p6)
```

To know the number of observations and columns that have the dataframe `no_outliers`.

```
cat('The dataframe called no_outliers has',
    dim(no_outliers)[1], 'rows and', dim(no_outliers)[2], 'columns.')
```

```
## The dataframe called no_outliers has 3806 rows and 11 columns.
```

2.3.2 Preparing the Train and Test Datasets

First, we must split the `outliers` dataset in two new datasets, we do that in the following way: Set the seed depending on the version of `R` to create the `train_outliers` and `test_outliers` datasets, where, `test_outliers` set will be 10% and `train_outliers` set 90% of `outliers` dataset.

```
######+++ Create the train_outliers set and test_outliers set from the outliers set +++
#
#
# Set the seed according to the R version
if (paste(v$major, v$minor, sep = ".") < "3.6.0"){
  print("version of R is MINOR to 3.6.0, use set.seed(1)")
  # if using R 3.5 or earlier, use `set.seed(1)`:
  set.seed(1)
}else{
  print("version of R is MAJOR or equal to 3.6.0, use set.seed(1, sample.kind=Rounding)")
  # if using R 3.6 or later:
  set.seed(1, sample.kind="Rounding")
}
```

```
## [1] "version of R is MAJOR or equal to 3.6.0, use set.seed(1, sample.kind=Rounding)"
```

```
######+++ Create `train_outliers` set, `test_outliers` set from the `outliers` set +++
#
# We must split the `outliers` set in 2 parts: the training and test sets.
#
# `test_outliers` set will be 10% and `train_outliers` set 90% of `outliers` data
test_index <- createDataPartition(y = outliers$price, times = 1, p = 0.1, list = FALSE)
train_outliers <- outliers[-test_index,]
test_outliers <- outliers[test_index,]


# Remove these objects from the Global Environment
rm(test_index, outliers)
```

We repeat the same process with the dataset `no_outliers` to create two new datasets.

```
###+++ Create the train_no_outliers set and test_no_outliers set from the no_outliers set +++
#
#
# Set the seed according to the R version
if (paste(v$major, v$minor, sep = ".") < "3.6.0"){
  print("version of R is MINOR to 3.6.0, use set.seed(1)")
  # if using R 3.5 or earlier, use `set.seed(1)`:
  set.seed(1)
}else{
  print("version of R is MAJOR or equal to 3.6.0, use set.seed(1, sample.kind=Rounding)")
  # if using R 3.6 or later:
  set.seed(1, sample.kind="Rounding")
}
```

```
## [1] "version of R is MAJOR or equal to 3.6.0, use set.seed(1, sample.kind=Rounding)"
```

```
###+++ Create `train_no_outliers` set, `test_no_outliers` set from the `no_outliers` set +++
#
# We must split the `no_outliers` set in 2 parts: the training and test sets.
#
# `test_no_outliers` set will be 10% and `train_no_outliers` set 90% of `no_outliers` data
test_index <- createDataPartition(y = no_outliers$price, times = 1, p = 0.1, list = FALSE)
train_no_outliers <- no_outliers[-test_index,]
test_no_outliers <- no_outliers[test_index,]


# Remove these objects from the Global Environment
rm(test_index, no_outliers, col_num, v, detect_outlier, remove_outlier)
```

And finally, the `train_outliers`, `test_outliers`, `train_no_outliers` and `test_no_outliers` objects (datasets) are saved in the file `cp_house_price.rda` in the `rdas` directory, this last process has the objective of being able to help carry out and facilitate collaborative work, share publications or reproducible research.

```
###+++ save objects train_outliers`, `test_outliers` +++
###+++ `train_no_outliers` and `test_no_outliers +++
#########+ in the file path: rdas/cp_house_price.rda +++++++++
save(train_outliers, test_outliers, train_no_outliers, test_no_outliers,
     file = "./rdas/cp_house_price.rda")
```

The following code verify if the `cp_house_price.rda` file exists in the `rdas` directory, if it exists it prints a message that it already exists and this process is finished. If it does not exist, then, a message is printed that it does not exist, it is downloaded from gitlab to `rdas` directory. Remove the `train_outliers`, `test_outliers`, `train_no_outliers` and `test_no_outliers` objects from the Global Environment if they exist and load the the `cp_house_price.rda` file.

```
# Verify if the "cp_house_price.rda" file exists in the "rdas" directory
if(file.exists(paste0(wd, "/rdas/cp_house_price.rda"))==TRUE){
  print("File cp_house_price.rda exists already")
}else{
  print("File cp_house_price.rda does NOT exist...downloading")
  # Download the file from gitlab to "rdas" directory
  download.file("https://gitlab.com/saulcol/rdas/-/raw/main/cp_house_price.rda",
                paste0(wd, "/rdas/cp_house_price.rda"))
}
```

```
## [1] "File cp_house_price.rda exists already"
```

```
# Remove these objects from the Global Environment if they exist
if(exists("train_outliers")) rm("train_outliers", envir = globalenv())
if(exists("test_outliers")) rm("test_outliers", envir = globalenv())
if(exists("train_no_outliers")) rm("train_no_outliers", envir = globalenv())
if(exists("test_no_outliers")) rm("test_no_outliers", envir = globalenv())

# load objects `train_outliers`, `test_outliers`, `train_no_outliers`
# and `test_no_outliers` from the file path:
# rdas/cp_house_price.rda. Take a few seconds
load("./rdas/cp_house_price.rda")
```

2.4 Modeling approach

2.4.1 Linear Regression Model

We start by creating a linear regression model in the dataset with outliers, and later we build another linear regression in the dataset without outliers to compare the performance. The general formula for those models is based on the one proposed by (James et al., 2021, p. 72) with p distinct predictors:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p + \varepsilon$$

where Y is our dependent variable (random variable) that needs to be predicted, β_0 is the Y intercept, X_j represents the jth predictor (independent variables) that are used to predict our resultant dependent value, also, β_j quantifies the association between the predictor and response variables. β_1, β_2, β_n are the slopes and ε are the independent errors sampled from the same distribution centered at 0 (random variation).

We use the `lm()` function to build our models.

2.4.2 Random Forest approach for Regression Model

Random forest uses or applies an ensemble technique that trains many decision trees using the bagging method. Broadly speaking, these are the steps carried out by the random forest algorithm described by (Saini, 2022):

1. *"We first make subsets of our original data. We will do row sampling and feature sampling that means we'll select rows and columns with replacement and create subsets of the training dataset"*.
2. *"We create an individual decision tree for each subset we take"*.
3. *"Each decision tree will give an output"*.
4. *"Final output is considered based on Majority Voting if it's a classification problem and average if it's a regression problem"*.

Now, to create a random forest regression model we need to fulfill a requirement that our **response variable** be **continuous**, in our case is the `price` variable. And apply the `randomForest` function from the `randomForest` package in two datasets, one with outliers and other without them.

Chapter 3

Results

It is time to describe the procedure **Model data**, here, we build the four models according to the results obtained or produced during the data exploration phase. We must also train and test them.

3.1 Defining the RMSE Function to Evaluate the Model

First, we define the loss function, in this case, the **Root Mean Squared Error (RMSE)** function, that we use it in the linear regression models.

```
# Define the Root Mean Squared Error (RMSE) function
RMSE <- function(true_values, predicted_values){
  sqrt(mean((true_values - predicted_values)^2))
}
```

3.2 Model 1: Linear Regression Model with Outliers

We fit a linear regression model with the `lm()` function using the `train_outliers` dataset and view its summary.

