Introduction à R et aux statistiques en sciences sociales

Lev Lhommeau

2023-03-02

Contents

Chapter 1

Remarques préliminaires

AVERTISSEMENT: il s'agit d'une publication en chantier (*work in progress*) qui n'a pas encore été corrigée, tant au niveau orthographique et grammatical que du contenu.

Ce syllabus[1] est une introduction au langage de programmation R et à l'analyse statistique en sciences sociales. Il s'adresse aux étudiants en sciences sociales, telles que la science politique, la sociologie et la communication, ainsi qu'à toute personne qui souhaite découvrir les bases de l'analyse statistique des phénomènes sociaux avec R. Ce syllabus veut contribuer à combler une lacune dans les publications existantes en langue française : il existe certes de très bons ouvrages théoriques d'introduction aux statistiques à l'usage des sciences sociales, tout comme il existe d'excellents ouvrages d'introduction à la programmation avec R. En revanche, une introduction pratique à l'analyse des phénomènes sociaux avec R fait encore défaut. C'est cette lacune que comble la présente introduction.

Ce syllabus se fonde sur les travaux pratiques que j'assure sous la direction de Philippe Bocquier et Bruno Schoumaker à l'UCLouvain. Au vu du public cible, les concepts mathématiques et statistiques mobilisés dans cette introduction sont tout à fait abordables. Cette introduction essaiera de faire au maximum l'économie de formules mathématiques et, lorsqu'elles sont indispensables, elles s'efforcera de les "traduire" en langue française. Il est cependant préférable, si le lecteur ne dispose pas de connaissances en statistique, d'accompagner la lecture de ce syllabus de la lecture d'un manuel de statistique ou du suivi d'un cours de statistiques. À cette fin, je conseille en langue française l'ouvrage suivant :

Masuy-Stroobant & Costa (2013): Analyser les données en sciences sociales. De

[1] Ce document étant développé lors d'enseignements à l'Université catholique de Louvain en Belgique, nous utilisons le belgicisme *syllabus* pour désigner ce support. Par ailleurs, nous utiliserons de nombreux exemples issus de la Belgique. Cela n'altère cependant en rien son utilité pour les lecteurs francophones d'autres pays.

la préparation des données à l'analyse multivariée, Bruxelles: P.I.E. Peter Lang.

La présente introduction a vocation à être accessible à des personnes sans aucune expérience en programmation. Aucun prérequis en programmation n'est donc nécessaire pour la suivre. Elle ne prétend pas à l'exhaustivité, mais souhaite donner un premier aperçu des statistiques en sciences sociales et des possibilités qu'offre le langage de programmation R. Vu qu'il s'agit d'une introduction aux statistiques, les autres usages de R, notamment pour les présentations (avec *R Markdown* ou *Shiny*) et l'automatisation de certaines tâches, ne seront pas traités. Si vous cherchez une introduction spécifique à la programmation en R en langue française, je vous conseille l'ouvrage suivant:

Vincent Goulet: Introduction à la programmation en R, (téléchargez le ici)

Notons enfin, que nous avons fait le choix de montrer, outre le logiciel R de base, les méthodes issues du *package tydiverse*. *Tydiverse* est ce qu'on appelle un "dialecte". Un "dialecte" modifie une langue de programmation sans l'altérer fondamentalement. Pourquoi ajouter un "dialecte" à cette introduction ? Le *tydiverse* est très utilisé et il est donc plus qu'utile de le connaître si l'on souhaite se lancer dans R. De plus, le *tydiverse* avec ses *package* tels que *dplyr* et *ggplot2* a grandement simplifié la manipulation des données et la réalisation de graphiques de qualité[2].

1.1 Installer R

Pour suivre cette introduction, veuillez installer le langage de programmation R et *ensuite* l'environnement de développement *RStudio*[3] (bien dans cet ordre-là: installez **d'abord** R et ensuite *RStudio Desktop*) sur votre ordinateur. R est une langue de programmation libre, c'est-à-dire qu'elle est entièrement gratuite ! Plus besoin d'acheter d'onéreuses licences pour des programmes d'analyse statistique payants tels que *Stata*, *SPSS* et *SAS*, bien souvent moins performants que R. *RStudio* est également gratuit dans sa version de base (*RStudio Desktop*), entièrement suffisante pour l'usage que nous allons en faire.

Pour télécharger R, rendez-vous sur le site du *CRAN* et installez la dernière version de R pour votre système d'exploitation (Linux, Windows ou macOS):

https://cran.r-project.org/

Pour télécharger RStudio rendez-vous sur le site de *posit*, l'entreprise qui développe *RStudio*, et installez la dernière version de *RStudio* pour votre système d'exploitation (Linux, Windows ou macOS):

[2] Si vous voulez lire une introduction en français plus complète au *tidyverse* nous renvoyons à l'ouvrage "Introduction à R et au tidyverse" de Julien Barnier.

[3] *RStudio* est un environnement de développement spécialisé pour R, mais il peut également exécuter le langage de programmation plus généraliste *Python*. Vous pouvez également utiliser R sans environnement de développement ou avec un autre environnement que *RStudio*, tel que *VSCode*. *RStudio* est cependant l'environnement le plus utilisé pour coder en R et certainement le plus adapté aux débutants.

https://posit.co/

1.2 Se retrouver à l'intérieur de ce syllabus

Ce syllabus est composé de plusieurs chapitres qui abordent chacun des questions différentes. Le syllabus suit une progression : la complexité augmente avec les chapitres. Il est donc conseillé, surtout au début, de bien suivre l'ordre de celui-ci. Au fur et à mesure que vous avancerez, vous pourrez prendre davantage de libertés avec l'ordre dans lequel vous lisez les chapitres.

Lorsqu'une fonction est introduite, j'ajoute un lien vers la documentation de cette fonction. Ainsi, vous pouvez facilement consulter les explications et les exemples. Malheureusement, la documentation "officielle" de R est en anglais. Si vous cherchez en ligne, vous trouverez cependant de nombreux guides en français. Si possible, je renverrai aussi vers ces sources en français. Vous trouverez des introductions et des fiches de résumé sur R (ou de certains de ces *packages* et *Rstudio*) dans plusieurs langues, dont le français, sur le site de *posit*:

https://posit.co/resources/cheatsheets/?type=translations/#translation-3

Vous pouvez naviguer entre les chapitres et les sous-sections dans le volet de gauche. Vous pouvez, par ailleurs, naviguer entre les chapitres en appuyant sur les touches de gauche et de droite. Vous pouvez télécharger la version PDF et *epub* de ce syllabus en cliquant sur l'icône de téléchargement en haut de la page[4].

Les codes R à introduire se trouvent dans des encadrés grisés comme celui-ci :

```
# Le signe # marque un commentaire qui sera ignoré par R.
#Si vous voulez afficher le fameux "Hello World", vous pouvez le
faire ainsi:
print("Hello world!")
#Ou ainsi:
message("Hello world!")
```

Si vous passez votre souris sur cet encadré, vous verrez apparaître, en haut à droite de l'encadré, une icône vous permettant de copier son contenu[5]. Celui-ci pourra être collé dans votre *R Script* puis exécuté. Si vous copiez ce code dans votre *R Script* puis l'exécutez (ou si vous le copiez directement dans votre console puis appuyez sur la touche "*Entrée*" de votre clavier), voilà ce qui devrait apparaître dans votre console :

```
## [1] "Hello world!"
```

```
## Hello world!
```

[4]Notez que les fonctions "interactives" de ce syllabus ne pourront pas être maintenus au format PDF et epub et que des difficultés de mise en page ne sont pas exclues.

[5]Cette fonctionnalité n'est malheureusement pas présente dans la version PDF et epub de ce syllabus.

Si vous avez des questions ou des remarques, n'hésitez pas à me contacter: lev.lhommeau@uclouvain.be / lhommeau@gmail.com[6].

[6]Si je ne réponds pas à l'une de ces adresses, essayez de me contacter par l'autre.

Chapter 2

Prise en main de R

2.1 Commandes dans R

Vous vous demandez peut-être comment exécuter des commandes dans R. Si vous utilisez *RStudio*, vous avez deux possibilités. Vous pouvez indiquer le code directement dans la console qui se trouve dans l'onglet "Console". Une fois que la commande est introduite, appuyez sur la touche *Entrée* de votre clavier pour la soumettre à l'interpréteur de R. Cependant, cette manière ne sera pas votre moyen principal de soumettre des commandes à R. Vous allez plutôt utiliser des fichiers *R Script* dans lesquels vous noterez votre code : pour créer un nouveau fichier *R Script*, cliquez en haut à gauche dans *RStudio* sur l'icône représentant une feuille blanche avec un signe plus (+) sur fond vert et choisissez *R Script* dans le menu déroulant.

Un fichier *R Script* est un document texte rassemblant l'ensemble des commandes que vous souhaitez soumettre à R. C'est un peu comme une "recette" où chaque commande correspond à un ingrédient ou une manipulation à faire pour obtenir le plat désiré. Sauf, bien entendu, que le plat désiré ne sera pas une mousse au chocolat ou un rôti de bœuf, mais le résultat d'un calcul, d'une manipulation de données, d'une analyse statistique ou une représentation graphique. Ce fichier *R Script* peut être enregistré sous un nom de votre choix suivi du suffixe "*nom*.R" (par exemple: "Analyse.R"). Cela vous permet de commenter et de reproduire les commandes que vous avez écrites : si vous indiquiez vos commandes uniquement dans la console, vous ne pourriez que difficilement reproduire vos manipulations et analyses. Or, la reproductibilité est un critère essentiel de la scientificité. Par ailleurs, le code que vous avez écrit peut être réutilisé lors d'analyses futures. Cela vous fera gagner du temps!

Mais comment exécuter les commandes que vous avez notées dans votre fichier de *R Script* ? Il suffit de placer le curseur sur une commande et d'appuyer sur la flèche verte "*Run*" qui se situe au-dessus à droite de l'onglet où se trouve

votre code. Vous pouvez aussi appuyer sur les touches "*Contrôle + Entrée*" de votre clavier. Par ailleurs, vous pouvez sélectionner une partie de votre code, puis exécuter cette partie du code de la même manière. Notez que les commandes précédées par "#" (losange/*hastag*) ne seront pas exécutées par R. L'interpréteur de R ignore ces commandes. Cela sert à commenter son code: on explique ce que l'on fait pour se retrouver plus tard dans son code.

2.2 Fonctions et aide dans R

Comme *Excel* (ou d'autres langues de programmation tels que *Python* ou *C*), R dispose de **fonctions** que vous allez pouvoir utiliser pour mener vos analyses statistiques et manipuler vos données. En informatique, une fonction est un programme qui exécute une (suite de) commande(s) dont elle retourne un résultat et que l'on peut réutiliser dans d'autres programmes. Les fonctions en R sont composées de leur nom[1], suivi d'une parenthèse dans laquelle vous pourrez indiquer, le cas échéant, les arguments séparés par des virgules. Ainsi, si vous voulez connaître la moyenne de la suite $A = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$, vous pouvez utiliser la fonction *mean()*. Avant il faut créer un **vecteur**[2], c'est-à-dire un objet composé d'une suite de chiffres, auquel vous devez donner un nom, par exemple *suite_a*, à l'aide de la fonction *c(arg)* qui crée un vecteur en concaténant la suite de chiffres contenue entre parenthèses, qui forme ainsi l'**argument** de la fonction *c()* :

```
#Nous créons la suite A allant de 0 à 10 et la nommons suite_a
dans R:
suite_a<-c(0,1,2,3,4,5,6,7,8,9,10)

#À l'aide de la fonction mean(), nous calculons la moyenne de
cette suite:
mean(suite_a)
```

```
## [1] 5
```

L'objet *suite_a* contenant la suite A est l'argument de la fonction *mean()* contenu dans la parenthèse de la fonction. Comme vous pouvez le voir, cette fonction ne renvoie qu'un seul résultat noté [1], la moyenne de la suite A, qui est égale à 5.

Si vous avez besoin d'aide concernant une fonction dans R, écrivez la fonction précédée de "?". Par exemple:

```
?setwd
```

[1] Attention: R est sensible à la casse: "Mean" est différent de "mean". Soyez donc toujours très attentif à la casse lorsque vous codez en R.

[2] Pour savoir comment créer des vecteurs en R, nous renvoyons à la page suivante: http://www.sthda.com/french/wiki/les-vecteurs-vector.

Dans *RStudio*, dans l'onglet "*Help*" en bas à droite[3], s'affiche alors l'aide. Vous pouvez également sélectionner une fonction et appuyer sur la touche "*F1*" de votre clavier. Par ailleurs, de bonnes recherches *google* et des questions pertinentes posées à *ChatGPT* peuvent également vous éclairer sur les fonctions. N'hésitez pas à demander à *ChatGPT* de vous donner des exemples de code[4].

Outre les fonctions déjà existantes dans R et ses *packages*, vous pouvez définir vos propres fonctions. Celles-ci vous permettent de réaliser ce dont vous avez besoin sur mesure (analyse statistique, transformation de données, graphiques, etc.). Cela peut être réalisé de manière assez simple dans R[5].

2.3 Espace de travail (*working directory*)

Lorsque l'on travaille avec R, il faut définir un espace de travail. C'est le dossier où R sauvegardera et charge les fichiers et bases de données avec lesquels vous travaillez. Si vous ne le déterminez pas, R choisit un dossier par défaut. Sur un ordinateur Windows, cela sera bien souvent le dossier "Documents". Nous vous conseillons de créer un dossier dédié à votre travail en R et de définir celui-ci comme répertoire de travail. Par exemple, si vous aviez créé un dossier "Espace_R" dans un dossier "Documents" sur votre disque dur C, vous soumettriez le code suivant à R pour en faire votre espace de travail :

```
setwd("C:/Documents/Espace_R") # Indiquez le chemin du dossier
qui doit être votre répertoire de travail
```

Vous aurez remarqué que le chemin d'accès et les dossiers en tant qu'argument de la fonction *setwd()* sont entre guillemets. Par ailleurs, dans R, les chemins d'accès entre dossiers ("Documents") et sous-dossiers ("Espace_R") sont séparés par des barres obliques (/), comme pour une division. Dans Windows, les chemins d'accès sont notés par des barres obliques inversées (\). Il faut donc corriger cela lorsque vous copiez des chemins d'accès dans R.

Sous Windows, vous pouvez aussi choisir manuellement le dossier dans lequel se trouve votre espace de travail en imbriquant la fonction *choose.dir()* dans la fonction *setwd()*:

```
setwd(choose.dir()) # Indiquez le chemin du dossier qui doit être
votre répertoire de travail
```

Si vous lancez ce code, une fenêtre s'ouvre dans laquelle vous pourrez cliquer sur le dossier que vous souhaitez déterminer en tant que dossier de travail. Mal-

[3]Vous trouverez, en langue française, une explication de l'interface de *RStudio* ici : https://posit.co/resources/cheatsheets/?type=translations/#translation-3

[4]Les règles générales de référencement et de citation s'appliquent également à *ChatGPT*. Indiquez-le donc si vous reprenez des indications *ChatGPT* lors de vos travaux.

[5]Ce syllabus contient des exemples de créations de fonctions. Cependant, l'apprentissage de la rédaction de fonctions ne fait pas partie de ses objectifs principaux.

heureusement, cela fonctionne uniquement sous Windows et pas sous Linux et macOS.

