

Découvrir R et RStudio

Thierry Zorn, Murielle Lethrosne, Vivien Roussez, Pascal Irz & Nicolas Torterotot

22 juin 2022

Chapitre 1 Introduction



Crédit photographique Pascal Boulin

1.1 Le parcours de formation

Ce dispositif de formation vise à faire monter en compétence les agents du MTECT (Ministère de la Transition écologique et de la Cohésion des territoires) et du MTE (Ministère de la Transition énergétique) dans le domaine de la science de la donnée avec le logiciel R. Il est conçu pour être déployé à l'échelle nationale par le réseau des CVRH (Centre de Valorisation des Ressources Humaines).

Le parcours proposé est structuré en modules de 2 jours chacun. Avoir suivi les deux premiers (ou disposer d'un niveau équivalent) est un pré-requis pour suivre les suivants qui sont proposés "à la carte" :

- Module 1 : Socle - Premier programme en R
- Module 2 : Socle - Préparation des données
- Module 3 : Statistiques descriptives
- Module 4 : Analyse des données multi-dimensionnelles
- Module 5 : Datavisualisation : Produire des graphiques, des cartes et des tableaux
- Module 6 : Publications reproductibles avec RMarkdown (à venir)
- Module 7 : Analyse spatiale
- Module 8 : Big data et optimisation du code (à venir)
- Module 9 : Applications interactives avec RShiny (à venir)

La mise à disposition des supports de formation se fait désormais par la [page d'accueil du parcours de formation](#). Ces supports sont en [licence ouverte](#).

Si vous souhaitez accéder aux sources ou aux données mobilisées pendant les formations, il faut directement les télécharger depuis le [Github du pôle ministériel](#).

Un package d'exercices, [{savoirfR}](#) rassemble toutes les données et les consignes d'exercices de ce parcours de formation (Modules 1, 2 et 5 seulement pour l'instant).

Pour vous tenir au courant de l'offre de formation proposée par le réseau des CVRH, [consultez la plateforme OUPS](#) (un accès intranet MTECT-MTE est nécessaire). Vous pouvez vous y abonner pour recevoir les annonces de formation qui vous intéressent.

Pour échanger de l'information, discuter autour de R ou encore faire part de difficultés et trouver ensemble les solutions, il existe deux canaux d'entraide :

- s'inscrire en envoyant un message vide à l'adresse sympa@developpement-durable.gouv.fr ;
- rejoindre le fil Ariane [#utilisateurs_r](#).

1.2 Objectifs du module 1

Ce module vise à faire découvrir :

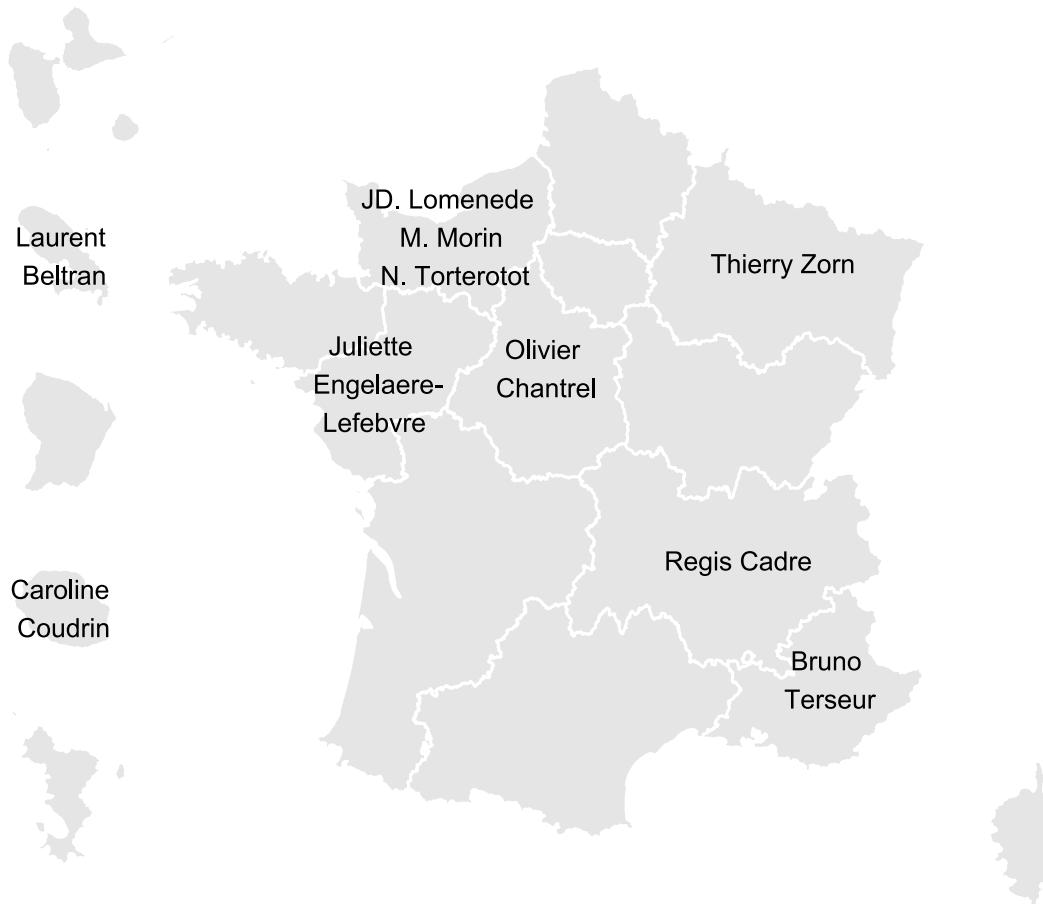
- R et son écosystème.
- L'interface de RStudio.
- Les méthodes élémentaires d'importation des données
- Des premières statistiques
- Des premières représentations graphiques

Il s'agit d'avoir les clefs pour commencer à travailler rapidement sur R en complément, voire en remplacement d'autres outils (Excel, SAS...). Les concepts de programmation sont également abordés pour comprendre les syntaxes proposées dans les différents forums d'aide.

Chapitre 2 R, son écosystème et ses possibilités

2.1 Le groupe de référents R du pôle ministériel

- Un groupe pour structurer une offre de formations sur R
- Un réseau d'entraide



2.2 R c'est quoi ?

- Un langage de programmation interprété exécutable pas à pas \(\rightarrow\) lignes de code
- Un logiciel interactif permettant de traiter divers types de données

2.3 Un peu d'histoire

- À l'origine existait le langage S, servant à programmer avec des données (*Bell laboratory*, années 70)
- Au milieu des années 90, mélange de S et du langage Lisp pour le dédier à l'analyse de données et aux graphiques \(\rightarrow\) création de R. Ihaka R, Gentleman R (1996) R: A Language for Data Analysis and Graphics. *Journal of Computational and Graphical Statistics*, 5(3), 299–314
- S'est d'abord développé dans le monde académique, puis a essaimé vers d'autres domaines professionnels, voire associatifs
- Système toujours d'actualité - Mise à jour tous les 6 mois
- Avenir dans la statistique publique : adopté comme standard par l'Insee (à l'horizon 2025) et divers SSM. Des packages pour :
 - charger les données produites par l'Insee au format sdmx,
 - charger les formats des organisations internationales (Eurostat, OCDE, ONU, etc.),
 - charger des couches spatiales shp ou autres,
 - charger le cadastre au format EDIGEO, ...
 - interroger facilement des API,
 - interagir avec des bases de données,
 - charger des fonds de cartes,
 - réaliser des publications conformes à la charte graphique de l'Etat,
 - gérer les millésimes du référentiel communal, etc.

2.4 Un logiciel libre

- Disponible en libre téléchargement sur [le site officiel du CRAN = Comprehensive R Archive Network](#) C'est le site de référence pour R. Il contient non seulement les installeurs du logiciel mais également de nombreuses ressources : documentations, FAQ, tutoriels...
- Installable sur la plupart des systèmes d'exploitation
- Utilisé en recherche, en enseignement et en entreprise
- Une communauté très active :
 - Forum des [utilisateurs de R en français](#)
 - Communauté française des utilisateurs de R (avec des packages spécifiques, qui permettent de gérer les changements de géographie communale par exemple) : [Frrrenchies](#)
 - Slack de ces utilisateurs (pour les questions/réponses, actualités...) [Slack](#)
 - R-bloggers : <https://www.r-bloggers.com/>
 - Stackoverflow : <https://stackoverflow.com/questions/tagged/r>

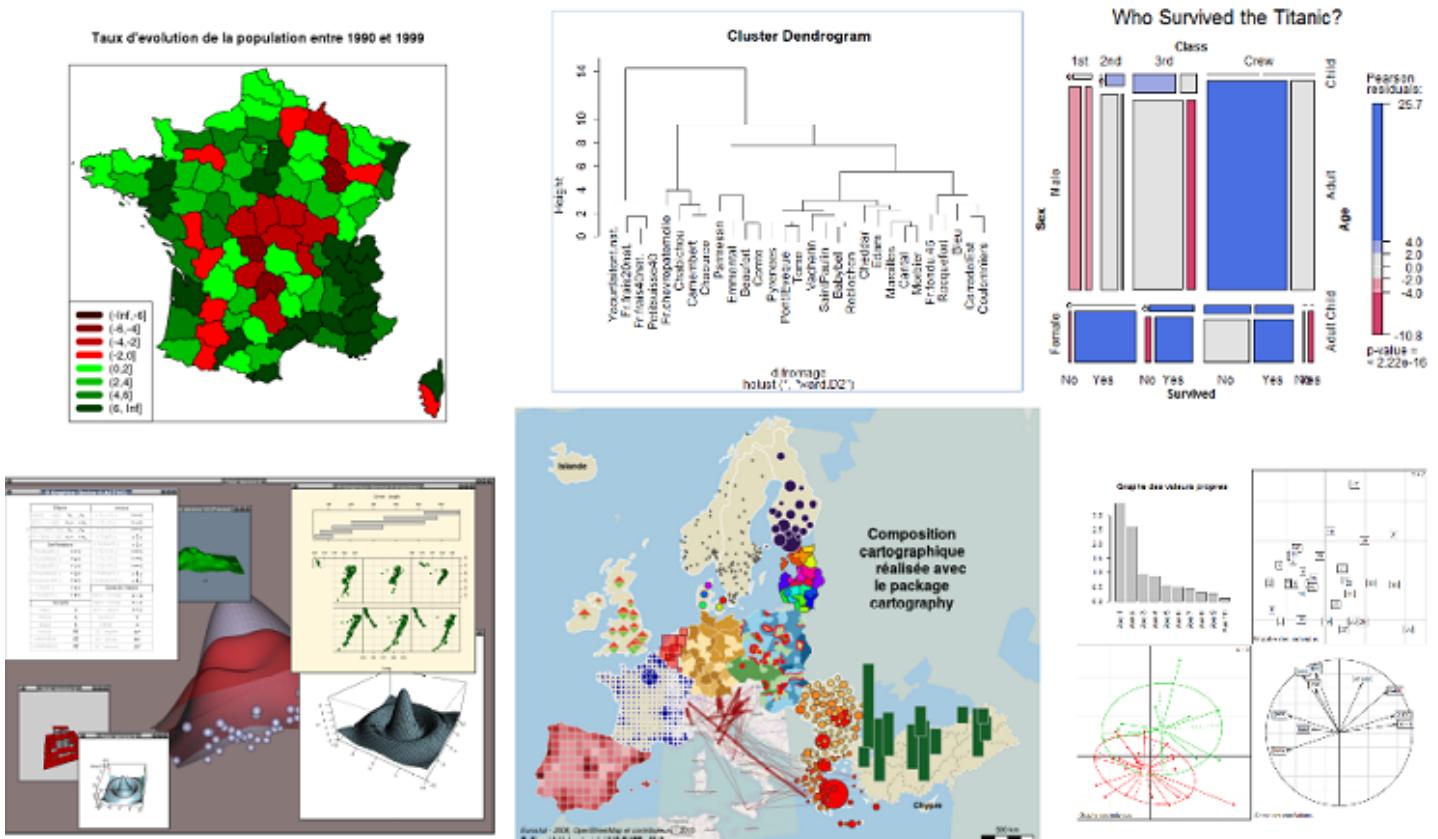
C'est un logiciel libre : les utilisateurs ont la liberté d'exécuter, copier, distribuer, étudier, modifier et améliorer ce logiciel. R fait partie de la "galaxie" GNU (GNU's Not Unix). R est donc libre, gratuit et multi-plateformes.

2.5 Que peut-on faire avec R ?

- Lire des données de formats divers
- Les manipuler (prétraitements)
- Les enrichir à partir de données externes
- Les analyser, les modéliser,
- Présenter les résultats (tableaux, graphiques, cartes)
- Publier...

... et ce au moyen d'une palette de méthodes extrêmement diversifiée.

Privilégie la flexibilité et le découpage des traitements en parties simples.



2.6 Un logiciel modulaire

- Le package `base` contient nombreuses fonctionnalités
- Extensible au moyen de **packages** (1 package = 1 ensemble de nouvelles fonctions)
- Nombreuses extensions spécialisées

De base, R permet déjà de faire un très grand nombre de choses avec son socle commun : le package `base`. Pour afficher la liste des fonctions disponibles dans le package `base` :

```
library(help = 'base')
```

Sur ce package `base` vient se greffer un ensemble de packages (plugins, modules) complémentaires. Un package est une bibliothèque de fonctions. La plupart de ceux qui font référence sont mis à disposition de tous sur le dépôt CRAN.

Cela permet d'étendre à l'infini les possibilités de R : celles-ci sont seulement limitées par la volonté de développement (et de maintien) des utilisateurs.

De même que les packages sont dépendants de fonctions du module principal `base`, certains packages peuvent être dépendants de fonctionnalités d'autres packages \(\rightarrow\) notion d'arbre des dépendances (géré par R de façon transparente).

Structure modulaire \(\rightarrow\) il existe de multiples façons d'effectuer une même tâche. Il suffit d'une méthode maîtrisée pour aller au résultat.

Les fonctionnalités utilisables dans R, sont en perpétuelle évolution et permettent d'intégrer les méthodes les plus récentes dans des domaines qui évoluent rapidement : modélisation, analyse spatiale, dataviz,...

Le groupe des référents R propose un tutoriel d'installation et de configuration de R sur <https://github.com/MTES-MCT/parcours-r/wiki/Installation-de-R-et-RStudio-Desktop-au-MTE>.

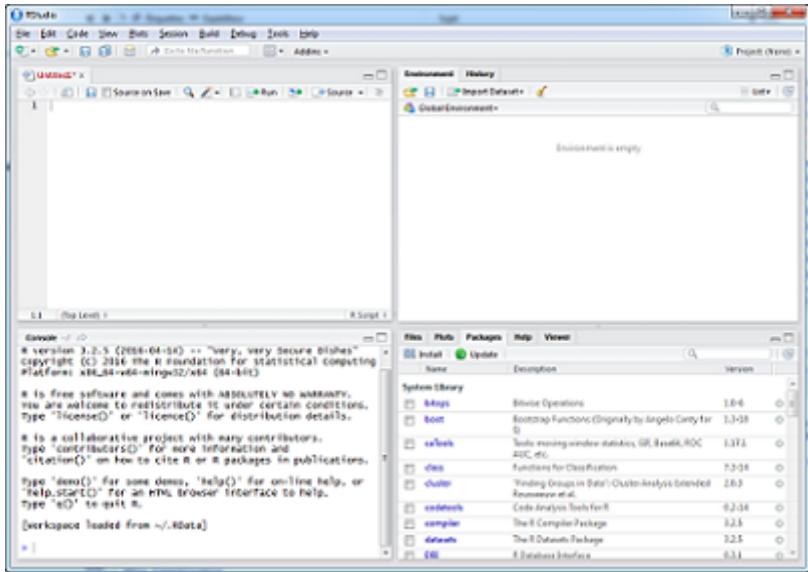
2.7 Des exemples de ce qu'on peut faire avec R :

- Le support de cette formation
- Les publications régionales sur le parc locatif social
- Visualisation du compte du logement
- Les pesticides dans les cours d'eau des Pays de la Loire
- Des exemples de ce qu'on peut faire avec shiny

Chapitre 3 Présentation de l'interface et premières manipulations

Ce chapitre permet de découvrir l'interface de travail RStudio et les principales fonctionnalités pour débuter.

3.1 Une interface dédiée : RStudio



- Environnement de développement conçu spécialement pour R
- Interface utilisateur simple, conviviale, configurable et intégrant plusieurs outils
- Disponible sur <http://www.rstudio.com>

L'interface RStudio est composée de différents panneaux, dont l'arrangement peut être reconfiguré, incluant une console, un navigateur de fichiers et graphiques, l'espace de travail et l'historique des commandes. D'autres environnements graphiques existent, Rstudio semble le plus approprié à nos métiers.

3.2 Session et répertoire de travail

- Session de travail R : commence à l'ouverture de RStudio et se termine en le quittant.
- Répertoire de travail R : dossier dans lequel le logiciel va chercher les fichiers de scripts et de données.
- Tout ce qui a été fait au cours d'une session peut être enregistré dans le répertoire de travail :
 - données
 - historique des fonctions ...

Nouvelles fonctions	Attention
<code>setwd()</code> pour définir un répertoire de travail	Seulement / et pas \
<code>help()</code> et <code>?</code> pour afficher l'aide	.
<code>dir()</code> pour lister un répertoire	.

Attention avec le répertoire de travail : si on l'écrit en dur `setwd('...')`, les anti-slash windows (\) doivent être remplacés par des slash(/)

Concernant le répertoire de travail, quelques conseils :

- Créer un nouveau répertoire pour un projet particulier qui sera votre répertoire de travail
- Créer des sous-répertoires dans ce répertoire : “./Data,” “./Figures,” “./Redaction”...
- Rassembler autant que possible les fichiers qui seront utilisés dans le cadre de ce projet et éviter d'aller chercher des fichiers ailleurs (sauf à inscrire dans le script une instruction de téléchargement exécutable par tous)

3.3 C'est à vous : créer un répertoire de travail

- Sur le poste de travail, créer un dossier ‘premiers_pas’ à partir du menu ‘New Folder’ (cadran droite bas)
- Dans Rstudio, définir ce dossier comme répertoire de travail directement en utilisant
`setwd('')`
- Appeler l'aide en ligne par `?setwd` ou `help(setwd)`
- Faire une recherche dans la partie Help de RStudio

- Créer un script à partir du menu ‘File/New File/R script’ et l'enregistrer sous ‘mes_tous_premiers_pas.R’
- Lire le contenu du répertoire de travail avec `dir()`

3.4 Prise en main de la console

- Effectuer et afficher les résultats de calculs de base (+, -, *, /, ^, ...)
- Utiliser des fonctions spécifiques : `sum` , `abs` , `round` ...
- On peut remonter dans l'historique des fonctions pour en rappeler une

Nouvelles fonctions	Attention
<code>sum()</code> pour sommer un résultat	Séparateur décimal point (.)
<code>abs()</code> pour retourner la valeur absolue	.
<code>round()</code> pour arrondir un nombre	.
flèches pour naviguer dans l'historique	.

3.5 Crédation d'une variable

On assigne un contenu à une variable au moyen de `<-` La flèche d'assignation peut être écrite avec le raccourci clavier alt + “-” (tiret du 6 ou signe ‘moins’ du pavé numérique).

```
ma_variable <- 2
ma_variable <- "Toulouse"
ma_variable <- c("Toulouse", "Nantes", "Strasbourg")
ma_variable <- 1:10
```

\(\rightarrow\) Les variables peuvent être numériques, de type texte ou booléen (TRUE/FALSE) et elles peuvent être réutilisées.

Nouvelles fonctions	Attention pour le nommage des variables
<code>ls()</code> pour faire lister les variables existantes	Casse -> seulement minuscules
<code>paste()</code> pour concaténer des variables textes	Pas de caractères spéciaux ni accentués
<code>rm()</code> pour supprimer une variable	Pas d'espaces ni tirets (-) ; préférer (_)

Adopter des règles de nommage cohérentes, par exemples [celles-ci](#). ATTENTION : un nom de variable ne peut pas commencer par un chiffre et certains noms sont interdits (voir 4.6)

3.6 C'est à vous : créer et manipuler des variables

- Créer plusieurs variables numériques par assignation `a <- 5 , b <- 4`
- Regarder l'onglet Environnement
- Afficher la liste des variables avec la fonction `ls()`
- Faire un calcul avec ces variables et voir le résultat `(a+b)` etc.
- Créer une troisième variable à partir des deux premières `c <- a+b*3`
- Ré-assigner une variable : `a<-10` et vérifier l'onglet environnement
- Créer une variable chaîne de caractère (utilisation des simples quotes et des double-quotes)
`t <- 'chaine'`
- Concaténer `a` et `t` avec `paste(a,t)`
- Expérimenter la casse des noms de variables : créer `A <- 15` et `B <- 12` et vérifier l'onglet environnement
- Supprimer les variables `A` et `B` avec la fonction `rm()`
- Aide en ligne `?ls` et `?rm`
- Attention : Pour supprimer toutes les variables `rm(list = ls())`

3.7 Utilité des scripts

- Garder la trace d'une longue succession de lignes de code
- Pouvoir exécuter ce code (pas à pas ou en entier)
- Le modifier plus rapidement pour l'adapter

- Pouvoir le réutiliser avec de nouvelles données

Nouvelles fonctions	Attention
# pour des commentaires explicatifs du code	Importance de bien commenter
print() pour afficher sur la console	.
Ctrl+Entrée pour lancer un script ligne par ligne	.

Utilisation simple de R → mode console. Chaque ordre, bout de code est rentré et exécuté au fur et à mesure sur la console.

Actions plus complexes, longues, nécessitant une maintenance, des modifications → travailler à partir de la fenêtre éditeur. Les parties de code sont enchaînées et enregistrées sous la forme d'un fichier texte réutilisable par la suite.

