Challenge B

Alexandre Briois, Anthony Ruiz

8 december 2017

Challenge B

This is the link of the github repo : https://github.com/Ashinnor/AlexandreBrioisAnthonyRuizRprog Please find in the github account the databases you need to load when asked by this markdown file. In order you'll be asked to open : train test The CNIL data The SIREN data

TAsk 1B - Predicting house prices in Ames, Iowa (continued)

Step 1 -

We have chosen the random forest method. Our intuition regarding of how it works is the following :

This method consisted in creating a certain number of regression trees (because we are in the case where the output is continuous, otherwise we talk about classification trees). We'll make 500 trees in our database.

The software takes 500 subsets of our observation data, and it creates one tree per subset. (In general, each subset represents around 2/3 of the total observations, or 90% depending on the software. . .).

To create a tree :

-It takes randomly explanatory variables of our database. The root of a tree will be observations of a variable. For the example, take a variable of our database : the size of the house in square feet.

-Then, its splits our tree in several branches with another value of another character (for example the number of bedrooms). It repeats this step an appropriate number of time (We can define it to find a compromise between precision of the results and the time of calculation of the software).

It's important to be precise so that any output (or any predictions) of the trees is hence a value of our database. That was not the case when we made a regression like OLS regression.

Now, if we want to estimate the housing price of another database, the software will follow ways created in previous trees. In other word, (if we take the same example as before), each tree "votes" for an output value, and we obtain a mean of these votes. And in this way, we'll obtain a value of the housing price of our house. That's what we have understood of this method of Machine Learning.

Step 2 -

Precisions regarding the error metrics (to compare our two models): The error metric we'll use to evaluate our models is the RMSE (root mean squared error). This error measure gives more weight to larger residuals than smaller ones). What this means is that we consider that missing the prediction for housing price by 2000 dollar, on a given house, is not only twice as bad as missing by 1000 dollar, but worse than that. We'll use the MAE (mean absolute error) as another error metric. It gives equal weight to the residuals, which means 2000 dollar is actually twice as bad as 1000 dollar.

Step 3 -

We perform the predictions on the test data. Details and comments of the code are in the code as requested. We finally obtain this table and the two graphs to make the comparison :

```
head(Final_table)
```

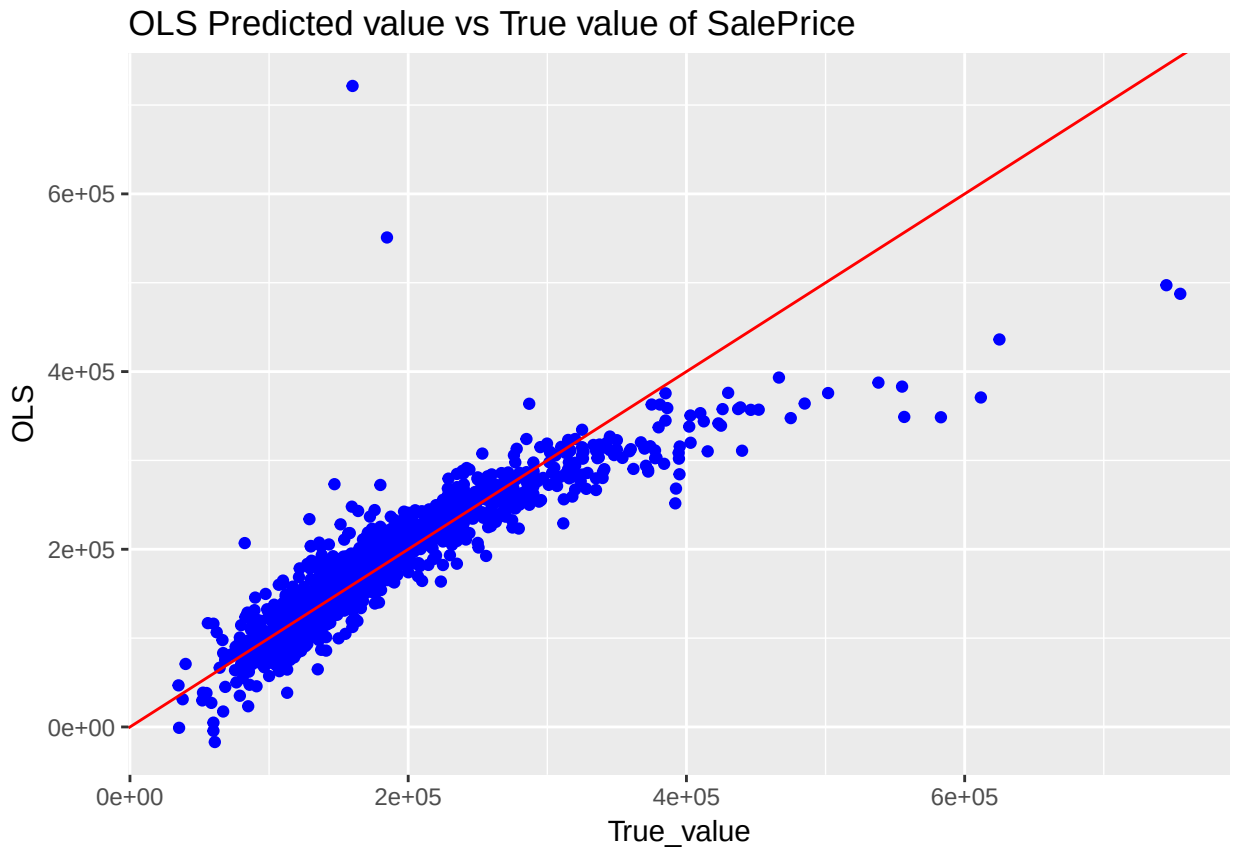
```
##   Id True_value      OLS RandomForest
## 1  1     208500 220479.6     207947.9
## 2  2     181500 189712.0     176487.2
## 3  3     223500 225568.8     221958.7
## 4  4     140000 181959.5     156454.4
## 5  5     250000 280926.3     272431.2
## 6  6     143000 165509.5     148848.7
```

