Group Assignment

Data Description

This project is based on "House Sales in King County, USA", this dataset includes 21.613 observations and 21 features.

1. "id": Unique observation id.
2. "date": The date upon which the estate was sold.
3. "price": The selling price of the estate.
4. "bedrooms": The number of bedrooms.
5. "bathrooms": The number of bathrooms.
6. "sqft_living": The sqft size of livable area.
7. "sqft_lot": The sqft of the lot of the estate.
8. "floors": The number of floors.
9. "waterfront": Dummy variable is 1 if the estate lies with a waterfront view.
10. "view": A discrete value, 0:4, based on the quality of the view.
11. "condition": Discrete value, 1:5, based on the condition of the estate.
12. "grade": Discrete value, 4:13, describing the quality of the contruction and design.
13. "sqft_above": The sqft of space above ground level.
14. "sqft_basement": The sqft of space below ground level.
15. "yr_built": The year the estate was initially built.
16. "yr_renovated": The year the estate was renovated - No value if not renovated
17. "zipcode": The zipcode of the area in which the estate lies.
18. "lat": The lattitude of the estate.
19. "long": The longitude of the estate.
20. "sqft_living15": The sqft living space of the nearest 15 neighbors.
21. "sqft_lot15": The sqft lot area of the nearest 15 neighbors.

Source: https://www.kaggle.com/harlfoxem/housesalesprediction Dataset Owner: harlfoxem Created: 2016-08-25

Google colab link: https://colab.research.google.com/drive/1JeTp7sTLlDc9HUYlqRCO51T1suHK0QwC?usp=sharing

Data Inspection, Preprocessing and Visualization.

```
data = read_csv("C:/Users/Christian/Documents/Uni/9.Semester/M1/kc_house_data.csv")
```

```
## Rows: 21613 Columns: 21

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr   (1): id
## dbl  (19): price, bedrooms, bathrooms, sqft_living, sqft_lot, floors, waterf...
## dttm  (1): date
```

```
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

In this glimpse command it is shown that the dataset includes 20 numerical features and 1 date feature. To simplify the dataset 6 featues has been removed, these 6 features are; date, view, lat, long, sqft_living15 and sqft_lot15. Date is not included because all the house sales occur within one year and house prices are assumed to be stable in such short timespan. View, lat, long, sqft_living15 and sqft_lot15 will be removed because they add less value than they consume in cluttering.

```
data %>% glimpse()
```

```
## Rows: 21,613
## Columns: 21
## $ id            <chr> "7129300520", "6414100192", "5631500400", "2487200875", ~
## $ date          <dttm> 2014-10-13, 2014-12-09, 2015-02-25, 2014-12-09, 2015-02~
## $ price         <dbl> 221900, 538000, 180000, 604000, 510000, 1225000, 257500,~
## $ bedrooms      <dbl> 3, 3, 2, 4, 3, 4, 3, 3, 3, 3, 3, 2, 3, 3, 5, 4, 3, 4, 2,~
## $ bathrooms     <dbl> 1.00, 2.25, 1.00, 3.00, 2.00, 4.50, 2.25, 1.50, 1.00, 2.~
## $ sqft_living   <dbl> 1180, 2570, 770, 1960, 1680, 5420, 1715, 1060, 1780, 189~
## $ sqft_lot      <dbl> 5650, 7242, 10000, 5000, 8080, 101930, 6819, 9711, 7470,~
## $ floors        <dbl> 1.0, 2.0, 1.0, 1.0, 1.0, 1.0, 2.0, 1.0, 1.0, 2.0, 1.0, 1~
## $ waterfront    <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ view          <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0,~
## $ condition     <dbl> 3, 3, 3, 5, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 3, 3, 3, 4, 4,~
## $ grade         <dbl> 7, 7, 6, 7, 8, 11, 7, 7, 7, 7, 8, 7, 7, 7, 7, 9, 7, 7, 7~
## $ sqft_above    <dbl> 1180, 2170, 770, 1050, 1680, 3890, 1715, 1060, 1050, 189~
## $ sqft_basement <dbl> 0, 400, 0, 910, 0, 1530, 0, 0, 730, 0, 1700, 300, 0, 0, ~
## $ yr_built      <dbl> 1955, 1951, 1933, 1965, 1987, 2001, 1995, 1963, 1960, 20~
## $ yr_renovated  <dbl> 0, 1991, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ zipcode       <dbl> 98178, 98125, 98028, 98136, 98074, 98053, 98003, 98198, ~
## $ lat           <dbl> 47.5112, 47.7210, 47.7379, 47.5208, 47.6168, 47.6561, 47~
## $ long          <dbl> -122.257, -122.319, -122.233, -122.393, -122.045, -122.0~
## $ sqft_living15 <dbl> 1340, 1690, 2720, 1360, 1800, 4760, 2238, 1650, 1780, 23~
## $ sqft_lot15    <dbl> 5650, 7639, 8062, 5000, 7503, 101930, 6819, 9711, 8113, ~
```

In this step we'll define the selected columns and redefine the yr_renovated column to be a binary variable with the name renovated, this is to define whether the estate has been renovated at all or not. The reasoning behind this, is that few estates has been renovated and therefore the year is irrelevant.

```
# Selecting data
data1 = data %>%
  select(2:9, 11:17)%>%
  mutate(yr_renovated = ifelse(yr_renovated==0, 0, 1), sqft_living = sqft_living*0.09290304,
                        sqft_lot = sqft_lot*0.09290304,
                        sqft_above = sqft_above*0.09290304,
                        sqft_basement = sqft_basement*0.09290304) %>%
  rename(renovated = yr_renovated, m2_living = sqft_living, m2_lot = sqft_lot,
                      m2_above = sqft_above, m2_basement = sqft_basement)
```

After the selection of columns the dataset includes 15 columns.

The next step is to clean the data for extreme values. It is observed in the head command that one house has 33 bedrooms, which seems to be very high considering it only has 1.75 bathrooms and 150 m2 of living space, therefore this is removed.

```
data1 %>%
  arrange(desc(bedrooms)) %>%
  head()
```

```
## # A tibble: 6 x 15
##   date                price bedrooms bathrooms m2_living m2_lot floors
##   <dttm>              <dbl>    <dbl>     <dbl>     <dbl>  <dbl>  <dbl>
## 1 2014-06-25 00:00:00  640000       33      1.75      151.   557.    1
## 2 2014-08-21 00:00:00  520000       11      3         279.   461.    2
## 3 2014-08-14 00:00:00 1148000       10      5.25      426.  1015.    1
## 4 2014-10-29 00:00:00  650000       10      2         335.  1107.    2
## 5 2014-12-29 00:00:00  660000       10      3         271.   348.    2
## 6 2014-05-07 00:00:00  599999        9      4.5       356.   649.    2.5
## # ... with 8 more variables: waterfront <dbl>, condition <dbl>, grade <dbl>,
## #   m2_above <dbl>, m2_basement <dbl>, yr_built <dbl>, renovated <dbl>,
## #   zipcode <dbl>
```

Next we will define a house to be livable when it has at least 1 bedroom and 1 bathroom.

```
data1 %>%
  arrange(bathrooms) %>%
  head()
```

```
## # A tibble: 6 x 15
##   date                price bedrooms bathrooms m2_living m2_lot floors
##   <dttm>              <dbl>    <dbl>     <dbl>     <dbl>  <dbl>  <dbl>
## 1 2014-06-12 00:00:00 1095000        0         0     285.   443.     3.5
## 2 2015-02-17 00:00:00   75000        1         0      62.2 4030.     1
## 3 2015-02-05 00:00:00  380000        0         0     137.    91.0    3
## 4 2014-11-04 00:00:00  280000        1         0      55.7 2276.     1
## 5 2014-06-24 00:00:00 1295650        0         0     447.  2602.     2
## 6 2015-04-29 00:00:00  355000        0         0     229.   748.     2
## # ... with 8 more variables: waterfront <dbl>, condition <dbl>, grade <dbl>,
## #   m2_above <dbl>, m2_basement <dbl>, yr_built <dbl>, renovated <dbl>,
## #   zipcode <dbl>
```

Filtering the data to only include houses with less than 33 bedrooms and more than or equal to 1 bedroom and bathroom.

```
data1 = data1 %>%
  filter(bedrooms < 33 & bedrooms >= 1 & bathrooms >= 1)

data1 %>% arrange(bedrooms) %>% head()
```

```
## # A tibble: 6 x 15
##   date                price bedrooms bathrooms m2_living m2_lot floors
##   <dttm>              <dbl>    <dbl>     <dbl>     <dbl>  <dbl>  <dbl>
## 1 2014-05-16 00:00:00 350000        1       1        65.0   474.      1
## 2 2014-11-18 00:00:00 157000        1       1        80.8  2446.      1
## 3 2014-07-14 00:00:00 200000        1       1.5      93.8   107.      2
## 4 2014-11-05 00:00:00 140000        1       1        67.8   640.      1
```

```
## 5 2014-09-17 00:00:00 279200        1       1        59.5   590.      1
## 6 2015-05-01 00:00:00 250000        1       1        61.3   242.      1
## # ... with 8 more variables: waterfront <dbl>, condition <dbl>, grade <dbl>,
## #   m2_above <dbl>, m2_basement <dbl>, yr_built <dbl>, renovated <dbl>,
## #   zipcode <dbl>
```

A quick apply command is utilized to check if any of the variables contain any NA or infinite values.

