Challenge B

Justine Chollet & William L'Heudé

7 December 2017

Link to our Git hub repo: https://github.com/Justinecholl/ChallengeB

Task 1B

Step 1 - We have choosen random forest as a ML technique. Random forest is a type of aggregation model, in particular it is a case of bootstrap aggregation (bagging) applied to decision trees. The bagging method computes the average of the predictive values of several independant models to reduce the variance and thus the prediction error. Random forest adds to this principle some randomness at the variable level. The idea is to build lots of trees in such a way to make the correlation between the trees smaller. It is a way to decorrelate the several trees which are generated on different bootstrapped samples taken from the train data. And then, it reduces the variance in the trees by averaging them: it helps to reduce the variance and also improve the performance of decision trees on test set and eventualy avoid overfitting.

Step 2 - Choosing and preparing the train data:

```
set.seed(1)
train<-read.csv('train.csv', header=TRUE, sep=",", dec=".", stringsAsFactors=FALSE)
setDT(train)
train$Alley[is.na(train$Alley)]<-'None'
train$MiscFeature[is.na(train$MiscFeature)]<-'None'
train$Fence[is.na(train$Fence)]<-'None'
train$BsmtQual[is.na(train$BsmtQual)]<-'None'
train$BsmtCond[is.na(train$BsmtCond)]<-'None'
train$BsmtExposure[is.na(train$BsmtExposure)]<-'None'
train$BsmtFinType1[is.na(train$BsmtFinType1)]<-'None'
train$BsmtFinType2[is.na(train$BsmtFinType2)]<-'None'
train$FireplaceQu[is.na(train$FireplaceQu)]<-'None'
train$GarageFinish[is.na(train$GarageFinish)]<-'None'
train$GarageQual[is.na(train$GarageQual)]<-'None'
train$GarageCond[is.na(train$GarageCond)]<-'None'
train$GarageType[is.na(train$GarageType)]<-'None'
train$PoolQC[is.na(train$PoolQC)]<-'None'

train$Id<- NULL # Deleting ID (not relevant)
train$LotFrontage<-NULL # Deleting LotFrontage (lots of missing values)
train$GarageYrBlt<-NULL # Deleting GarageYrBlt (lots of missing values)
train<-na.omit(train)  # Creating a new datatable without any rows containing NA
charactervar<-names(train)[which(sapply(train, is.character))]
train[,(charactervar) := lapply(.SD, as.factor), .SDcols = charactervar]
```

Using the random forest technique on the training dataset:

```
train_rf<-randomForest(train$SalePrice~ . , data=train)
```

Step 3 - Choosing and preparing the test data:

```
set.seed(1)
test<-read.csv('test.csv', header=TRUE, sep=",", dec=".", stringsAsFactors=FALSE)
```

```
setDT(test)
test$Alley[is.na(test$Alley)]<-'None'
test$MiscFeature[is.na(test$MiscFeature)]<-'None'
test$Fence[is.na(test$Fence)]<-'None'
test$BsmtQual[is.na(test$BsmtQual)]<-'None'
test$BsmtCond[is.na(test$BsmtCond)]<-'None'
test$BsmtExposure[is.na(test$BsmtExposure)]<-'None'
test$BsmtFinType1[is.na(test$BsmtFinType1)]<-'None'
test$BsmtFinType2[is.na(test$BsmtFinType2)]<-'None'
test$FireplaceQu[is.na(test$FireplaceQu)]<-'None'
test$GarageFinish[is.na(test$GarageFinish)]<-'None'
test$GarageQual[is.na(test$GarageQual)]<-'None'
test$GarageCond[is.na(test$GarageCond)]<-'None'
test$GarageType[is.na(test$GarageType)]<-'None'
test$PoolQC[is.na(test$PoolQC)]<-'None'

test$Id<- NULL # Deleting ID (not relevant)
test$LotFrontage<-NULL # Deleting LotFrontage (lots of missing values)
test$GarageYrBlt<-NULL # Deleting GarageYrBlt (lots of missing values)
test<-na.omit(train)  # Creating a new data.table without any rows containing NA
charactervar<-names(test)[which(sapply(test, is.character))]
test[,(charactervar) := lapply(.SD, as.factor), .SDcols = charactervar]
```

