

TP2 MRR

Andrieu Carla et Zaari Abdelouahab

15/10/2021

IV. Real estate data

```
housedata <- read.csv("C:\\\\Users\\\\Lenovo\\\\Desktop\\\\S3\\\\Régression régularisée\\\\Dataset\\\\housedata.csv")
data <- housedata
#removing id and date columns
data$id <- NULL
data$date <- NULL
```

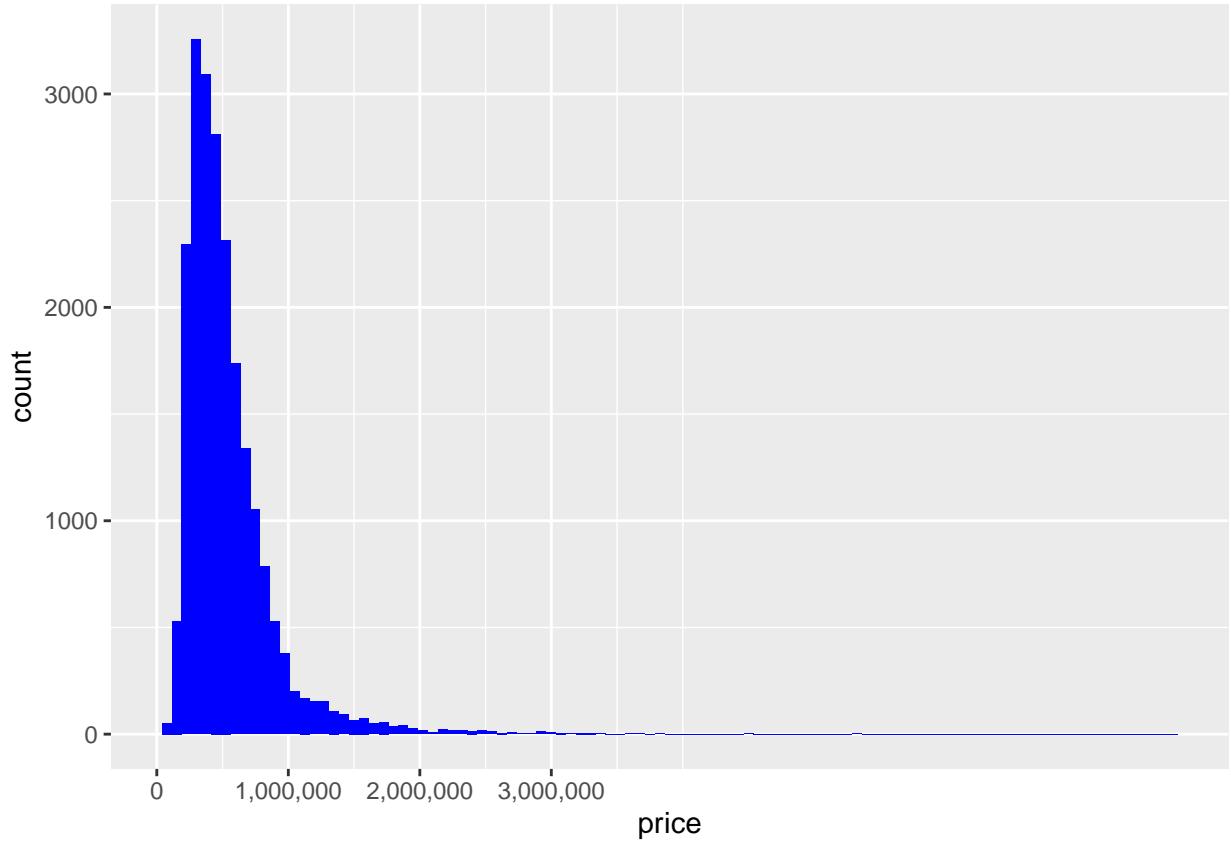
Dans un premier temps, après avoir récupéré les données, nous allons calculer le pourcentage des données manquantes :

```
sum(is.na(data)) / (nrow(data) *ncol(data))
```

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

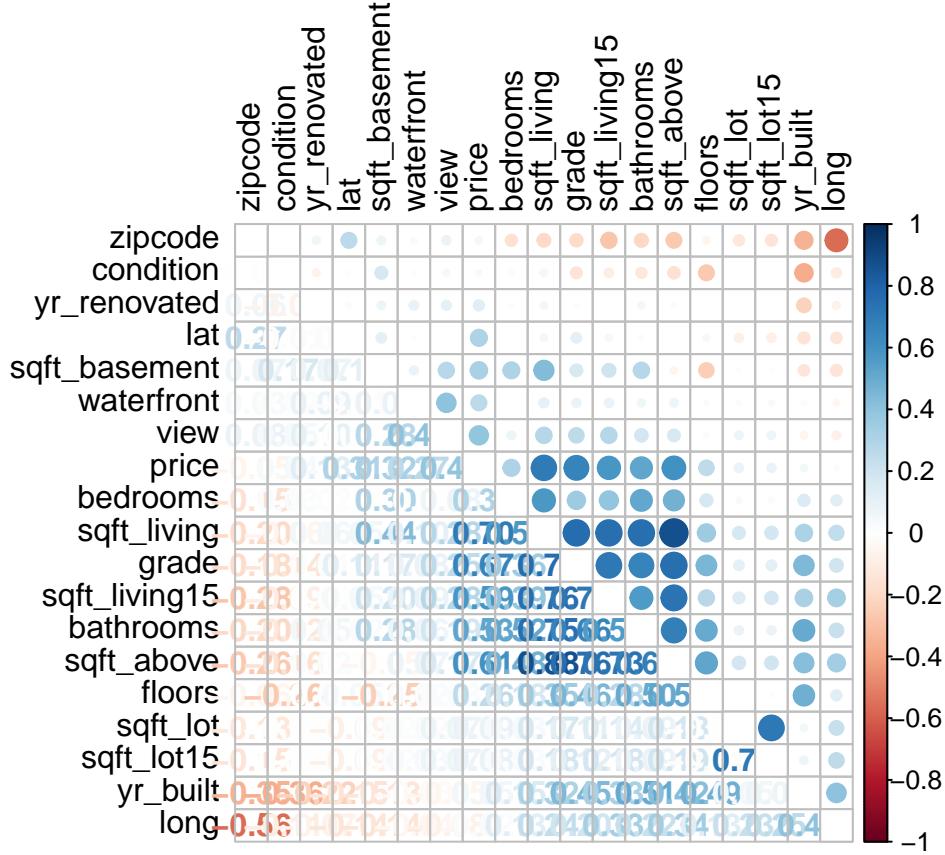
D'abord, on va voir la distribution et la forme de la valeur price que l'on souhaite prédire :

```
ggplot(data=data , aes(x=price)) +
  geom_histogram(fill="blue", binwidth = 75000) +
  scale_x_continuous(breaks= seq(0, 3000000, by=1000000), labels = comma)
```



On peut constater que la distribution des prix de vente des maisons semble suivre une loi normale centrée en 500 k€. Très peu de maisons ont un prix au-delà d'un million. On va chercher à déterminer les variables qui ont une corrélation très grande avec la valeur price :

```
M <- cor(data)
corrplot.mixed(M, tl.col="black",order = 'AOE',tl.pos = "lt")
```



