Chapitre 1 - Prise en main du logiciel R

Thibault LAURENT

06 octobre 2021

Contents

Ce document a été généré directement depuis **RStudio** en utilisant l'outil Markdown. La version .pdf se trouve ici.

1 Présentation de R

1.1 Généralités

R est un système qui est communément appelé langage et logiciel. Il permet, entre autres, de réaliser des analyses statistiques. Plus particulièrement, il comporte des moyens qui rendent possibles la manipulation des données, les calculs et les représentations graphiques. **R** a aussi la possibilité d'exécuter des programmes stockés dans des fichiers textes.

En effet **R** possède :

- un système efficace de manipulation et de stockage des données,
- différents opérateurs pour le calcul sur tableaux (et spécialement les matrices),

- un grand nombre d'outils pour l'analyse des données et les méthodes statistiques,
- des moyens graphiques pour visualiser les analyses,
- un langage de programmation simple et performant comportant : conditions, boucles, moyens d'entrées sorties, possibilité de définir des fonctions récursives.

La conception de **R** a été fortement influencée par deux langages :

- **S** qui est un langage développé par les *AT&T Bell Laboratories* et plus particulièrement par Rick Becker, John Chambers et Allan Wilks. **S** est un langage de haut niveau et est un environnement pour l'analyse des données et les représentations graphiques. **S** est utilisable à travers le logiciel **S-Plus** qui a été un des logiciels de statistiques les plus populaires et s'est imposé comme une référence dans le milieu statistique jusqu'à ce que **R** le détrône depuis quelques années déjà.
- **Scheme** de Sussman est un langage fonctionnel, le principe fondamental de ce langage est la récursivité. L'exécution et la sémantique de **R** sont dérivées de **Scheme**.

Le noyau de **R** est écrit en langage machine interprété qui a une syntaxe similaire au langage **C**, mais qui est réellement un langage de programmation avec des capacités identique au langage **Scheme**. La plupart des fonctions accessibles par l'utilisateur dans **R**, sont écrites en **R** (le système est lui-même écrit en **R**).

Pour les tâches intensives, les langages **C**, **C++** et **Fortran** ont été utilisés et liés pour une meilleure efficacité.

R permet aux utilisateurs d'accroître les possibilités du logiciel en créant de nouvelles fonctions. Les utilisateurs expérimentés peuvent écrire du code en **C** ou **C++** pour manipuler directement des objets **R**. **R** comporte un grand nombre de procédures statistiques. Parmi elles, nous avons : les modèles linéaires, les modèles linéaires généralisés, la régression non-linéaire, les séries chronologiques, les tests paramétriques et non paramétriques classiques, etc. Il y a également un grand nombre de fonctions fournissant un environnement graphique flexible afin de visualiser et créer divers genres de présentations de données.

Les utilisateurs pensent souvent que **R** est un système de statistique. Les concepteurs et développeurs préfèrent dire que c'est un environnement dans lequel des techniques statistiques sont exécutées.

R peut étendre ses fonctions par l'intermédiaire de librairies (appelées aussi bibliothèque, module ou "package" en anglais). Il existe à l'heure actuelle 30 modules fournis avec la distribution de **R** (R version 4.1.1 (2021-08-10)) et d'autres sont disponibles par l'intermédiaire du CRAN. Ils ont été créés pour des buts spécifiques et présentent une large gamme de statistiques modernes (analyse descriptive des données multidimensionnelles, arbres de régression et de classification, graphiques en trois dimensions, etc...).

R est développé pour pouvoir être utilisé avec les systèmes d'exploitation *Unix*, *GNU/Linux*, *Windows* et *MacOS*. **R** possède un site officiel à l'adresse http://www.R-project.org/. C'est un logiciel libre qui est distribué sous les termes de la **GNU Public Licence** (règle du copyleft) et il fait partie intégrante du projet **GNU**.

1.1.1 Les auteurs

R a été initialement créé par Robert Gentleman et Ross Ihaka du département de statistique de l'Université d'Auckland en Nouvelle Zélande. Depuis 1997, il s'est formé une équipe (la "**R** Core Team") qui développe **R**. Elle est constituée de :

- Douglas Bates (USA)
- Henrik Bengtsson (Sweden, USA)
- Roger Bivand (Norway)
- Jennifer Bryan (Canada)
- John Chambers (USA)
- Di Cook (Australia)
- Peter Dalgaard (Denmark)
- Dirk Eddelbuettel (USA)

