Challenge B

Fabien Dorati

29 novembre 2017

GitHub Link : https://github.com/Fab6531/Challenge-B-

TASK 1B

Step 1 :

We have decided to choose the random forest technique. This ML technique will predict differents values for the same characteristics using differents features (we will choose to use 5 features that will be randomly chosen) and it make the average to give us the best possible prediction.

Step 2 :

First, we delete the feature Id since it can't be relevant in order to determine the sale price.

Then we solve the problem of the missing values on the same way that in Challenge A.

```
set.seed(1)
#We use set.seed to have every time the same model
training_RF <- randomForest(SalePrice~., data=training2, ntree=500, mtry=5, na.acti
on = na.roughfix)
print(training_RF)
```

```
##
## Call:
##  randomForest(formula = SalePrice ~ ., data = training2, ntree = 500,      mtry
= 5, na.action = na.roughfix)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 5
##
##           Mean of squared residuals: 903366703
##                     % Var explained: 85.48
```

We use the randomForest function to create a model that we will be able to use to predict the Sale price of the houses. We choose 500 tries with 5 variables tried at each split. With this model, 85,48% of the variation of SalePrice is explained. Since it is over 80%, we can consider that it is a good model to predict.

Step 3 :

Then we create a prediction of the Sale Price with the Random Forest:

```
pre_train_RF <- predict(training_RF, data=test, type="response")
```

We predict the Sale Price thanks to a standard regression (using the regression founded in the solution of Challenge A :

```
##
## Call:
## lm(formula = SalePrice ~ MSZoning + LotArea + Neighborhood +
##     YearBuilt + OverallQual, data = training2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -217205  -22277   -1857   16999  342090
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)          -7.800e+05  1.820e+05  -4.286 1.95e-05 ***
## MSZoningFV            2.296e+03  2.059e+04   0.111 0.911242
## MSZoningRH           -3.057e+03  2.193e+04  -0.139 0.889180
## MSZoningRL            2.271e+04  1.769e+04   1.284 0.199468
## MSZoningRM            3.500e+03  1.673e+04   0.209 0.834332
## LotArea               1.221e+00  1.239e-01   9.859  < 2e-16 ***
## NeighborhoodBlueste  1.107e+04  3.195e+04   0.346 0.729084
## NeighborhoodBrDale  -7.151e+03  1.637e+04  -0.437 0.662347
## NeighborhoodBrkSide  3.049e+04  1.420e+04   2.147 0.032005 *
## NeighborhoodClearCr  3.779e+04  1.384e+04   2.731 0.006405 **
## NeighborhoodCollgCr  1.636e+04  1.078e+04   1.518 0.129350
## NeighborhoodCrawfor  5.981e+04  1.301e+04   4.596 4.73e-06 ***
## NeighborhoodEdwards  1.212e+04  1.208e+04   1.003 0.316055
## NeighborhoodGilbert  9.123e+03  1.129e+04   0.808 0.419164
## NeighborhoodIDOTRR   3.084e+04  1.640e+04   1.881 0.060220 .
## NeighborhoodMeadowV  2.584e+04  1.737e+04   1.488 0.137089
## NeighborhoodMitchel  1.815e+04  1.235e+04   1.470 0.141705
## NeighborhoodNAmes    1.809e+04  1.132e+04   1.598 0.110363
## NeighborhoodNoRidge  1.045e+05  1.221e+04   8.558  < 2e-16 ***
## NeighborhoodNPkVill -3.695e+03  1.745e+04  -0.212 0.832348
## NeighborhoodNridgHt  7.421e+04  1.135e+04   6.536 9.05e-11 ***
## NeighborhoodNWAmes   2.151e+04  1.159e+04   1.857 0.063553 .
## NeighborhoodOldTown  3.344e+04  1.441e+04   2.321 0.020423 *
## NeighborhoodSawyer   2.034e+04  1.202e+04   1.692 0.090816 .
## NeighborhoodSawyerW  2.118e+04  1.182e+04   1.792 0.073427 .
## NeighborhoodSomerst  3.461e+04  1.371e+04   2.525 0.011704 *
## NeighborhoodStoneBr  7.518e+04  1.326e+04   5.668 1.78e-08 ***
## NeighborhoodSWISU    2.592e+04  1.538e+04   1.685 0.092206 .
## NeighborhoodTimber   2.624e+04  1.254e+04   2.092 0.036620 *
## NeighborhoodVeenker  5.095e+04  1.636e+04   3.114 0.001886 **
## YearBuilt            3.539e+02  9.076e+01   3.899 0.000101 ***
## OverallQual          3.337e+04  1.295e+03  25.763  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 41850 on 1306 degrees of freedom
## Multiple R-squared:  0.7253, Adjusted R-squared:  0.7188
## F-statistic: 111.2 on 31 and 1306 DF,  p-value: < 2.2e-16
```