Que votre système d'exploitation soit Linux, macOS ou Windows, vous pouvez définir votre espace de travail grâce aux menus de *RStudio*. Allez sur le menu "Session" puis choisissez "*Set Working Directory*" dans le menu déroulant. Parmi les options disponibles cliquez sur "*Choose Directory*". Une fenêtre s'ouvre alors dans laquelle vous pourrez choisir le fichier de votre choix pour déterminer l'emplacement de votre espace de travail.

2.4 Installer et lancer des *packages* (bibliothèques)

Comme d'autres langues de programmation, R dispose d'un grand nombre de *packages*[6]. Ceux-ci enrichissent le langage de nouvelles fonctions qui n'étaient pas encore présentes dans sa version initiale. R est particulièrement riche en *packages*. Sur *CRAN*, le répertoire "officiel" de R, 19136 *packages* étaient répertoriés en janvier 2023.

Pour installer un *package*, il y a la commande générique *install.packages()*. Lors de cette formation, nous allons beaucoup utiliser des *packages* issus du *tydiverse*, un ensemble de *packages*, permettant de manipuler les données et de les représenter graphiquement. Cette suite de *packages* est particulièrement populaire et très largement utilisée. Nous allons donc installer cet ensemble de *packages* avec la commande *install.packages()* :

```
install.packages(tidyverse, dependencies=T) #l'argument
"dependencies=T" permet
        #d'installer automatiquement tous les packages dont a
        besoin le package
        #que l'on installe pour fonctionner correctement.
```

Installer un *package* ne suffit pas pour l'utiliser. Il doit être chargé à chaque fois que R est fermé puis redémarré. Cela se fait avec la commande *library()*. Lançons donc le package *tidyverse* :

```
library(tidyverse)
```

Le *package* est à présent chargé et peut être utilisé. Si R vous retourne une erreur lorsque vous chargez un *package*, cela est souvent dû au fait que le *package* demandé n'est pas (correctement) installé (vérifiez aussi de l'avoir orthographié correctement et d'avoir respecté la casse, c'est-à-dire les minuscules et majuscules). Si vous lancez une fonction et que celle-ci ne s'exécute pas (et renvoie une erreur), vérifiez que le *package* à laquelle elle appartient est bien chargé.

[6] En français, l'on parle de bibliothèques. Nous alignons cependant ce syllabus sur l'usage en anglais que l'on retrouve dans R. Nous parlerons donc de *package(s)*.

Vous verrez cela dans l'onglet "*Packages*" en bas à droite de *RStudio* : si le *package* apparaît dans la liste, c'est qu'il est installé. S'il est coché, c'est qu'il est chargé. S'il n'apparaît pas dans la liste, installez-le. S'il n'est pas coché, lancez-le. Vous pouvez le lancer avec la commande *library()*, mais vous pouvez aussi le lancer en cochant le *package* dans l'onglet.

Vous pouvez aussi exécutez la commande suivante[7] et vérifier si le *package* se trouve dans la liste. Au dessus à droite du tableau qui va apparaître, vous trouverez un champ pour rechercher dans le tableau :

```
if (!require("pacman")) install.packages("pacman")
pacman::p_load(tidyverse,DT)

DT::datatable(tibble::tibble(
     Package = names(installed.packages()[,3]),
     Version = unname(installed.packages()[,3])
 ))
```

Une façon plus simple d'installer et de charger les *packages* est le package *pacman*. Celui-ci permet d'installer les *packages* et de les lancer en une seule commande: *p_load*. Cette commande vérifie si un *package* est installé : s'il est installé, il le lance, s'il ne l'est pas, il l'installe, puis le lance. Cela peut rendre le lancement de *packages* plus facile et rapide. Par ailleurs, cela permet de plus facilement partager son code : l'interlocuteur peut exécuter votre code sans devoir vérifier s'il a installé les *packages* que vous utilisez :

```
if (!require("pacman")) install.packages("pacman") #Cela vérifie
                            #si le package pacman est installé.
                            #S'il ne l'est pas, il est installé.

pacman::p_load(tidyverse) #On lance le package tidyverse s'il est
installé,
                          #sinon on l'installe puis le lance
                          automatiquement.
```

Si vous voulez lancer plusieurs *packages* en même temps, il suffit de les ajouter à la suite, séparés par des virgules, entre les parenthèses suivant la fonction *p_load*. Ainsi, si nous voulions charger les *packages* du *tydiverse*, ainsi que le *package DT*[8] :

```
if (!require("pacman")) install.packages("pacman") #Cela vérifie
                            #si le package pacman est installé.
                            #S'il ne l'est pas, il est installé.
```

[7] Je me suis inspiré de la discussion suivante pour cette solution : https://stackoverflow.com/questions/38481980/get-the-list-of-installed-packages-by-user-in-r

[8] Le *package DT* permet de créer des tableaux interactifs assez pratiques pour se faire une idée des données. Attention cependant, car les bases de données avec un nombre trop important de lignes (des dizaines de milliers) peuvent faire "crasher" R.

```
pacman::p_load(tidyverse, DT) #La liste des packages, séparés
                      #par des virgules, que l'on souhaite
                      #charger et, si nécessaire, installer.
```

Vous pouvez aligner à la suite autant de *packages* que vous le souhaiter en les séparant par des virgules.

Dans ce syllabus, nous allons utiliser *pacman* pour éviter de devoir vérifier si certains *packages* sont installés avant de les lancer.

2.5 Objets et environnement dans R

R est une langue de programmation orientée **objet**. Cela signifie que l'on définit des objets[9] (bases de données, valeurs, fonctions, etc.) que l'on fait interagir les uns avec les autres pour obtenir le résultat escompté[10]. Vous pouvez vous imaginer cela un peu comme différentes pièces de "*Légo*" que vous assemblez pour construire un château à votre goût.

Dans R, comme nous l'avons fait plus haut pour créer un vecteur de chiffre *suite_a*, on crée un objet avec le symbole "<-", une flèche pointant vers la gauche[11]. La flèche montre vers le nom de l'objet à créer à gauche et à droite l'on définit le contenu de cet objet (valeur, vecteur, matrice, etc.).

L'on peut ainsi créer l'objet *bombe_atomique* qui contient la valeur 1000^2 (le chiffre mille au carré) :

```
bombe_atomique <- 1000^2 # À gauche, le nom de l'objet
(bombe_atomique),
                         # à droite, la valeurqu'elle contient
                         (1000^2)
bombe_atomique           # Maintenant, on affiche le contenu de
cet objet dans la console.
```

```
## [1] 1e+06
```

La valeur de l'objet *bombe_atomique* apparaît dans la console. Par ailleurs, le nom de l'objet apparaît en haut à droite dans l'onglet "*Environement*" de *RStu-*

[9] En programmation, l'on qualifie ce type d'objets fréquemment de "variables". Cependant, cela peut mener à des confusions: en statistiques, une variable est une caractéristique d'observations. Afin d'éviter ce type d'ambiguïtés, nous n'allons pas utiliser le terme variable au sens informatique, mais exclusivement au sens statistique/mathématique.

[10] Une introduction un peu technique à la programmation orientée objet (POO) se trouve ici: https://hdd34.developpez.com/cours/artpoo/

[11] On peut aussi utiliser le signe "=" comme dans d'autres langues de programmations telles que python. Cependant, notamment pour éviter des confusions avec d'autres utilisations du signe "=" dans R (notamment en tant qu'argument de fonctions) et ainsi faciliter la lecture du code, l'on conseille vivement d'utiliser "<-" pour créer des objets.

dio. Si vous avez calculé la moyenne de la suite A dans la section Fonctions et aide dans R, vous y verrez aussi l'objet *suite_a*. Dans R un **environnement**[12] est un espace où les objets sont enregistrés vous permettant de les analyser et de les manipuler.

Vous aurez peut-être remarqué que la valeur de l'objet *bombe_atomique* est affichée en notation scientifique (qui se lit : 1 suivi de 6 zéros) :

```
## [1] 1e+06
```

Vous pouvez désactiver cet affichage avec la commande suivante :

```
options(scipen=999)
```

Vérifiez à nouveau la valeur de l'objet *bombe_atomique* :

```
bombe_atomique
```

```
## [1] 1000000
```

Comme vous pouvez le voir, la notation scientifique à disparue. Cela peut faciliter la lecture des résultats de vos analyses !

2.6 Importer et enregistrer des données

R dispose de nombreuses fonctions intégrées pour importer des données sous différents formats (*read.table*, *read.delim*, *read.csv*, etc.). Il y a, par ailleurs, un certain nombre de *packages* qui ajoutent de nouvelles fonctions et d'autres formats de données que l'on peut intégrer.

Ainsi, le package *readr*, appartenant à "l'univers" *tydiverse*, étant ainsi automatiquement chargé si vous chargez le *tydiverse* (voir ici), ajoute des fonctions assez intéressantes pour charger des données au format texte (*csv*, *txt*) : *read_delim*, *read_csv*, *read_csv2*, etc. De même, le *package readxl* permet de charger des fichiers au format Excel dans R. Bien que ce *package* appartienne également à "l'univers" *tydiverse*, le *package readr* doit être chargé séparément. Un troisième *package* issu de cet "univers" est le *package haven* : il permet de charger des bases de données enregistrées au format des logiciels statistiques payants *SAS*, *Stata* et *SPSS*. En effet, beaucoup de bases de données sont enregistrées dans l'un de ces formats qui ont l'avantage, contrairement aux formats *CSV* et *Excel*, de pouvoir notamment enregistrer des libellées des variables et des valeurs. Le *package haven* permet de préserver ces avantages lorsque l'on importe les données dans R :

[12] R connaît plusieurs environnements. L'environnement où est enregistré l'objet définit sa portée, c'est-à-dire dans quelles circonstances les objets sont accessibles. Dans ce syllabus, nous utiliserons consciemment (pratiquement) exclusivement l'environnement global dénommé "*.GlobalEnv*". Vu qu'il s'agit d'une introduction, nous ne nous attardons pas sur ce point.

```
if (!require("pacman")) install.packages("pacman") #Cela vérifie
                              #si le package pacman est installé.
                              #S'il ne l'est pas, il est installé.

pacman::p_load(readxl,haven) #On lance les packages readxl et
haven.
```

Dans cette introduction nous allons utiliser les données issues de l'Enquête sociale européenne / *European social survey (ESS)*[13]. Nous allons utiliser la version 9 de l'*ESS* (ESS 9), contenant des données collectées en 2018. Les données de l'*ESS* sont mises à disposition aux formats *SPSS* (avec le suffixe ".sav"), *Stata* (avec le suffixe ".dta") et *SAS* (avec le suffixe ".sas"). Nous allons télécharger les données de l'*ESS* au format *SPSS* (avec le suffixe ".sav").

Pour pouvoir télécharger les données, il faut s'inscrire sur le site de l'*ESS*. Cette inscription est entièrement gratuite. Les données de l'*ESS* peuvent être librement utilisées à des fins non commerciales, notamment pour la recherche et l'enseignement. Nous vous demanderons de vous inscrire sur le site de l'*ESS* et de télécharger les données : téléchargez les données de l'*ESS* 9 (*ESS Round 9*) pour tous les pays au format *SPSS* (fichier se terminant par ".sav").

Nous vous laissons suivre les indications sur le site (en anglais) pour vous inscrire et télécharger les données[14]. Il existe le *package esssurvey* qui permettait de télécharger directement les données de l'*ESS* dans R. Malheureusement, en raison de changements concernant l'accès aux données de l'*ESS* au cours de 2022, ce *package* ne fonctionne plus (janvier 2023).

Une fois que vous avez téléchargé le fichier de l'*ESS* 9 (2018) au format *SPSS* (fichier se terminant par ".sav"), il faudra charger la base de données dans R. Pour cela, copiez le fichier dans le dossier dans lequel se trouve votre espace de travail R[15]. Par exemple, notre fichier pourrait se nommer "ESS9e03_1.sav". Adaptez le nom du fichier, s'il porte un autre nom chez vous. Une fois que le fichier de l'*ESS* 9 se trouve dans votre espace de travail, indiquez le code suivant[16] :

```
ESS9<-read_sav("ESS9e03_1.sav") #Vous indiquez le nom de votre
fichier
                        #entre guillemets (""). Si le fichier ne se
                        #trouve pas dans votre espace de travail,
                        vous
```

[13] Davantage d'informations sur l'Enquête sociale européenne peuvent être trouvées sur son site: https://www.europeansocialsurvey.org/about/

[14] Si vous suivez un enseignement avec Lev Lhommeau, il se peut que ces données vous soient mises à disposition dans ce cadre.

[15] Il est possible de charger des fichiers ne se trouvant pas dans votre espace de travail. Cela nécessite cependant d'indiquer l'ensemble du chemin d'accès dans R, ce qui peut être laborieux.

[16] L'exécution de cette commande nécessite le chargement préalable du *package* "*haven*", comme indiqué plus haut.

```
                    #devrez indiquer tout le chemin d'accès.
```

Avec cette commande, vous avez chargé la base de données dans l'environnement de R. Vous l'avez nommé "ESS9"[17]. Elle apparaît dans l'onglet "*Environment*" en haut à droite de *RStudio*. Si vous utilisez le système d'exploitation Windows, vous pouvez aussi utiliser la commande *choose.files()* à l'intérieur de la fonction *read_sav*. Lorsque vous exécutez cette commande, une fenêtre s'ouvre vous permettant de choisir manuellement le fichier à charger :

```
ESS9<-read_sav(choose.files())
```

Le fichier étant chargé dans l'environnement, vous pouvez l'étudier et le manipuler dans R. Sélectionnez l'objet et exécutez-le :

```
ESS9
```

Dans votre console s'affichera alors un extrait de la base de données :

```
## # A tibble: 49,519 x 572
##    name   essro~1 edition prodd~2  idno cntry    dweight pspwght pweight anwei~3 nwspol netus~
##    <chr>    <dbl> <chr>   <chr>   <dbl> <chr+lb>   <dbl>   <dbl>   <dbl>   <dbl> <dbl+> <dbl+l
##  1 ESS9e~       9 3.1     17.02.~    27 AT [Aus~   0.581   0.218   0.302  0.0659 60     5 [Eve
##  2 ESS9e~       9 3.1     17.02.~   137 AT [Aus~   1.06    0.413   0.302  0.125  10     5 [Eve
##  3 ESS9e~       9 3.1     17.02.~   194 AT [Aus~   1.38    2.27    0.302  0.686  60     4 [Mos
##  4 ESS9e~       9 3.1     17.02.~   208 AT [Aus~   0.993   0.386   0.302  0.117  45     5 [Eve
##  5 ESS9e~       9 3.1     17.02.~   220 AT [Aus~   0.377   1.03    0.302  0.312  30     1 [Nev
##  6 ESS9e~       9 3.1     17.02.~   254 AT [Aus~   1.48    0.576   0.302  0.174  45     2 [Onl
##  7 ESS9e~       9 3.1     17.02.~   290 AT [Aus~   0.992   0.721   0.302  0.218  60     1 [Nev
##  8 ESS9e~       9 3.1     17.02.~   301 AT [Aus~   0.310   0.130   0.302  0.0393 30     1 [Nev
##  9 ESS9e~       9 3.1     17.02.~   305 AT [Aus~   1.23    1.77    0.302  0.535  30     5 [Eve
## 10 ESS9e~       9 3.1     17.02.~   400 AT [Aus~   0.459   0.743   0.302  0.224  25     4 [Mos
## # ... with 49,509 more rows, 559 more variables: ppltrst <dbl+lbl>, pplfair <dbl+lbl>,
## #   pplhlp <dbl+lbl>, polintr <dbl+lbl>, psppsgva <dbl+lbl>, actrolga <dbl+lbl>, psppipla <dbl
## #   cptppola <dbl+lbl>, trstprl <dbl+lbl>, trstlgl <dbl+lbl>, trstplc <dbl+lbl>, trstplt <dbl+
## #   trstprt <dbl+lbl>, trstep <dbl+lbl>, trstun <dbl+lbl>, vote <dbl+lbl>, prtvtcat <dbl+lbl>,
## #   prtvtdbe <dbl+lbl>, prtvtdbg <dbl+lbl>, prtvtgch <dbl+lbl>, prtvtbcy <dbl+lbl>,
## #   prtvtecz <dbl+lbl>, prtvede1 <dbl+lbl>, prtvede2 <dbl+lbl>, prtvtddk <dbl+lbl>,
## #   prtvtgee <dbl+lbl>, prtvtees <dbl+lbl>, prtvtdfi <dbl+lbl>, prtvtdfr <dbl+lbl>, ...
```

Comme pour d'autres logiciels statistiques, les colonnes représentent les variables et les lignes les observations. Ici, une vingtaine de variables sont affichées (sur 572). De même, les dix premières observations (sur 49519) sont affichées[18].

