Kernel and Ensemble Methods

Saugat Gyawali/Bishal Neupane

Source of data set:

Source of data set is here

Reading csv file from Kaggle dataset

```
data <- read.csv("kc_house_data.csv")
dim(data)
```

```
## [1] 21613    21
```

Dividing the data into train, test and validate data.

We divide the data in 60:20:20 ratio meaning, 60 percentage is for training and 20% of data is for testing purpose and 20 for validation

```
set.seed(1234)
spec <- c(train=0.6, test=0.2, validate=0.2)
i <- sample(cut(1:nrow(data),
                nrow(data)*cumsum(c(0,spec)), labels=names(spec)))
train <- data[i=="train",]
test <- data[i=="test",]
vald <- data[i=="validate",]
```

Some of the data exploration using the training data

```
names(train)
```

```
##  [1] "id"            "date"          "price"         "bedrooms"
##  [5] "bathrooms"     "sqft_living"   "sqft_lot"      "floors"
##  [9] "waterfront"    "view"          "condition"     "grade"
## [13] "sqft_above"    "sqft_basement" "yr_built"      "yr_renovated"
## [17] "zipcode"       "lat"           "long"          "sqft_living15"
## [21] "sqft_lot15"
```

```
dim(train)
```

```
## [1] 12967    21
```

```r
summary(train)
```

```
##        id                date               price            bedroms
##  Min.   :1.000e+06   Length:12967       Min.   :  75000   Min.   : 0.000
##  1st Qu.:2.131e+09   Class :character   1st Qu.: 324850   1st Qu.: 3.000
##  Median :3.905e+09   Mode  :character   Median : 450000   Median : 3.000
##  Mean   :4.588e+09                      Mean   : 542890   Mean   : 3.372
##  3rd Qu.:7.331e+09                      3rd Qu.: 649000   3rd Qu.: 4.000
##  Max.   :9.900e+09                      Max.   :7062500   Max.   :11.000
##    bathrooms      sqft_living       sqft_lot           floors
##  Min.   :0.000   Min.   :  290   Min.   :    520   Min.   :1.000
##  1st Qu.:1.750   1st Qu.: 1420   1st Qu.:   5043   1st Qu.:1.000
##  Median :2.250   Median : 1920   Median :   7650   Median :1.500
##  Mean   :2.122   Mean   : 2088   Mean   :  15334   Mean   :1.493
##  3rd Qu.:2.500   3rd Qu.: 2560   3rd Qu.:  10800   3rd Qu.:2.000
##  Max.   :8.000   Max.   :13540   Max.   :1651359   Max.   :3.500
##    waterfront            view           condition         grade
##  Min.   :0.000000   Min.   :0.0000   Min.   :1.000   Min.   : 1.000
##  1st Qu.:0.000000   1st Qu.:0.0000   1st Qu.:3.000   1st Qu.: 7.000
##  Median :0.000000   Median :0.0000   Median :3.000   Median : 7.000
##  Mean   :0.007635   Mean   :0.2371   Mean   :3.408   Mean   : 7.661
##  3rd Qu.:0.000000   3rd Qu.:0.0000   3rd Qu.:4.000   3rd Qu.: 8.000
##  Max.   :1.000000   Max.   :4.0000   Max.   :5.000   Max.   :13.000
##    sqft_above   sqft_basement       yr_built     yr_renovated
##  Min.   : 290   Min.   :   0.0   Min.   :1900   Min.   :   0.00
##  1st Qu.:1200   1st Qu.:   0.0   1st Qu.:1951   1st Qu.:   0.00
##  Median :1560   Median :   0.0   Median :1975   Median :   0.00
##  Mean   :1795   Mean   : 292.4   Mean   :1971   Mean   :  81.13
##  3rd Qu.:2230   3rd Qu.: 560.0   3rd Qu.:1997   3rd Qu.:   0.00
##  Max.   :9410   Max.   :4130.0   Max.   :2015   Max.   :2015.00
##     zipcode           lat             long         sqft_living15
##  Min.   :98001   Min.   :47.16   Min.   :-122.5   Min.   : 460
##  1st Qu.:98032   1st Qu.:47.47   1st Qu.:-122.3   1st Qu.:1490
##  Median :98065   Median :47.57   Median :-122.2   Median :1840
##  Mean   :98078   Mean   :47.56   Mean   :-122.2   Mean   :1992
##  3rd Qu.:98117   3rd Qu.:47.68   3rd Qu.:-122.1   3rd Qu.:2370
##  Max.   :98199   Max.   :47.78   Max.   :-121.3   Max.   :6210
##    sqft_lot15
##  Min.   :   659
##  1st Qu.:  5100
##  Median :  7660
##  Mean   : 12877
##  3rd Qu.: 10125
##  Max.   :560617
```

```r
str(train)
```

```
## 'data.frame':    12967 obs. of  21 variables:
##  $ id           : num  7.13e+09 6.41e+09 5.63e+09 2.49e+09 1.95e+09 ...
##  $ date         : chr  "20141013T000000" "20141209T000000" "20150225T000000" "20141209T000000" ...
##  $ price        : num  221900 538000 180000 604000 510000 ...
##  $ bedrooms     : int  3 3 2 4 3 4 3 3 3 3 ...
##  $ bathrooms    : num  1 2.25 1 3 2 4.5 1.5 1 2.5 2.5 ...
```

```