Finally, we use the summary function to compare the two predictions :

```
summary(prediction)
```

```
##         Id       SalePrice_predict
##  Min.   :1461   Min.   : 11634
##  1st Qu.:1826   1st Qu.:132988
##  Median :2190   Median :168929
##  Mean   :2190   Mean   :179671
##  3rd Qu.:2554   3rd Qu.:213735
##  Max.   :2919   Max.   :391638
##                 NA's   :4
```

```
summary(pre_train_RF)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   92436  139661  172312  186583  214851  503277
```

We can see that the median and the mean are slightly bigger with the ML technique than with the standard linear regression. The quartiles are equivalent but there is huge differences in the minimum and in the maximum. The minimum with the Random Forest is 92 436 dollars, that is a more realistic case than the prediction of the standard linear model (11 634 dollars).

TASK 2B :

Before Starting

Let’s install packages and run them. I use np package for non parametric regression(npreg) and ggplot for plotting as usual.

As Challenge A, I generate 150 random samples of x and epsilon following normal distribution with mean 0 and standard deviation 1. True value for y is x^3, but assume we observe x^3 with epsilon noise, which is y here. Then, to make trainset and testset, I divide 150 random samples randomly into 120 samples for trainset and 30 samples for testset. Both of them contain randomly generated x and observed y, which is x^3+e.

```
set.seed(5)
x <- rnorm(150, mean = 0, sd = 1)
e <- rnorm(150, mean = 0, sd = 1)
y <- x^3 + e

tasktable <- cbind(x,y)

yhat <- x^3

train_idx <- sample(1:nrow(tasktable),120,replace=FALSE)
trainset <- tasktable[train_idx,]
testset <- tasktable[-train_idx,]

trainset <- as.data.frame(trainset)
testset <- as.data.frame(testset)
```

Question 1

I will use npreg function for non parametric regression methods. Among these methods, I want to use local linear method with bandwidth 0.5. In this stage, I only use trainset with fixed bandwidth 0.5. I call this model ll.fit.lowflex and I predict y or estimate y with given x in my trainset. And I form xs and estimated y into a data frame predict.low

```
##
## Regression Data: 120 training points, in 1 variable(s)
##                  x
## Bandwidth(s): 0.5
##
## Kernel Regression Estimator: Local-Constant
## Bandwidth Type: Fixed
## Residual standard error: 1.500308
## R-squared: 0.9143661
##
## Continuous Kernel Type: Second-Order Gaussian
## No. Continuous Explanatory Vars.: 1
```