[17] Vous aurez aussi pu la nommer différemment. Pour cela il suffit de changer le nom à gauche de la flèche (<-).

[18] Cet affichage est le réglage par défaut d'un *tibble*, une forme spéciale de base données dans R. Si vous avez une base de données par défaut (*data.frame*) de R, cela s'affichera un peu différemment. Pour plus de détails en français: https://juba.github.io/tidyverse/06-

Si vous voulez voir l'ensemble de la base de données, vous pouvez utiliser la fonction *View()*[19] :

```
View(ESS9)
```

Si vous voulez enregistrer la base de données sous un format R, utilisez le format *RDS*. Celui-ci vous permet d'enregistrer un objet R. La base de données de l'*ESS* étant chargé sous le nom "*ESS9*" comme objet R dans l'environnement global il vous suffira d'exécuter la commande *saveRDS()* pour l'enregistrer. Dans cette commande vous indiquez le nom de l'objet R à enregistrer en tant que premier argument. Ensuite, vous indiquez, séparé par une virgule, comme second argument, le nom sous lequel vous voulez enregistrer l'objet. Ce nom doit être indiqué entre guillemets. Ici, nous voulons enregistrer l'objet "*ESS9*" sous le nom "ESS9.RDS", afin de le réutiliser lors du prochain chapitre. Exécutez donc le code suivant :

```
saveRDS(ESS9,"ESS9.RDS")
```

Le fichier "ESS9.RDS" contenant votre base de données est alors enregistré dans votre espace de travail. Vous pourrez la charger ensuite pour continuer à travailler avec. C'est ce que nous allons faire dans le prochain chapitre portant sur les distributions.

2.7 Fermer R

Lorsque vous fermez R, *RStudio* vous propose d'enregistrer l'environnement global. Si vous l'acceptez, un fichier contenant tous les objets se trouvant dans votre environnement global sera enregistré dans votre espace de travail. Lorsque vous rouvrirez R plus tard, tous les objets que vous aviez dans votre espace de travail au moment de la fermeture de R seront à nouveau accessibles. Cela vous permettra de reprendre votre travail où vous en étiez avant de fermer R. Si vous n'enregistrez pas l'espace de travail, il sera vierge lorsque vous ouvrez R à nouveau. Il faudra exécuter les commandes de la dernière session pour arriver au même stade qu'avant la fermeture. **Attention :** même si vous enregistrez l'espace de travail, il faudra relancer les *packages* que vous souhaitez utiliser lorsque vous rouvrez R. Lorsque vous ouvrez *RStudio* les fichier *R Script* qui étaient ouverts lorsque vous avez fermé *RStudio* s'ouvrent automatiquement.

tidyverse.html

[19] Mais attention, la base de données étant grande avec près de 500000 observations, cela peut prendre un certain temps.

Chapter 3

Charger des bases de données et les filtrer

Dans ce chapitre, nous allons voir comment charger des bases de données, créer des *Codebook* et filtrer des données. L'objectif est de créer une base de données reprenant seulement les observations belges. Cela nous sera nécessaire pour analyser les salaires en Belgique au prochain chapitre.

Si vous avez éteint R depuis le dernier chapitre, il faudra lancer les *packages* avant de reprendre les analyses. À cette fin, exécutez les commandes suivantes :

```
if (!require("pacman")) install.packages("pacman") #Cela vérifie
                              #si le package pacman est installé.
                              #S'il ne l'est pas, il est installé.

pacman::p_load(tidyverse, descr, rcompanion, codebook,
               DT, sjPlot, labelled) #On lance les packages
```

3.1 Charger des données au format R (*.RDS*)

Pour commencer nos analyses, il faut charger les données. Dans le chapitre précédent, nous avons vu comment importer des données depuis le format *SPSS* (suffixe *.sav*) et comment les enregistrer au format "*RDS*" de R. Nous avions enregistré la base de données de 'l'Enquête sociale européenne / *European social survey (ESS)* "*ESS 9*" sous le nom "*ESS9.RDS*". Nous allons à présent charger ce fichier pour continuer nos analyses avec cette base de données.

Assurez-vous que le fichier "*ESS9.RDS*" se trouve dans votre espace de tra-

vail. S'il n'y figure pas, effectuez les étapes décrites dans le chapitre précédent[1]. Chargez donc le fichier à l'aide de la commande *readRDS()* :

```
ESS9<-readRDS("ESS9.RDS")#Vous nommez la base de données ESS9
```

Lorsque vous chargez une base de données (ou un autre objet R) à l'aide de la fonction "*readRDS()*", il faut lui attribuer un nom sous lequel le charger dans l'espace de travail. Ici nous le faisons en plaçant une flèche (<-) à gauche de la fonction et en indiquant le nom de l'objet, *ESS9*, au bout de cette flèche. Si vous ne nommez pas l'objet, et écrivez simplement la commande sans lui attribuer de nom, la base de données apparaîtra certes dans la sortie de la console, mais ne sera pas enregistrée dans l'environnement global. Vous ne pourrez alors pas faire d'analyse avec cette base de données. Assurez-vous que l'objet est bien chargé en vérifiant s'il apparaît dans l'onglet "*Environment*" à droite dans *RStudio*. Par ailleurs, affichez l'objet en exécutant le code suivant :

```
ESS9 #En exécutant le nom de l'objet, il s'affiche
```

```
## # A tibble: 49,519 x 572
##    name   essro~1 edition prodd~2  idno cntry    dweight pspwght pweight anwei~3 nws
##    <chr>    <dbl> <chr>   <chr>   <dbl> <chr+lb>   <dbl>   <dbl>   <dbl>   <dbl> <dl
##  1 ESS9e~       9 3.1     17.02.~    27 AT [Aus~   0.581   0.218   0.302  0.0659 60
##  2 ESS9e~       9 3.1     17.02.~   137 AT [Aus~   1.06    0.413   0.302  0.125  10
##  3 ESS9e~       9 3.1     17.02.~   194 AT [Aus~   1.38    2.27    0.302  0.686  60
##  4 ESS9e~       9 3.1     17.02.~   208 AT [Aus~   0.993   0.386   0.302  0.117  45
##  5 ESS9e~       9 3.1     17.02.~   220 AT [Aus~   0.377   1.03    0.302  0.312  30
##  6 ESS9e~       9 3.1     17.02.~   254 AT [Aus~   1.48    0.576   0.302  0.174  45
##  7 ESS9e~       9 3.1     17.02.~   290 AT [Aus~   0.992   0.721   0.302  0.218  60
##  8 ESS9e~       9 3.1     17.02.~   301 AT [Aus~   0.310   0.130   0.302  0.0393 30
##  9 ESS9e~       9 3.1     17.02.~   305 AT [Aus~   1.23    1.77    0.302  0.535  30
## 10 ESS9e~       9 3.1     17.02.~   400 AT [Aus~   0.459   0.743   0.302  0.224  25
## # ... with 49,509 more rows, 559 more variables: ppltrst <dbl+lbl>, pplfair <dbl+lbl
## #   pplhlp <dbl+lbl>, polintr <dbl+lbl>, psppsgva <dbl+lbl>, actrolga <dbl+lbl>, psp
## #   cptppola <dbl+lbl>, trstprl <dbl+lbl>, trstlgl <dbl+lbl>, trstplc <dbl+lbl>, trs
## #   trstprt <dbl+lbl>, trstep <dbl+lbl>, trstun <dbl+lbl>, vote <dbl+lbl>, prtvtcat
## #   prtvtdbe <dbl+lbl>, prtvtdbg <dbl+lbl>, prtvtgch <dbl+lbl>, prtvtbcy <dbl+lbl>,
## #   prtvtecz <dbl+lbl>, prtvede1 <dbl+lbl>, prtvede2 <dbl+lbl>, prtvtddk <dbl+lbl>,
## #   prtvtgee <dbl+lbl>, prtvtees <dbl+lbl>, prtvtdfi <dbl+lbl>, prtvtdfr <dbl+lbl>,
```

Si vous n'avez pas enregistré le fichier dans votre espace de travail, mais que vous voulez malgré tout le charger, vous pouvez indiquer le chemin entier en argument comme montré dans la section Espace de travail. Si vous utilisez Windows, vous pouvez aussi utiliser la fonction *choose.files()*, comme montré dans la section Importer et enregistrer des données :

[1] Si vous suivez un enseignement de Lev Lhommeau, il est probable que ce fichier vous ait déjà été fourni

```
ESS9<-readRDS(choose.files())
```

Une fois les données importées, nous pouvons avancer sur les manipulations et les analyses.

3.2 Explorer les données: créer un *codebook* interactif

Nous avons la base de données de l'ensemble du "*round*" 9 de l'Enquête sociale européenne / *European social survey (ESS)*. Quels pays figurent dans cette base de données?

Pour répondre à cette question, il faut savoir dans quelle variable cette information est stockée. Un **codebook** nous permet de répondre à cette question. Un codebook est un document résumant les variables avec leurs noms, leurs libellés et les libellés des variables. Des logiciels tels que *Stata* et *SPSS* proposent d'office cette possibilité. Ainsi *SPSS* propose l'onglet "*Variable View*" qui permet de se faire une idée des variables. À première vue, R semble moins convivial pour afficher les variables. Cependant, rien n'est plus faux : en effet, on peut créer un *codebook* à l'aide du *package codebook*. Ce *codebook* devient interactif, si l'on le combine avec la fonction *datatable* du *package* DT.

Il faut tout d'abord créer le *codebook* avec la fonction *codebook_table*. Nous l'appelons *codebook_ESS9* :

```
codebook_ESS9 <- codebook_table(ESS9)
```

Ensuite, vous rendez ce *codebook* interactif avec la fonction *datatable()*. Ici, vous créez l'objet *codebook_ESS9_DT* :

```
codebook_ESS9_DT <- datatable(codebook_ESS9, options =
list(scrollX = TRUE))
```

Notons que chez vous, il devrait suffire d'exécuter le code ainsi, sans ajouter l'option pour ajouter une barre de défilement horizontale ("*options = list(scrollX = TRUE)*") :

```
codebook_ESS9_DT <- datatable(codebook_ESS9)
```

Vous pouvez afficher ce *codebook* de la manière suivante :

```
codebook_ESS9_DT
```

Vous pouvez chercher des termes et des expressions dans le champ "*Search*". Si vous y cherchez "*country*" (pays), les variables où ce terme apparaît seront affichées : vous verrez apparaître la variable "*cntry*" qui reprend les pays. Quels pays sont représentés dans cette base de données ?

Pour répondre à cette question, il faut sélectionner la variable "*cntry*" et l'analyser. Mais comment sélectionne-t-on une variable ? Dans le R de base, on utilise le signe dollar ($) apposé au nom de la base de données. Cela a pour effet d'extraire cette variable en tant que vecteur. Concrètement, cela se fait ainsi :

```
ESS9$cntry
```

```
## <labelled<character>[49519]>: Country
##    [1] AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT
##   [33] AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT
##   [65] AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT
##   [97] AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT
##  [129] AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT
##  [161] AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT
##  [193] AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT
##  [225] AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT
##  [257] AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT
##  [289] AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT
##  [321] AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT
##  [353] AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT
##  [385] AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT
##  [417] AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT
##  [449] AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT
##  [481] AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT
##  [513] AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT
##  [545] AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT
##  [577] AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT
##  [609] AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT
##  [641] AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT
##  [673] AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT
##  [705] AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT
##  [737] AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT
##  [769] AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT
##  [801] AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT
##  [833] AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT
##  [865] AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT
##  [897] AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT
##  [929] AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT
##  [961] AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT AT
##  [993] AT AT AT AT AT AT AT AT
##  [ reached getOption("max.print") -- omitted 48519 entries ]
##
## Labels:
##  value          label
##     AT        Austria
```

```
##     BE        Belgium
##     BG       Bulgaria
##     CH    Switzerland
##     CY         Cyprus
##     CZ        Czechia
##     DE        Germany
##     DK        Denmark
##     EE        Estonia
##     ES          Spain
##     FI        Finland
##     FR         France
##     GB United Kingdom
##     HR        Croatia
##     HU        Hungary
##     IE        Ireland
##     IS        Iceland
##     IT          Italy
##     LT      Lithuania
##     LV         Latvia
##     ME     Montenegro
##     NL    Netherlands
##     NO         Norway
##     PL         Poland
##     PT       Portugal
##     RS         Serbia
##     SE         Sweden
##     SI       Slovenia
##     SK       Slovakia
```

Pour savoir quels pays sont dans la base de données, on utilise la fonction *table()* qui résume les données sous forme de tableau :

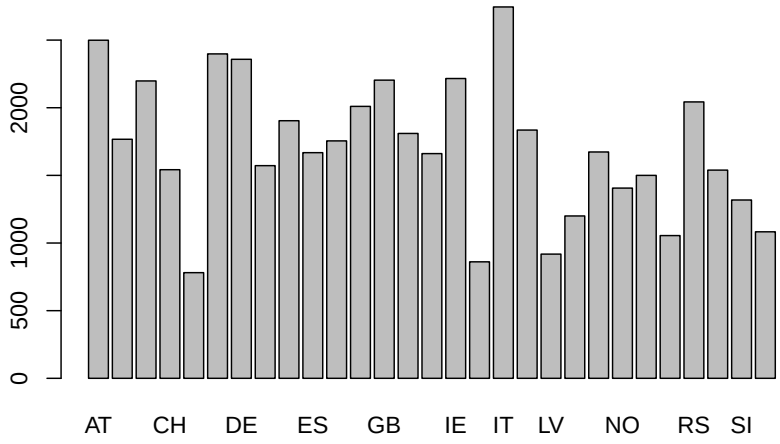
```
table(ESS9$cntry)
```

```
##
##   AT   BE   BG   CH   CY   CZ   DE   DK   EE   ES   FI   FR   GB   HR   HU   IE   IS   IT   LT
## 2499 1767 2198 1542  781 2398 2358 1572 1904 1668 1755 2010 2204 1810 1661 2216  861 2745 1835
##   ME   NL   NO   PL   PT   RS   SE   SI   SK
## 1200 1673 1406 1500 1055 2043 1539 1318 1083
```

