Melbourne_Property Pricing

Shane Pinto

2023-06-01

Table of Contents

Introduction and Overview

Australian real estate is one of the hottest markets worldwide with Melbourne in particular consistently ranked as the most livable city in the world. The aim of this project is to get a better insight into what determines the price of the property by creating a data model to predict the price of a property. This model would provide potential buyers a yardstick to measure their shortlisted property against what's available.

The understand the efficacy of our models, we will use the "Root Mean Square Error" or RMSE for short to test the findings of our systems. The RMSE is a measure of the differences between the results obtained VS the actual results. Lower the RMSE value, the more successful the recommended system is.

Data Preparation

Load the required libraries and download and read the data-file we will be using for our project.

```
# Install and Load the required libraries.
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(dplyr)) install.packages("dplyr", repos = "http://cran.us.r-
project.org")
if(!require(stringr)) install.packages("stringr", repos = "http://cran.us.r-
project.org")
if(!require(gridExtra)) install.packages("gridExtra", repos =
"http://cran.us.r-project.org")
if(!require(rpart)) install.packages("rpart",repos = "http://cran.us.r-
project.org")
if(!require(scales)) install.packages("scales",repos = "http://cran.us.r-
project.org")
if(!require(BBmisc)) install.packages("BBmisc",repos = "http://cran.us.r-
project.org")
if(!require(randomForest)) install.packages("randomForest",repos =
"http://cran.us.r-project.org")

library(tidyverse)
library(caret)
library(dplyr)
library(stringr)
library(gridExtra)
library(rpart)
library(scales)
library(BBmisc)
library(randomForest)
```

```
# Downloand and save the file in your "working directory"
"https://www.kaggle.com/datasets/anthonypino/melbourne-housing-
market?select=MELBOURNE_HOUSE_PRICES_LESS.csv"

## [1] "https://www.kaggle.com/datasets/anthonypino/melbourne-housing-
market?select=MELBOURNE_HOUSE_PRICES_LESS.csv"

# Read the file and set digits to 4
file <- "C:\\Users\\shane\\Documents\\R\\MELBOURNE_HOUSE_PRICES_LESS.csv"
property <- read.csv(file)
digits = 4
```

Dataset Inspection

Reviewing the data-set revels that there are 63000 rows of data with 13 variables namely"Suburb", "Address","Rooms" etc. The classes of each variable are in the correct format so no further modifications are required. We would however like to elaborate on the data in the "Type" and "Method" columns to make them easier for the users to understand. Also, there is approx. 15000 rows of missing data, this needs to be investigated further.

```
# Inspect the data-set
head(property)

##         Suburb          Address Rooms Type   Price Method  SellerG
Date
## 1   Abbotsford    49 Lithgow St     3    h 1490000      S   Jellis
1/4/2017
## 2   Abbotsford    59A Turner St     3    h 1220000      S Marshall
1/4/2017
## 3   Abbotsford    119B Yarra St     3    h 1420000      S   Nelson
1/4/2017
## 4   Aberfeldie       68 Vida St     3    h 1515000      S    Barry
1/4/2017
## 5 Airport West 92 Clydesdale Rd     2    h  670000      S   Nelson
1/4/2017
## 6 Airport West     4/32 Earl St     2    t  530000      S   Jellis
1/4/2017
##   Postcode            Regionname Propertycount Distance
## 1     3067 Northern Metropolitan          4019      3.0
## 2     3067 Northern Metropolitan          4019      3.0
## 3     3067 Northern Metropolitan          4019      3.0
## 4     3040  Western Metropolitan          1543      7.5
## 5     3042  Western Metropolitan          3464     10.4
## 6     3042  Western Metropolitan          3464     10.4
##                  CouncilArea
## 1         Yarra City Council
## 2         Yarra City Council
## 3         Yarra City Council
## 4 Moonee Valley City Council
```

```
## 5 Moonee Valley City Council
## 6 Moonee Valley City Council
```

```
str(property)
```

```
## 'data.frame':    63023 obs. of  13 variables:
##  $ Suburb       : chr  "Abbotsford" "Abbotsford" "Abbotsford" "Aberfeldie"
...
##  $ Address      : chr  "49 Lithgow St" "59A Turner St" "119B Yarra St" "68
Vida St" ...
##  $ Rooms        : int  3 3 3 3 2 2 2 3 6 3 ...
##  $ Type         : chr  "h" "h" "h" "h" ...
##  $ Price        : int  1490000 1220000 1420000 1515000 670000 530000
540000 715000 NA 1925000 ...
##  $ Method       : chr  "S" "S" "S" "S" ...
##  $ SellerG      : chr  "Jellis" "Marshall" "Nelson" "Barry" ...
##  $ Date         : chr  "1/4/2017" "1/4/2017" "1/4/2017" "1/4/2017" ...
##  $ Postcode     : int  3067 3067 3067 3040 3042 3042 3042 3042 3021 3206
...
##  $ Regionname   : chr  "Northern Metropolitan" "Northern Metropolitan"
"Northern Metropolitan" "Western Metropolitan" ...
##  $ Propertycount: int  4019 4019 4019 1543 3464 3464 3464 3464 1899 3280
...
##  $ Distance     : num  3 3 3 7.5 10.4 10.4 10.4 10.4 14 3 ...
##  $ CouncilArea  : chr  "Yarra City Council" "Yarra City Council" "Yarra
City Council" "Moonee Valley City Council" ...
```

```
colnames(property)
```

```
##  [1] "Suburb"        "Address"       "Rooms"         "Type"
##  [5] "Price"         "Method"        "SellerG"       "Date"
##  [9] "Postcode"      "Regionname"    "Propertycount" "Distance"
## [13] "CouncilArea"
```

```
# Check for missing data.
sum(is.na(property))
```

```
## [1] 14590
```

Dataset Modification

The data reveals that all missing data(NA) is from the “Price” column. While some methods of sale like passed in(PI), sold not disclosed(SN) and sold prior not disclosed(PN) will definitely not have the sales price disclosed but we also notice sold(S) and sold prior(SP) have quite a few missing values. We can put these missing prices to just incomplete data or data missed while creating the data-set and remove these entries.

```
# Investigating missing data
property %>% summarise_all(~ sum(is.na(.)))
```

```
##   Suburb Address Rooms Type Price Method SellerG Date Postcode Regionname
## 1      0       0     0    0 14590      0       0    0        0          0
##   Propertycount Distance CouncilArea
## 1             0        0           0
```

```
property %>% filter (is.na(Price)) %>%
        group_by(Method) %>%
        summarise(Count = n()) %>%
        arrange(desc(Count))
```

```
## # A tibble: 9 × 2
##   Method Count
##   <chr>  <int>
## 1 PI      3850
## 2 S       3439
## 3 SN      2674
## 4 SP      2436
## 5 VB       932
## 6 PN       651
## 7 W        484
## 8 SS        73
## 9 SA        51
```

```
#Remove rows with no prices
property <- property %>% filter(!(is.na(Price)))
nrow(property)
```

```
## [1] 48433
```

The “Type” column contains only 3 values, “h”,“t” and “u”. We will update these to read as “House”, “Townhouse” and “Unit” for better understanding.

```
# Inspect the "Type" variable
property %>% group_by(Type) %>%
        summarise(Count = n())
```

```
## # A tibble: 3 × 2
##   Type  Count
##   <chr> <int>
## 1 h     34161
## 2 t      4980
## 3 u      9292
```

```
# Updating the "Type" variable.
property$Type <- str_replace(property$Type,"^h", "House")
property$Type <- str_replace(property$Type,"^u", "Unit")
property$Type <- str_replace(property$Type,"^t", "Townhouse")
```

The “Method” column contains 5 values which we will elaborate for better clarity.

```
# Inspect the "Method" variable
property %>%
        group_by(Method) %>%
        summarise(Count = n())
```

```
## # A tibble: 5 × 2
##   Method Count
##   <chr>  <int>
## 1 PI      5940
## 2 S      30624
## 3 SA       365
## 4 SP      6480
## 5 VB      5024
```

```
# Updating the "Method" variable.
property$Method <- str_replace(property$Method,"^S$", "Sold")
property$Method <- str_replace(property$Method,"^PI$", "Passed In")
property$Method <- str_replace(property$Method,"^SA$", "Sold After")
property$Method <- str_replace(property$Method,"^SP$", "Sold Prior")
property$Method <- str_replace(property$Method,"^VB$", "Vendor Bid")
```

We also want to trim the output of the “CouncilArea” data for easier visualization later.

```
# Trim the "CouncilArea" data
property <- property %>% mutate(CityCouncil = word(CouncilArea, 1,-3)) %>%
select(-CouncilArea)
```

Finally, we remove the columns containing the actual address as we won’t require this for the modelling and the postcodes as this contains the same data already in the suburbs data.

```
# Remove the data columns we don't require
property <- property %>% select(-Address,-Postcode)
```

Data Analysis

We investigate the make up of the variables of this data-set to gain an understanding on what could help us in shaping our models.

The red dotted line in graphical representations when visible represents the average value of the variable being discussed.