On peut clairement voir qu'il y a une grande corrélation entre la variable *sqft_living* et *price* avec une corrélation de 0.7 ainsi *price* et *grade* d'une valeur d'environ 0.67. On peut aussi constater une très grande multicolinéarité entre *sqft_living* et *sqft_above*, *grade*, *sqft_living15*, *bathrooms*. Celle-ci peut causer une dégradation en terme prédition pour notre modèle. Pour avoir plus d'informations précises sur nos données, nous utilisons la fonction *summary* :

```
str(data)

## 'data.frame': 21613 obs. of 19 variables:
## $ 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 2.25 1.5 1 2.5 ...
## $ sqft_living: int  1180 2570 770 1960 1680 5420 1715 1060 1780 1890 ...
## $ sqft_lot   : int  5650 7242 10000 5000 8080 101930 6819 9711 7470 6560 ...
## $ floors     : num  1 2 1 1 1 1 2 1 1 2 ...
## $ 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 7 ...
## $ sqft_above : int  1180 2170 770 1050 1680 3890 1715 1060 1050 1890 ...
## $ sqft_basement: int  0 400 0 910 0 1530 0 0 730 0 ...
## $ yr_builtin : int  1955 1951 1933 1965 1987 2001 1995 1963 1960 2003 ...
## $ yr_renovated: int  0 1991 0 0 0 0 0 0 0 0 ...
## $ zipcode    : int  98178 98125 98028 98136 98074 98053 98003 98198 98146 98038 ...
## $ 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 2238 1650 1780 2390 ...
```

```

## $ sqft_lot15 : int  5650 7639 8062 5000 7503 101930 6819 9711 8113 7570 ...
summary(data)

##      price          bedrooms        bathrooms       sqft_living
##  Min.   : 75000   Min.   : 0.000   Min.   :0.000   Min.   : 290
##  1st Qu.: 321950  1st Qu.: 3.000   1st Qu.:1.750   1st Qu.: 1427
##  Median : 450000  Median : 3.000   Median :2.250   Median : 1910
##  Mean   : 540088  Mean   : 3.371   Mean   :2.115   Mean   : 2080
##  3rd Qu.: 645000  3rd Qu.: 4.000   3rd Qu.:2.500   3rd Qu.: 2550
##  Max.   :7700000  Max.   :33.000   Max.   :8.000   Max.   :13540
##      sqft_lot         floors      waterfront        view
##  Min.   : 520   Min.   :1.000   Min.   :0.000000   Min.   :0.0000
##  1st Qu.: 5040  1st Qu.:1.000   1st Qu.:0.000000  1st Qu.:0.0000
##  Median : 7618  Median :1.500   Median :0.000000  Median :0.0000
##  Mean   : 15107  Mean   :1.494   Mean   :0.007542  Mean   :0.2343
##  3rd Qu.: 10688  3rd Qu.:2.000   3rd Qu.:0.000000  3rd Qu.:0.0000
##  Max.   :1651359  Max.   :3.500   Max.   :1.000000  Max.   :4.0000
##      condition        grade      sqft_above     sqft_basement
##  Min.   :1.000   Min.   : 1.000   Min.   : 290   Min.   :  0.0
##  1st Qu.:3.000   1st Qu.: 7.000   1st Qu.:1190   1st Qu.:  0.0
##  Median :3.000   Median : 7.000   Median :1560   Median :  0.0
##  Mean   :3.409   Mean   : 7.657   Mean   :1788   Mean   : 291.5
##  3rd Qu.:4.000   3rd Qu.: 8.000   3rd Qu.:2210   3rd Qu.: 560.0
##  Max.   :5.000   Max.   :13.000   Max.   :9410   Max.   :4820.0
##      yr_built      yr_renovated      zipcode        lat
##  Min.   :1900   Min.   : 0.0   Min.   :98001   Min.   :47.16
##  1st Qu.:1951  1st Qu.: 0.0   1st Qu.:98033  1st Qu.:47.47
##  Median :1975  Median : 0.0   Median :98065   Median :47.57
##  Mean   :1971  Mean   : 84.4   Mean   :98078   Mean   :47.56
##  3rd Qu.:1997  3rd Qu.: 0.0   3rd Qu.:98118  3rd Qu.:47.68
##  Max.   :2015  Max.   :2015.0  Max.   :98199  Max.   :47.78
##      long      sqft_living15      sqft_lot15
##  Min.   :-122.5  Min.   : 399  Min.   : 651
##  1st Qu.:-122.3  1st Qu.:1490  1st Qu.: 5100
##  Median :-122.2  Median :1840  Median : 7620
##  Mean   :-122.2  Mean   :1987  Mean   :12768
##  3rd Qu.:-122.1  3rd Qu.:2360  3rd Qu.:10083
##  Max.   :-121.3  Max.   :6210  Max.   :871200

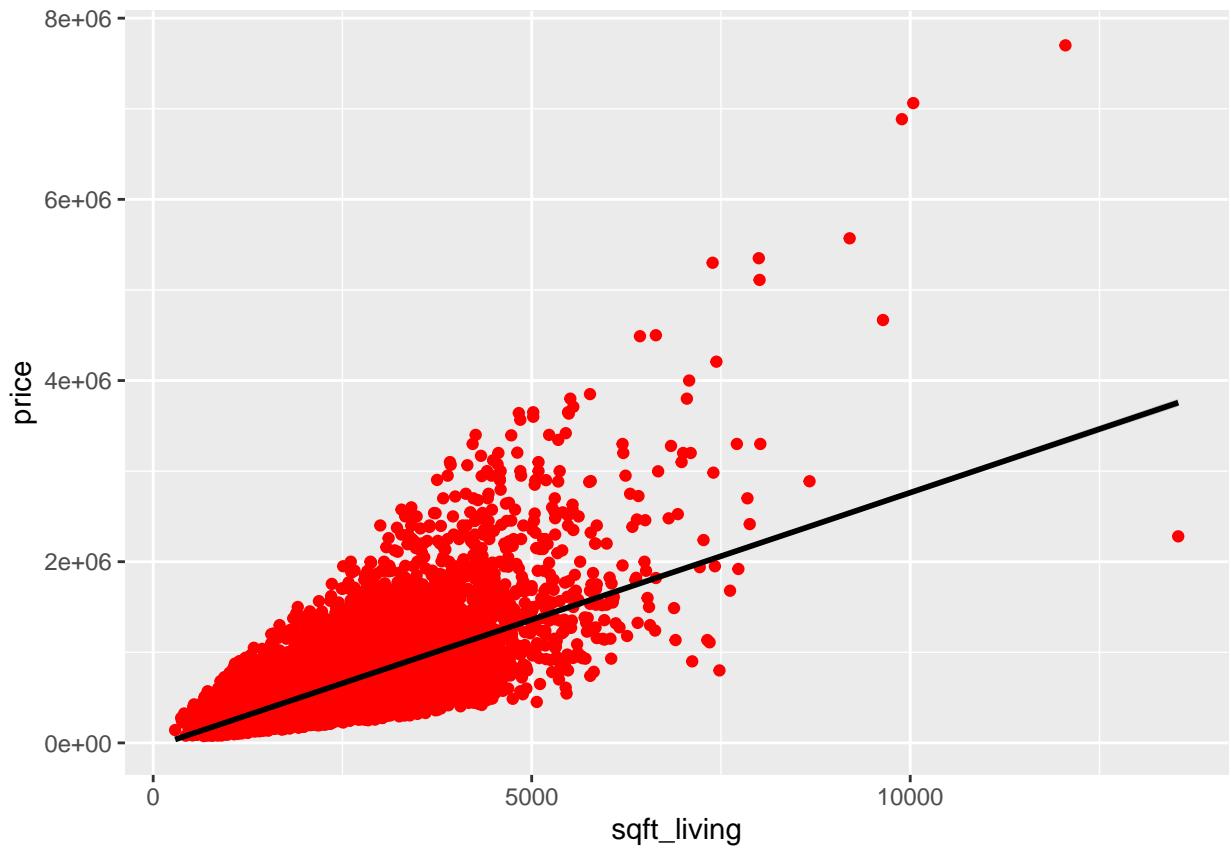
```