Pour obtenir un tableau plus lisible avec un diagramme en barre, vous pouvez utiliser la fonction *freq()* du package *descr* :

```
freq(ESS9$cntry)
```

Cela donne le tableau très pratique qui suit le graphique :




```
## Country
##       Frequency Percent
## AT         2499   5.047
## BE         1767   3.568
## BG         2198   4.439
## CH         1542   3.114
## CY          781   1.577
## CZ         2398   4.843
## DE         2358   4.762
## DK         1572   3.175
## EE         1904   3.845
## ES         1668   3.368
## FI         1755   3.544
## FR         2010   4.059
## GB         2204   4.451
## HR         1810   3.655
## HU         1661   3.354
## IE         2216   4.475
## IS          861   1.739
## IT         2745   5.543
## LT         1835   3.706
## LV          918   1.854
## ME         1200   2.423
## NL         1673   3.379
## NO         1406   2.839
```

```
## PL         1500   3.029
## PT         1055   2.130
## RS         2043   4.126
## SE         1539   3.108
## SI         1318   2.662
## SK         1083   2.187
## Total     49519 100.000
```

On voit qu'il y a un certain nombre de pays, dont la Belgique avec 1767 observations. Mais combien de pays y a-t-il exactement? R peut calculer cela pour vous. R comprend la fonction *unique()* qui vous retourne les valeurs uniques sous forme de vecteur :

```
unique(ESS9$cntry)
```

```
## <labelled<character>[29]>: Country
##  [1] AT BE BG CH CY CZ DE DK EE ES FI FR GB HR HU IE IS IT LT LV ME NL NO PL PT RS SE SI SK
##
## Labels:
##  value          label
##     AT        Austria
##     BE        Belgium
##     BG       Bulgaria
##     CH    Switzerland
##     CY         Cyprus
##     CZ        Czechia
##     DE        Germany
##     DK        Denmark
##     EE        Estonia
##     ES          Spain
##     FI        Finland
##     FR         France
##     GB United Kingdom
##     HR        Croatia
##     HU        Hungary
##     IE        Ireland
##     IS        Iceland
##     IT          Italy
##     LT      Lithuania
##     LV         Latvia
##     ME     Montenegro
##     NL    Netherlands
##     NO         Norway
##     PL         Poland
##     PT       Portugal
##     RS         Serbia
##     SE         Sweden
```

```
##     SI       Slovenia
##     SK       Slovakia
```

Cela vous donne la liste des pays (ou plutôt des valeurs uniques) sous forme d'un nouveau vecteur. Cependant, cela ne vous informe pas sur leur nombre exact. Vous pouvez obtenir ce nombre par la fonction *length()* qui compte le nombre d'éléments dans un vecteur. Vous pouvez l'appliquer soit en imbriquant les deux fonctions ainsi :

```
length(unique(ESS9$cntry))
```

Vous pouvez aussi procéder en deux étapes en créant d'abord l'objet *pays_uniques*, reprenant la liste des pays sous forme de vecteur :

```
pays_uniques<-unique(ESS9$cntry)
pays_uniques
```

```
## <labelled<character>[29]>: Country
##  [1] AT BE BG CH CY CZ DE DK EE ES FI FR GB HR HU IE IS IT LT LV ME NL NO PL PT RS S
##
## Labels:
##  value          label
##     AT        Austria
##     BE        Belgium
##     BG       Bulgaria
##     CH    Switzerland
##     CY         Cyprus
##     CZ        Czechia
##     DE        Germany
##     DK        Denmark
##     EE        Estonia
##     ES          Spain
##     FI        Finland
##     FR         France
##     GB United Kingdom
##     HR        Croatia
##     HU        Hungary
##     IE        Ireland
##     IS        Iceland
##     IT          Italy
##     LT      Lithuania
##     LV         Latvia
##     ME     Montenegro
##     NL    Netherlands
##     NO         Norway
##     PL         Poland
##     PT       Portugal
```

```
##     RS         Serbia
##     SE         Sweden
##     SI       Slovenia
##     SK       Slovakia
```

Puis, vous appliquez la fonction *length()* à l'objet *pays_uniques*. Cela vous donnera le nombre d'objets dans le vecteur *pays_uniques*, c'est-à-dire le nombre de pays dans la base de données *ESS9* :

```
length(pays_uniques)
```

```
## [1] 29
```

Cela nous indique, qu'il y a 29 pays dans la base de données *ESS9*. Attention, la fonction *length()* s'applique aux vecteurs ou listes, mais pas aux bases de données (*data.frame* ou *tibble*) ou matrices (*matrix*). Si nous voulions connaître le nombre d'observations dans la base de données *ESS9*, nous n'appliquons pas la fonction *length()*, mais la fonction *nrow()*. Celle-ci retourne le nombre de lignes (*rows*) dans la base de données, ce qui revient à nous indiquer le nombre d'observations dans la base de données (vu que chaque ligne correspond à une observation) :

```
nrow(ESS9)
```

```
## [1] 49519
```

Nous avons donc 49519 observations dans cette base de données. Mais comme nous l'avons expliqué, nous ne souhaitons pas analyser tous les pays, mais uniquement la Belgique. Pour continuer nos analyses, il faudra donc filtrer la base de données.

3.3 Filtrer les données

Vu que nous voulons analyser les salaires en Belgique, il faut sélectionner uniquement les observations belges. Il faut vérifier quelle valeur dans la variable *cntry* (*Country*) de la base de données *ESS9* (*ESS9$cntry*) correspond à la Belgique. Nous pouvons faire cela en utilisant la fonction *val_labels* du *package labelled* :

```
val_labels(ESS9$cntry)
```

```
##        Austria        Belgium       Bulgaria    Switzerland         Cyprus        Czechia
##           "AT"           "BE"           "BG"           "CH"           "CY"           "CZ"
##        Germany        Denmark        Estonia          Spain        Finland         France
##           "DE"           "DK"           "EE"           "ES"           "FI"           "FR"
## United Kingdom        Croatia        Hungary        Ireland        Iceland          Italy
##           "GB"           "HR"           "HU"           "IE"           "IS"           "IT"
##      Lithuania         Latvia     Montenegro    Netherlands         Norway        Poland
##           "LT"           "LV"           "ME"           "NL"           "NO"           "PL"
```

```
##       Portugal         Serbia         Sweden       Slovenia       Slovakia
##           "PT"           "RS"           "SE"           "SI"           "SK"
```

Nous voyons que "*BE*" correspond à la Belgique (*Belgium*). Il faut donc garder uniquement les observations dont la variable *cntry* (*country*) est égale à "BE".

La fonction de base de R *subset()* permet de filtrer les observations sur un ou plusieurs critères. Nous allons plutôt utiliser la fonction *filter()* du *package dplyr*[2], faisant partie du *tidyverse*, car celle-ci est facilement combinable avec d'autres manipulations.

Pour sélectionner uniquement les observations dont la variable *cntry* est égale à "*BE*", on utilise la fonction *filter()* de cette manière :

```
ESS9_BE<-filter(ESS9,cntry=="BE")
```

L'objet *ESS9_BE* contient maintenant uniquement les cas belges de la base de données. Comme vous le voyez, le premier argument de cette fonction *filter()* est la base de données *ESS9*. Puis, séparé par une virgule, le second argument est la condition à respecter. Ici, la condition est que la variable (colonne) *cntry* doit être égale à "BE".

En lisant ce code, vous aurez certainement remarqué deux choses:

- La première est que "BE" est noté entre guillemets. En effet, lorsque vous filtrez des chaînes de caractère (*string/character*), il faut utiliser des guillemets autour des caractères que l'on souhaite filtrer. Si on n'utilise pas de guillemets, R pense qu'on lui indique un objet ayant pour nom *BE*. Ne connaissant pas cet objet, R vous retournera une erreur. Avec les guillemets, R sait que l'on veut dire la chaîne de caractère "BE". Si, en revanche, pour une autre commande, vous vouliez filtrer un nombre (par exemple: 1, 1000 ou 1.35), vous n'avez pas besoin de ces guillemets.
- la deuxième est que l'on a indiqué un double égal ("=="), pour indiquer que l'on voudrait filtrer des éléments correspondant à ce qu'on indique (ici "BE").

R connaît ainsi un certain nombre d'**opérateurs logiques** :

- == : est égal à
- ! = : est différent de
- !x : n'est pas x
- $\%in\%$: est un des éléments dans la liste
- < : est plus petit que
- <= : est plus petit ou égal à
- > : est plus grand que
- >= : est plus grand ou égal à
- $x\&y$: condition x ET condition y

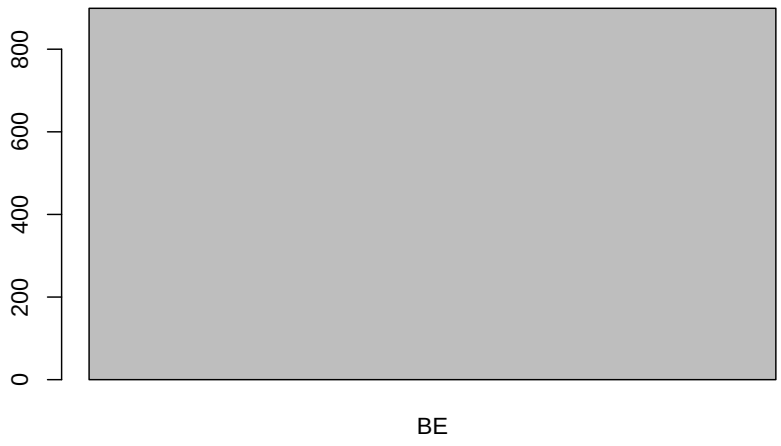
[2]La fonction *subset()* s'utilise de manière similaire à la fonction *filter()*. Vous pouvez donc l'utiliser de manière analogue à la fonction *filter()* présentée ici.

- $x|y$: condition x *OU* condition y (le signe "|", qui se prononce "*barre*" ou "*pipe*" en anglais, est accessible en tapant "Atl Gr" et la touche "1" sur votre clavier belge)

Prenons des exemples pour les deux derniers opérateurs de la liste "&" et "|" :

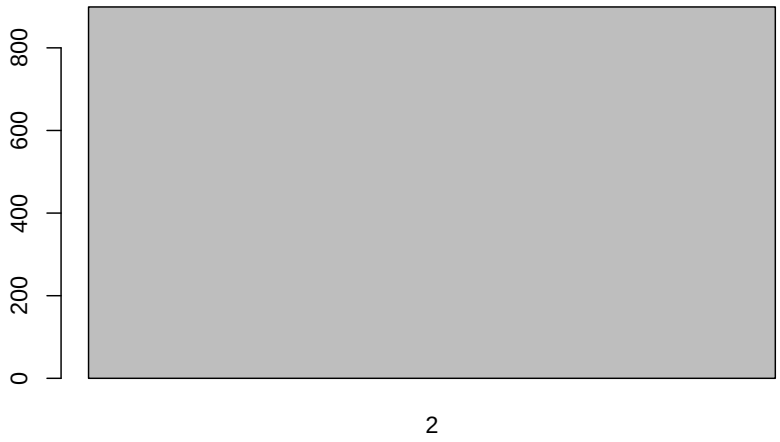
- Imaginons que vous vouliez garder les observations en Belgique, mais uniquement si elles sont des femmes. Dans ce cas, si vous utilisez le lien logique "&" entre les conditions, cela signifie que les deux conditions doivent être remplies simultanément. Les observations sélectionnées doivent concerner la Belgique *ET* les femmes. En d'autres termes, les observations en Belgique qui ne sont pas des femmes (donc les hommes en Belgique) ou les femmes qui ne sont pas en Belgique ne seront pas sélectionnées. Le code serait alors :

```
ESS9_BE_Women<-filter(ESS9,cntry=="BE" & gndr==2)

#Regardez le résultat:
freq(ESS9_BE_Women$cntry)
```




```
## Country
##       Frequency Percent
## BE          899     100
## Total       899     100
```

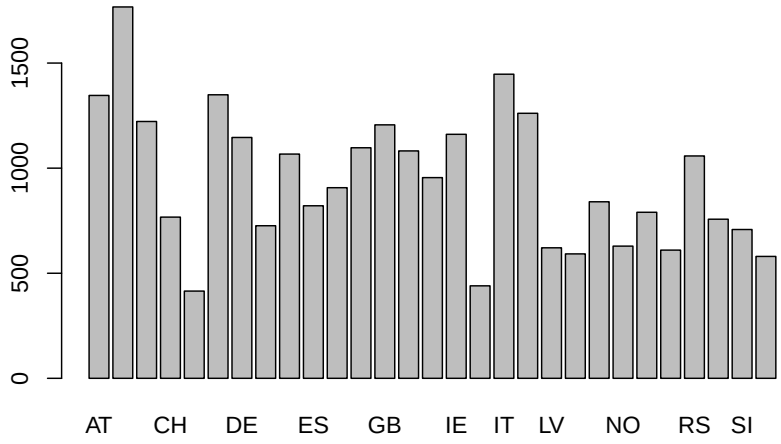
```
freq(ESS9_BE_Women$gndr)
```




```
## Gender
##       Frequency Percent
## 2           899     100
## Total       899     100
```

- Imaginons que vous vouliez garder toutes les observations en Belgique ainsi que toutes les femmes de tous les pays : dans ce cas, vous devriez séparer les deux conditions par le symbole "|". Les observations sélectionnées doivent concerner la Belgique *OU* bien les femmes. Toutes les observations belges sont alors sélectionnées et toutes les observations "femmes" également. Les observations en Belgique qui ne sont pas des femmes seront sélectionnées, tout comme les femmes qui ne sont pas en Belgique. Le code serait alors :

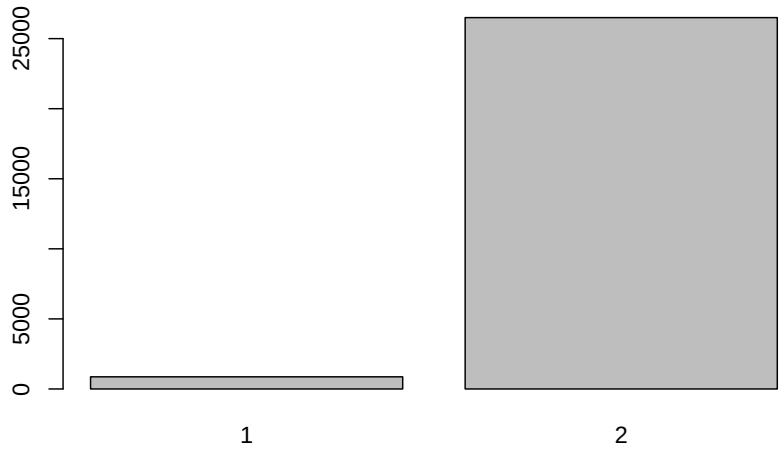
```
ESS9_BE_OR_Women<-filter(ESS9,cntry=="BE" | gndr==2)

#Regardez le résultat:
freq(ESS9_BE_OR_Women$cntry)
```




```
## Country
##       Frequency Percent
## AT         1346   4.918
## BE         1767   6.457
## BG         1222   4.465
## CH          767   2.803
## CY          415   1.516
## CZ         1349   4.929
## DE         1146   4.188
## DK          726   2.653
## EE         1067   3.899
## ES          821   3.000
## FI          907   3.314
## FR         1097   4.008
## GB         1206   4.407
## HR         1082   3.954
## HU          955   3.490
## IE         1161   4.242
## IS          440   1.608
## IT         1447   5.287
## LT         1261   4.608
## LV          621   2.269
## ME          592   2.163
## NL          840   3.069
## NO          629   2.298
```

```
## PL          790   2.887
## PT          610   2.229
## RS         1058   3.866
## SE          757   2.766
## SI          708   2.587
## SK          580   2.119
## Total     27367 100.000
```

```
freq(ESS9_BE_OR_Women$gndr)
```



