TP1 MRR

Abdelouahab Zaari et Carla Andrieu

9/29/2021

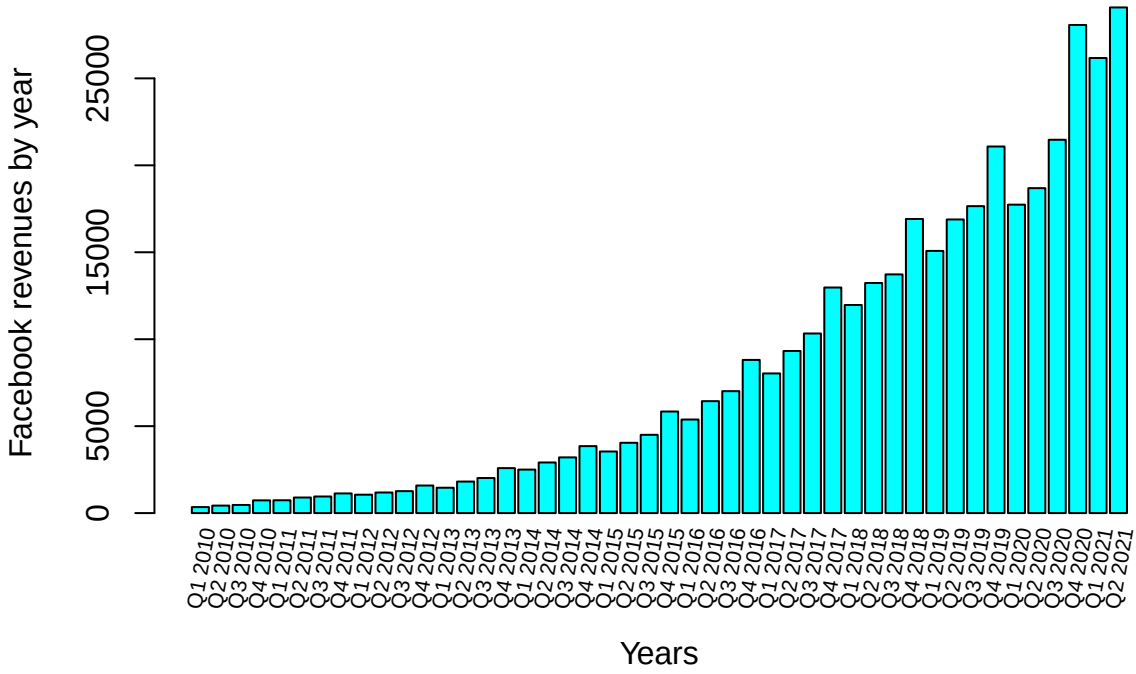
IV. Application: GAFAM or BATX data set

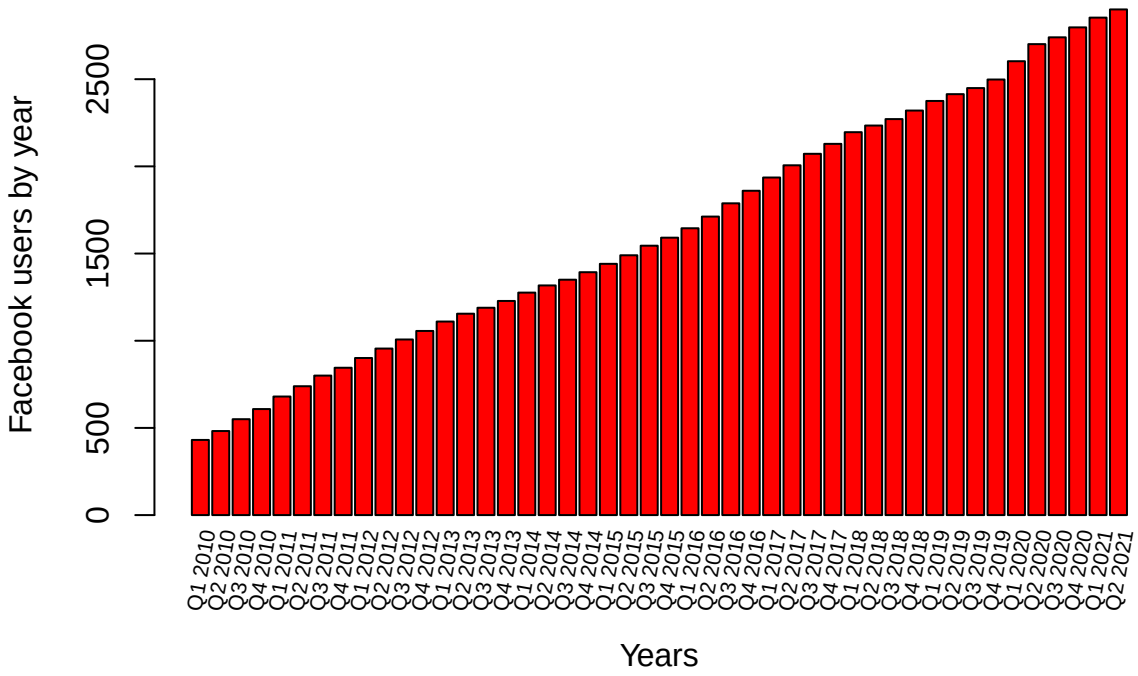
Les données sont composées du revenu annuel de Facebook en million de dollars et du nombre de million d'utilisateurs par an depuis 2010 jusqu'au deuxième trimestre 2021. Nous avons récupéré les données du site : https://www.investopedia.com/terms/q/quarter.asp

```
data <- data.frame(Date=c("Q1 2010","Q2 2010","Q3 2010","Q4 2010"
         ,"Q1 2011","Q2 2011","Q3 2011","Q4 2011"
         ,"Q1 2012","Q2 2012","Q3 2012","Q4 2012"
         ,"Q1 2013","Q2 2013","Q3 2013","Q4 2013"
         ,"Q1 2014","Q2 2014","Q3 2014","Q4 2014"
         ,"Q1 2015","Q2 2015","Q3 2015","Q4 2015"
         ,"Q1 2016","Q2 2016","Q3 2016","Q4 2016"
         ,"Q1 2017","Q2 2017","Q3 2017","Q4 2017"
         ,"Q1 2018","Q2 2018","Q3 2018","Q4 2018"
         ,"Q1 2019","Q2 2019","Q3 2019","Q4 2019"
         ,"Q1 2020","Q2 2020","Q3 2020","Q4 2020"
         ,"Q1 2021","Q2 2021")

         ,Revenue=c(345,431,467,731,737,895,954,
                    1131,1058,1184,1262,1585,1458,
                    1813,2016,2585,2502,2910,3203,
                    3851,3543,4042,4501,5842,5382,6436,
                    7011,8809,8032,9321,10328,12972,11966,
                    13231,13727,16914,15077,16886,17652,21082,
                    17737,18687,21470,28071,26171,29080)

         ,Users=c(431,482,550,608,680,739,800,845,901,955,
                  1007,1056,1110,1155,1189,1228,1276,1317,1350,
                  1393,1441,1490,1545,1591,1645,1712,1788,1860,1936,2006,
                  2072,2129,2196,2234,2271,2320,2375,2414,2449,2498,2603,
                  2701,2740,2797,2853,2900))
```