On va visualiser la distribution de la variable *sqft_living* qui est la plus variable corrélée avec le prix de la maison :

```

ggplot(data, aes(x =sqft_living , y = price)) +
  geom_point(col="red",pch=19) +
  stat_smooth(formula = y~x ,method = 'lm',col="black")

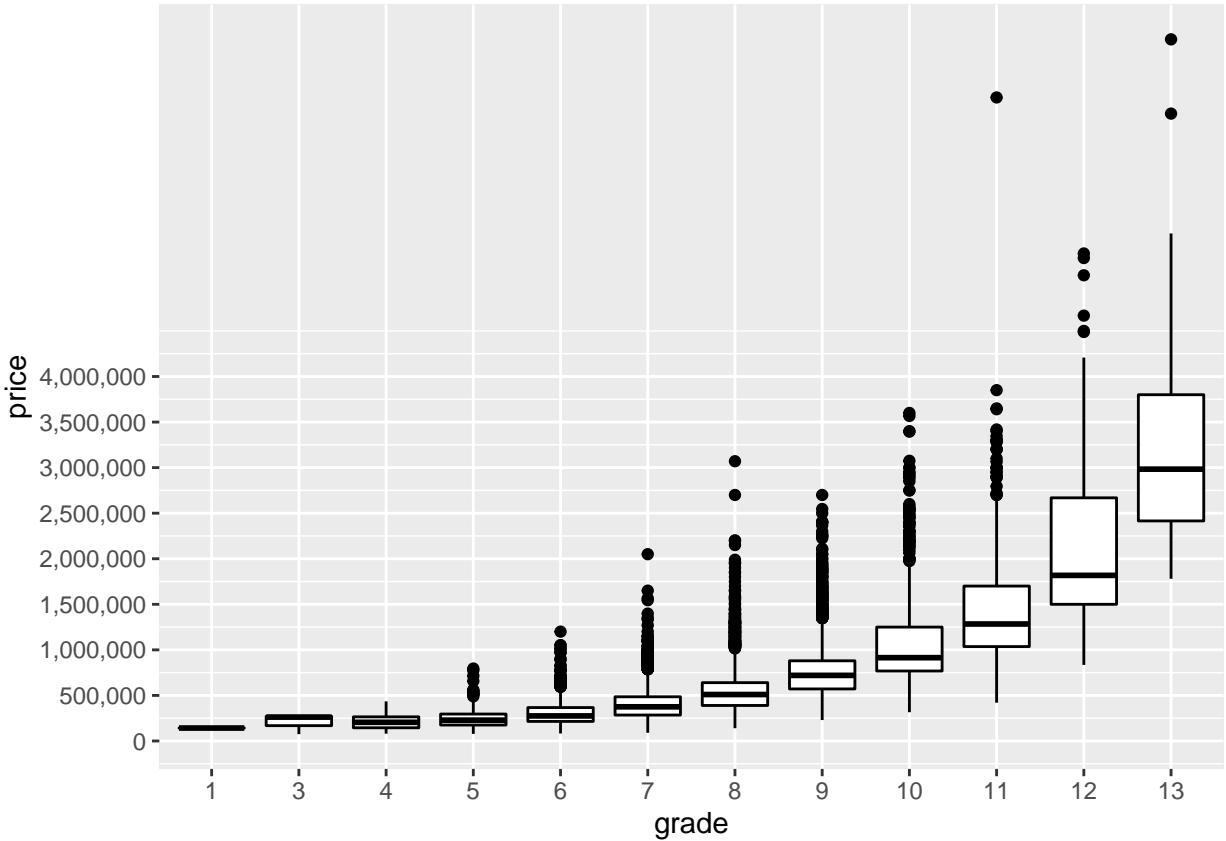
```



On peut clairement constater qu'il ya des valeurs qui sont extrêmes. Généralement, plus la surface de la maison est grande plus le prix augmente ce qui est le cas. Cependant, ici le prix ne paraît pas si élevé en comparaison avec la surface de la maison donc, cette variable risque de biaiser la prédiction du modèle et donner une grande RMSE.

Maintenant, visualisons l'effet de la variable *grade* qui est aussi une variable corrélée avec le prix :

```
ggplot(data=data , aes(x=factor(grade), y=price))+  
  geom_boxplot(col='black') + labs(x='grade') +  
  scale_y_continuous(breaks= seq(0,4000000, by=500000), labels = comma)
```

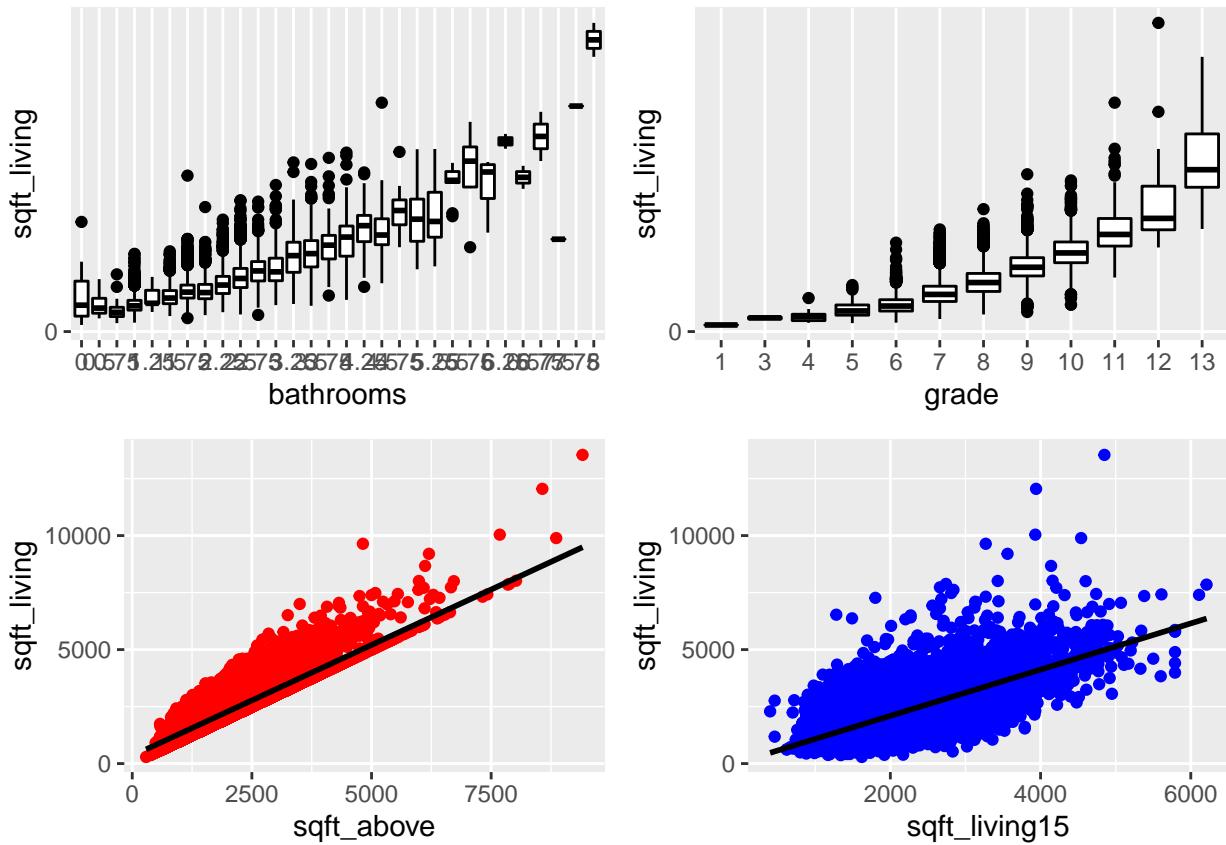


On constate aussi des valeurs anormales pour notre modèle comme les valeurs en 11 et 13 pour la variable *grade*. Celles-ci qui peuvent causer des problèmes de prédiction pour notre modèle final.