- John Fox (Canada)
- Robert Gentleman (USA)
- Bettina Grün (Austria)
- Frank Harrell (USA)
- Kurt Hornik (Austria)
- Torsten Hothorn (Switzerland)
- Stefano Iacus (Italy)
- Ross Ihaka (New Zealand)
- Julie Josse (France)
- Tomas Kalibera (Czechia, USA)
- Michael Lawrence (USA)
- Friedrich Leisch (Austria)
- Uwe Ligges (Germany)
- Thomas Lumley (USA, New Zealand)
- Martin Mächler (Switzerland)
- Martin Morgan (USA)
- Paul Murrell (New Zealand)
- Balasubramanian Narasimhan (USA)
- Martyn Plummer (UK)
- Edzer Pebesma (Germany)
- Gabriela de Queiroz (USA)
- Deepayan Sarkar (India)
- Marc Schwartz (USA)
- Duncan Temple Lang (USA)
- Luke Tierney (USA)
- Heather Turner (UK)
- Simon Urbanek (Germany, USA)
- Hadley Wickham (USA)
- Achim Zeileis (Austria)

1.2 Le CRAN

Le "Comprehensive **R** Archive Network" (CRAN) est un ensemble de sites qui fournit ce qui est nécessaire à la distribution de **R**, ses extensions, sa documentation, ses fichiers sources et ses fichiers binaires. Le CRAN est similaire au CPAN pour le langage **Perl** ou au CTAN pour **TeX/LaTeX**. Le site maître du CRAN est situé en Autriche à Vienne, accessible par l'URL http://cran.r-project.org/. Les sites miroirs sont localisés un peu partout à travers le monde avec une sur-représentation en Amérique du Nord et en Europe. Le continent africain est actuellement représenté par l'Algérie, l'Afrique du Sud et le Maroc. Pour candidater afin qu'une institution héberge une image du CRAN, il suffit de suivre les instructions présentées à ce lien. Pour télécharger les applications du projet **R**, il est conseillé d'accéder au site miroir le plus proche géographiquement de l'endroit de votre connexion.

1.3 Le point fort de R

Le logiciel étant dans le domaine public, son point fort est représenté par le développement d'applications, de modules qui sont mis à la disposition de tous les utilisateurs et développeurs. Son réseau international de développement est en perpétuelle évolution. L'intérêt majeur de **R** est qu'il est ouvert à tous. De ce fait, tout le monde peut "apporter sa pierre à l'édifice". Cela laisse envisager de phénoménales extensions au système. Le potentiel de **R** est donc énorme.

1.4 La communauté R

Le nombre d'utilisateurs de **R** a considérablement augmenté depuis sa création. Le logiciel, d'abord utilisé essentiellement dans le milieu académique (universités, instituts de recherche, etc.) s'est rapidement déployé

dans les entreprises. Les utilisateurs du logiciel, en devenant eux-mêmes des contributeurs, ont su s'adapter à l'évolution du milieu industriel et notament l'usage de données massives. Les utilisateurs de **R** se rencontrent chaque année dans un colloque international pouvant réunir jusqu'à 1200 participants. En 2018, le colloque UseR! a eu lieu en Australie, et à Toulouse en 2019. En 2020, la ville d'accueil était Saint-Louis, mais la conférence s'est déroulée entièrement à distance. En 2021, la conférence a eu lieu à Zurich en 2021. Les vidéos des présentations sont en ligne sur la chaîne Youtube du R consortium.

Il existe de nombreux forums dans lesquels les utilisateurs de **R** échangent sur des problèmes, publient des posts sur des expériences personnelles ou annoncent des événements liés à **R**. On citera notamment Stack Overflow ou R-bloggers. Sur ce dernier site, des offres d'emploi apparaissent régulièrement pour lesquelles des compétences en **R** sont exigées. Au niveau local, des réunions d'utilisateurs sont organisées dans certaines villes via les Meetup. En partant du constat que **R** était majoritairement utilisé et developpé par des hommes, une initiative a été lancée afin de promouvoir la diversité des genres parmi les utilisateurs de **R** : R-Ladies. Enfin le site R-exercices permet de s'entraîner sur des exercices corrigés.

R a également bénéficié du soutien de nombreuses entreprises qui ont embauché du personnel pour apporter des contributions au logiciel. Autour de **R**, se sont développées des activités de consulting, développement ou enseignement, souvent commerciales. On citera pour les cours en ligne DataCamp et pour le développement de modules complémentaires R Studio et R consortium.

2 La philosophie des logiciels libres : le projet GNU

Au début du projet, **R** était présenté comme un clone de **S**. En effet, **R** étant développé par le "GNU-project", il est défini par certains comme "GNU-S". Pour cela, il nous semble intéressant de présenter le projet GNU ainsi que la philosophie des logiciels libres. On notera qu'aujourd'hui, **S-Plus** est quasiment absent sur le marché des logiciels statistiques.