Pricing

The price varies greatly from a minimum of 85,000 dollars to a maximum of 12 million dollars. This could be due to the number of bedrooms, distance from the city center or just desirable post codes to live in. The histogram of all prices displays this variability with the majority of the prices around 100,000 dollars to about 2.5 million dollars with a average price of property being 1 million dollars.

```
# Summary of the Price variable
summary(property$Price)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
##    85000   620000   830000   997898  1220000 11200000
```

```
# Top and Bottom 5 properties by price
property %>% arrange(desc(Price)) %>% head(5)
```

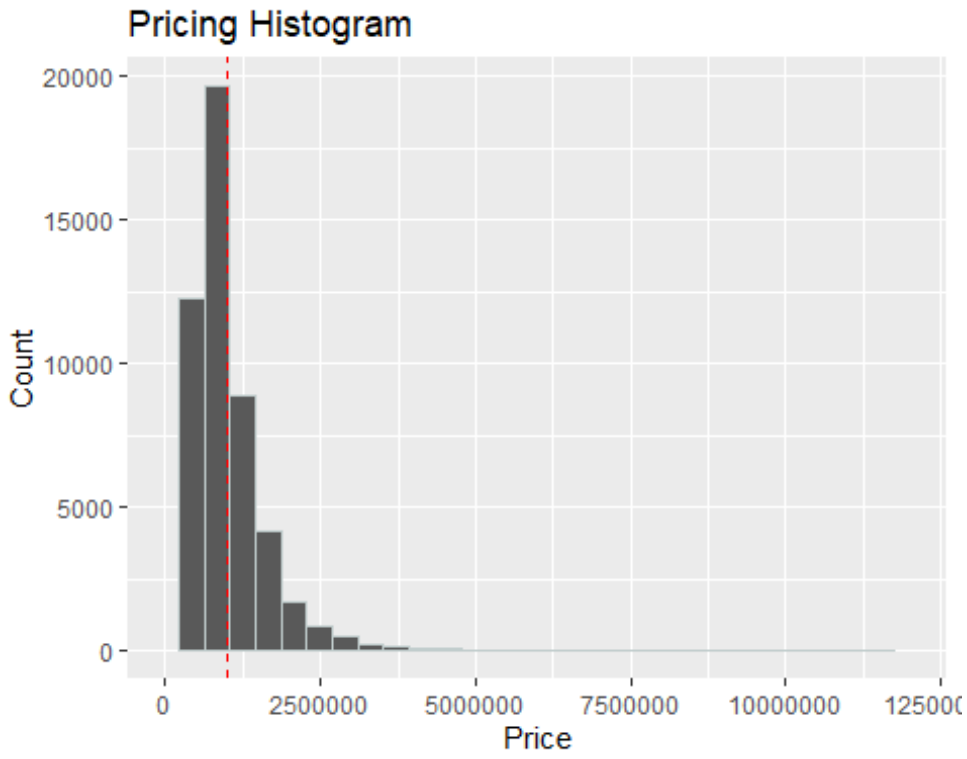
```
##       Suburb Rooms  Type    Price     Method       SellerG       Date
## 1   Brighton     4 House 11200000 Vendor Bid hockingstuart 28/10/2017
## 2   Mulgrave     3 House  9000000  Passed In          Hall  29/7/2017
## 3 Canterbury     5 House  8000000 Vendor Bid     Sotheby's  13/5/2017
## 4   Hawthorn     4 House  7650000       Sold  Abercromby's  17/6/2017
## 5   Armadale     4 House  7000000  Passed In           Kay 25/11/2017
##                    Regionname Propertycount Distance CityCouncil
## 1      Southern Metropolitan         10579     10.5     Bayside
## 2 South-Eastern Metropolitan          7113     18.8      Monash
## 3      Southern Metropolitan          3265      8.4  Boroondara
## 4      Southern Metropolitan         11308      5.3  Boroondara
## 5      Southern Metropolitan          4836      6.3 Stonnington
```

```
property %>% arrange(desc(Price)) %>% tail(5)
```

```
##             Suburb Rooms Type  Price    Method SellerG       Date
## 48429 Malvern East     1 Unit 112000      Sold     C21   8/1/2018
## 48430 Malvern East     1 Unit 112000      Sold     C21 16/12/2017
## 48431 Malvern East     1 Unit 112000      Sold     C21 23/12/2017
## 48432 Malvern East     1 Unit 112000      Sold     C21 30/12/2017
## 48433    Footscray     1 Unit  85000 Passed In Burnham   3/9/2016
##                  Regionname Propertycount Distance CityCouncil
## 48429 Southern Metropolitan          8801      8.4 Stonnington
## 48430 Southern Metropolitan          8801      8.4 Stonnington
## 48431 Southern Metropolitan          8801      8.4 Stonnington
```

```
## 48432 Southern Metropolitan          8801      8.4 Stonnington
## 48433  Western Metropolitan          7570      5.1 Maribyrnong
```

```
# Histogram of the prices
property %>% ggplot(aes(Price)) +
        geom_histogram(color = "azure3") +
        scale_x_continuous(limits = c(0, 12000000)) +
        geom_vline(xintercept = mean(property$Price)
                   , lty = 2, color = "red") +
        xlab("Price") +
        ylab("Count") +
        ggtitle("Pricing Histogram")
```

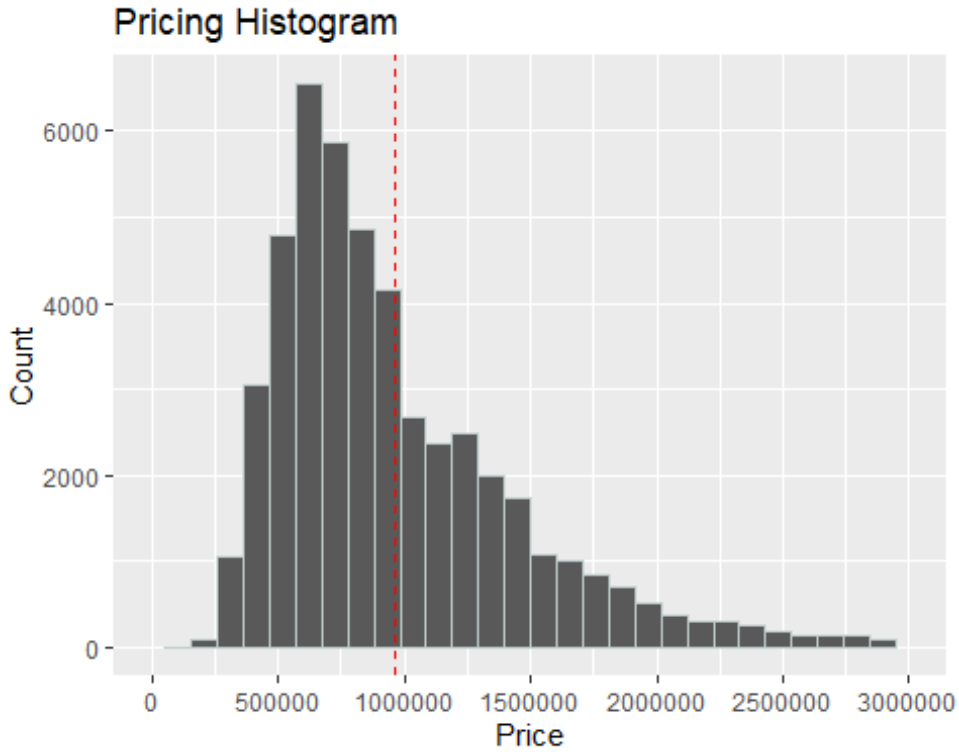


If we remove the outliers beyond 3 million dollars, the histogram gives us a better understanding of the pricing spread with average cost of property again around 1 million dollars. Removing these outliers still preserves 98% of our data.

```
# Removing outlier values greater than 3 million dollar
tibble(All = nrow(property), Upto_3M = nrow(property %>% filter( Price <=
3000000)))
```

```
## # A tibble: 1 × 2
##     All Upto_3M
##   <int>   <int>
## 1 48433   47825
```

```r
property <- property %>% filter( Price <= 3000000)

# Histogram of prices under 3 million
property %>% ggplot(aes(Price)) +
    geom_histogram(color = "azure3") +
    scale_x_continuous(limits = c(0, 3000000),
                       breaks = seq(0, 3000000, 500000)) +
    geom_vline(xintercept = mean(property$Price),
               lty = 2, color = "red") +
    xlab("Price") +
    ylab("Count") +
    ggtitle("Pricing Histogram")
```



The "Suburb" feature

We have 370 unique suburbs and on an average 130 properties were sold per suburb. Inspecting the 25 top and bottom selling suburbs we see quite a variation with the top end preforming well over the average sales price while the bottom 50 falling quite short. Comparing the top and bottom selling suburbs VS the top and bottom suburbs with the highest average selling prices reveals that there must be other factors that contribute to the overall sales prices of these properties.

```r
# The different suburbs,regions and council areas
property %>% summarise(uniq_Suburbs = n_distinct(Suburb),
                       uniq_Regions = n_distinct(Regionname),
                       uniq_Councils = n_distinct(CityCouncil))
```

```
##   uniq_Suburbs uniq_Regions uniq_Councils
## 1          370            8            34
```

```
# Average number of properties sold in a suburb
avg_properties_suburb <- property %>%
                         group_by(Suburb) %>%
                         summarise(Count = n()) %>%
                         summarise(Avg = mean(Count)) %>%
                         pull(.)
avg_properties_suburb
```

```
## [1] 129.2568
```

```
# Top and Bottom 5 suburbs by properties sold
property %>% group_by(Suburb) %>%
                summarise(Count = n()) %>%
                arrange(desc(Count)) %>%
                head(5)
```

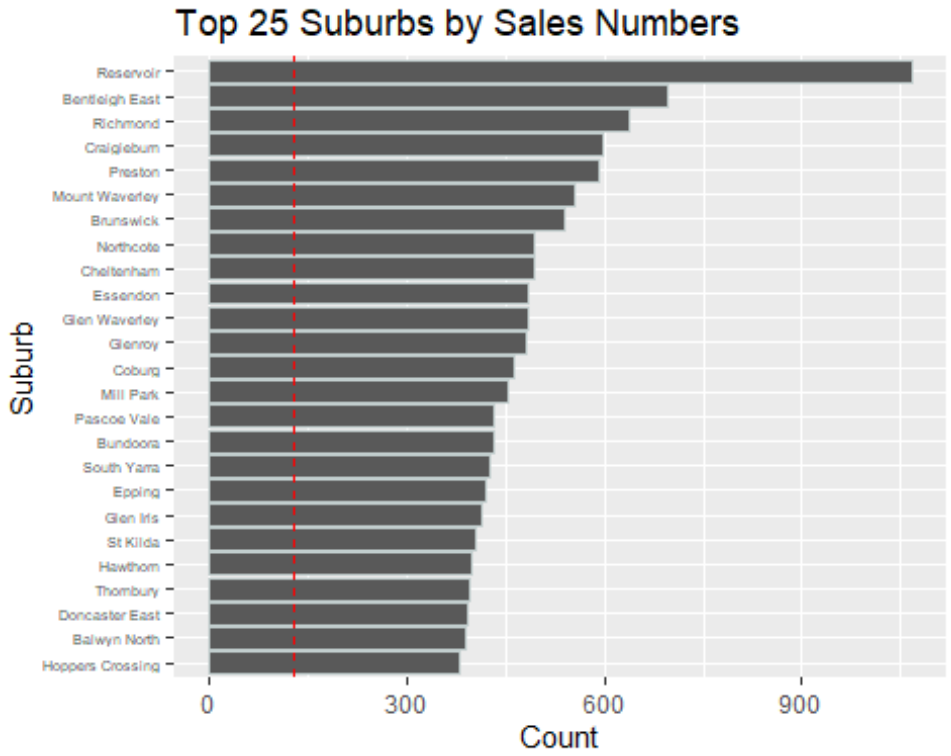
```
## # A tibble: 5 × 2
##   Suburb         Count
##   <chr>          <int>
## 1 Reservoir       1067
## 2 Bentleigh East   696
## 3 Richmond         637
## 4 Craigieburn      598
## 5 Preston          593
```

```
property %>% group_by(Suburb) %>%
                summarise(Count = n()) %>%
                arrange(desc(Count)) %>%
                tail(5)
```

