R Programming — Challenge B

August Aubach Altès and Víctor Quintas Martínez

December 8, 2017

Task 1B

Step 1

We are going to use a "Random Forest" regression to predict House Prices in Arizona.

What is a "Random Forest" regression?

"Random Forest" regression is a machine learning technique broadly based on "Decision Tree" learning.

In "Decision Tree" learning, we try to predict the target variable by recursively splitting the training set according to the value of the features. Starting from an initial node, training observations will be splitted in 2 *branches* based on the value of one feature. Each branch leads to a new node, and the subset of the training data that correspond to that node will be again splitted into 2 branches based on the value of the same or another feature. The process is iterated and stops after a number of subsequent splits, when adding new partitions would not add much information. The last nodes of the tree are called the *leaves*.

In a classification problem, one good moment to stop would be when the values of the target value are the same at each leaf. For example, the following is a "Decision Tree" trying to classify whether the characters in Titanic survived (taken from Wikipedia).

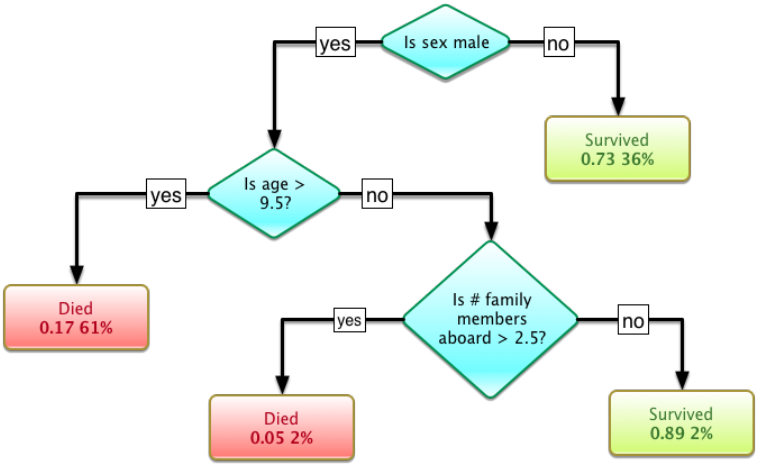


Figure 1: Titanic Decision Tree

For our case, since it is a regression and not a classification problem, it is a bit more complicated. The idea is, at each leaf, to assign the value of the target variable y that is "closer" to all observations in that leaf, in the sense that it minimizes the squared error. That is to say, if L denotes the subset of observations in one leaf, after the classification we will assign the value of the target

$$\hat{y}_L = \arg\min_y \sum_{i \in L} (y - y_i)^2.$$

It is easy to check that the solution to this problem is the average of y among the observations in the leaf.

Within the "Decision Tree" regression procedures, there is one ("Bagging") that consists in bootstrap-selecting subsamples of the training data, and training regression trees on each of this subsamples, that are then aggregated by averaging. Bootstraping means that the subsamples are selected at random and with replacement.

"Random Forest" regression is, in a sense, an improvement of this "Bagging" technique, aimed at reducing overfitting of the models (especially when you have a large number of features from which to choose). The idea is to randomize not only the subsample on which you are going to train your "Decision Trees", but also the features that you are going to use.

The corresponding package in R is called `randomForest`. We install it and load it:

```
install.packages("randomForest", repos = "http://cran.us.r-project.org")
library(randomForest)
```

Step 2

We load the data with `tidyverse` (don't forget to put the .csv files in your working directory):

```
install.packages("tydiverse", repos = "http://cran.us.r-project.org")
library(tidyverse)

train <- read_csv(file = "train.csv")
test <- read_csv(file = "test.csv")
```

And now we train our "Random Forest" (we take out Id, variables with too many NAs and observations with some NA). Notice that we also had to convert the character variables into factors and to change the names of the variables so that they respect R syntax (otherwise `randomForest` wouldn't run):

```
apply(is.na(train), 2, sum)
```

```
##           Id   MSSubClass     MSZoning  LotFrontage      LotArea
##            0            0            0          259            0
##       Street        Alley     LotShape  LandContour    Utilities
##            0         1369            0            0            0
##    LotConfig    LandSlope Neighborhood   Condition1   Condition2
##            0            0            0            0            0
##     BldgType   HouseStyle  OverallQual  OverallCond    YearBuilt
##            0            0            0            0            0
## YearRemodAdd    RoofStyle      RoofMatl  Exterior1st  Exterior2nd
##            0            0            0            0            0
##   MasVnrType   MasVnrArea    ExterQual    ExterCond   Foundation
##            8            8            0            0            0
##     BsmtQual     BsmtCond BsmtExposure BsmtFinType1   BsmtFinSF1
##           37           37           38           37            0
## BsmtFinType2   BsmtFinSF2   BsmtUnfSF  TotalBsmtSF      Heating
##           38            0            0            0            0
##    HeatingQC   CentralAir   Electrical     1stFlrSF     2ndFlrSF
##            0            0            1            0            0
## LowQualFinSF    GrLivArea BsmtFullBath BsmtHalfBath     FullBath
##            0            0            0            0            0
##     HalfBath BedroomAbvGr KitchenAbvGr  KitchenQual TotRmsAbvGrd
##            0            0            0            0            0
##   Functional   Fireplaces  FireplaceQu   GarageType  GarageYrBlt
##            0            0          690           81           81
## GarageFinish   GarageCars   GarageArea   GarageQual   GarageCond
```

```
##           81            0            0           81           81
##   PavedDrive   WoodDeckSF  OpenPorchSF EnclosedPorch    3SsnPorch
##            0            0            0            0            0
##  ScreenPorch     PoolArea       PoolQC        Fence  MiscFeature
##            0            0         1453         1179         1406
##      MiscVal       MoSold       YrSold     SaleType SaleCondition
##            0            0            0            0            0
##    SalePrice
##            0
```