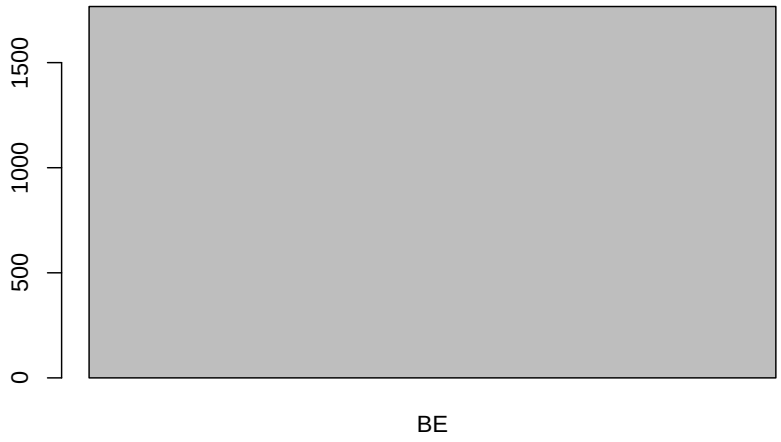

```
## Gender
##       Frequency Percent
## 1           868   3.172
## 2         26499  96.828
## Total     27367 100.000
```

Revenons à notre ojectif initial qui était de sélectionner les données de la Belgique, quel que soit le sexe. Une fois que nous avons exécuté la fonction *filter()* pour garder uniquement les observations en Belgique créant la base de données nommée *ESS9_BE*, nous allons vérifier qu'elle ne contient plus que des observations belges avec la fonction *table()* et *freq()* :

```
ESS9_BE<-filter(ESS9,cntry=="BE")

table(ESS9_BE$cntry) # Attention à bien sélectionner la base de
données ESS9_BE
```

```
##
##   BE
## 1767
```

```
freq(ESS9_BE$cntry) # Attention à bien sélectionner la base de
données ESS9_BE
```




```
## Country
##       Frequency Percent
## BE         1767     100
## Total      1767     100
```

Comme nous le voyons, le filtrage a fonctionné : il n'y a plus que 1767 observations belges. Nous pouvons donc procéder à une analyse descriptive des salaires en Belgique dans le chapitre suivant.

Chapter 4

Le salaire de Belges

Si vous avez fermé R depuis le dernier chapitre, il faudra à nouveau lancer les *packages*. Vous pouvez le faire avec le code suivant :

```
if (!require("pacman")) install.packages("pacman") #Cela vérifie
                            #si le package pacman est installé.
                            #S'il ne l'est pas, il est installé.

pacman::p_load(tidyverse, descr, rcompanion, codebook,
               DT, sjPlot, labelled) #On lance les packages
```

4.1 Statistiques descriptives et simples graphiques des salaires des Belges

Si l'on veut étudier les salaires des Belges il faut continuer à travailler avec la base de données *ESS9_BE* que nous avons créée dans le chapitre précédent qui contient exclusivement les observations belges. Par ailleurs, il faut identifier la variable de la base de données contenant les salaires, *grspnum*. Nous pouvons obtenir des statistiques descriptives de cette variable avec la commande *summary()* :

```
summary(ESS9_BE$grspnum)
```

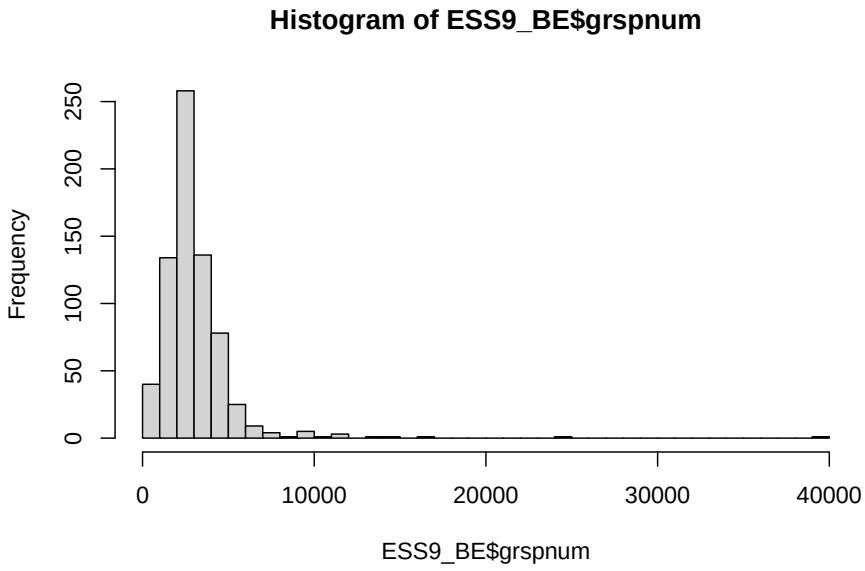
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##       0    2019    2800    3148    3700   40000    1068
```

On voit que la valeur minimale est de 0€, la valeur maximale de 40000€. La moyenne s'élève à 3148€ tandis que la médiane est sensiblement plus basse à 2800€. Cela suggère qu'un nombre restreint de valeurs très élevées tirent la moyenne vers le haut. Il est à noter qu'il y a un nombre important de valeurs

manquantes : 1068, soit 60% sur les 1767.

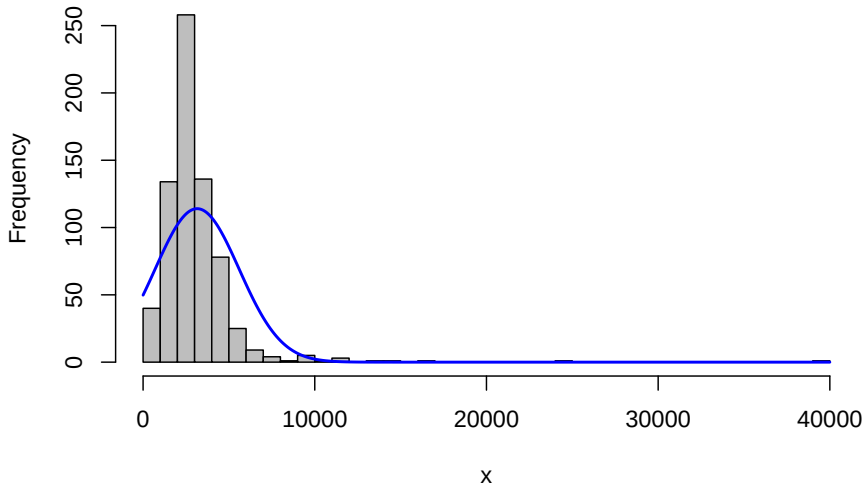
Faisons-nous une image de cette distribution. Vu qu'il s'agit d'une variable continue, nous la représentons sous forme d'histogramme avec la fonction de base *hist()* et la fonction un peu plus avancée *plotNormalHistogram* du *package* rcompanion y ajoutant une courbe de distribution normale ("cloche")[1] :

```
hist(ESS9_BE$grspnum, breaks = 50)
```




```
plotNormalHistogram(ESS9_BE$grspnum,
                    breaks = 50)
```

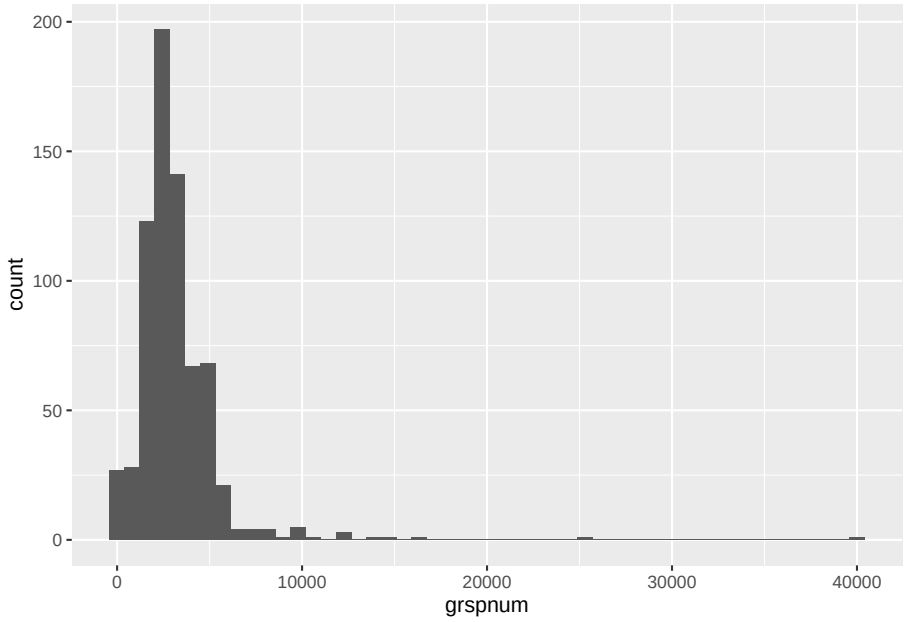
[1]Nous définissions qu'il y a 50 barres dans l'histogramme: "breaks = 50" comme second argument de la fonction *hist()* et *plotNormalHistogram*.



On peut aussi faire de graphiques plus attrayants avec les fonctions du *package ggplot2* de l'ensemble *tidyverse*[2] :

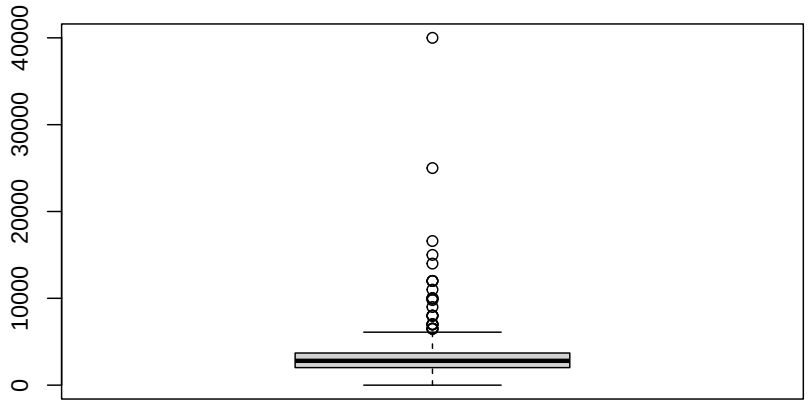
```
ggplot(ESS9_BE, aes(x=grspnum)) +
  geom_histogram(bins=50)
```

[2]Nous définissons qu'il y a 50 barres dans l'histogramme: "bins=50" comme second argument de la fonction *geom_histogram*.



On voit très clairement que la plupart des valeurs sont concentrées entre 0€ et 5000€ et qu'au-delà de 10000 il n'y a plus que quelques observations isolées. Celles-ci tirent la moyenne vers le haut. Un autre moyen de représenter cela est de faire à l'aide d'une "boîte à moustache" (*boxplot*) :

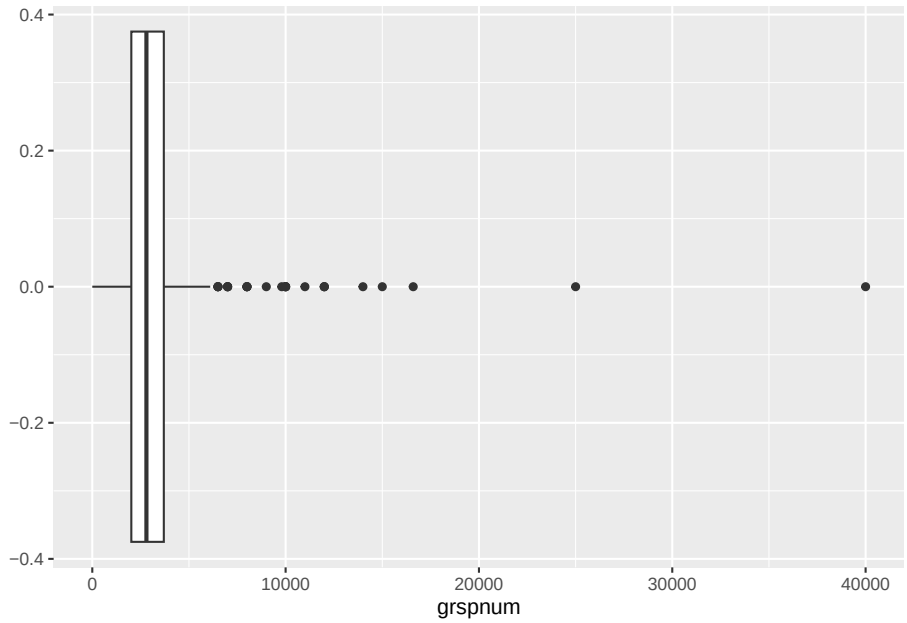
```
boxplot(ESS9_BE$grspnum)
```



Vous pouvez aussi la représenter à l'aide de du *package ggplot2*[3] :

```
ggplot(ESS9_BE, aes(x=grspnum)) +
  geom_boxplot()
```

[3]Si vous comparer le code du *boxplot* à l'histogramme avec *ggplot2*, vous remarquerez que seule la deuxième partie de la commande a changé. Cette similarité du code est un grand avantage du *package ggplot2*. Nous reviendrons plus tard sur la syntaxe et l'utilisation de *ggplot2*.



Les *boxplots* indiquent aussi très clairement qu'il y a quelques valeurs "extrêmes", très élevées, qui "tirent" la moyenne vers le haut. Par ailleurs, il y a un certain nombre d'observations avec des revenus très faibles, tels que 0€.

Il nous semble utile de considérer seulement les personnes travaillant à temps plein. Par ailleurs, il faut contrôler que les données soient cohérentes. La Belgique ayant un salaire minimum, il n'est pas légalement possible pour un salarié de percevoir un salaire de moins de 1000€ tout en travaillant à temps plein[4].

Nous nous concentrons ainsi sur les Belges travaillant plus de 34 heures pas semaine (temps plein) et gagnant plus de 1000€ par mois, mais moins de 15000€ par mois. La variable "*wkhtot*" indique le nombre d'heures travaillées par semaine. On utilise la fonction *filter()* pour sélectionner les observations. On crée une nouvelle base de données nommée *ESS9_BE_fulltime* :