Pour avoir une visibilité claire des données, on commence par afficher deux histogrammes :

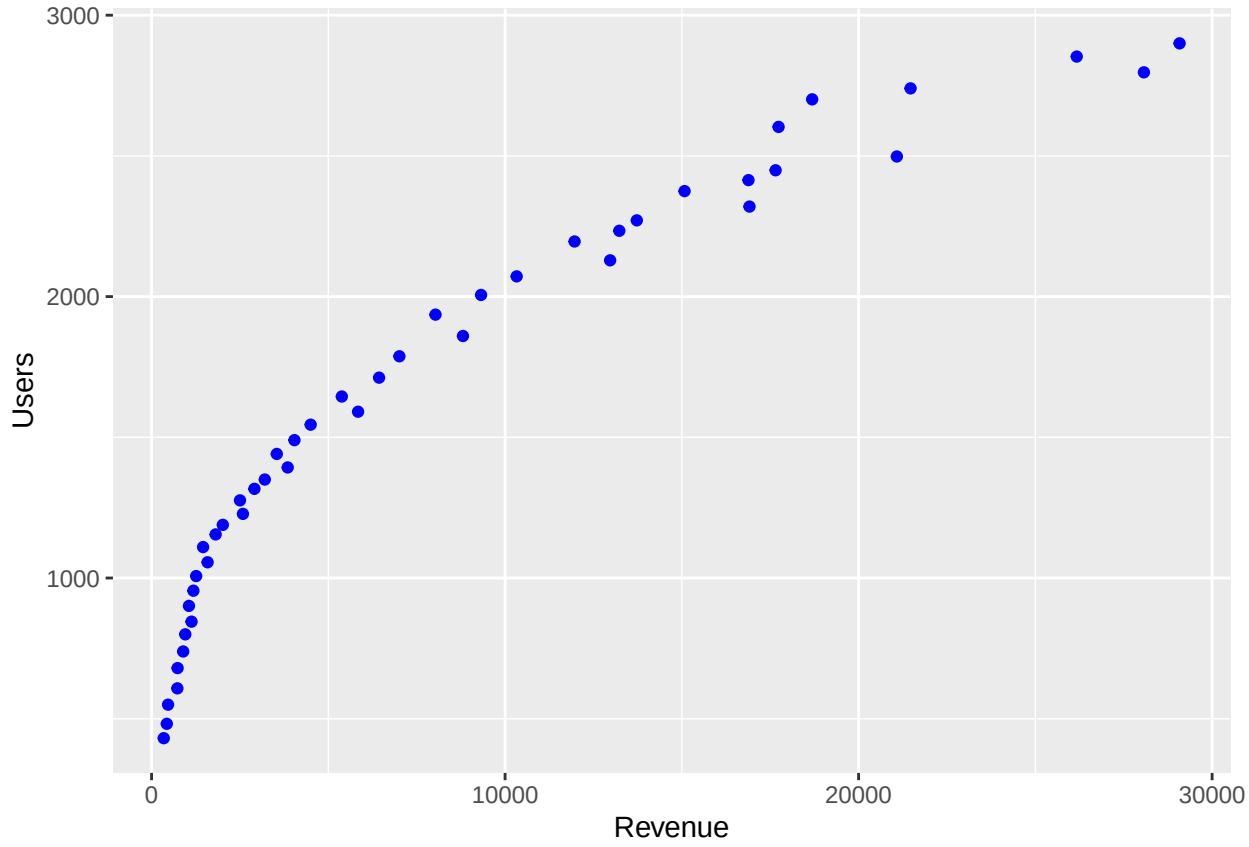
- le premier affiche l'évolution du revenue de Facebook
- le second affiche le nombre d'utilisateurs par année