Visualisons aussi la multicolinéarité entre *sqft_living* et *sqft_above*, *grade*, *sqft_living15*, *bathrooms*. Voici 4 distributions :

- la distribution *sqft_living* en fonction de la variable *bathrooms* concernant les salles de bains ;
- la distribution *sqft_living* en fonction de *grade* la note de la maison ;
- la distribution *sqft_living* en fonction du *sqft_above* ;
- la distribution *sqft_living* en fonction du *sqft_living15*.

```
p1<-ggplot(data , aes(x=factor(bathrooms), y=sqft_living))+  
  geom_boxplot(col='black') + labs(x='bathrooms') +  
  scale_y_continuous(breaks= seq(0,4000000, by=500000), labels = comma)  
  
p2<-ggplot(data , aes(x=factor(grade), y=sqft_living))+  
  geom_boxplot(col='black') + labs(x='grade') +  
  scale_y_continuous(breaks= seq(0,4000000, by=500000), labels = comma)  
  
p3<-ggplot(data, aes(x = sqft_above , y = sqft_living)) +  
  geom_point(col="red",pch=19) +  
  stat_smooth(formula = y~x ,method = 'lm',col="black")  
  
p4<-ggplot(data, aes(x = sqft_living15 , y = sqft_living)) +  
  geom_point(col="blue",pch=19) +  
  stat_smooth(formula = y~x ,method = 'lm',col="black")  
  
grid.arrange(p1, p2, p3, p4, ncol=2, nrow = 2)
```



On peut clairement constater dans ce cas aussi qu'il ya des valeurs extrêmes. Par exemple, pour la note *grade* de 12 contrairement à la variable *bathrooms* qui reste logique car plus la surface augmente plus le nombre de salle de bain augmente.

En revanche, nous constatons clairement avec la distribution des observations entre *sqft_above* et *sqft_living* une grande colinéarité (redondance) qui peut affecter les résultats de notre modèle. Donc le choix le plus optimal est de supprimer la variable *sqft_above* de notre base de données.

```
# Suppression de la variable sqft_above
data$sqft_above <- NULL
```

On va utiliser maintenant la méthode de **cross validation**. Pour se faire, nous divisons les données de notre base de données en deux ensembles :

- un ensemble d'apprentissage sur lequel s'entraînera notre modèle (70% des données) ;
- un ensemble de test, nommé *test*, avec lequel nous testerons notre modèle (30% restant).

Les données pour les deux data set sont tirées de manière aléatoire.

```
set.seed(2)
# Fractionnement des données en apprentissage et test
data_Split <- sort(sample(nrow(data), nrow(data)*.70))
train <- data[data_Split,]
test <- data[-data_Split,]
```

On peut ainsi créer notre premier modèle et commencer à réaliser des prédictions avec :

```

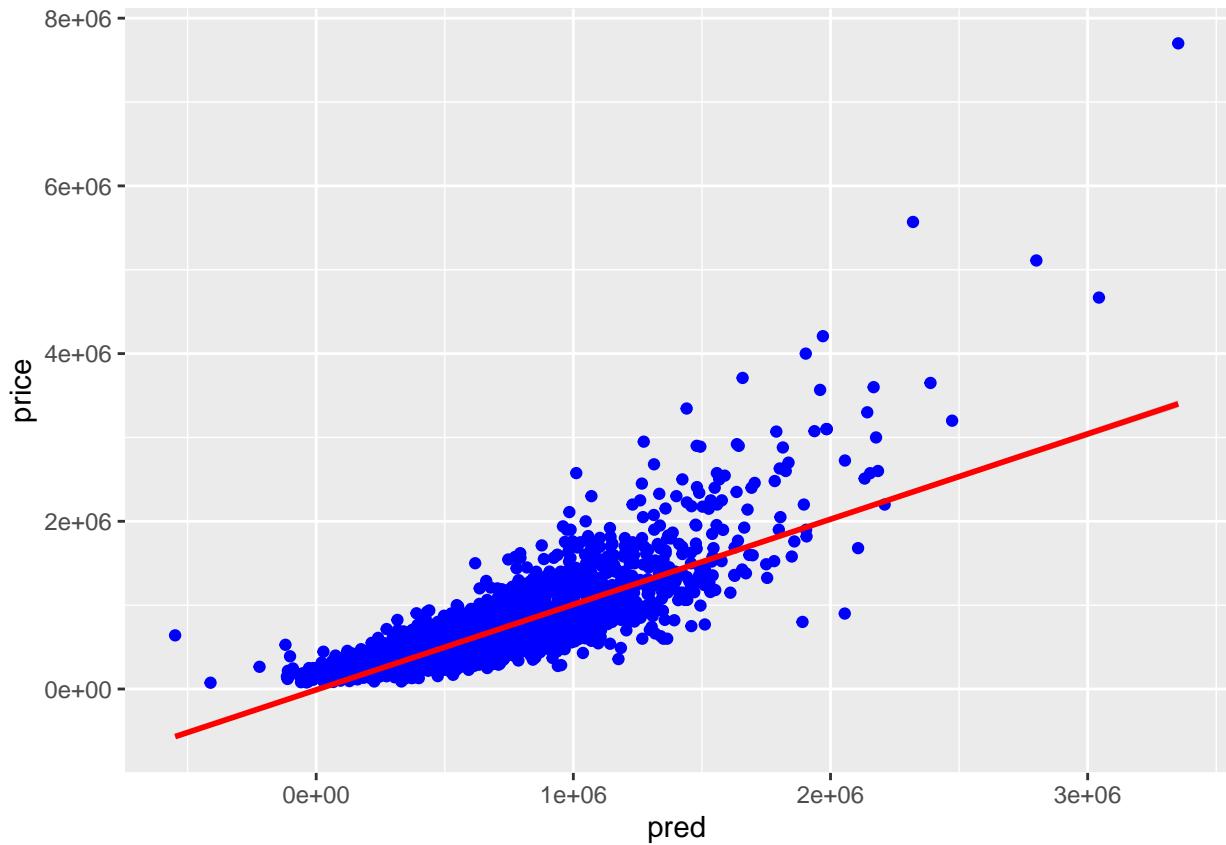
model <- lm(price~.,data=train)
summary(model)

##
## Call:
## lm(formula = price ~ ., data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -1242164   -99633    -9849    77211   4059525 
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 8.387e+06  3.464e+06   2.421   0.0155 *  
## bedrooms    -3.631e+04  2.334e+03 -15.556  < 2e-16 *** 
## bathrooms    4.893e+04  3.860e+03  12.674  < 2e-16 *** 
## sqft_living  1.803e+02  4.356e+00  41.398  < 2e-16 *** 
## sqft_lot     1.336e-01  5.631e-02   2.372   0.0177 *  
## floors       1.724e+03  4.255e+03   0.405   0.6853    
## waterfront   5.265e+05  2.122e+04  24.813  < 2e-16 *** 
## view         5.273e+04  2.498e+03  21.110  < 2e-16 *** 
## condition    2.633e+04  2.778e+03   9.479  < 2e-16 *** 
## grade        9.271e+04  2.554e+03  36.304  < 2e-16 *** 
## sqft_basement -4.391e+01  5.143e+00  -8.537  < 2e-16 *** 
## yr_builtin   -2.688e+03  8.595e+01  -31.273 < 2e-16 *** 
## yr_renovated  1.953e+01  4.330e+00   4.511  6.51e-06 *** 
## zipcode      -6.122e+02  3.894e+01  -15.722 < 2e-16 *** 
## lat          6.181e+05  1.267e+04  48.778  < 2e-16 *** 
## long         -2.199e+05  1.539e+04  -14.295 < 2e-16 *** 
## sqft_living15 2.586e+01  4.089e+00   6.325  2.60e-10 *** 
## sqft_lot15   -3.358e-01  8.542e-02  -3.932  8.47e-05 *** 
## ---      
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 
##
## Residual standard error: 198900 on 15111 degrees of freedom
## Multiple R-squared:  0.6985, Adjusted R-squared:  0.6981 
## F-statistic:  2059 on 17 and 15111 DF,  p-value: < 2.2e-16 

pred <- predict(model,test)

#Ploting
par(mfrow=c(1,1))
ggplot(test, aes(x = pred, y = price)) +
  geom_point(col="blue",pch=19) +
  stat_smooth(formula = y~x ,method = 'lm',col="red")