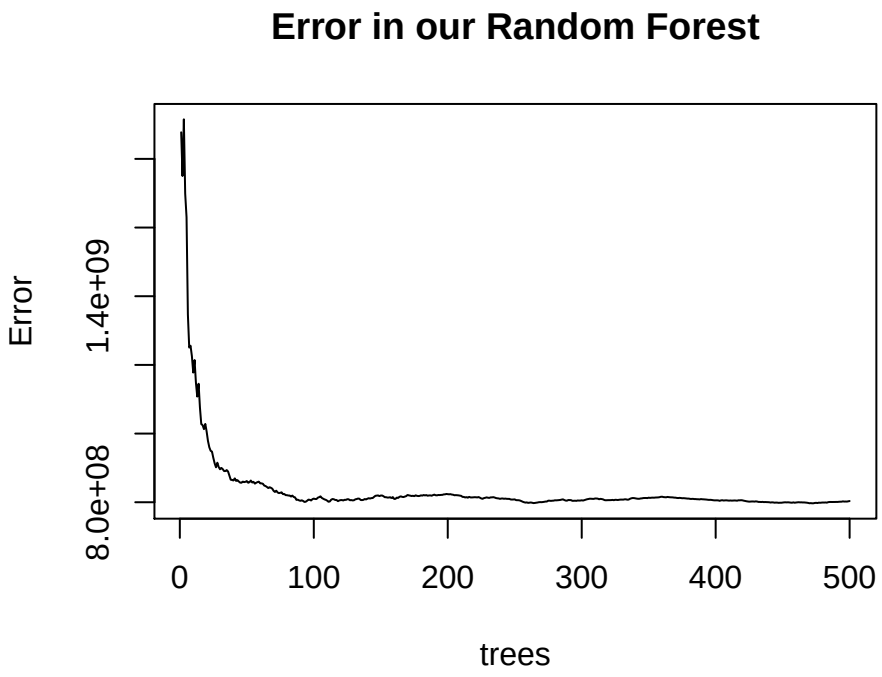
```
variables <- train %>% select(-c(Id, LotFrontage, Alley, FireplaceQu, PoolQC,
    Fence, MiscFeature))
variables <- variables %>% filter(is.na(MasVnrType) == FALSE, is.na(MasVnrArea) ==
    FALSE, is.na(BsmtQual) == FALSE, is.na(BsmtCond) == FALSE, is.na(BsmtFinType1) ==
    FALSE, is.na(BsmtFinType2) == FALSE, is.na(BsmtExposure) == FALSE, is.na(GarageType) ==
    FALSE, is.na(GarageYrBlt) == FALSE, is.na(GarageFinish) == FALSE, is.na(GarageQual) ==
    FALSE, is.na(GarageCond) == FALSE, is.na(Electrical) == FALSE)
sum(is.na(variables))
```

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

```
variables <- variables %>% mutate_if(is.character, as.factor)
names(variables) = make.names(names(variables))
train_rf <- randomForest(SalePrice ~ ., data = variables, importance = FALSE)
```

We are going to provide three simple visualizations for our estimated "Random Forest". The first one shows the error rates (MSE in the leaves) after the regression as a function of the number of trees.

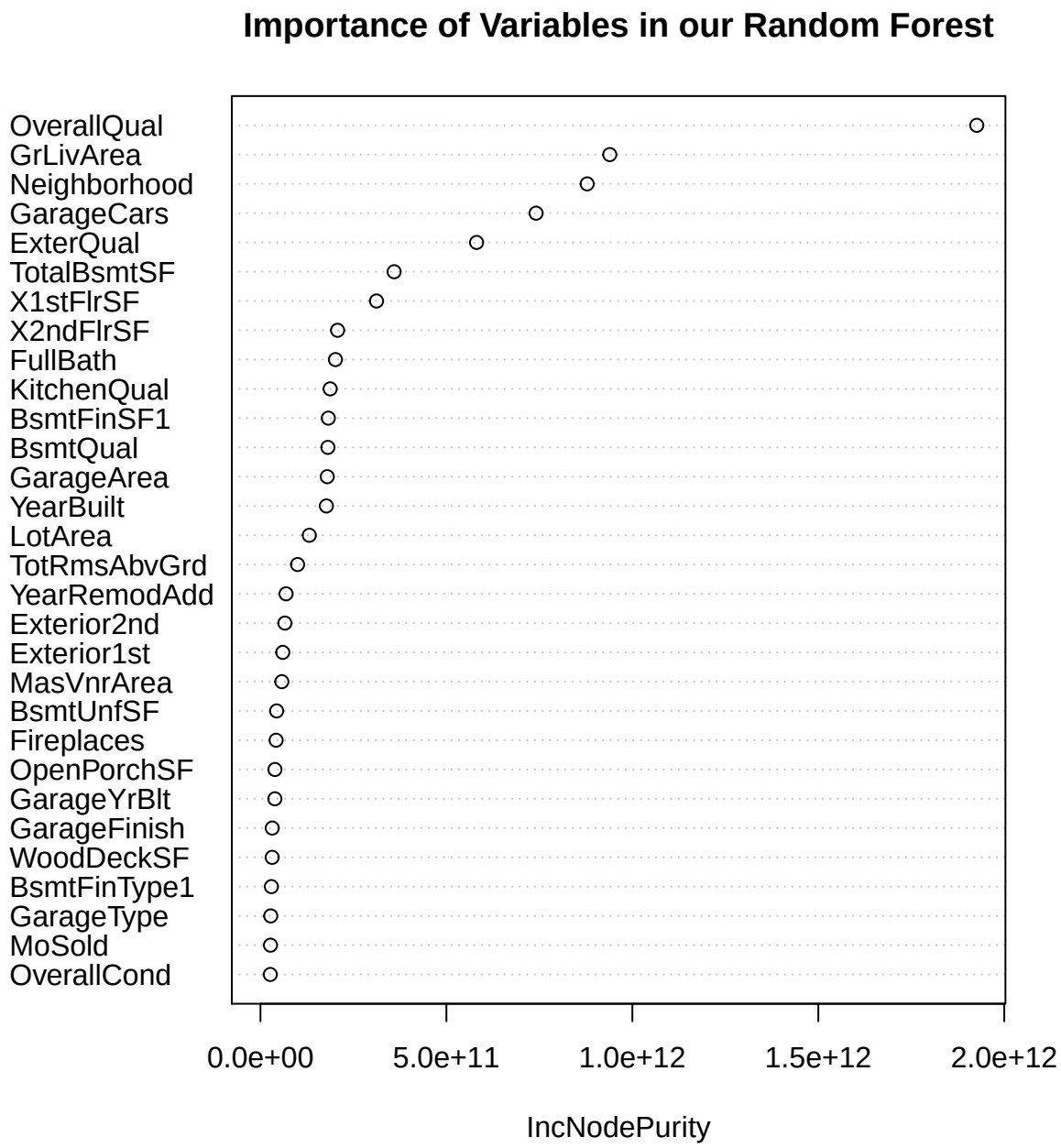
```
plot(train_rf, main = "Error in our Random Forest")
```



The second one consists of a plot of variable importance. In this case, `IncNodePurity` is a measure of variable importance that measures, at each split, how much the split reduces impurity of nodes (impurity is defined as

the difference between the RSS in the node before and after the splits).