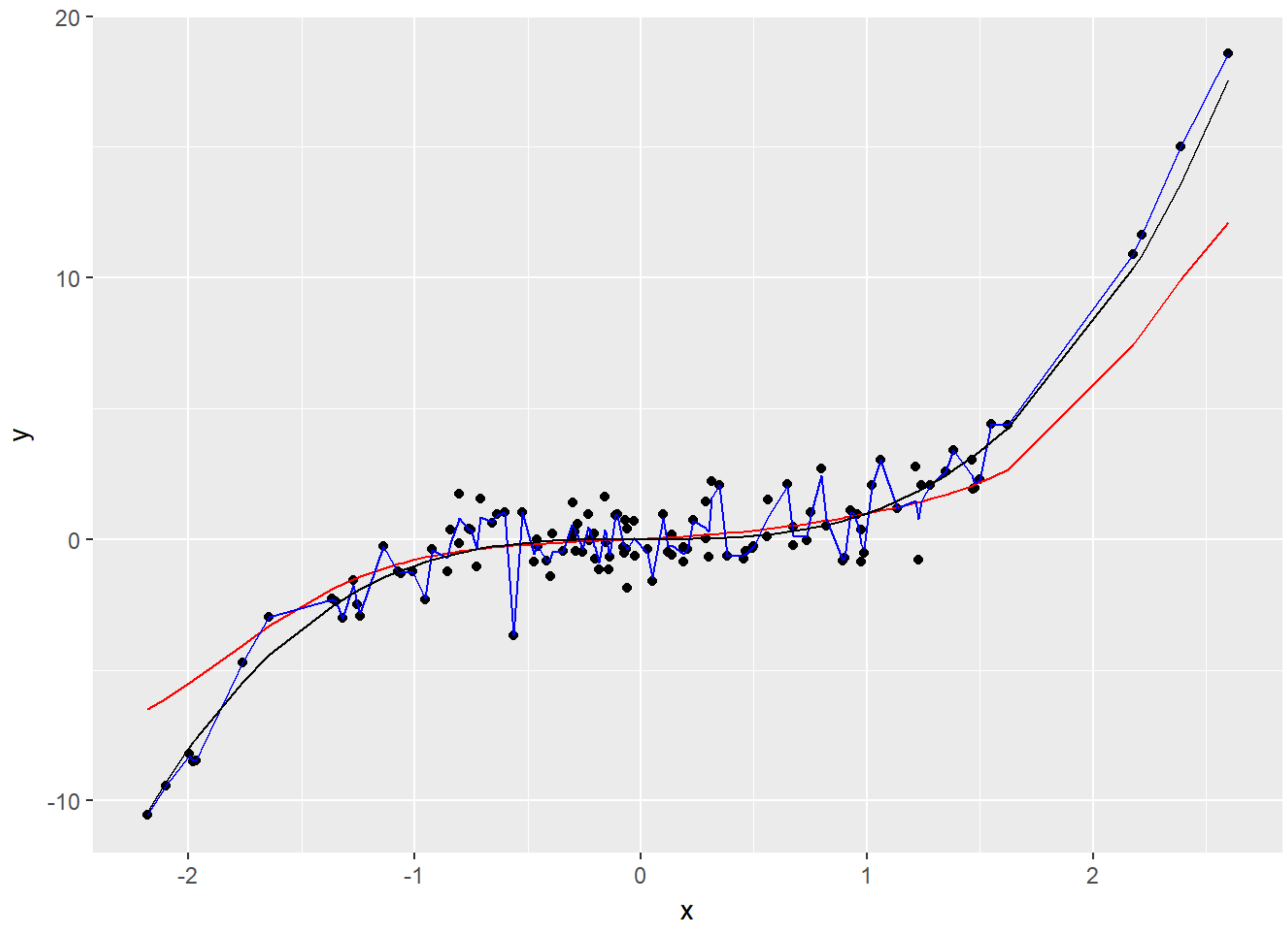
Question 2

Basically same steps are done for ll.fit.highflex, but bandwidth is different. Now flexibility is high, 0.01.

```
##
## Regression Data: 120 training points, in 1 variable(s)
##                   x
## Bandwidth(s): 0.01
##
## Kernel Regression Estimator: Local-Constant
## Bandwidth Type: Fixed
## Residual standard error: 0.4468237
## R-squared: 0.9840975
##
## Continuous Kernel Type: Second-Order Gaussian
## No. Continuous Explanatory Vars.: 1
```

Question 3

I want to plot 4 figures on the same plot. First, scatter point of observed y and x of trainset. Second, the black line telling the relationship between true y and x - of course this will be equation y=x^3. Closer my estimated ys are to true ys, my estimation is better. Third, the red line for ll.fit.lowflex. As we can see, the line is really smooth (Perhaps too smooth). ll.fit.lowflex is somehow strictforward in the sense of prediction, since its variance is low. However, ll.fit.lowflex does not allow enough exceptions. Therefore, bias is high. Fourth, the blue line for ll.fit.highflex. This line is rather complex (Perhaps too complex). It catches most of exceptions, but at the same time the relationship between y and x becomes too complex, because of its high variance.



Question 4

Comparing ll.fit.lowflex and ll.fit.highflex, ll.fit.highflex which is with lower bandwidth is more variable. It is natural that it gets lower bias, because it is flexible enough to contain exceptional ys. However, for ll.fit.lowflex, its bandwidth is too high to contain excetpional ys. Meanwhile, in the view of variance, low bandwidth is flexible, but has higher variance than high bandwidth. Here, we can see bias-variance tradeoff.

```
bias.low <- mean(predict.low[,2] - trainset[,2])
print(bias.low)  ## bias.low
```

```
## [1] -0.04062661
```

```
bias.high <- mean(predict.high[,2] - trainset[,2])
print(bias.high)  ## bias.high
```

```
## [1] -0.00514693
```

```
variance.low <- var(predict.low[,2])
print(variance.low)  ## variance.low
```