Making predictions on the test data using the random forest of train:

```
test_predictions_rf<-predict(
  train_rf,type="response",predict.all=FALSE,nodes=FALSE,data=test)
```

Making predictions on the test data using a linear regression which includes all the variables except ID and the variables with more than 5% of missing values:

```
test_predictions <- predict(lm(test$SalePrice ~ ., data=test))
```

Getting the summaries of the predicted values using the model created with random forest on the training data and using a linear regression on the testing data which includes all the variables:

```
summary(test_predictions) # Linear regression model
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   21755  128685  163929  180624  214905  661832
```

```
summary(test_predictions_rf) # Random forest model
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   67813  133351  163317  180345  210915  561028
```

The summaries of the predicted values using the model created with random forest on the training data or using a linear regression on the testing data which includes all the variables show quite similar results. For example, the medians of the predictions using the training random forest model and the linear regression are respectively \$163,317 and \$163,929. This result is quite the same for the mean. Nevertheless, the more the predicted values are far from the median, the more different are the predictions between the two models. We can see that for the first and third quantiles. Another noticeable dissemblance is the difference in values between the minimum and the maximum of the two methods (minimum of \$21,755 using the linear regression compared to \$67,813 using random forest). The random forest model seems more appropriate.

Task 2B

Preparing the training and testing data:

```
set.seed(1)
Nsim <- 150
b <- c(0,1)
x0 <- rep(1, Nsim)
x1 <- rnorm(n = Nsim)
X <- cbind(x0, x1^3)
yt <- X %*% b
eps <- rnorm(n = Nsim)
y <- X %*% b + eps

df <- tbl_df(y[,1]) %>% rename(y = value) %>%
  bind_cols(tbl_df(x1)) %>% rename(x = value) %>%
  bind_cols(tbl_df(yt[,1])) %>% rename(yt = value)

training.index <- createDataPartition(y = y, times = 1, p = 0.8)
df <- df %>% mutate(which.data = ifelse(1:n() %in%
  training.index$Resample1, "training", "testing"))

training <- df %>% filter(which.data == "training")
testing <- df %>% filter(which.data == "testing")
lm.fit <- lm(y ~ x, data = training)
df <- df %>% mutate(y.lm = predict(object = lm.fit, newdata = df))
training <- training %>% mutate(y.lm = predict(object = lm.fit))
```

Step 1 - Estimating a low-flexibility local linear model on the training data:

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

```
##
## Regression Data: 122 training points, in 1 variable(s)
##                  x
## Bandwidth(s): 0.5
##
## Kernel Regression Estimator: Local-Constant
## Bandwidth Type: Fixed
## Residual standard error: 1.442574
## R-squared: 0.8569977
##
## Continuous Kernel Type: Second-Order Gaussian
## No. Continuous Explanatory Vars.: 1
```

Step 2 - Estimating a high-flexibility local linear model on the training data:

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

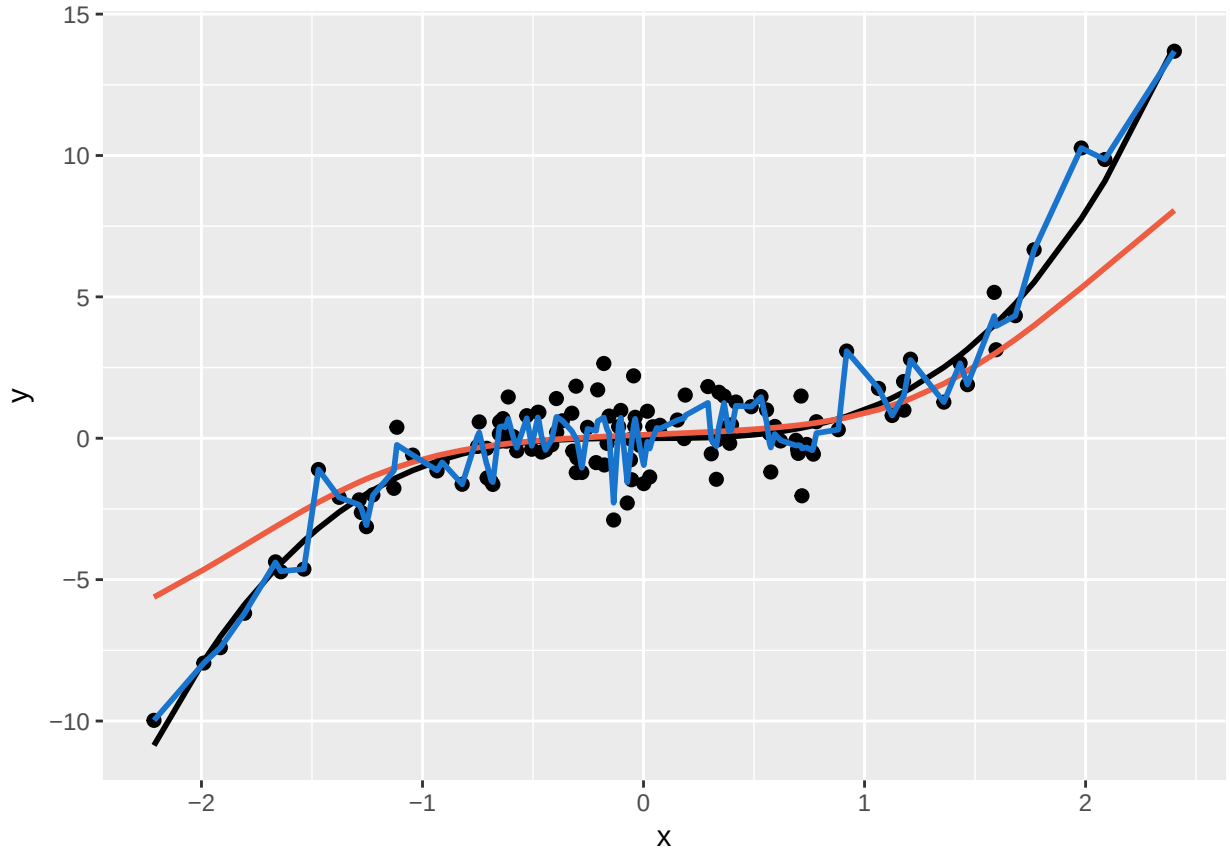
```
##
## Regression Data: 122 training points, in 1 variable(s)
##                   x
## Bandwidth(s): 0.01
##
## Kernel Regression Estimator: Local-Constant
## Bandwidth Type: Fixed
## Residual standard error: 0.5882872
```

```
## R-squared: 0.9569811
##
## Continuous Kernel Type: Second-Order Gaussian
## No. Continuous Explanatory Vars.: 1
```

Step 3 - Plotting the training and testing subsets, and the predictions of the low- and high-flexibility local linear models on the training data:

```
y.lowflex_training <- predict(object=ll.fit.lowflex, newdata=training)
y.highflex_training <- predict(object = ll.fit.highflex, newdata=training)

ggplot(training) + geom_point(data = training, aes(x, y), size=2) +
  geom_line(mapping = aes(x, yt), size=1) +
  geom_line(mapping = aes(x, y.lowflex_training), size=1, col="tomato2") +
  geom_line(mapping = aes(x, y.highflex_training), size=1, col="dodgerblue3")
```



Step 4 - Calculating the variances of the low- and high-flexibility local linear models on the training data:

```
var(y.lowflex_training)
```

```
## [1] 2.764222
```

```
var(y.highflex_training)
```