Facebook revenues by year
0
5000
15000
25000
Q1 2010
Q2 2010
Q3 2010
Q4 2010
Q1 2011
Q2 2011
Q3 2011
Q4 2011
Q1 2012
Q2 2012
Q3 2012
Q4 2012
Q1 2013
Q2 2013
Q3 2013
Q4 2013
Q1 2014
Q2 2014
Q3 2014
Q4 2014
Q1 2015
Q2 2015
Q3 2015
Q4 2015
Q1 2016
Q2 2016
Q3 2016
Q4 2016
Q1 2017
Q2 2017
Q3 2017
Q4 2017
Q1 2018
Q2 2018
Q3 2018
Q4 2018
Q1 2019
Q2 2019
Q3 2019
Q4 2019
Q1 2020
Q2 2020
Q3 2020
Q4 2020
Q1 2021
Q2 2021
Years



On constate que le revenu annuel de Facebook croît de manière exponentielle tandis que le nombre d'utilisateurs croît de manière linéaire. Ensuite, on affiche le nombre d'utilisateurs en fonction des revenus :

```
ggplot(data, aes(x = Revenue, y = Users)) +
  geom_point(col="blue",pch=19)
```



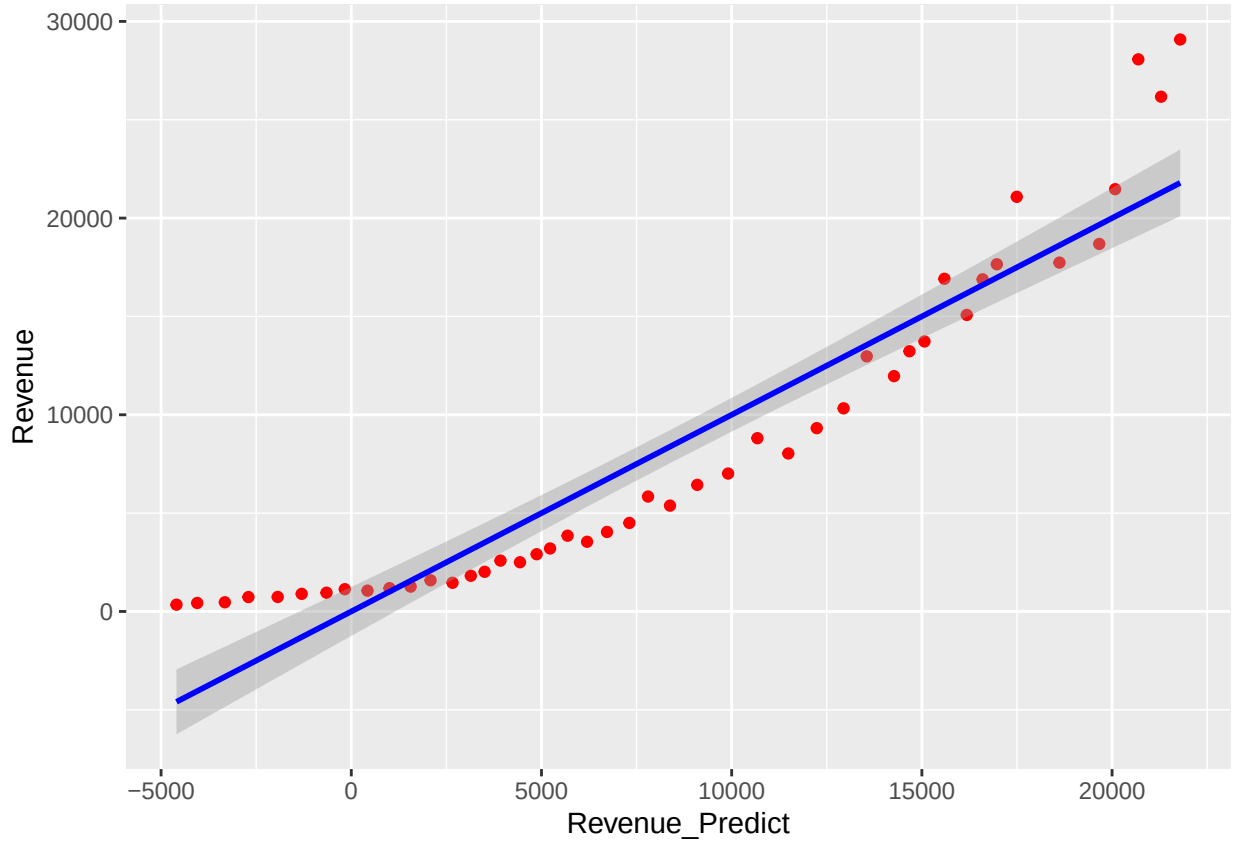
Après avoir pris connaissance des données, nous pouvons désormais créer notre modèle grâce à la fonction *lm* puis nous pouvons faire un premier test de prédiction des revenus grâce au modèle créé :

```
model <- lm(Revenue~Users,data=data)
summary(model)
```

```
##
## Call:
## lm(formula = Revenue ~ Users, data = data)
##
## Residuals:
##    Min     1Q Median     3Q    Max
##  -3457  -1963  -1040   1372   7381
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -9201.1045  1043.2094   -8.82 2.77e-11 ***
## Users          10.6869     0.5815   18.38  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2829 on 44 degrees of freedom
## Multiple R-squared:  0.8847, Adjusted R-squared:  0.8821
## F-statistic: 337.7 on 1 and 44 DF,  p-value: < 2.2e-16
```

```
Revenue_Predict <- predict(model,data)
```

```
ggplot(model, aes(x = Revenue_Predict, y = Revenue),
       xlab="Revenue prediction",ylab="Actual Revenue") +
  geom_point(col="red",pch=19) +
  stat_smooth(method = 'lm',formula = y~x,col="blue")
```