```
data1 %>%
  apply(2, function(x) any(is.na(x) | is.infinite(x)))
```

```
##        date       price    bedrooms   bathrooms   m2_living      m2_lot
##       FALSE       FALSE       FALSE       FALSE       FALSE       FALSE
##      floors  waterfront   condition       grade    m2_above m2_basement
##       FALSE       FALSE       FALSE       FALSE       FALSE       FALSE
##    yr_built   renovated     zipcode
##       FALSE       FALSE       FALSE
```

The data is clear of NA's and infinite values.

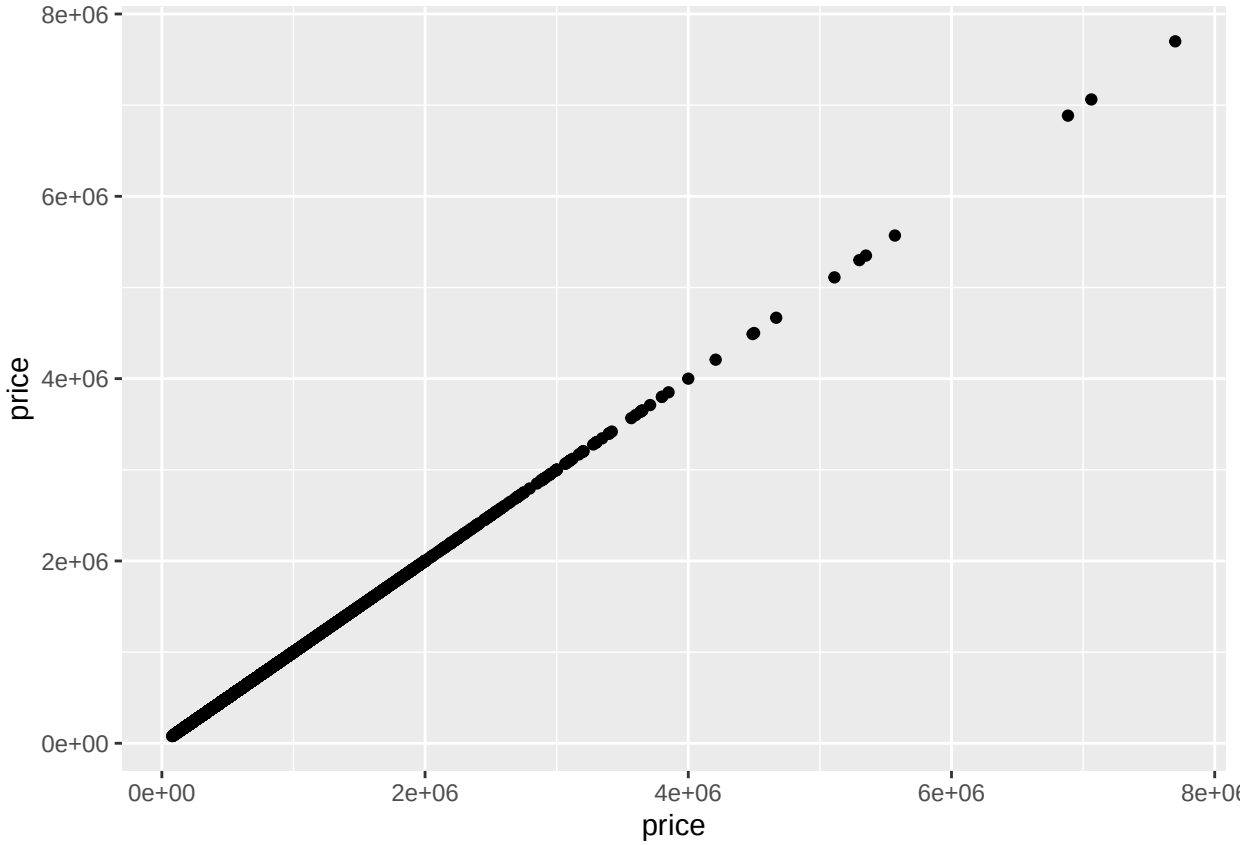
```
data1 %>%
  select(!date) %>%
  summarise(across(everything(), list(min=min, mean=mean, max=max, sd=sd),
                   .names = "{.col}-{.fn}")) %>%
  round(2) %>%
  pivot_longer(everything(), names_sep = "-", names_to = c("variable", ".value"))
```

```
## # A tibble: 14 x 5
##    variable        min      mean       max        sd
##    <chr>         <dbl>     <dbl>     <dbl>     <dbl>
##  1 price       78000   541064.   7700000   367400.
##  2 bedrooms        1        3.38      11        0.9
##  3 bathrooms       1        2.12       8        0.77
##  4 m2_living      36.2    194.      1258.      85.2
##  5 m2_lot         48.3   1402.    153416.    3853.
##  6 floors          1        1.5        3.5      0.54
##  7 waterfront      0        0.01       1        0.09
##  8 condition       1        3.41       5        0.65
##  9 grade           4        7.66      13        1.17
## 10 m2_above       36.2    166.       874.      76.8
## 11 m2_basement     0       27.2      448.      41.2
## 12 yr_built     1900     1971.      2015       29.4
## 13 renovated       0        0.04       1        0.2
## 14 zipcode     98001    98078.     98199       53.5
```

Here is a summerization of the data, with min, mean, max and standard deviation values for each variable.

Next we will try to examine possible connections between variables and price. The next plot shows the distribution of estates based on price.

```
data1 %>%
  ggplot(aes(x=price,y=price)) +
  geom_point()
```



The plot visualizes that the dataset includes less estates the higher the price. It is expected that waterfront estates are more expensive. This is examined through an average price of estates with and without the waterfront tag.

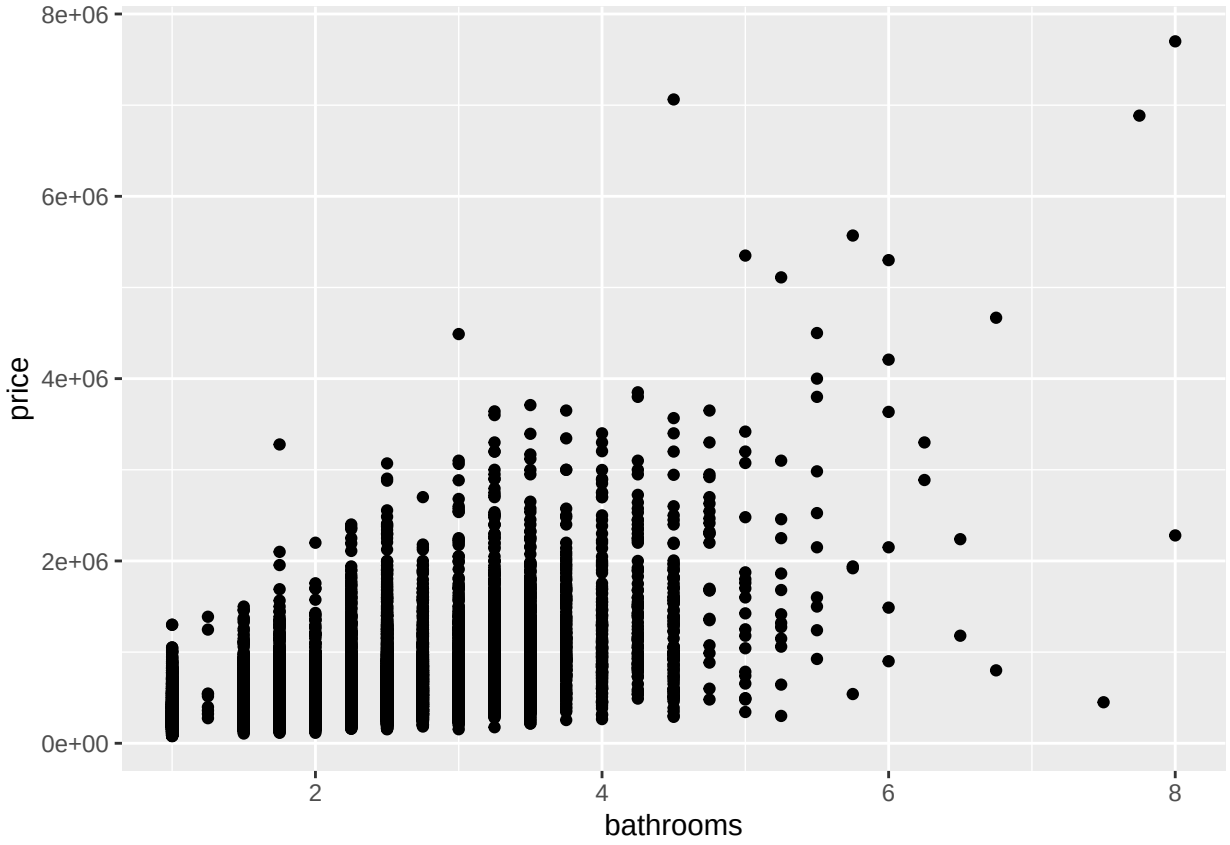
```
data1 %>%
  group_by(waterfront) %>%
  summarize(avg = mean(price))
```

```
## # A tibble: 2 x 2
##   waterfront      avg
##        <dbl>    <dbl>
## 1          0  532481.
## 2          1 1708967.
```

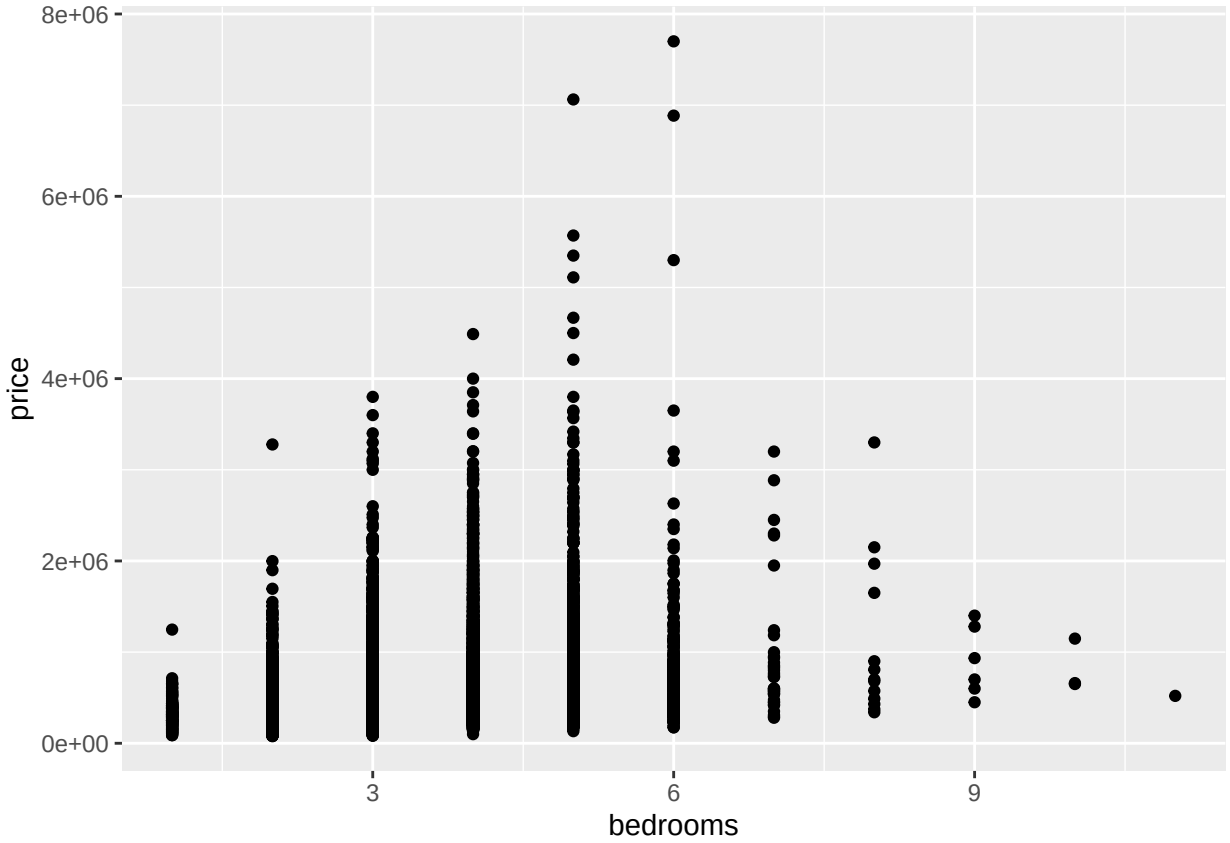
The hypothesis is true, the price does get more expensive when lying on the waterfront.

Next the number of bathrooms and bedrooms are plotted against the price, it is expected that the price will increase with the number of rooms.