We see that both are very close to the reality. It's difficult to say which method is the best among them

Maybe, with a graph it will be easier to compare : (We put the true value on the x-axe and the estimation on the y-axe, and put the line y = x to show if the estimation is close or not of the true value)

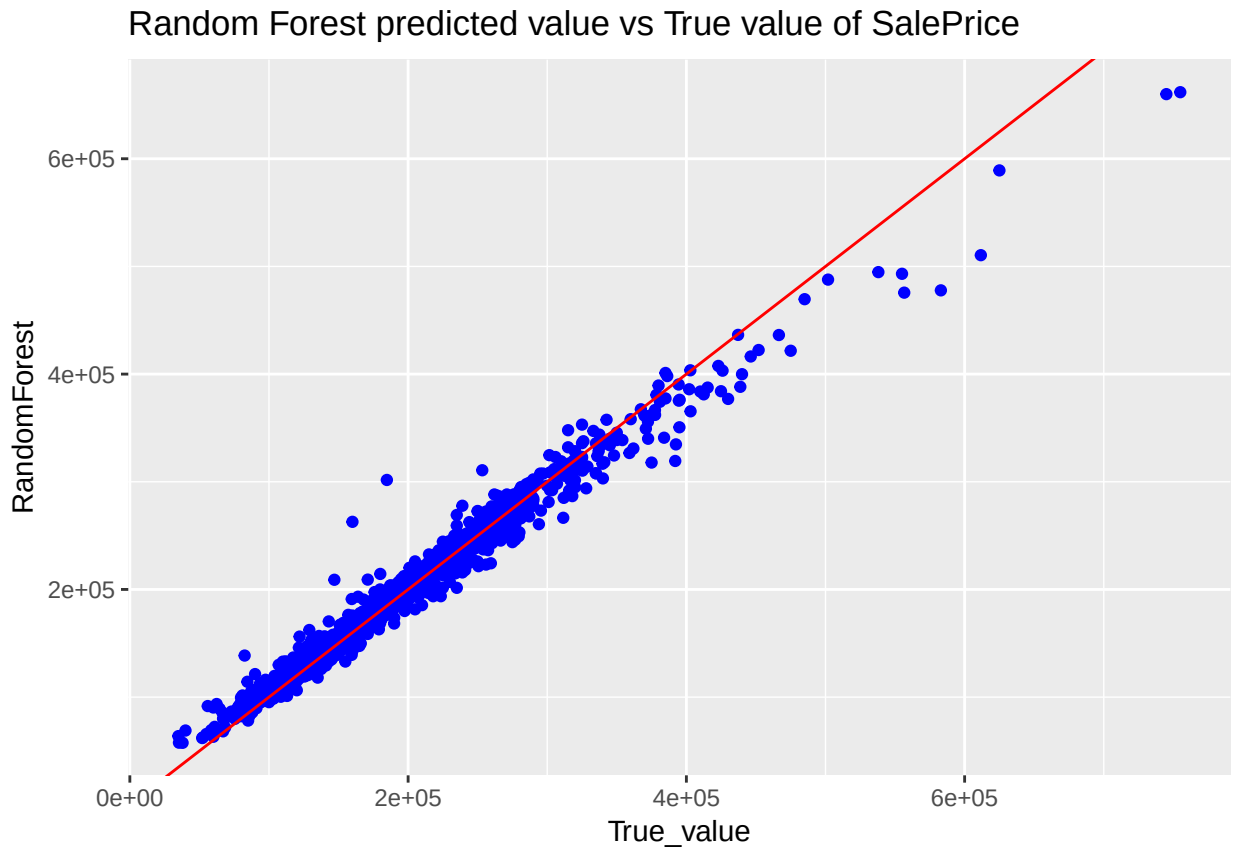
Plot of the OLS and the True Value of SalePrice :

```
ggplot(data = Final_table,aes(x = True_value, y = OLS)) +
  geom_point(colour = "blue") +
  geom_abline(intercept = 0, slope = 1, colour = "red") +
  ggtitle("OLS Predicted value vs True value of SalePrice")
```