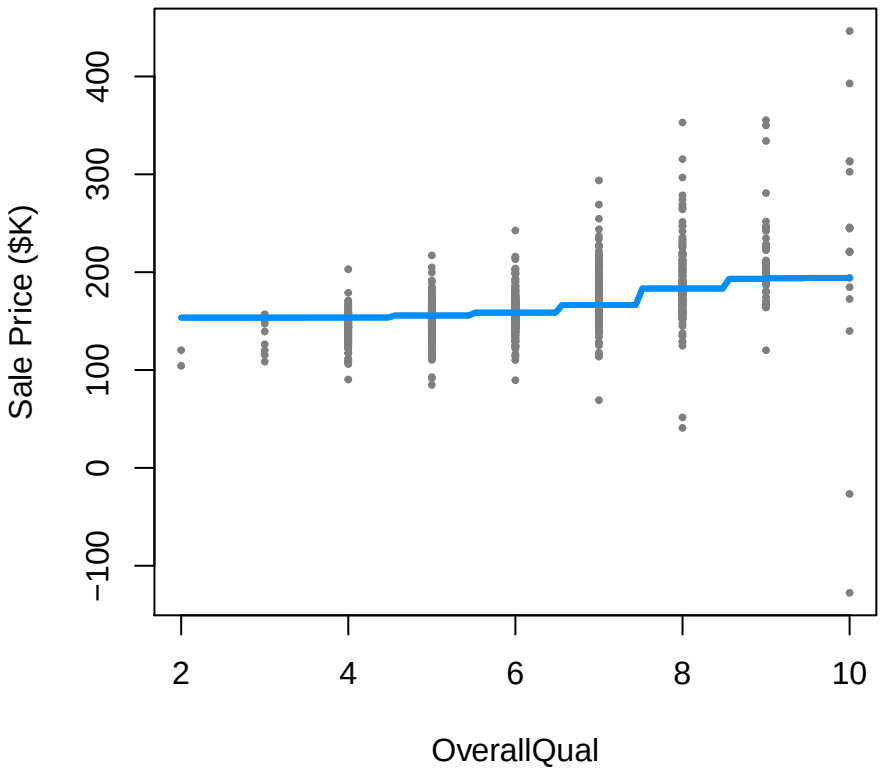
```
varImpPlot(train_rf, main = "Importance of Variables in our Random Forest")
```



For the third one, we are actually going to see the predictions that our Random Forest makes with respect to the first variable ordered by importance (OverallQual). For that we need to use the package `visreg`, that we install and load:

```
install.packages("visreg", repos = "http://cran.us.r-project.org")
library(visreg)
```

```
visreg(train_rf, "OverallQual", ylab = "Sale Price ($K)", yaxt = "n")
axis(2, at = 1e+05 * (-1):4, labels = c("-100", "0", "100", "200", "300", "400"))
```



Step 3

As a linear prediction of our choice, we are going to use the same model that we ended up selecting in Challenge A. We compute the predictions (remember that the test data was already loaded before):

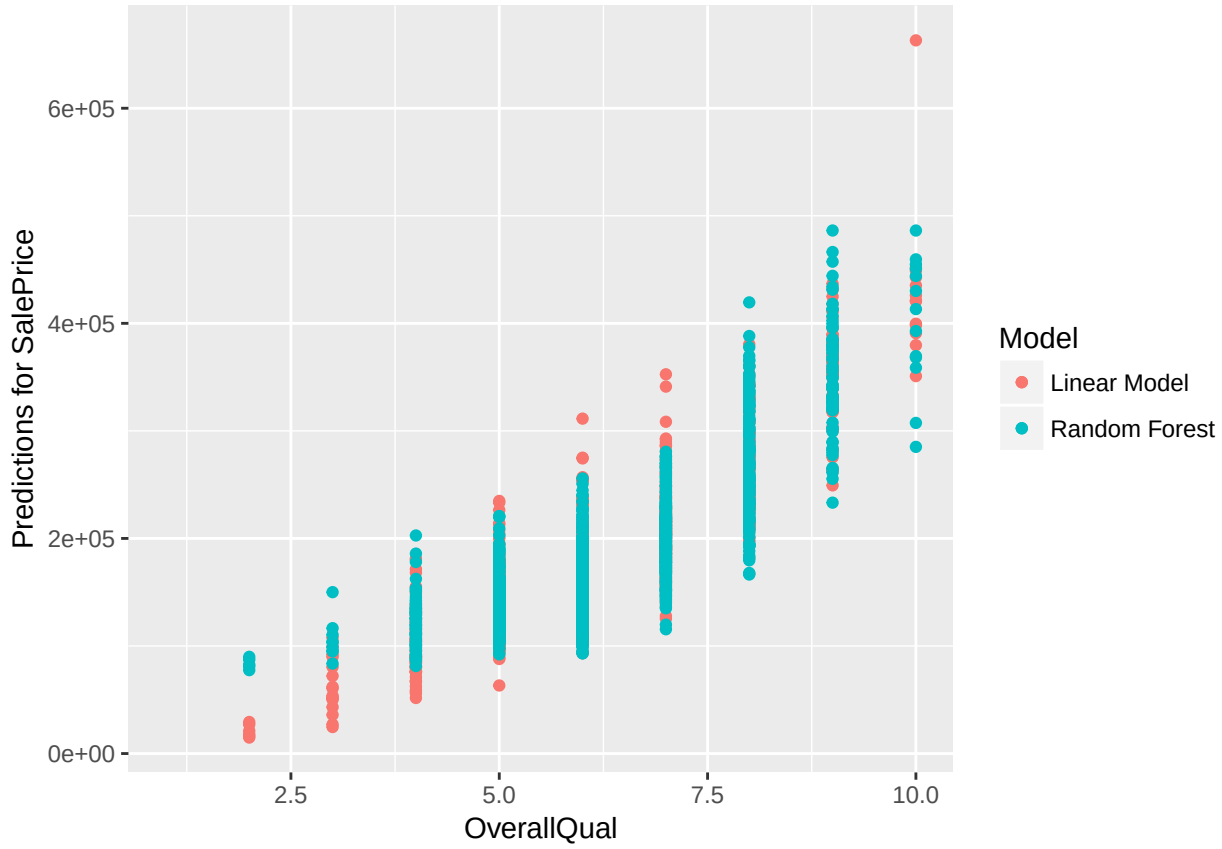
```
names(test) <- make.names(names(test))
test <- test %>% mutate_if(is.character, as.factor)
test <- test %>% select(-c(Id, LotFrontage, Alley, FireplaceQu, PoolQC, Fence,
    MiscFeature))
test_variables <- test %>% select(c(LotArea, OverallQual, OverallCond, RoofMatl,
    ExterQual, BsmtQual, X1stFlrSF, X2ndFlrSF, KitchenQual, PoolArea))
lin_variables <- variables %>% select(c(LotArea, OverallQual, OverallCond, RoofMatl,
    ExterQual, BsmtQual, X1stFlrSF, X2ndFlrSF, KitchenQual, PoolArea, SalePrice))
train_lin <- lm(SalePrice ~ ., data = lin_variables)
lin_predictions <- predict(train_lin, test_variables)
```