On vérifie si le modèle est performant grâce à sa précision (*accuracy*) en utilisant la RMSE (**R**oot **M**ean **S**quare **E**rror) :

```
RMSE <- sqrt(mean(Revenue_Predict-data$Revenue)^2)
RMSE
```

```
## [1] 2.392115e-12
```

Nous obtenons une précision de l'ordre de 10^{-12}. Plus cette valeur est proche de 0, meilleur est l'ajustement aux données. Améliorons notre modèle en utilisant les revenus normalisés :

```
Revenue_normalize <- log10(data$Revenue)

model_2 <- lm(Revenue_normalize~Users,data=data)
summary(model)
```

```
##
## Call:
## lm(formula = Revenue ~ Users, data = data)
##
## Residuals:
##    Min     1Q Median     3Q    Max
##  -3457  -1963  -1040   1372   7381
```

```
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -9201.1045  1043.2094   -8.82 2.77e-11 ***
## Users          10.6869     0.5815   18.38  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2829 on 44 degrees of freedom
## Multiple R-squared:  0.8847, Adjusted R-squared:  0.8821
## F-statistic: 337.7 on 1 and 44 DF,  p-value: < 2.2e-16
```

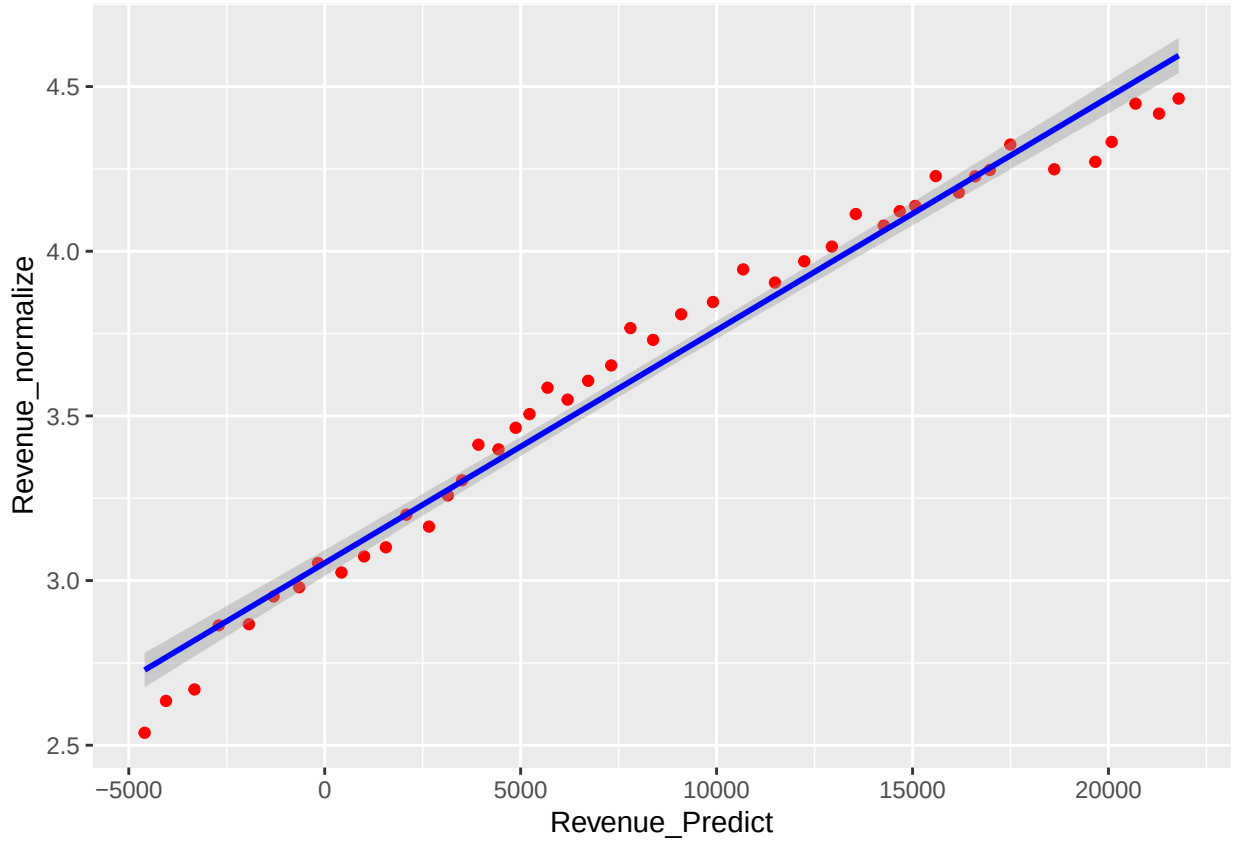
Voici une comparaison entre les revenus normalisés réels et ceux prédits ainsi qu'un graphique les représentant :

```
Revenue_Predict_norm <- predict(model_2,data)

Comparaison <- data.frame(Revenue_normalize,Revenue_Predict_norm)

Comparaison
```

```
##    Revenue_normalize Revenue_Predict_norm
## 1           2.537819             2.728391
## 2           2.634477             2.766929
## 3           2.669317             2.818313
## 4           2.863917             2.862141
## 5           2.867467             2.916548
## 6           2.951823             2.961132
## 7           2.979548             3.007226
## 8           3.053463             3.041231
## 9           3.024486             3.083547
## 10          3.073352             3.124352
## 11          3.101059             3.163646
## 12          3.200029             3.200673
## 13          3.163758             3.241478
## 14          3.258398             3.275483
## 15          3.304491             3.301175
## 16          3.412461             3.330645
## 17          3.398287             3.366917
## 18          3.463893             3.397898
## 19          3.505557             3.422835
## 20          3.585574             3.455328
## 21          3.549371             3.491599
## 22          3.606596             3.528626
## 23          3.653309             3.570187
## 24          3.766562             3.604947
## 25          3.730944             3.645752
## 26          3.808616             3.696381
## 27          3.845780             3.753811
## 28          3.944927             3.808217
## 29          3.904824             3.865647
## 30          3.969463             3.918543
## 31          4.014016             3.968416
## 32          4.113007             4.011488
## 33          4.077949             4.062117
```

```
## 34          4.121593            4.090831
## 35          4.137576            4.118790
## 36          4.228246            4.155817
## 37          4.178315            4.197378
## 38          4.227527            4.226849
## 39          4.246794            4.253296
## 40          4.323912            4.290323
## 41          4.248880            4.369667
## 42          4.271540            4.443721
## 43          4.331832            4.473191
## 44          4.448258            4.516263
## 45          4.417820            4.558580
## 46          4.463594            4.594096
```

```
ggplot(model_2, aes(x = Revenue_Predict, y = Revenue_normalize),
       xlab="Revenue prediction",ylab="Actual Revenue") +
  geom_point(col="red",pch=19) +
  stat_smooth(method = 'lm',formula = y~x,col="blue")
```