```
# Fit the model
lm1 <- lm(price ~ bedrooms + bathrooms + sqft_living + sqft_lot + floors + condition +
                sqft_above + sqft_basement + yr_built + yr_renovated, data = train_outliers)

# View its summary
lm1_summary <- (summary(lm1))
lm1_summary
```

```
##
## Call:
## lm(formula = price ~ bedrooms + bathrooms + sqft_living + sqft_lot +
##     floors + condition + sqft_above + sqft_basement + yr_built +
##     yr_renovated, data = train_outliers)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2230958  -134298   -22497    83502 26339359
##
## Coefficients: (1 not defined because of singularities)
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)    5.027e+06  7.588e+05   6.624 3.94e-11 ***
## bedrooms      -6.517e+04  1.133e+04  -5.751 9.51e-09 ***
## bathrooms      6.039e+04  1.859e+04   3.248  0.00117 **
## sqft_living    2.520e+02  2.265e+01  11.126  < 2e-16 ***
## sqft_lot      -6.851e-01  2.310e-01  -2.965  0.00304 **
## floors         4.191e+04  2.046e+04   2.048  0.04060 *
## condition      3.081e+04  1.442e+04   2.136  0.03277 *
## sqft_above     2.225e+01  2.319e+01   0.959  0.33736
## sqft_basement         NA         NA      NA       NA
## yr_built      -2.603e+03  3.782e+02  -6.881 6.82e-12 ***
## yr_renovated   8.427e+00  9.518e+00   0.885  0.37599
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 520900 on 4130 degrees of freedom
## Multiple R-squared:  0.1932, Adjusted R-squared:  0.1914
## F-statistic: 109.9 on 9 and 4130 DF,  p-value: < 2.2e-16
```

From the results above, we can fit a new model without the `sqft_basement` variable and to see what happens.

```
# Fit new model without `sqft_basement`
lm1 <- lm(price ~ bedrooms + bathrooms + sqft_living + sqft_lot + floors + condition +
                sqft_above + yr_built + yr_renovated, data = train_outliers)

# View its summary
lm1_summary <- (summary(lm1))
lm1_summary
```

```
##
## Call:
## lm(formula = price ~ bedrooms + bathrooms + sqft_living + sqft_lot +
##     floors + condition + sqft_above + yr_built + yr_renovated,
##     data = train_outliers)
##
```

```
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2230958  -134298   -22497    83502 26339359
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   5.027e+06  7.588e+05   6.624 3.94e-11 ***
## bedrooms     -6.517e+04  1.133e+04  -5.751 9.51e-09 ***
## bathrooms     6.039e+04  1.859e+04   3.248  0.00117 **
## sqft_living   2.520e+02  2.265e+01  11.126  < 2e-16 ***
## sqft_lot     -6.851e-01  2.310e-01  -2.965  0.00304 **
## floors        4.191e+04  2.046e+04   2.048  0.04060 *
## condition     3.081e+04  1.442e+04   2.136  0.03277 *
## sqft_above    2.225e+01  2.319e+01   0.959  0.33736
## yr_built     -2.603e+03  3.782e+02  -6.881 6.82e-12 ***
## yr_renovated  8.427e+00  9.518e+00   0.885  0.37599
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 520900 on 4130 degrees of freedom
## Multiple R-squared:  0.1932, Adjusted R-squared:  0.1914
## F-statistic: 109.9 on 9 and 4130 DF,  p-value: < 2.2e-16
```

To determinate how good this model fits the data, we can observe the **Multiple R-squared** metric. It measures how strong is the linear relationship between the predictor variables and the response variable. Its value is 0.1931795, as we expected it is low, but most of the predictor variables are statistically significant except `sqft_above` and `yr_renovated`. Also, we can get the **R-squared** from the Multiple R-squared like this `(lm1_summary$r.squared)^2` equal to 0.0373183. It indicates that 3.73% of the variance in `price` can be explained by the predictors in the model.

Now, we predict the target value in the `test_outliers` dataset, create the `scores` table to store the **RMSE** of the `lm1` model and display the scores.

```
# Predict the target value in the test_outliers set
pred1 <- predict(lm1, newdata = test_outliers)

# Create the scores table to store the error scores
scores <- tibble(Method = "RMSE lm1 outliers",
                 RMSE = RMSE(test_outliers$price, pred1))

# Display the scores table
scores %>%
  as_hux() %>%
  set_font_size(9) %>%
  set_tb_padding(2) %>%
```

```
  set_col_width(c(.7, .1)) %>%
  set_number_format(everywhere, 2, 7) %>%
  set_latex_float("h!") %>%
  theme_basic()
```

Method	RMSE
RMSE lm1 outliers	276244.5579709

We compare the minimum, mean and maximum values from `test_outliers$price` (actual values) with the RMSE obtained in this table.

```
# Compare the Min, Mean, Max from `test_outliers$price`
# with the Rmse obtained in this the table
test_outliers %>%
  pivot_longer(cols = c(1),
               names_to = 'Variable',
               values_to = 'value') %>%
  group_by(Variable) %>%
  summarise(Min = min(value),
            Mean = mean(value),
            Max = max(value)) %>%
  mutate(Rmse = RMSE(test_outliers$price, pred1)) %>%
  as_hux() %>%
  set_font_size(9) %>%
  set_tb_padding(2) %>%
  set_col_width(c(.1, .1, .1, .1, .1)) %>%
  set_number_format(everywhere, 2:5, 3) %>%
  set_latex_float("h!") %>%
  theme_basic()
```

Variable	Min	Mean	Max	Rmse
price	0.000	549438.213	3800000.000	276244.558

The value of RMSE indicates that our model is on average wrong by 276,244.558 units of `price`. We continue to look at the first six rows of the actual (observed) and the predicted values of `price`.

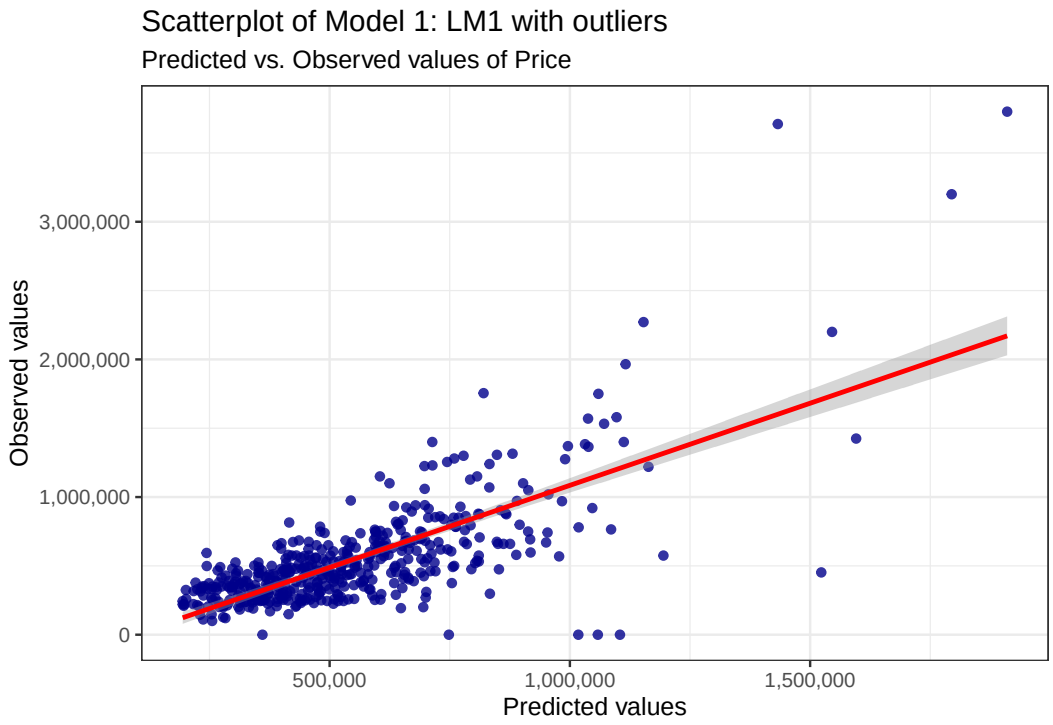
```
# look at the first six rows of the actual (observed)
# and the predicted values of `price`
data.frame(cbind(actuals=test_outliers$price, predicteds = pred1)) %>%
  head() %>%
  as_hux() %>%
```

```
  set_font_size(9) %>%
  set_tb_padding(2) %>%
  set_col_width(c(.1, .1)) %>%
  set_number_format(everywhere, 1:2, 2) %>%
  set_latex_float("h!") %>%
  theme_basic()
```

actuals	predicteds
407500.00	485392.84
783500.00	762488.22
650000.00	600112.11
220000.00	194347.09
416286.00	367213.55
279900.00	412397.19

We plot the observed (actual) versus predicted values of `price`.