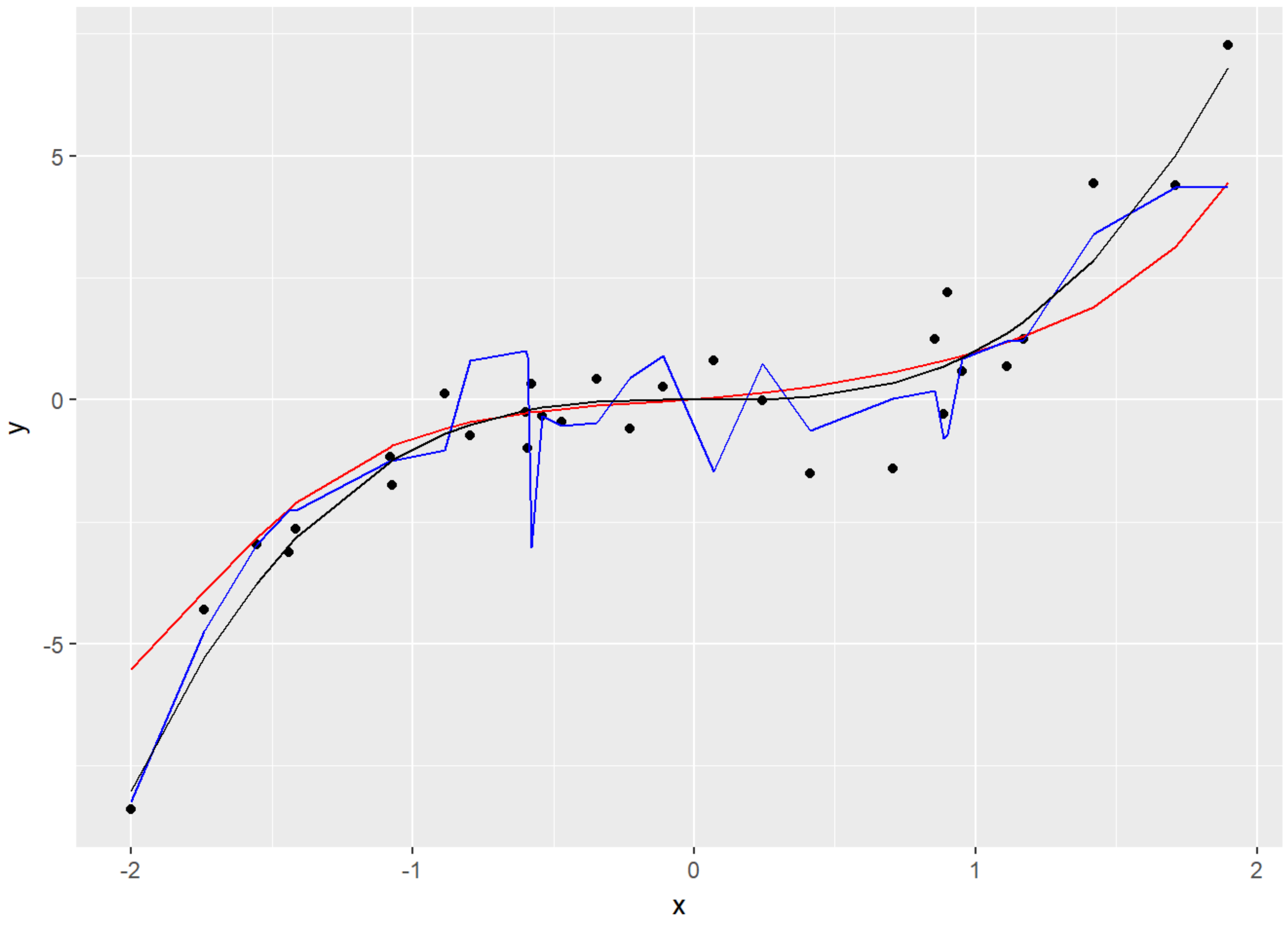
```
## [1] 5.335855
```

```
variance.high <- var(predict.high[,2])
print(variance.high)  ## variance.high
```

```
## [1] 12.32129
```

Question 5

Based on models I made with trainset data, ll.fit.lowflex and ll.fit.highflex, I want to test with my 30 samples in testset. Here, I use predict function. After getting estimated ys of testset, I combine with xs of test set. Here, I want to show 4 figures on the same plot. First, scatter points of x and y in testset. Second, the black like which is showing true relationship between true y and x of testset - of course this will be equation y=x^3. Third, the red line with predicted y or estimated y of using ll.fit.lowflex on testset x. Fourth, the blue line with predicted y or estimated y of using 11.fit.highflex on testset x.