##  $ sqft_living  : int  1180 2570 770 1960 1680 5420 1060 1780 1890 3560 ...
##  $ sqft_lot     : int  5650 7242 10000 5000 8080 101930 9711 7470 6560 9796 ...
##  $ floors       : num  1 2 1 1 1 1 1 1 2 1 ...
##  $ waterfront   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ view         : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ condition    : int  3 3 3 5 3 3 3 3 3 3 ...
##  $ grade        : int  7 7 6 7 8 11 7 7 7 8 ...
##  $ sqft_above   : int  1180 2170 770 1050 1680 3890 1060 1050 1890 1860 ...
##  $ sqft_basement: int  0 400 0 910 0 1530 0 730 0 1700 ...
##  $ yr_built     : int  1955 1951 1933 1965 1987 2001 1963 1960 2003 1965 ...
##  $ yr_renovated : int  0 1991 0 0 0 0 0 0 0 0 ...
##  $ zipcode      : int  98178 98125 98028 98136 98074 98053 98198 98146 98038 98007 ...
##  $ lat          : num  47.5 47.7 47.7 47.5 47.6 ...
##  $ long         : num  -122 -122 -122 -122 -122 ...
##  $ sqft_living15: int  1340 1690 2720 1360 1800 4760 1650 1780 2390 2210 ...
##  $ sqft_lot15   : int  5650 7639 8062 5000 7503 101930 9711 8113 7570 8925 ...
```

```
head(train)
```

```
##           id            date   price bedrooms bathrooms sqft_living sqft_lot
## 1 7129300520 20141013T000000  221900        3      1.00        1180     5650
## 2 6414100192 20141209T000000  538000        3      2.25        2570     7242
## 3 5631500400 20150225T000000  180000        2      1.00         770    10000
## 4 2487200875 20141209T000000  604000        4      3.00        1960     5000
## 5 1954400510 20150218T000000  510000        3      2.00        1680     8080
## 6 7237550310 20140512T000000 1225000        4      4.50        5420   101930
##   floors waterfront view condition grade sqft_above sqft_basement yr_built
## 1      1          0    0         3     7       1180             0     1955
## 2      2          0    0         3     7       2170           400     1951
## 3      1          0    0         3     6        770             0     1933
## 4      1          0    0         5     7       1050           910     1965
## 5      1          0    0         3     8       1680             0     1987
## 6      1          0    0         3    11       3890          1530     2001
##   yr_renovated zipcode     lat     long sqft_living15 sqft_lot15
## 1            0   98178 47.5112 -122.257          1340       5650
## 2         1991   98125 47.7210 -122.319          1690       7639
## 3            0   98028 47.7379 -122.233          2720       8062
## 4            0   98136 47.5208 -122.393          1360       5000
## 5            0   98074 47.6168 -122.045          1800       7503
## 6            0   98053 47.6561 -122.005          4760     101930
```

```
tail(train)
```

```
##               id            date   price bedrooms bathrooms sqft_living
## 21605 9834201367 20150126T000000  429000        3      2.00        1490
## 21606 3448900210 20141014T000000  610685        4      2.50        2520
## 21607 7936000429 20150326T000000 1007500        4      3.50        3510
## 21611 1523300141 20140623T000000  402101        2      0.75        1020
## 21612  291310100 20150116T000000  400000        3      2.50        1600
## 21613 1523300157 20141015T000000  325000        2      0.75        1020
##       sqft_lot floors waterfront view condition grade sqft_above sqft_basement
## 21605     1126      3          0    0         3     8       1490             0
## 21606     6023      2          0    0         3     9       2520             0
```

```
## 21607       7200       2           0     0           3      9        2600             910
## 21611       1350       2           0     0           3      7        1020               0
## 21612       2388       2           0     0           3      8        1600               0
## 21613       1076       2           0     0           3      7        1020               0
##         yr_built yr_renovated zipcode     lat     long sqft_living15 sqft_lot15
## 21605       2014            0   98144 47.5699 -122.288          1400       1230
## 21606       2014            0   98056 47.5137 -122.167          2520       6023
## 21607       2009            0   98136 47.5537 -122.398          2050       6200
## 21611       2009            0   98144 47.5944 -122.299          1020       2007
## 21612       2004            0   98027 47.5345 -122.069          1410       1287
## 21613       2008            0   98144 47.5941 -122.299          1020       1357
```