```
test_outliers %>%
  ggplot(aes(pred1, price)) +
  geom_point(color = "darkblue", alpha = 0.8)+
  stat_smooth(aes(x = pred1,  y = price), method = "lm", color = "red")+
  ggtitle("Scatterplot of Model 1: LM1 with outliers",
          subtitle = "Predicted vs. Observed values of Price") +
  xlab("Predicted values") +
  ylab("Observed values") +
  scale_x_continuous(labels = comma) +
  scale_y_continuous(labels = comma) +
  theme_bw()
```



As we can see, the first model does not do a good job in general terms, because the outliers created some noise when we built it, and that affects it both in its performance and when we use it to predict.

3.3 Model 2: Random Forest approach for Regression Model with Outliers

We fit a random forest regression model with the `randomForest()` function belonging to the package `randomForest` using the `train_outliers` dataset and display the content of the model.

```
# Fit the model
rf1 <- randomForest(price ~ ., data = train_outliers, ntree = 500,
                        importance = TRUE, type = "regression")

# Display the content of the model
rf1
```

```
##
## Call:
##  randomForest(formula = price ~ ., data = train_outliers, ntree = 500,      importance = TRUE,
##                Type of random forest: regression
##                      Number of trees: 500
```

```
## No. of variables tried at each split: 3
##
##           Mean of squared residuals: 280129375027
##                     % Var explained: 16.5
```

We observe that **No. of variables tried at each split** is 3, that is correct, because `randomForest()` uses $p/3$, where p is the number of predictor variables, for regression and $\sqrt{p}$ for classification. In our case, we employ 10 predictors and we can calculate `10/3` is 3.3333333. We can change this number with the argument `mtry` if we want.

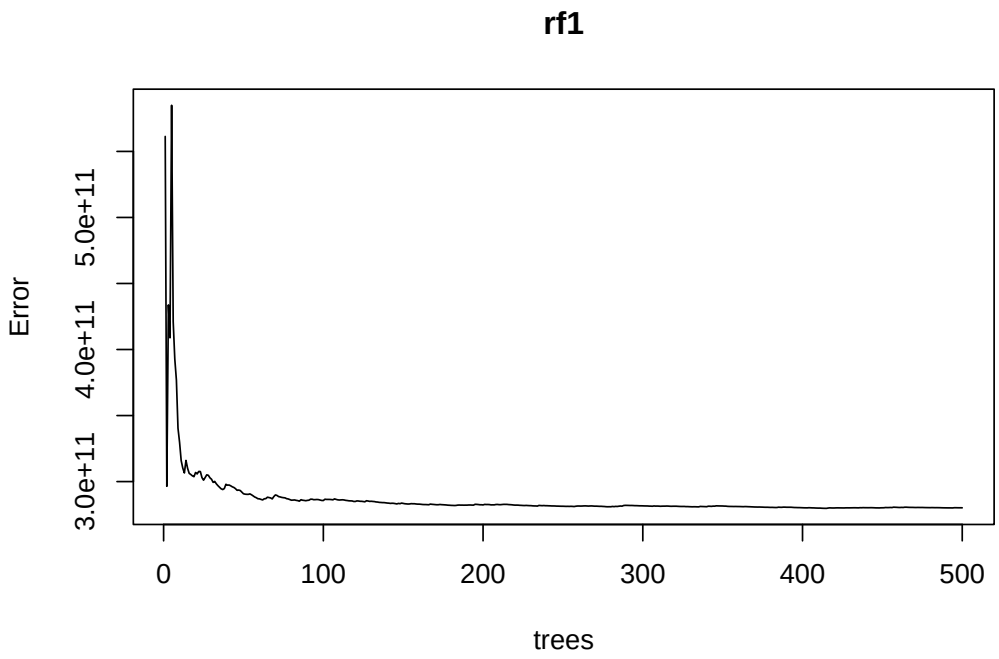
We predict the target value in the `test_outliers` dataset, update the `scores` table to store the **RMSE** of the `rf1` model and display the scores.

```
# Predict the target value in the test_outliers set
pred2 <- predict(rf1, newdata = test_outliers)

# Update the scores table
scores <- bind_rows(scores,
                    tibble(Method = "RMSE rf1 outliers",
                           RMSE = RMSE(test_outliers$price, pred2)))

# Display the scores table
scores %>%
  as_hux() %>%
  set_font_size(9) %>%
  set_tb_padding(2) %>%
  set_col_width(c(.7, .1)) %>%
  set_number_format(everywhere, 2, 7) %>%
  set_latex_float("h!") %>%
  theme_basic()
```

Method	RMSE
RMSE lm1 outliers	276244.5579709
RMSE rf1 outliers	293019.2976780

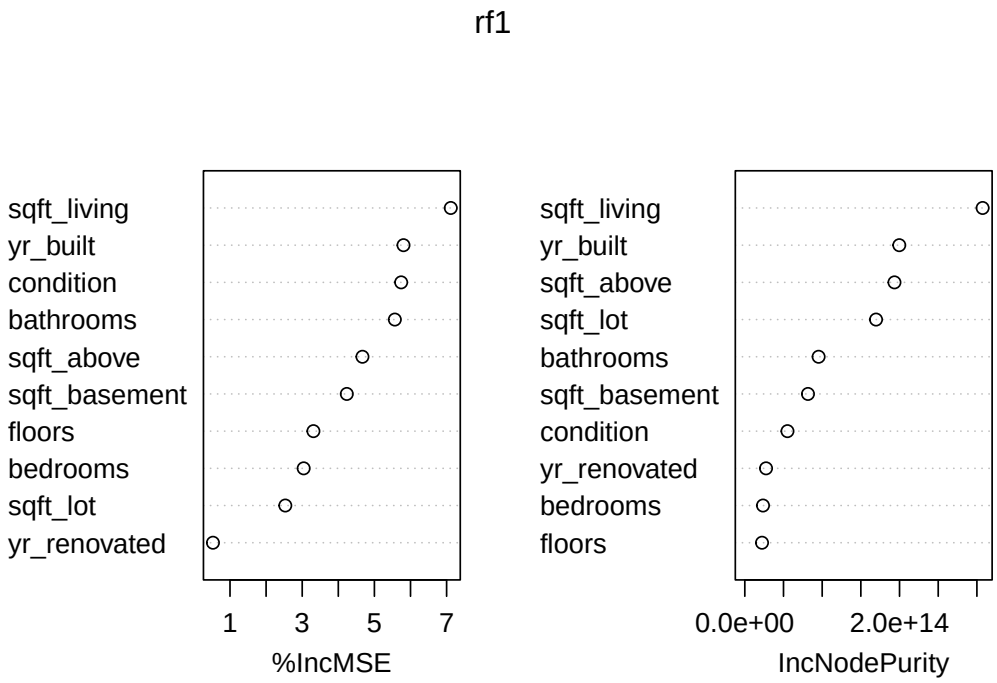
We show a plot of the test MSE (Errors) by number of trees.

```
# Plot of the test MSE by number of trees
plot(rf1)
```



Show the variable importance plot.

```
# show the variable importance plot
varImpPlot(rf1)
```



From the plot we see that `sqft_living` and `yr_built` are the more important variables follow by `sqft_above` and `bathrooms`.

We compare the minimum, mean and maximum values from `test_outliers$price` (actual values) with the RMSE obtained in this table.