Again, let's see what happens for bias and variance. ll.fit.highflex is still more variable than ll.fit.lowflex. Its variance is higher. The least biased model was ll.fit.highflex. Comparing to former bias of ll.fit.highflex on trainset, now it gets bigger bias (in absolute value).

```
bias.low.test <- mean(predict.low.test[,2] - testset[,2])
print(bias.low.test) ##bias.low.test
```

```
## [1] 0.0746114
```

```
bias.high.test <- mean(predict.high.test[,2] - testset[,2])
print(bias.high.test) ##bias.high.test
```

```
## [1] -0.1556178
```

```
variance.low.test <- var(predict.low.test[,2])
print(variance.low.test) ##variance.low.test
```

```
## [1] 3.616815
```

```
variance.high.test <- var(predict.high.test[,2])
print(variance.high.test) ##variance.high.test
```

```
## [1] 6.196173
```

Question 6

I make bandwidth vector starting from 0.01 to 0.5 with step 0.001. First I make vector from 10 to 500 which is increasing by 1. Then I divide with 1000. It is exactly same. Since, 10/1000=0.01, 500/1000=0.5, 1/1000=0.001

```
bandwidth <- 10:500
bandwidth <- bandwidth/1000
```

Question 7

I want to use loop. First as a starting matrix, I make a vector with length 120. Then I put new column with 120 rows which is predicted y or estimated y with each element in bandwidth vector - so it is bandwidth starting form 0.01 to 0.5. loop keeps putting this new column to starting matrix. When the loop ends, I delete the first column which is irrelevant. I call outcome matrix lltabletrain. Its size is then 120 X 491

```
lltabletrain <- matrix(0,120,1)

for (i in 1:length(bandwidth)) {
  ll.fit <- npreg(y~x, data = trainset, method='ll', bws=bandwidth[i])
  col <- predict(ll.fit, newdata = trainset)
  lltabletrain <- cbind(lltabletrain, col)
}

lltabletrain <- lltabletrain[,-1]
```

Question 8

Again, I want to use loop function. I add new column which is MSE. As I mentioned before, each column of lltabletrain is predicted y with each bandwidth. With each column, I use MSE formulating function with mean function. As a result, I get matrix with 491 MSE coming from each bandwidth.

```r
msetabletrain <- 0

for (i in 1:ncol(lltabletrain)) {
  square.error <- (lltabletrain[,i] - trainset[,2])^2
  t <- mean(square.error)
  msetabletrain <- cbind(msetabletrain, t)
}

msetabletrain <- msetabletrain[,-1]
```