When trying to make the "Random Forest" predictions, we got an error that for some factor variable, there were levels in the test set that did not appear in the training set. To try and fix that, we are going to merge the train and test sets in just one dataset to reharmonize the values of the factor variables:

```r
# Combine
data <- bind_rows(variables, test)

# Reconvert to factors
data <- data %>% mutate_if(is.character, as.factor)

# De-combine
variables <- data %>% slice(1:1338)
test <- data %>% slice(1339:2797)

# Retrain the model
train_rf <- randomForest(SalePrice ~ ., data = variables)

# Predict
rf_predictions <- predict(train_rf, test)
```

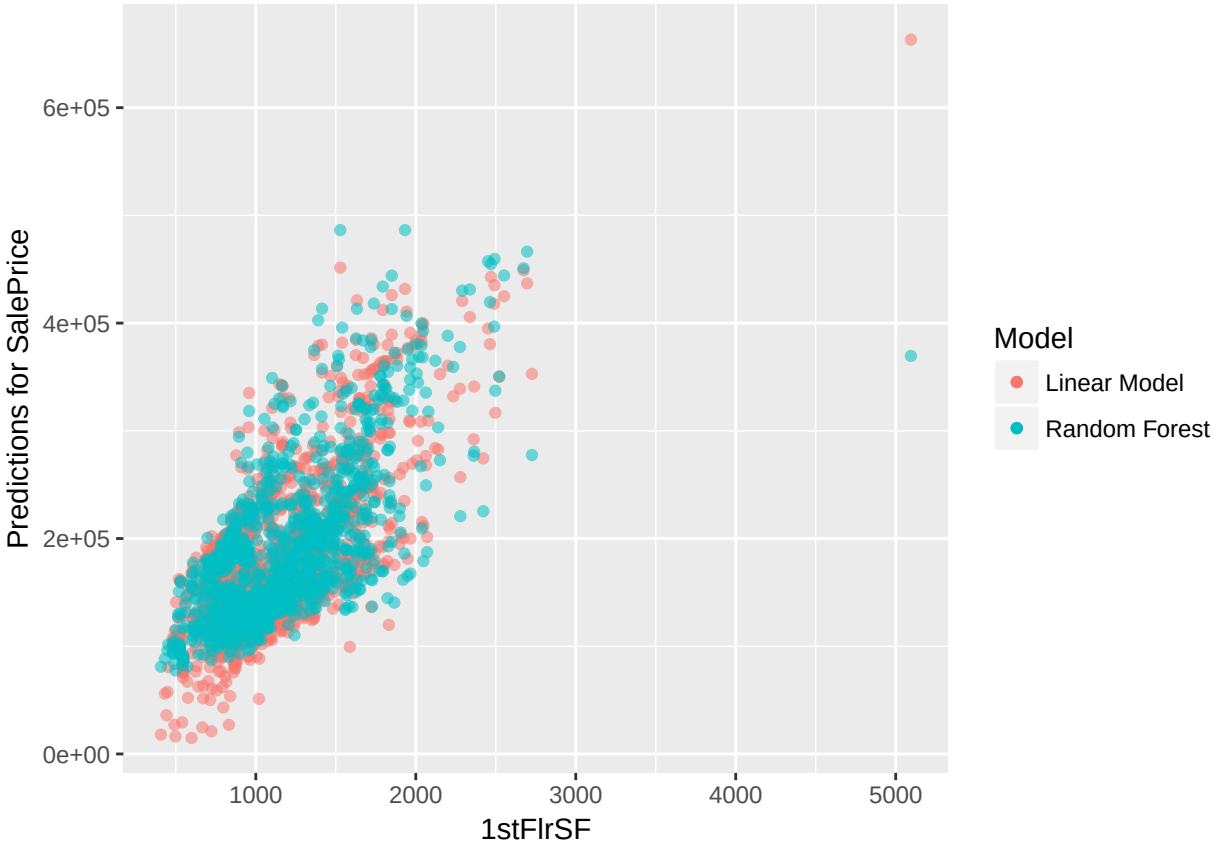
We can visualize the results in a plot (we plot the predictions as a function of our most "important" variable):

```r
install.packages("ggplot2", repos = "http://cran.us.r-project.org")
library(ggplot2)
ggplot(mapping = aes(x = test$OverallQual)) + geom_point(mapping = aes(y = lin_predictions,
    col = "Linear Model")) + geom_point(mapping = aes(y = rf_predictions, col = "Random Forest")) +
    labs(x = "OverallQual", y = "Predictions for SalePrice", colour = "Model")
```



Maybe we will see more if we plot as a function of a continuous variable:

```
ggplot(mapping = aes(x = test$X1stFlrSF)) + geom_point(mapping = aes(y = lin_predictions,
    col = "Linear Model", alpha = 0.1)) + geom_point(mapping = aes(y = rf_predictions,
    col = "Random Forest", alpha = 0.1)) + labs(x = "1stFlrSF", y = "Predictions for SalePrice",
    colour = "Model") + guides(alpha = FALSE)
```



As a general comment, we see that the linear predictions display a bit more of variation and more outliers than the ones coming from the "Random Forest".