```
data1 %>%
  ggplot(aes(x=bathrooms,y=price)) +
  geom_point()
```



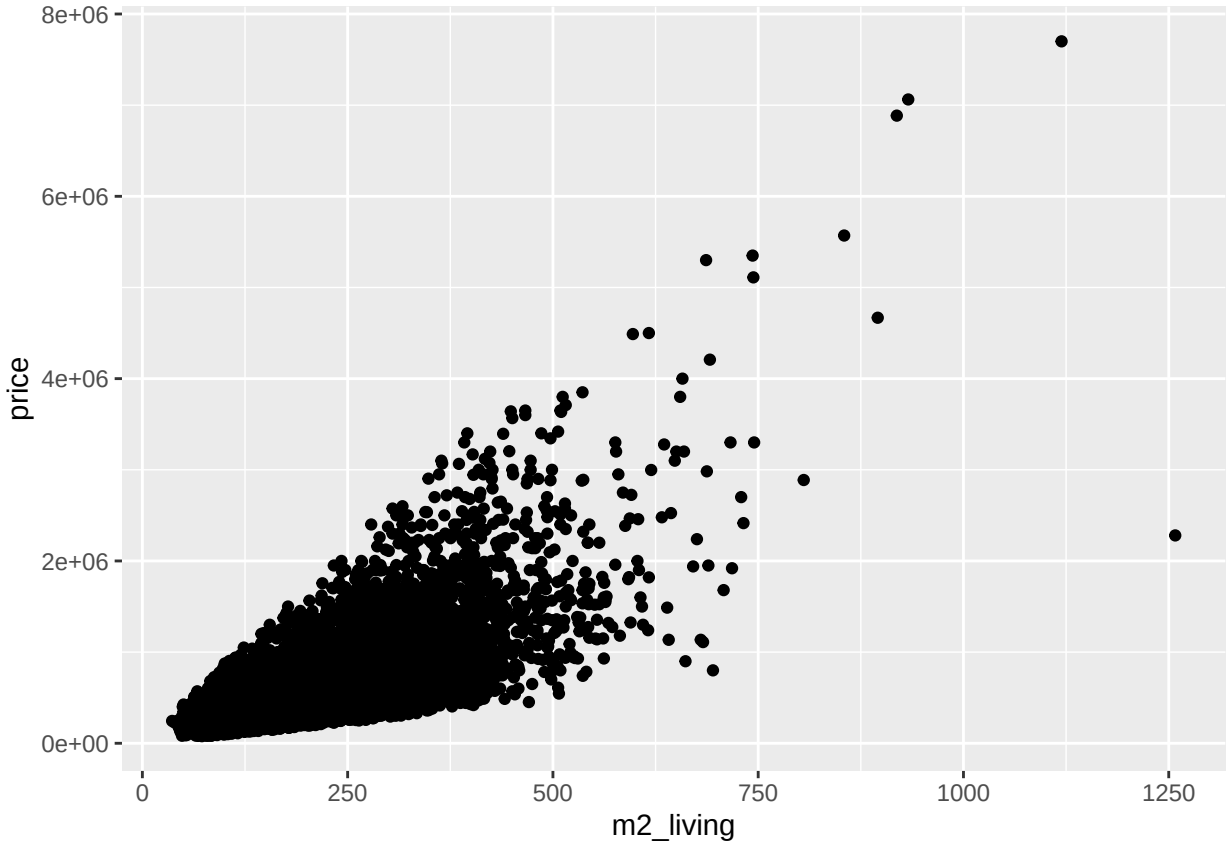

```
data1 %>%
  ggplot(aes(x=bedrooms,y=price)) +
  geom_point()
```



Surprisingly the price seems to have a correlation with the number of bathrooms but not the number of bedrooms.

The next plot visualizes the connection between living space and price. Same here is that bigger houses is more expensive.

```
data1 %>%
  ggplot(aes(x=m2_living,y=price)) +
  geom_point()
```



Here the connection is very clear.

Lastly the zipcodes are examined, it is expected that the zipcodes have some sort of relation to the price.

```
data1 %>%
  group_by(zipcode) %>%
  summarize(avg = mean(price)) %>%
  arrange(desc(avg)) %>%
  head(10)
```

```
## # A tibble: 10 x 2
##    zipcode      avg
##      <dbl>    <dbl>
##  1   98039 2160607.
##  2   98004 1355927.
##  3   98040 1194230.
##  4   98112 1096975.
##  5   98102  899395.
##  6   98109  879624.
##  7   98105  862825.
##  8   98006  860917.
##  9   98119  851084.
## 10   98005  810165.
```

It seems that some neighborhoods are more expensive than others. Because of this we will redefine zipcode, to better distinguish the cheaper zipcodes from the more expensive ones. Where the higher number is more

expensive. zip_index is the new variables, which gives each zipcode a value from 0-1 based on the mean price in the area, indexed against the most expensive zipcode.

```
data_zip = data1 %>%
  group_by(zipcode) %>%
  summarize(avg = mean(price)) %>%
  arrange(desc(avg)) %>%
  mutate(zip_index = avg/2160606.6)

data2 = data1 %>%
  left_join(data_zip, by="zipcode") %>%
  select(-avg, -zipcode)
```

Unsupervised Machine Learning

First part of Unsupervised Machine Learning is to scale the data, so that all variables has mean 0 and standard deviation of 1. This assures that the variance of the variables are not widely different, also it assures that variables in different scales of measurement(like $ and m2) is weighted equally.

```
data_UML = data2 %>%
    select(!date) %>%
    scale()

data_UML %>%
    colMeans()
```

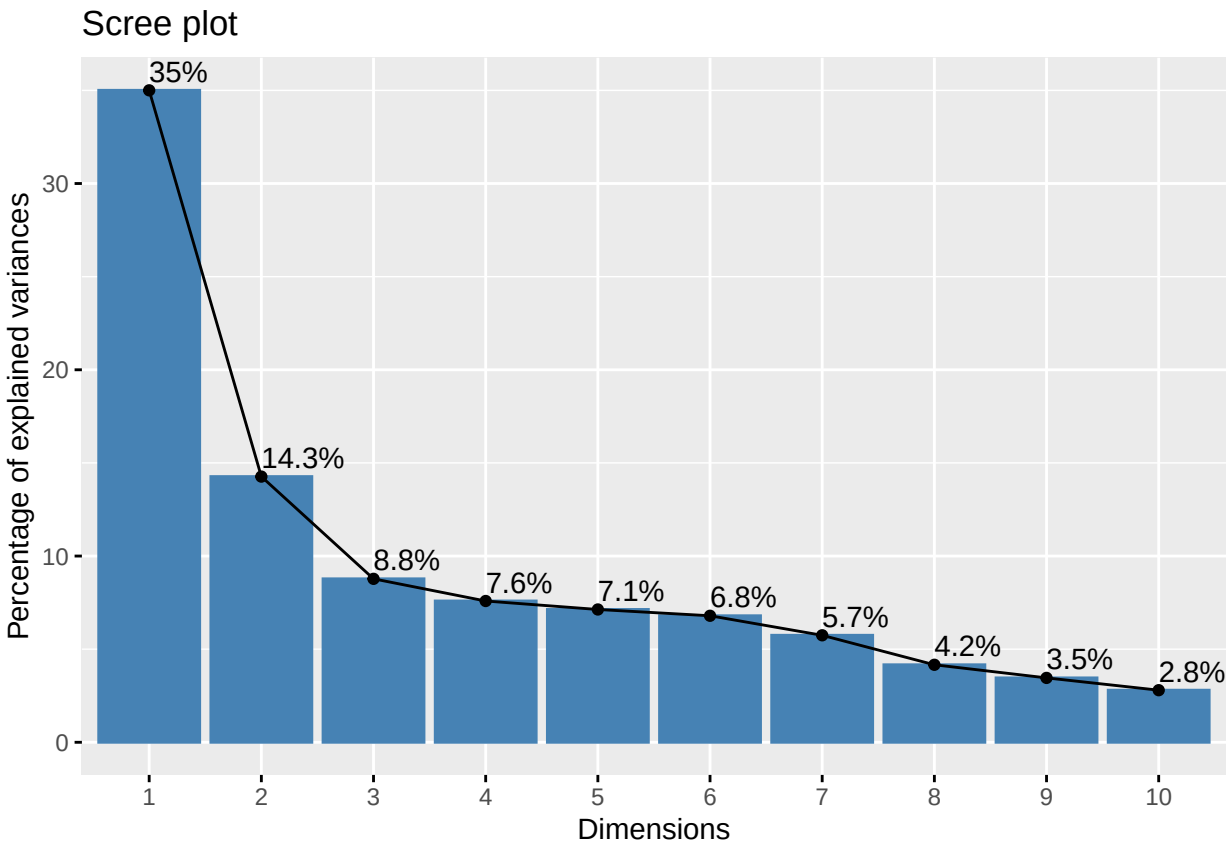
```
##         price      bedrooms     bathrooms     m2_living        m2_lot
## -1.011569e-16 -1.459318e-16  2.126414e-16  5.344554e-17 -2.873889e-18
##        floors    waterfront     condition         grade      m2_above
## -8.679147e-17  1.581169e-18  1.014218e-17 -1.585038e-16 -3.021245e-17
##   m2_basement      yr_built     renovated     zip_index
## -1.371706e-17  3.295285e-15  3.442460e-17  1.678941e-16
```

Due to the colMeans function it is possible to see that all the variables have mean 0 and are scaled correctly.

Dimensionality Reduction

First of all we'll do a PCA. To determine how many components to include it is possible to create a screeplot, this plot shows how much of the variance in percentages that each components/dimension explains. These are ranked highest to lowest. Normally the amount of components to include would be based on where the elbow would be visible, however this can sometimes be quite hard to spot. In this example it looks like the elbow occurs at 3 components. The reasoning behind the elbow is that there is a steep drop-off in the variance explained and therefore you would not include more components, since including more would lead to a cluttered model.