Nous avons désormais une nouvelle RMSE :

```
## [1] 9.657649e-18
```

La précision de notre second modèle est de l'ordre de 10^{-18}. Nous pouvons en conclure que la RMSE de notre second modèle est meilleure qu'elle celle du premier ($9.66e-18 << 2.39e-12$).

V. Real estate data

Après avoir nettoyé l'environnement et enregistré les nouvelles données du fichier *housedata* dans la variable *data*, nous commençons par supprimer les colonnes *id* et *date*.

```
data$id <- NULL
data$date <- NULL
```

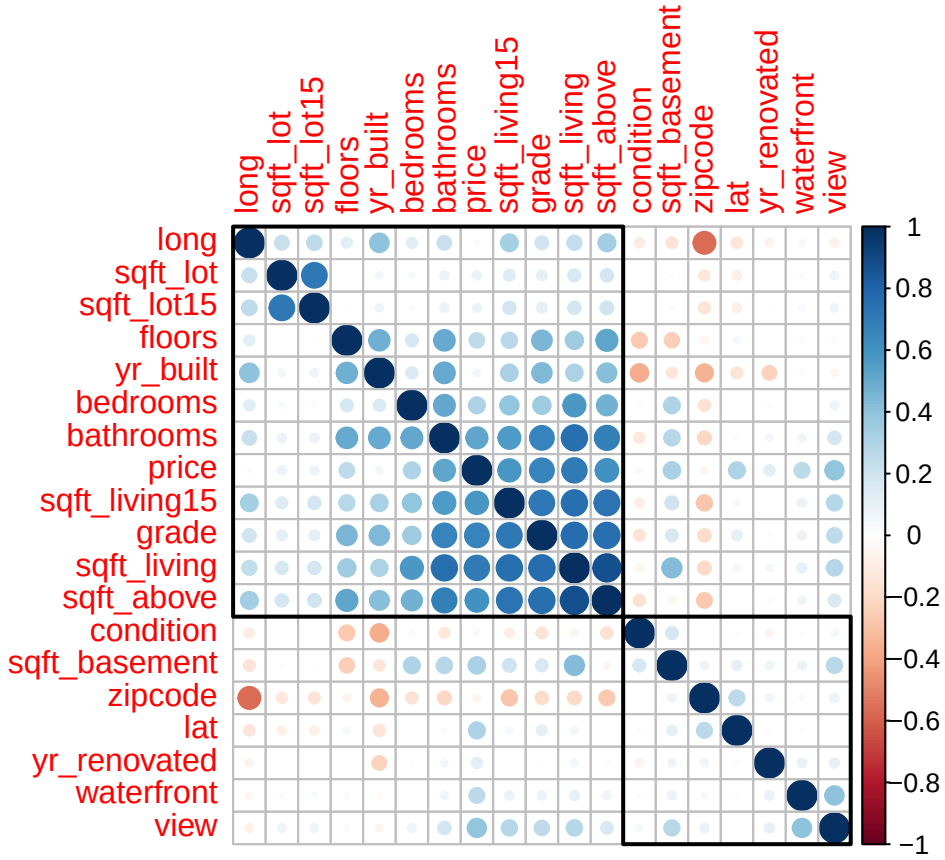
Puis, nous calculons le pourcentage de données manquantes afin de s'assurer de travailler avec des données complètes. Le résultat suivant nous montre qu'aucunes données n'est manquantes.