```



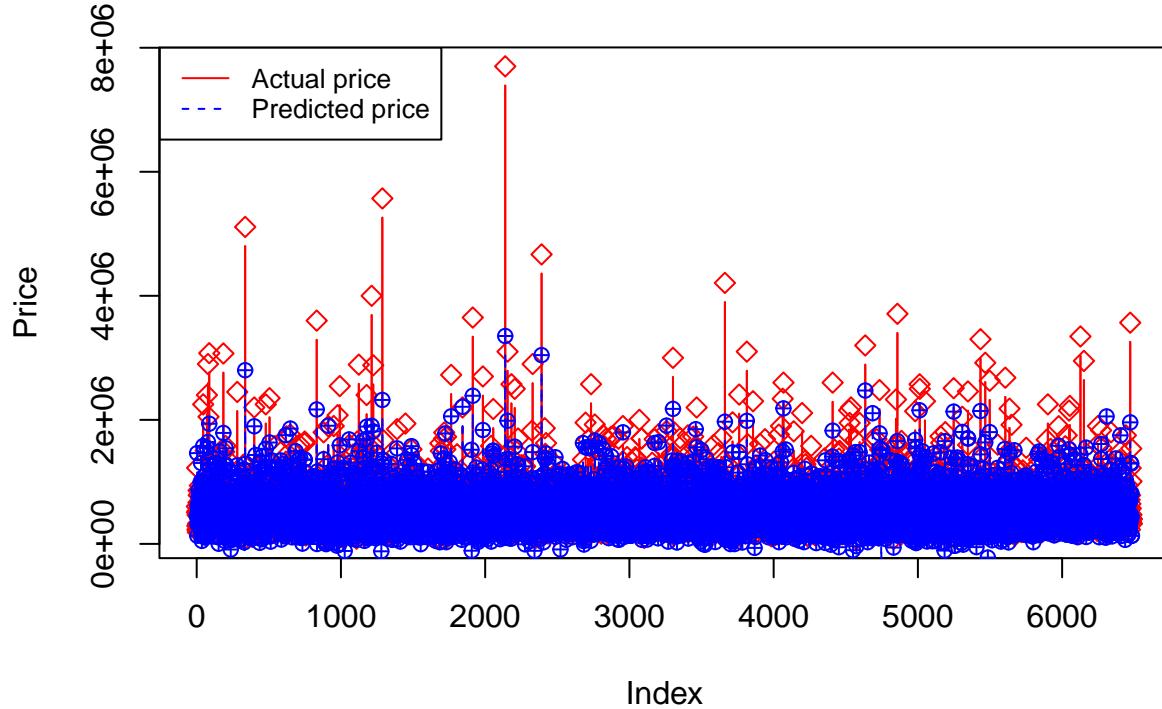
Comparaison entre le prix réel et le prix prédit et affichage de nos résultats :

```
results <- data.frame(actual =test$price,prediction=pred)
head(results)

##      actual prediction
## 4    604000    454804.4
## 5    510000    444259.5
## 6   1225000   1463761.9
## 8    291850    130731.0
## 9    229500    315935.1
## 12   468000    441831.2

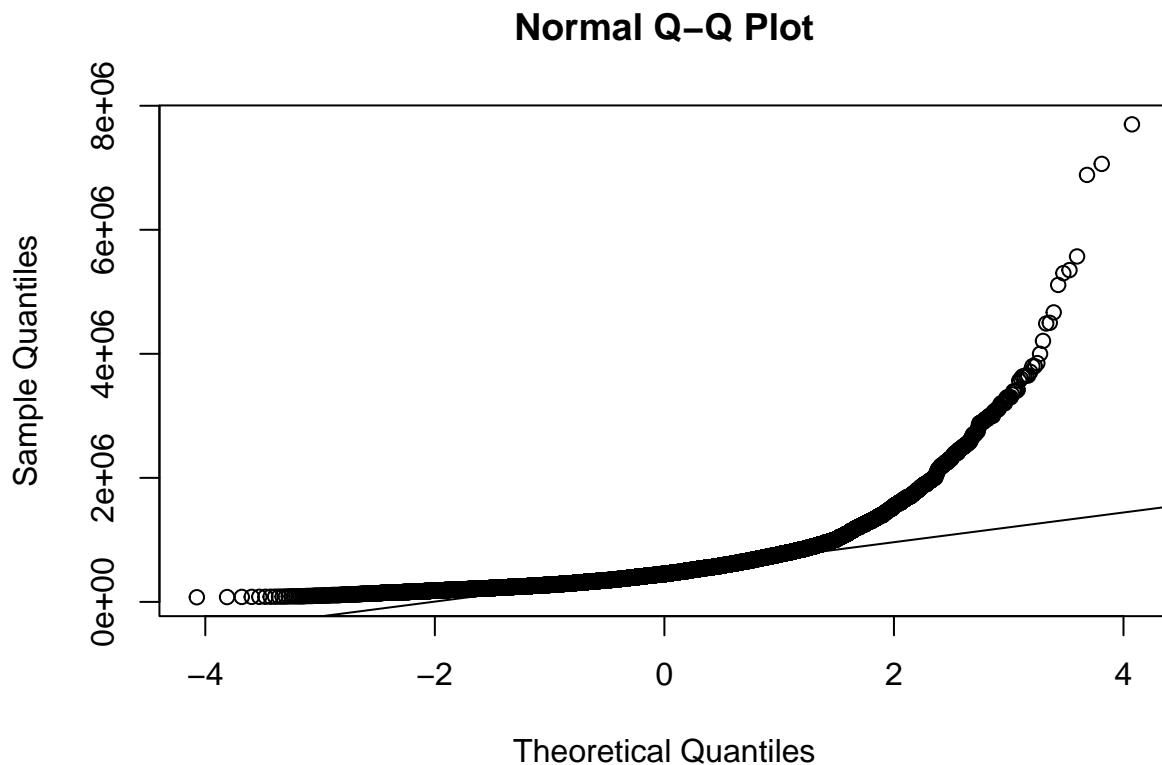
plot(test$price,type="b", pch=5, col="red",ylab="Price",main="Actual vs Predicted price")
# Ajouter une ligne
lines(pred, pch=10, col="blue", type="b", lty=2)
# Ajouter une légende
legend("topleft",legend=c("Actual price", "Predicted price"),
       col=c("red", "blue"), lty=1:2000, cex=0.8)
```