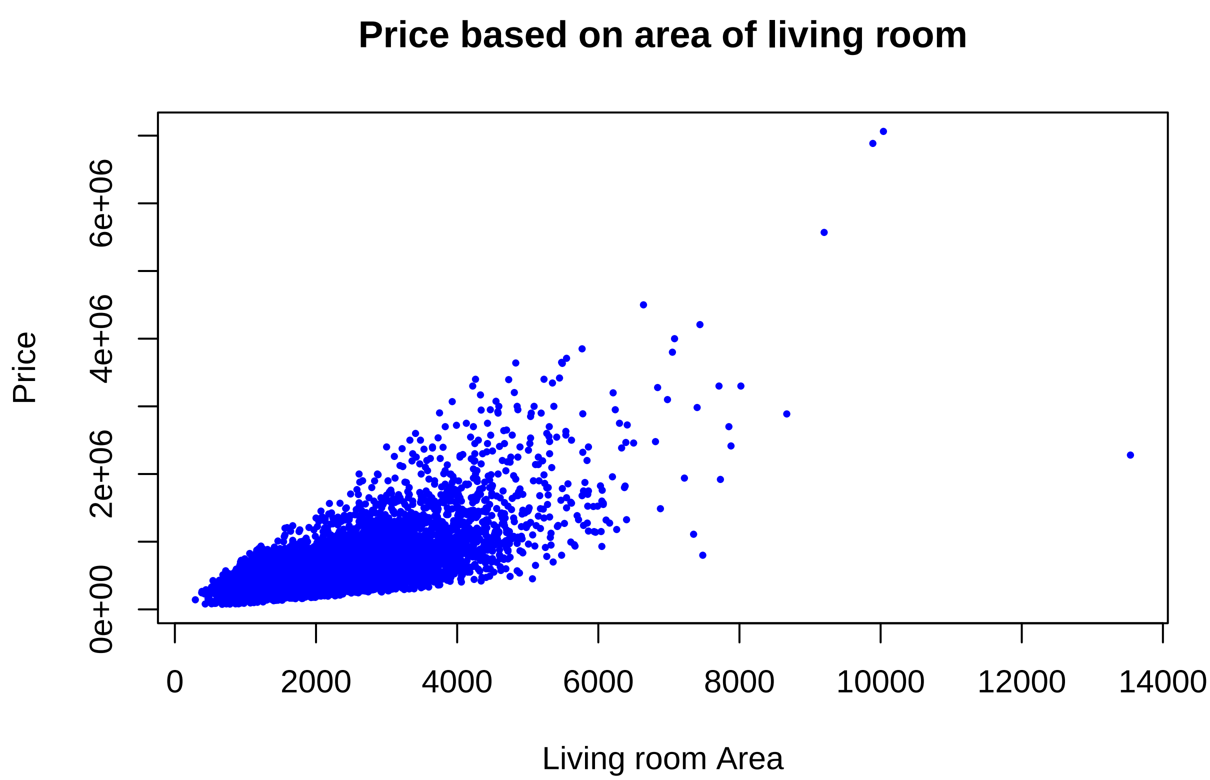
```
sum(is.na(train))
```

```
## [1] 0
```