2.1 Le projet

Le projet GNU a été lancé en 1984 afin de développer un système d'exploitation complet, semblable à Unix et qui soit un logiciel libre : le système GNU. Des variantes du système d' exploitation GNU, basées sur le noyau "Linux", sont largement utilisées à présent, bien que ces systèmes soient communément appelés par le terme "Linux". Ils le seraient plus exactement par "GNU/Linux". Les logiciels libres disponibles complètent le système.

2.2 La philosophie

L'expression "Logiciel libre" fait référence à la liberté et non pas au prix. Pour comprendre le concept, vous devez penser à la "liberté d'expression", pas à "l'entrée libre". L'expression "Logiciel libre" fait référence à la liberté pour les utilisateurs d'éxécuter, de copier, de distribuer, d'étudier, de modifier et d'améliorer le logiciel.

Plus précisément, elle fait référence à quatre types de liberté pour l'utilisateur du logiciel :

- La liberté d'exécuter le programme, pour tous les usages (liberté 0).
- La liberté d'étudier le fonctionnement du programme, et de l'adapter à vos besoins (liberté 1). Pour ceci l'accès au code source est une condition requise.
- La liberté de redistribuer des copies, donc d'aider votre voisin (liberté 2).
- La liberté d'améliorer le programme et de publier vos améliorations, pour en faire profiter toute la communauté (liberté 3). Pour ceci l'accès au code source est une condition requise.

Un programme est un logiciel libre si les utilisateurs ont toutes ces libertés. Ainsi, il est libre de redistribuer des copies, avec ou sans modification, gratuitement ou non, à tout le monde, partout. Etre libre de faire ceci signifie entre autre que vous n'avez pas à demander ou à payer pour en avoir la permission.

Vous devez aussi avoir la liberté de faire des modifications et de les utiliser à titre personnel dans votre travail ou vos loisirs, sans en mentionner l'existence. Si vous publiez vos modifications, vous n'êtes pas obligé de prévenir quelqu'un de particulier ou de le faire d'une manière particulière.

La liberté d'utiliser un programme est la liberté pour tout type de personne ou d'organisation de l'utiliser pour tout type de système informatique, pour tout type de tâche et sans être obligé de communiquer ultérieurement avec le développeur ou tout autre entité spécifique.

La liberté de redistribuer des copies doit inclure les formes binaires ou exécutables du programme à la fois pour les versions modifiées ou non modifiées du programme.

Pour avoir la liberté d'effectuer des modifications et de publier des versions améliorées, vous devez avoir l'accès au code source du programme. Par conséquent, l'accessibilité du code source est une condition requise pour un logiciel libre. Pour que ces libertés soient réelles, elles doivent être irrévocables tant que vous n'avez rien fait de mal. Si le développeur du logiciel a le droit de révoquer la licence sans que vous n'ayez fait quoi que ce soit pour le justifier, le logiciel n'est pas libre. Cependant, certains types de règles sur la manière de distribuer le logiciel libre sont acceptables tant que ces règles ne rentrent pas en conflit avec les libertés fondamentales.

Par exemple, le "copyleft" est une règle qui établit que lorsque vous redistribuez les programmes, vous ne pouvez ajouter de restrictions pour retirer les libertés fondamentales au public. Cette règle ne rentre pas en conflit avec les libertés fondamentales ; en fait, elle les protège. Ainsi, vous pouvez avoir à payer pour obtenir une copie d'un logiciel du projet GNU ou vous pouvez l'obtenir gratuitement. Mais indifféremment de la manière dont vous vous l'êtes procuré, vous avez toujours la liberté de copier et de modifier un logiciel et même d'en vendre des copies.

"Logiciel libre" ne signifie pas "non commercial". Un logiciel libre doit être disponible pour un usage commercial. Le développement commercial de logiciel libre n'est plus l'exception. Les règles sur la manière d'empaqueter une version modifiée sont acceptables si elles n'entravent pas votre liberté de la publier. Les règles disant "si vous publiez le programme par ce moyen, vous devez le faire par ce moyen aussi" sont acceptables aux mêmes conditions. Dans le projet GNU, il est utilisé le "copyleft" pour protéger ces libertés. Mais des logiciels libres "noncopyleftés" c'est-à-dire qui ne sont pas protégés par la règle existent aussi. Nous croyons qu'il y a de bonnes raisons qui font qu'il est mieux d'utiliser le "copyleft", mais si votre programme est libre "noncopylefté", nous pouvons tout de même l'utiliser.

Quand vous parlez des logiciels libres, il est préférable de ne pas utiliser de termes comme "donner" ou "gratuit", car ils laissent supposer que la finalité des logiciels libres est la gratuité et non la liberté. Pour résumer, un logiciel libre est un logiciel qui est fourni avec l'autorisation pour quiconque de l'utiliser, de le copier, et de le distribuer, soit sous une forme conforme à l'original, soit avec des modifications, ou encore gratuitement ou contre un certain montant. Ceci signifie en particulier que son code source doit être disponible. "S'il n'y a pas de sources, ce n'est pas du logiciel". Pour en savoir plus sur le projet et la philosophie GNU, il est possible de consulter le site officiel pour lequel l'ensemble des informations précédentes ont été tirées.