Actual vs Predicted price



Maintenant on affiche le plot Normal Q-Q plot pour avoir une idée sur l'asymétrie de la variable *price* que l'on souhaite prédire :

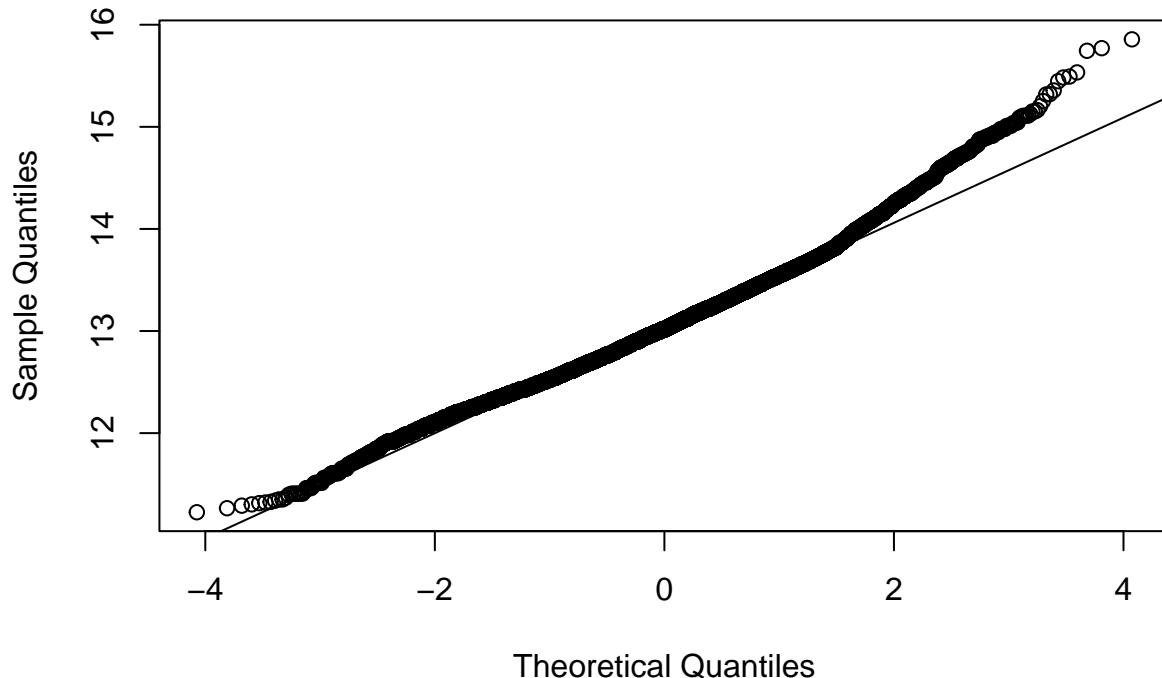
```
qqnorm(data$price)  
qqline(data$price)
```



On peut clairement constater que notre variable n'est pas normalement distribuée. Pour corriger ce problème, et avant de débuter notre modélisation avec Lasso et Ridge régression, on normalise notre base de données pour avoir plus de précision en terme prédition et d'erreur. Pour cela, nous utilisons la fonction *log* puis on re-effectue le test de normalité QQplot :

```
data$price <- log(data$price)
qqnorm(data$price)
qqline(data$price)
```

Normal Q-Q Plot



Désormais, nous pouvons observer que la variable est distribuée de façon normale. On peut donc l'utiliser pour la modélisation car elle vérifie l'hypothèse d'être Gaussienne. Par la suite, on peut aussi supprimer la variable *sqft_basement* car l'information donnée par celle-ci est déjà contenue dans les autres variables et est donc redondante.

Ajustons notre modèle en éliminant la colonne *sqft_basement* avec de nouveaux train dataset et test dataset :

```
data$sqft_basement <- NULL
train <- data[data_Split,]
test <- data[-data_Split,]

# Variables prédictives
X <- model.matrix(price~., data= train)[,-1]
# Variable de résultat (Target)
Y<- train$price
```

Nous allons utiliser la fonction R *glmnet()* pour calculer les modèles de régression linéaire pénalisés et nous allons prendre alpha=1. Premièrement, nous allons travailler avec la méthode de lasso :

```
glmnet(X, Y, alpha = 1 , lambda = NULL)
```

```
##
## Call: glmnet(x = X, y = Y, alpha = 1, lambda = NULL)
##
##      Df  %Dev  Lambda
## 1    0  0.00 0.36800
## 2    2  9.05 0.33530
## 3    2 16.89 0.30550
```

```

## 4 2 23.40 0.27840
## 5 2 28.80 0.25370
## 6 2 33.29 0.23110
## 7 3 38.47 0.21060
## 8 3 43.87 0.19190
## 9 3 48.36 0.17480
## 10 3 52.08 0.15930
## 11 3 55.17 0.14520
## 12 4 57.76 0.13230
## 13 4 59.98 0.12050
## 14 4 61.82 0.10980
## 15 4 63.35 0.10000
## 16 5 64.94 0.09116
## 17 5 66.25 0.08306
## 18 5 67.35 0.07568
## 19 5 68.26 0.06896
## 20 5 69.01 0.06283
## 21 6 69.74 0.05725
## 22 6 70.62 0.05217
## 23 7 71.42 0.04753
## 24 8 72.12 0.04331
## 25 9 72.79 0.03946
## 26 9 73.39 0.03596
## 27 9 73.89 0.03276
## 28 10 74.32 0.02985
## 29 10 74.67 0.02720
## 30 10 74.97 0.02478
## 31 11 75.25 0.02258
## 32 11 75.49 0.02058
## 33 11 75.69 0.01875
## 34 11 75.86 0.01708
## 35 11 76.00 0.01556
## 36 13 76.12 0.01418
## 37 13 76.26 0.01292
## 38 13 76.38 0.01177
## 39 13 76.47 0.01073
## 40 14 76.57 0.00978
## 41 14 76.66 0.00891
## 42 14 76.73 0.00812
## 43 14 76.79 0.00739
## 44 14 76.84 0.00674
## 45 14 76.88 0.00614
## 46 14 76.91 0.00559
## 47 14 76.94 0.00510
## 48 15 76.96 0.00464
## 49 15 76.99 0.00423
## 50 15 77.01 0.00385
## 51 15 77.02 0.00351
## 52 15 77.04 0.00320
## 53 15 77.05 0.00292
## 54 15 77.06 0.00266
## 55 15 77.07 0.00242
## 56 15 77.08 0.00221
## 57 15 77.08 0.00201

```

```
## 58 15 77.09 0.00183
## 59 16 77.09 0.00167
## 60 16 77.09 0.00152
## 61 16 77.10 0.00139
## 62 16 77.10 0.00126
## 63 16 77.10 0.00115
## 64 16 77.11 0.00105
## 65 16 77.11 0.00096
## 66 16 77.11 0.00087
## 67 16 77.11 0.00079
## 68 16 77.11 0.00072
## 69 16 77.11 0.00066
## 70 16 77.11 0.00060
```

Nous spécifions une constante lambda pour ajuster le montant du coefficient de retrait et nous allons utiliser après la fonction `cv.glmnet()` pour identifier la meilleure valeur lambda qui minimise l'erreur quadratique moyenne du modèle.