Some informative graphs

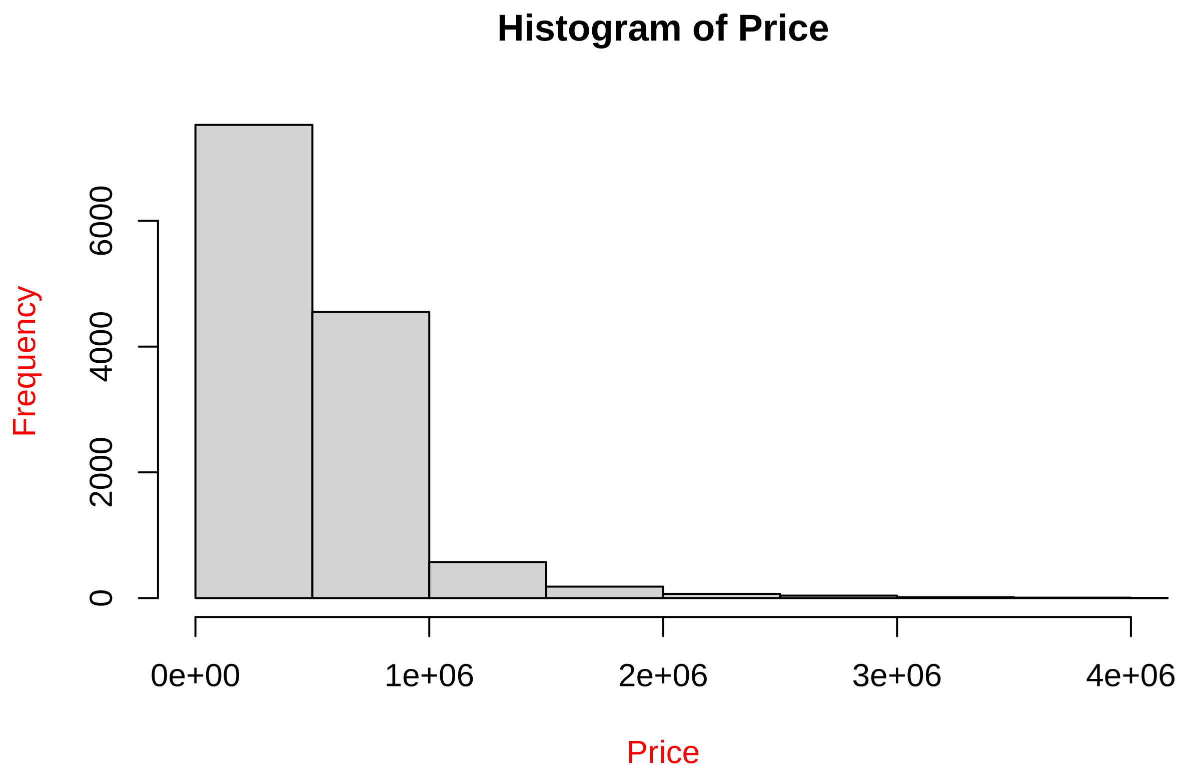
Price vs Area of living room

```
plot(train$sqft_living, train$price, pch = 16, col="blue", cex=0.5,
     main="Price based on area of living room", xlab="Living room Area", ylab="Price")
```



Histogram of Price

```r
Price <- train$price
hist(Price, col.lab="red", xlim=c(0e+00, 4e+06))
```