```
ESS9_BE_fulltime<-filter(ESS9_BE, #Base de donnés
                    wkhtot>34 & # Travaillant plus de 34 heures
                      grspnum>1000 & # Percevant plus de 1000€,
                      grspnum<15000) # mais moins de 15.000€.
```

Vous pouvez vérifier cette base de données, notamment en vous assurant qu'il n'y a personne qui travaille moins de 35 heures et perçoit moins de 1000€ et plus de 15000€. Vous pouvez utiliser la fonction *summary()* pour vérifier que

[4]Il est certainement possible qu'il y ait des cas très spécifiques où cela puisse se produire. Mais nous excluons ces cas de nos analyses.

les variables *grspnum* et *wkhtot* dans la base de données *ESS9_BE_fulltime* ne contiennent plus que des valeurs compatibles avec le filtre que l'on vient d'appliquer :

```
summary(ESS9_BE_fulltime$grspnum)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1200    2400    2986    3365    4000   14000
```

```
summary(ESS9_BE_fulltime$wkhtot)
```

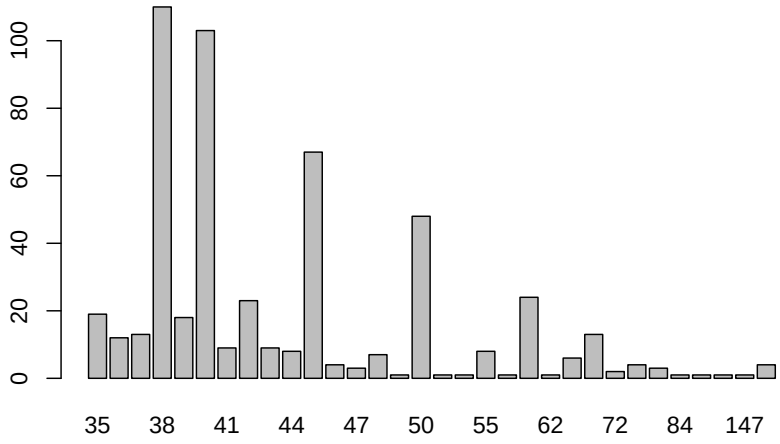
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   35.00   38.00   40.00   45.58   46.00  168.00
```

Le filtrage a fonctionné : dans notre échantillon, personne ne travaille moins de 35 heures et personne ne perçoit moins de 1000€ (le minimum est de 1200€) et plus de 15000€ (le maximum est 14000€). Cependant, des anomalies apparaissent au niveau du temps de travail. On s'en rend compte avec les fonctions *table()* et *freq()* :

```
table(ESS9_BE_fulltime$wkhtot)
```

```
##
##  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  52  53  55  56  60  62  65  7
##  19  12  13 110  18 103   9  23   9   8  67   4   3   7   1  48   1   1   8   1  24   1   6  1
##  75  80  84 100 105 147 168
##   4   3   1   1   1   1   4
```

```
freq(ESS9_BE_fulltime$wkhtot)
```



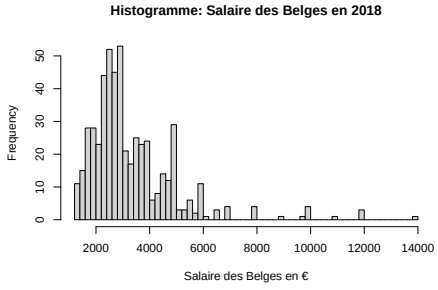

```
## Total hours normally worked per week in main job overtime included
##       Frequency  Percent
## 35           19   3.6122
## 36           12   2.2814
## 37           13   2.4715
## 38          110  20.9125
## 39           18   3.4221
## 40          103  19.5817
## 41            9   1.7110
## 42           23   4.3726
## 43            9   1.7110
## 44            8   1.5209
## 45           67  12.7376
## 46            4   0.7605
## 47            3   0.5703
## 48            7   1.3308
## 49            1   0.1901
## 50           48   9.1255
## 52            1   0.1901
## 53            1   0.1901
## 55            8   1.5209
## 56            1   0.1901
## 60           24   4.5627
## 62            1   0.1901
## 65            6   1.1407
```

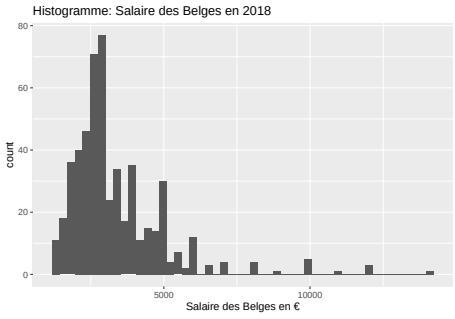
```
## 70            13   2.4715
## 72             2   0.3802
## 75             4   0.7605
## 80             3   0.5703
## 84             1   0.1901
## 100            1   0.1901
## 105            1   0.1901
## 147            1   0.1901
## 168            4   0.7605
## Total        526 100.0000
```

Ainsi 4 répondants affirment travailler 168 heures par semaine. Vu qu'une semaine compte 168 heures et que l'être humain a un besoin physiologique de sommeil, ces réponses semblent aberrantes[5]. Vu que notre analyse ne porte pas sur le temps de travail, nous n'allons ni investiguer ces cas plus en détail, ni les exclure de nos analyses.

La distribution de la variable (après filtration) se rapproche-t-elle d'une distribution normale ? Analysons cela à nouveau avec les histogrammes et "boxplots" :

```
#Avec R de base:

hist(ESS9_BE_fulltime$grspnum,
     breaks = 50,
     main="Histogramme: Salaire des Belges en 2018",
     xlab="Salaire des Belges en €")

#Avec ggplot2:

ESS9_BE_fulltime %>% ggplot(aes(x=grspnum)) +
  geom_histogram(bins = 50)+
  ggtitle("Histogramme: Salaire des Belges en 2018") +
  xlab("Salaire des Belges en €")
```

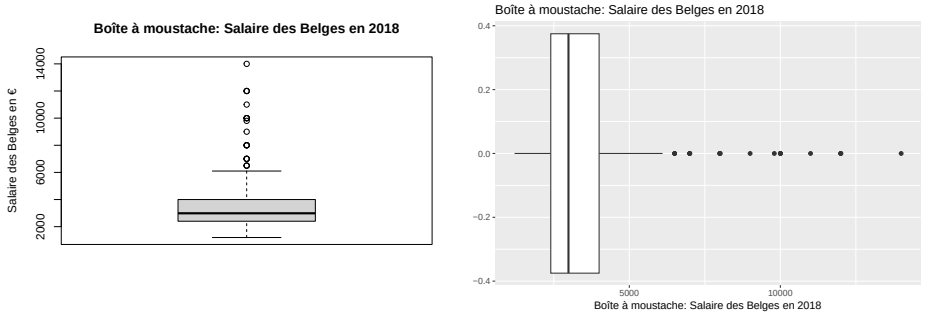




[5] Il faudrait analyser plus en détail ces cas anormaux: peut-être que les personnes concernées exercent des professions en tant qu'indépendants et s'estiment "de garde" en permanence.

Avec des *boxplot*, la variable *grspnum* donne le résultat suivant :

```
#Avec R de base:

boxplot(ESS9_BE_fulltime$grspnum,
        ylab = "Salaire des Belges en €",
     main = "Boîte à moustache: Salaire des Belges en 2018")

#Avec ggplot2:

ESS9_BE_fulltime %>% ggplot(aes(x=grspnum)) +
  geom_boxplot() + ggtitle("Boîte à moustache: Salaire des Belges
  en 2018")+
  xlab("Boîte à moustache: Salaire des Belges en 2018")
```



L'histogramme et la boîte à moustache montrent une distribution moins "écrasée". Elle reste cependant asymétrique avec des valeurs très élevées – et une "coupure" vers le bas à 1200€. La distribution semble donc toujours loin d'être normale.

4.2 Comparer une distribution à une distribution normale

Visuellement, la distribution de la variable des salaires (*grspnum*) ne semble pas normale. Essayons d'objectiver cela. Nous rappelons quelques caractéristiques d'une distribution normale :

- la distribution est symétrique ;
- moyenne = médiane = mode ;
- $\approx 68\%$ des observations se trouvent dans l'intervalle entre -1 écart type (σ) et $+1$ écart type (σ) de la moyenne (μ) : $[\mu - \sigma; \mu + \sigma]$
- $\approx 95\%$ des observations se trouvent dans l'intervalle entre $-1,96$ écart type (σ) et $+1,96$ écart type (σ) de la moyenne (μ) : $[\mu - 1,96\sigma; \mu + 1,96\sigma]$
- $\approx 95,5\%$ des observations se trouvent dans l'intervalle entre -2 écart type (σ) et $+2$ écart type (σ) de la moyenne (μ) : $[\mu - 2\sigma; \mu + 2\sigma]$

- $\approx 2,5\%$ des observation sont supérieures à $1,96$ écart type (σ) de la moyenne (μ) : $[\mu + 1,96\sigma; +\infty)$
- $\approx 2,5\%$ des observation inférieurs à $-1,96$ écart type (σ) de la moyenne (μ) : $[\mu - 1,96\sigma; -\infty)$

Nous allons créer une nouvelle fonction, *nonantecinq()*, pour comparer la distribution d'une variable avec les caractéristiques d'une distribution normale. Pour définir une nouvelle fonction, il faut utiliser la fonction *fonction()*[6] :

```
nonantecinq<-function(x){
  #On exclut les valeurs manquantes de la variable analysée
  x1<-x[is.na(x)!=T]
  #On calcule l'effectif total
  x1_l1<-length(x[is.na(x)!=T])
  #On calcule l'effectif se situant à -1,96 écart types de la
  moyenne
  x1_low<-length(x1[x1 < mean(x1, na.rm=T)-1.96*sd(x1, na.rm=T)])
  #On calcule l'effectif se situant à +1,96 écart types de la
  moyenne
  x1_high<-length(x1[x1 > mean(x1, na.rm=T)+1.96*sd(x1,
na.rm=T)])
  #On calcule l'effectif se situant à -1 écarts types de la
  moyenne (point d'inflexion si normal)
  x1_low_inflex<-length(x1[x1 < mean(x1, na.rm=T)-sd(x1,
na.rm=T)])
  #On calcule l'effectif se situant à +1 écart types de la
  moyenne (point d'inflexion si normal)
  x1_high_inflex<-length(x1[x1 > mean(x1, na.rm=T)+sd(x1,
na.rm=T)])

  #Pourcentages:
  pourcent_bas<-x1_low/x1_l1
  pourcent_haut<-x1_high/x1_l1
  pourcent_1.96<-(1-(pourcent_bas+pourcent_haut))*100
  pourcent_inflex_bas<-x1_low_inflex/x1_l1
  pourcent_inflex_haut<-x1_high_inflex/x1_l1
  pourcent_1_inflex<-(1- (pourcent_inflex_bas +
pourcent_inflex_haut))*100

  #Mode(s)
  mode_func <- function(x_1) {
    modes_multi<-which(table(x_1)==max(table(x_1)))
```

[6]Vu qu'il ne s'agit pas d'une introduction à la programmation avec R, nous ne nous attardons pas sur l'écriture de fonctions dans R. Nous renvoyons à l'ouvrage "Introduction à la programmation en R" de Vincent Goulet et plus particulièrement au chapitre 5: "Fonctions définies par l'usager".

```
    mode_final<-as.numeric(names(modes_multi))
    return(mode_final)
  }

  mode_calc<-mode_func(x)

  #Affichage des résultats
  message(paste(" Moyenne=",round(mean(x1, na.rm=T),2),"\n",
                "Écart type=",round(sd(x1, na.rm=T),2),"\n",
                "Médiane=",round(median(x1, na.rm=T),2),"\n",
                "Mode (si multiples, le plus
                faible)=",ifelse(length(mode_calc)>1,
                round(mode_calc[1],2), round(mode_calc,2)),"\n",
                "Le point d'inflexion (si la distribution est
                normale) à -1 écart type prend la valeur:",
                round(mean(x1, na.rm=T)-sd(x1, na.rm=T),2),"\n",
                "Le point d'inflexion (si la distribution est
                normale) à +1 écart type prend la valeur:",
                round(mean(x1, na.rm=T)+sd(x1, na.rm=T),2),"\n",
                round(pourcent_1_inflex,2),"%"," des valeurs se
                situent entre -1 et +1 écart types de la
                moyenne.", "\n",
                round(pourcent_inflex_bas*100,2),"%"," des
                valeurs se situent en dessous de -1 écart types
                de la moyenne.","\n",
                round(pourcent_inflex_haut*100,2),"%"," des
                valeurs se situent au-dessus de 1 écart types de
                la moyenne.","\n",
                "-1,96 écarts types prend la valeur:",
                round(mean(x1, na.rm=T)-1.96*sd(x1, na.rm=T),2),
                "\n",
                "+1,96 écarts types prend la valeur:",
                round(mean(x1, na.rm=T)+1.96*sd(x1, na.rm=T),2),
                "\n",
                round(pourcent_1.96,2),"%", " des valeurs se
                situent entre -1,96 et +1,96 écart types de la
                moyenne.", "\n",
                round(pourcent_bas*100,2),"%"," des valeurs se
                situent en dessous de -1,96 écart types de la
                moyenne.","\n",
                round(pourcent_haut*100,2),"%", " des valeurs se
                situent au-dessus de 1,96 écart types de la
                moyenne.")
  )
}
```

Une fois que vous avez exécuté le code ci-dessus, la fonction *nonantecinq()* apparaîtra dans l'environnement global de R. Si vous fermez R et que vous n'enregistrez pas cet environnement global, vous devrez récréer la fonction *nonantecinq()* avec le code ci-dessus. Lorsque la fonction *nonantecinq()* est définie, vous pouvez la lancer comme toutes les fonctions de R de base ou des *packages*. Vous indiquez simplement le nom de la fonction avec les parenthèses dans lesquelles vous ajoutez, le cas échéant le ou les arguments. La fonction *nonantecinq()* ne prend qu'un seul argument qui doit être un vecteur numérique (une variable contenant des nombres). Ici, nous voulons vérifier le salaire des Belges (*grspnum*) :

```
nonantecinq(ESS9_BE_fulltime$grspnum)
```

```
##  Moyenne= 3365.44
##  Écart type= 1673.51
##  Médiane= 2986
##  Mode (si multiples, le plus faible)= 3000
##  Le point d'inflexion (si la distribution est normale) à -1 écart type prend la valeur: 1691.9
##  Le point d'inflexion (si la distribution est normale) à +1 écart type prend la valeur: 5038.9
##  85.36 %  des valeurs se situent entre -1 et +1 écart types de la moyenne.
##  5.51 %  des valeurs se situent en dessous de -1 écart types de la moyenne.
##  9.13 %  des valeurs se situent au-dessus de 1 écart types de la moyenne.
##  -1,96 écarts types prend la valeur: 85.35
##  +1,96 écarts types prend la valeur: 6645.53
##  96.39 %  des valeurs se situent entre -1,96 et +1,96 écart types de la moyenne.
##  0 %  des valeurs se situent en dessous de -1,96 écart types de la moyenne.
##  3.61 %  des valeurs se situent au-dessus de 1,96 écart types de la moyenne.
```