Plot of the Random Forest and the true value of SalePrice :

```
ggplot(data = Final_table,aes(x = True_value, y = RandomForest)) +
  geom_point(colour = "blue") +
  geom_abline(intercept = 0, slope = 1, colour = "red") +
  ggtitle("Random Forest predicted value vs True value of SalePrice")
```



Thanks to theses two graphs, we can say the Random Forest seems more precise than OLS, essentially with large value of SalePrice (from 300 000 dollar). Indeed, there's less dispersion around the estimation line.

Task 2B - Overfitting in Machine Learning (continued)

Step 1 -

We take back our previous code to simulate the data and the training and test samples in the first part of the code. On the second part, we simulate the local linear model. We will use the function npreg to obtain the local linear models.

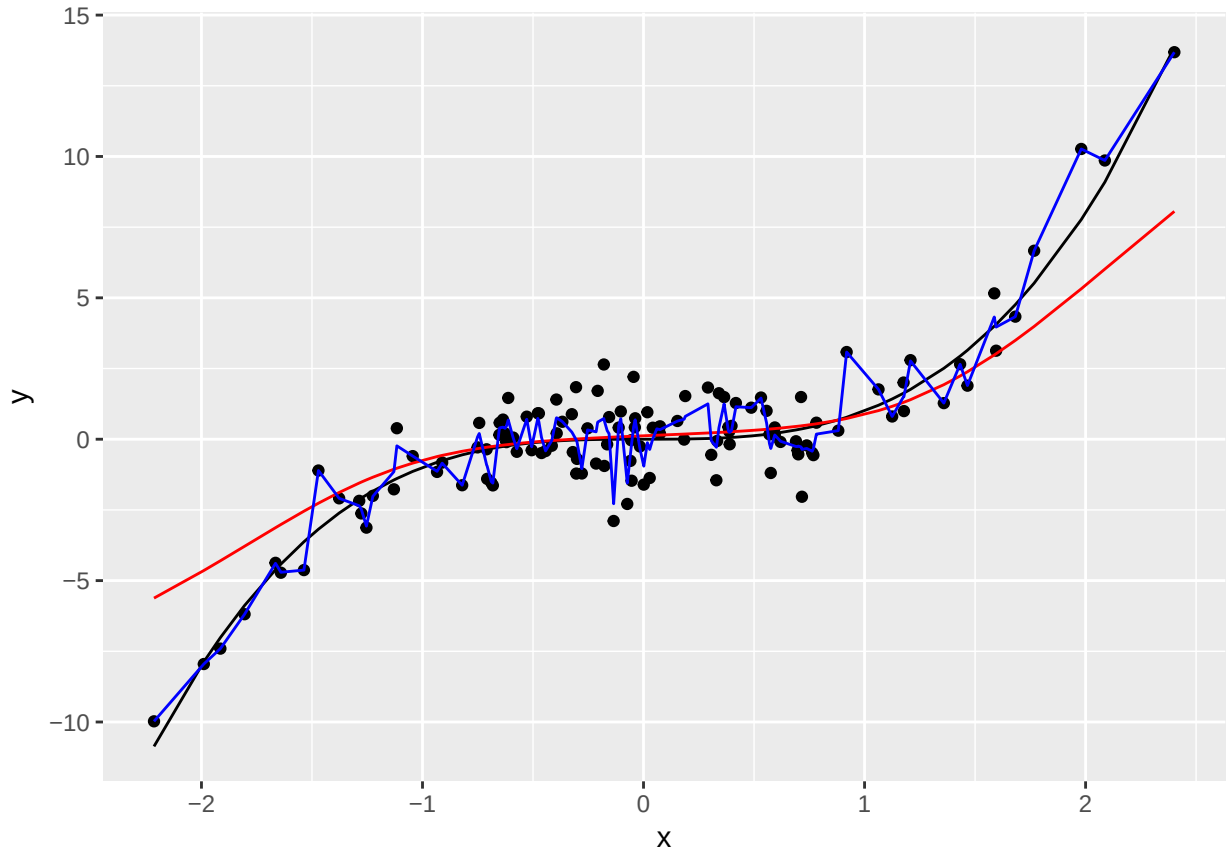
```
ll.fit.lowflex <- npreg(y ~ x, data = training, method = "ll", bws = 0.5)
```

Step 2 - And for the high-flexibility one :

```
ll.fit.highflex <- npreg(y ~ x, data = training, method = "ll", bws = 0.01)
```

Step 3 - Scatterplot

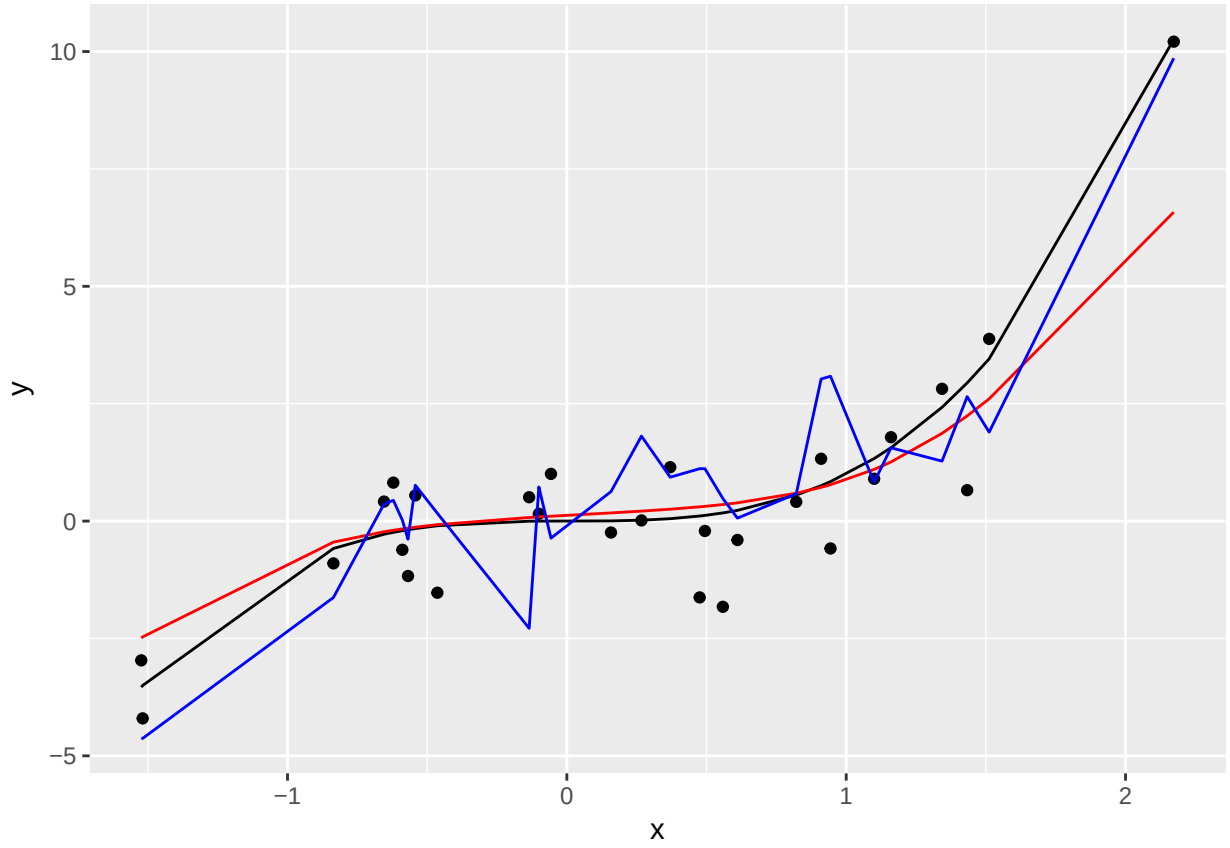
```
ggplot(training) + geom_point(mapping = aes(x = x, y = y)) +
  geom_line(mapping = aes(x = x, y = y.true)) +
  geom_line(mapping = aes(x = x, y = y.ll.lowflex), color = "red") +
  geom_line(mapping = aes(x = x, y = y.ll.highflex), color = "blue")
```