Toujours utiliser des commentaires. Permet de pouvoir mieux comprendre ce que l'on a fait lorsqu'on reprend un programme plus tard ou lorsqu'on le donne à quelqu'un.

3.8 C'est à vous : utiliser un script

- Recopier le script ci-dessous et changer les paramètres pour le re-exécuter plusieurs fois
- Sauvegarder ce script dans votre répertoire de travail, fermez le
- Redémarrer votre session RStudio, le rouvrir, et ré-exécuter le script

```
# Supprimer toutes les variables existantes
rm(list = ls())

# Création de mes variables taille et poids ; attention au point décimal
poids <- 91
taille <- 1.87

# Calcul de l'IMC : poids sur taille au carré
imc <- poids / (taille ^ 2)

# Affichage du résultat
print (imc)
```

```
## [1] 26.02305
```

3.9 Installer et charger un package

Pour manipuler notre base de données, nous allons nous servir de fonctionnalités présentes dans le package `tidyverse`.

La première fois que l'on veut utiliser un package, il faut le télécharger sur notre machine, avec

```
install.packages() :
```

```
install.packages("tidyverse")
```

Puis à l'ouverture d'une nouvelle session R, le charger (si on a besoin de ses fonctions), avec

```
library() :
```

```
library(tidyverse)
```

Le package est prêt à être utilisé !

Pour appliquer une fonction à un objet, la syntaxe est :

```
nom_de_la_fonction(objet, attribut1, attribut2, ...)
```

Pour stocker le résultat de cette fonction dans une variable :

```
ma_variable <- nom_de_la_fonction(objet, attribut1, attribut2, ...)
```

Une erreur fréquente est : `Error in nom_de_le_fonction() : impossible de trouver la fonction "nom_de_la_fonction"`

Cela signifie que le package auquel appartient la fonction n'est pas chargé, voire pas installé.

Il peut exister des fonctions homonymes dans différents packages. Sans précision, la fonction qui sera appliquée sera celle du package chargé en dernier. Pour spécifier :

```
nom_du_package::nom_de_la_fonction(objet)
```

Chapitre 4 Bien commencer

4.1 Créer un projet sous Rstudio pour vous permettre de recenser vos travaux.

Pourquoi travailler avec les projets Rstudio plutôt que les scripts R ?

- Cela permet la portabilité : le répertoire de travail par défaut d'un projet est le répertoire où est ce projet. Si vous transmettez celui-ci à un collègue, le fait de lancer un programme ne dépend pas de l'arborescence de votre machine.

Fini les `setwd("chemin/qui/marche/uniquement/sur/mon/poste")` !

- Toujours sur la portabilité, un projet peut être utilisé avec un outil comme `renv` qui va vous intégrer en interne au projet l'ensemble des packages nécessaires au projet. Cela permet donc à votre collègue à qui vous passez votre projet de ne pas avoir à les installer et, surtout, si vous mettez à jour votre environnement R, votre projet restera toujours avec les versions des packages avec lesquelles vous avez fait tourner votre projet à l'époque. Cela évite d'avoir à subir les effets d'une mise à jour importante d'un package qui casserait votre code.

Pour activer `renv` sur un projet, il faut l'installer avec `install.packages("renv")`. Pour initialiser la sauvegarde des packages employés dans le projet, il faut utiliser `renv::init()`. Les packages chargés dans le projet sont enregistrés dans un sous-dossier dédié. En cours de travail sur le projet, la commande `renv::snapshot()` permet de faire une sauvegarde, la commande `renv::restore()` permet de charger la dernière sauvegarde.

[En savoir plus sur `renv`](#)

- Cela permet de se forcer à travailler en mode projet : on intègre à un seul endroit tout ce qui est lié à un projet : données brutes, données retravaillées, scripts, illustrations, documentations, publications... et donc y compris les packages avec `renv`.

- On peut travailler sur plusieurs projets en même temps, Rstudio ouvre autant de sessions que de projets dans ce cas.
- Les projets Rstudio intègrent une interface avec les outils de gestion de version Git et SVN. Cela veut dire que vous pouvez versionner votre projet et l'héberger simplement comme répertoire sur des plateformes de gestion de code telle que Github ou Gitlab.

Pour créer un projet :

- Cliquez sur *Project* en haut à droite puis *New Project*.



- Cliquez sur *New Directory*.



4.2 Utilisation du package `{savoirfR}`

Pour faciliter le déroulé de ce module, l'ensemble des exercices (énoncés, corrigés et données) a été intégré à un package réalisé par le groupe des référents R : `{savoirfR}`

```
install.packages('remotes')
remotes::install_github("MTES-MCT/savoirfR")
```

Pour l'utiliser, il suffit de créer un nouveau projet dans un nouveau répertoire, en sélectionnant le "Project Type" **Exercice Parcours R MTES-MCT**.



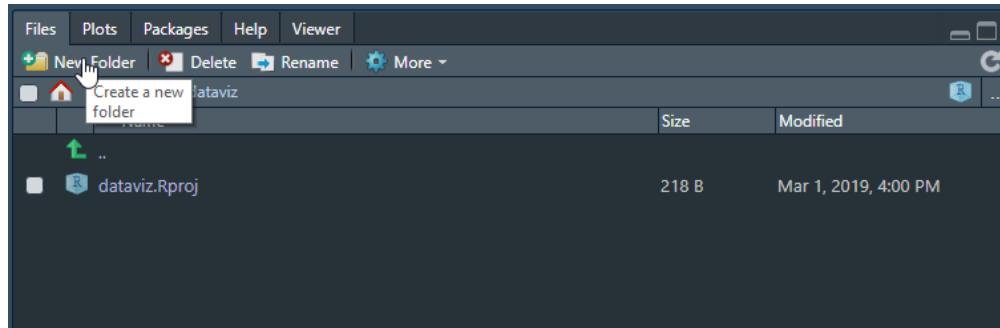
Remplissez et sélectionnez le module suivi.



4.3 Intégrer vos données

Une bonne pratique est de créer un sous répertoire `/data` pour stocker les données sur lesquelles vous aurez à travailler.

Vous pouvez le faire depuis l'explorateur de fichier de votre système d'exploitation ou directement à partir de l'explorateur de fichier de RStudio.



Cela marche bien quand on a un seul type de données, mais en général on va avoir à travailler sur des données brutes que l'on va retravailler ensuite et vouloir stocker à part. Si par la suite vous souhaitez avoir des exemples de bonnes pratiques sur comment structurer vos données, vous pouvez vous référer au [chapitre data](#) du livre d'Hadley Wickham sur la construction de packages R (tout package R étant aussi un projet !).

4.4 Créer votre arborescence de projet

- Créer un répertoire `/src` ou vous mettrez vos scripts R.
- Créer un répertoire `/figures` ou vous mettrez vos illustrations issues de R.

4.5 Activer les packages nécessaires

Commencer par rajouter un script dans le répertoire `/src` à votre projet qui commencera par :

- activer l'ensemble des packages nécessaires
- charger les données dont vous aurez besoin.

```
library(tidyverse)
library(GGally)
library(plotly)

base <- read.csv(file = "extdata/Base_synth_territoires.csv",
                 header = TRUE, sep = ";", dec = ",")
```

4.6 Bien structurer ses projets *data*

Plusieurs documents peuvent vous inspirer sur la structuration de vos projets *data* par la suite.

En voici quelques uns :

- <https://github.com/pavopax/new-project-template>
- <https://nicercode.github.io/blog/2013-04-05-projects/>
- <https://www.inwt-statistics.com/read-blog/a-meaningful-file-structure-for-r-projects.html>
- <http://projecttemplate.net/architecture.html>

À partir du moment où quelques grands principes sont respectés (un répertoire pour les données brutes en lecture seule par exemple), le reste est surtout une question d'attraction plus forte pour l'une ou l'autre solution. L'important est de vous tenir ensuite à conserver toujours la même arborescence dans vos projets afin de vous y retrouver plus simplement.

Chapitre 5 Mon premier jeu de données

5.1 Les types de valeurs dans R

Une valeur constitue l'unité de base des données pour R. Comme pour la plupart des logiciels, elles peuvent être de trois **types** :

- **Numérique** : entier, double
- **Caractère** : texte ou code
- **Logique** : booléens

En anglais : numeric, character et logical !

Les formats de dates sont de type character.

5.2 Les vecteurs

Les valeurs peuvent être structurées au sein de vecteurs. Ces vecteurs peuvent être vus comme des colonnes de valeurs toutes du même type.

Différentes fonctions permettent de créer des vecteurs.

```
vect_num <- c(1, 160, 2, 9)

vect_txt <- c("Je", "programme", "en", "R")

sequence <- seq(from = 1, to = 10, by = 1)
sequence_pareille <- 1:10

repetition <- rep("bla", 3)
```

5.3 Les dataframes

Plusieurs vecteurs de types différents possédant un même nombre de lignes peuvent être accolés pour former une **dataframe**. Les dataframes sont les objets les plus courants dans le traitement de données usuel. Il s'agit de tableaux dont les lignes correspondent à des observations et les colonnes à des variables.

Ces dataframes peuvent être créées par association des vecteurs avec les fonctions :

```
dataframe_a <- data.frame(vect_num, vect_txt)

dataframe_b <- bind_cols("vect_num" = vect_num, "vect_txt" = vect_txt)
```

Une dataframe peut aussi être créée par l'import d'un tableau, voir le chapitre dédié.

Il est possible d'accéder aux éléments d'une dataframe à partir du numéro de ligne et de colonne, grâce aux crochets :

	V1	V2	V3	V4	V5	V6	...	Vp
1								
2								
3								
4								
5								
6								
7								
8								
9								
10								
11								
12								
:								
n								

- `base[1,3]` \(\rightarrow\) valeur de la première ligne et de la troisième colonne
- `base[2,]` \(\rightarrow\) toutes les variables pour la 2e observation
- `base[,4]` \(\rightarrow\) toutes les observations de la quatrième colonne
- `base[, 'V6']` \(\rightarrow\) toutes les observations de la variable V6
- \(\rightarrow\) Utile pour sélectionner une partie d'une table : `base[1:4, c(3, 6)]`

5.4 Les fonctions et les valeurs particulières

- NA : valeur manquante (*Not Available*) dans une dataframe ou en résultat d'une fonction
- NaN : pas un nombre (*Not a Number*) lorsqu'une fonction tente de diviser par 0
- -Inf, Inf : infini positif ou négatif lorsque une fonction diverge

Une valeur manquante peut perturber l'exécution d'une fonction :

```
V1 <- c(1, 14, NA, 32.7)
mean(V1)           # renvoie NA. Not good !
```

```
## [1] NA
```

```
mean(V1, na.rm = TRUE)    # renvoie 15.9 - OK !
```

```
## [1] 15.9
```

Les noms de ces valeurs particulières sont “réservés” par R, ils ne peuvent pas être utilisés comme nom de variable. C'est également le cas des booléens (`TRUE` , `FALSE` , `T` et `F`)

\(\Rightarrow\) Le module 2 “Préparation des données” aborde la gestion de ces valeurs particulières.

5.5 Importation de données

Conseil : exporter les données au format CSV ; c'est le format le plus interopérable (supporté par tous les logiciels stat). Utiliser ensuite la fonction `read.csv` après avoir défini le répertoire de travail (ou en donnant le chemin complet)

```
base0 <- read.csv(file = "extdata/Base_synth_territoires.csv",
                  header = TRUE,
                  sep = ";",
                  dec = ",")
```

- `header =` indique la présence des noms de variable sur la première ligne

- `sep` = indique le séparateur de champ : ';' '/t' pour tabulation
- `dec` = indique le séparateur de décimale ('.' par défaut)
- `colClasses` = permet de préciser le type de chaque colonne de la donnée en entrée, par exemple pour une table de 3 colonnes de types texte, puis 2 fois numériques : `colClasses = c("character", "numeric", "numeric")`. `NA` pour laisser R deviner, `"NULL"` pour éviter d'importer la colonne.

\(\rightarrow\) ?read.csv pour plus d'options

Autres façons d'importer les données

- Fonction `read_delim`, du package `readr`, plus rapide
- Fonction `fread`, du package `data.table`, beaucoup plus rapide !!
- Pour importer les fichiers XLS, ODT ou DBF, il existe des fonctions et des packages spécifiques
- **Le passage par un fichier csv est très recommandé**

5.6 Gérer le type des variables

Chaque variable est du type de son contenu (numeric, character, logical). Si les valeurs prises par la variables correspondent à un nombre fini de modalités, **la variable peut être du type "factor"**. Ses composantes sont toujours d'un des 3 types décrits ci-dessus mais il est possible d'employer des fonctions spécifiques au traitement de modalités. À chaque type de variable correspond une utilisation. Lors de l'import des données, un type est affecté automatiquement par R. Mais le type peut être erroné. Il suffit alors de les convertir :

```
base <- mutate(base0, LIBGEO = as.character(LIBGEO))
```

ou `as.factor()` , `as.numeric()` , etc...

Pour être sûr de ne pas faire de bêtise, il vaut mieux gérer les types au moment de l'importation avec le paramètre `colClasses` \(\rightarrow\) exercice !

5.7 Structure des données : le *dataframe*

	V1	V2	V3	V4	V5	V6	...	Vp
1								
2								
3								
4								
5								
6								
7								
8								
9								
10								
11								
12								
:								
n								

- n lignes (observations)
- p colonnes (variables)

Les fonctions suivantes permettent de connaître la structure de la dataframe, d'en connaître les variables, d'en observer certaines lignes ou d'en vérifier le type.

```
nrow(base) # nombre de Lignes
```

```
## [1] 36689
```

```
ncol(base) # nombre de colonnes
```

```
## [1] 38
```

```
dim(base) # nombre de Lignes et de colonnes
```

```
## [1] 36689      38
```

```
names(base) # noms des variables
```

```
## [1] "CODGEO"          "LIBGEO"           "REG"              "DEP"              "ZAU"              "ZE  
## [10] "NAIS0914"        "DECE0914"         "P14_MEN"          "NAISD15"          "DECESD15"         "P1  
## [19] "P14_RP_PROP"     "NBMENFISC13"     "PIMP13"           "MED13"            "TP6013"           "P1  
## [28] "P14_CHOM1564"    "P14_ACT1564"     "ETTOT14"          "ETAZ14"           "ETBE14"           "ET  
## [37] "ETTEF114"         "ETTEFP1014"
```

```
str(base) # nom, type et extrait des variables
```

```
## 'data.frame': 36689 obs. of 38 variables:
## $ CODGEO : chr "01001" "01002" "01004" "01005" ...
## $ LIBGEO : chr "L'Abergement-Clémenciat" "L'Abergement-de-Varey" "Ambérieu-en-Bugey" ...
## $ REG : int 84 84 84 84 84 84 84 84 84 ...
## $ DEP : chr "01" "01" "01" "01" ...
## $ ZAU : chr "120 - Multipolarisée des grandes aires urbaines" "112 - Couronne de Lyon" ...
## $ ZE : chr "8213 - Villefranche-sur-Saône" "8201 - Ambérieu-en-Bugey" "8201 ...
## $ P14_POP : int 767 239 14022 1627 109 2570 743 338 1142 397 ...
## $ P09_POP : int 787 207 13350 1592 120 2328 660 336 960 352 ...
## $ SUPERF : num 15.95 9.15 24.6 15.92 5.88 ...
## $ NAIS0914 : int 40 16 1051 117 8 175 59 12 56 25 ...
## $ DECE0914 : int 25 7 551 41 3 78 20 11 32 10 ...
## $ P14_MEN : num 306 99.3 6161.1 621.1 52.5 ...
## $ NAISD15 : int 13 5 222 15 2 21 11 2 18 4 ...
## $ DECESD15 : int 5 1 121 7 2 9 3 3 5 0 ...
## $ P14_LOG : num 342.7 161.2 6838.4 661.8 71.5 ...
## $ P14_RP : num 306 99.3 6161.1 621.1 52.5 ...
## $ P14_RSECOCC : num 14 47.3 121.6 10.9 10.9 ...
## $ P14_LOGVAC : num 22.74 14.55 555.64 29.85 8.14 ...
## $ P14_RP_PROP : num 260 84.6 2769 473.3 37.7 ...
## $ NBMENFISC13 : int 297 99 6034 617 47 1014 299 140 431 137 ...
## $ PIMP13 : num NA NA 57.4 NA NA ...
## $ MED13 : num 22130 23213 19554 22388 21872 ...
## $ TP6013 : num NA NA 15.1 NA NA ...
## $ P14_EMPLT : num 85.16 12.81 7452.93 280.57 5.95 ...
## $ P14_EMPLT_SAL: num 52.19 4.95 6743.37 206.38 3.96 ...
## $ P09_EMPLT : num 65.57 17.64 7551.68 286.61 5.29 ...
## $ P14_POP1564 : num 463 141.6 8962.8 1043.1 71.3 ...
## $ P14_CHOM1564 : num 33 9.84 1059.73 66.33 7.93 ...
## $ P14_ACT1564 : num 376 121 6681.9 842.1 57.5 ...
## $ ETTOT14 : int 47 22 1316 141 7 203 66 39 54 36 ...
## $ ETAZ14 : int 9 1 7 14 0 21 2 5 5 6 ...
## $ ETBE14 : int 2 3 60 7 0 18 0 2 6 4 ...
## $ ETFZ14 : int 5 1 131 19 0 21 9 1 13 3 ...
## $ ETGU14 : int 25 14 892 85 5 114 45 27 24 18 ...
```

```

## $ ETGZ14      : int 6 4 283 19 1 28 16 6 9 7 ...
## $ ETOQ14      : int 6 3 226 16 2 29 10 4 6 5 ...
## $ ETTEF114     : int 9 2 385 27 0 38 17 6 9 7 ...
## $ ETTEFP1014   : int 0 0 109 5 0 15 2 0 3 0 ...

```

```
head(base) # 6 premières Lignes
```

	CODGEO	LIBGEO	REG	DEP	ZAL				
## 1	01001	L'Abergement-Clémenciat	84	01 120	- Multipolarisée des grandes aires urbaines				
## 2	01002	L'Abergement-de-Varey	84	01	112 - Couronne d'un grand pôle				
## 3	01004	Ambérieu-en-Bugey	84	01	112 - Couronne d'un grand pôle				
## 4	01005	Ambérieux-en-Dombes	84	01	112 - Couronne d'un grand pôle				
## 5	01006	Ambléon	84	01	300 - Autre commune multipolarisée				
## 6	01007	Ambronay	84	01	112 - Couronne d'un grand pôle				
##	P14_MEN	NAISD15	DECESD15	P14_LOG	P14_RP	P14_RSECOCC	P14_LOGVAC	P14_RP_PROP	N
## 1	306.00000	13	5	342.73473	306.00000	13.99418	22.740550	260.00000	
## 2	99.33745	5	1	161.16023	99.33745	47.27625	14.546538	84.58436	
## 3	6161.06200	222	121	6838.35437	6161.06200	121.64795	555.644416	2769.00170	
## 4	621.05374	15	7	661.76017	621.05374	10.85505	29.851387	473.32736	
## 5	52.51818	2	2	71.51818	52.51818	10.85714	8.142857	37.65455	
## 6	1028.00000	21	9	1160.00000	1028.00000	56.00000	76.000000	779.00000	
##	P09_EMPLT	P14_POP1564	P14_CHOM1564	P14_ACT1564	ETTOT14	ETAZ14	ETBE14	ETFZ14	ETGU14
## 1	65.566193	463.00000	33.000000	376.00000	47	9	2	5	25
## 2	17.644456	141.62963	9.835391	120.97531	22	1	3	1	14
## 3	7551.682296	8962.84216	1059.728437	6681.86216	1316	7	60	131	892
## 4	286.611037	1043.12909	66.326127	842.14083	141	14	7	19	85
## 5	5.285714	71.34545	7.927273	57.47273	7	0	0	0	5
## 6	491.138876	1614.00000	108.000000	1267.00000	203	21	18	21	114

```
tail(base) # 6 dernières Lignes
```

```

##      CODGEO          LIBGEO REG DEP          ZAU
## 36684 97419      Sainte-Rose   4 974 400 - Commune isolée hors influence des pôles   €
## 36685 97420      Sainte-Suzanne   4 974           112 - Couronne d'un grand pôle 04€
## 36686 97421          Salazie   4 974 400 - Commune isolée hors influence des pôles   €
## 36687 97422       Le Tampon   4 974        111 - Grand pôle (plus de 10 000 emplois) 04
## 36688 97423 Les Trois-Bassins   4 974           112 - Couronne d'un grand pôle 04€
## 36689 97424          Cilaos   4 974 400 - Commune isolée hors influence des pôles 04
##      DECESD15    P14_LOG    P14_RP P14_RSECOCC P14_LOGVAC P14_RP_PROP NBMENFISC13 PIMP13
## 36684      43 2542.478 2322.000    28.58054   191.8979    1799.954        NA     NA
## 36685     100 8328.616 7686.912    50.94097   590.7629    4450.648        NA     NA
## 36686      46 2987.000 2420.000   174.00000   393.0000    1849.000        NA     NA
## 36687     420 32710.091 29662.460  1004.31703  2043.3140   15372.921        NA     NA
## 36688      54 2890.404 2484.887    67.58626   337.9313    1773.765        NA     NA
## 36689      38 2732.690 2016.000   239.56501   477.1253   1522.480        NA     NA
##      P14_ACT1564 ETTOT14 ETAZ14 ETBEE14 ETFZ14 ETGU14 ETGZ14 ETQ14 ETTEF114 ETTEFP1014
## 36684 2850.155     414    125     44     31    159     59     55     82      8
## 36685 10456.196    1323    136    139    190    694    249    164    267     66
## 36686 3108.000     490    148     29     43    214     73     56     95     11
## 36687 34446.789    5476    565    398    665   3025    999    823   1032    204
## 36688 3524.287     456    42      39     61    223     76     91     59     20
## 36689 2415.666     360    49      17     35    193     49     66     65     26

```

```
class(base) # La classe de l'objet (du point de vue Langage orienté objet)
```

```
## [1] "data.frame"
```

```
typeof(base) # Le type d'objet du point de vue "interne" à R
```

```
## [1] "list"
```

5.8 Exercice 2 : importer des données et premier coup d'oeil

On peut importer n'importe quel format de données en R (Excel, SAT, Stata, SQL...). Beaucoup sont abordés lors du [module 2 “Préparation des données”](#). Pour ce module, nous ne voyons que l'importation de fichier `.csv`. Si vous avez une base de données en Excel ou LibreOffice Calc, sauvegardez l'onglet que vous souhaitez en faisant “enregistrer sous” \rightarrow “délimité CSV.”