Task 2B

Step 1

To begin with, we generate the data the same way that we did in Challenge A:

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

Then we split it into training and test datasets:

```
install.packages("caret", repos = "http://cran.us.r-project.org")
library(caret)
index <- createDataPartition(y, p = 0.8, list = F)
```

```
training <- data %>% slice(index)
test <- data %>% slice(-index)
```

We install and the np package for the local linear models:

```
install.packages("np", repos = "http://cran.us.r-project.org")
library(np)
```

And now we estimate our model:

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

```
##
## Regression Data: 122 training points, in 1 variable(s)
##                  x
## Bandwidth(s): 0.5
##
## Kernel Regression Estimator: Local-Linear
## Bandwidth Type: Fixed
## Residual standard error: 1.085438
## R-squared: 0.8540956
##
## Continuous Kernel Type: Second-Order Gaussian
## No. Continuous Explanatory Vars.: 1
```

Step 2

We estimate the new model in (almost) the same way:

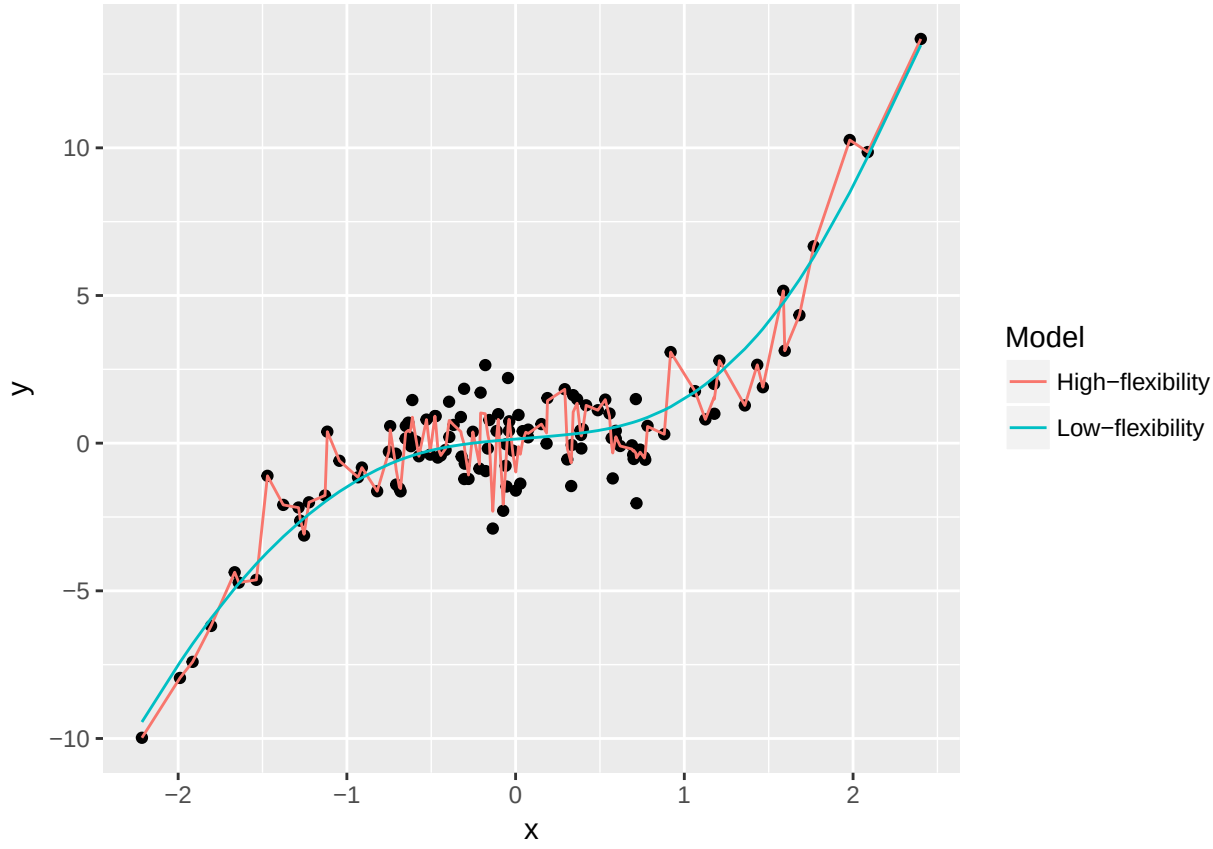
```
ll.fit.highflex <- npreg(y ~ x, bws = 0.01, data = training, regtype = "ll")
summary(ll.fit.highflex)
```

```
##
## Regression Data: 122 training points, in 1 variable(s)
##                   x
## Bandwidth(s): 0.01
##
## Kernel Regression Estimator: Local-Linear
## Bandwidth Type: Fixed
## Residual standard error: 0.5070779
## R-squared: 0.9680171
##
## Continuous Kernel Type: Second-Order Gaussian
## No. Continuous Explanatory Vars.: 1
```

Step 3

```
yhatlowflex <- predict(ll.fit.lowflex, training, se.fit = TRUE)
yhathighflex <- predict(ll.fit.highflex, training, se.fit = TRUE)
predictions <- tibble(training$x, yhatlowflex$fit, yhathighflex$fit)
names(predictions) <- c("x", "Low-flexibility", "High-flexibility")
predictionslong <- predictions %>% gather(key = "Model", value = "yhat", "Low-flexibility",
    "High-flexibility")
```

```
ggplot(data = training, aes(x, y)) + geom_point(aes(x, y)) + geom_line(data = predictionslong,
    aes(x, yhat, colour = Model))
```