Nous trouvons le meilleur lambda en utilisant cross-validation :

```
set.seed(123)
model_cross_valid <- cv.glmnet(X, Y, alpha = 1)
```

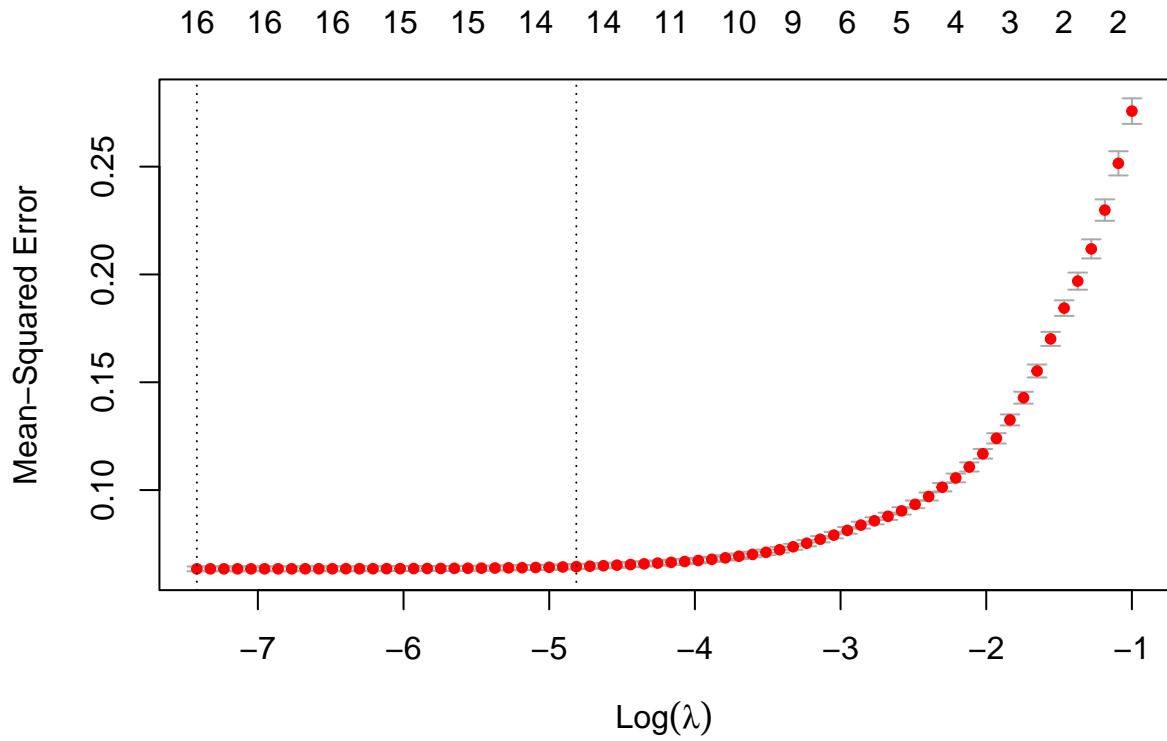
Puis, on affiche la meilleure valeur de lamnbda pour notre modèle :

```
best_lambda <- model_cross_valid$lambda.min
best_lambda
```

```
## [1] 0.0005997742
```

On peut comparer la MSE (Mean Square Error) en fonction de la valeur de lambda :

```
plot(model_cross_valid)
```



Enfin, on ajuste le modèle final avec les données d'entraînement :

```
model <- glmnet(X, Y, alpha = 1, lambda = best_lambda)
# affichons les coefficients de la régression
coef(model)
```

```
## 17 x 1 sparse Matrix of class "dgCMatrix"
##           s0
## (Intercept) -4.477456e+00
## bedrooms    -1.298304e-02
## bathrooms     7.710849e-02
## sqft_living   1.414572e-04
## sqft_lot      4.415349e-07
## floors       6.706635e-02
## waterfront    3.362495e-01
## view         6.047368e-02
## condition    6.203636e-02
## grade        1.545825e-01
## yr_builtin   -3.462372e-03
## yr_renovated  3.958035e-05
## zipcode      -6.604441e-04
## lat          1.412827e+00
## long         -1.619923e-01
## sqft_living15 1.006053e-04
## sqft_lot15    -1.576575e-07
```

La partie apprentissage étant terminée, nous allons tester le modèle sur notre ensemble de données de test :

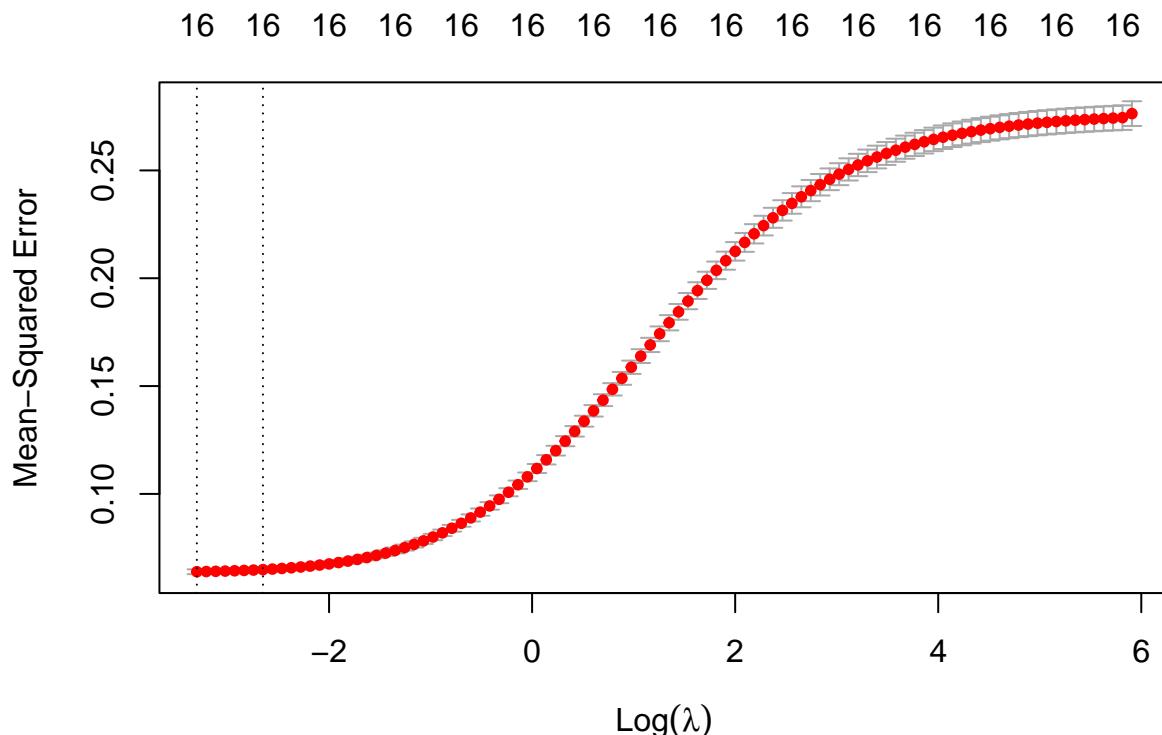
```
X_test <- model.matrix(price ~ ., test) [,-1]
pred_lasso <- predict(model, s=best_lambda, newx= X_test)
```