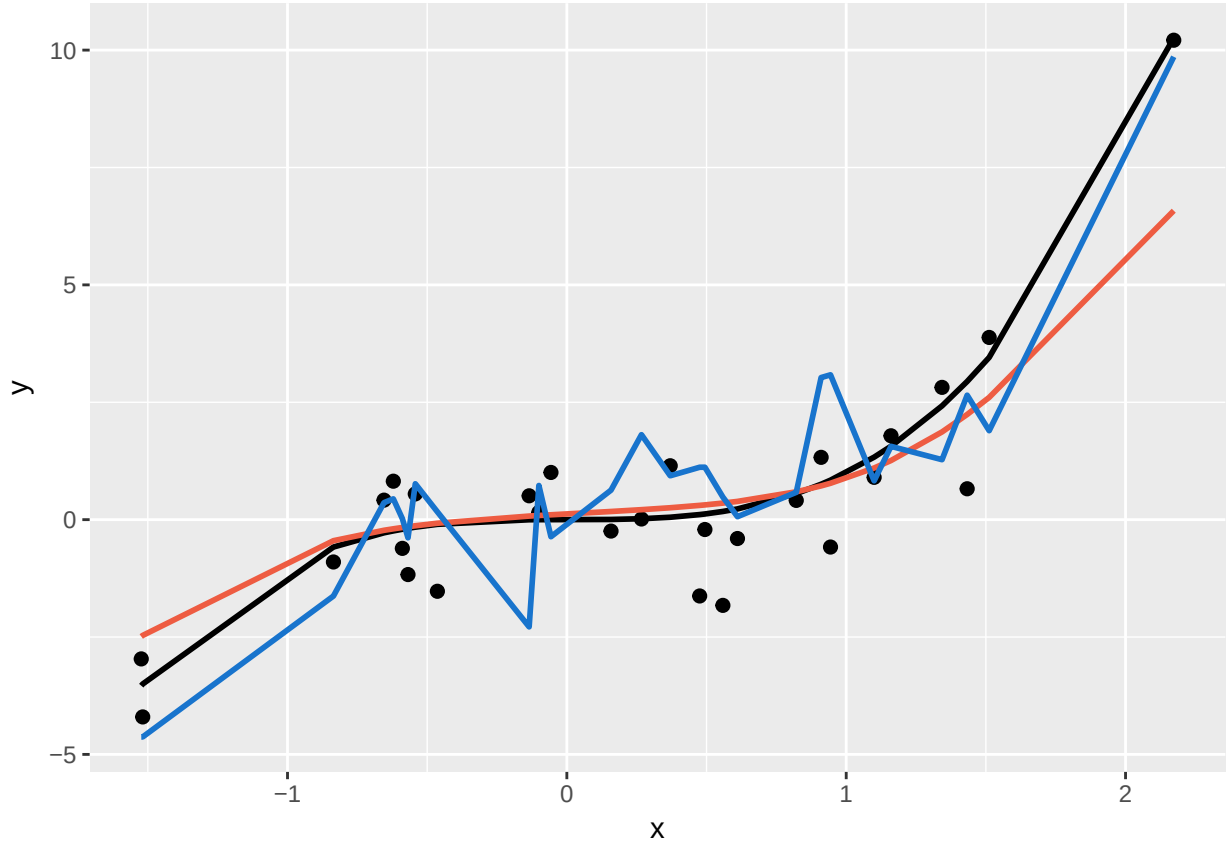
```
## [1] 7.569123
```

The predictions of the highflex model are more variable as the variance of this model is higher. The predictions are less biased with the high flexibility model. Indeed, this high-flexibility linear model is closer to the best predictions line of the true model.