3 Installation du logiciel

3.1 Systèmes d'exploitations et R

R a été développé pour les systèmes d'exploitations Linux, Windows et macOS (ex (Mac) OS X). Les fichiers disponibles sont précompilés pour certains systèmes (Windows et macOS) alors qu'il faut les compiler pour d'autres (Linux).

3.2 Obtention de R

Les fichiers sources, binaires et de documentation de **R** peuvent être obtenus par le CRAN (Comprehensive **R** Archive Network), le réseau complet des archives de **R**, en chargeant le fichier de **R** correspondant à la version la plus récente R-x.y.z (actuellement R-4.1.1). La marche à suivre est :

1. Lancer un navigateur (Mozilla Firefox, Internet Explorer, Chrome, etc) sur Internet,
2. Se connecter sur le site du CRAN ou un site mirroir,
3. Aller sur la page correspondant au lien "Download **R** for ..." en choisissant votre système d'exploitation et télécharger le fichier d'installation.

3.3 Installation de R

3.3.1 Sous Linux

Choisir l'onglet "Download **R** for Linux". L'installation dépend ensuite de la distribution que vous utilisez : debian, redhat, suse ou ubuntu. Sur Ubuntu, vous devrez dans un premier temps ajouter dans votre fichier /etc/apt/sources.list un des liens suivants (selon la version de votre Ubuntu):

```
deb https://cloud.r-project.org/bin/linux/ubuntu focal-cran40/
deb https://cloud.r-project.org/bin/linux/ubuntu bionic-cran40/
deb https://cloud.r-project.org/bin/linux/ubuntu xenial-cran40/
```

Ajouter également dans le fichier /etc/apt/sources.list la ligne suivante pour permettre de télécharger certains programmes Ubuntu nécessaires aux fonctionnements de certains packages **R** (la liste des sites miroires pour Ubuntu se trouvent ici : https://launchpad.net/ubuntu/+archivemirrors):

```
deb https://<my.favorite.ubuntu.mirror>/ bionic-backports main restricted universe
```

Ensuite, dans un terminal, il faut executer le code suivant :

```
sudo apt-get update
sudo apt-get install r-base
```

Et pour avoir une distribution plus complète (optionnel) :

```
sudo apt-get install r-base-dev
```

Enfin, pour autoriser les mises à jour directement sur le CRAN, il faut ajouter la clé de sécurité suivante :