Comparison of correlation between different parameters

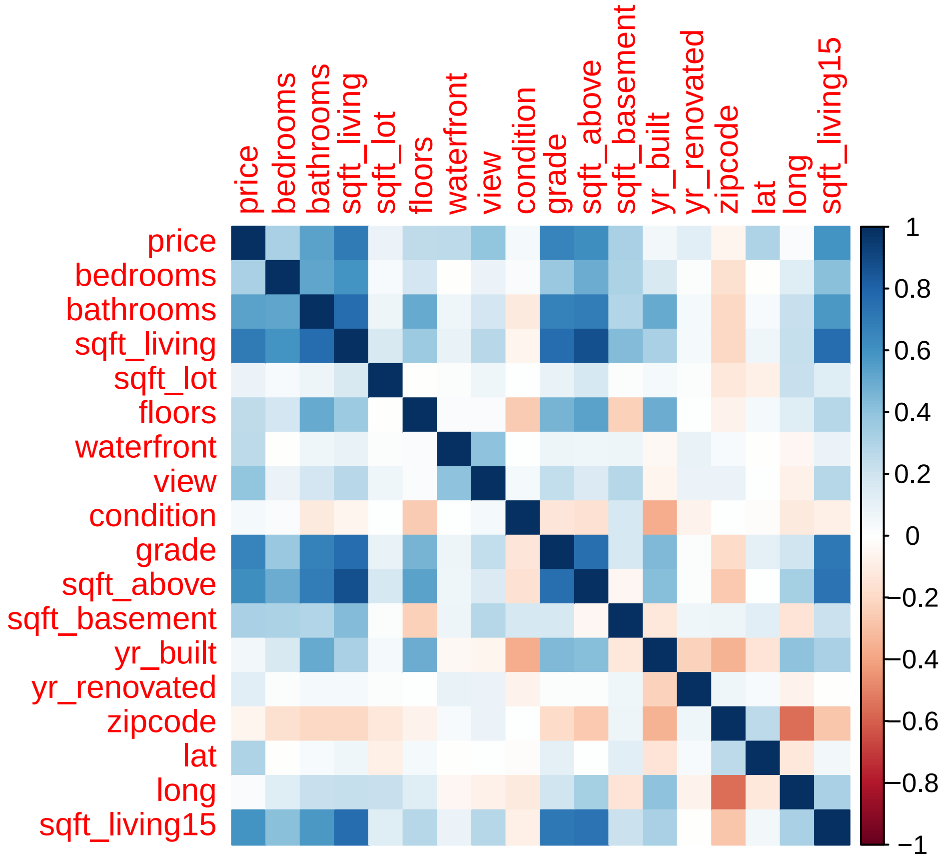
```r
#install.packages("corrplot")
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```r
trainData <- train[, 3:20]

M <- cor(trainData)

corrplot(M, method="color")
```