```
# Compare the Min, Mean, Max from `test_outliers$price`
# with the Rmse obtained in this the table
test_outliers %>%
  pivot_longer(cols = c(1),
               names_to = 'Variable',
               values_to = 'value') %>%
  group_by(Variable) %>%
  summarise(Min = min(value),
            Mean = mean(value),
            Max = max(value)) %>%
  mutate(Rmse = RMSE(test_outliers$price, pred2)) %>%
  as_hux() %>%
  set_font_size(9) %>%
  set_tb_padding(2) %>%
  set_col_width(c(.1, .1, .1, .1, .1)) %>%
  set_number_format(everywhere, 2:5, 3) %>%
  set_latex_float("h!") %>%
  theme_basic()
```

Variable	Min	Mean	Max	Rmse
price	0.000	549438.213	3800000.000	293019.298

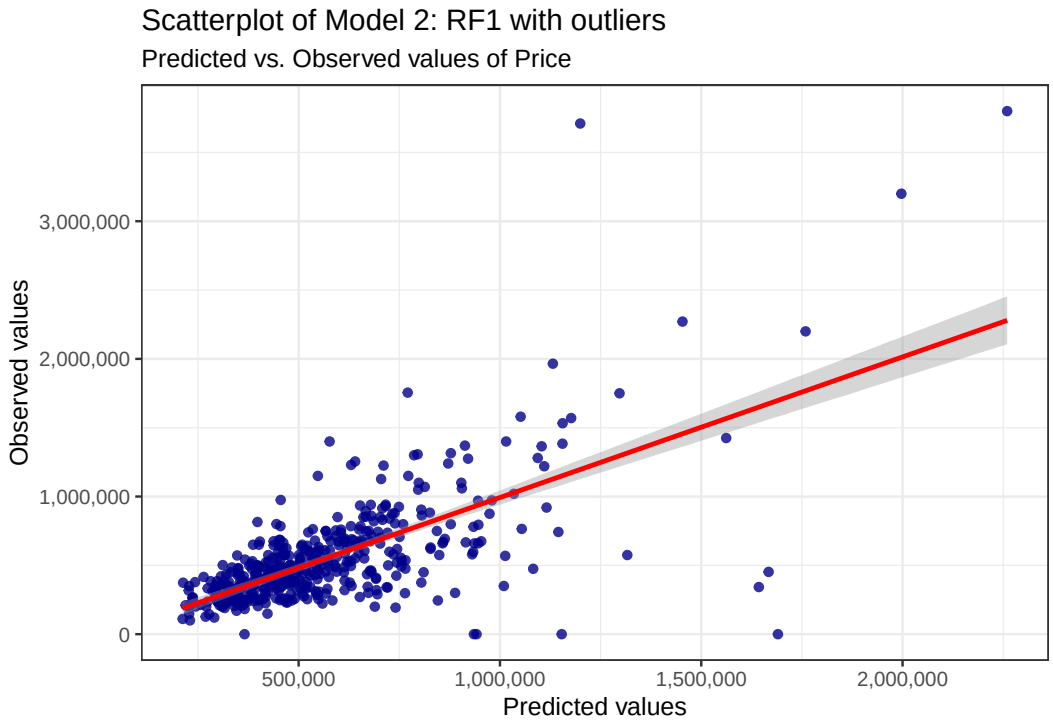
The value of RMSE indicates that our model is on average wrong by 293,019.298 units of `price`. We continue to look at the first six rows of the actual (observed) and the predicted values of `price`.

```
# look at the first six rows of the actual (observed)
# and the predicted values of `price`
data.frame(cbind(actuals=test_outliers$price, predicteds = pred2)) %>%
  head() %>%
  as_hux() %>%
  set_font_size(9) %>%
  set_tb_padding(2) %>%
  set_col_width(c(.1, .1)) %>%
  set_number_format(everywhere, 1:2, 2) %>%
  set_latex_float("h!") %>%
  theme_basic()
```

actuals	predicteds
407500.00	418664.56
783500.00	653715.56
650000.00	508960.54
220000.00	317722.01
416286.00	442290.04
279900.00	347826.90

We plot the observed (actual) versus predicted values of `price`.

```
test_outliers %>%
  ggplot(aes(pred2, price)) +
  geom_point(color = "darkblue", alpha = 0.8)+
  stat_smooth(aes(x = pred2,  y = price), method = "lm", color = "red")+
  ggtitle("Scatterplot of Model 2: RF1 with outliers",
          subtitle = "Predicted vs. Observed values of Price") +
  xlab("Predicted values") +
  ylab("Observed values") +
  scale_x_continuous(labels = comma) +
  scale_y_continuous(labels = comma) +
  theme_bw()
```



The second model is a little worse than the first one and does not do a good job in general terms.

3.4 Model 3: Linear Regression Model without Outliers

We fit a linear regression model with the `lm()` function using the `train_no_outliers` dataset and view its summary.

```
# Fit the model
lm2 <- lm(price ~ bedrooms + bathrooms + sqft_living + sqft_lot + floors + condition +
                sqft_above + sqft_basement + yr_built + yr_renovated, data = train_no_outliers)

# View its summary
lm2_summary <- (summary(lm2))
lm2_summary
```

```
##
## Call:
## lm(formula = price ~ bedrooms + bathrooms + sqft_living + sqft_lot +
##     floors + condition + sqft_above + sqft_basement + yr_built +
##     yr_renovated, data = train_no_outliers)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -886484 -102889   -5534   99122  755525
##
## Coefficients: (1 not defined because of singularities)
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)    3.436e+06  2.642e+05  13.005  < 2e-16 ***
## bedrooms      -3.535e+04  4.197e+03  -8.424  < 2e-16 ***
## bathrooms      3.366e+04  6.912e+03   4.869 1.17e-06 ***
## sqft_living    1.897e+02  9.752e+00  19.457  < 2e-16 ***
## sqft_lot      -6.823e+00  9.325e-01  -7.317 3.14e-13 ***
## floors         4.327e+04  8.034e+03   5.386 7.69e-08 ***
## condition      2.458e+04  5.098e+03   4.822 1.48e-06 ***
## sqft_above     4.109e+00  9.633e+00   0.427     0.67
## sqft_basement         NA         NA      NA       NA
## yr_built      -1.725e+03  1.324e+02 -13.022  < 2e-16 ***
## yr_renovated   3.296e+00  3.315e+00   0.994     0.32
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 163900 on 3413 degrees of freedom
## Multiple R-squared:  0.4108, Adjusted R-squared:  0.4092
## F-statistic: 264.4 on 9 and 3413 DF,  p-value: < 2.2e-16
```

From the results above, we can fit a new model without the `sqft_basement` variable and to see what happens.

```r
# Fit new model without `sqft_basement`
lm2 <- lm(price ~ bedrooms + bathrooms + sqft_living + sqft_lot + floors + condition
                + sqft_above + yr_built + yr_renovated, data = train_no_outliers)

# View its summary
lm2_summary <- (summary(lm2))
lm2_summary
```

```
##
## Call:
## lm(formula = price ~ bedrooms + bathrooms + sqft_living + sqft_lot +
##     floors + condition + sqft_above + yr_built + yr_renovated,
##     data = train_no_outliers)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -886484 -102889   -5534   99122  755525
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   3.436e+06  2.642e+05  13.005  < 2e-16 ***
## bedrooms     -3.535e+04  4.197e+03  -8.424  < 2e-16 ***
## bathrooms     3.366e+04  6.912e+03   4.869 1.17e-06 ***
## sqft_living   1.897e+02  9.752e+00  19.457  < 2e-16 ***
## sqft_lot     -6.823e+00  9.325e-01  -7.317 3.14e-13 ***
## floors        4.327e+04  8.034e+03   5.386 7.69e-08 ***
## condition     2.458e+04  5.098e+03   4.822 1.48e-06 ***
## sqft_above    4.109e+00  9.633e+00   0.427     0.67
## yr_built     -1.725e+03  1.324e+02 -13.022  < 2e-16 ***
## yr_renovated  3.296e+00  3.315e+00   0.994     0.32
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 163900 on 3413 degrees of freedom
## Multiple R-squared:  0.4108, Adjusted R-squared:  0.4092
## F-statistic: 264.4 on 9 and 3413 DF,  p-value: < 2.2e-16
```

To determinate how good this model fits the data, we can observe the **Multiple R-squared** metric. It measures how strong is the linear relationship between the predictor variables and the response variable. Its value is 0.41078, it increases 3 times with respect of the value of the first model, it is almost medium, but most of the predictor variables are statistically significant except `sqft_above` and `yr_renovated`. Also, we can get the **R-squared** from the Multiple R-squared like this `(lm2_summary$r.squared)^2` equal to 0.1687402. It indicates that 16.87% of the variance in `price` can be explained by the predictors in the model.

Now, we predict the target value in the `test_no_outliers` dataset, update the `scores` table to store the **RMSE** of the `lm2` model and display the scores.