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys E298A3A825C0D65DFD57CBB651716619E084DAB9
```

Plus d'informations sur ce lien.

3.3.2 Sous Windows

Choisir l'onglet "Download **R** for Windows". Cliquer sur l'onglet "base", puis "Download **R** 4.1.1 for Windows" pour une installation automatique. Dans la majorité des cas, l'installation devrait fonctionner sans problème quelque soit la version de Windows (XP, Vista, 7, 8, 10, 11). Si exceptionnellement des messages d'avertissement ou d'erreurs s'affichaient, il peut y avoir des instructions particulières à faire (voir Onglet "Does **R** run under my version of Windows?").

Si vous êtes en 64 bit, l'installation produira à la fois une version 32b et une version 64b de **R**. Le choix de la version (32b ou 64b) dépend des caractéristiques de votre ordinateur (voir l'onglet "Should I run 32-bit or 64-bit R"). La logique est d'utiliser la version 64b si vous possédez une machine 64b et dans le cas contraire, vous n'avez pas le choix (32b).

La version Windows de **R** a été créée par Robert Gentleman, et est maintenant développée et maintenue par D. J. Murdoch et U. Ligges. Regardez le document "R for Windows FAQ" pour plus de détails.

On pourra également consulter la vidéo "Installer **R** sur Windows" lien ici disponible également sur l'espace de cours.

3.3.3 Sous macOS

Chosir l'onglet "Download **R** for Mac OS X" puis cliquer sur "R.4.1.1.pkg". Après avoir téléchargé le fichier, l'installation se fait automatiquement en double cliquant sur le fichier comme le montre la vidéo suivante. Regarder le document "R for Mac OS X FAQ" pour plus de détails disponible sur le CRAN.

3.4 Modules supplémentaires et manipulation

3.4.1 Les modules supplémentaires

La distribution courante de **R** est actuellement fournie avec des modules supplémentaires qui s'installent en même temps que le logiciel.

- **base** : la plupart des fonctions de base de **R** (comme par exemple calculer une moyenne, un écart-type, etc...),
- **boot** : permet de faire du bootstrap en optimisant le temps de calcul et en utilisant notament le calcul parallèle lorsque la machine le permet.
- **class** : méthodes de classification.
- **cluster** : méthodes de classification.
- **codetools** : comprend des fonctions qui permettent de trouver des erreurs dans les fonctions programmées.
- **compiler** : transforme le code **R** en codebyte ce qui permet d'améliorer les temps de calcul.
- **datasets** : comprend de nombreux jeux de données qui peuvent être utiliser comme exemples.
- **foreign** : fonctions qui permettent d'importer/exporter des fichiers de données provenant d'autres logiciels (**Excel**, **Stata**, **SAS**, etc).
- **graphics** : comprend les outils graphiques statistiques de base.
- **grDevices** : outils pour gérer les palettes de couleurs et pour régler les paramètres des fenêtres graphiques.
- **grid** : permet de découper les fenêtres graphiques avec des grilles.
- **KernSmooth** : estimateur non paramétrique à noyaux de la densité en 2d et 3d.
- **lattice** : outils graphiques complémentaires au module graphics pour l'analyse exploratoire de données.
- **MASS** : fonctions et jeux de données illustrant le livre classique "Modern Applied Statistics with **S**".
- **Matrix** : calcul matriciel pour des matrices creuses. Evite d'utiliser de la mémoire de stockage pour de grosses matrices.
- **methods** : méthodes explicitement définies et outils de programmation.
- **mgcv** : méthodes pour la régression non paramétrique "Generalized Additive Model".
- **nlme** : méthodes linéaire et non linéaire pour des modèles à effets mixtes.
- **nnet** : package pour les réseaux de neurones.
- **parallel** : package pour faire du calcul parallèle lorsque la machine utilisée le permet.
- **rpart** : arbres de régression.
- **spatial** : méthodes de krigeage pour la géostatistique.
- **splines** : splines.
- **stats** : complémentaire au package base, comprend également des fonctions de base plus orientées vers la statistique (par exemple modèle de régression linéaire).

- **stats4** : complémentaire au package stats.
- **survival** : analyse de données de survie.
- **tcltk** : interface pour le logiciel Tcl/Tk.
- **tools** : outils pour le développement des modules et l'administration.
- **translations** : permet de traduire les messages (d'accueil, d'erreur, etc) dans la langue choisie d'installation.
- **utils** : complémentaire au package base, stats et stats4; comprend des fonctions de base (plutôt liées à la notion de classe d'objets).

Même si ces packages sont installés, ils ne sont pas pour autant chargés lorsqu'on ouvre une nouvelle session. Seuls les packages suivants sont automatiquement chargés:

```
getOption("defaultPackages")
```

```
## [1] "datasets"  "utils"     "grDevices" "graphics"  "stats"     "methods"
```

Cela signifie qu'on peut utiliser n'importe quelle fonction incluse dans l'un de ces packages. C'est le cas par exemple de la fonction *sum()* :

```
sum(c(1, 2, 3, 4, 5, 6))
```

```
## [1] 21
```

Pour charger un package qui a été déjà installé, on utilise la fonction *library()*. Par exemple, pour charger le package **KernSmooth** :

```
library("KernSmooth")
```

```
## KernSmooth 2.23 loaded
## Copyright M. P. Wand 1997-2009
```

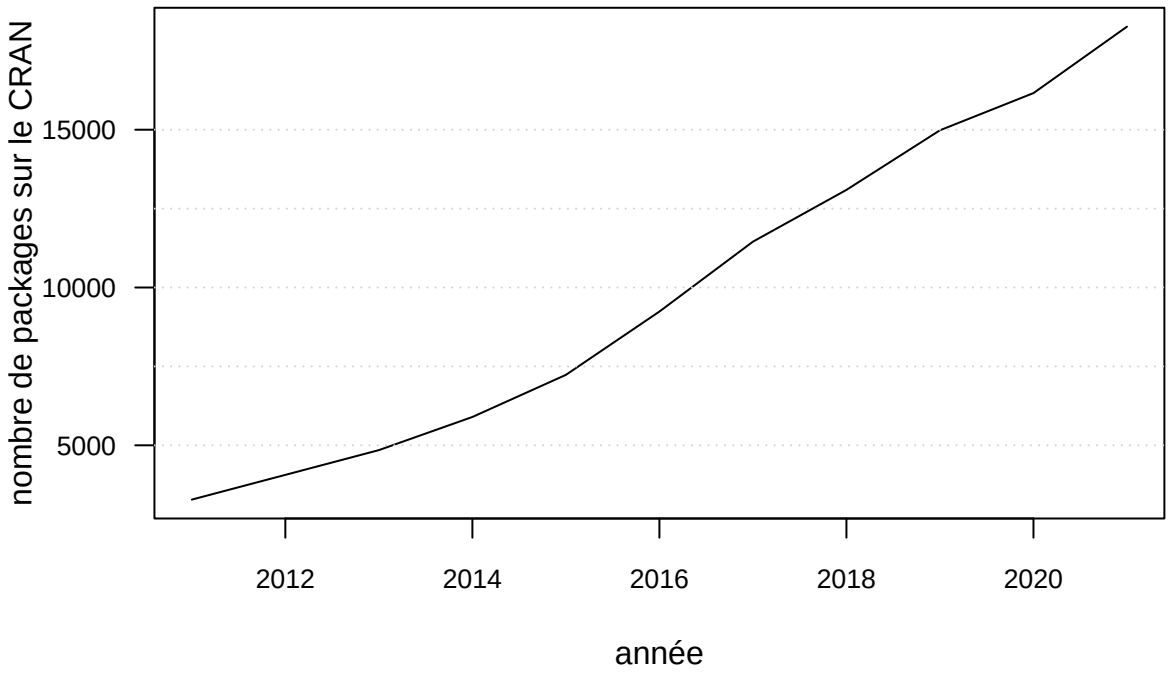
Pour afficher l'ensemble des fonctions contenues dans un package (préalablement installé), il suffit de taper dans la console :