Step 5 - Plotting the testing subset, and the predictions of the low- and high-flexibility local linear models on the testing data:

```
y.lowflex_testing <- predict(object=ll.fit.lowflex, newdata=testing)
y.highflex_testing <- predict(object = ll.fit.highflex, newdata = testing)

ggplot(testing) + geom_point(data = testing, aes(x, y), size=2) +
  geom_line(mapping = aes(x, yt), size=1) +
  geom_line(mapping = aes(x, y.lowflex_testing), size=1, col="tomato2") +
  geom_line(mapping = aes(x, y.highflex_testing), size=1, col="dodgerblue3")
```



Calculating the variances of the low- and high-flexibility local linear models on the testing data:

```
var(y.lowflex_testing)
```

```
## [1] 2.572402
```

```
var(y.highflex_testing)
```

```
## [1] 6.582451
```

The predictions of the highflex model are more variable as the variance of this model is higher. The bias of the high flexibility model is higher using the testing data as there are less points to estimate the model than using the training data, so here the high flexibility model fits less the true model.

Step 6 - Creating a vector of bandwidth going from 0.01 to 0.5 with a step of 0.001:

```
bandwidth <- seq(0.01, 0.5, by=0.001)
```

The bandwidth determines the flexibility of a model: the higher the bandwidth, the less flexible the model.

Step 7 - Estimating a local linear model $y \sim x$ on the training fata with each bandwidth:

```
ll.fit.bandwidth <- lapply(X=bandwidth, FUN=function(bandwidth)
  {npreg(y~x, bws=bandwidth, data=training, method="ll")})
```

Step 8 - Computing for each bandwidth the MSE on the training data:

```
mse_training <- function(fit){predictions<-predict(object=fit, newdata=training)
training %>% mutate(squared.error=(y-predictions)^2) %>%
  summarize(mse=mean(squared.error))}
mse_training_table <- unlist(lapply(X=ll.fit.bandwidth, FUN=mse_training))
```

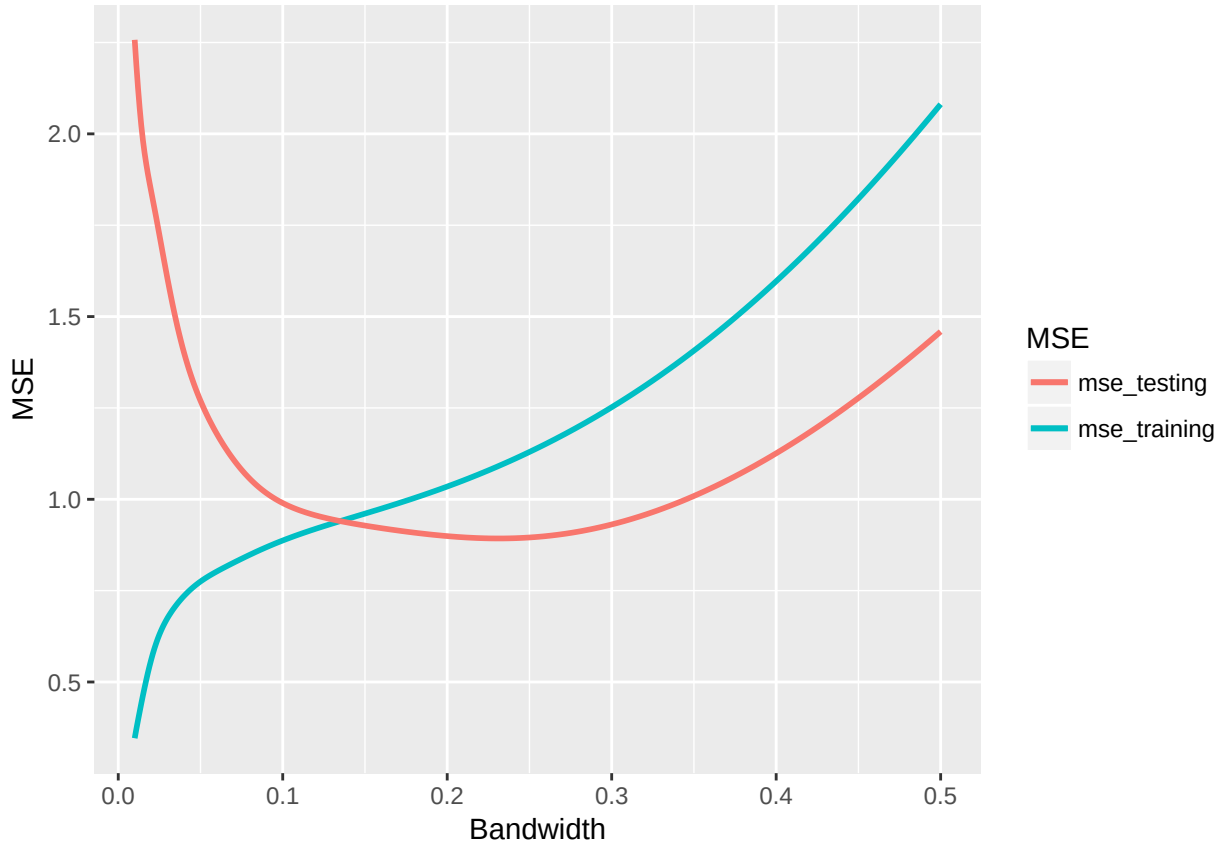
Step 9 - Computing for each bandwidth the MSE on the testing data:

```
mse_testing <- function(fit){predictions<-predict(object=fit, newdata=testing)
testing %>% mutate(squared.error=(y-predictions)^2) %>%
  summarize(mse=mean(squared.error))}
mse_testing_table <- unlist(lapply(X=ll.fit.bandwidth, FUN=mse_testing))
```