```
res_pca = data_UML %>%
    PCA(graph = FALSE)

res_pca %>% fviz_screeplot(addlabels = TRUE, ncp = 10, ggtheme = theme_gray())
```



PCA

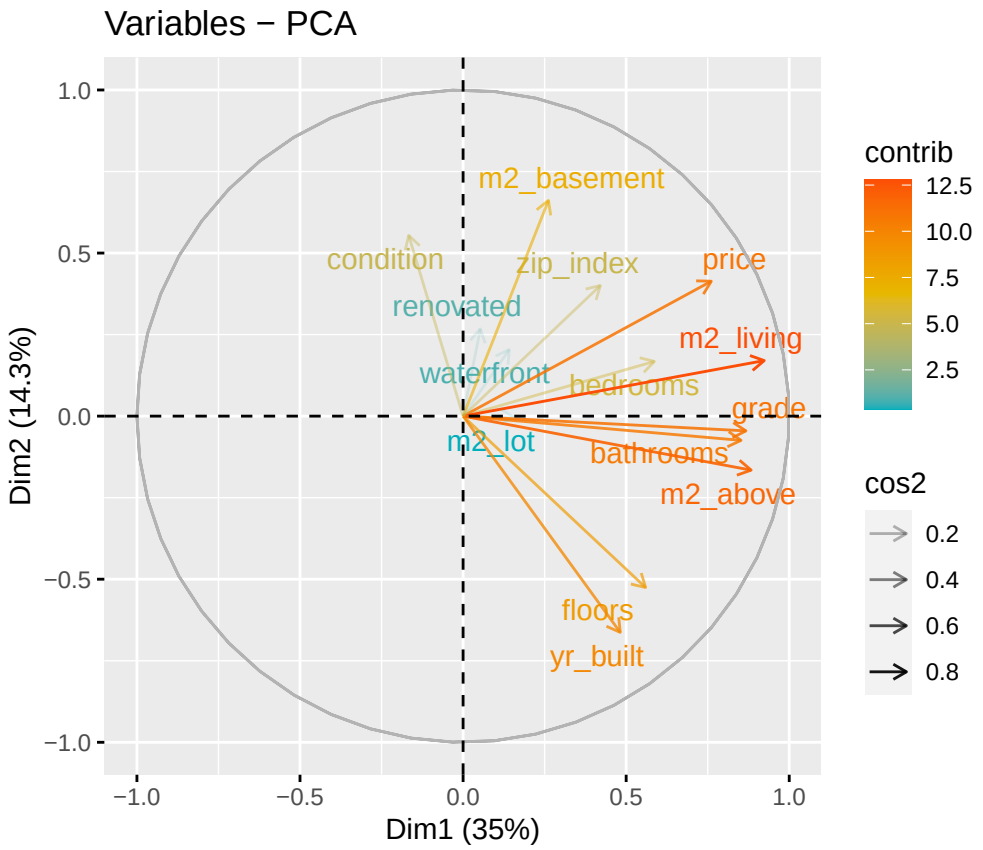
An alternative way to determine the number of components is to look at the eigenvalues. The number of components to include is based on the number of components with an eigenvalue of more than 1.

```
res_pca$eig %>% as_tibble()
```

```
## # A tibble: 14 x 3
##    eigenvalue `percentage of variance` `cumulative percentage of variance`
##         <dbl>                    <dbl>                                <dbl>
##  1   4.90e+ 0                3.50e+ 1                                 35.0
##  2   2.00e+ 0                1.43e+ 1                                 49.3
##  3   1.23e+ 0                8.78e+ 0                                 58.0
##  4   1.06e+ 0                7.59e+ 0                                 65.6
##  5   9.98e- 1                7.13e+ 0                                 72.8
##  6   9.51e- 1                6.79e+ 0                                 79.6
##  7   8.04e- 1                5.75e+ 0                                 85.3
##  8   5.83e- 1                4.16e+ 0                                 89.5
##  9   4.84e- 1                3.46e+ 0                                 92.9
## 10   3.92e- 1                2.80e+ 0                                 95.7
## 11   2.46e- 1                1.76e+ 0                                 97.5
## 12   1.98e- 1                1.41e+ 0                                 98.9
## 13   1.55e- 1                1.11e+ 0                                100
## 14   1.34e-29                9.56e-29                                100
```

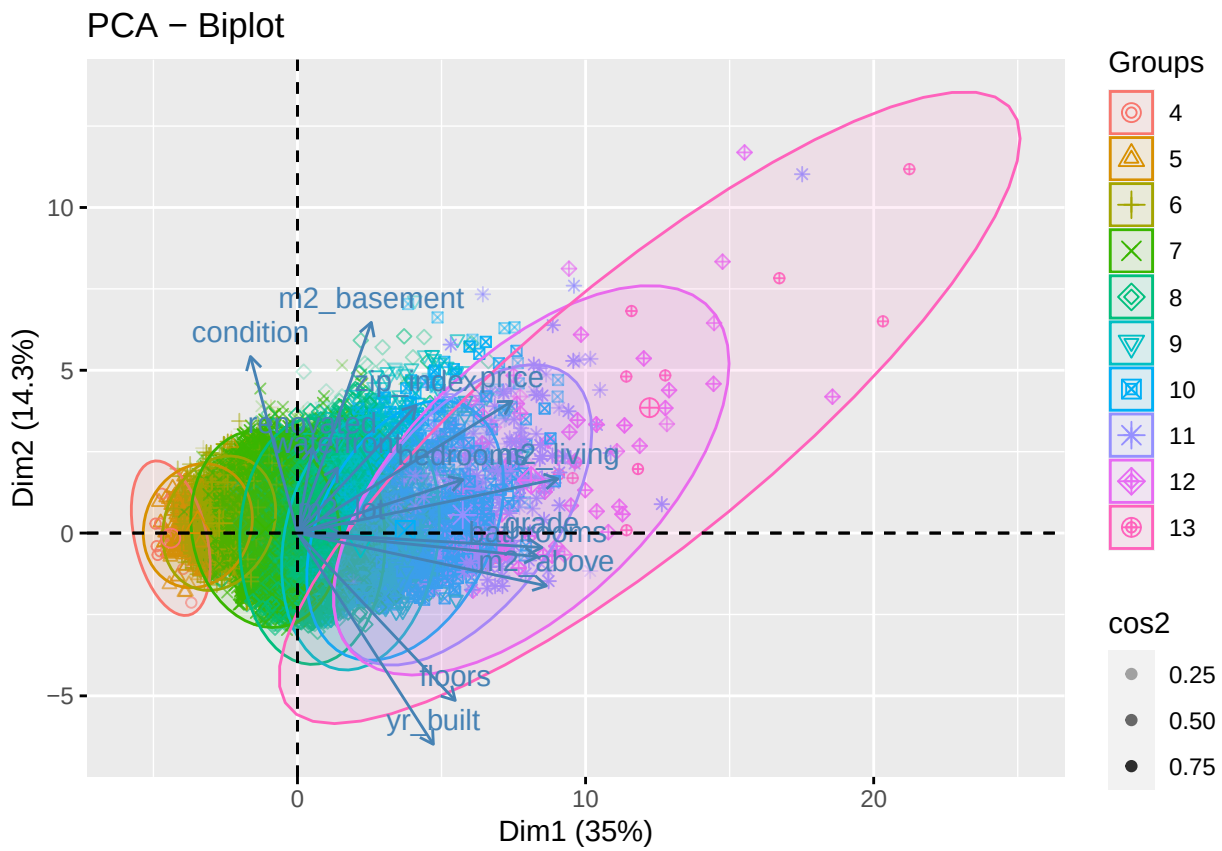
Based on the eigenvalues the number of components should be 4. Furthermore it is possible to see the cumulative percantage of variance, which describes how much of the variance is explained by the number of components. With 4 components, about 65% of the variance is explained.

```
res_pca = data_UML %>%
    PCA(ncp = 4, graph = FALSE)