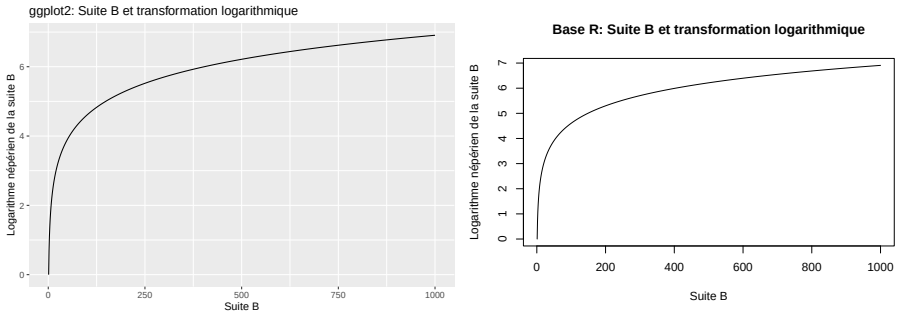
On voit donc également de manière "objective" que la variable salaire (*grspnum*) n'est absolument pas normale :

- La distribution n'est pas symétrique: $5,51\%$ des valeurs se situent en dessous de -1 écart type de la moyenne, tandis que $9,13\%$ des valeurs se situent au-dessus de 1 écart type de la moyenne.
- La moyenne, la médiane et le mode sont différents. En particulier, la moyenne est plus élevée que le mode et la médiane.
- $\approx 85\%$ des observations se trouvent entre -1 et +1 écart type (contre 68% pour une distribution normale)
- 0% des observations se trouvent en dessous de $1,96$ écart type (contre $2,5\%$ pour une distribution normale)
- $3,6\%$ des observations se trouvent au-dessus de $1,96$ écart type (contre $2,5\%$ pour une distribution normale)

4.3 Distribution normale et transformation logarithmique

La distribution de la variable des salaires (*grspnum*) est très éloignée d'une distribution normale. Quelques observations très élevées "tirent" la moyenne vers le haut. Cela est habituel pour les variables de type salaire ou patrimoine. Cependant, on peut transformer ces variables de manière à les rapprocher d'une distribution normale. Une transformation très usitée pour cela est la transformation logarithmique (on prend souvent le logarithme népérien). Cela a pour effet de davantage espacer les valeurs faibles tout en réduisant les espaces entre les valeurs élevées. Illustrons cela graphiquement à l'aide une suite B allant de 1 à 1000 :

```
suite_B<-1:1000 # Créons vecteur de 1 à 1000
log_suite_B<-log(suite_B) # Vecteur du logarithme de la suite B
suite_B_matrix<-cbind(suite_B,log_suite_B) # Nous créons une
matrice avec les deux vecteurs
suite_B_df<-as.data.frame(suite_B_matrix) # Nous la transformons
en base de données (dataframe)
suite_B_df %>% # nous utilisions une "pipe"
  ggplot(aes(x=suite_B,y=log_suite_B)) +
  geom_line() + xlab("Suite B") +
  ylab("Logarithme népérien de la suite B") +
  ggtitle("ggplot2: Suite B et transformation logarithmique")

plot(suite_B_df$suite_B, suite_B_df$log_suite_B, type = "l",
     main="Base R: Suite B et transformation logarithmique",
     xlab = "Suite B",
     ylab="Logarithme népérien de la suite B")
```



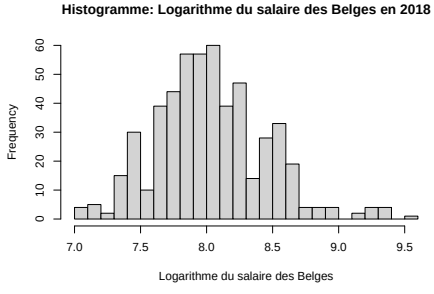
En abscisse, on voit la suite B (l'objet *suite_B*) allant de 1 à 1000. En ordonné, nous avons cette même suite B transformée de manière logarithmique. On voit très clairement que la courbe augmente fortement lors de faibles valeurs puis s'aplanit. Cela correspond à l'augmentation de l'espacement des valeurs faibles et au rapprochement des grandes valeurs.

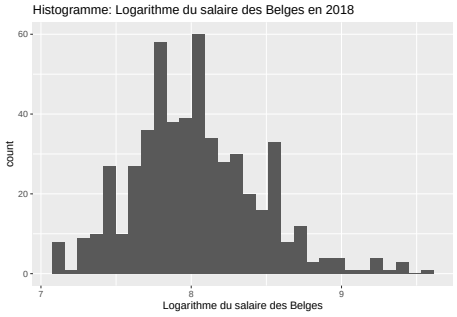
Nous allons donc appliquer cette transformation à la variable *grspnum* reprenant les salaires. Pour effectuer cette transformation, nous utilisons la fonction *mutate()* du *package dplyr* : elle permet de créer de nouvelles variables dans une base de données. Cette fonction s'utilise de la manière suivante : vous indiquez en premier argument la base de donnés dans laquelle vous souhaitez ajouter une variable; et en deuxième argument le nom de cette nouvelle variable suivi du signe = et de la (des) valeur(s) que doit prendre cette nouvelle variable. On indique ainsi que cette nouvelle variable doit correspondre au *logarithme népérien* de la variable *grspnum*. Le logarithme népérien se calcule avec la fonction *log()* :

```
ESS9_BE_fulltime <-
mutate(ESS9_BE_fulltime,log_grspnum=log(grspnum))
```

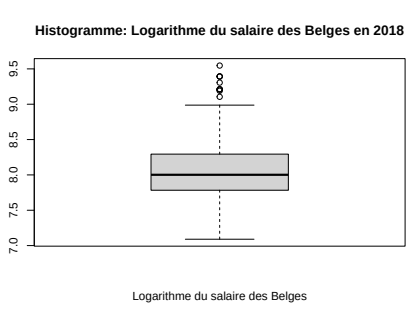
Regardons, dans quelle mesure la distribution a changée et si l'on s'est rapproché d'une distribution normale. Vérifions à l'aide d'un histogramme et d'un *boxplot* :

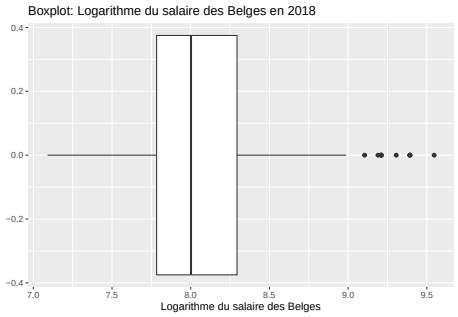
```
#Avec R de base:

hist(ESS9_BE_fulltime$log_grspnum,
     breaks = 30,
   xlab = "Logarithme du salaire des Belges", # Libellé de
   l'abscisse.
     main = "Histogramme: Logarithme du salaire des Belges en
     2018") # Titre du graphique

#Avec ggplot2:

ESS9_BE_fulltime %>% ggplot(aes(x=log_grspnum)) +
  geom_histogram(bins = 30) +
  ggtitle("Histogramme: Logarithme du salaire des Belges en
  2018") +
  xlab("Logarithme du salaire des Belges")
```





```
#Avec R de base:

boxplot(ESS9_BE_fulltime$log_grspnum,
   xlab = "Logarithme du salaire des Belges", # Libellé de
   l'abscisse.
     main = "Histogramme: Logarithme du salaire des Belges en
     2018") # Titre du graphique

#Avec ggplot2:

ESS9_BE_fulltime %>% ggplot(aes(x=log_grspnum)) +
  geom_boxplot() +
  ggtitle("Boxplot: Logarithme du salaire des Belges en 2018") +
  xlab("Logarithme du salaire des Belges")
```

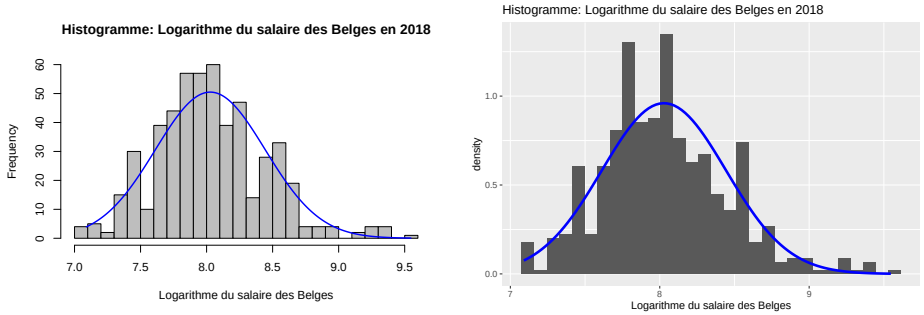




La distribution est sensiblement moins étirée et ressemble plus à une distribution normale. Comparons l'histogramme avec une courbe de densité d'une distribution normale :

```
#Avec R de base:

plotNormalHistogram(ESS9_BE_fulltime$log_grspnum,
     breaks = 30,
   xlab = "Logarithme du salaire des Belges", # Libellé de
   l'abscisse.
     main = "Histogramme: Logarithme du salaire des Belges en
     2018") # Titre du graphique

#Avec ggplot2:

ESS9_BE_fulltime %>% ggplot(aes(x=log_grspnum)) +
  geom_histogram(aes(y =..density..), bins = 30) +
  stat_function(fun = dnorm,
                args = list(mean =
                mean(ESS9_BE_fulltime$log_grspnum),
```

```
                sd = sd(ESS9_BE_fulltime$log_grspnum)),
                col="blue", size=1.25) +
  ggtitle("Histogramme: Logarithme du salaire des Belges en
  2018") +
  xlab("Logarithme du salaire des Belges")
```



Vérifions cela à l'aide de la fonction *nonantecinq* :

```
nonantecinq(ESS9_BE_fulltime$log_grspnum)
```

```
##  Moyenne= 8.03
##  Écart type= 0.42
##  Médiane= 8
##  Mode (si multiples, le plus faible)= 8.01
##  Le point d'inflexion (si la distribution est normale) à -1 écart type prend la valeur: 7.61
##  Le point d'inflexion (si la distribution est normale) à +1 écart type prend la valeur: 8.44
##  67.3 %  des valeurs se situent entre -1 et +1 écart types de la moyenne.
##  15.78 %  des valeurs se situent en dessous de -1 écart types de la moyenne.
##  16.92 %  des valeurs se situent au-dessus de 1 écart types de la moyenne.
##  -1,96 écarts types prend la valeur: 7.21
##  +1,96 écarts types prend la valeur: 8.84
##  94.68 %  des valeurs se situent entre -1,96 et +1,96 écart types de la moyenne.
##  1.71 %  des valeurs se situent en dessous de -1,96 écart types de la moyenne.
##  3.61 %  des valeurs se situent au-dessus de 1,96 écart types de la moyenne.
```

Les données numériques sont tout autant instructives que le diagnostic visuel :

- Le mode, la médiane et la moyenne sont tous très proches de 8 (et donc presque identiques).
- 67% des observations se trouvent entre -1 et $+1$ écart type (contre $\approx 68\%$ d'une distribution normale).
- $\approx$ 95% des observations se trouvent entre $-1,96$ et $+1,96$ écart types (comme pour une distribution normale).
- 2% des observations se trouvent en dessous de $-1,96$ écart types et 4% se trouvent au-dessus de $+1,96$ écart types (contre $2,5\%$ en dessous et au-dessus dans une distribution normale).

Empiriquement, nous ne sommes pas face à une distribution tout à fait normale, mais nous en sommes très proches. Cela d'autant plus que le salaire (*grspnum*, d'où *log_grspnum*) est une variable continue, que la réponse est une déclaration spontanée (les sondés auront tendance à donner des chiffres "ronds")[7] et que l'échantillon est assez grand ($n = 526$). Il y a donc une très forte probabilité que le logarithme de la variable sous-jacente dans la population suive une distribution normale. On peut donc traiter la variable salaire transformée (*log_grspnum*) comme une distribution normale. Cela nous permet à calculer des probabilités comme nous allons le voir dans les prochaines sections.

4.4 Distribution normale et probabilités

La distribution normale a un certain nombre de propriétés très intéressantes. L'une d'entre elles et que lorsqu'on connait la moyenne et la variance (ou l'écart type qui est la racine carrée de la variance) d'une distribution normale, on peut facilement connaître la probabilité d'obtenir une valeur dans un certain intervalle. On se sert de cette propriété dans la statistique inférentielle (qui vise à apporter des connaissances sur la population et pas seulement sur l'échantillon), car les moyennes d'échantillons d'assez grande taille ($n > 30$) tirés d'une même population suivent une loi normale (même si la variable étudiée ne suit pas une distribution normale). Cela permet de calculer des intervalles de confiance et le risque de faux positifs (*p-valeur*) et de tirer des conclusions sur la population dont est issu l'échantillon[8].

4.5 Connaître la probabilité des salaires

La variable *log_grspnum* que l'on a créée précédemment reprend la transformation logarithmique du revenu mensuel des Belges travaillant à temps plein (> 34 heures de travail), sans les valeurs aberrantes (trop faibles pour un travail à temps plein) et des revenus dépassant les 15.000€. Cette variable *log_grspnum* peut être considérée comme suivant une distribution normale. Par conséquent, on peut facilement déterminer la probabilité qu'un Belge travaillant à temps plein touche un salaire dans un certain intervalle. On peut, par exemple, déterminer quelle est la probabilité de bénéficier d'un revenu supérieur à 3000€, inférieur à 1000€ ou bien situé entre 1350€ et 1400€.

[7] Si l'on avait accès aux données de revenus comme l'a l'administration fiscale (en Belgique: SPF Finances) et les organismes de la sécurité sociale (en Belgique notamment par le biais de la BCSS), la distribution serait vraisemblablement encore plus proche de la distribution normale.

[8] Pour les concepts d'échantillon et de population, d'intervalles de confiance, de *p-valeur* nous renvoyons aux ouvrages d'introduction aux statistiques. Notons que contrairement à des formulations que l'on peut trouver ça et là, les "vraies" valeurs dans la population sont et restent toujours inconnues. Les estimations que l'on peut obtenir par des procédés mathématiques ne permettent pas de connaître cette "vraie" valeur. L'on peut simplement connaître la probabilité d'obtenir une valeur dans l'intervalle calculée si l'on prend un nouvel échantillon dans la population.

R permet de calculer cette probabilité de manière très simple à l'aide de la fonction *pnorm()*. Celle-ci permet de calculer la probabilité qu'une valeur se trouve dans un certain intervalle, en indiquant la valeur dont on veut connaitre la probabilité, la moyenne, l'écart type et si l'on cherche l'intervalle supérieur ou inférieur à la valeur recherchée. R permet, par ailleurs, de représenter graphiquement ces intervalles.

Notre exemple comporte une difficulté supplémentaire, car nous avons transformé le salaire avec le logarithme népérien. En effet, nous ne pouvons pas directement indiquer l'intervalle de revenu dont nous souhaitons connaître la probabilité. Il faut que nous transformions le salaire que nous voulons tester à l'aide du logarithme népérien[9]. Vu que R permet d'imbriquer des fonctions, on peut assez facilement surmonter cette difficulté.