```r
# Predict the target value in the test_no_outliers set
pred3 <- predict(lm2, newdata = test_no_outliers)

# Update the scores table
scores <- bind_rows(scores,
                    tibble(Method = "RMSE lm2 no outliers",
                           RMSE = RMSE(test_no_outliers$price, pred3)))

# Display the scores table
scores %>%
  as_hux() %>%
  set_font_size(9) %>%
  set_tb_padding(2) %>%
  set_col_width(c(.7, .1)) %>%
  set_number_format(everywhere, 2, 7) %>%
  set_latex_float("h!") %>%
  theme_basic()
```

Method	RMSE
RMSE lm1 outliers	276244.5579709
RMSE rf1 outliers	293019.2976780
RMSE lm2 no outliers	154046.4453468

We compare the minimum, mean and maximum values from `test_no_outliers$price` (actual values) with the RMSE obtained in this table.

```r
# Compare the Min, Mean, Max from `test_no_outliers$price`
# with the Rmse obtained in this the table
test_no_outliers %>%
  pivot_longer(cols = c(1),
               names_to = 'Variable',
               values_to = 'value') %>%
  group_by(Variable) %>%
  summarise(Min = min(value),
            Mean = mean(value),
            Max = max(value)) %>%
  mutate(Rmse = RMSE(test_no_outliers$price, pred3)) %>%
  as_hux() %>%
  set_font_size(9) %>%
  set_tb_padding(2) %>%
  set_col_width(c(.1, .1, .1, .1, .1)) %>%
  set_number_format(everywhere, 2:5, 3) %>%
  set_latex_float("h!") %>%
  theme_basic()
```

Variable	Min	Mean	Max	Rmse
price	0.000	469724.229	1087500.000	154046.445

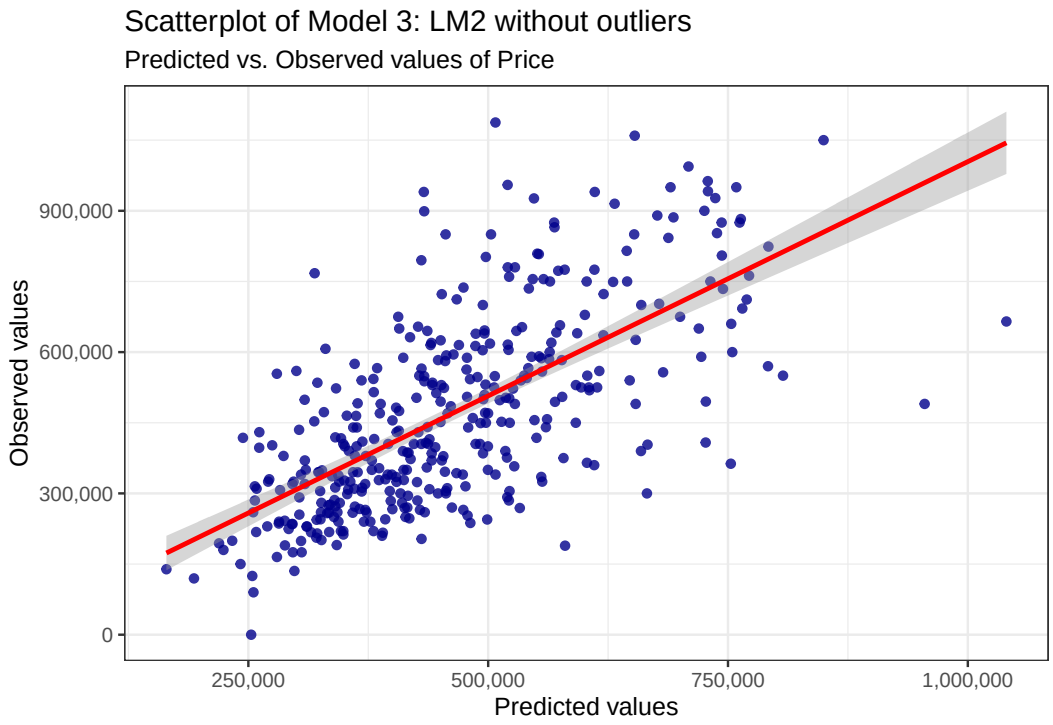
The value of RMSE indicates that our model is on average wrong by 154,046.445 units of `price`. We continue to look at the first six rows of the actual (observed) and the predicted values of `price`.

```
# look at the first six rows of the actual (observed)
# and the predicted values of `price`
data.frame(cbind(actuals=test_no_outliers$price, predicteds = pred3)) %>%
  head() %>%
  as_hux() %>%
  set_font_size(9) %>%
  set_tb_padding(2) %>%
  set_col_width(c(.1, .1)) %>%
  set_number_format(everywhere, 1:2, 2) %>%
  set_latex_float("h!") %>%
  theme_basic()
```

actuals	predicteds
407500.00	435090.70
206000.00	320944.71
449250.00	492108.39
329950.00	392907.53
272000.00	361075.78
513000.00	445892.85

We plot the observed (actual) versus predicted values of `price`.

```
test_no_outliers %>%
  ggplot(aes(pred3, price)) +
  geom_point(color = "darkblue", alpha = 0.8)+
  stat_smooth(aes(x = pred3,  y = price), method = "lm", color = "red")+
  ggtitle("Scatterplot of Model 3: LM2 without outliers",
          subtitle = "Predicted vs. Observed values of Price") +
  xlab("Predicted values") +
  ylab("Observed values") +
  scale_x_continuous(labels = comma) +
  scale_y_continuous(labels = comma) +
  theme_bw()
```



As we can see, the third model is much better and it does almost a good job in general terms than the second and first models, because we eliminate some noise created by the outliers that affected the first two models in their performance and when we used them to predict.

3.5 Model 4: Random Forest approach for Regression Model without Outliers

We fit a random forest regression model with the `randomForest()` function belonging to the package `randomForest` using the `train_no_outliers` dataset and display the content of the model.

```
# Fit the model
rf2 <- randomForest(price ~ ., data = train_no_outliers, ntree = 500,
                        importance = TRUE, type = "regression")

# Display the content of the model
rf2
```

```
##
## Call:
##  randomForest(formula = price ~ ., data = train_no_outliers, ntree = 500,      importance = TR
##                Type of random forest: regression
```

```
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##           Mean of squared residuals: 26448958819
##                     % Var explained: 41.79
```

We observe that **No. of variables tried at each split** is 3, that is correct, because `randomForest()` uses $p/3$, where p is the number of predictor variables, for regression and $\sqrt{p}$ for classification. In our case, we employ 10 predictors and we can calculate `10/3` is 3.3333333. We can change this number with the argument `mtry` if we want.

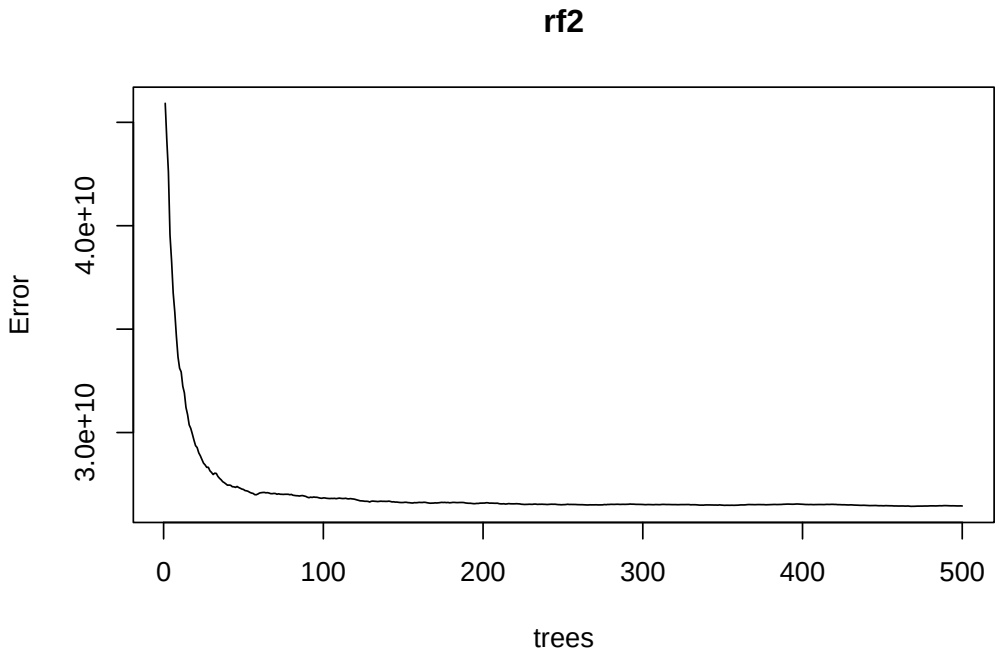
We predict the target value in the `test_no_outliers` dataset, update the `scores` table to store the **RMSE** of the `rf2` model and display the scores.