Ici, nous travaillerons sur une base de données communales fournie par l'Insee, dite “comparateur de territoires.”

- Utiliser la fonction `read.csv()` pour importer ce fichier et stocker le dans un objet `df`.
Veillez à ce que la région soit bien importée comme un facteur et non un entier
- Inspecter le dataframe avec les fonctions vues auparavant pour connaître le nombre de lignes, de colonnes, ...

5.9 Catalogue d'attributs de la base

- *CODGEO* : [text] Code du département suivi du numéro de commune ou du numéro d'arrondissement municipal
- *LIBGEO* : [text] Libellé de la commune ou de l'arrondissement municipal pour Paris
- *REG*: [text] Région
- *DEP* : [text] Département
- *ZAU* : [text] Zonage en aire urbaine
- *ZE* : [text] Zone d'emploi
- *P14_POP*: [double] Population en 2014
- *P09_POP*: [double] Population en 2009
- *SUPERF*: [double] Superficie (en km²)
- *NAIS0914*: [double] Nombre de naissances entre le 01/01/2009 et le 01/01/2014
- *DECE0914*: [double] Nombre de décès entre le 01/01/2009 et le 01/01/2014
- *P14_MEN*: [double] Nombre de ménages en 2014
- *NAISD15*: [double] Nombre de naissances domiciliées en 2015
- *DECESD15*: [double] Nombre de décès domiciliés en 2015

- $P14_LOG$: [double] Nombre de logements en 2014
- $P14_RP$: [double] Nombre de résidences principales en 2014
- $P14_RSECOCC$: [double] Nombre de résidences secondaires et logements occasionnels en 2014
- $P14_LOGVAC$: [double] Nombre de logements vacants en 2014
- $P14_RP_PROP$: [double] Nombre de résidences principales occupées par propriétaires en 2014
- $NBMENFISC13$: [double] Nombre de ménages fiscaux en 2013
- $PIMP13$: [text] Part des ménages fiscaux imposés en 2013
- $MED13$: [double] Médiane du niveau de vie en 2013
- $TP6013$: [text] Taux de pauvreté en 2013
- $P14_EMPLT$: [double] Nombre d'emplois au lieu de travail en 2014
- $P14_EMPLT_SAL$: [double] Nombre d'emplois salariés au lieu de travail en 2014
- $P09_EMPLT$: [double] Nombre d'emplois au lieu de travail en 2009
- $P14_POP1564$: [double] Nombre de personnes de 15 à 64 ans en 2014
- $P14_CHOM1564$: [double] Nombre de chômeurs de 15 à 64 ans en 2014
- $P14_ACT1564$: [double] Nombre de personnes actives de 15 à 64 ans en 2014
- $ETTOT14$: [double] Total des établissements actifs au 31 décembre 2014
- $ETAZ14$: [double] Etablissements actifs de l'agriculture, sylviculture et pêche au 31/12/2014
- $ETBE14$: [double] Etablissements actifs de l'industrie au 31/12/2014
- $ETFZ14$: [double] Etablissements actifs de la construction au 31/12/2014
- $ETGU14$: [double] Etablissements actifs du commerce, transports et services divers au 31/12/2014
- $ETGZ14$: [double] Etablissements actifs du commerce et réparation automobile au 31/12/2014
- $ETOQ14$: [double] Etablissements actifs de l'administration publique, enseignement, santé et action sociale au 31/12/2014
- $ETTEF114$: [double] Etablissements actifs de 1 à 9 salariés au 31 décembre 2014
- $ETTEFP1014$: [double] Etablissements actifs de 10 salariés ou plus au 31 décembre 2014

Chapitre 6 Première manipulation des données

```
library("dplyr")
```

6.1 Afficher les valeurs et manipuler les variables

- Pour afficher la table, plusieurs façons : “clic” dans l'environnement Rstudio, `View(base)` ,
`print(base)` , `base` .
 - Pour accéder à une variable : fonction `pull()`

Par exemple :

```
str(pull(base, DEP))
```

6.2 Crer de nouvelles variables



La fonction `mutate()` permet de créer/modifier une variable (ou plusieurs).

```
TableEnSortie <- mutate(TableEnEntree,  
                        NouvelleVariable = DefinitionDeLaVariable)
```

Par exemple :

```
base <- mutate(base, log_SUPERF = log(SUPERF))
```

Nb : `mutate()` permet également de modifier une variable. Dans ce cas la syntaxe est la même que ci-dessus, mais les noms d'entrée et de sortie sont les mêmes :

```
base <- mutate(base, log_SUPERF = 100 * log_SUPERF)
```

\(\Rightarrow\) La table base contient de nouvelles colonnes

6.3 Sélectionner des variables



La fonction `select()` permet de sélectionner les variables voulues.

- sélection par liste blanche

```
TableEnSortie <- select(TableEnEntree, Variable1, Variable2, ..., VariableN)
```

- sélection par liste noire (supprimer)

```
TableEnSortie <- select(TableEnEntree, -Variable1, -Variable2, ..., -VariableN)
```

Par exemple :

```
base_select <- select(base, CODGEO, LIBGEO, P14_POP)
base_select <- select(base, -CODGEO)
```

6.4 Filtrer des observations



La fonction `filter()` permet de sélectionner les observations, selon une condition (ou plusieurs).

```
TableEnSortie <- filter(TableEnEntree, Condition1, ..., ConditionN)
```

Par exemple :

```
base_filter <- filter(base, DEP == "01" & P14_POP > 10000)
```

\(\Rightarrow\) Attention à l'opérateur de comparaison : “`==`” et non pas “`=`”

6.5 Les tests logiques dans R

Syntaxe	Action
<code>==</code>	Test d'égalité
<code>!=</code>	Different de
<code>%in% c(...)</code>	Dans une liste de valeurs
<code>>, >= , <, <=</code>	Supérieur (ou inférieur) (ou égal)
<code>! (x %in% c(...))</code>	N'est pas dans une liste de valeurs

```
TableEnSortie <- filter(TableEnEntree, x==a & y==b)      # x vaut a **ET** y vaut b  
TableEnSortie <- filter(TableEnEntree, x==a | y==b)      # x vaut a **OU** y vaut b (barre v)
```

6.6 Renommer des colonnes

La fonction `rename()` permet de renommer une variable (ou plusieurs).

```
base <- rename(base, nouveau_nom = ancien_nom)
```

Exemple

```
base_rename <- rename(base, ZONE_EMPLOI = ZE)
```

6.7 Exercice 3 : créer, filtrer, sélectionner

- En utilisant la fonction `mutate()`, créer une nouvelle variable correspondant à la densité de population (rapport de la population à la superficie de la commune), ainsi que les taux de natalité et de mortalité (en pour mille de la population 2014)
- A l'aide de la fonction `select()`, créer une nouvelle table en ne conservant que le code commune, le type de commune (ZAU), la région, le département et les variables que vous venez de créer.
- Enfin, ne conserver les communes correspondant à votre département de naissance et stocker ce *dataframe*. Attention au type de la variable département !
- Avec les opérateurs logiques, faire des essais pour sélectionner des échantillons différents (autres départements, densité, l'un et l'autre...).

6.8 Nom d'un pipe %>% !

Pour enchaîner des opérations, on peut créer des variables successives :

```
df <- mutate(base, densite = P14_POP / SUPERF,
             tx_natal = 1000 * NAISD15 / P14_POP,
             tx_mort = 1000 * DECESD15 / P14_POP)

selection <- select(df, CODGEO, ZAU, REG, DEP, densite, tx_natal)
filtre_62 <- filter(selection, DEP == "62")
```

Cela peut être relativement clair si on respecte de bonnes pratiques de nommage mais encombre inutilement l'environnement de travail.

On peut emboîter les fonctions :

```
selection_62 <- filter(select(mutate(base, densite = P14_POP / SUPERF,
                                         tx_natal = 1000 * NAISD15 / P14_POP,
                                         tx_mort = 1000 * DECESD15 / P14_POP),
                                         CODGEO, ZAU, REG, DEP, densite, tx_natal),
                                         DEP == "62")
```

Ce choix préserve l'environnement de travail mais la lecture est particulièrement confuse et le risque d'oubli de parenthèses important.

Il y a enfin une solution du package `magrittr` faisant partie du `tidyverse`. On peut combiner les opérations en une seule ligne à l'aide de l'opérateur pipe `%>%` :

```
selection_62 <- base %>%
  mutate(densite = P14_POP / SUPERF,
        tx_natal = 1000 * NAISD15 / P14_POP,
        tx_mort = 1000 * DECESD15 / P14_POP) %>%
  select(CODGEO, ZAU, REG, DEP, densite, tx_natal) %>%
  filter(DEP == "62")
```

Cette écriture permet d'enchaîner les opérations telles qu'on les décrirait à l'oral. L'objet auquel s'applique chaque nouvelle opération est le résultat de l'opération précédente.

Chapitre 7 Premiers traitements statistiques

7.1 Obtenir des informations

La fonction `summary()` peut s'appliquer à une table entière ou un vecteur. Appliquée sur un tableau, elle donne les statistiques principales sur chacune des variables, en s'adaptant au type de celles-ci (numérique ou texte).

```
base_extrait <- base %>%
  select(1, 3, 5, 7:12)
summary(base_extrait)
```

```
##      CODGEO             REG            ZAU          P14_POP          P09_POP
##  Length:36689    Min.   : 1.00  Length:36689    Min.   :     0  Min.   :     0
##  Class  :character  1st Qu.:28.00  Class  :character  1st Qu.:    197  1st Qu.:    193
##  Mode   :character  Median :44.00   Mode   :character  Median :    444  Median :    431
##                  Mean   :52.05                Mean   :   1838  Mean   :   1793
##                  3rd Qu.:76.00                3rd Qu.:   1110  3rd Qu.:   1072
##                  Max.   :94.00                Max.   :2220445  Max.   :2234105
##                               NA's   :821       NA's   :821
##      P14_MEN
##  Min.   :     0.0
##  1st Qu.:   83.8
##  Median : 183.2
##  Mean   : 802.0
##  3rd Qu.: 454.9
##  Max.   :1147990.9
##  NA's   :821
```

- Les variables quantitatives

```
summary(pull(base_extrait, NAIS0914))
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.   NA's
##      0.0     9.0    23.0    114.4    60.0 150843.0      821
```

- Les variables qualitatives

```
summary(pull(base_extrait, ZAU))
```

```
##      Length     Class     Mode
##      36689 character character
```

7.2 Calculer des statistiques spécifiques

Les fonctions `sum()` , `mean()` , `median()` , `min()` , `max()` , `var()` , `sd()` ... résument l'information pour en donner une statistique. La fonction `quantile()` renvoie les quartiles de la variables (ou bien tout autre découpage qu'on lui renseigne).

```
sum(pull(base_extrait, P14_POP), na.rm = TRUE)
```

```
## [1] 65907160
```

```
mean(pull(base_extrait, P14_POP), na.rm = TRUE)
```

```
## [1] 1837.492
```

```
median(pull(base_extrait, P14_POP), na.rm = TRUE)
```

```
## [1] 444
```

```
quantile(pull(base_extrait, P14_POP), probs = c(0.25, 0.5, 0.75), na.rm = T)
```

```
## 25% 50% 75%
## 197 444 1110
```

Ces fonctions retournent une valeur, ou bien un ensemble de valeur (pour `quantile()`). Le résultat est donc un vecteur de un ou plusieurs nombres.

7.3 Agréger des données

7.3.1 Globalement



La fonction `summarise()` permet d'aggréger des données, en appliquant une fonction sur les variables pour construire une statistique sur les observations de la table. C'est une fonction dite de “résumé.”

```
summarise(TableEnEntree, NomVariableAgreegee = Fonction(NomVariableEtude))
```

```
base_med <- base_extrait %>%
  summarise(population_med = median(P14_POP, na.rm = T))
```

La fonction `summarise()` retourne un `data.frame`.

7.3.2 Selon un facteur



La fonction `summarise()` couplée à `group_by()` permet de calculer des statistiques pour chaque modalité d'une variable qualitative. Avec `group_by()`, on précise les variables qui formeront des groupes, sur lesquels on appliquera une fonction :

```
TableauGroupes <- group_by(TableEnEntree, Variable1, ..., VariableN)
summarise(TableauGroupes, NomVariableAgreee = Fonction (NomVariableEtude))
```

Par exemple, si on veut avoir la médiane de la variable P14_POP, pour chaque ZAU et chaque région :

```
base_reg_ann <- base_extrait %>%
  group_by(ZAU, REG) %>%
  summarise(population_med = median (P14_POP, na.rm = TRUE))
```

7.4 Tableau de contingence

La fonction `table()` calcule les effectifs d'un tableau croisé :

```
t <- base_extrait %>%
  select(ZAU, REG) %>%
  table()

print(t)
```

	REG	1	2	3	4	11	24	27	2
## ZAU									
## 111 - Grand pôle (plus de 10 000 emplois)		17	16	3	10	413	103	140	21
## 112 - Couronne d'un grand pôle		6	0	3	3	853	734	1299	112
## 120 - Multipolarisée des grandes aires urbaines		1	4	0	4	3	188	336	48
## 211 - Moyen pôle (5 000 à 10 000 emplois)		0	3	2	2	3	30	31	3
## 212 - Couronne d'un moyen pôle		0	0	0	0	2	72	122	10
## 221 - Petit pôle (de 1 500 à 5 000 emplois)		3	2	1	0	0	46	60	7
## 222 - Couronne d'un petit pôle		0	0	0	0	0	19	137	2
## 300 - Autre commune multipolarisée		2	4	0	1	7	375	737	76
## 400 - Commune isolée hors influence des pôles		3	5	13	4	0	275	969	39

7.5 Tableau de proportions

La fonction `prop.table()` prend en entrée un objet `table` (tableau de contingence avec les effectifs) et calcule les pourcentages (total, ligne, colonne) associés \(\rightarrow\) `?prop.table`

```
# Calcule la fréquence en % (la somme de tous les pourcentages vaut 100)
(prop.table(t) * 100) %>% round(digits = 1)
```

	REG	1	2	3	4	11	24	27	28	32	4
## ZAU											
## 111 - Grand pôle (plus de 10 000 emplois)		0.0	0.0	0.0	0.0	1.1	0.3	0.4	0.6	1.3	0.
## 112 - Couronne d'un grand pôle		0.0	0.0	0.0	0.0	2.3	2.0	3.5	3.1	4.1	4.
## 120 - Multipolarisée des grandes aires urbaines		0.0	0.0	0.0	0.0	0.0	0.5	0.9	1.3	2.0	2.
## 211 - Moyen pôle (5 000 à 10 000 emplois)		0.0	0.0	0.0	0.0	0.0	0.1	0.1	0.1	0.1	0.
## 212 - Couronne d'un moyen pôle		0.0	0.0	0.0	0.0	0.0	0.2	0.3	0.3	0.0	0.
## 221 - Petit pôle (de 1 500 à 5 000 emplois)		0.0	0.0	0.0	0.0	0.0	0.1	0.2	0.2	0.1	0.
## 222 - Couronne d'un petit pôle		0.0	0.0	0.0	0.0	0.0	0.1	0.4	0.1	0.0	0.
## 300 - Autre commune multipolarisée		0.0	0.0	0.0	0.0	0.0	1.0	2.0	2.1	1.9	3.
## 400 - Commune isolée hors influence des pôles		0.0	0.0	0.0	0.0	0.0	0.7	2.6	1.1	0.8	2.

```
# Calcule la fréquence en % par région (la somme de tous les pourcentages d'une colonne vau
(prop.table(t,"REG") * 100) %>%
  round(digits = 1)
```

	REG								
##	1	2	3	4	11	24	27	2	
## ZAU									
## 111 - Grand pôle (plus de 10 000 emplois)	53.1	47.1	13.6	41.7	32.2	5.6	3.7	6.	
## 112 - Couronne d'un grand pôle	18.8	0.0	13.6	12.5	66.6	39.8	33.9	34.	
## 120 - Multipolarisée des grandes aires urbaines	3.1	11.8	0.0	16.7	0.2	10.2	8.8	15.	
## 211 - Moyen pôle (5 000 à 10 000 emplois)	0.0	8.8	9.1	8.3	0.2	1.6	0.8	1.	
## 212 - Couronne d'un moyen pôle	0.0	0.0	0.0	0.0	0.2	3.9	3.2	3.	
## 221 - Petit pôle (de 1 500 à 5 000 emplois)	9.4	5.9	4.5	0.0	0.0	2.5	1.6	2.	
## 222 - Couronne d'un petit pôle	0.0	0.0	0.0	0.0	0.0	1.0	3.6	0.	
## 300 - Autre commune multipolarisée	6.2	11.8	0.0	4.2	0.5	20.4	19.2	23.	
## 400 - Commune isolée hors influence des pôles	9.4	14.7	59.1	16.7	0.0	14.9	25.3	12.	

Attention: la fonction `prop.table()` a été améliorée avec le changement de version de R base. La version <3.6.2 ne permettait pas d'utiliser le nom de la variable (ici "REG") pour spécifier le mode de calcul des fréquences, l'option était alors `margin=` (1: en ligne, 2: en colonne)

7.6 Exercice 4 : calcul de statistiques

- Utilisez la fonction `summary()` pour obtenir un résumé de l'ensemble des variables de la table df
- Calculez maintenant les moyenne, médiane, écart-type et variance de la variable de densité de population. Que constatez-vous ?
- Utilisez le paramètre `na.rm = TRUE` pour gérer les valeurs manquantes
- Calculez à présent les quartiles puis déciles de cette variables
- Calculez la version centrée réduite de la variable de densité. Rappel : on calcule la version centrée réduite d'une variable X en lui appliquant la transformation suivante : $\frac{X - \bar{X}}{\sigma_X}$ où \bar{X} est la moyenne empirique de X et σ_X son écart-type

Tableaux croisés :

- Calculer le nombre de communes par type d'espace à l'aide de la fonction `table` , et le pourcentage associé
- Calculer le nombre de communes par région et type d'espace, et les pourcentages associés

Pour aller plus loin et ajouter des variables de pondération, calculer les profils-ligne ou profils-colonne, rendez-vous au [module 3 “Statistiques descriptives”](#) ou demander à un GF (Gentil Formateur).

Chapitre 8 Premiers graphiques

8.1 Package ggplot2

Pour réaliser des graphiques, nous choisissons de nous servir du package `ggplot2`, qui permet de faire de meilleures réalisations que les fonctions basiques. Il est intégré dans le méta-package `tidyverse` donc il n'est pas utile de le re-charger dans notre session. Mais si vous devez le faire, le code est le suivant :

```
install.packages("ggplot2")
library("ggplot2")
library(dplyr)
```

La fonction `ggplot()` fonctionne d'une manière particulière. La structure ressemble à ceci :

```
ggplot(TableEnEntree, aes(VariablesATracer)) + geom_FonctionAChoisir()
```

Pour découvrir les nombreuses possibilités de `ggplot2`, vous pouvez vous référer au [Module 5 : « Valoriser ses données avec R »](#) ou consulter les sites suivants :

- [Version anglaise](#)
- [Version française](#)

La fonction `aes()` (pour “aesthetics”), utilisée dans l'instruction `ggplot()` permet de définir les données à tracer. On y indique les dimensions que l'on veut représenter sur le graphique. On peut représenter jusqu'à 5 dimensions sur un même graphique, mais attention à la lisibilité !

- 2 variables quanti : x en fonction de y \rightarrow 2 dimensions (nuage de points)
- taille du point \rightarrow 3e dimension (quanti)
- couleur des points \rightarrow 4e dimension (quali)
- juxtaposer des graphiques en fonction d'une variable quali \rightarrow 5e dimension !