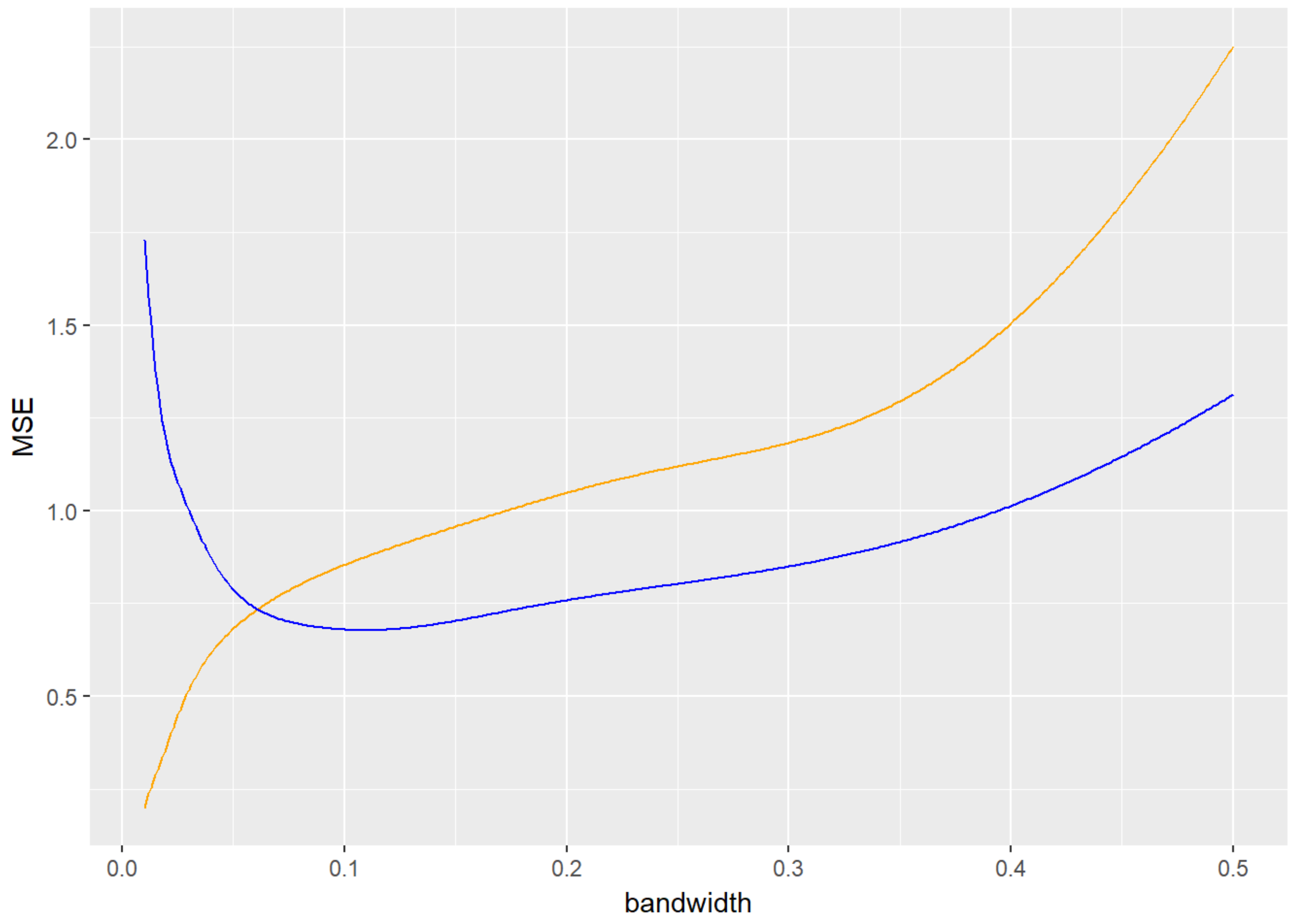
Question 9

Now, I use the same two loop functions for testset. The difference is that predicted or estimated ys are based on x of testset. ll.fit model is made with trainset x and y. We put x of testset as newdata and get predicted y from this model.

```r
lltabletest <- matrix(0,30,1)

for (i in 1:length(bandwidth)) {
  ll.fit <- npreg(y~x, data = trainset, method='ll', bws=bandwidth[i])
  col2 <- predict(object = ll.fit, newdata=testset)
  lltabletest <- cbind(lltabletest, col2)
}

lltabletest <- lltabletest[,-1]


msetabletest <- 0

for (i in 1:ncol(lltabletest)) {
  square.error2 <- (lltabletest[,i]-testset[,2])^2
  d <- mean(square.error2)
  msetabletest <- cbind(msetabletest, d)
}

msetabletest <- msetabletest[,-1]
```

Question 10

I make two tables. First column is common and it is for bandwidth from 0.01 to 0.5. For second column, for MSEtrain, it is MSE from trainset. For MSEtest, it is from testset. On the plot, orange line is showing how MSE of on the trainset is changing according to bandwidth from 0.01 to 0.5. Blue line is showing it of testset.



Indeed, MSE can be decomposed into two parts. MSE = Var + Bias^2. For the trainset MSE line, When bandwidth is small, it has bigger bias but smaller variance. However, as bandwidth increases, bias gets smaller, but variance increases. If the speed of incrasing variance is faster than decreasing bias, MSE, sum of these two, will increase.

However, it does not mean we should choose the bandwidth which is giving the least MSE on trainset. We should also consider what MSE on testset such bandwidth brings. As we can see in the plot, the least MSE on trainset is given when bandwidth 0.01. However, considering MSE on trainset, it does not give lowest MSE. It would be because its bandwidth is too low that bias is too high when it is applied to testset.

So for higher accuracy of our model, we need to compare both MSE on trainset and MSE on testset.