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

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

Par suite, il est possible de visualiser la corrélation entre les variables grâce à la fonction *cor* :

```
M <- cor(data)
corrplot(M, order = 'hclust', addrect = 2)
```



On constate que certaines données sont très fortement corrélées comme la surface habitable *sqft_living* et la surface hors sol *sqft_above* tandis que d'autre ne le sont pas. L'objectif étant de trouver un modèle linéaire afin de prédire le prix, nous pouvons déjà avoir une idée des varibales qui auront de l'impotance dans le calcul du prix. Par exemple, le prix semble être fortement corrélé à la surface habitable (≈ 0.8), la note *grade* (≈ 0.6) et un peu corrélé à la variable salle de bain *bathroom* (≈ 0.3). Plus le coéfficient de corrélation est proche de 1, plus les variables sont corrélées.

Il est aussi possible d'avoir des informations sur les données grâce à la fonction *summary* (voir ci-dessous) comme la moyenne de chaque variable, leur minimum etmaximum, les quartiles, etc.

```
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_built     : 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
```

Nous allons maintenant séparer notre data set en deux data set indépendants afin d'avoir un ensemble de données pour l'entraînement de notre modèle (variable *train*) et un ensemble pour le tester (variable *test*).

```
set.seed(2)
data_Split <- sort(sample(nrow(data),nrow(data)*.70))
train <- data[data_Split,]
test <- data[-data_Split,]
```

```
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: (1 not defined because of singularities)
##                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    1.364e+02  5.213e+00  26.169  < 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_above     4.391e+01  5.143e+00   8.537  < 2e-16 ***
## sqft_basement         NA         NA      NA       NA
## yr_built      -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
```

On constate d'après le message *Coefficients: (1 not defined because of singularities)* que Certaines des variables ne sont pas définies. En effet, la singularité signifie que les variables ne sont pas linéairement indépendantes.

Si on supprime la variable qui retourne NA (*sqft_basement*) dans le résumé ci-dessus, alors le résultat pour les autres variables reste inchangé. En effet, l'information donnée par cette variable est déjà contenue dans les autres variables et est donc redondante.

Ajustons notre modèle en éliminant la colonne *sqft_basement* :

```
data$sqft_basement <- NULL
```