Step 4 - We can see that between the two models, the more variable predictions are the ones from the high-flexibility model (the blue line displays a lot of fluctuations). However, it's bias is lowered by those fluctuations : the blue line covers much more space/points than the red one. So the high-flexibility model is the more variable and shows the lowest bias.

Step 5 -

```
#We repeat the same plot on the test data
ggplot(test) + geom_point(mapping = aes(x = x, y = y)) +
  geom_line(mapping = aes(x = x, y = y.true)) +
  geom_line(mapping = aes(x = x, y = y.ll.lowflex), color = "red") +
  geom_line(mapping = aes(x = x, y = y.ll.highflex), color = "blue")
```



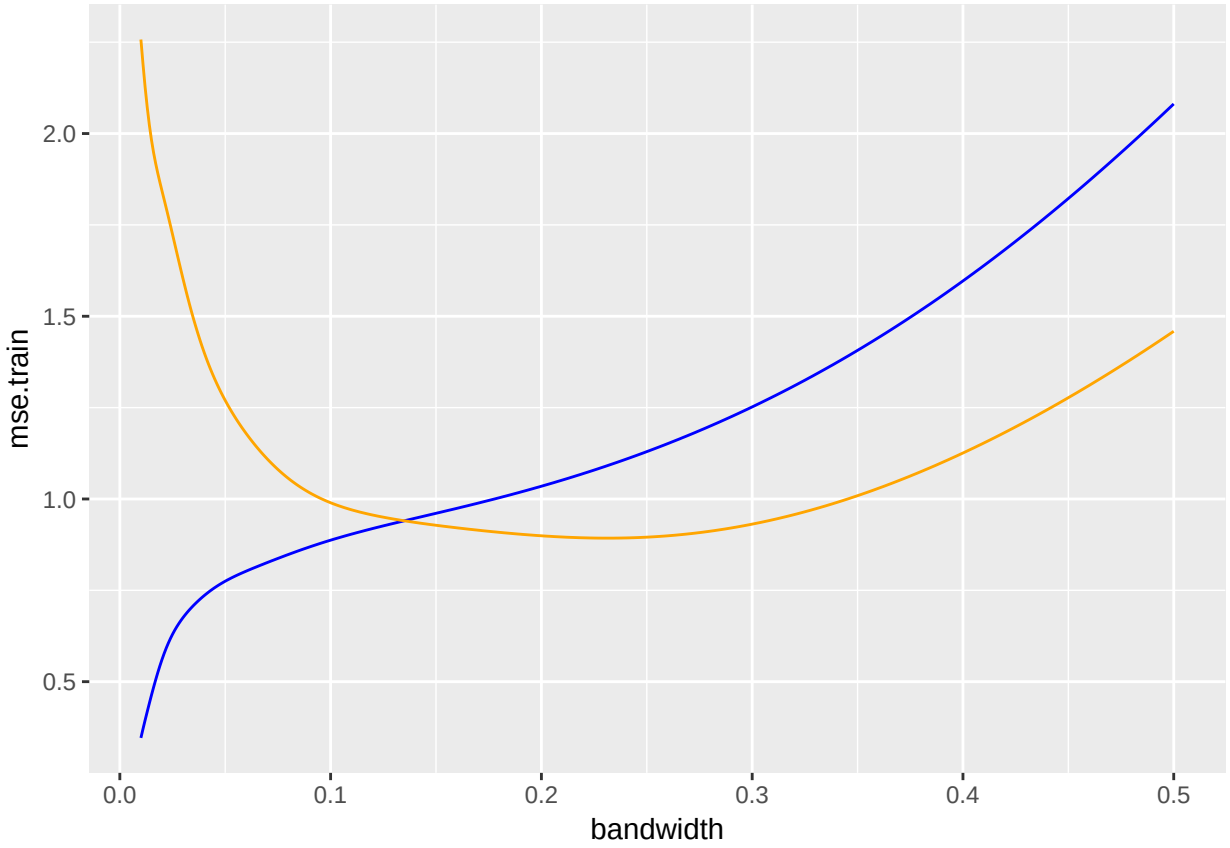
On this new graph we can see that the roles haven't really switched, the high-flexibility model has more variable predictions. However, we can suppose that it's bias has increased since there's more dispersion between the points and the blue line. It's harder to tell which model is the best now.