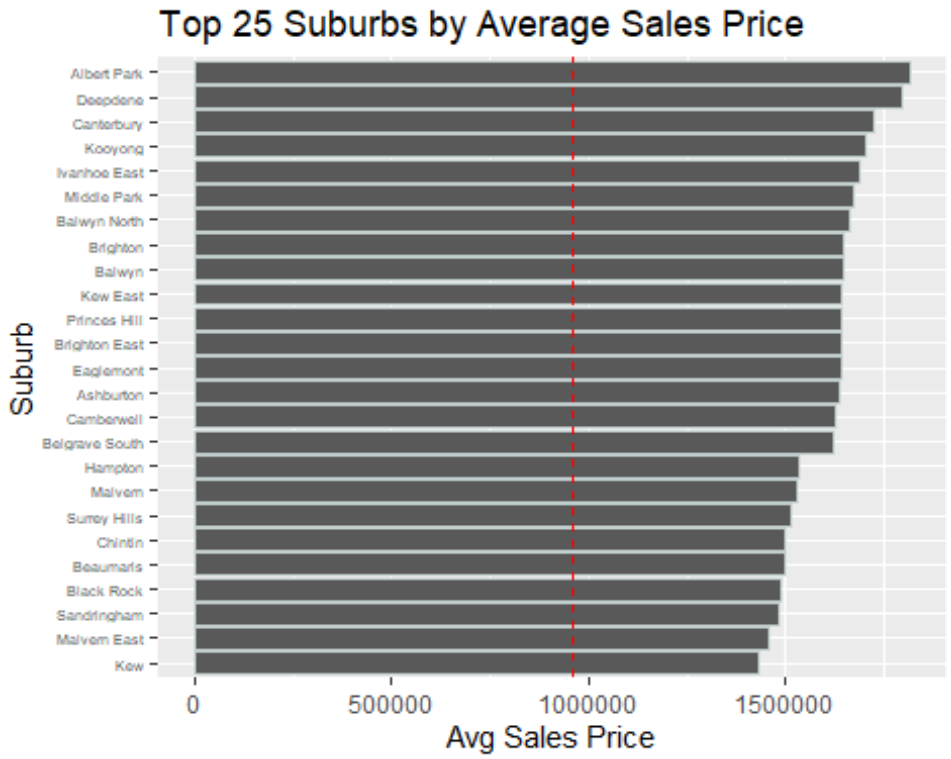
```
## # A tibble: 5 × 2
##   Suburb         Count
##   <chr>          <int>
## 1 Woori Yallock      1
## 2 Yan Yean           1
## 3 Yarra Junction     1
## 4 croydon            1
## 5 viewbank           1
```

```
# Top 25 suburbs by sales numbers
property %>%
        group_by(Suburb) %>%
        summarise(Count = n()) %>%
        arrange(desc(Count)) %>%
        head(25) %>%
        ggplot(aes(Count, reorder(Suburb,Count))) +
        geom_col(color = "azure3") +
        geom_vline(xintercept = avg_properties_suburb,lty = 2, color = "red")
```

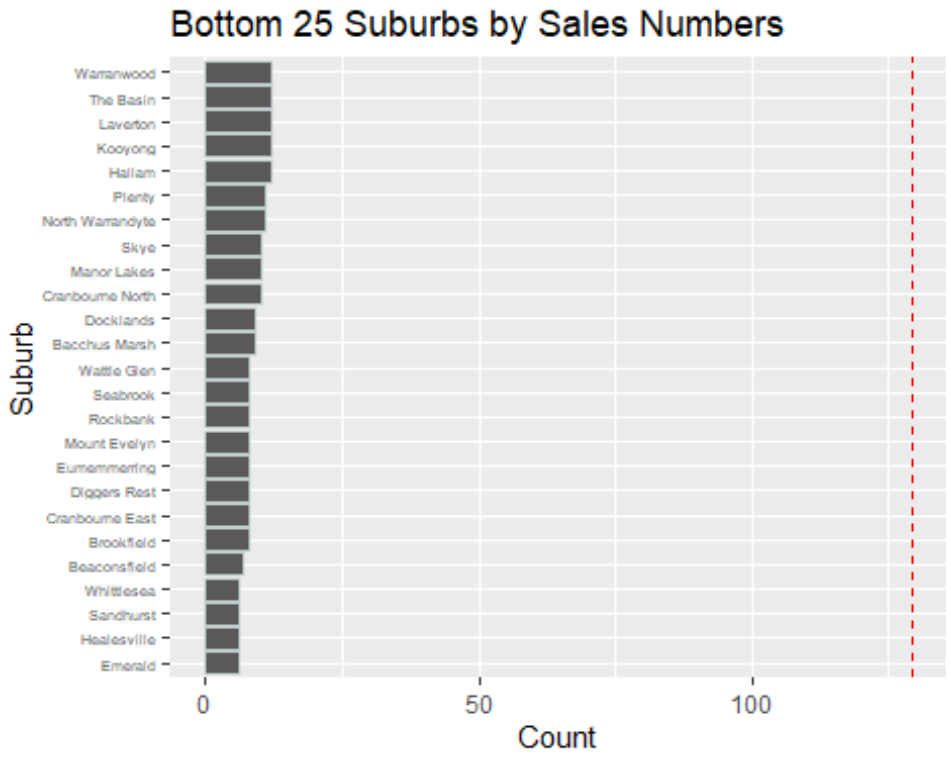
```
+
        theme(axis.text.y = element_text(size = 5)) +
        xlab("Count") +
        ylab("Suburb") +
        ggtitle("Top 25 Suburbs by Sales Numbers")
```



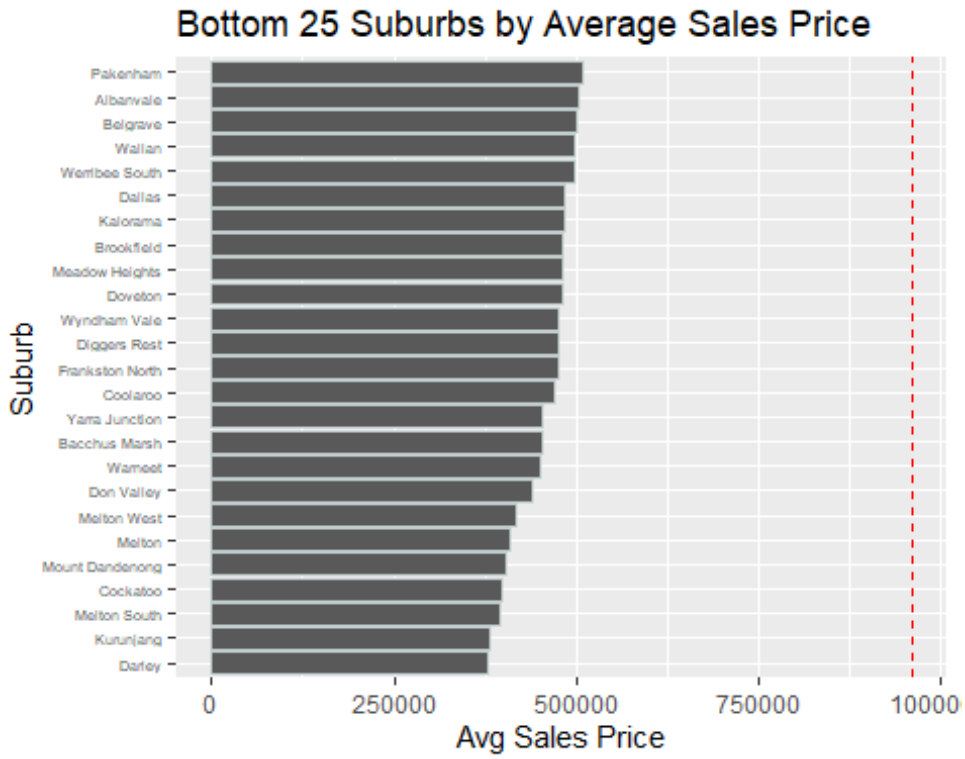

```
# Top 25 suburbs by average sales price
property %>%
        group_by(Suburb) %>%
        summarise(Count = n(), Avg_Price = mean(Price)) %>%
        arrange(desc(Avg_Price)) %>%
        head(25) %>%
        ggplot(aes(x = Avg_Price,y = reorder(Suburb,Avg_Price))) +
        geom_col(color = "azure3") +
        geom_vline(xintercept = mean(property$Price), lty = 2, color = "red")
+
        theme(axis.text.y = element_text(size = 5)) +
        xlab("Avg Sales Price") +
        ylab("Suburb") +
        ggtitle("Top 25 Suburbs by Average Sales Price")
```




```
# Bottom 25 suburbs by sales numbers
property %>%
        group_by(Suburb) %>%
        summarise(Count = n()) %>%
        filter(Count > 5) %>%
        arrange(desc(Count)) %>%
        tail(25) %>%
        ggplot(aes(Count, reorder(Suburb,Count))) +
        geom_col(color = "azure3") +
        geom_vline(xintercept = avg_properties_suburb,lty = 2, color = "red")
+
        theme(axis.text.y = element_text(size = 5)) +
        xlab("Count") +
        ylab("Suburb") +
        ggtitle("Bottom 25 Suburbs by Sales Numbers")
```




```r
# Bottom 25 suburbs by average sales price
property %>%
        group_by(Suburb) %>%
        summarise(Count = n(), Avg_Price = mean(Price)) %>%
        arrange(desc(Avg_Price)) %>%
        tail(25) %>%
        ggplot(aes(x = Avg_Price,y = reorder(Suburb,Avg_Price))) +
        geom_col(color = "azure3") +
        geom_vline(xintercept = mean(property$Price), lty = 2, color = "red")
+
        theme(axis.text.y = element_text(size = 5)) +
        xlab("Avg Sales Price") +
        ylab("Suburb") +
        ggtitle("Bottom 25 Suburbs by Average Sales Price")
```



The "Rooms" feature

This feature reveals that 3 bedrooms properties are the most popular followed 4 and then 2 bedrooms. On the other scale much larger properties in excess of 6-7 bedrooms are very few. This result meets the normal expectation as the average person/family would purchase a 2-4 bedroom property.