```
help(package="nom_package")
```

Pour connaître la liste des packages déjà installés, on utilise la fonction *installed.packages()*.

On notera qu'un très grand nombre d'autres modules sont disponibles sur Internet par l'intermédiaire du CRAN (18265 le 6 ocotbre 2021, 16160 le 27/08/2020, 14982 le 30/09/2019, 13093 le 28/09/2018, 11453 le 19/09/2017, 9238 le 26/09/2016, 7232 le 28/09/2015, 5898 le 30/09/2014, 4847 le 23/09/2013, 4063 le 01/10/2012 et 3280 le 13/09/2011). Ainsi, le nombre de packages augmente chaque année, ce qui montre le succès de ce logiciel. Le code **R** suivant permet de représenter sur un graphique ces chiffres :

```
years <- c(2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021)
r_packages <- c(3280, 4063, 4847, 5898, 7232, 9238, 11453, 13093, 14982, 16160, 18265)
par(las = 1)
plot(years, r_packages,
     xlab = "année",
     ylab = "nombre de packages sur le CRAN",
     type = "l",
     cex.axis = 0.8)
abline(h = seq(0, 15000, by = 2500), col = "lightgray", lty = "dotted")
```



Pour installer un package disponible sur le CRAN, on utilise la commande *install.packages()*. Par exemple, pour télécharger le package **sf** qui permet de manipuler des données spatiales :

```
install.packages("sf")
```

4 Environnement de travail

Sous Windows, lorsque le logiciel **R** est lancé (double cliquer sur l'icône **R** située sur le bureau), il apparait une nouvelle interface (Rgui) comportant une fenêtre intérieure (la console) et une barre d'outils. Sur Linux et macOS, pour ouvrir une session il suffit de taper **R** dans un terminal :

```
R
```

Comme vous le comprendrez par la suite, **R** est un logiciel de commandes en ligne, c'est à dire que les instructions sont données les unes à la suite des autres. C'est dans la console que l'utilisateur écrit celles-ci et que s'affichent leurs éventuels résultats. Les lignes de code peuvent prendre la forme d'opérations élémentaires (faire par exemple une simple addition) ou plus sophistiqués (faire par exemple des boucles). Certaines de ces commandes peuvent être effectuées manuellement par le truchement de la barre d'outils. Nous ne présenterons pas véritablement cette dernière ici car son maniement est assez intuitif.

4.1 Notions élémentaires

R est un langage orienté objet, c'est à dire que le résultat des instructions peut être stocké dans des objets qui pourront être, à leur tour, utilisés par la suite. Pour pouvoir donner une instruction, l'utilisateur doit "avoir la main", c'est à dire que le logiciel doit être prêt à la recevoir. C'est le cas lorsque le prompt (matérialisé par le signe ">") est visible dans la console. Lorsqu'une instruction comporte des erreurs, le message **Error : syntax error** apparait. Si elle est incomplète, le signe "+" apparaît, indiquant que **R** attend la fin d'une instruction.

Les notions présentées ci-dessous vous permettront de pouvoir utiliser R.

- L'opérateur d'affectation est le signe "**<-**" ou le signe "=". Vous pouvez donc appeler l'un ou l'autre de ces opérateurs. Attention, **R** fait la distinction entre les lettres minuscules et les lettres MAJUSCULES. **a** est différent de **A**.

Exemple : On affecte à l'objet x la valeur 10 par :

```
x <- 10
```

équivalent à :

```
x = 10
```

- Les différentes commandes sont séparées par des retours à la ligne. Dans le cas où l'on souhaite mettre plusieurs instructions sur une même ligne, on peut utiliser le symbole **;** entre les instructions. Par exemple :