Step 10 - Plotting the MSE against the bandwidth:

```
mse_table <- tbl_df(data.frame(
  bandwidth=bandwidth, mse_training=mse_training_table, mse_testing=mse_testing_table))

ggplot(mse_table, aes(x = Bandwidth, y = MSE)) +
  geom_line(aes(x = bandwidth, y = mse_training, color = "mse_training"), size=1) +
  geom_line(aes(x = bandwidth, y = mse_testing, color = "mse_testing"), size=1) +
  labs(colour="MSE")
```



The MSE is a measure of the bias and the variance. The above figure shows that, as seen before, using the training data (which contains 80% of the observations), the high flexibility model (with the lower bandwidth) fits more the true model than the low flexibility model (with the higher bandwidth). Nevertheless, as regard to the testing data (which contains only 20% of the observations), the high flexibility model is quite far away from the true model as there are not enough observations to well represent the true model. This result is the same using a low flexibility model: indeed, the model is too smooth to fit well the true model as, again, the observations do not represent well the true model. Allowing a medium flexibility (a bandwidth of around 0.25) permits to reduce the MSE such that the model will fit better the true model than a high or a low flexibility.

To conclude, we should use a high flexibility model if we have a lot of observations and a medium flexibility model if we have few observations in order to fit better the model.

Task 3B

Step 1 - Choosing and importing the CNIL dataset:

```
cnil<-read.csv('OpenCNIL_Organismes_avec_CIL_VD_20171115.csv',
               header=TRUE, sep=";", dec=",", stringsAsFactors=FALSE)
```

Step 2 - Creating a table with the number of organizations that has nominated a CNIL per department:

```
colnames(cnil) # Identyfing the different columns of the CNIL dataset

cnil2 <- subset(cnil, nchar(cnil$Code_Postal) > 4,)
# Selecting the postal codes longer than 4 and placing them into cnil2

cnil3 <- subset(cnil2, nchar(cnil2$Code_Postal) < 6,)
# Among cnil2, selecting the postal codes smaller than 6
# Now, cnil3 contains only the correct postal codes with 5 numbers

cp <- sub ("^(\\d{2}).*$","\\1",cnil3$Code_Postal); cp2 <- subset(cp, nchar(cp) < 3,)
# Creating a dataframe which contains only the 2 first numbers of the postal codes

# Creating the table containing the number of organizations per department:
nicetable <- data.frame(table(unlist(cp2)))
colnames(nicetable)
colnames(nicetable)[colnames(nicetable)=="Var1"] <- "Departement"
colnames(nicetable)[colnames(nicetable)=="Freq"] <- "Number of organizations"
```