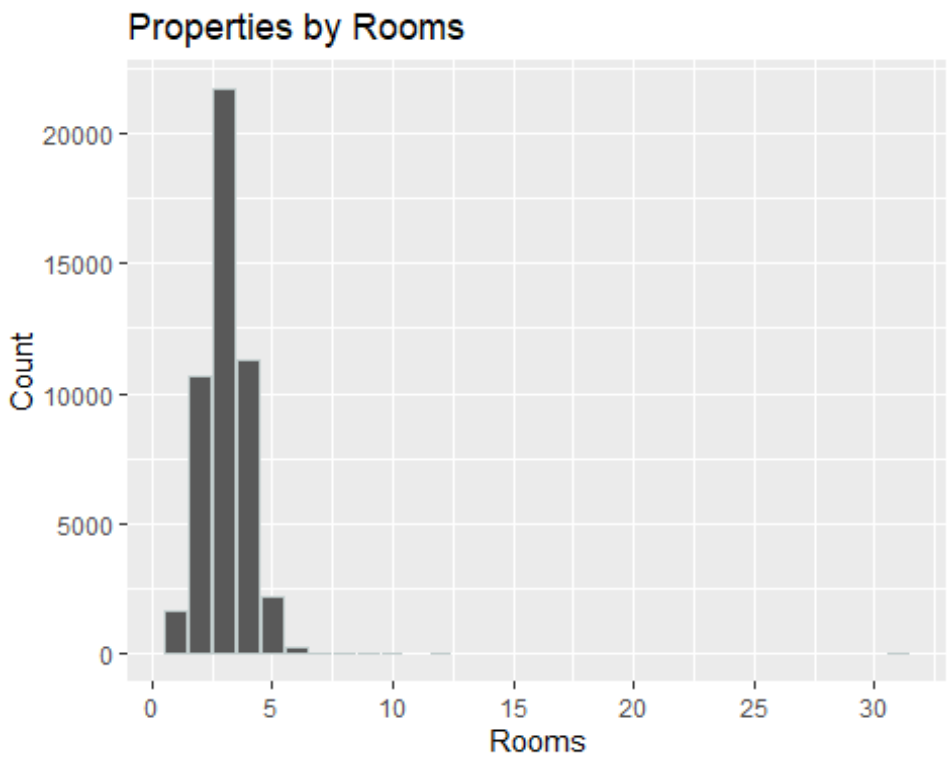
We can remove all listings in excess of 5 bedrooms and still retain 99% of our data.

```
# Summarise the number of rooms by sold properties
property %>% group_by(Rooms) %>%
        summarise(Count = n()) %>%
        arrange(desc(Count))
```

```
## # A tibble: 12 × 2
##    Rooms Count
##    <int> <int>
##  1     3 21720
##  2     4 11295
##  3     2 10668
##  4     5  2151
##  5     1  1669
##  6     6   264
##  7     7    32
##  8     8    17
##  9    10     5
## 10     9     2
```

```
## 11    12     1
## 12    31     1
```

```
# Count of Properties by the Number of Rooms
property %>% group_by(Rooms) %>%
        summarise(Count = n(), Avg_Price_Room = mean(Price)) %>%
        ggplot(aes(Rooms,Count)) +
        geom_col(color = "azure3") +
        scale_x_continuous(breaks = seq(0,32,5)) +
        ggtitle("Properties by Rooms")
```

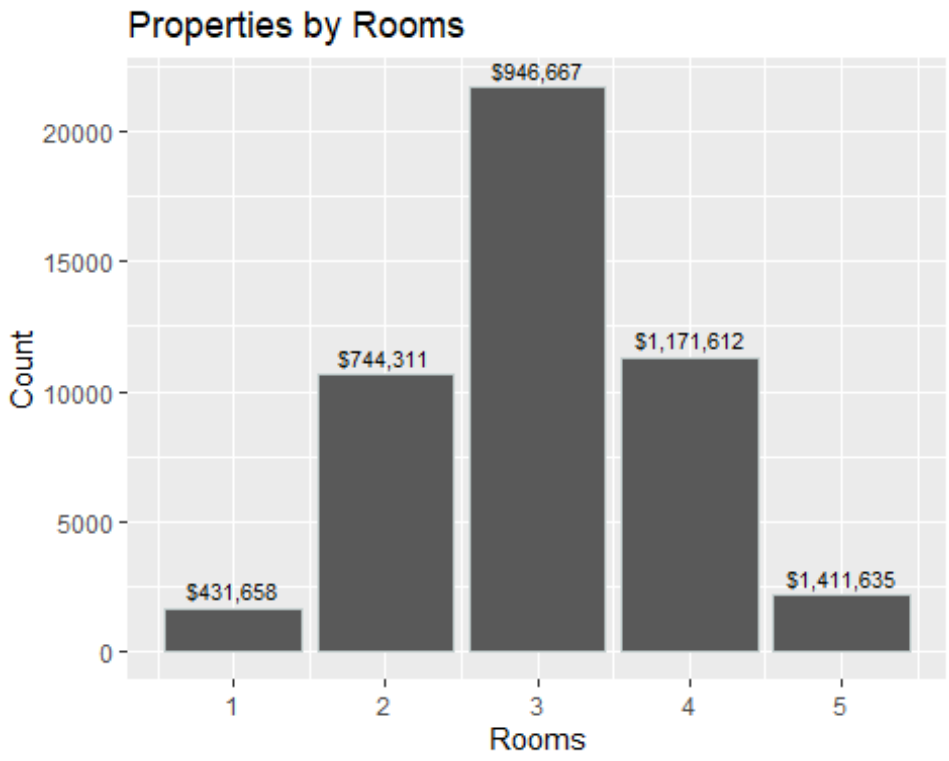



```
# Keep properties with 5 or less bedrooms.
tibble(All_Rooms = nrow(property), Upto_5rooms = nrow(property %>%
filter(Rooms <= 5)))
```

```
## # A tibble: 1 × 2
##   All_Rooms Upto_5rooms
##       <int>       <int>
## 1     47825       47503
```

```
property <- property %>% filter(Rooms <= 5)

# Property count with 5 or less bedrooms along with average price per type
property %>% group_by(Rooms) %>%
        summarise(Count = n(), Avg_Price_Room = mean(Price)) %>%
        ggplot(aes(Rooms,Count, label = dollar(round(Avg_Price_Room)))) +
        geom_col(color = "azure3") +
```

```
        geom_text(position = position_dodge(width = .9),
                  vjust = -0.5, size = 3) +
        scale_x_continuous(breaks = seq(0,5,1)) +
        ggtitle("Properties by Rooms")
```



Property "Type" feature

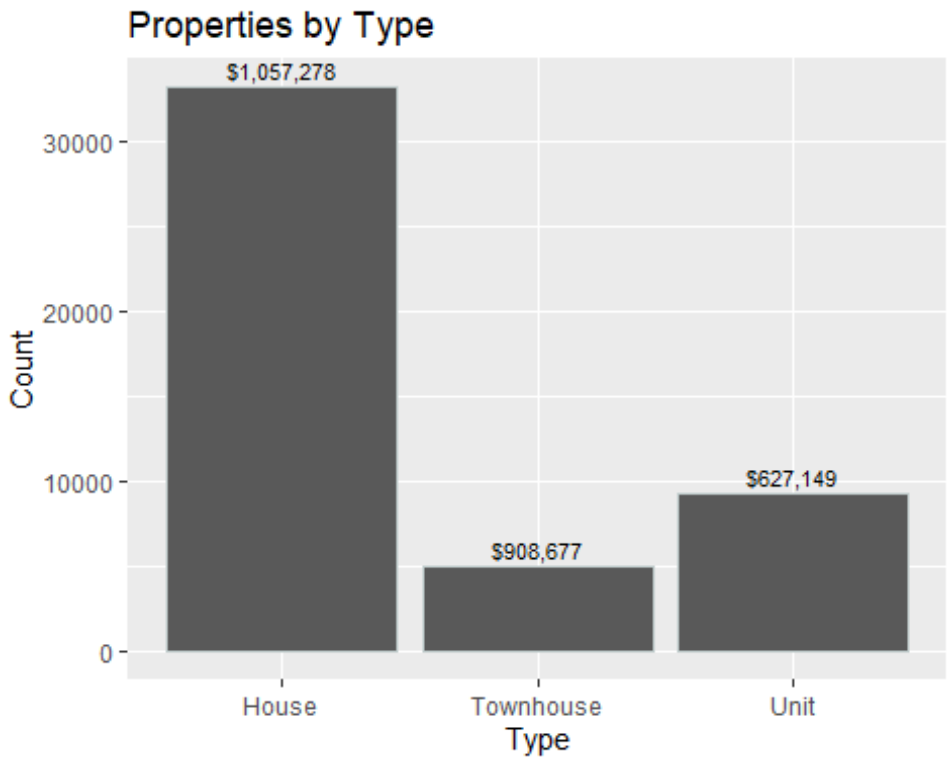
The "Type" data reveals that standalone houses are the most popular choice followed by units and townhouses. Standalone house is the expected top choice given most people's requirements but surprisingly unit sales out preform townhouses which is unexpected, maybe pricing has some effect on this.