```
a <- 1; b <- 2
```

- Pour insérer un commentaire dans un programme, il suffit de précéder, sur chaque ligne, le texte à commenter par le symbole #. Par exemple :

```
a <- 1; b <- 2
d <- a + 5 # On affecte à d la valeur a+5
e <- 5
```

- Pour afficher la valeur d'un objet, il suffit de taper le nom de l'objet dans la console. Cela revient à utiliser la fonction *print()*. Par exemple, pour afficher la valeur de **d** :

```
print(d)
```

```
## [1] 6
```

```
d
```

```
## [1] 6
```

Remarque : Lorsque **R** affiche un résultat, on observe au début de la ligne les caractères **[1]**. Lorsque le résultat à afficher est sur plusieurs lignes (par exemple un vecteur), le chiffre entre les crochets permet de savoir quel est l'indice de l'élément affiché au début de la ligne.

- Pour avoir accès directement à une commande précédemment soumise, il suffit de "remonter l'historique" à l'aide des flèches haut et bas de votre clavier. Ainsi, si vous avez suivi les instructions jusqu'ici, vous obtenez la commande suivante en appuyant deux fois sur la flèche du haut.

```
d <- a + 5
```

- L'instruction suivante permet d'obtenir des informations sur la session (version de **R** utilisée, caractères utilisés, packages chargés).

```
sessionInfo()
```

```
## R version 4.1.1 (2021-08-10)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 20.04.3 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/atlas/libblas.so.3.10.3
## LAPACK: /usr/lib/x86_64-linux-gnu/atlas/liblapack.so.3.10.3
##
## locale:
##  [1] LC_CTYPE=fr_FR.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=fr_FR.UTF-8        LC_COLLATE=fr_FR.UTF-8
##  [5] LC_MONETARY=fr_FR.UTF-8    LC_MESSAGES=fr_FR.UTF-8
##  [7] LC_PAPER=fr_FR.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=fr_FR.UTF-8 LC_IDENTIFICATION=C
```

```
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] KernSmooth_2.23-20
##
## loaded via a namespace (and not attached):
##  [1] compiler_4.1.1    magrittr_2.0.1    tools_4.1.1       htmltools_0.5.1.1
##  [5] yaml_2.2.1        stringi_1.6.2     rmarkdown_2.11    highr_0.9
##  [9] knitr_1.33        stringr_1.4.0     xfun_0.23         digest_0.6.27
## [13] rlang_0.4.11      evaluate_0.14
```

- Les deux instructions suivantes (équivalentes), permettent d'obtenir la liste des objets qui ont été créés dans la session en cours :

```
ls()
```

```
## [1] "a"          "b"          "d"          "e"          "r_packages"
## [6] "x"          "years"
```

```
objects()
```

```
## [1] "a"          "b"          "d"          "e"          "r_packages"
## [6] "x"          "years"
```

- L'instruction suivante supprime l'objet dont le(s) nom(s) est donné en argument. Par exemple, pour supprimer les objets **a** et **b** :

```
rm(a, b)
ls()
```

```
## [1] "d"          "e"          "r_packages" "x"          "years"
```

Pour effacer tous les objets créés dans la session en cours, on fait :

```
rm(list = ls())
```

Remarque : ci-dessus, nous n'avons pas utilisé la lettre **c** pour définir un nouvel objet, car la lettre **c** est une fonction prédéfinie de **R** que nous verrons très rapidement. De manière générale, il est conseillé de ne pas nommer un nouvel objet avec un nom déjà utilisé par **R** (par exemple une fonction, un mot clé tel que **for** ou **while**).

4.2 Gestion du répertoire de travail

Les objets créés au cours d'une session de travail ne sont pas sauvegardés automatiquement. Si la possibilité de le faire vous est donnée à la fin d'une session (lorsque vous fermez **R**, une fenêtre vous demande si vous souhaitez sauvegarder votre espace de travail), il est souvent préférable d'effectuer des sauvegardes régulières. Pour cela, vous disposez de la fonction *save.image()* qui stocke dans un fichier **.RData** l'ensemble des objets existants.

Si vous ne souhaitez sauvegarder qu'une partie des objets que vous avez créés (par exemple un objet dont le temps de calcul pour se créer est long), dans ce cas, vous pouvez utilisez la fonction *save()* pour sauvegarder les objets désirés. Vous utiliserez la fonction *load()* pour les importer lors d'une nouvelle session. Par exemple, pour sauvegarder les objets **a** et **d** dans le fichier "res.RData", on fait :

```
a <- 1
d <- a + 5
save(a, d, file = "res.RData")
```

Si on ne précise par un nom de chemin en entier (comme c'est le cas dans l'exemple ci-dessus), le fichier "res.RData" sera enregistré dans le répertoire de travail "courant". Pour connaître le chemin de ce répertoire de travail, on utilise l'instruction suivante, très utile :