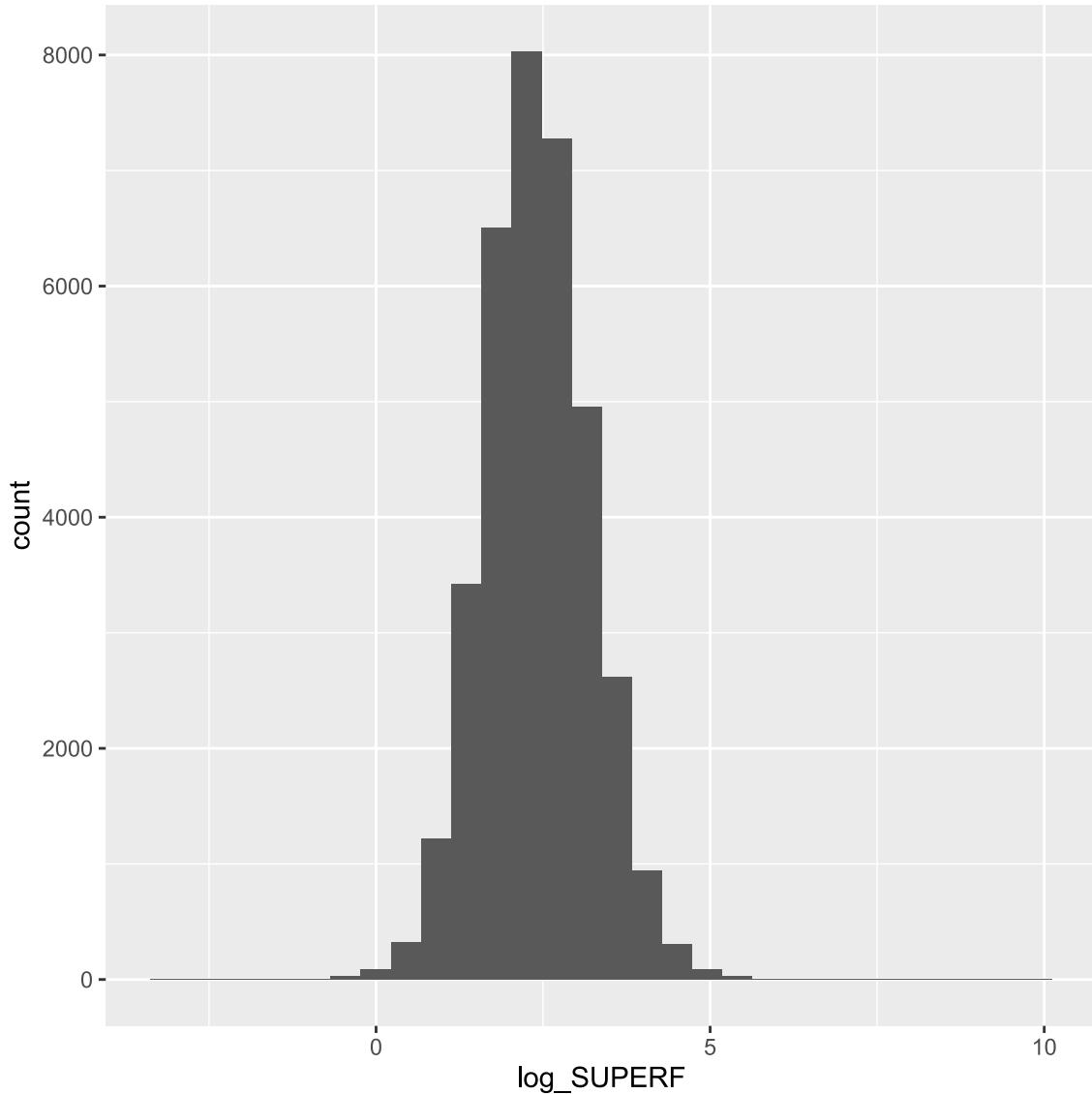
8.2 Histogramme

Si on désire un histogramme de log_SUPERF, on fera appel à la fonction `geom_histogram()`. Ce graphique ne présente qu'une seule dimension (la variable quanti dont on veut visualiser la distribution)

```
rm(list = ls())

base <- read.csv(file = "extdata/Base_synth_territoires.csv",
                 header = T, sep=";", dec=",") %>%
  select(1:24) %>%
  mutate(log_SUPERF = log(SUPERF),
        REG = as.factor(REG),
        densite = P14_POP / SUPERF,
        tx_natal = 1000 * NAISD15 / P14_POP,
        tx_mort = DECESD15 / P14_POP)

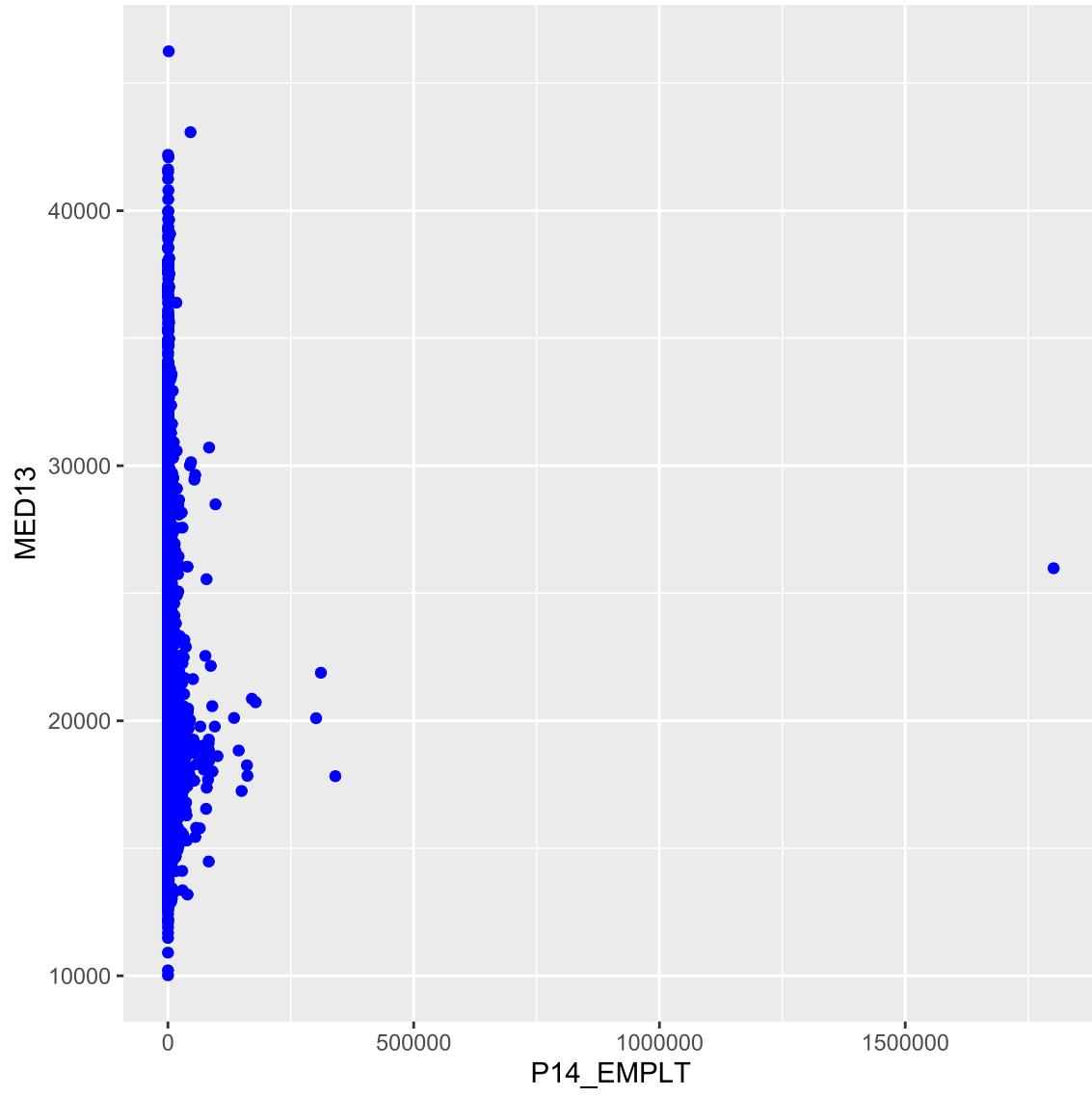
ggplot(data = base, mapping = aes(x = log_SUPERF)) + geom_histogram()
```



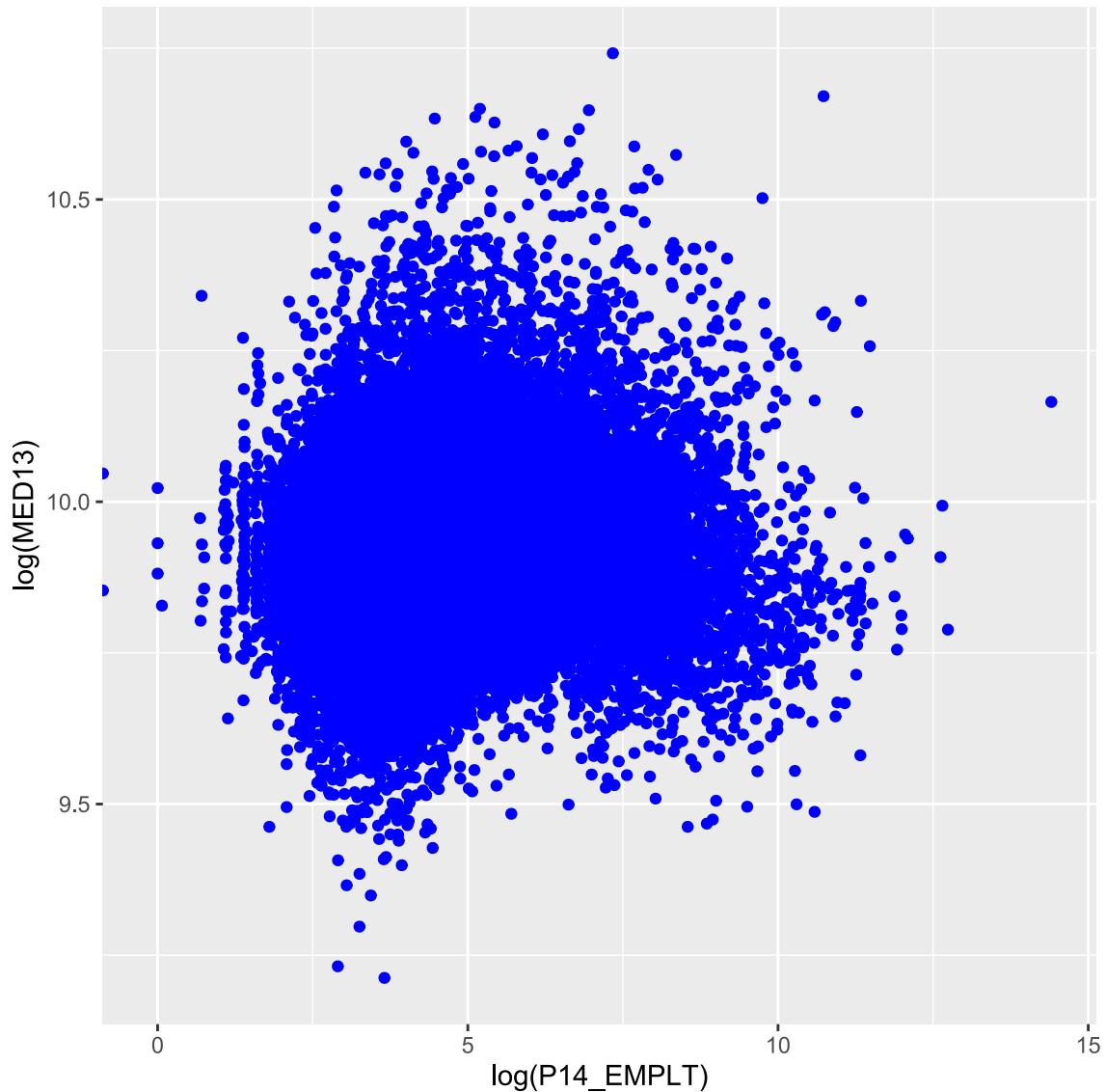
8.3 Nuages de points

Selon les graphiques que l'on veut tracer, on peut renseigner plusieurs variables. Dans le cas d'un nuage de points, par exemple, on croise 2 variables quantitatives :

```
ggplot(base, aes(x = P14_EMPLT, y = MED13)) + geom_point(colour = "blue")
```



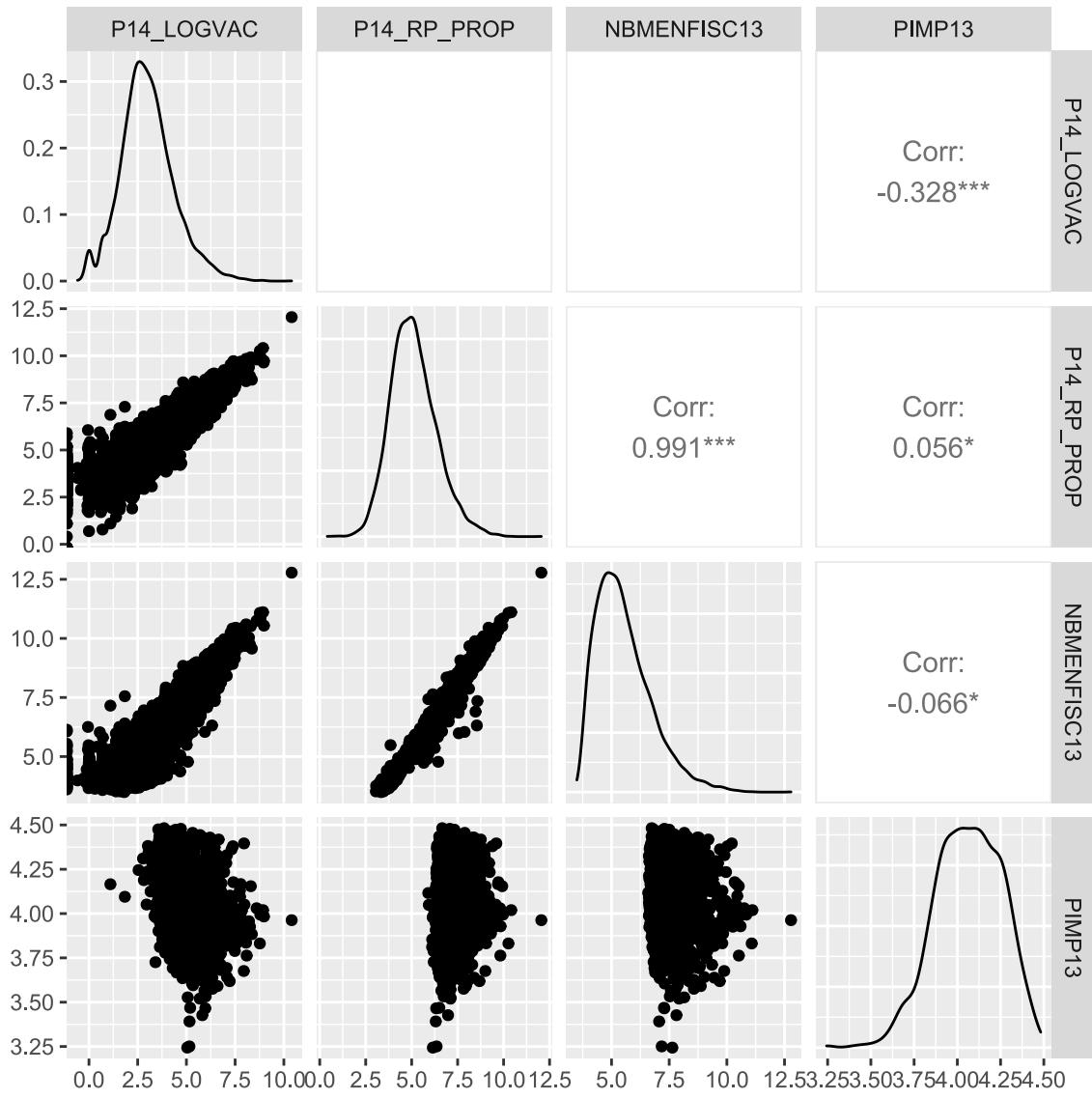
```
ggplot(base, aes(x = log(P14_EMPLT), y = log(MED13))) + geom_point(colour = "blue")
```



8.4 Matrice de nuages

Pour explorer son jeu de données, on peut réaliser un nuage de points pour plusieurs croisements de variables possibles. Ici, en conservant quelques variables quantitatives, on peut réaliser un ensemble de graphiques. Cette “matrice” de nuages fonctionne avec le package `GGally`.

```
# install.packages("GGally")
library("GGally")
num <- select(base, P14_LOGVAC:PIMP13) %>% sample_n(10000) %>% log()
ggpairs(num) ## fonction ggpairs() de GGally
```



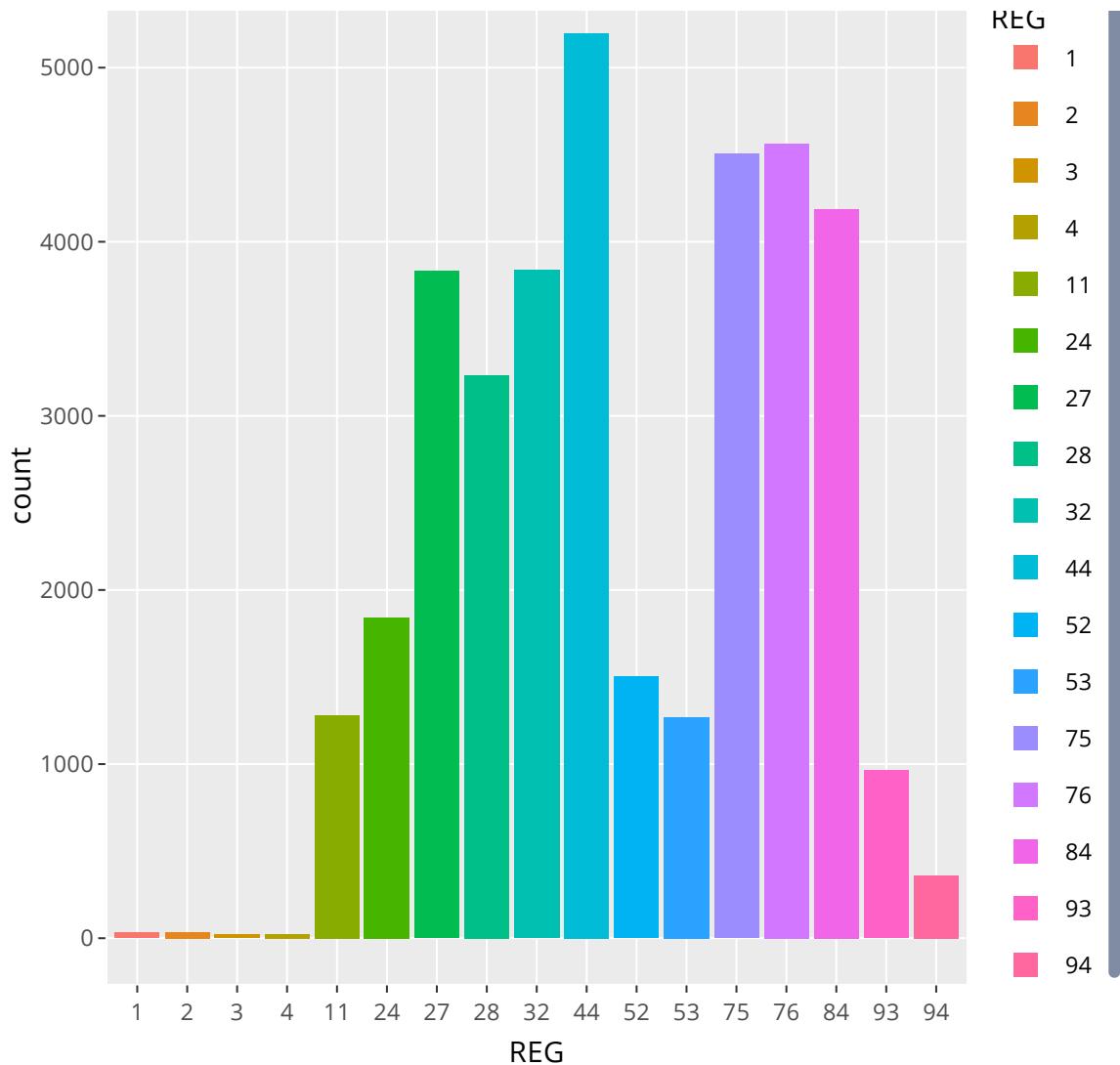
8.5 Bonus : faire un graphique “dynamique”

Une fois qu'on a généré un graphique avec ggplot, on peut le passer dans la fonction `ggplotly()` (package `plotly`) qui permet de le rendre dynamique.

```
library(plotly)

g <- ggplot(data = base, aes(x = REG, fill = REG)) +
  geom_bar()

ggplotly(g)
```



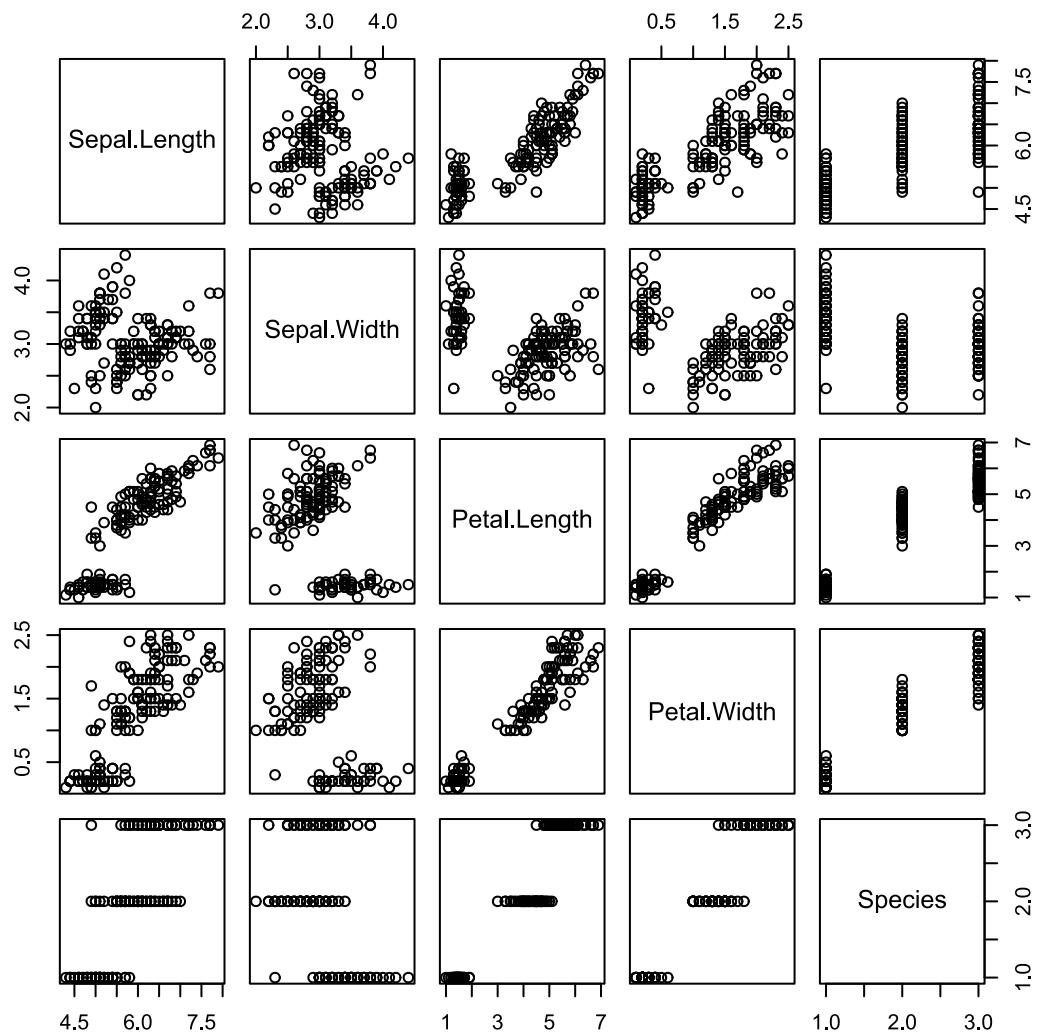
8.6 Exercice : créer des graphiques

À l'aide de l'aide mémoire `ggplot2` :

- Réaliser un histogramme de la population communale
- Transformer les données avec la fonction `log` pour y voir plus clair
- Faire un barplot du nombre de communes par REG
- Utiliser le paramètre `fill` de la fonction `aes()` pour améliorer le graphique
- Réaliser un graphique (nuage de points) croisant la densité de population et le taux de mortalité
- Ajouter une dimension supplémentaire avec la couleur des points (paramètre `color` de `aes()`)

Note : avec les fonctions de base, on peut obtenir de nombreux graphiques avec très peu de code, mais moins jolis :

```
plot(iris)
```



Des possibilités infinies à approfondir dans [les modules 3 et 5 !!](#)

Chapitre 9 Sauvegarder son travail

Après avoir réalisé ces traitements, on peut exporter son travail afin de le finaliser dans un tableur ou un traitement de texte en vue d'une publication ou autre. On peut donc avoir à exporter soit des tableaux qu'on retouchera par la suite, soit des images de ses graphiques qu'on intégrera dans un document. Une dernière possibilité consiste à sauvegarder un ensemble d'objets R dans un seul fichier (RData) afin de retrouver son environnement de travail facilement en rouvrant une session de R.