TASK 3B :

Step 1

We import the CNIL DATA thanks to the read.csv2 function :

Step 2

We want to create a table with the number of organizations that has nominated a CNIL per department. First we define a variable Dep with the first two number of the “Code Postal”:

We remove the potential duplicates that we could have on the data :

Then we create the related table:

```
##
##              EXTERNE  INTERNE  MUTUALISE  PROFESSIONNEL  SALARIE
##   1    0       0      21        20            89       4
##   2    0       2      25         8            68       3
##   3    0       0      15        13            42       1
##   4    0       2      13         3            56       0
##   5    0       0       9         5            40       0
##   6    1      22      65        48           122       1
##   7    0       1      13         5            42       0
##   8    0       2      10         4            65       1
##   9    0       0       5         6             9       1
##   10   0       2      19        14            67       1
##   11   0       3      17        13            60       0
##   12   0       0       7         8            73       1
##   13   2      21     100       103           232      10
##   14   4      28      42        29           141      15
##   15   0       0       9         4            41       0
##   16   0       2      26        16            76       3
##   17   1       1      34        20            95       0
##   18   0       1      18        11            53       2
##   19   2       3       9        11            29       0
##   20   0       2      21         1            70       2
##   21   2       3      47        10            81       6
##   22   0       1      29         2            78       4
##   23   0       2       8         4            17       1
##   24   0       3      18        13            49       1
##   25   0       0      35        14            93       3
##   26   0       0      18        14           101       4
##   27   1       1      20         5            87       0
##   28   0       0      16        15            63       2
##   29   0       4      49        25           103       2
##   30   0       4      26        15            90       1
##   31   7      16      96        70           116      12
##   32   0       2      10         5            63       3
##   33   2      29      98        68           168       6
##   34   1      14      76        43           151       6
##   35   0       5      77        39           159       3
##   36   1       0      15         8            29       0
##   37   2       6      38        16           119       5
##   38   4       6      80        54           271       6
##   39   0       1       7         6            55       0
##   40   0      77      20         7            71       2
##   41   1       4      17         8            62       5
##   42   0       2      44        37           131       4
##   43   0       0       4         2            93       3
##   44   1      15      81        62           163      18
##   45   1       3      41        33            87      17
##   46   2       0      12         5            39       2
##   47   1       0      16        13            78       1
##   48   1       1       2         1             7       0
##   49   2       2      38        36           130       5
##   50   1      11      18        10            91       3
##   51   0       1      27        43            98       3
##   52   0       0       8         2            39       2
##   53   0       2      10       219            80       5
##   54   1      10      57        16           121       2
##   55   0       3       9         3            50       0
##   56   0       6      33        16           121       3
##   57   3       8      60        65           107       3
##   58   0       0      11         7            27       0
##   59   7      29     169       107           218      12
##   60   0      54      30        10           117       1
##   61   0       0      13         6            56       0
##   62   0       6      39        22           151       2
##   63   3       3      33        25            73       5
##   64   0       5      38        40            77       2
##   65   0       4      10         3            51       3
##   66   2       4      13        23            67       1
##   67   2      24      73        35           132      15
##   68   4       1      38        24           102       2
##   69   3      23     142       166           239      25
##   70   0       2      10         3            53       2
##   71   0       3      21        11            83       5
##   72   0       1      31        20            68      13
##   73   0       5      21        14            59       3
##   74   0       2      35        26           121       4
##   75  17     168     682       733           415      74
##   76   3       6      75        45           155      15
##   77   2      16      48        31           116      11
##   78   2      14      86        78            99       8
##   79   2       1      19        34            72       7
##   80   2       2      37         9           105       3
##   81   2       4      14         5            91       2
##   82   0       1      13        15            34       2
##   83   0      30      35        31            99       4
##   84   1       9      25        11            81       3
##   85   0       2      23        11           157       3
##   86   0       3      36        28            93       0
##   87   1       7      33        26            46       4
##   88   3       3      15        10            94       2
##   89   0       1      20         3            65       1
##   90   0       0      10         4            10       0
##   91   1      19      60        47            92       5
##   92  10      62     301       408           136      21
##   93   2      16      85       123            71      13
##   94   2      15      86       107            74       7
##   95   3       6      44        55            64       5
##   97   1      11     102        33            94      11
##   98   0       3      18        11             0       0
```

Step 3 :

First we import 10000 rows from the data SIREN

In order to merge the whole database, we create a function that will merge the data by their Siren number, and we repeat it 1000 times to get the best database. Since our computers have no enough memory, we put eval = FALSE.

Then, in order to merge the data, we want that the column have the same name, so we change the name into SIREN for the CNIL file.