```
# Predict the target value in the test_no_outliers set
pred4 <- predict(rf2, newdata = test_no_outliers)

# Update the scores table
scores <- bind_rows(scores,
                    tibble(Method = "RMSE rf2 no outliers",
                           RMSE = RMSE(test_no_outliers$price, pred4)))

# Display the scores table
scores %>%
  as_hux() %>%
  set_font_size(9) %>%
  set_tb_padding(2) %>%
  set_col_width(c(.7, .1)) %>%
  set_number_format(everywhere, 2, 7) %>%
  set_latex_float("h!") %>%
  theme_basic()
```

Method	RMSE
RMSE lm1 outliers	276244.5579709
RMSE rf1 outliers	293019.2976780
RMSE lm2 no outliers	154046.4453468
RMSE rf2 no outliers	144981.5620889

We show a plot of the test MSE (Errors) by number of trees.

```
# Plot of the test MSE by number of trees
plot(rf2)
```



Show the variable importance plot.

```
# show the variable importance plot
varImpPlot(rf2)
```

From the plot we see that `sqft_living` and `yr_built` are the more important variables follow by `sqft_above`, `sqft_lot` and `bathrooms`.

We compare the minimum, mean and maximum values from `test_no_outliers$price` (actual values) with the RMSE obtained in this table.

```
# Compare the Min, Mean, Max from `test_no_outliers$price`
# with the Rmse obtained in this the table
test_no_outliers %>%
  pivot_longer(cols = c(1),
               names_to = 'Variable',
               values_to = 'value') %>%
  group_by(Variable) %>%
  summarise(Min = min(value),
            Mean = mean(value),
            Max = max(value)) %>%
  mutate(Rmse = RMSE(test_no_outliers$price, pred4)) %>%
  as_hux() %>%
  set_font_size(9) %>%
  set_tb_padding(2) %>%
  set_col_width(c(.1, .1, .1, .1, .1)) %>%
  set_number_format(everywhere, 2:5, 3) %>%
  set_latex_float("h!") %>%
  theme_basic()
```

Variable	Min	Mean	Max	Rmse
price	0.000	469724.229	1087500.000	144981.562

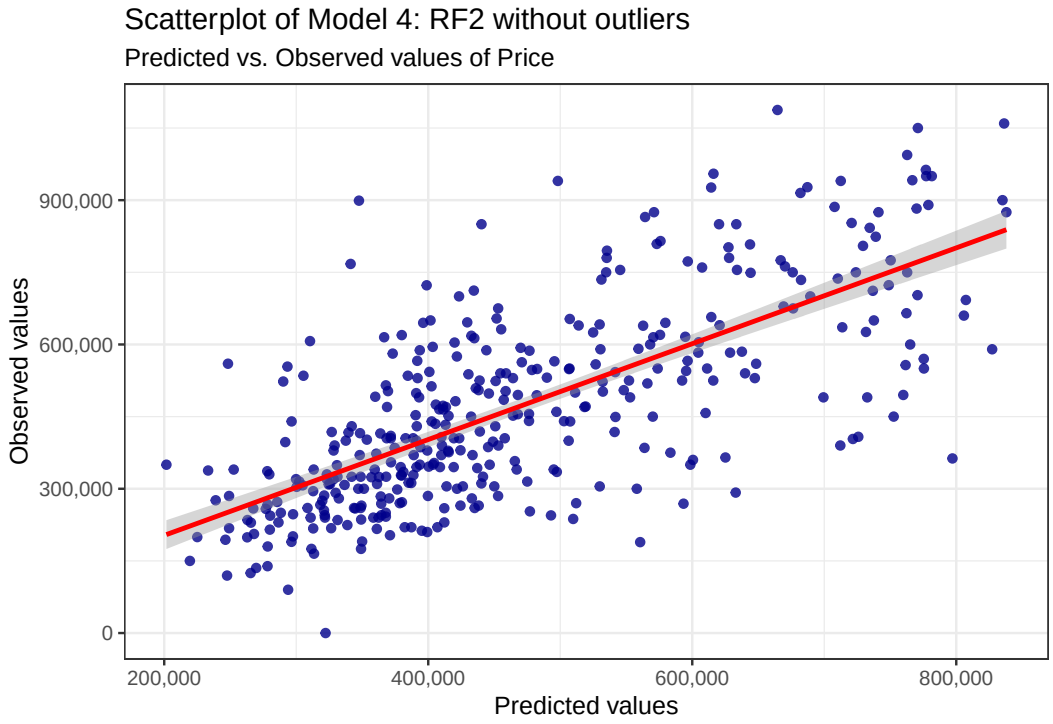
The value of RMSE indicates that our model is on average wrong by 144,981.562 units of `price`. We continue to look at the first six rows of the actual (observed) and the predicted values of `price`.

```
# look at the first six rows of the actual (observed)
# and the predicted values of `price`
data.frame(cbind(actuals=test_no_outliers$price, predicteds = pred4)) %>%
  head() %>%
  as_hux() %>%
  set_font_size(9) %>%
  set_tb_padding(2) %>%
  set_col_width(c(.1, .1)) %>%
  set_number_format(everywhere, 1:2, 2) %>%
  set_latex_float("h!") %>%
  theme_basic()
```

actuals	predicteds
407500.00	410025.37
206000.00	268020.00
449250.00	541850.24
329950.00	323226.43
272000.00	380080.39
513000.00	402490.49

We plot the observed (actual) versus predicted values of `price`.

```
test_no_outliers %>%
  ggplot(aes(pred4, price)) +
  geom_point(color = "darkblue", alpha = 0.8)+
  stat_smooth(aes(x = pred4,  y = price), method = "lm", color = "red")+
  ggtitle("Scatterplot of Model 4: RF2 without outliers",
          subtitle = "Predicted vs. Observed values of Price") +
  xlab("Predicted values") +
  ylab("Observed values") +
  scale_x_continuous(labels = comma) +
  scale_y_continuous(labels = comma) +
  theme_bw()
```



The fourth model is a little better than the third one, and much better than the second and first models. It looks better and it does almost a good job in general terms.

3.6 Making Predictions Using the Final Model

We determinate that our final model with the best RMSE was the **Model 4: Random Forest approach for Regression Model without Outliers** (`rf2`). Here are the final scores.

```
# Display the scores table
scores %>%
  as_hux() %>%
  set_font_size(9) %>%
  set_tb_padding(2) %>%
  set_col_width(c(.7, .1)) %>%
  set_number_format(everywhere, 2, 7) %>%
  set_latex_float("h!") %>%
  theme_basic()
```

Method	RMSE
RMSE lm1 outliers	276244.5579709
RMSE rf1 outliers	293019.2976780
RMSE lm2 no outliers	154046.4453468
RMSE rf2 no outliers	144981.5620889

We use the `rf2` model to make predictions on a new house (observation) with this code chunk.