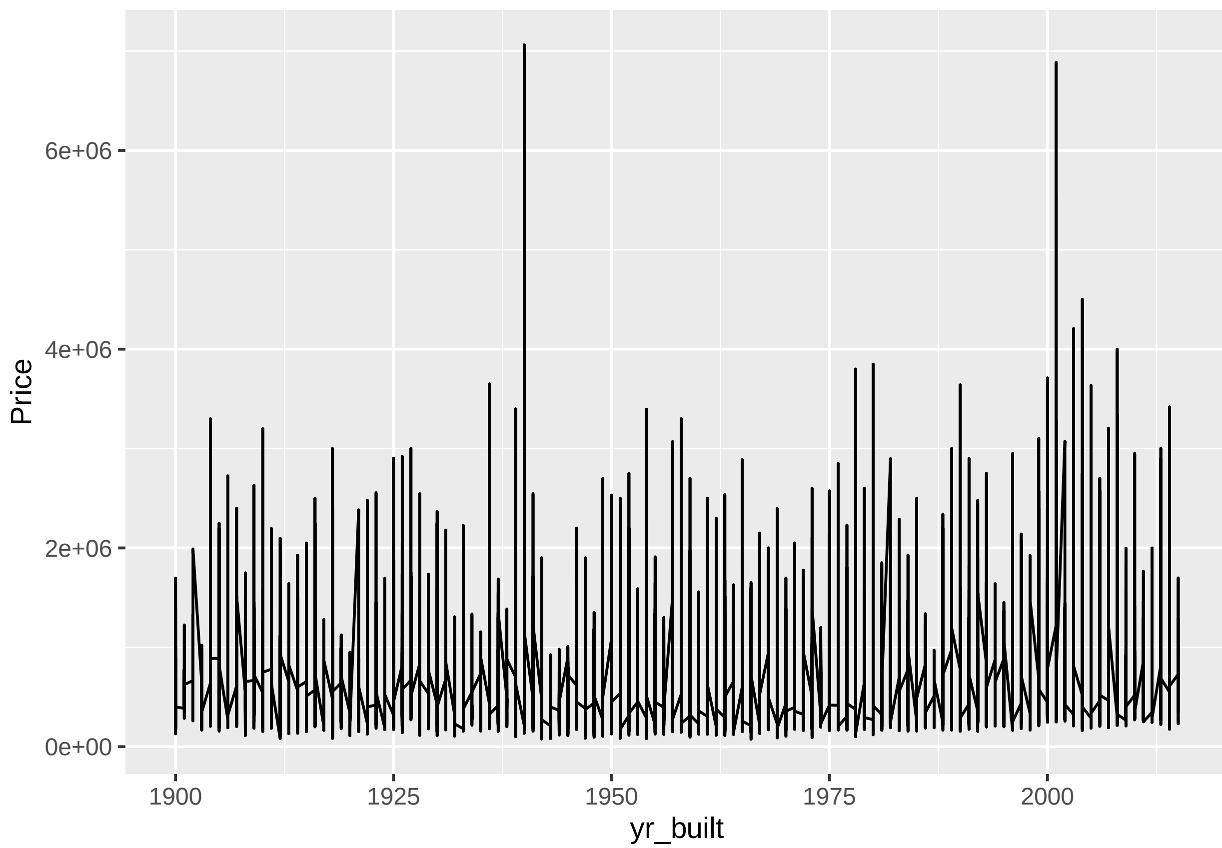


Finding trend of price based on year built

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.3.6      v purrr   0.3.4
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.4.1
## v readr   2.1.3      v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
ggplot(data=train, mapping=aes(x=yr_built,y=Price)) + geom_line()
```



Performing SVM Regression using linear kernel

First we will put random cost C=10.

```
library(e1071)
svm_fit <- svm(price~sqft_living + sqft_above + grade + bathrooms, data=train, kernel="linear", cost=10
summary(svm_fit)
```

```
##
## Call:
## svm(formula = price ~ sqft_living + sqft_above + grade + bathrooms,
##     data = train, kernel = "linear", cost = 10, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  linear
##        cost:  10
##       gamma:  0.25
##     epsilon:  0.1
##
##
## Number of Support Vectors:  12967
```

```r
svm_pred1 <- predict(svm_fit, newdata=test)
cor1 <- cor(svm_pred1, test$price)
mse_svm1 <- mean((svm_pred1-test$price)^2)
rmse_svm1 <- sqrt(mse_svm1)
print(paste('Cor: ', cor1))
```

```
## [1] "Cor:  0.713377395210028"
```

```r
print(paste('mse: ', mse_svm1))
```

```
## [1] "mse:  77893900396.1234"
```

```r
print(paste('rmse: ', rmse_svm1))
```

```
## [1] "rmse:  279094.787475731"
```

Tuning parameters

```r
tune_svm1 <- tune(svm, price~sqft_living + sqft_above + grade + bathrooms, data=vald, kernel="linear", r
summary(tune_svm1)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   100
##
## - best performance: 56725175879
##
## - Detailed performance results:
##    cost       error  dispersion
## 1 1e-03 61274879334 19617624331
## 2 1e-02 57495331448 17940559940
## 3 1e-01 56833972503 17460773779
## 4 1e+00 56768412478 17410462608
## 5 5e+00 56747170465 17388044965
## 6 1e+01 56746614796 17391081726
## 7 1e+02 56725175879 17350737422
```

Evaluating on best linear svm

```r
pred <- predict(tune_svm1$best.model, newdata=test)
cor_svm1_tune <- cor(pred, test$price)
mse_svm1_tune <- mean((pred-test$price)^2)
rmse_svm1_tune <- sqrt(mse_svm1_tune)
print(paste('Cor: ', cor_svm1_tune))
```

```
## [1] "Cor:  0.731165668266165"
```

```r
print(paste('mse: ', mse_svm1_tune))
```

```
## [1] "mse:  74470771542.7002"
```

```r
print(paste('rmse: ', rmse_svm1_tune))
```

```
## [1] "rmse:  272893.333635507"
```

There was a slight increase in cor(pred, test$price).

Performing SVM regression using polynomial kernel

```r
library(e1071)
svm_fit2 <- svm(price~sqft_living + sqft_above + grade + bathrooms, data=train, kernel="polynomial", co
summary(svm_fit2)
```