Si vous nous voulons connaitre la probabilité de bénéficier d'un revenu supérieur à 2500€, nous pouvons le faire ainsi :

```
pnorm(log(2500), # Le logarithme népérien de 2500
      mean = mean(ESS9_BE_fulltime$log_grspnum), #la moyenne de
      la variable log_grspnum
      sd=sd(ESS9_BE_fulltime$log_grspnum),#l'écart type de la
      variable log_grspnum
      lower.tail = F # F pour "False", car nous voulons connaitre
      la probabilité que la valeur soit au-dessus de 2500
      )
```

```
## [1] 0.6882183
```

La probabilité de percevoir un salaire supérieur à 2500€ est exprimé en probabilité: $0,69$, ce qui correspond à 69%.

Pour calculer la probabilité de percevoir un salaire inférieur à 2500€, on utilise le code suivant :

```
pnorm(log(2500), # Le logarithme népérien de 2500
      mean = mean(ESS9_BE_fulltime$log_grspnum), #la moyenne de
      la variable log_grspnum
      sd=sd(ESS9_BE_fulltime$log_grspnum),#l'écart type de la
      variable log_grspnum
      lower.tail = T # T pour "True" car nous voulons connaitre
      la probabilité que la valeur soit en dessous de 2500
      )
```

```
## [1] 0.3117817
```

La probabilité de percevoir un salaire inférieur à 2500€ est donc de $0,31$ soit 31%. Vous aurez noté que la probabilité cumulé de percevoir un salaire inférieur

[9]Attention: Si nous travaillons avec des données qui ne sont pas transformées, il n'y pas besoin de faire une telle transformation!

à 2500€ et supérieur à 2500€ est de 1 ($0,69 + 0,31 = 1$). On peut utiliser cela pour calculer la probabilité de percevoir un salaire dans un certain intervalle. Si l'on voulait connaître la probabilité de percevoir un salaire supérieur à 2500€, mais inférieur à 2800€, on calcule simplement:

```
#La probabilité de bénéficier d'un salaire inférieur à 2800 €
#On l'appelle proba_inf_2800
proba_inf_2800<-pnorm(log(2800), # Le logarithme népérien de 2800
      mean = mean(ESS9_BE_fulltime$log_grspnum), #la moyenne de
      la variable log_grspnum
      sd=sd(ESS9_BE_fulltime$log_grspnum),#l'écart type de la
      variable log_grspnum
      lower.tail = T #Nous voulons connaitre la probabilité que
      la valeur soit en dessous de 2800
      )
proba_inf_2800
```

```
## [1] 0.413685
```

```
#La probabilité de bénéficier d'un salaire inférieur à 2500 €
#On l'appelle proba_inf_2500
proba_inf_2500<-pnorm(log(2500), # Le logarithme népérien de 2500
      mean = mean(ESS9_BE_fulltime$log_grspnum), #la moyenne de
      la variable log_grspnum
      sd=sd(ESS9_BE_fulltime$log_grspnum),#l'écart type de la
      variable log_grspnum
      lower.tail = T #Nous voulons connaitre la probabilité que
      la valeur soit en dessous de 2500
      )
proba_inf_2500
```

```
## [1] 0.3117817
```

```
#La différence entre la probabilité
#de bénéficier d'un salaire supérieur à 2500€ (proba_inf_2500) et
#la probabilité de bénéficier d'un salaire inférieur à 2800 €
(proba_inf_2800):
proba_combi_2500_2800<-proba_inf_2800-proba_inf_2500

#Cette différence donne:
proba_combi_2500_2800
```

```
## [1] 0.1019033
```

La probabilité de bénéficier d'un salaire entre 2500€ et 2800€ est donc de $0,1$ soit 10%. En effet, la probabilité de bénéficier d'un salaire de moins de 2800€ est de $0,41$ (41%). La probabilité de bénéficier d'un salaire de moins de 2500€ est de $0,31$ (31%). D'où, en prenant la différence entre les deux, une probabilité

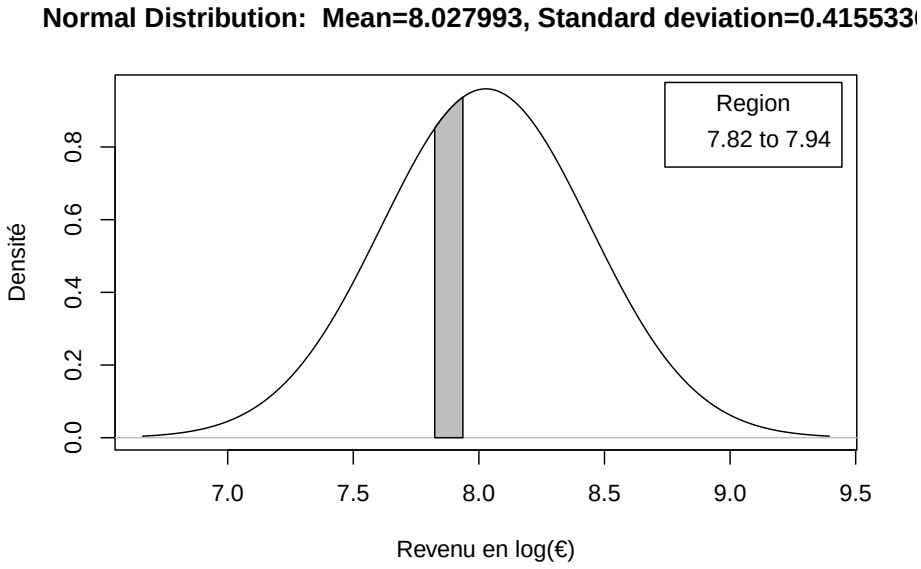
de bénéficier d'un salaire entre 2500€ et 2800€ $(0,41 - 0,31 = 0,1)$.

Pour représenter cela de manière visuelle, nous utiliserons le code généré par l'interface graphique "*R Commander*" (*Rcmdr*)[10]. D'autres manières de représenter cela impliquent de combiner, voir d'imbriquer, des fonctions telles que polygon()/geom_polygon(), dnorm() et éventuellement rev() et rep() qui nous semble trop complliqué à expliquer dans ce syllabus. Pour utiliser les fonctions issues du *package Rcmdr*, lancez le ainsi :

Ensuite, vous pouvez soit faire les manipulations dans "R Commander", soit utiliser le code dans le *R Script* . Si vous souhaiter utiliser le code dans le R Script, je vous conseille de fermer la fenêtre du "R Commander" pour que les graphiques générés n'apparaissent pas dans la fenêtre du "R Commander" mais à leur emplacement habituel :

```
.x <- seq(6.661, 9.395, length.out=1000)
  plotDistr(.x,
            dnorm(.x,
                  mean=mean(ESS9_BE_fulltime$log_grspnum),
                  sd=sd(ESS9_BE_fulltime$log_grspnum)),
            cdf=FALSE,
            xlab="Revenu en log(€)",
            ylab="Densité",
  main=paste("Normal Distribution:  Mean=8.027993, Standard
  deviation=0.4155336"),
  regions=list(c(log(2500), log(2800))),
  col=c('#BEBEBE', '#BEBEBE'),
  legend.pos='topright')
```

[10]Ce syllabus ne s'attarde pas sur le fonctionnement de cet interface graphique. Vous trouverez des explications en ligne, notamment: https://lms.fun-mooc.fr/c4x/UPSUD/42001S02/asset/introRcmdr.html



Grâce à ce graphique on voit immédiatement où se situent les 10% de probabilité de percevoir entre 2500€ et 2800€.

Pour calculer le montant à partir duquel on perçoit un revenu supérieur à 80% de la population, ou bien à partir duquel on appartient à la catégorie des Belges percevant les 10% de revenus les plus faibles. Cela se calcule aisément avec la fonction *qnorm()*. On indique en tant que premier argument le pourcentage (de 0 à 1) qu'on souhaiterait connaitre, puis comme deuxième et troisième argument la moyenne et l'écart type. Par défaut, R vous retourne la probabilité de percevoir moins. Si vous voulez connaître la probabilité de percevoir plus, il faut spécifier un quatrième argument: "*lower.tail=F*".

Calculons donc la probabilité de percevoir un revenu supérieur à 80% de la population :

```
qnorm(0.8, # 80%
      mean = mean(ESS9_BE_fulltime$log_grspnum), #la moyenne de
      la variable log_grspnum
      sd=sd(ESS9_BE_fulltime$log_grspnum),#l'écart type de la
      variable log_grspnum
      lower.tail = F #Pour connaitre la probabilité que la valeur
      soit au-dessus de 0.8 (80%)
      )
```

```
## [1] 7.678271
```

Le montant à partir duquel on perçoit plus que 80% de la population est de

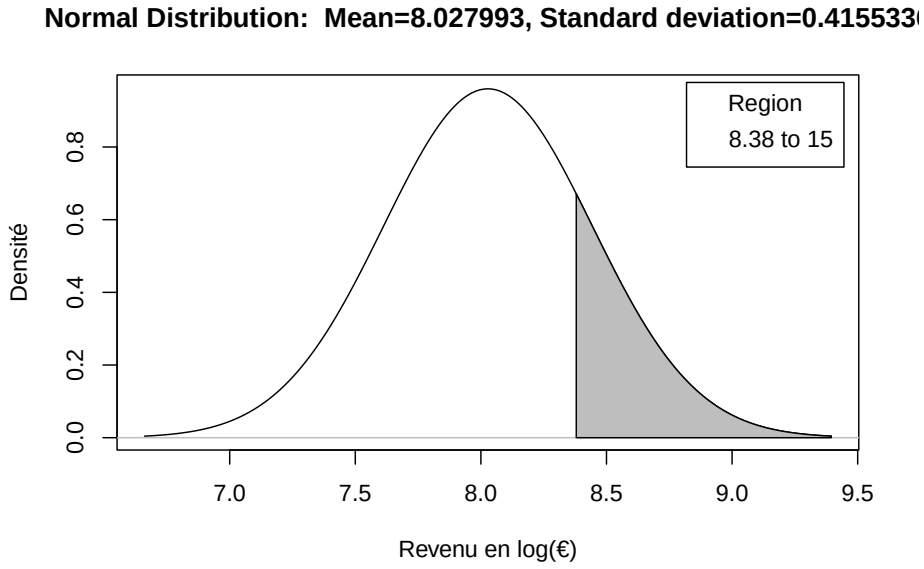
7.68. Ce montant est difficilement interprétable. En effet, il s'agit du logarithme népérien du revenu. Il faut donc transformer cette valeur. On passe du logarithme népérien à la valeur d'origine en appliquant une transformation exponentielle à la valeur. Pour cela, l'on peut utiliser la fonction *exp()* :

```
#On calcule la valeur exacte et l'appelle plus_80_pc
plus_80_pc<-qnorm(0.8, # 80%
      mean = mean(ESS9_BE_fulltime$log_grspnum), #la moyenne de
      la variable log_grspnum
      sd=sd(ESS9_BE_fulltime$log_grspnum),#l'écart type de la
      variable log_grspnum
      lower.tail=T #Pour connaitre la probabilité que la valeur
      soit en dessous de 0.8 (80%)
      )
#On transforme cette valeur en euro:
exp(plus_80_pc)
```

```
## [1] 4349.057
```

Il en découle que l'on doit percevoir plus de 4349, 06€ par mois pour gagner plus de 80% des personnes travaillant à temps plein en Belgique. On peut représenter cela graphiquement :

```
.x <- seq(6.661, 9.395, length.out=1000)
  plotDistr(.x,
            dnorm(.x,
                  mean=mean(ESS9_BE_fulltime$log_grspnum),
                  sd=sd(ESS9_BE_fulltime$log_grspnum)),
            cdf=FALSE,
            xlab="Revenu en log(€)",
            ylab="Densité",
  main=paste("Normal Distribution:  Mean=8.027993, Standard
  deviation=0.4155336"),
  regions=list(c(plus_80_pc, 15)),
  col=c('#BEBEBE', '#BEBEBE'),
  legend.pos='topright')
```



Pour déterminer à partir de quelle salaire on appartient aux 10% des Belges travaillant à temps plein percevant le moins de revenu, le code suivant s'applique :

```
#On calcule la valeur exacte et l'appelle moins_10_pc
moins_10_pc<-qnorm(0.1, # 80%
      mean = mean(ESS9_BE_fulltime$log_grspnum), #la moyenne de
      la variable log_grspnum
      sd=sd(ESS9_BE_fulltime$log_grspnum),#l'écart type de la
      variable log_grspnum
      lower.tail = T #Nous voulons connaitre la probabilité que
      la valeur soit en dessous de 0.1 (10%)
      )
moins_10_pc
```

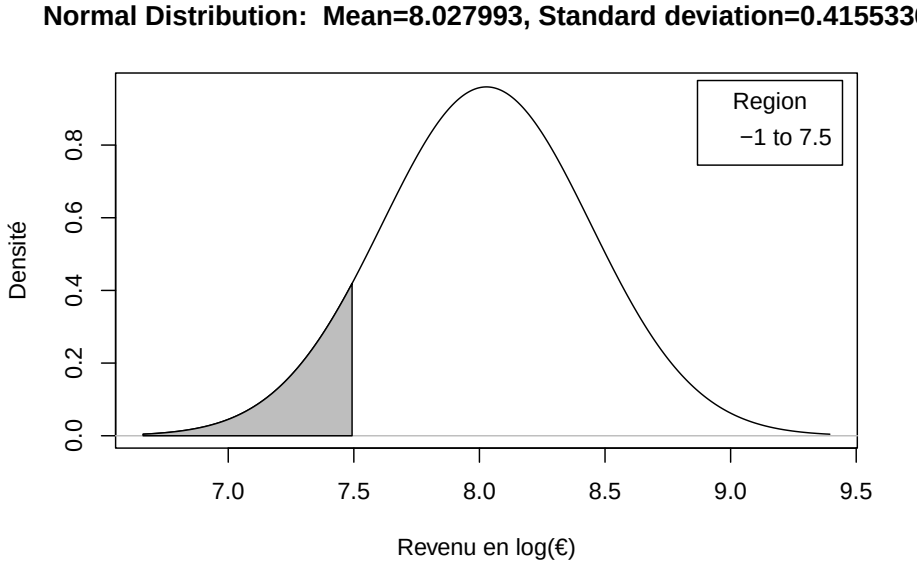
```
## [1] 7.495465
```

```
#On transforme cette valeur en euro:
exp(moins_10_pc)
```

```
## [1] 1799.861
```

Il faut donc percevoir moins de 1799, 86€ pour appartenir au 10% des Belges travaillant à temps plein percevant les revenus les plus faibles. On peut également représenter cela graphiquement :

```
.x <- seq(6.661, 9.395, length.out=1000)
  plotDistr(.x,
            dnorm(.x,
                  mean=mean(ESS9_BE_fulltime$log_grspnum),
                  sd=sd(ESS9_BE_fulltime$log_grspnum)),
            cdf=FALSE,
            xlab="Revenu en log(€)",
            ylab="Densité",
  main=paste("Normal Distribution:  Mean=8.027993, Standard
  deviation=0.4155336"),
  regions=list(c(-1, moins_10_pc)),
  col=c('#BEBEBE', '#BEBEBE'),
  legend.pos='topright')
```



On peut également faire des calculs similaires pour d'autres types de distributions que la distribution normale. Vous trouverez ici une liste non exhaustive des distributions pour lesquelles R propose des fonctions :

- la distribution binomiale
- la distribution du khi-deux
- la distribution exponentielle
- la distribution F (loi de Fisher-Snedecor)
- la distribution logistique
- la distribution log-normale
- La distribution poisson

- La distribution t de *Student*
- La distribution uniforme