res_pca %>%
  fviz_pca_var(alpha.var = "cos2",
               col.var = "contrib",
               gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
               repel = TRUE,
               ggtheme = theme_gray())
```



The biplot plot above shows how much each variable contributes to the first two dimensions. E.g. the variables that contribute to the first dimension will point along the x-axis and the second dimension of the y-axis. The contribution is measured by the color and the length of the vector. The variables are correlated if they point in the same direction, has a negative correlation if the point directly opposite each other and has 0 correlation if they are orthogonal to each other. It looks like m2_living has the most significant impact on price, due to its correlation and contribution.

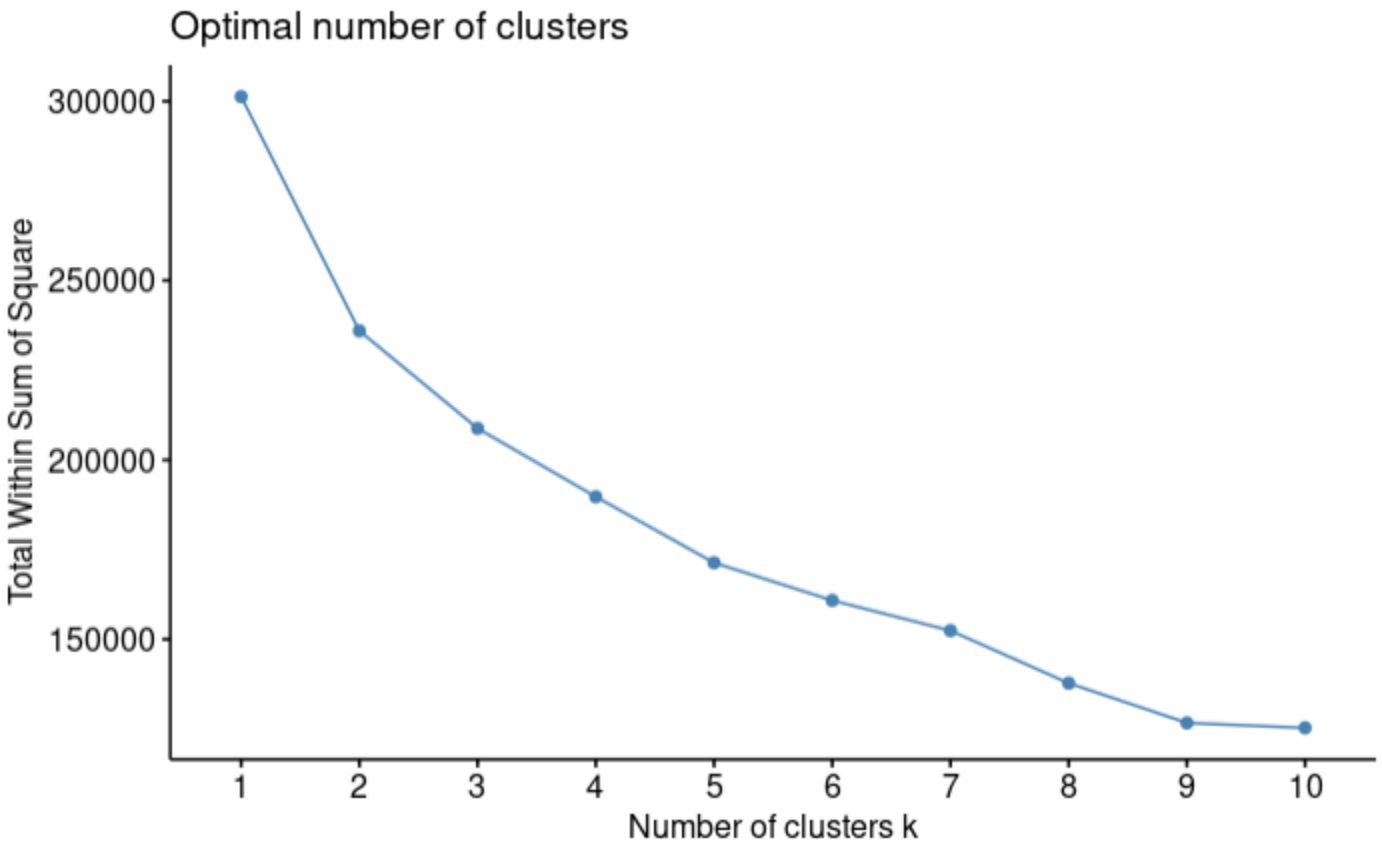
```
res_pca %>%
      fviz_pca_biplot(alpha.ind = "cos2", habillage = data1 %>% pull(grade) %>%
      factor(), addEllipses = TRUE, geom = "point", ggtheme = theme_gray())
```



This plot shows how the grade is distributed across the two components, where it seems like the grade has a significant influence. Recall that price was in the first quadrant with about at 40 degree angle.

Clustering In this part we will experience the data clustering through a kmeans approach. Kmeans utilizes centroids through its clustering. Each centroid is randomly placed and assigned all the closests datapoints to its cluster, the mean point is then calculated for each centroid and then the closests datapoint are distributed again, this procedure is repeated multiple times until the datapoints are stable within the clusters.

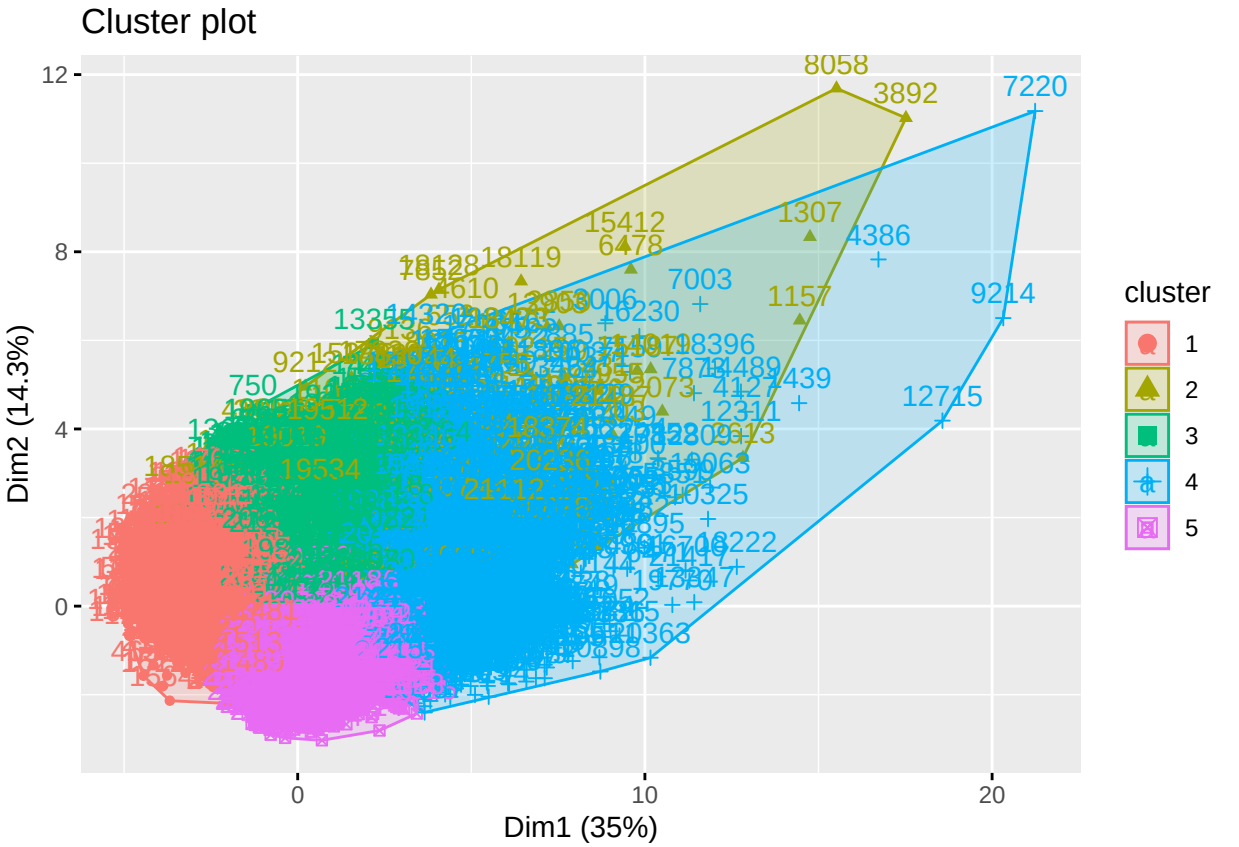
```
data2 %>%
  select(-date) %>%
  scale() %>%
  fviz_nbclust(kmeans, method = "wss")
```



Here is the screeplot, there is a dropoff in variance explained at about 5 clusters, but it is hard to determine. Due to this, 5 clusters have been chosen for further use.

Next the kmeans function is utilized with the 5 clusters and 20 iterations.

```
set.seed(12)
res_km = data_UML %>% kmeans(centers = 5, nstart = 20)

data4 =  data2 %>% select(-date)

res_km %>%
  fviz_cluster(data = data2 %>% select(-date) ,
               ggtheme = theme_gray())
```



This plot shows the distribution of the clusters within the PCA framework, with the two first dimensions. There is an observable separation between the clusters, however the second and fourth cluster are overlapping. This can be explained by including more dimensions, but that is not feasible to show in a plot. To further investigate the distribution of clusters, it is possible to make tables based on chosen variables in the dataset and combine it with the clusters. Firstly with the waterfront variable:

```
table(data2$waterfront, res_km$cluster, dnn = c("Waterfront", "Cluster"))
```

```
##           Cluster
## Waterfront    1    2    3    4    5
##          0 7831    0 4754 2573 6206
##          1    0  157    0    0    0
```

There is a very clear cluster of waterfront estates, within the second cluster. The kmeans clustering managed to separate these completely. Next the variable grade is included.

```
table(data2$grade, res_km$cluster, dnn = c("Grade", "Cluster"))
```

```
##       Cluster
## Grade    1    2    3    4    5
##     4   14    0    0    0    0
##     5  221    2    4    0    0
##     6 1859    9  118    1   23
##     7 5012   22 2207   28 1687
##     8  713   40 1966  198 3147
```

```
##    9    12   25  418  924 1236
##    10    0   28   41  953  112
##    11    0   20    0  378    1
##    12    0   11    0   78    0
##    13    0    0    0   13    0
```

Here it is apparent that the clusters have some sort of distinguishment of grade as well. However it is not as clear as the waterfront variable. The first cluster is generally lower in grade compared the to other clusters. Cluster 3 and 5 are in the middle and the cluster with the highest grades is the fourth cluster.

The next table includes price divided into 9 sections (There are 9 sections in grade, so we used 9 sections here aswell), where the higher is the more expensive. Here it is easy to see that cluster 2 and 4 are generally more expensive. Recall that the second cluster was solely waterfront estates, thereby the waterfront tag seems to have an impact on the price. The lower graded houses are typically cheaper. The fifth cluster seem to be in the middle class. If we combine the earlier table with this one, it gives the impression that the third cluster is more expensive when considering the lower grade.

```
data2 = data2 %>%
  mutate(quantile = ntile(price, 9))

table(data2$quantile, res_km$cluster, dnn = c("Price", "Cluster"))
```

```
##      Cluster
## Price    1    2    3    4    5
##     1 2081    0  104    0  207
##     2 1389    1  285    2  715
##     3 1101    2  382    2  904
##     4 1069    3  418    4  897
##     5  882    4  574   25  906
##     6  661    4  699   55  972
##     7  415   13  849  155  959
##     8  198   15  855  760  563
##     9   35  115  588 1570   83
```

Lastly we can do the same for our zip_index variable, this could explain the price/grade asymmetry in the third cluster. Here the zip_index is divided into 9 sections like price was before. The second cluster which was in the higher price range and the middle grade range, is fairly equally distributed across zip_codes, which means that waterfront estates are more expensive no matter the zipcode. The third and fourth cluster lie in the more expensive zipcodes. This could explain the higher prices in the third cluster, and therefore it makes sense for those to be lower grade. The fourth cluster is generally the "best" houses, which are of the highest grades, neighborhood and so the price.

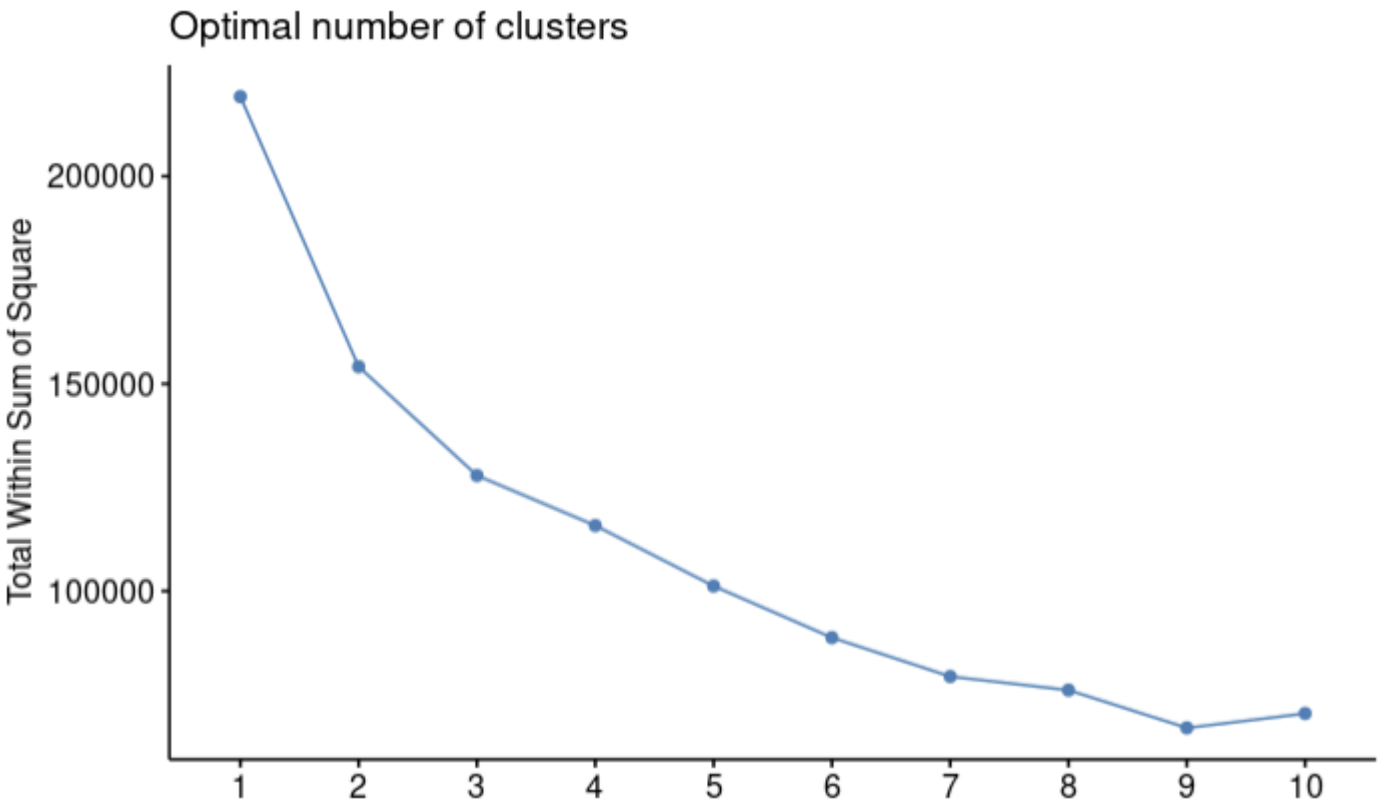
```
data2 = data2 %>%
  mutate(zip_quantile = ntile(zip_index, 9))