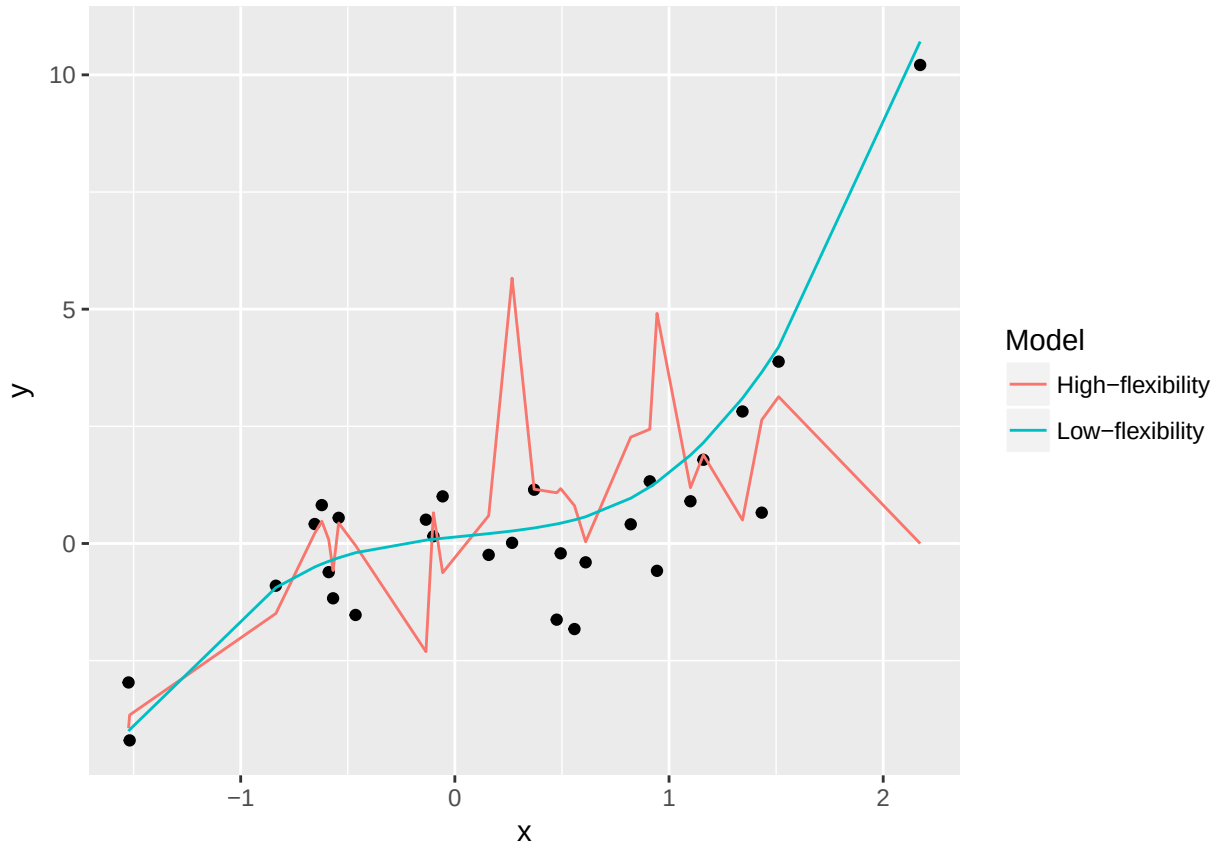
Step 4

It is easy to see in the graph that the High-flexibility predictions clearly overfit the data, in contrast to the Low-flexibility model. So, the High-flexibility has **smaller bias** but **more variance** than the Low-flexibility one.

Step 5

We do it the same way:

```
yhatlowflextest <- predict(ll.fit.lowflex, newdata = test, se.fit = TRUE)
yhathighflextest <- predict(ll.fit.highflex, newdata = test, se.fit = TRUE)
predictionstest <- tibble(test$x, yhatlowflextest$fit, yhathighflextest$fit)
names(predictionstest) <- c("x", "Low-flexibility", "High-flexibility")
predictionslongtest <- predictionstest %>% gather(key = "Model", value = "yhat",
    "Low-flexibility", "High-flexibility")
ggplot(data = test, aes(x, y)) + geom_point(aes(x, y)) + geom_line(data = predictionslongtest,
    aes(x, yhat, colour = Model))
```



We clearly see that, again, High-flexibility predictions are more variable, but also they are much more biased now. In fact, we can compare the MSE for both:

```
MSElf <- mean((test$y - yhatlowflextest$fit)^2)
MSEhf <- mean((test$y - yhathighflextest$fit)^2)
c(MSElf, MSEhf)
```

```
## [1] 1.217407 7.630138
```

We see that the second one is much bigger than the first.

Step 6

We construct a sequence of numbers from 0.01 to 0.5, with a step of 0.001, that are the different bandwidths of our models.

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

Steps 7, 8 and 9

Let us carry out the next three steps together. We perform 7, 8 and 9 regressing each model, doing the predictions and then saving the mean square error in a dataframe. Let us explain what we have done step by step. First off, we estimate a local linear model y ~ x on the training data with each of our bandwidth. Then, we compute for each bandwidth the MSE on the test dataset, for that we use the function `fitted.values`, which is a generic function that extracts fitted values from objects returned by modeling functions. Similarly, we compute for each bandwidth the MSE on the training data.