En suivant le même processus que précédement, nous obtenons un nouveau modèle :

```
set.seed(2)
data_Split <- sort(sample(nrow(data),nrow(data)*.70))
train <- data[data_Split,]
test <- data[-data_Split,]

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

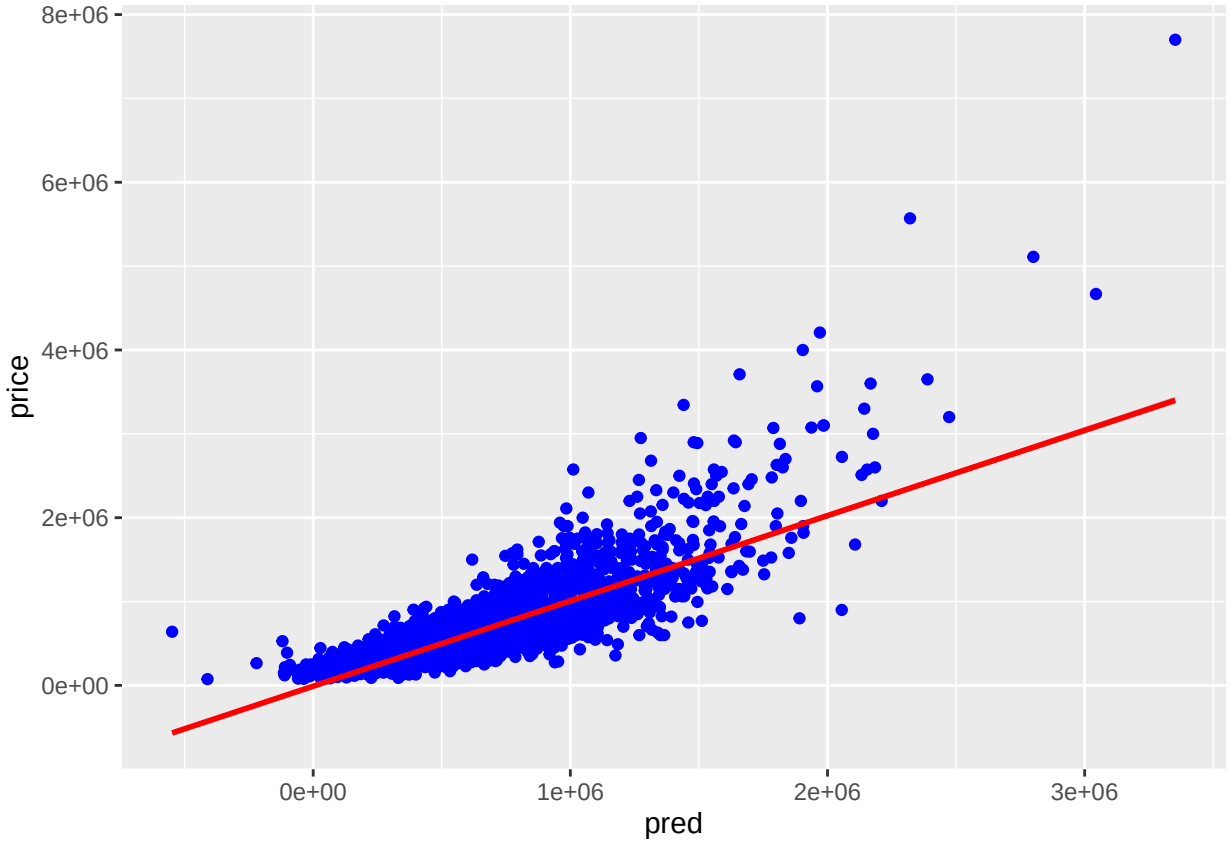
```
##
## Call:
## lm(formula = price ~ ., data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1242164   -99633    -9849    77211  4059525
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)    8.387e+06  3.464e+06   2.421   0.0155 *
## bedrooms      -3.631e+04  2.334e+03 -15.556  < 2e-16 ***
## bathrooms      4.893e+04  3.860e+03  12.674  < 2e-16 ***
## sqft_living    1.364e+02  5.213e+00  26.169  < 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_above     4.391e+01  5.143e+00   8.537  < 2e-16 ***
## yr_built      -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
```

Ainsi nous pouvons lire les coéfficients estimés pour chaque variable comme par exemple 4.893e+04 pour la variable salle de bain ou encore 1.364e+02 pour la surface habitable. Nous avons aussi le résultat des tests statistiques comme l'erreur standard. La dernière colonne qui contient les astérisques * indique le risque de

se tromper. Plus il y a d'astérisques, plus le risque de se tromper est faible. Deux étoiles correspondent à un risque de première espèce α de 10^{-2}. Si la p-value est inférieure au risque α, alors on conserve le coéfficient. Cependant, si la p-value est supérieure au risque α, alors on accèpte l'hypothèse que le coéfficient soit nul et donc que le prix ne dépende pas de celui-ci.

Enfin, on calcul la prédiction grâce à notre modèle sur l'échantillon de test puis on affiche le résultat :

```
pred <- predict(model,test)

ggplot(test, aes(x = pred, y = price)) +
  geom_point(col="blue",pch=19) +
  stat_smooth(formula = y~x ,method = 'lm',col="red")
```



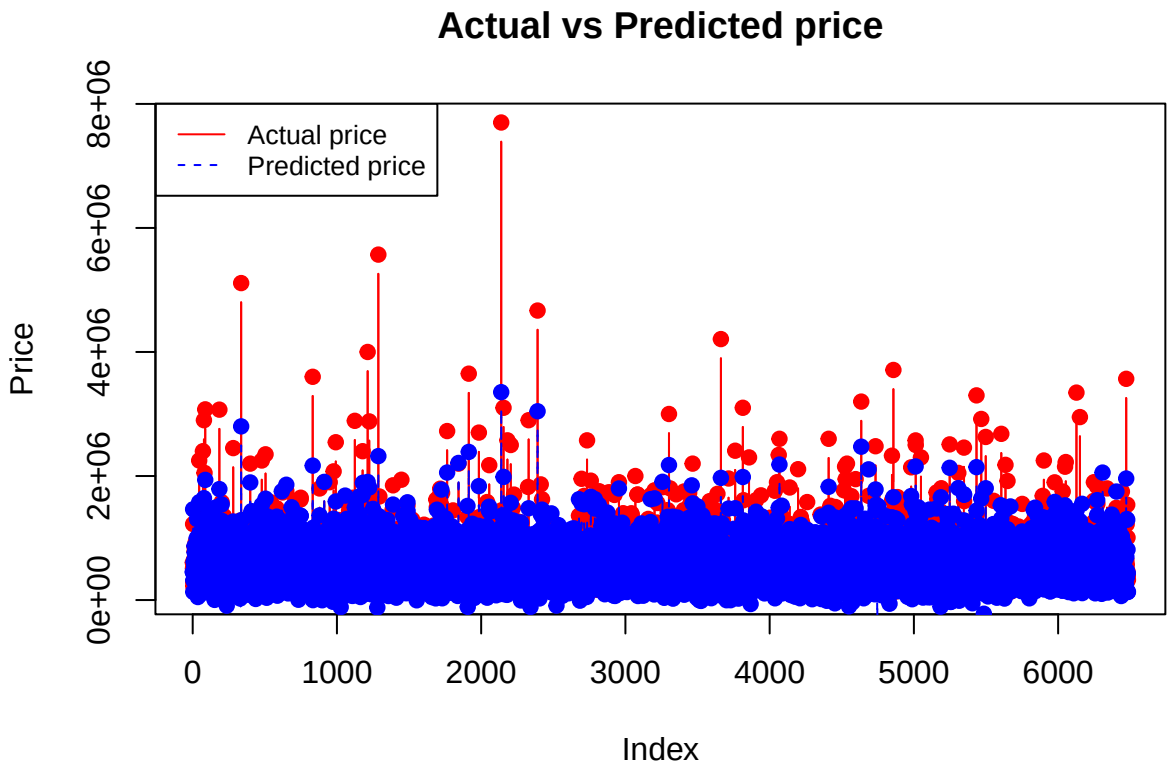
Voici une comparaison entre le prix réel et le prix prédit ainsi qu'un graphique illustrant cette comparaison :