table(data2$zip_quantile, res_km$cluster, dnn = c("Zip_Sections", "Cluster"))
```

```
##             Cluster
## Zip_Sections    1    2    3    4    5
##            1 1282    4  373   39  694
##            2 1145   18  343   61  825
##            3 1099   12  416   66  798
##            4 1062   24  471  126  708
```

```
##             5  864   38  469  228  792
##             6  820    2  549  187  833
##             7  701   10  730  320  630
##             8  501   26  520  710  634
##             9  357   23  883  836  292
```

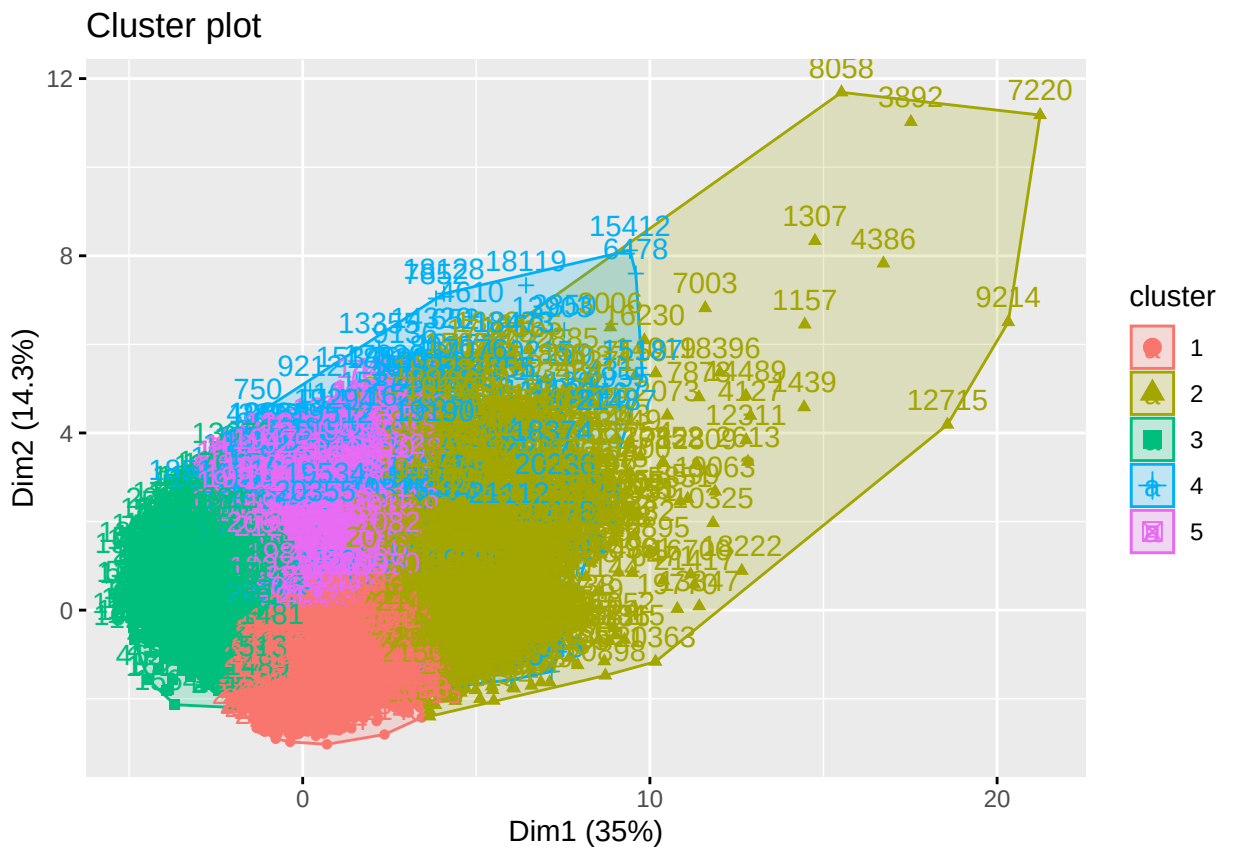
Now the same analysis will be done on data from the PCA after the dimensionality reduction, to examine if the PCA improves the clustering.

```
res_pca$ind$coord %>%
  fviz_nbclust(kmeans, method = "wss")
```



Not much has changed on the screeplot and 5 clusters will be used again like earlier for consistency.

```
set.seed(12)
res_km_pca = res_pca$ind$coord %>% kmeans(centers = 5, nstart = 20)

res_km_pca %>%
  fviz_cluster(data = data4,
               ggtheme = theme_gray())
```



The plot shows some changes to the clustering compared to the Kmeans without PCA, however this does not mean it's an improvement yet. In these tables and the plot above the clusters have switched places. Cluster 1 is now 5, cluster 2 is now 4, 3 is 1, 4 is 2 and 5 is 3.

```
table(data2$waterfront, res_km_pca$cluster, dnn = c("Waterfront", "Cluster"))
```

```
##            Cluster
## Waterfront    1    2    3    4    5
##          0 6458 2104 7539  730 4533
##          1    0    8    0  149    0
```

The PCA data made the separation of waterfronts less accurate, cluster 4 (was 2) now also includes several non waterfront estates.

```
table(data2$grade, res_km_pca$cluster, dnn = c("Grade", "Cluster"))
```

```
##       Cluster
## Grade    1    2    3    4    5
##     4    0    0   14    0    0
##     5    0    0  219    4    4
##     6   10    0 1858   42  100
##     7 1596   27 4878  299 2156
##     8 3194  138  563  295 1874
##     9 1435  662    7  150  361
##     10 214  819    0   63   38
```

```
##    11    9  372    0   18    0
##    12    0   81    0    8    0
##    13    0   13    0    0    0
```

It seems that the clusters all are pulled towards the middle of the grade scale.

```
table(data2$quantile, res_km_pca$cluster, dnn = c("Price", "Cluster"))
```

```
##      Cluster
## Price    1    2    3    4    5
##     1  184    0 2095   17   96
##     2  698    1 1385   26  282
##     3  890    1 1077   40  383
##     4  891    4 1022   59  415
##     5  899   24  829   75  564
##     6  986   42  592   96  675
##     7 1013   91  353  102  832
##     8  765  523  156  165  782
##     9  132 1426   30  299  504
```

In the second cluster (was 4) there has been a reduction of estates in the upper price range. In the first cluster (was 3) there has been an increase in the amount of less expensive estates.

```
table(data2$zip_quantile, res_km_pca$cluster, dnn = c("Zip_Sections", "Cluster"))
```

```
##             Cluster
## Zip_Sections    1    2    3    4    5
##            1  704   32 1269   20  367
##            2  825   45 1112   78  332
##            3  788   42 1082   95  384
##            4  714   89 1029  103  456
##            5  820  177  832  104  458
##            6  835  155  789  106  506
##            7  676  245  657  114  699
##            8  750  595  449   96  501
##            9  346  732  320  163  830
```

For the zipcodes not much have changed, except the number of estates included in the second cluster (was 4).

Overall the PCA didn't seem to improve the clustering of the data.

Supervised Machine Learning

The purpose of this section is to fit a regression model within the SML setting, which can be used to predict the prices of estates in King County. The generated model will produce the best results when performed on this dataset, however it may be useful on other datasets as long as the general characteristics of the area does not change, like culture, climate and other such values.

```
set.seed(12)
data_SML = data4 %>% rename(y = price)

data_split <- initial_split(data_SML, prop = 0.75, strata = y)

data_train <- data_split  %>%  training()
data_test <- data_split %>% testing()
```

This is to restrict the data, so that the last 25% are excluded from the training data, this will later be used to verify the quality of the predictions from the regression model.

Next step is to create the data_recipe on which the models will be fitted, the data has to be scaled like earlier with mean 0 and standard deviation of 1. Furthermore will predictors with zero variance be removed because, they don't aid the predictions, since they are constants.

```
set.seed(12)
data_recipe <- data_train %>%
  recipe(y ~.) %>%
  step_center(all_numeric(), -all_outcomes()) %>% # Centers all numeric variables to mean = 0
  step_scale(all_numeric(), -all_outcomes()) %>% # scales all numeric variables to sd = 1
  step_nzv(all_predictors())  %>% # Removed predictors with zero variance
  prep()
```

Three models will be made, to find out which one is best. The models are based on linear regression, random forest and xgboost methods. The random forest and xgboost regressions can define more complex connections than the linear models normally are able to, however it is the most simple model and therefore it is included. Some parameters in the random forest and xgboost models have to be tuned, this is done in a later step.