Step 6 - We create the vector of bandwidth going from 0.01 to 0.5 with a step of 0.001.

We use the function lapply and the function npreg to train the model on each bandwidth. Step 7, 8 and 9 are detailed in the code.

Step 10 - Here is the final plot on the evolution of the MSE of both datas with respect to the change in bandwidth :

```
mse.df <- tbl_df(data.frame(bandwidth = bw, mse.train = mse.train.results, mse.test = mse.test.results))
ggplot(mse.df) +
  geom_line(mapping = aes(x = bandwidth, y = mse.train), color = "blue") +
  geom_line(mapping = aes(x = bandwidth, y = mse.test), color = "orange")
```



We can see on this final plot that for a very low bandwidth, the best estimation is made on the test data. However, as the bandwidth increases, it is the training data that takes over. This conclusion matches our guess from step 4 and 5.

Task 3B - Privacy regulation compliance in France

Step 1 - CNIL dataset import. See code.

Step 2 -

Details are in the code. We obtain this final table :

```
cbind(Step2Table)
```

```
##    Code_Postal    x
## 1           10  119
## 2           11  116
## 3           12  106
## 4           13  474
## 5           14  269
## 6           15   59
## 7           16  129
## 8           17  163
## 9           18   84
## 10          19   54
## 11          20  101
## 12          21  172
## 13          22  133
```