9.1 Exportation des résultats

- Exporter une table en csv

```
res <- summary(base)

write.table(x = res, file = 'outputs/resultat_R.csv', sep = ';', row.names = FALSE)
# row.names=F pour éviter un décalage entre première ligne et les suivantes

write.table(x = base, file = 'outputs/base_R.csv', sep = ';', row.names = FALSE)
```

- Exporter un graphique pour l'intégrer à un document

```
png('outputs/mongraphe.png') # Alloue et ouvre le fichier où inscrire le graphe
ggplot(base, aes(x = P14_EMPLT, y = MED13)) +
  geom_point(colour = "blue")
dev.off()                  # Ferme le fichier
```

\(\rightarrow\) Beaucoup d'autres fonctions : jpeg , pdf , postscript , svg ...

9.2 Environnement et .RData

Il est possible de sauvegarder des objets R (*dataframe*, *vecteur*, etc...) directement sur son ordinateur. Une liste d'objets R est enregistrée sous le format `.RData`.

```
save(list = ls(), file = "outputs/env_entier.RData") # sauvegarde de tout l'environnement
rm(list = ls()) # suppression de notre environnement dans R
load("outputs/env_entier.RData") # chargement de l'environnement stocké sur l'ordinateur

save(base, res, file = "outputs/petit_env.RData") # sauvegarde des éléments base et V1
rm(list = ls()) # suppression de notre environnement
load("outputs/petit_env.RData")
```

Avantages

- Un seul fichier peut contenir des dataframes, des graphiques, des fonctions, etc.
- La lecture est très rapide (\(\Rightarrow\)) utile pour les gros volumes de données

Chapitre 10 Aller plus loin avec les objets et la programmation fonctionnelle

Ce qui a été présenté dans ce module repose sur les fonctions du package `tidyverse`. Cette approche tend à se généraliser depuis quelques années, mais quand on cherche la réponse à un problème sur Internet, on trouve d'autres façons de programmer en R, qui font appel aux fonctions du package `base` et non du `tidyverse` \(\Rightarrow\) Cette partie donne quelques clés de compréhension.

10.1 Les objets dans R, plus de détails

Rappel : en informatique, un objet est défini par : ses *attributs* et ses *méthodes* (fonctions). Dans l'exemple du jeu d'échec, chaque pièce peut être vue comme un objet :

- sa position sur le plateau constitue ses attributs
- sa façon de se déplacer peut être vue comme une fonction qui ne s'applique qu'à ce type de pièce, donc une méthode

R est un langage orienté objet ; ces objets permettent de structurer les données selon leurs caractéristiques \(\Rightarrow\) on retrouve les données dans les attributs. Les méthodes sont en général transparentes pour l'utilisateur (cf. utilisation des fonctions `summary` , `plot` ...). Les objets les plus courants sont les suivants :

- **Vecteurs** : suite unidimensionnelle de **valeurs** ayant le même type.
- **Facteurs** : vecteur qui prend un nombre limité de modalités (exemple : sexe). Il est défini par les niveaux (`levels`) et les libellés associés (`labels`).
- **Matrice et arrays** : suites multidimensionnelles de **valeurs** (matrices=dimension 2 ; array=dimension n). A la différence d'une dataframe, les valeurs d'une matrice sont toutes du même type. Les arrays peuvent être très puissants pour gérer des millésimes.
- **Liste** : ensemble d'**objets** différents. On peut stocker un vecteur alphanumérique + une matrice numérique dans une liste.

- **Tableaux** (`data.frame`) : Objet qui ressemble le plus aux tables Excel, SAS ou SPSS... : description d'individus statistiques (observations, en ligne) par des caractéristiques (variables, en colonnes).
- **Fonctions** : Objets particuliers qui donnent un *résultat* à partir de paramètres en entrée.
- **Autres objets** : Il existe un très grand nombre d'objets *ad hoc* dans R. Par exemple
 - `ts` (time serie) pour les séries temporelles,
 - `lm` (linear model) qui contient tous les résultats d'une régression linéaire...
 - des graphiques
 - On peut même en définir de nouveaux soi-même !

10.2 Créez une nouvelle fonction en R

La fonction est un objet comme les autres, qu'on crée avec l'opérateur d'affectation. Elle est définie par des paramètres et elle se termine par la fonction `return()`. On reprend l'exemple du calcul de l'IMC

```
calcul_IMC <- function (poids, taille)
{
  ## La taille est exprimée en mètres
  imc <- poids / taille ^ 2
  return (imc)
}
calcul_IMC (poids = 80, taille = 1.89)
```

```
## [1] 22.39579
```

```
calcul_IMC (poids = 60, taille = 1.55)
```

```
## [1] 24.97399
```

10.3 Les boucles conditionnelles

Les commandes `if` et `else` sont bien entendues utilisables. Le “then” n’existe pas : il est implicite après les accolades.

```
diag_IMC <- function(poids,taille)
{
  imc <- poids / taille ^ 2
  if (imc < 18.5) {diag <- "maigre"}
  else if (imc < 25) {diag <- "normal"}
  else {diag <- "surpoids"}
  return (diag)
}
diag_IMC (poids=60,taille=1.89)
```

```
## [1] "maigre"
```

```
diag_IMC (poids=80,taille=1.89)
```

```
## [1] "normal"
```

```
diag_IMC (poids=80,taille=1.55)
```

```
## [1] "surpoids"
```

10.4 Les boucles

On peut utiliser les boucles classiques : `repeat` , `while` , `for` :

```
for (pp in seq(from = 50, to = 100, by = 5))  
{  
  print(paste ("Taille = 1,70m, poids =", pp, "Diagnostic :",
    diag_IMC (poids = pp, taille = 1.70)))  
}
```

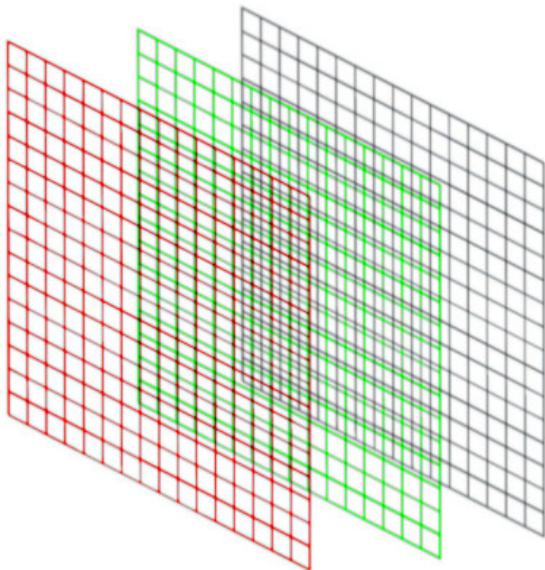
```
## [1] "Taille = 1,70m, poids = 50 Diagnostic : maigre"  
## [1] "Taille = 1,70m, poids = 55 Diagnostic : normal"  
## [1] "Taille = 1,70m, poids = 60 Diagnostic : normal"  
## [1] "Taille = 1,70m, poids = 65 Diagnostic : normal"  
## [1] "Taille = 1,70m, poids = 70 Diagnostic : normal"  
## [1] "Taille = 1,70m, poids = 75 Diagnostic : surpoids"  
## [1] "Taille = 1,70m, poids = 80 Diagnostic : surpoids"  
## [1] "Taille = 1,70m, poids = 85 Diagnostic : surpoids"  
## [1] "Taille = 1,70m, poids = 90 Diagnostic : surpoids"  
## [1] "Taille = 1,70m, poids = 95 Diagnostic : surpoids"  
## [1] "Taille = 1,70m, poids = 100 Diagnostic : surpoids"
```

10.5 Pour aller plus loin

10.5.1 matrices et arrays

Les matrices et les arrays permettent des calculs rapides et efficaces, et peuvent être très pratiques et optimisent le stockage des données. Ils demandent cependant plus de réflexion en amont quant à leur utilisation. On accède aux éléments avec les [].

Un hypercube de trois dimensions peut être représenté comme suit :



On peut par exemple créer une matrice à 10 lignes et 10 colonnes remplie avec un tirage aléatoire selon une loi normale centrée réduite. De la même façon on peut créer un hypercube avec la fonction avec 10 lignes, 5 colonnes et de profondeur 3, toujours avec un tirage aléatoire selon une loi normale

```
mat <- matrix(rnorm(50), ncol = 5, nrow = 10)
arr <- array(rnorm(150), dim = c(10,5,3))
mat
```

```
##          [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] -1.97845438 -1.0292804 -0.0451246 -0.17092875  0.36493920
## [2,]  0.09380613  0.9538236  0.1718605 -0.28602231  1.86614898
## [3,]  0.52732138 -0.8432101 -1.2282006  0.55350037  1.36510441
## [4,]  0.70328953 -1.2208678  1.2849234 -0.58591939 -0.07607886
## [5,]  0.42622016  0.7211529  0.1783541  1.83756193 -2.40509757
## [6,] -0.17926000 -0.9144441  1.1968805  1.34672578  0.38480824
## [7,] -0.837444801  0.9225914 -0.2801438  0.75241221  0.29743323
## [8,]  0.13087517  1.6810856  0.3247341  0.52951628  1.42803628
## [9,]  0.58819777 -2.5533504  0.9627679  0.08241596  1.37516818
## [10,] 1.30519479 -0.1188781  1.5381373 -0.34137718  0.31902135
```

```
arr
```

```

## , , 1
##
##          [,1]      [,2]      [,3]      [,4]      [,5]
## [1,]  0.04822708 -0.845273458  0.1105436 -0.7735786 -0.6809051
## [2,] -0.79358584 -0.599481007  0.3403277  1.1197879  0.1975281
## [3,] -1.27201975  0.489449820 -0.8254127  0.1748668  1.5450289
## [4,]  0.41315565 -0.601574713 -0.2424114  1.6348745 -0.7180578
## [5,]  0.61434049 -0.562798594 -0.3257788 -1.3222006  1.1067603
## [6,] -1.48550658 -1.290653266 -1.3896863  1.2459762 -1.5472859
## [7,] -0.55967403 -0.787368862 -0.6655275  1.9764144 -1.5748639
## [8,] -0.57809964  0.004800054 -1.5368412  0.4221216  1.0246647
## [9,]  0.59776298 -0.055766499  1.6571962 -0.4023696  0.9557019
## [10,] 2.51471221  0.534561535  2.2038455 -1.1496743 -0.3745959
##
## , , 2
##
##          [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] -0.54886156 -0.49925880  0.24132365  0.1024592 -0.83562104
## [2,]  0.47579110 -1.05392837  0.66544966 -0.1049089  2.39313176
## [3,] -0.18736715 -1.22174816  2.87008075  0.4193938  0.07039024
## [4,] -1.19388001  1.25178413 -0.09673511  0.1011698 -0.85548087
## [5,]  2.74047546 -1.10903838 -0.70918484  0.2225875  0.80310590
## [6,]  0.50706548  1.67403411  2.21023576 -0.3217425  1.28493379
## [7,]  0.03474085  0.04201808 -0.44172951  0.8012551 -0.37494874
## [8,]  0.28469996 -0.04721744  0.07427556  0.6184994 -0.24891635
## [9,]  0.10310304  0.96419547 -2.68336999  1.3843293  1.64970050
## [10,] 0.48266435  0.74357933 -1.63809934  0.6522574 -1.01342810
##
## , , 3
##
##          [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] -2.02685011  1.6446183251  0.7012437  1.61546321  0.03401273
## [2,]  0.06697214 -1.8452351803  0.7048438 -0.56612930 -0.23531098
## [3,]  0.97884632  1.6201831027 -0.8161593  0.32345200  0.77569740
## [4,]  0.54992538 -0.0001044614  0.3330132  2.25263549  0.69898532

```

```
## [5,] -1.94102492  0.7045413773 -1.5330500  0.13571626 -1.53540167
## [6,] -0.38325492  0.9556258025 -0.7651049  1.29022514  0.71557323
## [7,]  1.85127463  0.5200881435 -0.7334773  0.58697883 -0.49056456
## [8,]  2.36538635 -0.3697448176  0.8622206 -0.70999956 -1.46458241
## [9,]  1.09606728 -0.1190142980 -1.5062966  2.46261529 -1.18181951
## [10,] -0.19283256  0.5383142146 -0.3514569 -0.01789158  0.58347299
```

Pourquoi s'embêter avec ça ? Parce qu'on peut appliquer des fonctions facilement sur les lignes, colonnes et autres dimensions grâce à la fonction `apply()`. Exemple : résultats de validations croisées par bloc, simulations de loi selon différents paramètres. Et on calcule facilement des statistiques “marginales.”

Par, exemple, sur une matrice, on peut calculer des statistiques par lignes :

```
apply(mat, MARGIN = 1, FUN=mean)
```

```
## [1] -0.57176979  0.55992337  0.07490310  0.02106938  0.15163830  0.36694207  0.17096906
```

Ou par colonnes :

```
apply(mat, MARGIN = 2, FUN=mean)
```

```
## [1]  0.07797425 -0.24013775  0.41041888  0.37178849  0.49194834
```

Sur notre hypercube de type `array`, on peut aussi calculer des stats sur ses différentes dimensions :

```
apply (arr, MARGIN = 3, FUN=mean)
```

```
## [1] -0.04056687  0.21366531  0.16365373
```

```
apply (arr, MARGIN = c(2,3), FUN = mean)
```

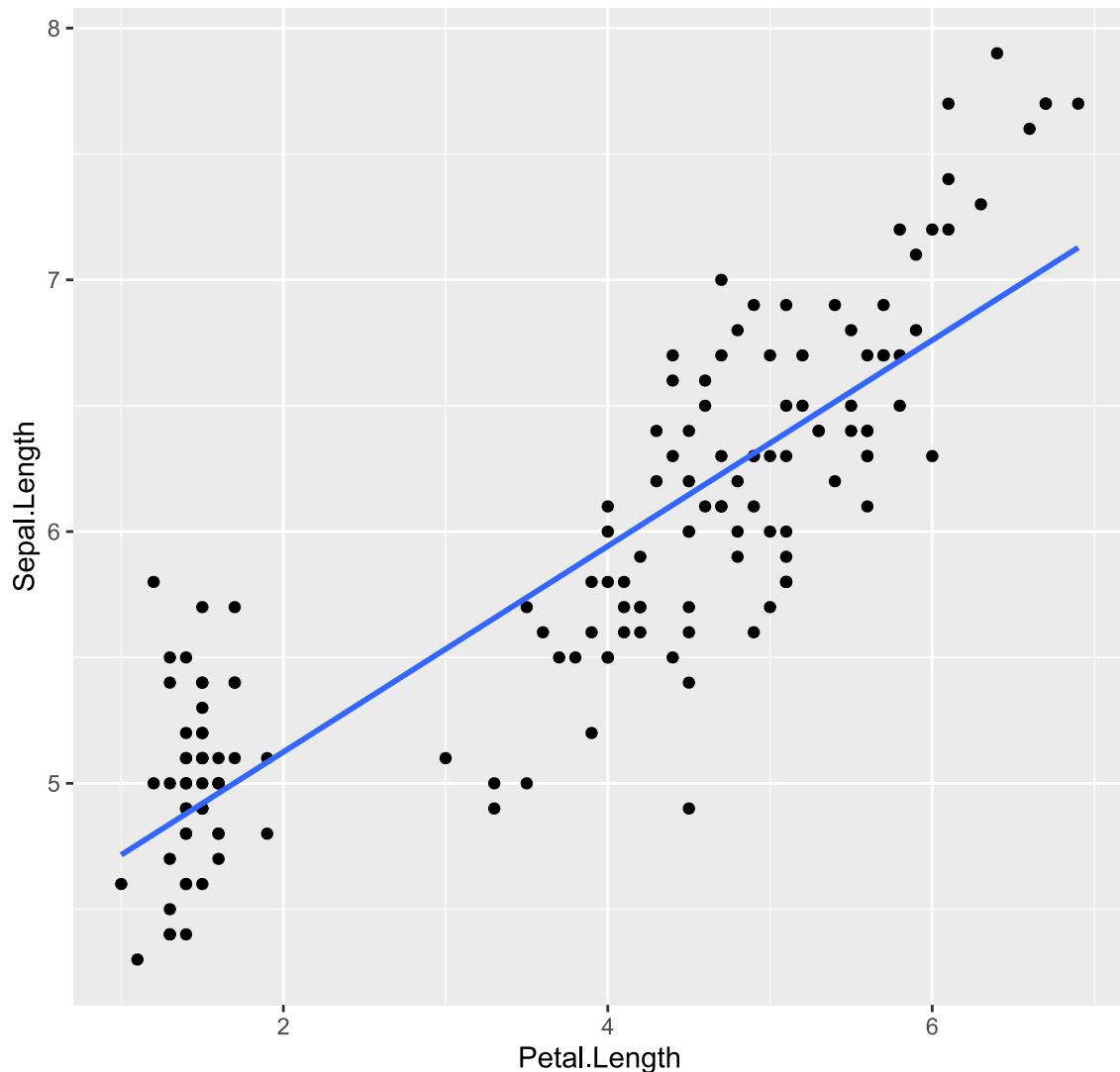
```

##          [,1]      [,2]      [,3]
## [1,] -0.05006874 0.26984315 0.2364510
## [2,] -0.37141050 0.07444200 0.3649272
## [3,] -0.06737450 0.04922466 -0.3104224
## [4,]  0.29262184 0.38753001 0.7373066
## [5,] -0.00660247 0.28728671 -0.2099937

```

10.5.2 Inspection d'un objet : la régression

Illustration de la régression linéaire simple



La régression linéaire consiste à exprimer une variable Y en fonction d'une variable X dans une fonction linéaire. C'est à dire qu'on cherche a et b tels que : $Y = a \cdot X + b + \epsilon$ où ϵ est le résidu de la régression. On utilise dans cet exemple la table des iris de Fisher,

existant dans R base qu'il suffit d'appeler avec `data(iris)` (il existe d'autres dataframe inclus dans les packages et qui sont utilisés en exemple dans l'aide).

```
data ("iris")
str (iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
## $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
## $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
## $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
## $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
## $ Species      : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1 ...
```

Faire la régression de la Sepal.Length sur Petal.length à l'aide de la fonction `lm()`

```
lm (data = iris, formula = Sepal.Length ~ Petal.Length)
```

```
##
## Call:
## lm(formula = Sepal.Length ~ Petal.Length, data = iris)
##
## Coefficients:
## (Intercept)  Petal.Length
##           4.3066        0.4089
```

On a les paramètres a et b mais on aimerait en savoir plus... Au moins la qualité d'ajustement (le R^2) par exemple), et un graphique des résidus pour détecter une éventuelle structure. Pour cela, stocker le résultat dans un nouvel objet, et explorez-le avec les fonctions `str()`,
`summary()` et `plot()`

```
reg <- lm(data = iris, formula = Sepal.Length ~ Petal.Length)
str (reg)
```