If we investigate the average prices for each property type we notice that units are the cheapest, this may explain why units have sold so strongly and is the second highest type by sales numbers.

```
# Count and Average Price by each property type
property %>% group_by(Type) %>% summarise(Count = n(),Avg_Price =
mean(Price)) %>% arrange(desc(Count))
```

```
## # A tibble: 3 × 3
##   Type      Count Avg_Price
##   <chr>     <int>     <dbl>
## 1 House     33246  1057278.
## 2 Unit       9282   627149.
## 3 Townhouse  4975   908677.
```

```
# Histogram of counts and average prices by property "Types"
property %>% group_by(Type) %>%
      summarise(Count = n(), Avg_Price_Type = mean(Price)) %>%
      ggplot(aes(Type,Count,label = dollar(round(Avg_Price_Type)))) +
      geom_col(color = "azure3") +
      geom_text(position = position_dodge(width = .9), vjust = -0.5, size =
3) +
      ggtitle("Properties by Type")
```



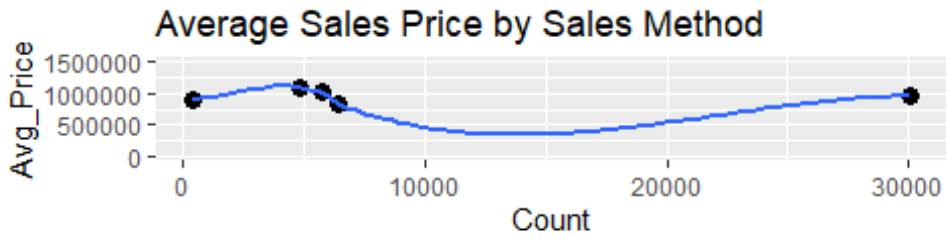
The sales "Method" feature

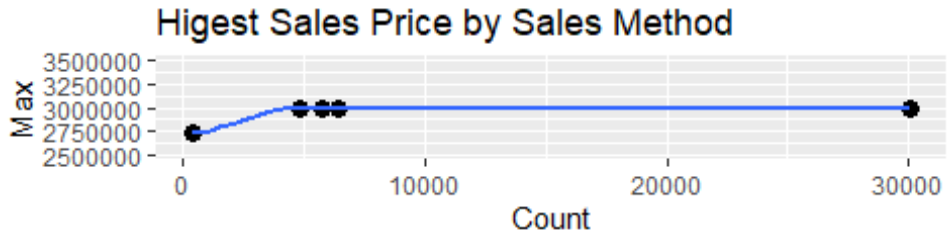
The average sales price and the top sales price by Sales Method are fairly similar to each other but if we inspect the lowest achieved price by sales method we see quite a variation with properties "Passed-in" achieving the lowest sales price by type. There could be various reasons for this but it makes sense to include this variable in our models to account for this unpredictability.

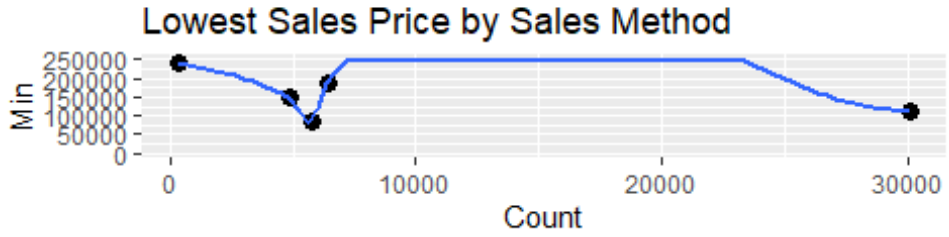
```
# Statistics of Sales using different sales methods
property %>% group_by(Method) %>% summarise(Count = n(), Avg_Price =
mean(Price),Min = min(Price), Max = max(Price))

## # A tibble: 5 × 5
##   Method     Count Avg_Price    Min     Max
##   <chr>      <int>     <dbl>  <int>   <int>
## 1 Passed In   5769  1012446.  85000 3000000
## 2 Sold       30112   952865. 112000 3000000
## 3 Sold After   361   900925. 240000 2725000
```

```
## 4 Sold Prior  6411   823951. 190000 3000000
## 5 Vendor Bid  4850  1103314. 148000 3000000
```

```r
# Histogram of counts vs average, higest and lowest prices by sales "Methods"
method_avg_price <- property %>%
                        group_by(Method) %>%
                        summarise(Count = n(), Avg_Price = mean(Price)) %>%
                        ggplot(aes(Count,Avg_Price)) +
                        geom_point(size = 3) +
                        ylim(0, 1500000) +
                        geom_smooth() +
                        ggtitle("Average Sales Price by Sales Method")

method_max_price <- property %>%
                        group_by(Method) %>%
                        summarise(Count = n(), Max = max(Price)) %>%
                        ggplot(aes(Count,Max)) +
                        geom_point(size = 3) +
                        ylim(2500000, 3500000) +
                        geom_smooth() +
                        ggtitle("Higest Sales Price by Sales Method")

method_min_price <- property %>%
                        group_by(Method) %>%
                        summarise(Count = n(), Min = min(Price)) %>%
                        ggplot(aes(Count,Min)) +
                        geom_point(size = 3) +
                        ylim(0,250000) +
                        geom_smooth() +
                        ggtitle("Lowest Sales Price by Sales Method")

grid.arrange(method_avg_price,method_max_price,method_min_price,nrow = 3)
```







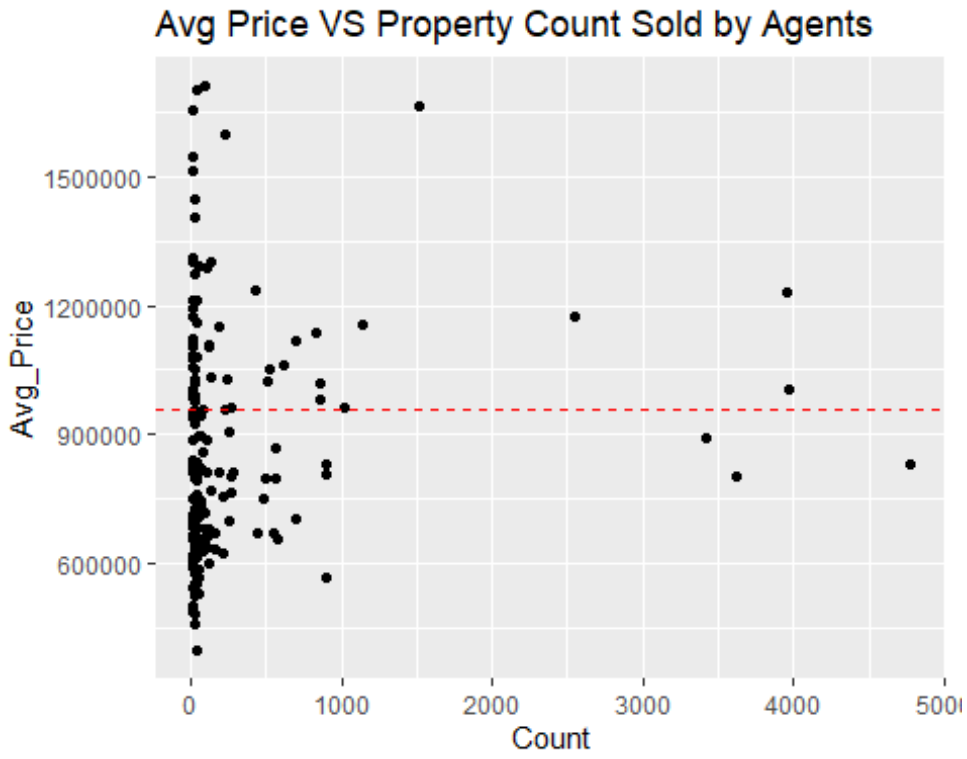
The selling agency feature, "SellerG"

We investigate if a particular agency has an impact on the sales price of the property irrespective of the property type or suburb location.

We have a total of 419 different agents that sold the 47500 properties in this data-set. The histogram of the average sales price of a property against the total properties sold by an agent revels that even if we have a high performing agency it doesn't necessarily translate to a higher comparative selling price.

```
# View number of unique properties & agents along with the average sales
price
property %>% summarise(Properties_Sold = nrow(property),Number_of_Agents =
n_distinct(SellerG),Avg_Sales_Price = round(mean(property$Price), digits =
0))

##   Properties_Sold Number_of_Agents Avg_Sales_Price
## 1           47503              419          957669

# Histogram of Average sales prices by number of properties sold
property %>% group_by(SellerG) %>%
        summarise(Count = n(), Avg_Price = mean(Price)) %>%
        filter(Count >=10) %>%
        ggplot(aes(Count, Avg_Price)) +
        geom_point() +
        geom_hline(yintercept = mean(property$Price),
```