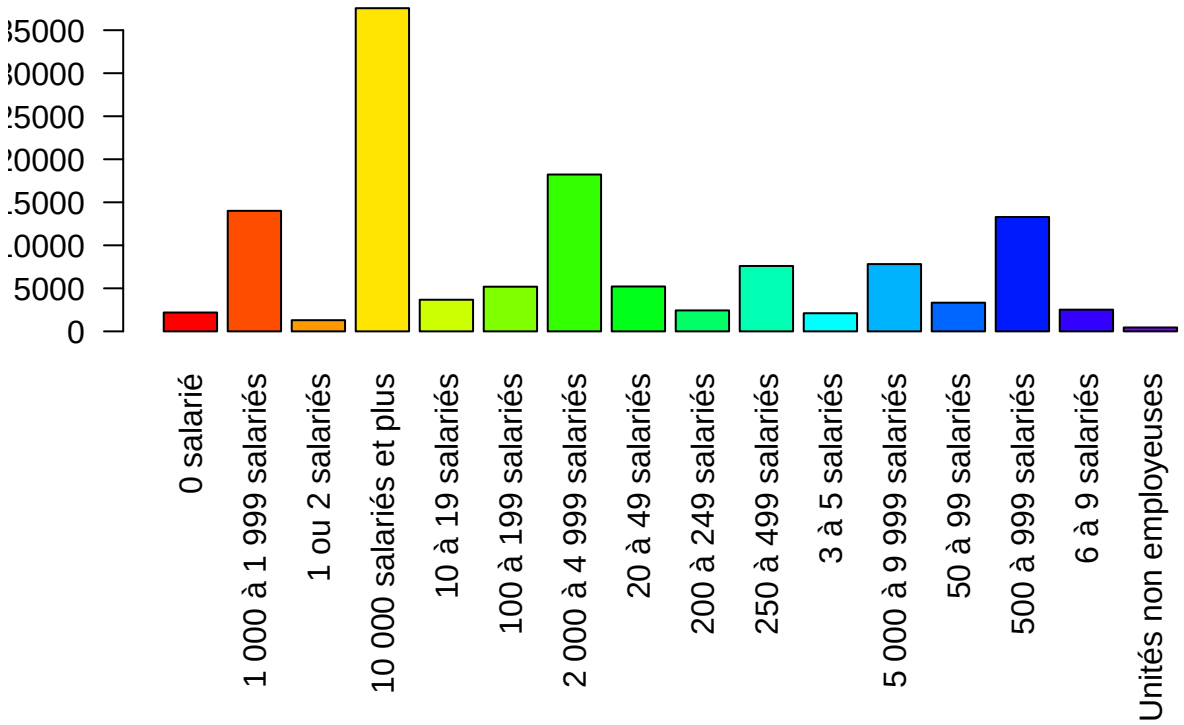
```
## 14          23   52
## 15          24   90
## 16          25  146
## 17          26  138
## 18          27  115
## 19          28  103
## 20          29  174
## 21          30  143
## 22          31  323
## 23          32   94
## 24          33  369
## 25          34  293
## 26          35  283
## 27          36   55
## 28          37  183
## 29          38  420
## 30          39   67
## 31          40  188
## 32          41  117
## 33          42  229
## 34          43  114
## 35          44  339
## 36          45  182
## 37          46   60
## 38          47  109
## 39          48   13
## 40          49  209
## 41          50  146
## 42          51  185
## 43          52   57
## 44          53  320
## 45          54  199
## 46          55   67
## 47          56  180
## 48          57  247
## 49          58   44
## 50          59  529
## 51          60  250
## 52          61  103
## 53          62  269
## 54          63  158
## 55          64  189
## 56          65   98
## 57          66  132
## 58          67  285
## 59          68  181
## 60          69  600
## 61          70   74
## 62          71  141
## 63          72  143
## 64          73  112
## 65          74  191
## 66          75 2042
## 67          76  288
```

```
## 68          77  226
## 69          78  283
## 70          79  133
## 71          80  172
## 72          81  132
## 73          82   83
## 74          83  212
## 75          84  143
## 76          85  195
## 77          86  158
## 78          87  113
## 79          88  126
## 80          89   90
## 81          90   29
## 82          91  225
## 83          92  935
## 84          93  310
## 85          94  289
## 86          95  177
## 87          97  247
## 88          98   25
```

Step 3 - The import takes 5 to 10 minutes here. To gain time, we are only taking the SIREN and the size of the firm from the database. After this, we just merge the two databases with respect to the SIREN.

Step 4 - After removing the duplicates, we are left with 15 811 observations, giving us this nice plot of the number of companies, by size. N.B. : this step won't work if you didn't run manually the step before with the huge database.

```
#This is just to adjust the margins of the barplot
par(mar=c(11,3,3,1))
barplot(table(LIBTEFEN),las=2,col=rainbow(20))
```


0
5000
0 salarié
1 000 à 1 999 salariés
1 ou 2 salariés
10 000 salariés et plus
10 à 19 salariés
100 à 199 salariés
2 000 à 4 999 salariés
20 à 49 salariés
200 à 249 salariés
250 à 499 salariés
3 à 5 salariés
5 000 à 9 999 salariés
50 à 99 salariés
500 à 999 salariés
6 à 9 salariés
Unités non employeuses