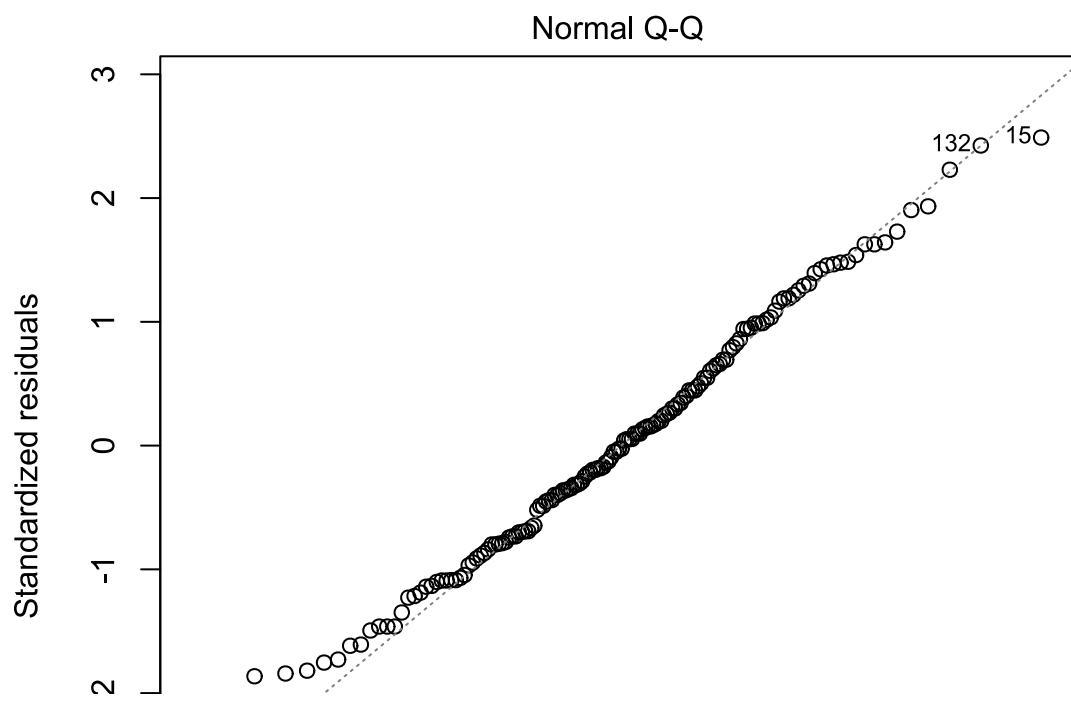
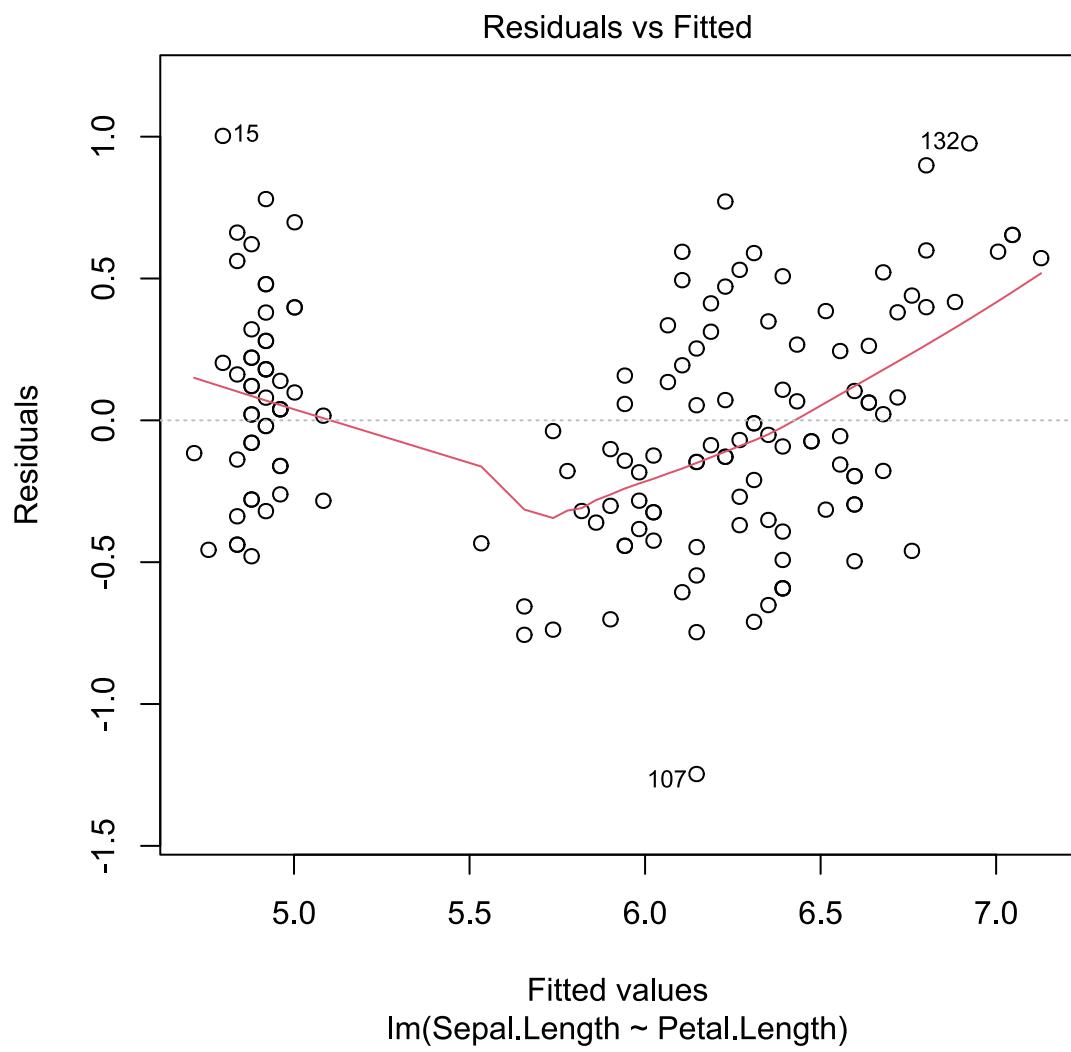
```
## List of 12
## $ coefficients : Named num [1:2] 4.307 0.409
##   ..- attr(*, "names")= chr [1:2] "(Intercept)" "Petal.Length"
## $ residuals     : Named num [1:150] 0.2209 0.0209 -0.1382 -0.32 0.1209 ...
##   ..- attr(*, "names")= chr [1:150] "1" "2" "3" "4" ...
## $ effects       : Named num [1:150] -71.566 8.812 -0.155 -0.337 0.104 ...
##   ..- attr(*, "names")= chr [1:150] "(Intercept)" "Petal.Length" "" "" ...
## $ rank          : int 2
## $ fitted.values: Named num [1:150] 4.88 4.88 4.84 4.92 4.88 ...
##   ..- attr(*, "names")= chr [1:150] "1" "2" "3" "4" ...
## $ assign         : int [1:2] 0 1
## $ qr            :List of 5
##   ..$ qr    : num [1:150, 1:2] -12.2474 0.0816 0.0816 0.0816 0.0816 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. . . $ : chr [1:150] "1" "2" "3" "4" ...
##   .. . . . $ : chr [1:2] "(Intercept)" "Petal.Length"
##   .. . .- attr(*, "assign")= int [1:2] 0 1
##   ..$ qraux: num [1:2] 1.08 1.1
##   ..$ pivot: int [1:2] 1 2
##   ..$ tol  : num 1e-07
##   ..$ rank : int 2
##   ..- attr(*, "class")= chr "qr"
## $ df.residual  : int 148
## $ xlevels      : Named list()
## $ call          : language lm(formula = Sepal.Length ~ Petal.Length, data = iris)
## $ terms         :Classes 'terms', 'formula'  language Sepal.Length ~ Petal.Length
##   .. ..- attr(*, "variables")= language list(Sepal.Length, Petal.Length)
##   .. ..- attr(*, "factors")= int [1:2, 1] 0 1
##   .. . . .- attr(*, "dimnames")=List of 2
##   .. . . . . $ : chr [1:2] "Sepal.Length" "Petal.Length"
##   .. . . . . . $ : chr "Petal.Length"
##   .. . .- attr(*, "term.labels")= chr "Petal.Length"
##   .. . .- attr(*, "order")= int 1
##   .. . .- attr(*, "intercept")= int 1
##   .. . .- attr(*, "response")= int 1
```

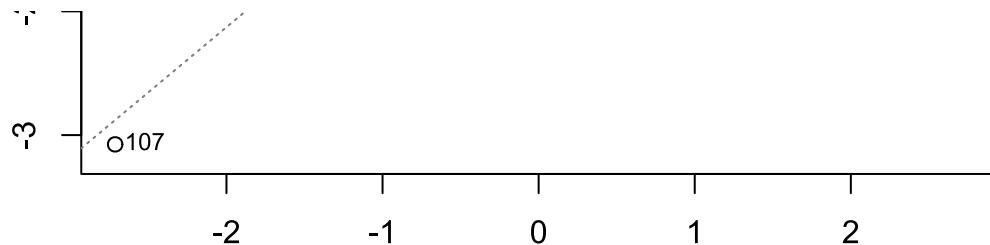
```
## .. .- attr(*, ".Environment")=<environment: R_GlobalEnv>
## .. .- attr(*, "predvars")= language list(Sepal.Length, Petal.Length)
## .. .- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
## .. .- attr(*, "names")= chr [1:2] "Sepal.Length" "Petal.Length"
## $ model      : 'data.frame':   150 obs. of  2 variables:
##   ..$ Sepal.Length: num [1:150] 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##   ..$ Petal.Length: num [1:150] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##   ..- attr(*, "terms")=Classes 'terms', 'formula'  language Sepal.Length ~ Petal.Length
##   ..- attr(*, "variables")= language list(Sepal.Length, Petal.Length)
##   ..- attr(*, "factors")= int [1:2, 1] 0 1
##   ..- attr(*, "dimnames")=List of 2
##     ..$ : chr [1:2] "Sepal.Length" "Petal.Length"
##     ..$ : chr "Petal.Length"
##   ..- attr(*, "term.labels")= chr "Petal.Length"
##   ..- attr(*, "order")= int 1
##   ..- attr(*, "intercept")= int 1
##   ..- attr(*, "response")= int 1
##   ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
##   ..- attr(*, "predvars")= language list(Sepal.Length, Petal.Length)
##   ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
##   ..- attr(*, "names")= chr [1:2] "Sepal.Length" "Petal.Length"
## - attr(*, "class")= chr "lm"
```

summary (reg)

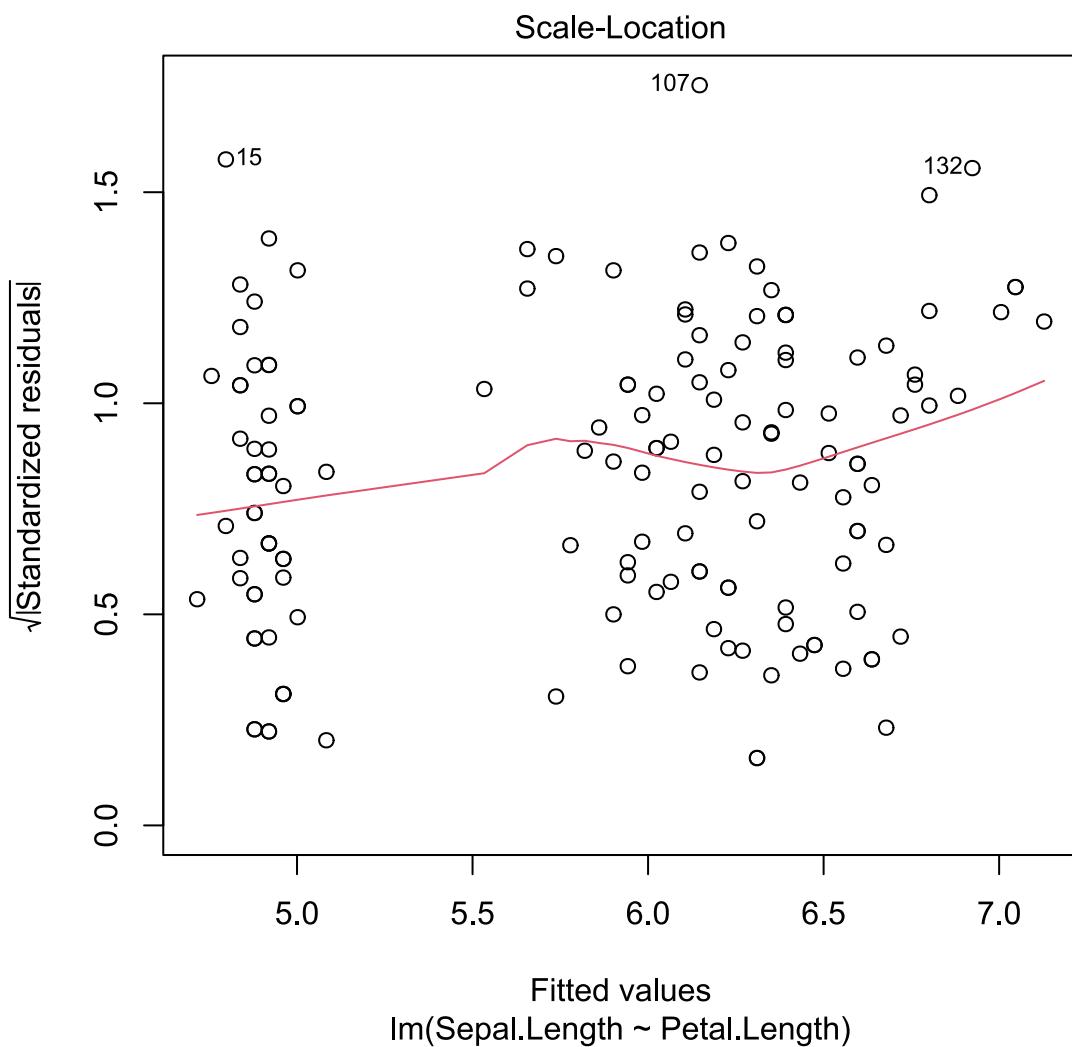
```
##  
## Call:  
## lm(formula = Sepal.Length ~ Petal.Length, data = iris)  
##  
## Residuals:  
##      Min       1Q   Median      3Q     Max  
## -1.24675 -0.29657 -0.01515  0.27676  1.00269  
##  
## Coefficients:  
##                 Estimate Std. Error t value Pr(>|t|)  
## (Intercept) 4.30660    0.07839   54.94 <2e-16 ***  
## Petal.Length 0.40892    0.01889   21.65 <2e-16 ***  
## ---  
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1  
##  
## Residual standard error: 0.4071 on 148 degrees of freedom  
## Multiple R-squared: 0.76, Adjusted R-squared: 0.7583  
## F-statistic: 468.6 on 1 and 148 DF, p-value: < 2.2e-16
```

```
plot (reg)
```

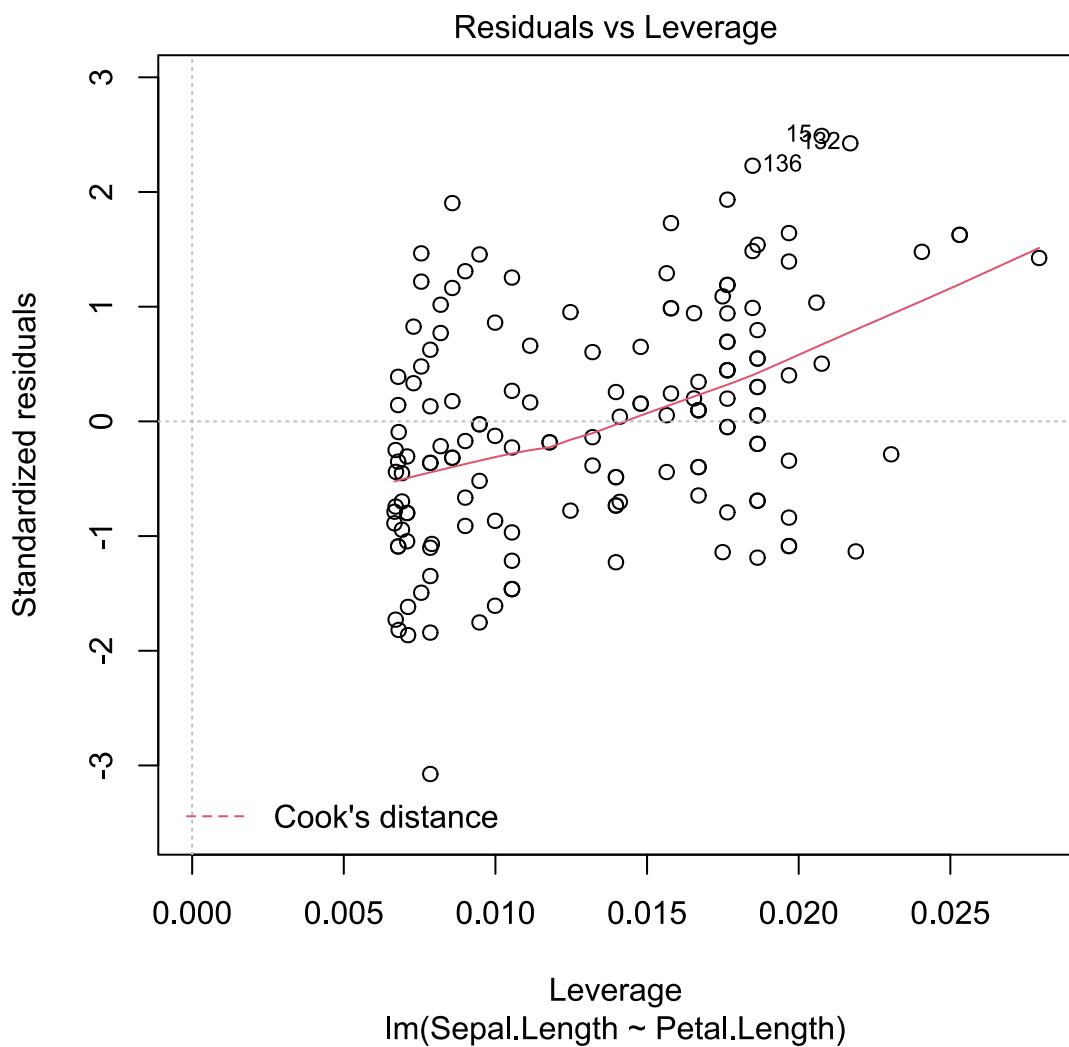





Theoretical Quantiles
Im(Sepal.Length ~ Petal.Length)



Scale-Location
Im(Sepal.Length ~ Petal.Length)



Les **méthodes** `summary`, `print` et `plot` sont implémentées pour tous les objets en R, et on peut les utiliser pour avoir un premier aperçu de ce que l'on obtient avec la fonction.

Chapitre 11 Exercices corrigés

11.1 05 - importer des données et premier coup d'oeil

```
df <- read.csv(file = "extdata/Base_synth_territoires.csv", header = TRUE, sep = ";", dec =
               colClasses = c(rep("character", 2), rep("factor", 4) , rep(NA, 32)))

str(df)
```

```
## 'data.frame': 36689 obs. of 38 variables:
## $ CODGEO : chr "01001" "01002" "01004" "01005" ...
## $ LIBGEO : chr "L'Abergement-Clémenciat" "L'Abergement-de-Varey" "Ambérieu-en-Bu
## $ REG : Factor w/ 17 levels "01","02","03",...: 15 15 15 15 15 15 15 15 15 15 ...
## $ DEP : Factor w/ 100 levels "01","02","03",...: 1 1 1 1 1 1 1 1 1 1 ...
## $ ZAU : Factor w/ 9 levels "111 - Grand pôle (plus de 10 000 emplois)",...: 3 2
## $ ZE : Factor w/ 321 levels "0050 - Mont-de-Marsan",...: 260 248 248 260 263 2
## $ P14_POP : int 767 239 14022 1627 109 2570 743 338 1142 397 ...
## $ P09_POP : int 787 207 13350 1592 120 2328 660 336 960 352 ...
## $ SUPERF : num 15.95 9.15 24.6 15.92 5.88 ...
## $ NAIS0914 : int 40 16 1051 117 8 175 59 12 56 25 ...
## $ DECE0914 : int 25 7 551 41 3 78 20 11 32 10 ...
## $ P14_MEN : num 306 99.3 6161.1 621.1 52.5 ...
## $ NAISD15 : int 13 5 222 15 2 21 11 2 18 4 ...
## $ DECESD15 : int 5 1 121 7 2 9 3 3 5 0 ...
## $ P14_LOG : num 342.7 161.2 6838.4 661.8 71.5 ...
## $ P14_RP : num 306 99.3 6161.1 621.1 52.5 ...
## $ P14_RSECOCC : num 14 47.3 121.6 10.9 10.9 ...
## $ P14_LOGVAC : num 22.74 14.55 555.64 29.85 8.14 ...
## $ P14_RP_PROP : num 260 84.6 2769 473.3 37.7 ...
## $ NBMENFISC13 : int 297 99 6034 617 47 1014 299 140 431 137 ...
## $ PIMP13 : num NA NA 57.4 NA NA ...
## $ MED13 : num 22130 23213 19554 22388 21872 ...
## $ TP6013 : num NA NA 15.1 NA NA ...
## $ P14_EMPLT : num 85.16 12.81 7452.93 280.57 5.95 ...
## $ P14_EMPLT_SAL: num 52.19 4.95 6743.37 206.38 3.96 ...
## $ P09_EMPLT : num 65.57 17.64 7551.68 286.61 5.29 ...
## $ P14_POP1564 : num 463 141.6 8962.8 1043.1 71.3 ...
## $ P14_CHOM1564 : num 33 9.84 1059.73 66.33 7.93 ...
## $ P14_ACT1564 : num 376 121 6681.9 842.1 57.5 ...
## $ ETTOT14 : int 47 22 1316 141 7 203 66 39 54 36 ...
## $ ETAZ14 : int 9 1 7 14 0 21 2 5 5 6 ...
## $ ETBE14 : int 2 3 60 7 0 18 0 2 6 4 ...
## $ ETFZ14 : int 5 1 131 19 0 21 9 1 13 3 ...
## $ ETGU14 : int 25 14 892 85 5 114 45 27 24 18 ...
```

```

## $ ETGZ14      : int 6 4 283 19 1 28 16 6 9 7 ...
## $ ETOQ14      : int 6 3 226 16 2 29 10 4 6 5 ...
## $ ETTEF114     : int 9 2 385 27 0 38 17 6 9 7 ...
## $ ETTEFP1014   : int 0 0 109 5 0 15 2 0 3 0 ...