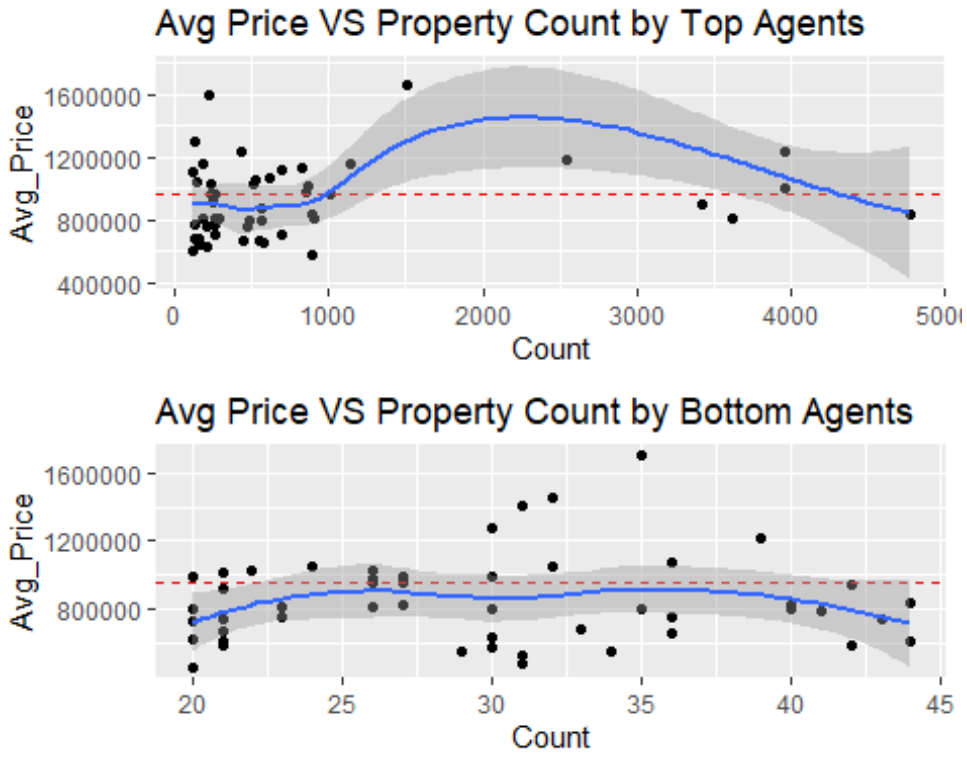
```
                  lty = 2, color = "red") +
        ggtitle("Avg Price VS Property Count Sold by Agents")
```



We can study the top and bottom 50 agents by total number of properties sold a bit deeper to see if we are correct to infer the agency alone doesn't affect the sales price. These scaled down histograms confirm that the total number of properties sold is independent of the sales prices and there we can ignore the "SellerG" column as well for our analysis.

```
# Histogram of Average Sales by Top and Bottom 50 Agents by number of
properties sold
top_agents <- property %>%
                group_by(SellerG) %>%
                summarise(Count = n(),Avg_Price = mean(Price)) %>%
                arrange(desc(Count)) %>%
                head(50) %>%
                ggplot(aes(Count, Avg_Price)) +
                geom_point() +
                geom_hline(yintercept = mean(property$Price),
                           lty = 2, color = "red") +
                scale_y_continuous(breaks = seq(400000,1800000,400000)) +
                geom_smooth() +
                ggtitle("Avg Price VS Property Count by Top Agents")

# Filtering only those agents who sold 20 or more properties to give us a
better insight
bottom_agents <- property %>%
                group_by(SellerG) %>%
```

```
                summarise(Count = n(), Avg_Price = mean(Price)) %>%
                filter(Count >= 20) %>%
                arrange(Count) %>%
                head(50) %>%
                ggplot(aes(Count, Avg_Price)) +
                geom_point() +
                geom_hline(yintercept = mean(property$Price),
                           lty = 2, color = "red") +
                scale_y_continuous(breaks = seq(400000,1800000,400000)) +
                geom_smooth() +
                ggtitle("Avg Price VS Property Count by Bottom Agents")

grid.arrange(top_agents,bottom_agents)
```




```
# Remove the selling agents information.
property <- property %>% select(-SellerG)
```

The Region Name and Distance variable

Studying the RegionName data along with the relative distances to the center of Melbourne shows a general trend of prices decreasing the further out they are with some exceptions like “Northern Metropolitan” and “Western Metropolitan”.

```
# Properties sold per Region and relative distace to the city centre
property %>% group_by(Regionname) %>%
                summarise(Count = n(), Avg_Price = mean(Price),
```

```
                  Proximity = min(Distance)) %>%
                arrange(desc(Avg_Price))
```

```
## # A tibble: 8 × 4
##   Regionname                 Count Avg_Price Proximity
##   <chr>                      <int>     <dbl>     <dbl>
## 1 Southern Metropolitan      11954  1256378.       0.7
## 2 Eastern Metropolitan        7477  1059156.       7.8
## 3 South-Eastern Metropolitan  3970   836475.      15.5
## 4 Northern Metropolitan      13509   813509.       0
## 5 Western Metropolitan        9595   795251.       4.3
## 6 Eastern Victoria             370   687684.      25.2
## 7 Northern Victoria            448   635831.      20.1
## 8 Western Victoria             180   410271.      29.8
```

Inspecting the contents of Northern Metropolitan revels 8 different suburbs within this Region with relative distances from 0KM to 29.8KM. This data once broken down is actually the same as what we have in the CityCouncil and Suburbs data so we can ignore this variable as it just duplicates the data.

```
# Inspecting the contents of the Northern Metropolitan region.
property %>% filter(Regionname == "Northern Metropolitan") %>%
                group_by(CityCouncil) %>%
                summarise(Proximity = min(Distance)) %>%
                arrange(desc(Proximity))
```

```
## # A tibble: 7 × 2
##   CityCouncil Proximity
##   <chr>           <dbl>
## 1 Whittlesea       15.3
## 2 Hume             13.1
## 3 Banyule          12.1
## 4 Darebin           5.3
## 5 Moreland          3.6
## 6 Yarra             2
## 7 Melbourne         0
```

```
# Removing the RegionName column from the data-set
property <- property %>% select(-Regionname)
```

The City Council and Distance variable

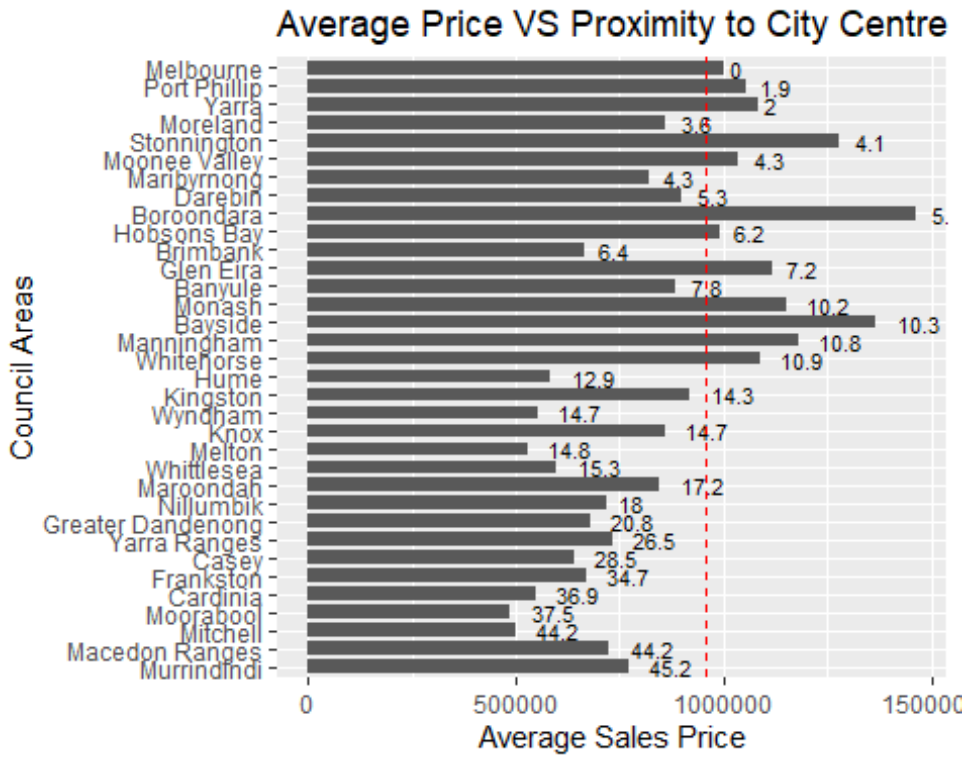
The "CityCouncil" data revels that the most popular suburbs by the number of properties sold is neither the cheapest nor the most expensive and is close to the average of 1 million dollars. The relative distance of these council areas to the city center is is also not a clear indicator of the preference.

We would have to put these preferences to uncontrollable factors like individual basis, family and friends living in the area or even quality of schools in the area.

```r
# Viewing the average, minimum and maximum selling prices by Council Area
property %>% group_by(CityCouncil) %>%
                summarise(Count = n(), Avg_Price = mean(Price),
                  Proximity = min(Distance),
                  Min = min(Price), Max = max(Price)) %>%
                arrange(desc(Count)) %>%
                head(10)
```

```
## # A tibble: 10 × 6
##    CityCouncil   Count Avg_Price Proximity    Min     Max
##    <chr>         <int>     <dbl>     <dbl>  <int>   <int>
##  1 Darebin        3454   899470.       5.3 145000 2990000
##  2 Boroondara     3164  1462585.       5.3 160000 3000000
##  3 Banyule        2880   885907.       7.8 290000 2990000
##  4 Brimbank       2683   664485.       6.4 145000 1750000
##  5 Moreland       2512   859627.       3.6 170000 2875000
##  6 Monash         2396  1150707.      10.2 305000 2950000
##  7 Bayside        2385  1365685.      10.3 132500 3000000
##  8 Hume           2350   583983.      12.9 221000 2450000
##  9 Glen Eira      2338  1116861.       7.2 131000 3000000
## 10 Moonee Valley  2147  1036668.       4.3 193000 2905000
```