```
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
```

```
# Affichage du prix réel en rouge
plot(test$price,type="b", pch=19, col="red",ylab="Price",main="Actual vs Predicted price")
# Ajout du prix prédit en bleu
```

```
lines(pred, pch=19, col="blue", type="b", lty=2)
# Ajout de la légende
legend("topleft",legend=c("Actual price", "Predicted price"),
       col=c("red", "blue"), lty=1:2000, cex=0.8)
```



Pour conclure, nous calculons la précision de notre modèle :

```
RMSE <- sqrt(mean(pred-test$price)^2)
RMSE
```

```
## [1] 281.519
```

La précision n'est pas bonne car elle est très grande donc on ne peut pas valider notre modèle, mais il y a une solution c'est de normaliser la variable qu'on veut prédire tout en utilisant la fonction log :

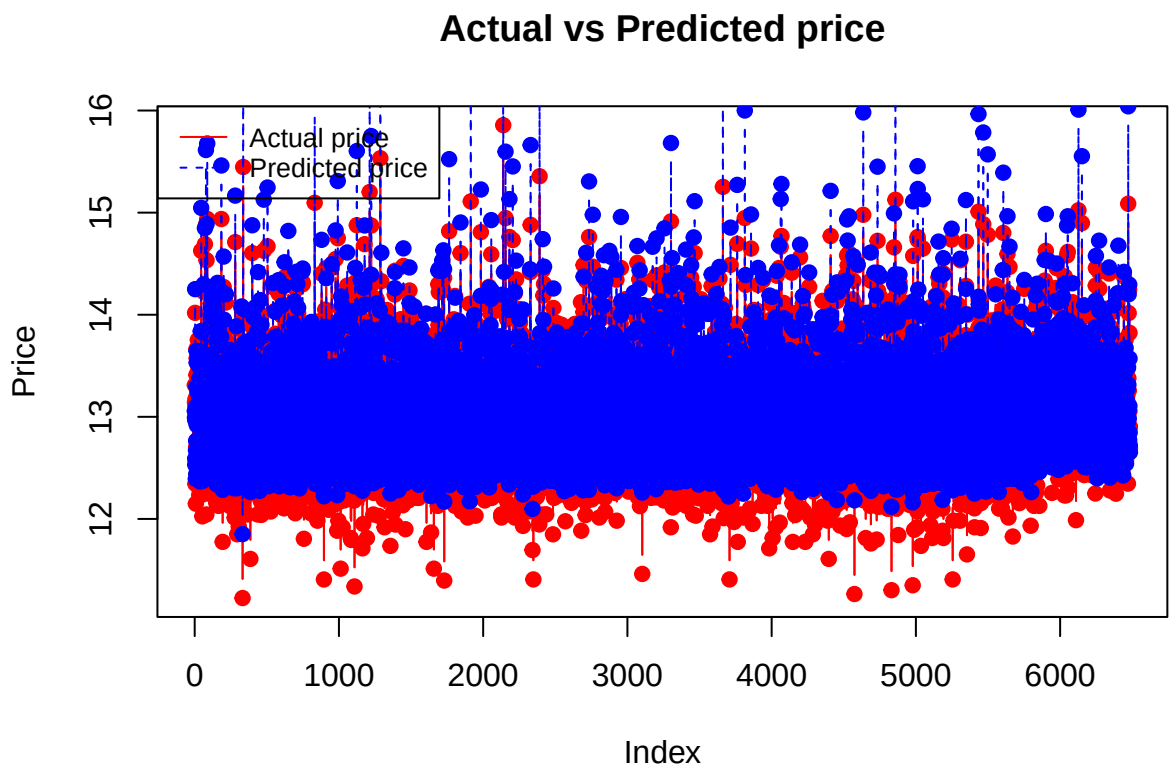
```
data$price_normalize <- log(data$price)
```

Ensuite on va diviser nos données en apprentissage et du test et créant le nouveau modèle :

```
set.seed(2)
#Data test and train split
data_Split <- sort(sample(nrow(data),nrow(data)*.70))
train <- data[data_Split,]
test <- data[-data_Split,]

#We create the model
model_normalize <- lm(price_normalize~.,data=train)
#Prediction
pred <- predict(model_normalize,test)
```

Ensuite on affiche le graphe *Actual vs predicted price*:

```
plot(test$price_normalize,type="b", pch=19, col="red",ylab="Price",main="Actual vs Predicted price")
# Ajouter une ligne
lines(pred, pch=19, 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)
```



Et finalement on affiche **RMSE** de notre nouveau modèle:

```
RMSE <- sqrt(mean(pred-test$price_normalize)^2)
RMSE
```

```
## [1] 0.0020114
```

Donc comme conclusion on a pu trouver RMSE d'ordre *0.002* qui est une bonne valeur pour la validation du modèle.