```
model_lm <- linear_reg(mode = 'regression') %>%
  set_engine('lm')
```

```
model_rf <- rand_forest(mode = 'regression',
                        trees = 25,
                        mtry = tune(),
                        min_n = tune()
                        ) %>%
  set_engine('ranger', importance = 'impurity')
```

```
model_xg <- boost_tree(mode = 'regression',
                       trees = 100,
                       mtry = tune(),
                       min_n = tune(),
                       tree_depth = tune(),
                       learn_rate = tune()
                       ) %>%
  set_engine("xgboost")
```

Next we split the data into equally sized folds, 5 in this case. The number of folds are based on the size of the dataset, since our dataset is relatively large, 5 folds should be sufficient. By stratifying the data, each fold has an equal distribution of y.

```
set.seed(12)
data_resample <- data_train %>%
  vfold_cv(strata = y,
           v = 5,
           repeats = 3)
```

Next the workflows for each model is defined, this is based on the first general workflow, which combines the data recipe with the models.

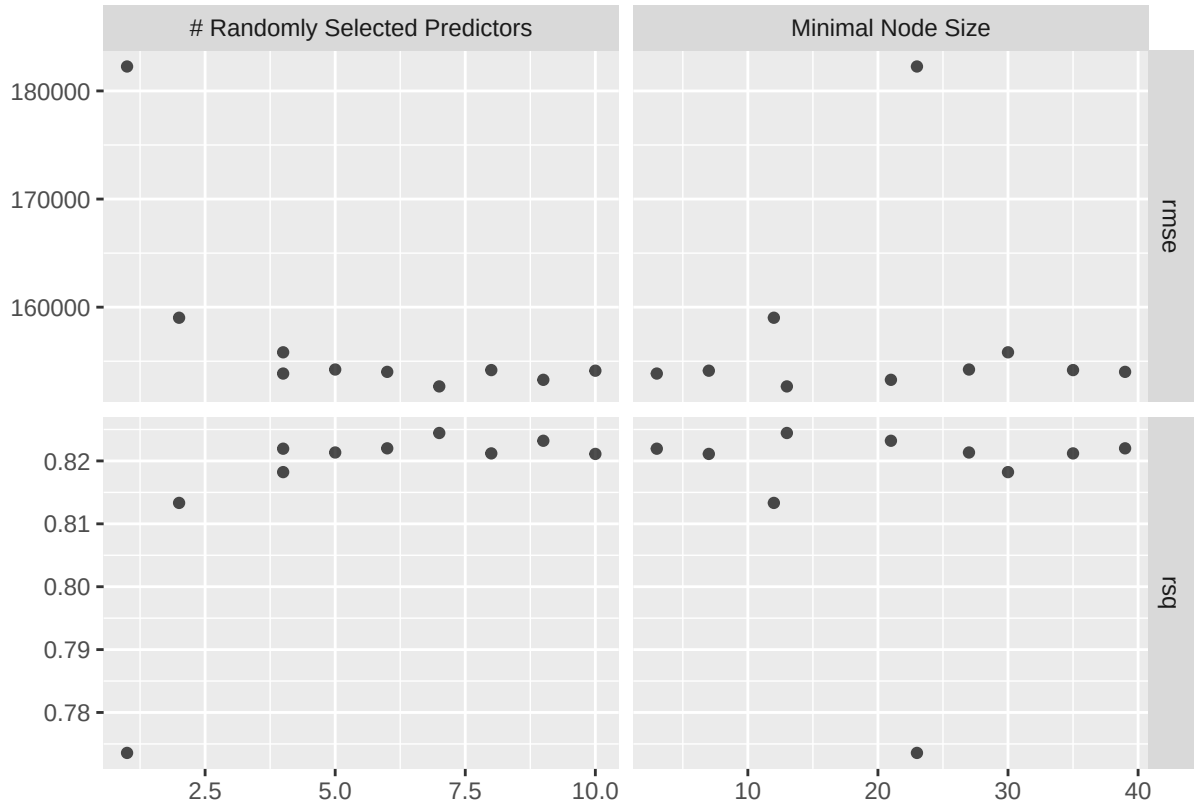
```
workflow_general <- workflow() %>%
  add_recipe(data_recipe)

workflow_lm <- workflow_general %>%
  add_model(model_lm)

workflow_rf <- workflow_general %>%
  add_model(model_rf)

workflow_xg <- workflow_general %>%
  add_model(model_xg)
```

Here the tuning for the random forest model will be done. The parameters are supposed to be the best tuned values, which is done by computing the rmse and rsq for a lot of different model setups. Thereafter it is possible to use the best possible parameters for the tuning.

```
set.seed(12)
tune_rf <-
  tune_grid(
    workflow_rf,
    resamples = data_resample,
    grid = 10
  )
```

```
## i Creating pre-processing data to finalize unknown parameter: mtry
```

```
tune_rf %>% autoplot()
```



The best parameters are selected based on rmse and rsq, both indicate the same parameters.

```
best_param_rf <- tune_rf %>% select_best(metric = 'rmse')
best_param_rf
```

```
## # A tibble: 1 x 3
##    mtry min_n .config
##   <int> <int> <chr>
## 1     7    13 Preprocessor1_Model10
```

```
tune_rf %>% show_best(metric = 'rsq', n = 1)
```

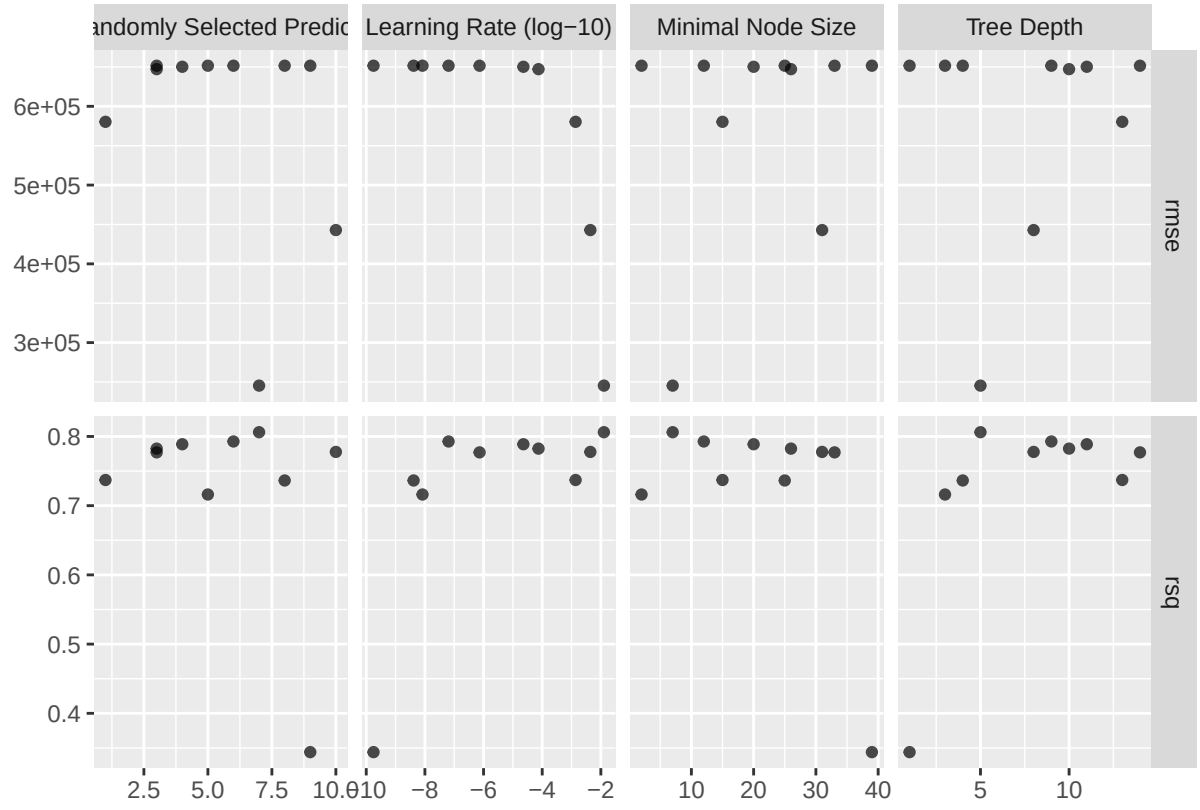
```
## # A tibble: 1 x 8
##    mtry min_n .metric .estimator  mean     n std_err .config
##   <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     7    13 rsq     standard   0.824    15 0.00352 Preprocessor1_Model10
```

The same tuning is done here for the xgboost model.

```
set.seed(12)
tune_xg <-
  tune_grid(
    workflow_xg,
    resamples = data_resample,
    grid = 10
  )
```

```
## i Creating pre-processing data to finalize unknown parameter: mtry
```

```
tune_xg %>% autoplot()
```




```
best_param_xg <- tune_xg %>% select_best(metric = 'rmse')
best_param_xg
```

```
## # A tibble: 1 x 5
##    mtry min_n tree_depth learn_rate .config
##   <int> <int>      <int>      <dbl> <chr>
## 1     7     7          5     0.0127 Preprocessor1_Model01
```

```
tune_xg %>% show_best(metric = 'rsq', n = 1)
```

```
## # A tibble: 1 x 10
##    mtry min_n tree_depth learn_rate .metric .estimator  mean     n std_err
##   <int> <int>      <int>      <dbl> <chr>   <chr>      <dbl> <int>   <dbl>
## 1     7     7          5     0.0127 rsq     standard   0.806    15 0.00300
## # ... with 1 more variable: .config <chr>
```

Now that the best parameters are collected for the models, we can create the final workflows.

```
workflow_final_rf <- workflow_rf %>%
  finalize_workflow(parameters = best_param_rf)

workflow_final_xg <- workflow_xg %>%
  finalize_workflow(parameters = best_param_xg)
```

Next the models are fitted based on the training data, recall that this is contains 75% of the total dataset.

```
fit_lm <- workflow_lm %>%
  fit(data_train)

fit_rf <- workflow_final_rf %>%
  fit(data_train)

fit_xg <- workflow_final_xg %>%
  fit(data_train)
```

Finally we can compute the predictions and compare them to the true values.