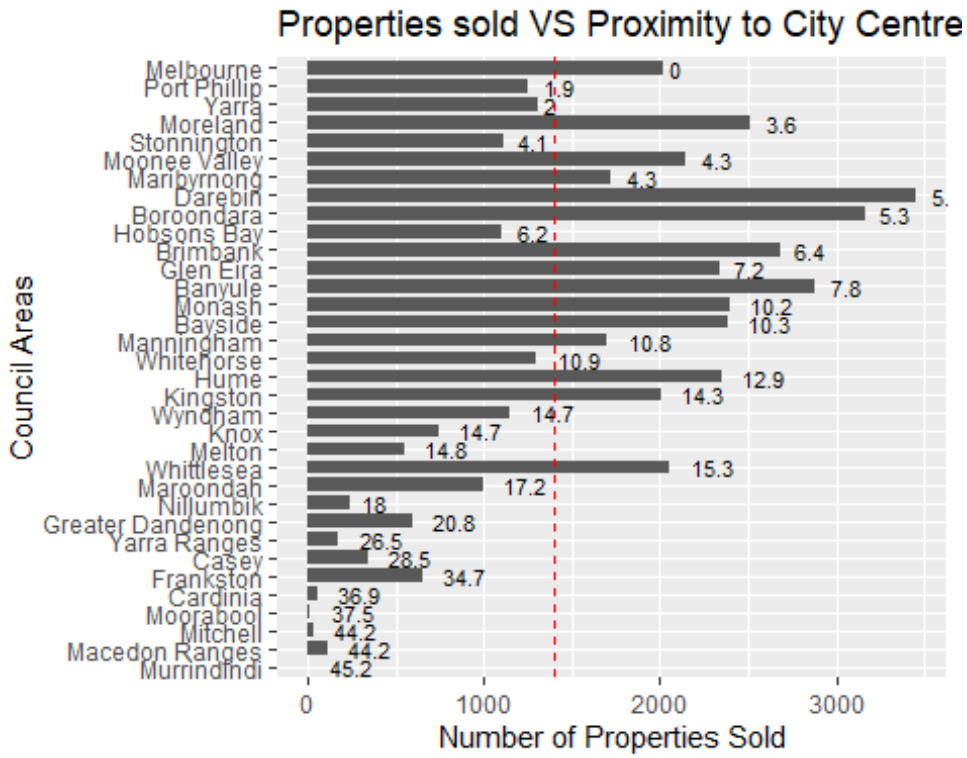
```r
# Average Prices by Council Area's vs Relative distance to the city center
property %>%
        group_by(CityCouncil) %>%
        summarise(Count = n(), Avg_Price = mean(Price),
                  Proximity = min(Distance)) %>%
        ggplot(aes(Avg_Price, reorder(CityCouncil,-Proximity),
                  label = Proximity)) +
        geom_col(width = 0.75) +
        geom_text(position = position_dodge(width = .9),
                  size = 3,hjust = -0.5) +
        geom_vline(xintercept = mean(property$Price),
                  lty = 2, color= "red") +
        xlab("Average Sales Price") +
        ylab("Council Areas") +
        ggtitle("Average Price VS Proximity to City Centre")
```




```
# Calculate the average properties sold per council area
avg_properties_council <- property %>%
                          group_by(CityCouncil) %>%
                          summarise(Count = n()) %>%
                          summarise(Avg = mean(Count)) %>%
                          pull(.)
avg_properties_council
```

```
## [1] 1397.147
```

```
# Number of properties sold by council area vs Proximity to the city center
property %>%
        group_by(CityCouncil) %>%
        summarise(Count = n(), Avg_Price = mean(Price),
                Proximity = min(Distance)) %>%
        ggplot(aes(Count, reorder(CityCouncil,-Proximity),
                label = Proximity)) +
        geom_col(width = 0.75) +
        geom_text(position = position_dodge(width = .9),
                  size = 3,hjust = -0.5) +
        geom_vline(xintercept = avg_properties_council,
                   lty = 2, color= "red") +
        xlab("Number of Properties Sold") +
        ylab("Council Areas") +
        ggtitle("Properties sold VS Proximity to City Centre")
```


Properties sold VS Proximity to City Centre
Council Areas
Melbourne
Port Phillip
Yarra
Moreland
Stonnington
Moonee Valley
Maribyrnong
Darebin
Boroondara
Hobsons Bay
Brimbank
Glen Eira
Banyule
Monash
Bayside
Manningham
Whitehorse
Hume
Kingston
Wyndham
Knox
Melton
Whittlesea
Maroondah
Nillumbik
Greater Dandenong
Yarra Ranges
Casey
Frankston
Cardinia
Mooraboo
Mitchell
Macedon Ranges
Murrindindi
0
1.9
2
3.6
4.1
4.3
4.3
5.
5.3
6.2
6.4
7.2
7.8
10.2
10.3
10.8
10.9
12.9
14.3
14.7
14.7
14.8
15.3
17.2
18
20.8
26.5
28.5
34.7
36.9
37.5
44.2
44.2
45.2
0
1000
2000
3000
Number of Properties Sold

Model Building

Now the data is ready for modelling, the last step is to normalize the data. Different numerical data columns may have vastly different ranges, making a direct comparison inconclusive. For example, the price columns has data ranging in the millions while the Room's column has data from 1 to 5. Normalization is a technique for bringing all data into a comparable range so that comparisons are more relevant.

After that, we create a training, testing and a smaller training subset to tune our models. We will use these data sets to build and analyse our various models and achieve an optimum model we can recommend.

```
# Normalize the data in the data-set
property <- normalize(property,method = "range", range = c(0,1))

# Creating the Test, Training and smaller Training subset
set.seed(10, sample.kind="Rounding")
test_index <- createDataPartition(y = property$Price, times = 1, p = 0.1,
list = FALSE)
train <- property[-test_index,]
test <- property[test_index,]
train_subset <- train %>% sample_n(10000)
```

We now use these data sets to start building different models and validate them for efficacy.

Average model

This model takes the average price and predicts it for every new listing. This is our base naive model which simply takes the average of the sales price in the test set.

```
# Prediction based on Mean Prices
mu <- mean(train$Price)
mu

## [1] 0.299427

# Calculating RMSE using the mean
mu_rmse <- RMSE(test$Price, mu)
mu_rmse

## [1] 0.1648784

# Initializing a RMSE table to save the results
rmse_results <- tibble(Method = "Average Price Model", RMSE = mu_rmse)
rmse_results %>% knitr::kable()
```

Method	RMSE
Average Price Model	0.1648784

This is will be our baseline and we will try to use the insights we learned from our data analysis to build models with lower RMSE values. We understand from our findings that buyers have a basis towards the physical location of the property(Suburb,Council Area) and the type of property (House,Townhouse,Unit). We will incorporate these bias into our models along with various other regression models to try and obtain the best RMSE value.

Suburb's Model

We see that some suburbs sales prices are stronger than the average by a noticeable margin. With all factors being equal we put this down to the buyers personal bias for selecting this suburb. We incorporate this bias into our calculations by deducting the mean prices by suburbs from the overall mean prices of the data-set.

```
# Prediction based on the Suburbs
suburb_avgs <- train %>%
         group_by(Suburb) %>%
         summarise(sub = mean(Price - mu))

predicted_pricing <- mu + test %>%
        left_join(suburb_avgs, by='Suburb') %>%
        pull(sub)

# Calculating RMSE using the suburbs bias
rmse_model_suburb <- RMSE(predicted_pricing, test$Price)
rmse_model_suburb

## [1] 0.1263415

# Adding RMSE results to the Table
rmse_results <- bind_rows(rmse_results,
                          tibble(Method="Suburbs Model",
                          RMSE = rmse_model_suburb))
rmse_results %>% knitr::kable()
```

Method	RMSE
Average Price Model	0.1648784
Suburbs Model	0.1263415

We see a noticeable improvement in the RMSE obtained by factoring in the buyers preference for a particular suburb over another. We now look at accounting for the property type bias.

Property Type Model

From the data exploration we know that buyers have a bias towards the type of property they want to purchase independent of other factors. We incorporate this property type bias into our Suburb model by deducting the mean prices by property types from the Suburb model we created.

```
# Prediction based on the Suburbs and Property Types
types_avgs <- train %>%
        left_join(suburb_avgs, by="Suburb") %>%
        group_by(Type) %>%
        summarise(type = mean(Price - mu - sub))

predicted_ratings <- test %>%
        left_join(suburb_avgs, by="Suburb") %>%
        left_join(types_avgs, by="Type") %>%
        mutate(pred = mu + sub + type) %>%
        pull(pred)

# Calculating RMSE using the suburbs and property types effects
rmse_model_types <- RMSE(predicted_ratings, test$Price)
rmse_model_types
```

```
## [1] 0.1062553
```

```
# Adding RMSE results to the Table
rmse_results <- bind_rows(rmse_results,
                          tibble(Method="Suburbs and Property Types Model",
                                 RMSE = rmse_model_types))
rmse_results %>% knitr::kable()
```

Method	RMSE
Average Price Model	0.1648784
Suburbs Model	0.1263415
Suburbs and Property Types Model	0.1062553

Including the Property Types bias into our existing model has improved the RMSE value fairly yet again.

Regularisation

Regularization is used to control the variability of the effects size which can influence the prediction and skew our results. For example, a property can archive an overly high sales price in a particular suburb hence skewing the average sales price for that suburb which will affect all other similar types of properties in that suburb. This unusually high sales price could be due to additional features in the property like a swimming pool, cinema room or even a recent renovation.

We uses a tuning parameter, λ, to minimize the RMSE to account for these variability.

```
# Predict via regularisation on Suburb and Property  Types model
# Calculating RMSE using multiple values of lambda.
lambdas <- seq(0, 10, 0.25)

rmses <- sapply(lambdas, function(l){
```

```
  mu <- mean(train$Price)

  suburb_avgs <- train %>%
          group_by(Suburb) %>%
          summarise(sub = sum(Price - mu)/(n()+l))

  types_avgs <- train %>%
        left_join(suburb_avgs, by= "Suburb") %>%
        group_by(Type) %>%
        summarise(type = sum(Price - sub - mu)/(n()+l))

  predicted_ratings <- test %>%
          left_join(suburb_avgs, by = "Suburb") %>%
          left_join(types_avgs, by = "Type") %>%
          mutate(pred = mu + sub + type) %>%
          pull(pred)

  return(RMSE(predicted_ratings, test$Price))
  })

rmse_regularisation <- min(rmses)
rmse_regularisation
```

```
## [1] 0.1062553
```

```
# Adding RMSE results to the Table
rmse_results <- bind_rows(rmse_results,
                          tibble(Method="Regularised Suburbs and Property
Types Model",
                                 RMSE = rmse_regularisation))
rmse_results %>% knitr::kable()
```

Method	RMSE
Average Price Model	0.1648784
Suburbs Model	0.1263415
Suburbs and Property Types Model	0.1062553
Regularised Suburbs and Property Types Model	0.1062553

The regularized model model doesn't yield any improvement to our previous model.

Linear Regression Model

A linear regression model describes the relationship between a dependent variable, in our case the the Price of the property and one or more independent variables like the Suburbs, Rooms, Property Types etc. We need to set a cross validation range for this model to work. The basic idea behind cross validation consists of dividing the data into training and the test sets which is then used to validate the model by estimating the prediction error.