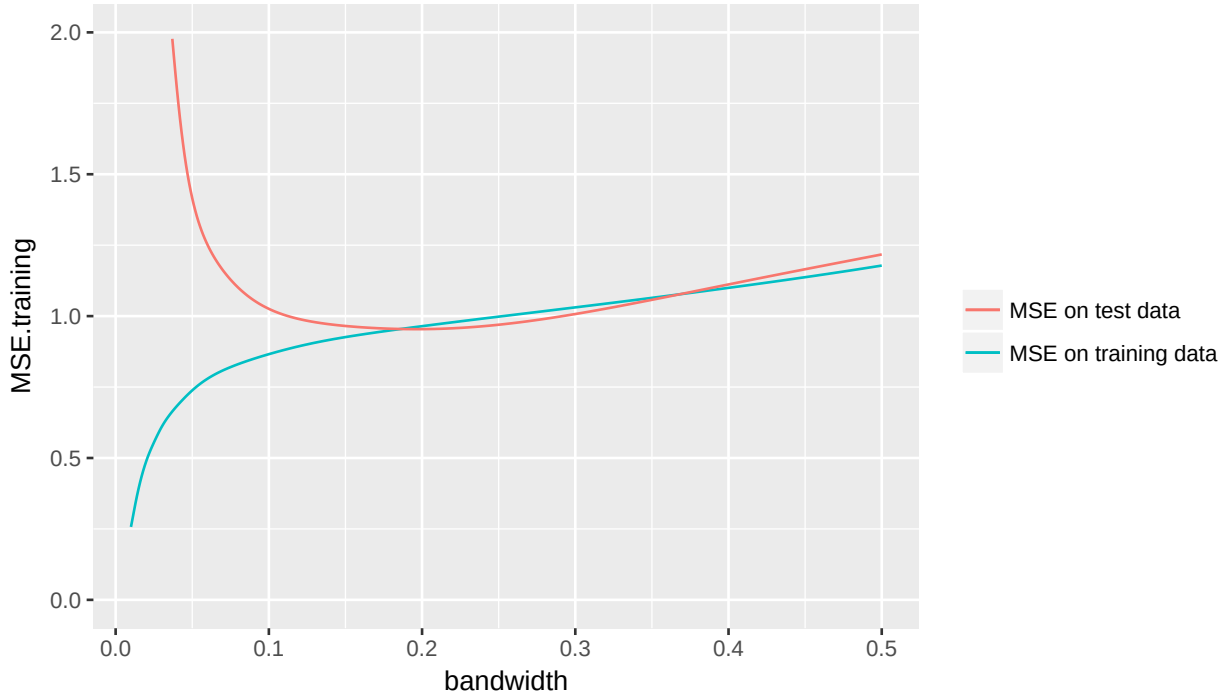
```
MSE <- matrix(nrow = length(bandwidth), ncol = 2)
for (i in 1:length(bandwidth)) {
    Rg <- npreg(y ~ x, bws = bandwidth[i], training, regtype = "ll")
    PredRg1 <- fitted.values(Rg)
    PredRg2 <- predict(Rg, newdata = test)
    MSE[i, 1] <- mean((PredRg1 - training$y)^2)
    MSE[i, 2] <- mean((PredRg2 - test$y)^2)
}
MSE <- data.frame(MSE.training = MSE[, 1], MSE.test = MSE[, 2])
```

Step 10

In this step, using `ggplot`, we draw on the same plot how the MSE on training data, and test data, change when the bandwidth increases. It is easy to see in the graph that, when the bandwidth increases, the MSE on training data increases, as we are going from something that is super flexible to something that is less and less flexible. So we are making it less flexible and for this reason it has more problems adapting to that dataset.

Regarding the MSE on test data. At the beginning, when there is a lot of flexibility, the model is exclusively dedicated to training data, so it only knows the training data and it is effective just for this data, which means that it is a poorly predictor for the test dataset. Then, when the bandwidth increases, it starts to generalize, and by generalizing it makes the model more distant from the training dataset, i.e. closer to the test. Consequently, the orange line decreases until it reaches a point where the benefit it gets generalizing the model is lower than the cost it suffers making it less flexible, and at that point it starts to rise.

```
ggplot(MSE) + geom_line(aes(x = bandwidth, y = MSE.training, colour = "MSE on training data"),
    data = MSE) + geom_line(aes(x = bandwidth, y = MSE.test, colour = "MSE on test data"),
    data = MSE) + ylim(0, 2) + guides(colour = guide_legend(title = ""))
```



Task 3B

Step 1

We import the CNIL dataset using `read_delim`, which is useful for reading the most common types of flat file data, comma separated values and tab separated values, respectively.

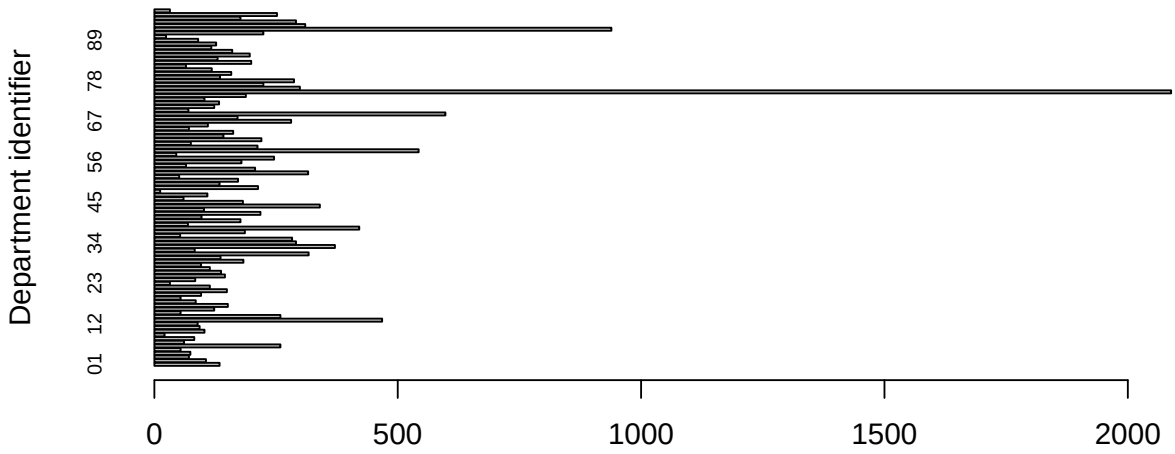
```
library(readr)
CNIL <- read_delim("cnil.csv", ";", escape_double = FALSE, trim_ws = TRUE)
colnames(CNIL)[1] <- "SIREN"

system.time(CNIL <- read_delim("cnil.csv", ";", escape_double = FALSE, trim_ws = TRUE))
```

```
##    user  system elapsed
##   0.145   0.003   0.149
```