So we make a simple merger to have an sample of what we would get with the previous function

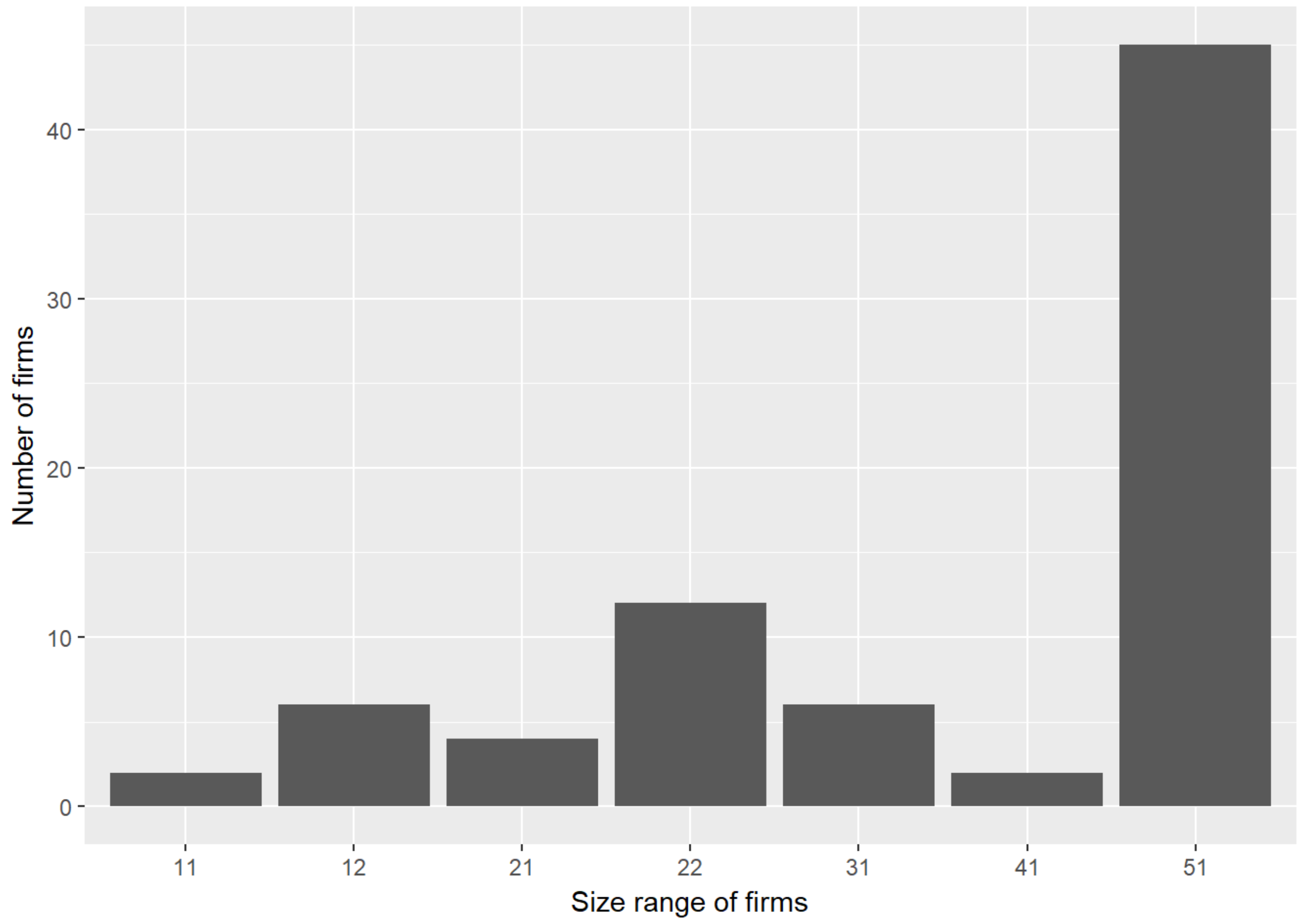
```
Merger <- merge.data.frame(CNIL, SIREN, by = 'SIREN')
```

Step 4

We use the ggplot function in order to create an histogram with the number of firms on the y-axe and the size range as abscissa.

```
ggplot(Merger) +
  geom_histogram(aes(TEFEN), stat="count") +
  ylab("Number of firms") + xlab("Size range of firms")
```

```
## Warning: Ignoring unknown parameters: binwidth, bins, pad
```


Number of firms
Size range of firms