Deuxièmement, on va tester le modèle sur la régression Ridge afin de comparer les résultats de la régression linéaire et les régressions de Ridge et lasso. Avec la même méthode que précédemment :

- nous trouvons le meilleur lambda en utilisant cross-validation ;
- on affiche le tracé de la MSE pour le test par rapport à la valeur de lambda ;
- on ajuste le modèle final sur les données d'entraînement.

```
set.seed(123)
model_cross_valid <- cv.glmnet(X, Y, alpha = 0)
# on affiche la meilleure valeur de lambda
best_lambda <- model_cross_valid$lambda.min
best_lambda

## [1] 0.03680159
plot(model_cross_valid)
```



```
model <- glmnet(X, Y, alpha = 0, lambda = best_lambda)
# affichons les coefficients de la régression
coef(model)

## 17 x 1 sparse Matrix of class "dgCMatrix"
##           s0
## (Intercept) -1.536591e+01
```

```

## bedrooms      -6.604406e-03
## bathrooms     7.834437e-02
## sqft_living   1.328569e-04
## sqft_lot       4.437857e-07
## floors        6.586719e-02
## waterfront    3.310533e-01
## view          6.156082e-02
## condition     6.227460e-02
## grade          1.395581e-01
## yr_built      -2.993088e-03
## yr_renovated   4.599842e-05
## zipcode        -5.235870e-04
## lat            1.337461e+00
## long           -1.636987e-01
## sqft_living15  1.134618e-04
## sqft_lot15     -1.450333e-07

```

Enfin, on va tester sur notre test dataset et effectuer les prédictions :

```
X_test <- model.matrix(price ~ ., test) [,-1]

pred_Ridge <- predict(model, s=best_lambda, newx= X_test)
```

Résultats

Comparaison des trois modèles sur les prédictions du prix

```

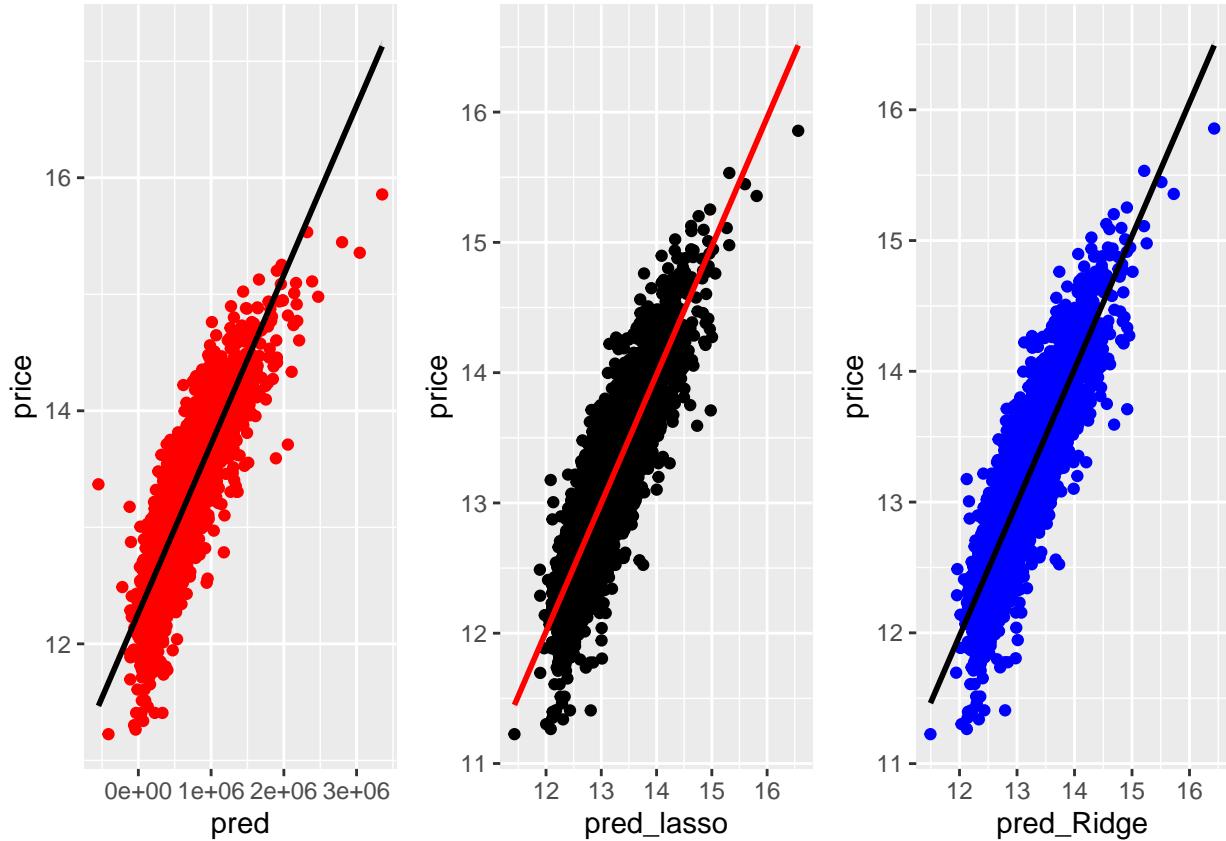
#Ploting pred vs price
r1<- ggplot(test, aes(x = pred, y = price)) +
  geom_point(col="red",pch=19) +
  stat_smooth(formula = y~x ,method = 'lm',col="black")

#Ploting pred_lasso vs price
r2<- ggplot(test, aes(x = pred_lasso, y = price)) +
  geom_point(col="black",pch=19) +
  stat_smooth(formula = y~x ,method = 'lm',col="red")

#Ploting pred_Ridge vs price
r3 <- ggplot(test, aes(x = pred_Ridge, y = price)) +
  geom_point(col="blue",pch=19) +
  stat_smooth(formula = y~x ,method = 'lm',col="black")

grid.arrange(r1, r2, r3, ncol=3, nrow = 1)

```



Comparaisons des erreurs de nos trois modèles

```

RMSE_lasso <- sqrt(mean(pred_lasso-test$price)^2)
RMSE_Ridge <- sqrt(mean(pred_Ridge-test$price)^2)
SSE_lasso <- sum((pred_lasso - test$price)^2)
SSE_Ridge <- sum((pred_Ridge - test$price)^2)
SST <- sum((test$price - mean(test$price))^2)
#RMSE et R squared
results_RMSE <- data.frame(RMSE_RIDGE=RMSE_Ridge,
                             RMSE_LASSO=RMSE_lasso,
                             R_SQUARED_RIDGE=1-(SSE_Ridge/SST) ,
                             R_SQUARED_lasso=1-(SSE_lasso/SST))
results_RMSE

##      RMSE_RIDGE RMSE_LASSO R_SQUARED_RIDGE R_SQUARED_lasso
## 1 0.002076624 0.00220109         0.767082        0.7681027

```

Pour conclure, nous pouvons bien constater que les modèles de régression régularisés Ridge et Lasso fonctionnent mieux en terme de prédiction et d'erreur que le modèle de régression linéaire. De plus, d'après la comparaison entre Lasso et Ridge, on constate que la régression Ridge est meilleure en terme de prédiction et d'erreur quadratique moyenne ($0.00207 < 0.00221$).