Step 2

Firstly, to answer this question, we create a new column with the first two digits of the postcode, as long as a department in France is uniquely identified by this first two digits. Then, and after cleaning our dataset, we design our barplot, using the `barplot`, of the *Number of organizations that has nominated a CIL*.

```
library(stringr)
CP2 <- str_sub(CNIL$Code_Postal, start = 1, end = 2)
CNIL$CP2 <- CP2
CNIL$numeric <- CNIL$CP2 %in% c("01", "02", "03", "04", "05", "06", "07", "08",
    "09", 0:100)
CNIL <- CNIL[CNIL$numeric == TRUE, ]
count <- table(CNIL$CP2)
barplot(count, ylab = "Department identifier", horiz = T, cex.names = 0.7)
```



Step 3

Let us explain how we import Siren dataset. In order to import such a huge dataframe we use the function `fread()`, i.e., a similar function to `read.table but faster and more convenient. Concretely,`fread()‘ function, also described as the “fast and friendly file finagler”, is meant to import data from regular delimited files directly into R, without any detours or nonsense. Note that “regular” in this case means that every row of your data needs to have the same number of columns. Last, our laptop performed the task in about 5 minutes, but we wouldn’t recommend it to our worst enemy.

Then, after eliminating some duplicates of the Siren dataset, we merge it with Cnil data using `merge`, which just merge two data frames by common columns or row names, in our case column SIREN.

```
install.packages("data.table", repos = "http://cran.us.r-project.org")
library(data.table)
SIREN <- fread("siren.csv")
dup <- duplicated(SIREN[, 1])
SIREN <- SIREN[!dup, ]
merge <- merge(SIREN, CNIL, by = "SIREN")
write.csv(merge, file = "merge.csv")

system.time(SIREN <- fread("siren.csv"))
```

```
## Read 10831176 rows and 100 (of 100) columns from 8.068 GB file in 00:04:41
##    user  system elapsed
## 153.945 142.497 445.545
```

Now, we load the file that we uploaded.

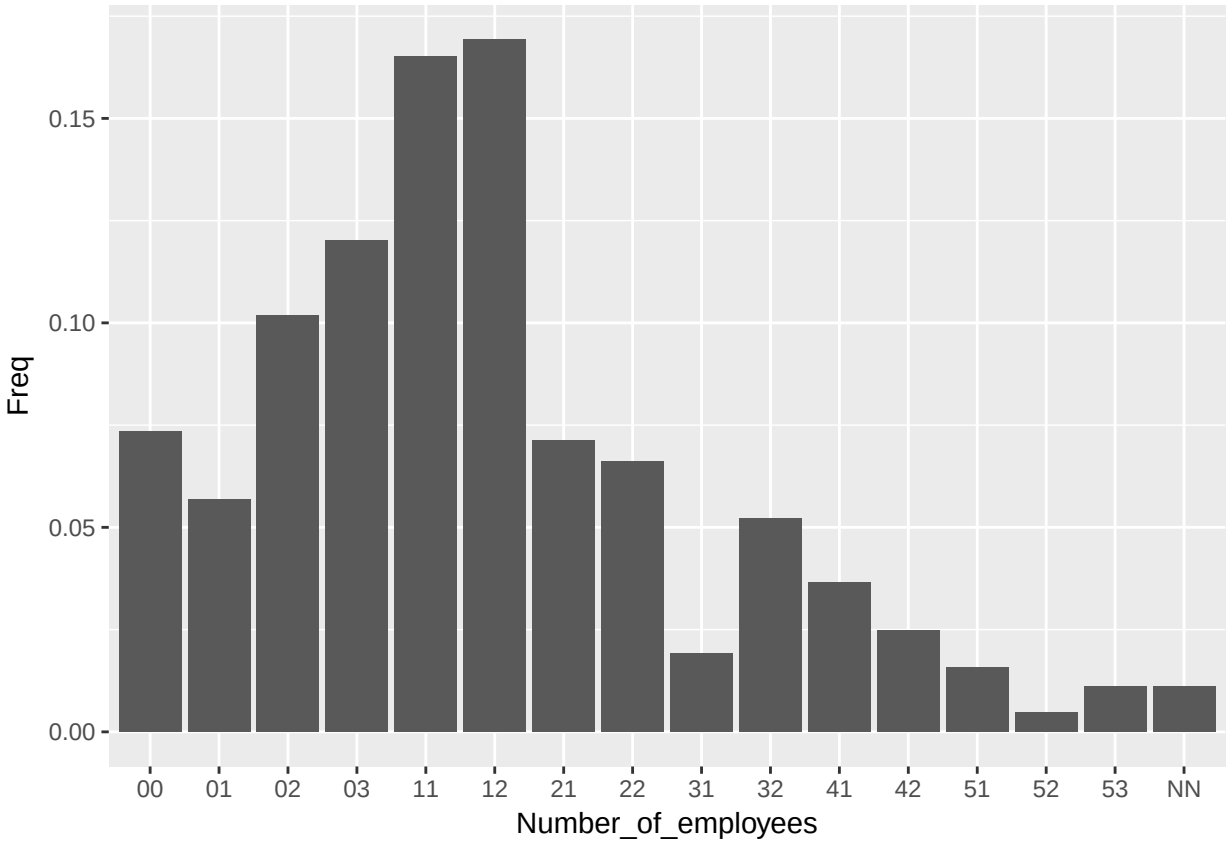
```
merge <- read_csv(file = "merge.csv")

system.time(merge <- read_csv(file = "merge.csv"))
```

```
##    user  system elapsed
##   0.339   0.017   0.358
```

Step 4

After some adjustments, which are shown below, we use `ggplot` function in order to plot the histogram of the size of the companies that nominated a CIL. We can observe in the graph that the histogram is slightly skewed to the right, or positively skewed, due to more than 50 percent of firms have between 3 and 49 employees. Without meaning to lose detail or relevance, and just in a way to improve the graph’s intuition, one could eliminate some of the higher intervals as long as they might be considered outliers.

```
sum <- sum(table(merge$TEFEN))
prop <- table(merge$TEFEN)/sum
pro <- data.frame(prop)
colnames(pro)[1] <- "Number_of_employees"
ggplot(data = pro, aes(x = Number_of_employees, y = Freq)) + geom_bar(stat = "identity")
```



Value	Interpretation
0	0 employee (but having employed during the reference year)
1	1 or 2 employees
2	3 to 5 employees
3	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	10000 employees and more
NN	No employee during the reference year