```
# Create a new observation
new_house <- data.frame(bedrooms=3,
                        bathrooms=2.50,
                        sqft_living=3300,
                        sqft_lot=8100,
                        floors=2,
                        condition=4,
                        sqft_above=2500,
                        sqft_basement=300,
                        yr_built=2001,
                        yr_renovated=2003)

# Use the fitted model to predict the `price` value of new observation
print(paste0("The price of the house is: $ ", round(predict(rf2, new_house), 2)))
```

```
## [1] "The price of the house is: $ 657553.43"
```

Chapter 4

Conclusion

We began by preparing our project, downloading our dataset and loading it to `data`. Immediately, we performed the data exploration and visualization on the `data` dataset to find the relationship between the predict variable and the possible predictor variables for our model. First, we did an overall exploration in `data` to know its structure, detect missing values, and get a general information from the all variables. We continued to perform a data analysis in numerical columns to detect outliers, we made a copy from `data` to `data2`, we converted the integer columns to numeric, then, we made histograms to observe their distributions to detect possible anomalies. we placed emphasis on these variables: `price`, `sqft_lot`, `view`, `waterfront`, `condition`, `sqft_basement` and `yr_renovated`. Then, we analyzed our response variable `price` by creating bins in it to make some plots. With the realization of these processes, we detected that 6 variables had outliers: `price`, `sqft_basement`, `yr_renovated`, `sqft_lot`, `view` and `waterfront` and we analyzed how the predictor variables affect the `price` variable.

Next, we proceeded to prepare and clean the datasets, we created the `outliers` dataset with make a copy from `data2` by removing these columns `date`, `waterfront`, `view`, `street`, `city`, `statezip`, `country` and `pricebin`. Also, we created the `no_outliers` dataset from `data2` by removing these columns `date`, `waterfront`, `view`, `street`, `city`, `statezip`, `country` and `pricebin`. Created the `detect_outlier` and `remove_outlier` functions to remove outliers from multiple columns (`price`, `sqft_lot`, `sqft_basement` and `yr_renovated`) with the Interquartile range method suggets by (GeeksforGeeks, 2022).

We continued to preparing the train and test datasets, first we split the `outliers` dataset in two new datasets, we did that in the following way: Set the seed depending on the version of `R` to create the `train_outliers` and `test_outliers` datasets, where, `test_outliers` set was 10% and `train_outliers` set 90% of `outliers` dataset. We repeated the same process with the `no_outliers` dataset to create the `train_no_outliers` and `test_no_outliers` datasets. Then, we saved them in the file `cp_house_price.rda` in the `rdas` directory.

Afterwards, we defined our **loss function** (**RMSE**) and created our first model a linear regression model (`lm1`), in which we had to remove the `sqft_basement` variable. Next, we built the second model a random forest regression model (`rf1`), where this model was a little worse than the first one, because it got a RMSE just a little high than the first model. These first two models used the `train_outliers` and `test_outliers` datasets, these datasets contained outliers.

Then, we created the third model a linear regression model (`lm2`), in which we had to delete the `sqft_basement` variable. This model was much better and it did almost a good job in general terms than the second and first models, because we eliminated some noise created by the outliers. We built the last model a random forest regression model (`rf2`), where this model was a little better than the third one, and much better than the second and first models. These last two models used the `train_no_outliers` and `test_no_outliers` datasets, these datasets did not contain outliers.

Finally, we made a prediction using **the Final Model** (`rf2`), on a new house (observation), because it got a **RMSE** value of **144981.5620889**.

4.1 Limitations

Some limitations that we were able to find are:

- The computer equipment that does not have the necessary characteristics such as the processor and/or the ram memory to be able to run or execute some machine learning models, for example the models that we created, as well as, the internet connection we had to download the required data sets.
- We tried to use **the feature selection** process with the `varrank` package, which caused our code to hang. We also tried with the `relaimpo` package using the `calc.relimp` function, it throwing an error: `contrasts can be applied only to factors with 2 or more levels`, we tried to solve the error by converting the `character` type variables to `factor` and nothing continued to throw the same mistake. We tried with the `varImp` function from the `caret` package, it worked, but when creating the model using **the one hot encoding** technique with the `lm()` function, it sent an error in the chosen `factor` type variables (`city` and `statezip`): `contrasts can be applied only to factors with 2 or more levels`. In `R`, it is not necessary to create dummy variables to apply one hot encoding, it is only required that the predictor variables be of type `factor` and `R` together with the `lm()` function generate the dummy variables, and take the first level of the `factor` as baseline.

4.2 Future work

- We can use the *"Factor Variables with Many Levels"* technique proposed by the authors (Bruce et al., 2020, p. 167) to the `city` and `statezip` variables in the linear regression models, for example, we can group the postal codes (`statezip`) according to another variable, it can be `price` or better yet, group the postal codes (`statezip`) using the residuals of a model initial. Also, we can try to use **the target encoding** technique.
- We could test with other remove outliers techniques known, like Cook's distance, z-scores, percentiles, and so on.

- We can try to normalize the `price` variable and predictors variables to see what happens.
- We could try to tune the hyper parameters in the random forest models.
- We can apply other algorithms such as Boosting, XGB, SVM, neural networks, and so on. All of them focused on regression that were not implemented in this document.

References

Allwright, S. (2022, August 27). *How to interpret RMSE (simply explained)*. Stephen Allwright. Retrieved September 26, 2022, from https://stephenallwright.com/interpret-rmse/.

Bluman, A. G. (2011, January 14). *Elementary Statistics: A Step by Step Approach (8th ed.)*. McGraw-Hill.

Bruce, P., Bruce, A., & Gedeck, P. (2020, June 2). *Practical Statistics for Data Scientists: 50+ Essential Concepts Using R and Python (2nd ed.)*. O'Reilly Media.

DataCamp. (2022, March 4). *What is Kaggle?*. Datacamp. Retrieved September 26, 2022, from https://www.datacamp.com/blog/what-is-kaggle.

GeeksforGeeks. (2022, February 3). How to Remove Outliers from Multiple Columns in R DataFrame? Retrieved October 10, 2022, from https://www.geeksforgeeks.org/how-to-remove-outliers-from-multiple-columns-in-r-dataframe/

Gupta, S. (2022, May 16). *Data Science Process: A Beginner's Guide in Plain English*. Springboard Blog. Retrieved September 28, 2022, from https://www.springboard.com/blog/data-science/data-science-process/.

Irizarry, R. A. (2022, October 6). *Introduction to Data Science: Data Analysis and Prediction Algorithms with R*. Retrieved October 12, 2022, from https://rafalab.dfci.harvard.edu/dsbook/.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021, July 29). *An Introduction to Statistical Learning: with Applications in R (Springer Texts in Statistics) (2nd ed.)*. Springer.

Nantasenamat, C. (2022, January 14). *The Data Science Process - Towards Data Science*. Medium. Retrieved September 28, 2022, from https://towardsdatascience.com/the-data-science-process-a19eb7ebc41b.

Otto, S.A. (2019, Jan.,7). *How to normalize the RMSE [Blog post]*. Retrieved October 3, 2022, from https://www.marinedatascience.co/blog/2019/01/07/normalizing-the-rmse/.

Saini, A. (2022, August 26). *An Introduction to Random Forest Algorithm for beginners*. Analytics Vidhya. Retrieved October 15, 2022, from https://www.analyticsvidhya.com/blog/2021/10/an-introduction-to-random-forest-algorithm-for-beginners/.

Shree. (2018, August 26). *House price prediction*. Kaggle. Retrieved September 26, 2022, from https://www.kaggle.com/shree1992/housedata.

Singh, A. (2019, March 20). *Evaluation Metrics for Regression models- MAE Vs MSE Vs RMSE vs RMSLE*. Akhilendra.Com. Retrieved September 30, 2022, from https://akhilendra.

com/evaluation-metrics-regression-mae-mse-rmse-rmsle/.

Appendix

.1 Appendix A - Computer equipment with Windows 10 OS

The Operating Systems (OS) where this code was made is Windows 10 and its specs are:

```
System type: 64 bits
Edition:     Windows 10 Home
Version:     21H2
OS build:    19044.1889
Experience:  Windows Feature Experience Pack 120.2212.4180.0
```

Hardware specs:

```
Processor: Intel(R) Core(TM) i5-10500H CPU @ 2.50GHz
    - Cores: 6
    - Logical processors (Threads): 12
RAM Memory:  24 GB.
```

```
sessionInfo()
```

```
## R version 4.2.1 (2022-06-23 ucrt)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 19044)
##
## Matrix products: default
##
## Random number generation:
##  RNG:     Mersenne-Twister
##  Normal:  Inversion
##  Sample:  Rounding
##
## locale:
## [1] LC_COLLATE=Spanish_Latin America.utf8
```