```

```
head(df)
```

	CODGEO	LIBGEO	REG	DEP	ZAL				
## 1	01001	L'Abergement-Clémenciat	84	01 120	- Multipolarisée des grandes aires urbaines				
## 2	01002	L'Abergement-de-Varey	84	01	112 - Couronne d'un grand pôle				
## 3	01004	Ambérieu-en-Bugey	84	01	112 - Couronne d'un grand pôle				
## 4	01005	Ambérieux-en-Dombes	84	01	112 - Couronne d'un grand pôle				
## 5	01006	Ambléon	84	01	300 - Autre commune multipolarisée				
## 6	01007	Ambronay	84	01	112 - Couronne d'un grand pôle				
##	P14_MEN	NAISD15	DECESD15	P14_LOG	P14_RP	P14_RSECOCC	P14_LOGVAC	P14_RP_PROP	N
## 1	306.00000	13	5	342.73473	306.00000	13.99418	22.740550	260.00000	
## 2	99.33745	5	1	161.16023	99.33745	47.27625	14.546538	84.58436	
## 3	6161.06200	222	121	6838.35437	6161.06200	121.64795	555.644416	2769.00170	
## 4	621.05374	15	7	661.76017	621.05374	10.85505	29.851387	473.32736	
## 5	52.51818	2	2	71.51818	52.51818	10.85714	8.142857	37.65455	
## 6	1028.00000	21	9	1160.00000	1028.00000	56.00000	76.000000	779.00000	
##	P09_EMPLT	P14_POP1564	P14_CHOM1564	P14_ACT1564	ETTOT14	ETAZ14	ETBE14	ETFZ14	ETGU14
## 1	65.566193	463.00000	33.000000	376.00000	47	9	2	5	25
## 2	17.644456	141.62963	9.835391	120.97531	22	1	3	1	14
## 3	7551.682296	8962.84216	1059.728437	6681.86216	1316	7	60	131	892
## 4	286.611037	1043.12909	66.326127	842.14083	141	14	7	19	85
## 5	5.285714	71.34545	7.927273	57.47273	7	0	0	0	5
## 6	491.138876	1614.00000	108.000000	1267.00000	203	21	18	21	114

```
tail(df)
```

```

##      CODGEO          LIBGEO REG DEP          ZAU
## 36684 97419      Sainte-Rose 04 974 400 - Commune isolée hors influence des pôles   €
## 36685 97420      Sainte-Suzanne 04 974           112 - Couronne d'un grand pôle 04€
## 36686 97421      Salazie 04 974 400 - Commune isolée hors influence des pôles   €
## 36687 97422      Le Tampon 04 974       111 - Grand pôle (plus de 10 000 emplois) 04
## 36688 97423 Les Trois-Bassins 04 974           112 - Couronne d'un grand pôle 04€
## 36689 97424      Cilaos 04 974 400 - Commune isolée hors influence des pôles 04
##      DECESD15    P14_LOG    P14_RP P14_RSECOCC P14_LOGVAC P14_RP_PROP NBMENFISC13 PIMP13
## 36684     43 2542.478 2322.000    28.58054 191.8979 1799.954        NA        NA
## 36685    100 8328.616 7686.912    50.94097 590.7629 4450.648        NA        NA
## 36686     46 2987.000 2420.000   174.00000 393.0000 1849.000        NA        NA
## 36687    420 32710.091 29662.460 1004.31703 2043.3140 15372.921        NA        NA
## 36688     54 2890.404 2484.887    67.58626 337.9313 1773.765        NA        NA
## 36689     38 2732.690 2016.000   239.56501 477.1253 1522.480        NA        NA
##      P14_ACT1564 ETTOT14 ETAZ14 ETBE14 ETFZ14 ETGU14 ETGZ14 ET0Q14 ETTEF114 ETTEFP1014
## 36684 2850.155     414    125     44     31    159     59     55     82      8
## 36685 10456.196    1323    136    139    190    694    249    164    267     66
## 36686 3108.000     490    148     29     43    214     73     56     95     11
## 36687 34446.789    5476    565    398    665    3025    999    823   1032    204
## 36688 3524.287     456    42      39     61    223     76     91     59     20
## 36689 2415.666     360    49      17     35    193     49     66     65     26

```

```
names(df)
```

```

## [1] "CODGEO"          "LIBGEO"          "REG"            "DEP"            "ZAU"            "ZE
## [10] "NAIS0914"        "DECE0914"        "P14_MEN"         "NAISD15"        "DECESD15"        "P1
## [19] "P14_RP_PROP"     "NBMENFISC13"    "PIMP13"         "MED13"          "TP6013"         "P1
## [28] "P14_CHOM1564"    "P14_ACT1564"    "ETTOT14"        "ETAZ14"         "ETBE14"         "ET
## [37] "ETTEF114"         "ETTEFP1014"

```

```
class(df)
```

```
## [1] "data.frame"
```

```
typeof(df)
```

```
## [1] "list"
```

11.2 06 - créer, filtrer, sélectionner

```
df <- mutate(df, densite = P14_POP / SUPERF,
             tx_natal = 1000 * NAISD15 / P14_POP,
             tx_mort = 1000 * DECESD15 / P14_POP)
```

```
selection <- select(df, CODGEO, ZAU, REG, DEP,
                     densite, tx_natal, tx_mort)
```

```
S0 <- filter(selection, DEP == "62")
```

```
S1 <- filter(selection, DEP != "62") # tout sauf le 62 :(
S2 <- filter(selection, DEP %in% c("59", "62")) # L'ancien NPdC :
S3 <- filter(selection, !(DEP %in% c("59", "62"))) # Le "sud" de La France
S4 <- filter(selection, densite > 100) # L'urbain
S5 <- filter(selection, DEP == "62" & densite > 100) # Le PdC urbain
S6 <- filter(selection, DEP == "62" | densite > 100) # Le PdC et l'urbain
```

11.3 07 - calcul de statistiques

```
df <- df %>%  
  select(1:24) %>%  
  mutate(densite = P14_POP / SUPERF,  
         tx_natal = 1000 * NAISD15 / P14_POP,  
         tx_mort = 1000 * DECESD15 / P14_POP  
    )  
  
summary(df)
```

##	CODGEO	LIBGEO	REG	DEP	
##	Length:36689	Length:36689	44 : 5198	62 : 895	112 - Couronne d'
##	Class :character	Class :character	76 : 4565	02 : 816	400 - Commune isc
##	Mode :character	Mode :character	75 : 4505	80 : 782	300 - Autre commu
##			84 : 4189	76 : 745	120 - Multipolari
##			32 : 3838	57 : 730	111 - Grand pôle
##			27 : 3831	14 : 707	221 - Petit pôle
##			(Other):10563	(Other):32014	(Other)
##	P14_POP	P09_POP	SUPERF	NAIS0914	DECE0914
##	Min. : 0	Min. : 0	Min. : 0.04	Min. : 0.0	Min. : 0
##	1st Qu.: 197	1st Qu.: 193	1st Qu.: 6.44	1st Qu.: 9.0	1st Qu.: 8
##	Median : 444	Median : 431	Median : 10.81	Median : 23.0	Median : 17
##	Mean : 1838	Mean : 1793	Mean : 17.64	Mean : 114.4	Mean : 77
##	3rd Qu.: 1110	3rd Qu.: 1072	3rd Qu.: 18.58	3rd Qu.: 60.0	3rd Qu.: 43
##	Max. :2220445	Max. :2234105	Max. :18360.00	Max. :150843.0	Max. :69907
##	NA's :821	NA's :821	NA's :821	NA's :821	NA's :821
##	P14_LOG	P14_RP	P14_RSECOCC	P14_LOGVAC	P14_RF
##	Min. : 0.0	Min. : 0.0	Min. : 0.00	Min. : 0.00	Min. : 0
##	1st Qu.: 115.0	1st Qu.: 83.8	1st Qu.: 7.00	1st Qu.: 8.00	1st Qu.: 8
##	Median : 239.1	Median : 183.2	Median : 19.00	Median : 18.00	Median : 17
##	Mean : 970.2	Mean : 802.0	Mean : 91.63	Mean : 76.60	Mean : 74
##	3rd Qu.: 565.0	3rd Qu.: 454.9	3rd Qu.: 49.29	3rd Qu.: 43.75	3rd Qu.: 42
##	Max. :1362181.9	Max. :1147990.9	Max. :107061.99	Max. :107129.02	Max. :107129.02
##	NA's :821	NA's :821	NA's :821	NA's :821	NA's :821
##	TP6013	P14_EMPLT	densite	tx_natal	tx_mort
##	Min. : 5.00	Min. : 0.0	Min. : 0.00	Min. : 0.000	Min. : 0.00
##	1st Qu.: 8.73	1st Qu.: 26.0	1st Qu.: 18.59	1st Qu.: 5.679	1st Qu.: 4.44
##	Median :11.97	Median : 66.8	Median : 40.35	Median : 9.264	Median : 7.87
##	Mean :13.35	Mean : 733.9	Mean : 160.15	Mean : 9.699	Mean : 9.32
##	3rd Qu.:16.80	3rd Qu.: 229.5	3rd Qu.: 94.57	3rd Qu.: 12.931	3rd Qu.: 12.38
##	Max. :44.84	Max. :1801865.8	Max. :27126.14	Max. :111.111	Max. :157.73
##	NA's :32531	NA's :821	NA's :821	NA's :827	NA's :827

```
df %>% pull(densite) %>% mean()
df %>% pull(densite) %>% sd()
df %>% pull(densite) %>% median()
df %>% pull(densite) %>% var()
```

On a des NA car les valeurs manquantes sont absorbantes !

```
df %>% pull(densite) %>% mean(na.rm = T)
df %>% pull(densite) %>% sd(na.rm = T)
df %>% pull(densite) %>% median(na.rm = T)
df %>% pull(densite) %>% var(na.rm = T)
```

```
df <- df %>%
  mutate(std_dens = (densite - mean(densite, na.rm = T)) / sd(densite, na.rm = T))
```

Avantage des variables centrées réduites : on élimine les effets d'unité (d'ordre de grandeur), et on peut donc comparer les distributions de deux variables qui ont des unités différentes (voir module 3)

```
df %>% pull(densite) %>% quantile(na.rm = T)
```

```
##          0%        25%        50%        75%       100%
## 0.00000 18.59047 40.35457 94.57430 27126.14108
```

```
seq(0, 1, 0.1) # vérifier la séquence qu'on souhaite
```

```
## [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

```
df %>% pull(densite) %>% quantile(probs = seq(0, 1, 0.1), na.rm = T)
```

```
##          0%       10%      20%      30%      40%      50%      60%
## 0.00000 10.03439 15.65357 21.84208 29.76144 40.35457 54.82089 7
```

```
t <- table(df$ZAU)
```

```
t
```

```
##          111 - Grand pôle (plus de 10 000 emplois)      112 - Couronne d'un gr
##                                         3285
##          211 - Moyen pôle (5 000 à 10 000 emplois)      212 - Couronne d'un moy
##                                         456
##          222 - Couronne d'un petit pôle      300 - Autre commune multipc
##                                         582
```

```
100 * prop.table(t) %>% round(digits = 4)
```

```
##          111 - Grand pôle (plus de 10 000 emplois)      112 - Couronne d'un gr
##                                         8.95
##          211 - Moyen pôle (5 000 à 10 000 emplois)      212 - Couronne d'un moy
##                                         1.24
##          222 - Couronne d'un petit pôle      300 - Autre commune multipc
##                                         1.59
```

- Deux variables

```
t <- table(df$REG, df$ZAU)
```

```
t
```

```

##  

##      111 - Grand pôle (plus de 10 000 emplois) 112 - Couronne d'un grand pôle 120 - Mult  

## 01                      17                      6  

## 02                      16                      0  

## 03                      3                       3  

## 04                      10                      3  

## 11                     413                     853  

## 24                      103                     734  

## 27                      140                     1299  

## 28                      216                     1126  

## 32                      481                     1505  

## 44                      322                     1721  

## 52                      108                     535  

## 53                      89                      415  

## 75                      333                     1161  

## 76                      258                     1124  

## 84                      548                     1484  

## 93                      220                     229  

## 94                      8                       99  

##  

##      211 - Moyen pôle (5 000 à 10 000 emplois) 212 - Couronne d'un moyen pôle 221 - Peti  

## 01                      0                       0  

## 02                      3                       0  

## 03                      2                       0  

## 04                      2                       0  

## 11                      3                       2  

## 24                      30                      72  

## 27                      31                      122  

## 28                      34                      104  

## 32                      33                      18  

## 44                      54                      102  

## 52                      23                      44  

## 53                      47                      18  

## 75                      51                      81  

## 76                      79                     155

```

```
## 84                      43                      84
## 93                      20                      11
## 94                      1                       2
##
##      300 - Autre commune multipolarisée 400 - Commune isolée hors influence des pôles
## 01                      2                       3
## 02                      4                       5
## 03                      0                      13
## 04                      1                      4
## 11                      7                      0
## 24                     375                     275
## 27                     737                     969
## 28                     762                     396
## 32                     711                     289
## 44                     1155                    815
## 52                     386                     143
## 53                     325                     181
## 75                     1002                    1379
## 76                     877                     1527
## 84                     517                     963
## 93                     107                     269
## 94                     53                      152
```

```
100 * prop.table(t) %>% round(digits = 4)
```

```

##  

##      111 - Grand pôle (plus de 10 000 emplois) 112 - Couronne d'un grand pôle 120 - Mult  

## 01                      0.05                  0.02  

## 02                      0.04                  0.00  

## 03                      0.01                  0.01  

## 04                      0.03                  0.01  

## 11                      1.13                  2.32  

## 24                      0.28                  2.00  

## 27                      0.38                  3.54  

## 28                      0.59                  3.07  

## 32                      1.31                  4.10  

## 44                      0.88                  4.69  

## 52                      0.29                  1.46  

## 53                      0.24                  1.13  

## 75                      0.91                  3.16  

## 76                      0.70                  3.06  

## 84                      1.49                  4.04  

## 93                      0.60                  0.62  

## 94                      0.02                  0.27  

##  

##      211 - Moyen pôle (5 000 à 10 000 emplois) 212 - Couronne d'un moyen pôle 221 - Peti  

## 01                      0.00                  0.00  

## 02                      0.01                  0.00  

## 03                      0.01                  0.00  

## 04                      0.01                  0.00  

## 11                      0.01                  0.01  

## 24                      0.08                  0.20  

## 27                      0.08                  0.33  

## 28                      0.09                  0.28  

## 32                      0.09                  0.05  

## 44                      0.15                  0.28  

## 52                      0.06                  0.12  

## 53                      0.13                  0.05  

## 75                      0.14                  0.22  

## 76                      0.22                  0.42

```

```

##   84           0.12          0.23
##   93           0.05          0.03
##   94           0.00          0.01
##
##      300 - Autre commune multipolarisée 400 - Commune isolée hors influence des pôles
##   01           0.01          0.01
##   02           0.01          0.01
##   03           0.00          0.04
##   04           0.00          0.01
##   11           0.02          0.00
##   24           1.02          0.75
##   27           2.01          2.64
##   28           2.08          1.08
##   32           1.94          0.79
##   44           3.15          2.22
##   52           1.05          0.39
##   53           0.89          0.49
##   75           2.73          3.76
##   76           2.39          4.16
##   84           1.41          2.62
##   93           0.29          0.73
##   94           0.14          0.41