```
##
## Call:
## svm(formula = price ~ sqft_living + sqft_above + grade + bathrooms,
##     data = train, kernel = "polynomial", cost = 10, degree = 3, scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  polynomial
##        cost:  10
##      degree:  3
##       gamma:  0.25
##      coef.0:  0
##     epsilon:  0.1
##
##
## Number of Support Vectors:  10878
```

```r
svm_pred1 <- predict(svm_fit2, newdata=test)
cor1 <- cor(svm_pred1, test$price)
mse_svm1 <- mean((svm_pred1-test$price)^2)
rmse_svm1 <- sqrt(mse_svm1)
print(paste('Correlation: ', cor1))
```

```
## [1] "Correlation:  0.69944621693413"
```

```r
print(paste('mse: ', mse_svm1))
```

```
## [1] "mse:  75924306234.3812"
```

```r
print(paste('rmse: ', rmse_svm1))
```

```
## [1] "rmse:  275543.655768703"
```

There was slight decrease in correlation and increase in mean square error compared to linear kernel.

Performing SVM Regression, polynomial kernel using C = 1

```r
library(e1071)
svm_fit2 <- svm(price~sqft_living + sqft_above + grade + bathrooms, data=train, kernel="polynomial", co
summary(svm_fit2)
```

```
##
## Call:
## svm(formula = price ~ sqft_living + sqft_above + grade + bathrooms,
##     data = train, kernel = "polynomial", cost = 1, degree = 3, scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  3
##       gamma:  0.25
##      coef.0:  0
##     epsilon:  0.1
##
##
## Number of Support Vectors:  10890
```

```r
svm_pred1 <- predict(svm_fit2, newdata=test)
cor1 <- cor(svm_pred1, test$price)
mse_svm1 <- mean((svm_pred1-test$price)^2)
rmse_svm1 <- sqrt(mse_svm1)
print(paste('Correlation: ', cor1))
```

```
## [1] "Correlation:  0.704066745837198"
```

```r
print(paste('mse: ', mse_svm1))
```

```
## [1] "mse:  74880977205.6095"
```

```r
print(paste('rmse: ', rmse_svm1))
```

```
## [1] "rmse:  273643.887572168"
```

Performing SVM regression using Radial Kernel.

Cost = 1, Gamma = 1

```
svm_fit2 <- svm(price ~ sqft_living + sqft_above + grade + bathrooms, data=train, kernel="radial", cost=
svm_pred2 <- predict(svm_fit2, newdata=test)
cor2 <- cor(svm_pred2, test$price)
mse_svm2 <- mean((svm_pred2-test$price)^2)
rmse_svm2 <- sqrt(mse_svm2)
print(paste('Correlation: ', cor2))
```

```
## [1] "Correlation:  0.718036367057499"
```

```
print(paste('mse: ', mse_svm2))
```

```
## [1] "mse:  72735527557.8683"
```

```
print(paste('rmse: ', rmse_svm2))
```

```
## [1] "rmse:  269695.249416574"
```

There was vast decrease in correlation and increase in mean square error while keeping radial kernel. However, we will try to optimize it by tuning hyperparameters.

Performing using different hyperparameters

Cost = 1 and Gamma = 0.5

```
svm_fit2 <- svm(price ~ sqft_living + sqft_above + grade + bathrooms, data=train, kernel="radial", cost=
svm_pred2 <- predict(svm_fit2, newdata=test)
cor2 <- cor(svm_pred2, test$price)
mse_svm2 <- mean((svm_pred2-test$price)^2)
rmse_svm2 <- sqrt(mse_svm2)
print(paste('Correlation: ', cor2))
```

```
## [1] "Correlation:  0.738897184930017"
```

```
print(paste('mse: ', mse_svm2))
```

```
## [1] "mse:  68775454018.4682"
```

```
print(paste('rmse: ', rmse_svm2))
```

```
## [1] "rmse:  262250.746459316"
```

Tuning hyperparameters for Radial Kernel

Decrasing a validate data because it was taking a lot of time.