Here is our nice table:

```
kable(nicetable)
```

Departement	Number of organizations
01	134
02	105
03	71
04	74
05	54
06	256
07	61
08	82
09	21
10	102

Departement	Number of organizations
11	93
12	89
13	468
14	259
15	53
16	123
17	151
18	84
19	54
20	96
21	149
22	114
23	32
24	84
25	145
26	137
27	113
28	96
29	181
30	136
31	314
32	83
33	366
34	290
35	281
36	53
37	186
38	421
39	69
40	177
41	96
42	218
43	102
44	339
45	181
46	60
47	109
48	12
49	212
50	134
51	172
52	51
53	316
54	207
55	64
56	179
57	245
58	45
59	542
60	212
61	75
62	219

Departement	Number of organizations
63	142
64	162
65	71
66	110
67	280
68	170
69	598
70	70
71	123
72	133
73	103
74	188
75	2074
76	298
77	224
78	286
79	134
80	156
81	118
82	64
83	199
84	130
85	196
86	158
87	116
88	126
89	89
90	24
91	220
92	934
93	308
94	289
95	177
97	252
98	32

Step 3 - Importing the SIRENE database in one shot is not possible due to its size. We have decided to import 50,000 rows at a time and repeat the operation until the all dataset is loaded. This method takes roughly 13 minutes. Then, we put in the final SIRENE database only the companies that have nominated a CIL according to their SIREN number (cf. CNIL dataset):

```
start<-Sys.time()
sirene <- 'sirc-17804_9075_14209_201710_L_M_20171101_030132835.csv'
chunksize<-50000
connector<-file(description = sirene, open="r")
index<-0
initiator <- read.table(connector, nrows=2, skip=0, header=TRUE, fill=TRUE, sep=";")
columns <- names(initiator)
sirene.final <- data.frame(matrix(ncol=length(columns), nrow = 0))
colnames(sirene.final) <- columns

colnames(cnil)[colnames(cnil)=="ï..Siren"] <- "SIREN"; cnil.final <- c(cnil$SIREN)
```

```
repeat{index <- index+1
print(paste('Processing rows:', index*chunksize))
sirene.chunck <- read.table(connector, nrows = chunksize, skip=0, header = FALSE,
      fill=TRUE, sep=";", col.names=columns, stringsAsFactors = FALSE)
sirene.chunck <- sirene.chunck[order(sirene.chunck[,'DATEMAJ'],decreasing = TRUE),]
sirene.chunck.reduced1 <- sirene.chunck[which(!duplicated(sirene.chunck$SIREN)),]
sirene.chunck.reduced2 <- sirene.chunck[sirene.chunck$SIREN %in% cnil.final,]
sirene.final <- rbind(sirene.final,sirene.chunck.reduced2)
if (nrow(sirene.chunck)!= chunksize){
  print('Finished.')
  break}}
close(connector)
```

```
end<-Sys.time()
end-start
```

```
## Time difference of 15.45461 mins
```

Now, we can merge the data from the final SIRENE database into the CNIL database by SIREN number to keep the information only about the companies that have nominated a CIL:

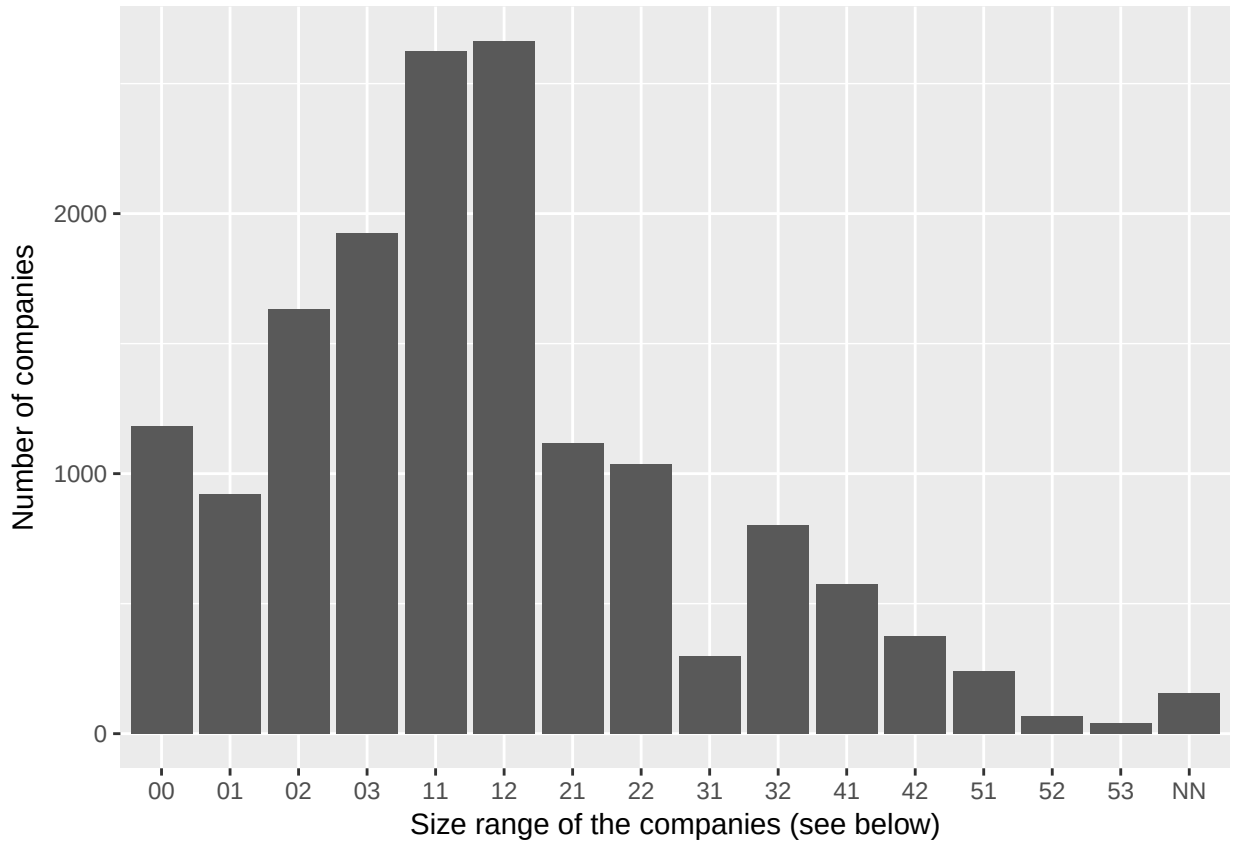
```
colnames(sirene.final)
colnames(cnil)

merged<-left_join(cnil,sirene.final,by="SIREN")
merged2<-setDT(merged)[,.SD[which.max(DAPET)],keyby=SIREN]
```

Our merged2 dataframe only contains the most up-to-date information about each of these companies.

Step 4 - Then, we plot the histogram of the size of these companies:

```
ggplot(merged2) +
  geom_histogram(aes(TEFEN), stat="count") +
  ylab("Number of companies") + xlab("Size range of the companies (see below)")
```



Size ranges:
NN: No employees during the base year and at the 31 December
00: 0 employee at 31 December but some during the year
01: 1 or 2 employees
02: 3 to 5 employees
03: 6 to 9 employees
11: 10 to 19 employees
12: 20 to 49 employees
21: 50 to 99 employees
22: 100 to 199 employees
31: 200 to 249 employees
32: 250 to 499 employees
41: 500 to 999 employees
42: 1 000 to 1 999 employees
51: 2 000 to 4 999 employees
52: 5 000 to 9 999 employees
53: 10 000 employees and more