```
## [2] LC_CTYPE=Spanish_Latin America.utf8
## [3] LC_MONETARY=Spanish_Latin America.utf8
## [4] LC_NUMERIC=C
## [5] LC_TIME=Spanish_Latin America.utf8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] randomForest_4.7-1.1 gridExtra_2.3        GGally_2.1.2
##  [4] huxtable_5.5.0       Hmisc_4.7-1          Formula_1.2-4
##  [7] survival_3.3-1       scales_1.2.1         ggthemes_4.2.4
## [10] data.table_1.14.2    caret_6.0-93         lattice_0.20-45
## [13] forcats_0.5.2        stringr_1.4.1        dplyr_1.0.9
## [16] purrr_0.3.4          readr_2.1.2          tidyr_1.2.0
## [19] tibble_3.1.8         ggplot2_3.3.6        tidyverse_1.3.2
## [22] this.path_0.8.0
##
## loaded via a namespace (and not attached):
##  [1] googledrive_2.0.0    colorspace_2.0-3     deldir_1.0-6
##  [4] ellipsis_0.3.2       class_7.3-20         htmlTable_2.4.1
##  [7] base64enc_0.1-3      fs_1.5.2             rstudioapi_0.14
## [10] farver_2.1.1         listenv_0.8.0        prodlim_2019.11.13
## [13] fansi_1.0.3          lubridate_1.8.0      xml2_1.3.3
## [16] codetools_0.2-18     splines_4.2.1        knitr_1.40
## [19] jsonlite_1.8.0       pROC_1.18.0          broom_1.0.1
## [22] cluster_2.1.3        dbplyr_2.2.1         png_0.1-7
## [25] compiler_4.2.1       httr_1.4.4           backports_1.4.1
## [28] assertthat_0.2.1     Matrix_1.4-1         fastmap_1.1.0
## [31] gargle_1.2.0         cli_3.3.0            htmltools_0.5.3
## [34] tools_4.2.1          gtable_0.3.0         glue_1.6.2
## [37] reshape2_1.4.4       Rcpp_1.0.9           cellranger_1.1.0
## [40] vctrs_0.4.1          nlme_3.1-157         iterators_1.0.14
## [43] timeDate_4021.104    gower_1.0.0          xfun_0.32
## [46] globals_0.16.1       rvest_1.0.3          lifecycle_1.0.1
## [49] googlesheets4_1.0.1  future_1.27.0        MASS_7.3-57
## [52] ipred_0.9-13         hms_1.1.2            parallel_4.2.1
## [55] RColorBrewer_1.1-3   yaml_2.3.5           rpart_4.1.16
## [58] reshape_0.8.9        latticeExtra_0.6-30  stringi_1.7.8
## [61] foreach_1.5.2        checkmate_2.1.0      hardhat_1.2.0
## [64] lava_1.6.10          commonmark_1.8.0     rlang_1.0.4
## [67] pkgconfig_2.0.3      evaluate_0.16        labeling_0.4.2
## [70] htmlwidgets_1.5.4    recipes_1.0.1        tidyselect_1.1.2
## [73] parallelly_1.32.1    plyr_1.8.7           magrittr_2.0.3
## [76] R6_2.5.1             generics_0.1.3       DBI_1.1.3
```

```
## [79] mgcv_1.8-40          pillar_1.8.1         haven_2.5.1
## [82] foreign_0.8-82       withr_2.5.0          nnet_7.3-17
## [85] future.apply_1.9.0   modelr_0.1.9         crayon_1.5.1
## [88] interp_1.1-3         utf8_1.2.2           tzdb_0.3.0
## [91] rmarkdown_2.16       jpeg_0.1-9           grid_4.2.1
## [94] readxl_1.4.1         ModelMetrics_1.2.2.2 reprex_2.0.2
## [97] digest_0.6.29        stats4_4.2.1         munsell_0.5.0
```

.2 Appendix B - Computer equipment with an Ubuntu Linux Distribution OS

The Operating Systems (OS) where this code was tested is Zorin OS 16.1 and its specs are:

```
System type:     64 bits
Edition:         Zorin OS Education
Description:     Zorin OS 16.1
Ubuntu_codename: focal
```

Hardware specs:

```
Processor: Intel(R) Core(TM) i5-2430M CPU @ 2.40GHz
    - Cores: 2
    - Logical processors (Threads): 4
RAM Memory:  8 GB.
```

```
sessionInfo()
```

```
R version 4.2.1 (2022-06-23)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Zorin OS 16.2

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/liblapack.so.3

locale:
 [1] LC_CTYPE=es_MX.UTF-8       LC_NUMERIC=C               LC_TIME=es_MX.UTF-8        LC_COLLATE=es_MX.UTF-8
 [5] LC_MONETARY=es_MX.UTF-8    LC_MESSAGES=es_MX.UTF-8    LC_PAPER=es_MX.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C             LC_MEASUREMENT=es_MX.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
 [1] randomForest_4.7-1.1 gridExtra_2.3        GGally_2.1.2         huxtable_5.5.0       Hmisc_4.7-0
 [6] Formula_1.2-4        survival_3.4-0       scales_1.2.0         ggthemes_4.2.4       data.table_1.14.2
[11] caret_6.0-93         lattice_0.20-45      forcats_0.5.1        stringr_1.4.0        dplyr_1.0.9
[16] purrr_0.3.4          readr_2.1.2          tidyr_1.2.0          tibble_3.1.8         ggplot2_3.3.6
[21] tidyverse_1.3.2      this.path_0.8.0
```

```
loaded via a namespace (and not attached):
 [1] googledrive_2.0.0    colorspace_2.0-3     deldir_1.0-6         ellipsis_0.3.2       class_7.3-20
 [6] htmlTable_2.4.1      base64enc_0.1-3      fs_1.5.2             rstudioapi_0.13      listenv_0.8.0
[11] prodlim_2019.11.13   fansi_1.0.3          lubridate_1.8.0      xml2_1.3.3           codetools_0.2-18
[16] splines_4.2.1        knitr_1.39           jsonlite_1.8.0       pROC_1.18.0          broom_1.0.0
[21] cluster_2.1.4        dbplyr_2.2.1         png_0.1-7            compiler_4.2.1       httr_1.4.3
[26] backports_1.4.1      assertthat_0.2.1     Matrix_1.5-1         fastmap_1.1.0        gargle_1.2.0
[31] cli_3.3.0            htmltools_0.5.3      tools_4.2.1          gtable_0.3.0         glue_1.6.2
[36] reshape2_1.4.4       Rcpp_1.0.9           cellranger_1.1.0     vctrs_0.4.1          nlme_3.1-160
[41] iterators_1.0.14     timeDate_4021.104    xfun_0.32            gower_1.0.0          globals_0.16.0
[46] rvest_1.0.2          lifecycle_1.0.1      googlesheets4_1.0.0  future_1.27.0        MASS_7.3-58.1
[51] ipred_0.9-13         hms_1.1.1            parallel_4.2.1       RColorBrewer_1.1-3   yaml_2.3.5
[56] rpart_4.1.16         reshape_0.8.9        latticeExtra_0.6-30  stringi_1.7.8        foreach_1.5.2
[61] checkmate_2.1.0      hardhat_1.2.0        lava_1.6.10          rlang_1.0.4          pkgconfig_2.0.3
[66] evaluate_0.16        recipes_1.0.1        htmlwidgets_1.5.4    tidyselect_1.1.2     parallelly_1.32.1
[71] plyr_1.8.7           magrittr_2.0.3       R6_2.5.1             generics_0.1.3       DBI_1.1.3
[76] pillar_1.8.0         haven_2.5.0          foreign_0.8-82       withr_2.5.0          nnet_7.3-18
[81] future.apply_1.9.0   modelr_0.1.8         crayon_1.5.1         interp_1.1-3         utf8_1.2.2
[86] tzdb_0.3.0           rmarkdown_2.14       jpeg_0.1-9           grid_4.2.1           readxl_1.4.0
[91] ModelMetrics_1.2.2.2 reprex_2.0.1         digest_0.6.29        stats4_4.2.1         munsell_0.5.0
```