We train this model with 10 fold cross validation using the variables we explored as the predictors to build this model.

```
# Assigning the cross validating number
control <- trainControl(method = "cv", number = 10)

# Fitting and Predicting using Linear Regression
fit_lm <- train(Price ~ ., method = "lm", data = train, trControl = control)
predictions_lm <- predict(fit_lm, test)
lm_rmse <- RMSE(test$Price, predictions_lm)
lm_rmse

## [1] 0.08696702

# Adding RMSE results to the Table
rmse_results <- bind_rows(rmse_results,
                          tibble(Method="Linear Regression Model",
                                 RMSE = lm_rmse))
rmse_results %>% knitr::kable()
```

Method	RMSE
Average Price Model	0.1648784
Suburbs Model	0.1263415
Suburbs and Property Types Model	0.1062553
Regularised Suburbs and Property Types Model	0.1062553
Linear Regression Model	0.0869670

There is a significant improvement as compared to the previous models and looks very promising. We will try some more regression models to see if we can improve on this.

Regression Tree model

A decision tree algorithm breaks down a data-set into smaller and smaller subsets based on certain conditions. Like a branching tree with leaves and nodes, it starts with a single root node and expands into multiple branches, each representing a decision based on a feature's value. The final leaves of the tree are the possible outcomes or predictions.

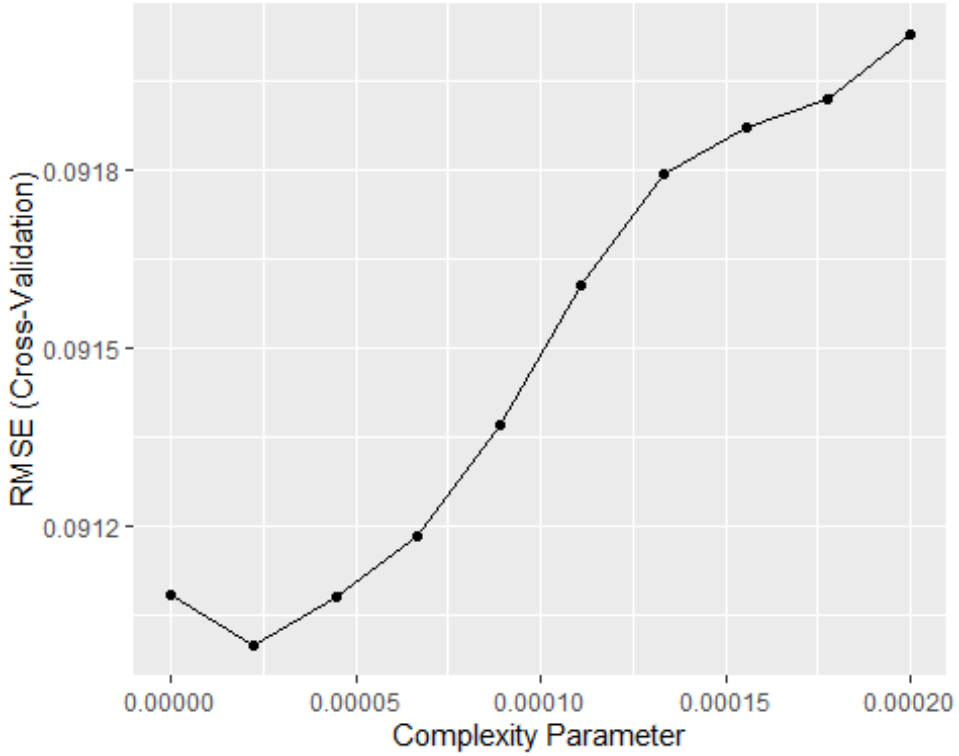
For the regression tree model, we will start by training a small subset with 10 fold cross validation to tune the "complexity parameter". The complexity parameter(cp) helps to determine the right size of the tree.

```
# Plotting different values of the cp parameter and their corresponding rmse
values.
tune <- expand.grid(cp = seq(0, 0.0002, len = 10))
train_rt <- train(Price ~ ., method = "rpart", data = train_subset,
                  trControl = control, tuneGrid = tune)
```

```
# Identify and plot the best tune
train_rt$bestTune
```

```
##             cp
## 2 2.222222e-05
```

```
ggplot(train_rt)
```



Using the best cp parameter we train the regression tree model on the training data-set.

```
fit_rt <- rpart(Price ~ ., data = train,
                control = rpart.control(cp = train_rt$bestTune))
predictions_rt <- predict(fit_rt, test)
rt_rmse <- RMSE(test$Price, predictions_rt)
rt_rmse
```

```
## [1] 0.09302775
```

```
# Adding RMSE results to the Table
rmse_results <- bind_rows(rmse_results,
                          tibble(Method="Regresion Tree Model",
                                 RMSE = rt_rmse))
rmse_results %>% knitr::kable()
```

Method	RMSE
Average Price Model	0.1648784
Suburbs Model	0.1263415

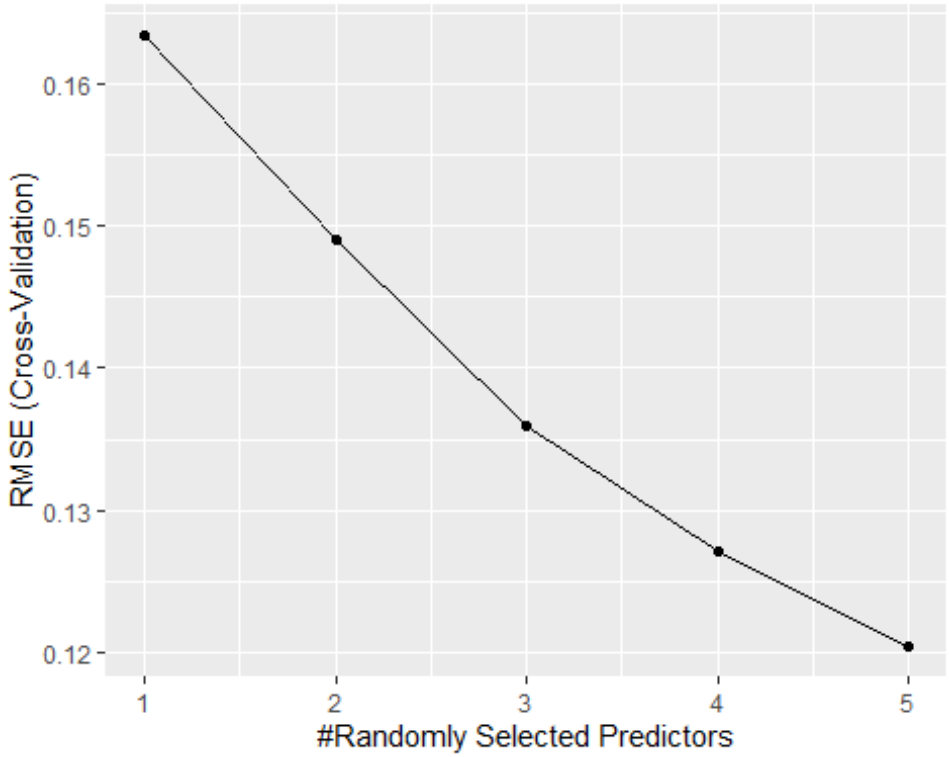
Method	RMSE
Suburbs and Property Types Model	0.1062553
Regularised Suburbs and Property Types Model	0.1062553
Linear Regression Model	0.0869670
Regresion Tree Model	0.0930278

The RMSE though low is not an improvement over the linear regression model we previously looked at.

Random Forest Model

The random forest algorithm is built on the regression tree modeled earlier but additionally grows multiple decision tress and compiles their results into one. Random forest selects random parameters for the decision making to the model while growing the trees which leads to searching for the best feature among a random subset of features.

```
# Using the train_subset to find the best tune for the model
mtry <- round(ncol(train) / 3)
tune <- expand.grid(mtry = seq(mtry - 2, mtry + 2, len = 5))
train_rf <- train(Price ~ ., method = "rf", data = train_subset,
                  ntree = 100, trControl = control, tuneGrid = tune)
ggplot(train_rf)
```




```
# Fitting the best tune to the training set
fit_rf <- randomForest(Price ~ ., data = train,
```

```
                        minNode = train_rf$bestTune$mtry, ntree = 100)

predictions_rf <- predict(fit_rf, test)
rf_rmse <- RMSE(test$Price, predictions_rf)
rf_rmse

## [1] 0.08412544

# Adding RMSE results to the Table
rmse_results <- bind_rows(rmse_results,
                          tibble(Method="Random Forest Model",
                                 RMSE = rf_rmse))
rmse_results %>% knitr::kable()
```

Method	RMSE
Average Price Model	0.1648784
Suburbs Model	0.1263415
Suburbs and Property Types Model	0.1062553
Regularised Suburbs and Property Types Model	0.1062553
Linear Regression Model	0.0869670
Regresion Tree Model	0.0930278
Random Forest Model	0.0841254

The Random Forest Model has yielded the best RMSE yet as compared to all previous models and is huge improvement from the Average Price Model we started with.

Discussions and Conclusion

This table shows us the different models considered and the RMSE obtained from each.

Method	RMSE
Average Price Model	0.1648784
Suburbs Model	0.1263415
Suburbs and Property Types Model	0.1062553
Regularised Suburbs and Property Types Model	0.1062553
Linear Regression Model	0.0869670
Regresion Tree Model	0.0930278
Random Forest Model	0.0841254

The Random Forest Model has been the most successful with 50% improvement from the first model, the Average Price Model has a RMSE value of 0.0841254.

As with any machine learning algorithms, the models could be improved upon by having a larger data-set with even more predictors over a longer time frame as compared to just 2 years as with this data-set. Also, more variables like plot size, number of bathrooms, age of the property, energy efficient ratings etc could help to build more accurate models by factoring all these price dependent elements.

Refrences: The dataset can be sourced from the link and all credits are due to the author for compiling the information. https://www.kaggle.com/datasets/anthonypino/melbourne-housing-market?select=MELBOURNE_HOUSE_PRICES_LESS.csv