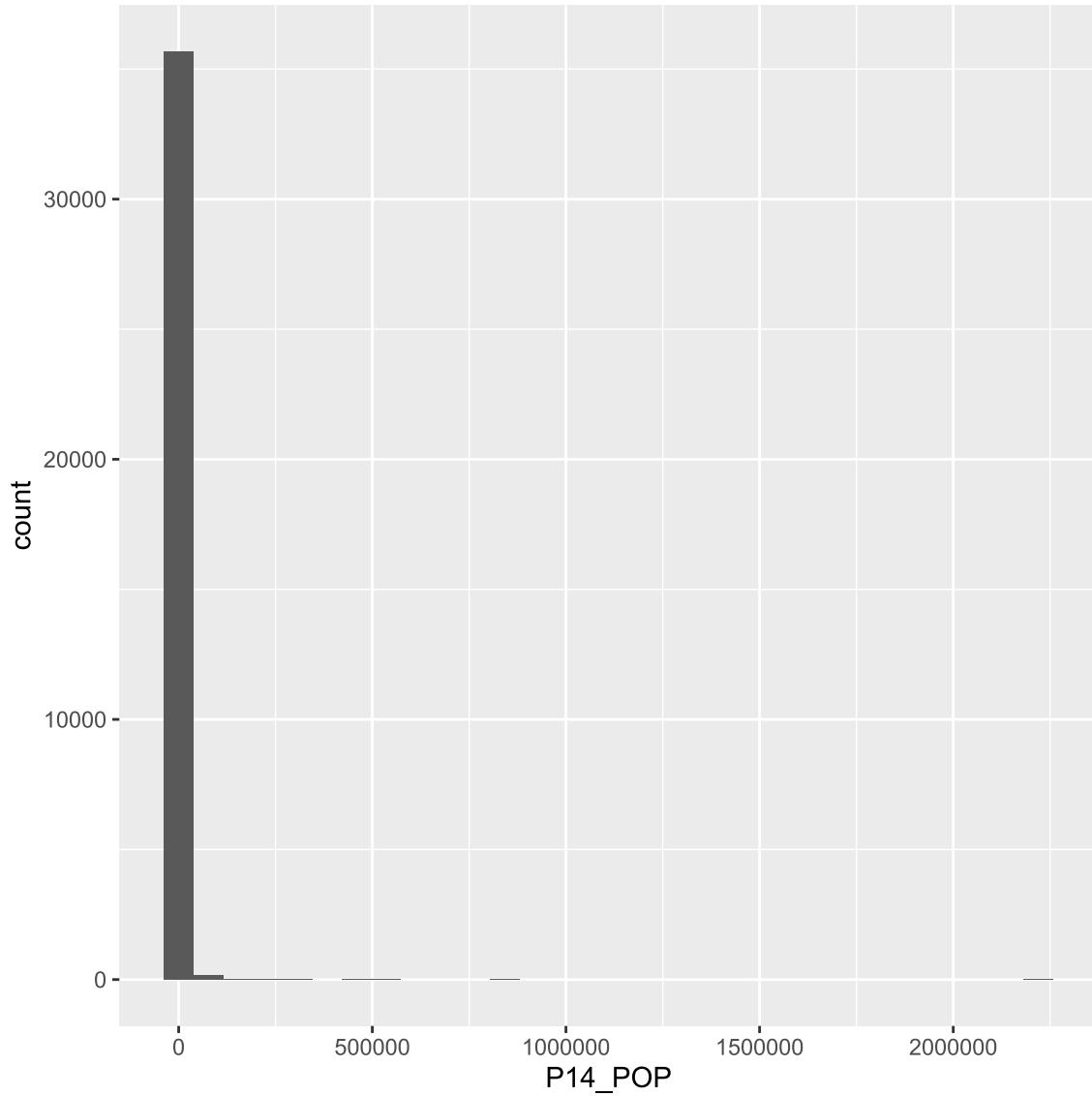
```

11.4 08 - créer des graphiques

```

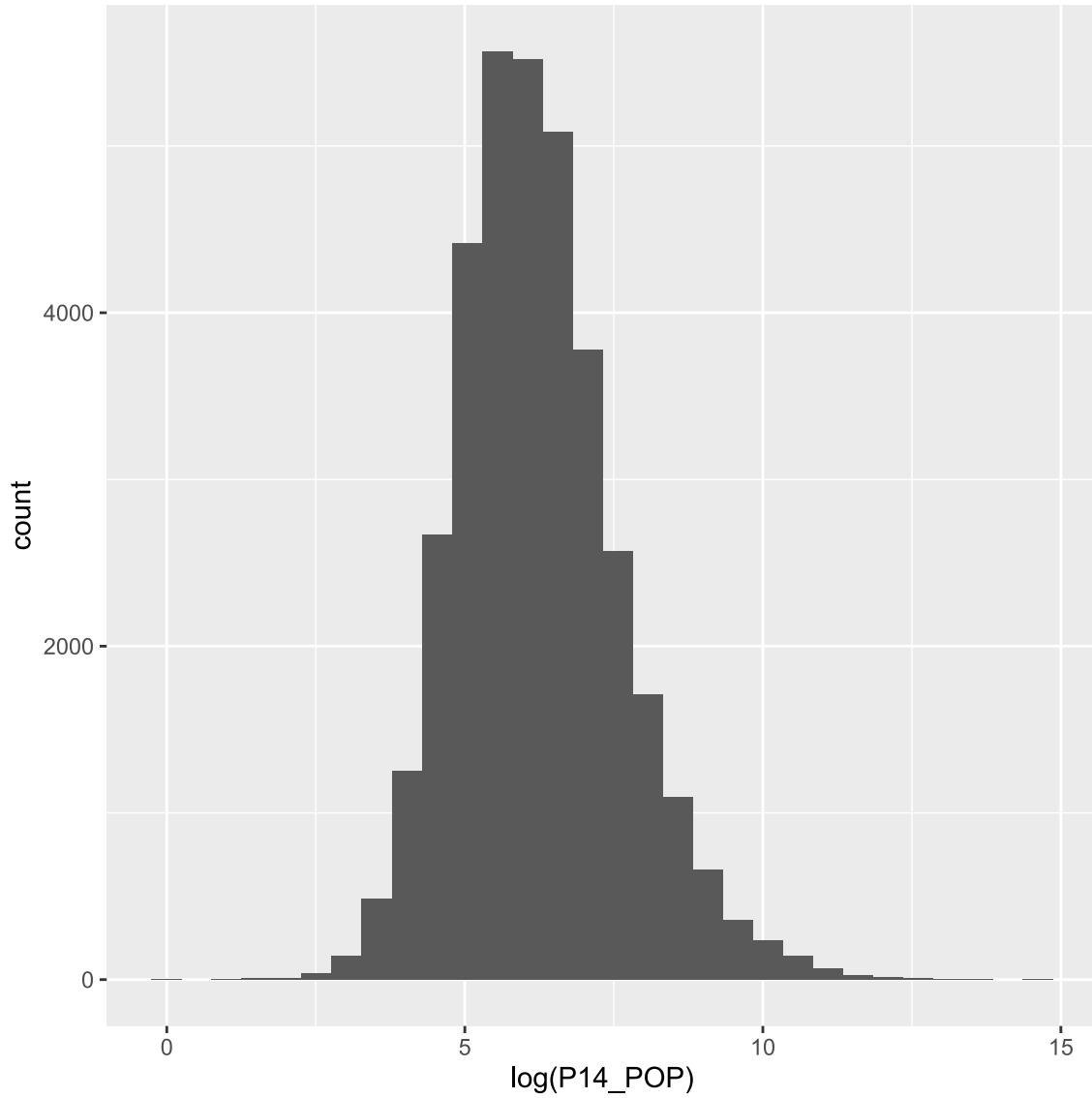
ggplot(data = df, aes(x = P14_POP)) +
  geom_histogram()

```



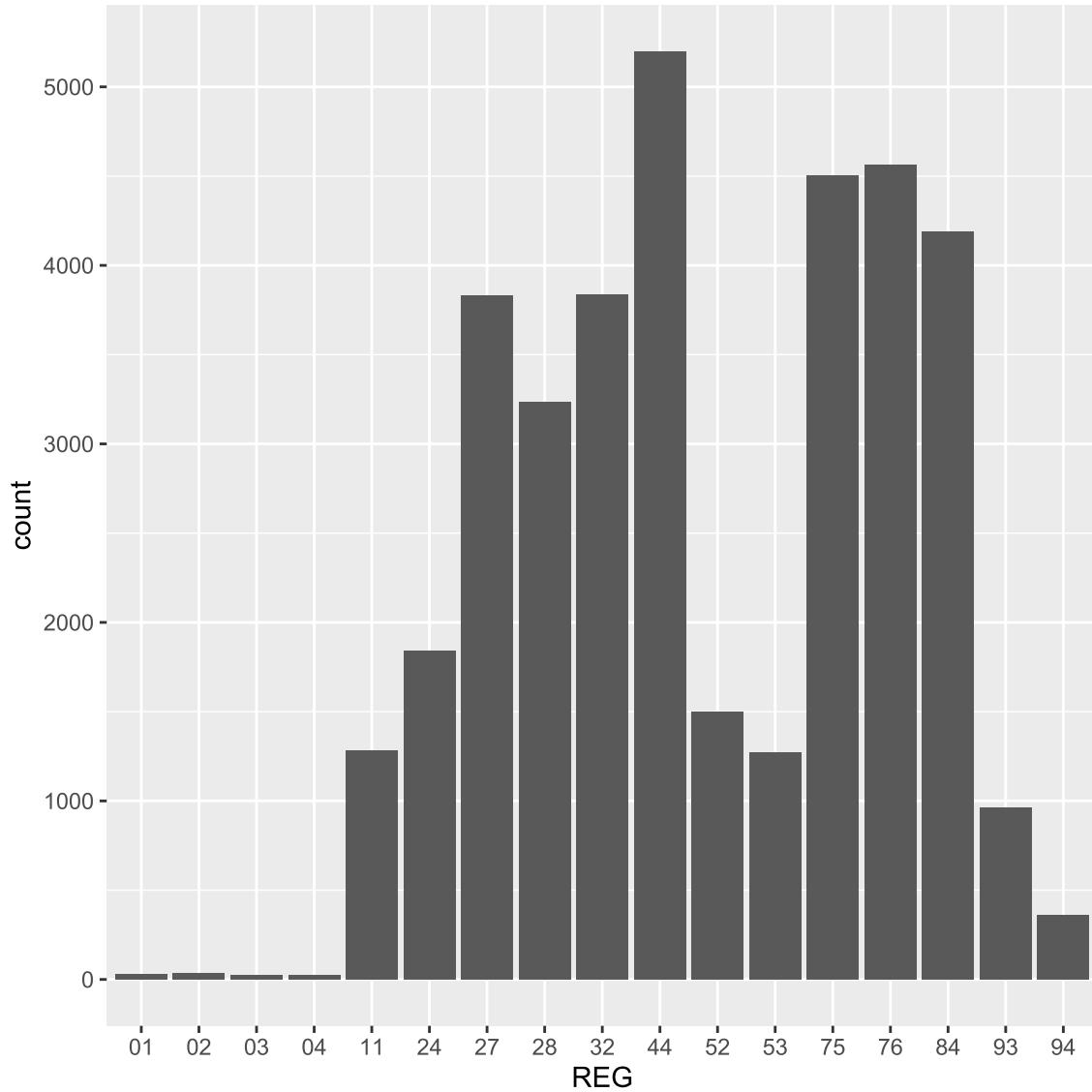
Ce n'est pas très informatif, mais on peut faire une transformation log pour y voir plus clair !

```
ggplot(data = df, aes(x = log(P14_POP))) +  
  geom_histogram()
```



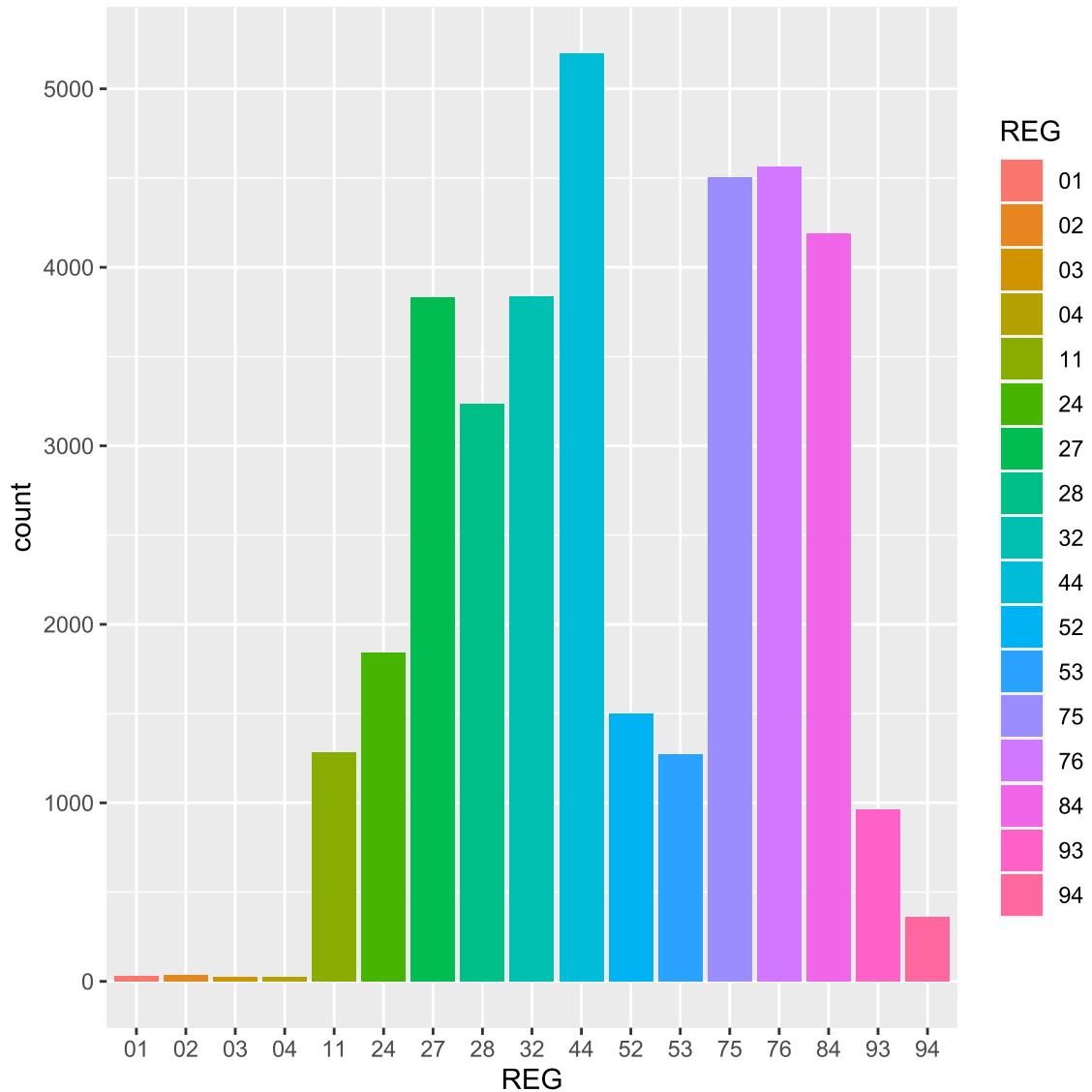
Faites maintenant un barplot (qui n'est pas un histogramme !!!!) du nombre de communes par REG.

```
ggplot(data = df, aes(x = REG)) +  
  geom_bar()
```

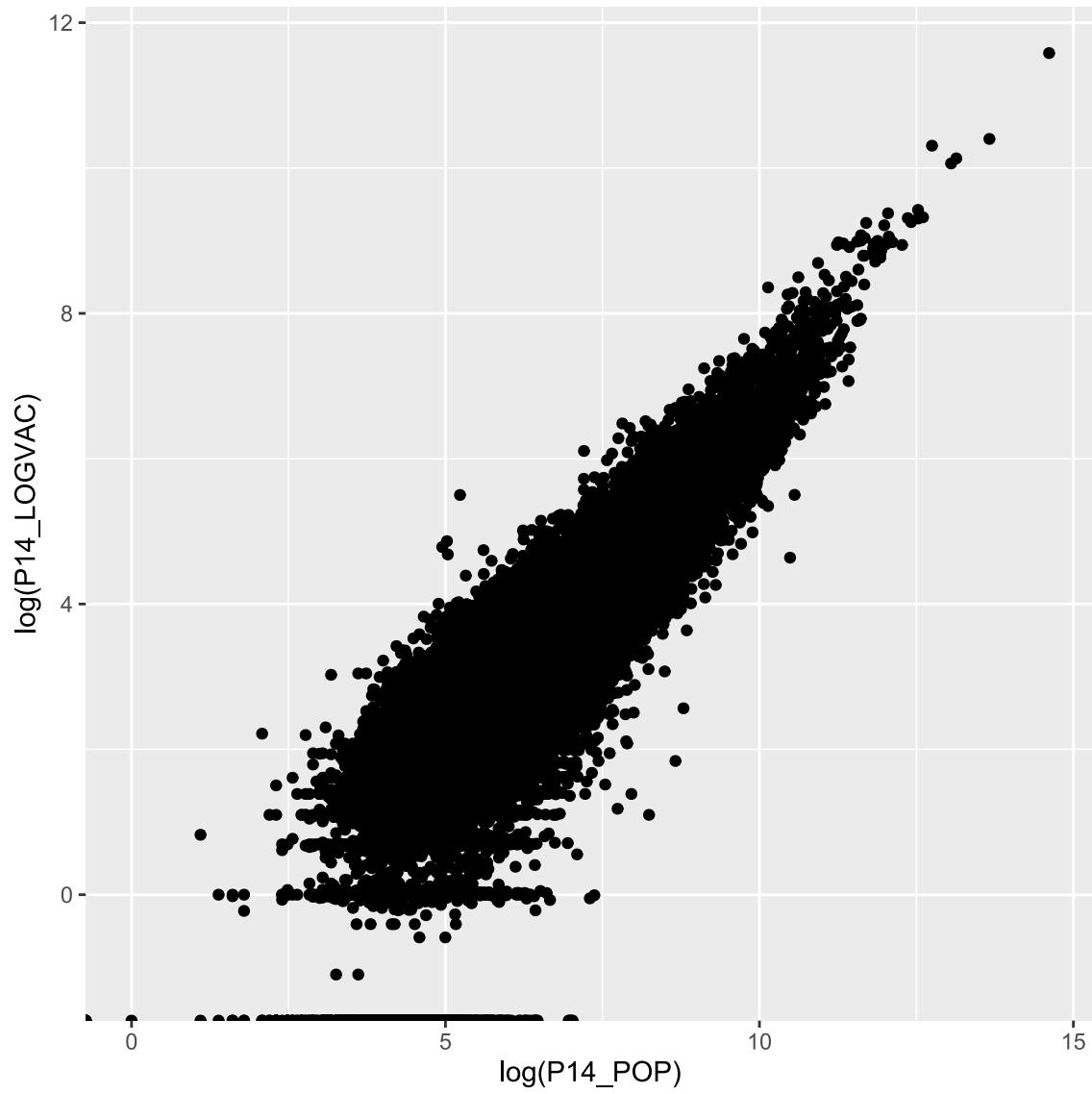


On va essayer d'y voir plus clair avec le paramètre `fill`

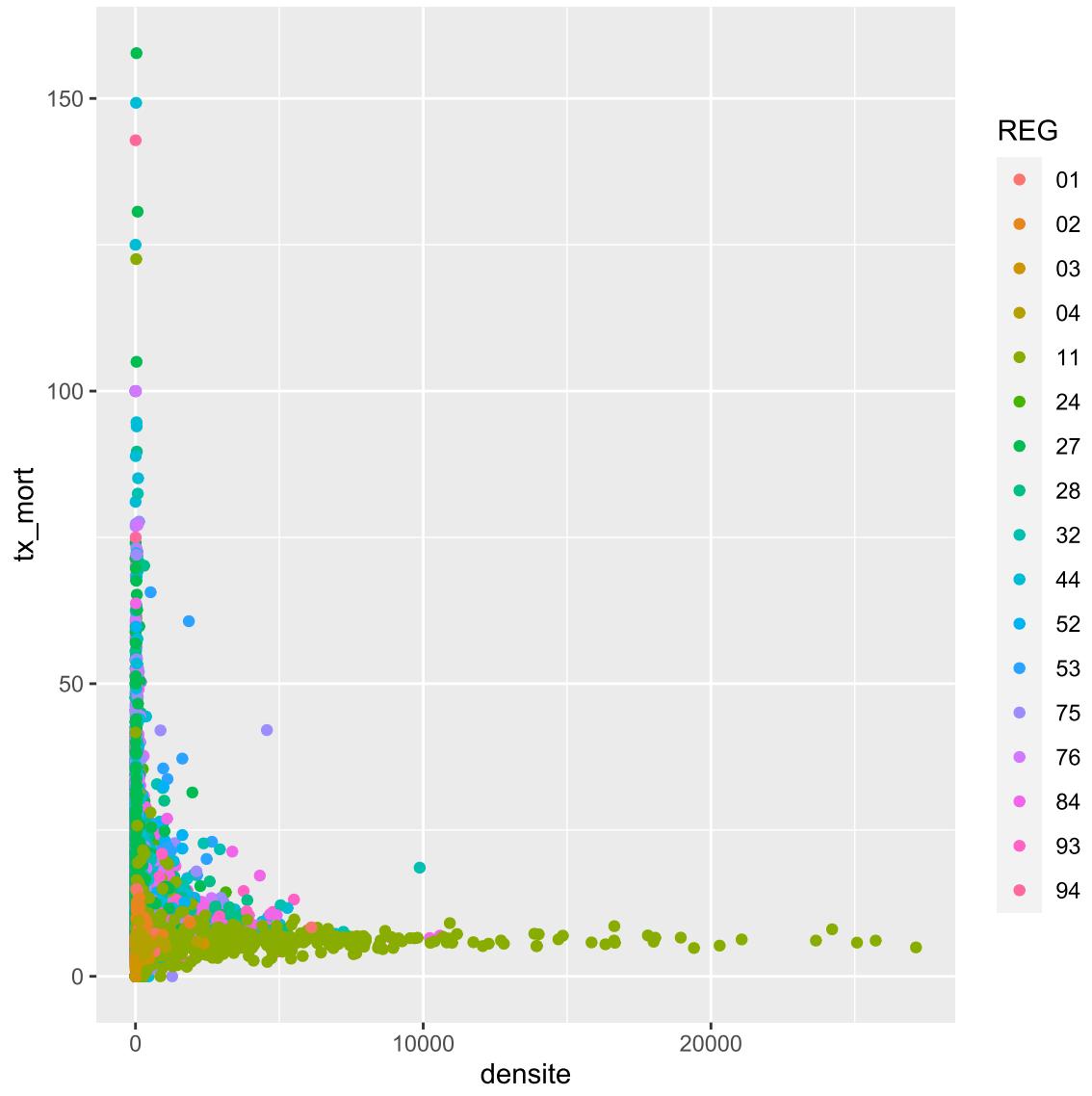
```
ggplot(data = df, aes(x = REG, fill = REG)) +  
  geom_bar()
```



```
ggplot(data = df, aes(x = log(P14_POP), y = log(P14_LOGVAC))) +  
  geom_point()
```



```
ggplot(data = df, aes(x = densite, y = tx_mort, color = REG)) +  
  geom_point()
```



Là encore il faudrait faire une transformation logarithmique, mais tout ça est abordé dans le module 3 !

Chapitre 12 Exercices pour R Studio

12.1 Exercice 1 : Prendre en main RStudio (10 min.)

- Depuis le gestionnaire de fichier de RStudio (panneau bas-droite), vous allez créer un dossier qui va servir de dossier de travail au cours de la formation. Pour cela : utilisez le bouton **New folder** (expl. Formation_R).
- Pour définir ce dossier comme répertoire de travail de R, sélectionnez-le et allez dans le menu **More/Set as working directory**. Regardez ce qui se passe dans la console (panneau bas gauche). Qu'est-ce que cela signifie ?
- Si vous voulez avoir plus de renseignements sur la fonction setwd, vous pouvez utiliser l'aide en ligne de R en écrivant `?setwd` ou `help(setwd)` dans la console. Qu'est ce passe-t-il ?
- Dans l'onglet **Help** de RStudio, vous pouvez également faire une recherche à partir du moteur de recherche. Essayez d'obtenir des informations sur la fonction `dir()`.

12.2 Exercice 2 : Faire ses premiers calculs (15 min.)

- Dans la console, faites des calculs simples : `2 + 3` , `5 * 4` , `5 + (6 * 4)` , `2.5 * 1.3` ,
`3**3`
- Que se passe-t-il lorsque vous écrivez **pi** dans la console ? Qu'est-ce que cela signifie ?
- Calculez maintenant `cos(pi)` et `sin(pi/2)` . Que voyez-vous apparaître au moment où vous écrivez ces fonctions ?
- Faites maintenant la moyenne des nombres : 4, 5, 2, 1. Avez-vous utilisé un calcul ou une fonction de R ?
- Calculez la valeur absolue de l'opération `2 - 6` .
- Calculez maintenant `round(2.125412)` . Que voyez-vous ? Comment pouvez-vous faire pour faire un arrondi de `2.125412` à deux chiffres après la virgule (vous pouvez utiliser l'aide de R).
- Que se passe-t-il lorsque vous manipulez les flèches haut et bas depuis la console ?

12.3 Exercice 3 : Créer ses premières variables (15 min.)

- Dans la console, créer deux variables numériques a et b de valeur 5 et 4 par assignation :
`a<-5` et `b<-4`
- Que se passe-t-il dans le panneau Environnement (en haut à droite) ?
- Faites maintenant un calcul avec ces variables comme `a+b` et `a*b`.
- Vous allez ensuite créer une troisième variable à partir d'un calcul sur les deux premières :
`c<-a+3*b`
- Vous pouvez afficher la liste des variables avec la fonction `ls()`.
- Modifiez maintenant la valeur de `a` en lui donnant la valeur 10. Vérifiez votre ré-assignation dans la panneau Environnement.
- Créez une variable chaîne de caractère : `t<-'texte'` (vous pouvez également utiliser les doubles quotes comme "texte" mais ne les mélangez pas).
- Concaténez (mettez bout à bout) `a` et `t` avec la fonction `paste(a,t)`. Que constatez-vous ?
- Pour plus de renseignements sur la fonction `paste()`, pensez à utiliser l'aide de R avec
`help(paste)`.
- Créez maintenant les variables `A<-15` et `B<-12`. Que voyez-vous dans la panneau Environnement ?
- Vous pouvez supprimer la variables A et B en utilisant la fonction `rm()` (utilisez l'aide de R pour avoir plus de renseignements sur la fonction `rm`).

12.4 Exercice 4 : Créer son premier script (20 min.)

- Depuis le panneau d'édition de script de RStudio (en haut à gauche), vous allez créer un script ré-utilisable permettant de calculer l'indice de masse corporelle d'un individu (IMC) et d'afficher le résultat dans la console sous la forme d'un phrase explicite. L'IMC est calculé par la masse sur la taille au carré d'un individu ($(\text{masse} / \text{taille}^2)$).
- Quelques petits conseils :
 - de façon générale, n'hésitez pas à commenter votre script pour vous aider à le comprendre lorsque vous serez amené à le reprendre,

- pensez à bien assigner vos variables poids et taille pour une meilleure ré-utilisabilité du script,
 - la fonction print() permet d'afficher une chaîne de caractères dans la console,
 - pensez à arrondir le résultat de votre calcul d'IMC.
- Enregistrez ensuite votre script dans votre répertoire de travail. L'extension d'un fichier script pour R est .R.