```
set.seed(12)
pred_collected <- tibble(
  truth = data_train %>% pull(y),
  base = mean(truth),
  lm = fit_lm %>% predict(new_data = data_train) %>% pull(.pred),
  rf = fit_rf %>% predict(new_data = data_train) %>% pull(.pred),
  xg = fit_xg %>% predict(new_data = data_train) %>% pull(.pred),
  ) %>%
  pivot_longer(cols = -truth,
               names_to = 'model',
               values_to = '.pred')

pred_collected %>% head(8)
```

```
## # A tibble: 8 x 3
##    truth model   .pred
##    <dbl> <chr>   <dbl>
## 1 221900 base  540384.
## 2 221900 lm    189371.
## 3 221900 rf    234559.
## 4 221900 xg    165601.
## 5 257500 base  540384.
## 6 257500 lm    232562.
## 7 257500 rf    261287.
## 8 257500 xg    194582.
```

This table shows the true value of a couple of observations and the predicted value from each of the 4 models. These values suggest that the rf model is the most accurate. This can be further investigated by the use of rmse.

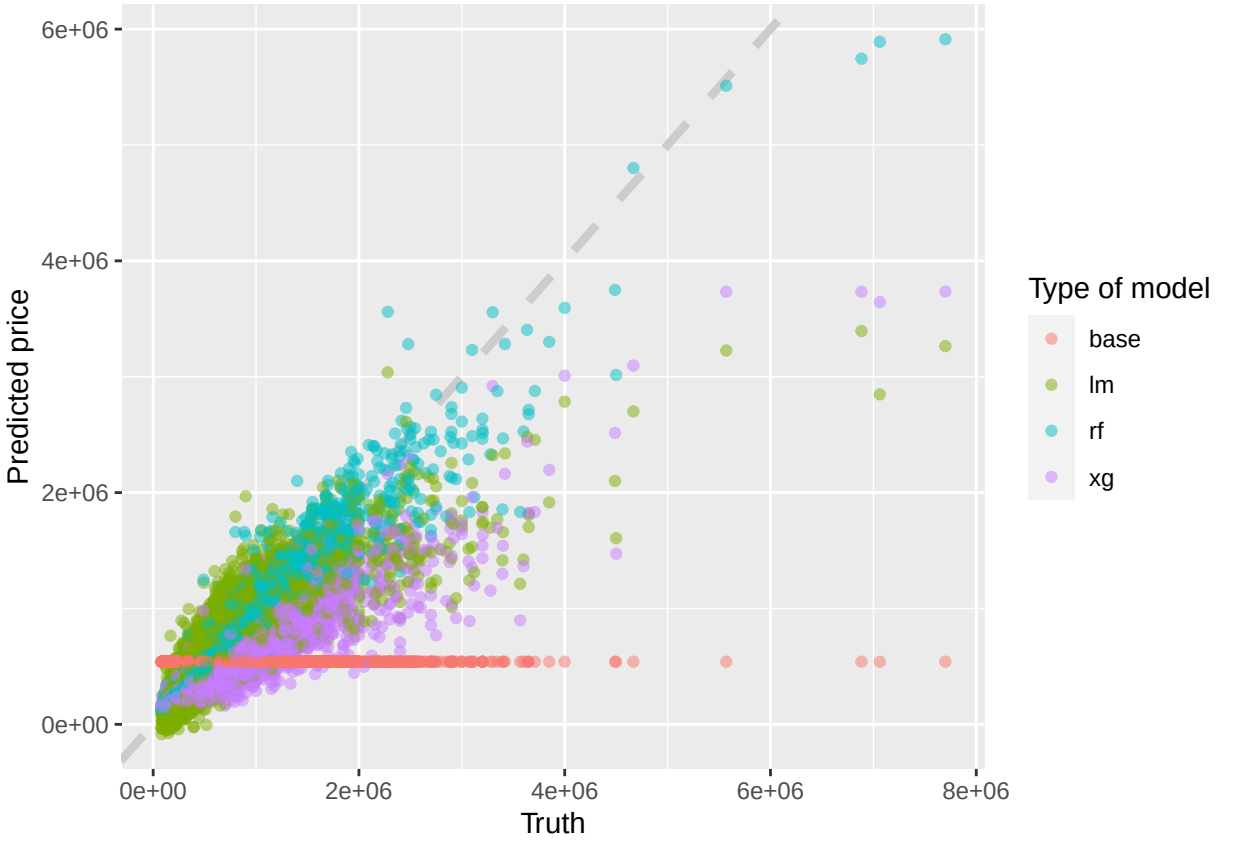
Next we can group the data by model and calculate the rmse for each model based on the training dataset. It can be observed that the random forest model has the lowest rmse value of 99.287.

```
pred_collected %>%
  group_by(model) %>%
  rmse(truth = truth, estimate = .pred) %>%
  select(model, .estimate) %>%
  arrange(.estimate)
```

```
## # A tibble: 4 x 2
##   model .estimate
##   <chr>     <dbl>
## 1 rf       92167.
## 2 lm      184956.
## 3 xg      238154.
## 4 base    364197.
```

This is a visualization of the models predicted value against the true values. The dashed grey line indicates the true values and thereby perfect prediction. Since the random forest model had to lowest rmse, it should also be the closest to the dashed line. The larger the rmse the more spread are the predicted values compared to the true values.

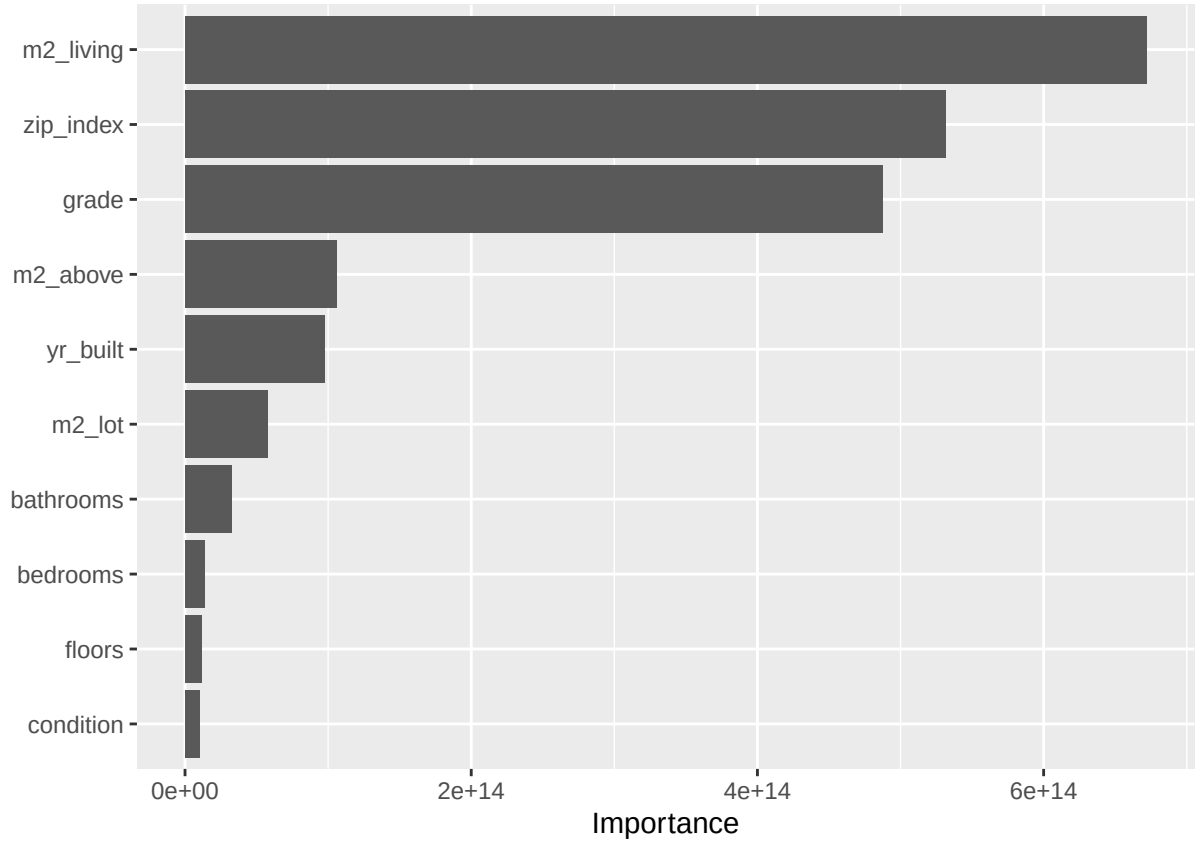
```
pred_collected %>%
  ggplot(aes(x = truth, y = .pred, color = model)) +
  geom_abline(lty = 2, color = "gray80", size = 1.5) +
  geom_point(alpha = 0.5) +
  labs(
    x = "Truth",
    y = "Predicted price",
    color = "Type of model"
  )
```



The random forest model is based on the variables in the dataset, the plot below shows the significance of each variable, based on the prediction. It shows that the first 3 variables are by far the most important in determining the price of an estate in King County. Surpringly the waterfront variable is not shown in the figure, this can be explained by the nature of the variable, since its a dummy variable that takes the value 1 in very few observations. If it were higher in importance, houses without the waterfront tag would be punished harder in price.

```
fit_last_rf <- workflow_final_rf %>% last_fit(split = data_split)

fit_last_rf %>%
  pluck(".workflow", 1) %>%
  pull_workflow_fit() %>%
  vip::vip(num_features = 10)
```

```
## Warning: `pull_workflow_fit()` was deprecated in workflows 0.2.3.
## Please use `extract_fit_parsnip()` instead.
```



Lastly we perform the predictions on the test data with the random forest model. The rmse and rsq is given in the table. The rmse increased on the test dataset, which is to be expected. The models predictions are expected to predict house prices fairly precise. This is based on the fact that the rmse is low compared to the standard deviation of the price.

```
fit_last_rf %>% collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric .estimator  .estimate .config
##   <chr>   <chr>           <dbl> <chr>
## 1 rmse    standard   164152.    Preprocessor1_Model1
## 2 rsq     standard        0.813 Preprocessor1_Model1
```

The same conclusion can be visualized in the last two plots. The first is based on the training data and thereby the lower rmse. The second plot shows the test data prediction.

```
df = pred_collected %>%
        filter(model =="rf")

p1 = df %>%
  ggplot(aes(x = truth, y = .pred)) +
  geom_abline(lty = 2, color = "gray80", size = 1.5) +
  geom_point(alpha = 0.3, color = "blue") +
  labs(
    x = "Truth",
    y = "Predicted price",
    title = "Random Forest \nTraining Data Prediction")
```

```
fit_last_rf_test = fit_last_rf %>% unnest(.predictions) %>% pull(.pred)

fit_last_rf_test = as_tibble(fit_last_rf_test) %>% rename("pred" = "value")

fit_last_rf_test$y = fit_last_rf %>% unnest(.predictions) %>% pull(y)

p2 = fit_last_rf_test %>%
  ggplot(aes(x = y, y = pred)) +
  geom_abline(lty = 2, color = "gray80", size = 1) +
  geom_point(alpha = 0.3, color="blue") +
  labs(
    x = "Truth",
    y = "Predicted price",
    title = "Random Forest \nTest Data Prediction")
```

```
multiplot(p1,p2, cols=2)
```