```r
set.seed(12)
k <- sample(1:nrow(vald), nrow(vald) * 0.10, replace=FALSE)
tempVald <- vald[k,]
dim(tempVald)
```

```
## [1] 432  21
```

```r
set.seed(1234)
tune.out <- tune(svm, price ~ sqft_living + sqft_above + grade + bathrooms, data=tempVald, kernel="radia
                 ranges=list(cost=c(0.1,1,10,100,1000),
                             gamma=c(0.5,1,2,3,4)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1   0.5
##
## - best performance: 94739193786
##
## - Detailed performance results:
##     cost gamma        error   dispersion
## 1  1e-01   0.5 112276179923 115447527950
## 2  1e+00   0.5  94739193786 110137656828
## 3  1e+01   0.5 104519553456  99120138663
## 4  1e+02   0.5 134971876991 100802723854
## 5  1e+03   0.5 305838274511 195049987917
## 6  1e-01   1.0 120851013212 116747556836
## 7  1e+00   1.0  97625829108 110552668275
## 8  1e+01   1.0 103708750474 100762433028
## 9  1e+02   1.0 140432502621 106778991299
## 10 1e+03   1.0 230105380901 154792935242
## 11 1e-01   2.0 132472613615 119774697926
## 12 1e+00   2.0 109364573713 109708732144
## 13 1e+01   2.0 107511414759 104085366133
## 14 1e+02   2.0 122338343544 108395579442
## 15 1e+03   2.0 185062051169 126901256881
## 16 1e-01   3.0 138014775087 121617277438
## 17 1e+00   3.0 117418791546 110425200314
## 18 1e+01   3.0 112215487291 106944272452
## 19 1e+02   3.0 128313031003 110675477206
## 20 1e+03   3.0 222882403857 162681631602
## 21 1e-01   4.0 141435167914 122870688157
## 22 1e+00   4.0 122386914295 111954880545
## 23 1e+01   4.0 117260287589 109630047513
## 24 1e+02   4.0 140911024743 114236865301
## 25 1e+03   4.0 236600950525 201999549805
```

Evaluating on best cost and gamma

Best cost for radial kernel was found to be 1 and gamma was found to be 0.5.

```
pred <- predict(tune.out$best.model, newdata=test)
cor_svm1_tune <- cor(pred, test$price)
mse_svm1_tune <- mean((pred-test$price)^2)
rmse_svm1_tune <- sqrt(mse_svm1_tune)
print(paste('Cor: ', cor_svm1_tune))
```

```
## [1] "Cor:  0.647508920207381"
```

```
print(paste('mse: ', mse_svm1_tune))
```

```
## [1] "mse:  85349031356.4053"
```

```
print(paste('rmse: ', rmse_svm1_tune))
```

```
## [1] "rmse:  292145.565354679"
```

I got the result of 0.64 correlation coefficient and error of 86349031356 while using radial kernel.

Analysis of kernels

First of all, we need to know what is kernel, it is a function which is used in Support vector machine to solve problems. One of the advantage of kernel is that we can go for large dimensions and produce a smooth result with it. So, how kernel works then? it solves non-linear problems with the help of linear classifiers.

Linear kernel

When the data is linearly seperable or it can be seperated using only linear lines. It is manily used kernels when there are a lots of features in particular data. Advantages of using linear kernel is that it is faster than other kernel, and there is only need of one hyperparameter. In the dataset, I used, linear kernel was found to be the best one because of less error. At first when I used C value equal to 10, the error was a bit more as compared to the best model when the C value was found to be 100 a better one by tuning it. Unlike the linear regression, this tries to fit the best line between the border or boundary line and hyperplance. The results was the most likely achieved because all of the predictors was found to be linearly related because they were strongly correlated. k(x,x') = x^T x'

Polynomial kernel

Our datasets worked good while using the polynomial kernel but it was not as good as a linear kernel. The good thing about the polynomial kernel is that it is does not only looks for the feature, wheras it uses combination of a feature too. It is also known as interaction feature. Polynomial kernels k(x,x') = (1+ x^T x')^d for d > 0 which contains all polynomials terms up to degree d. Support vector machine with a polynomial kernel is used to compute the relationships betwee the observation in a higher dimension.

RBF Kernel

First when we used a random hyperparameter and gamma, the error was found to be massive, but after tuning it the error was less as compared to the first one. The raidal kernel has an additional parameters called gamma which controls the shape of hyperplane boundary. Smaller gammas give sharper peaks in high dimension whereas larger gammas give a peak that are rounded. This means that when we use a high gamma value, the final result is also affected by the points close to the decision boundary. RBF kernel result was bad for this datasets. It might be because the dataset is more linearly seperable and it accounts for the data inside the boundary.