```
getwd()
```

```
## [1] "/media/thibault/My Passport/course/R_base/chapitre 1"
```

Afin que tous les fichiers ne soient pas stockés dans ce même répertoire de travail, il est fortement conseillé de créer un emplacement spécifique pour chaque nouvelle session de travail. Vous pourrez aussi organiser votre répertoire en sous-dossiers dans lesquels vous stockerez bases de données, résultats, figures, etc... Pour indiquer à **R** l'emplacement de ce répertoire, il faut utiliser l'instruction *setwd("chemin_acces")*. Pour la suite, nous nous placerons dans le répertoire "C:/logicielsStat/R/etheme1", ce qui sous-entend que ce répertoire de travail existe bien et que vous l'avez donc créé au préalable.

```
setwd("C:/logiciel_stat/R/etheme1")
# équivalent à
setwd("C:\\logiciel_stat\\R\\etheme1")
```

Remarque : pour spécifier les chemins d'accès et séparer les noms de répertoire entre eux, utiliser le symbole / plutôt que \. En effet, ce dernier symbole a une signification lorsqu'il est dans une chaîne de caractère. Par exemple, **\n** dans une chaîne de caractères indique un saut de ligne. C'est pourquoi il faut deux barres obliques pour obtenir le symbole désiré. Pour vous en convaincre, taper l'instruction suivante :

```
cat("Saut de 1 ligne : \nSaut de 2 lignes \n\nAfficher l'antislash : \\ \n")
```

```
## Saut de 1 ligne :
## Saut de 2 lignes
##
## Afficher l'antislash : \
```

Les éventuelles sorties-écran apparaissent par défaut dans la console. Mais il est parfois souhaitable de les rediriger vers un fichier. Pour cela, vous disposez de la fonction *sink()* qui permet de créer un fichier dans lequel seront stockées tous les résultats. Cela peut être le cas si l'on souhaite par exemple récupérer un tableau de résultats sous forme d'un fichier ASCII. Reprenons l'ensemble des instructions de ce paragraphe.

```
a <- 1
b <- 2
d <- a + 5
e <- 5
```

Nous allons maintenant imprimer les valeurs de ces 5 objets dans le fichier "sortie.txt", qui se situera dans le répertoire de travail, à savoir "C:/logiciel_stat/R/etheme1/" dans la mesure où vous avez bien exécuté l'instruction précédente (*setwd("C:/logiciel_stat/R/etheme1")*). La fonction *cat()* concatène et affiche les valeurs des objets qu'on lui met comme arguments d'entrée et c'est le résultat de cette fonction qui sera donc sauvegardée dans le fichier "sortie.txt" :

```
sink("sortie.txt")
cat(a, "\t", b, "\t", d, "\t", e)
sink()
# A partir de là, les résultats seront de nouveau affichés dans
# la console et plus sauvegardés dans le fichier "sortie.txt"
```

En ouvrant le fichier "sortie.txt" avec votre éditeur de texte, on constate que les résultats qui auraient dû être affichés à la console ont été imprimés dans ce fichier. Cette fonctionnalité aura un intérêt limité dans le cours et nous ne l'utiliserons vraisemblablement pas dans la suite.

4.3 L'aide en ligne

Il est possible d'obtenir de l'aide par défault au format html. **R** propose une aide en ligne en tapant la commande

```
help.start()
```

On recommande en particulier le manuel "An Introduction to R" qui est très complet. En cliquant sur le lien "Packages", vous trouverez des renseignements sur l'ensemble des librairies installées (celles par défaut et celles que vous aurez installés vous-mêmes).

L'aide pour une commande particulière est possible en tapant **?nom_commande** ou **help(nom_commande)**. Par exemple :

```
?mean
help(mean)
```

Une fenêtre **html** apparaît dans le navigateur internet par défault, contenant des renseignements sur la fonction *mean()*.

De plus, il est possible de réaliser une recherche de fonctions à l'aide de mots-clés grâce à la fonction *apropos()*. Pour obtenir toutes les fonctions qui contiennent le mot "mean" sont affichées, on fait :

```
apropos("mean")
```

```
##  [1] ".colMeans"     ".rowMeans"     "colMeans"      "kmeans"
##  [5] "mean"          "mean.Date"     "mean.default"  "mean.difftime"
##  [9] "mean.POSIXct"  "mean.POSIXlt"  "rowMeans"      "weighted.mean"
```

Enfin la fonction *help.search("mot_cle")* permet de renvoyer parmi toutes les fonctions appartenant aux librairies installées lesquelles contiennent le "mot_clé" dans leur aide.

```
help.search("mean")
```

4.4 Editeurs de textes

Les programmes que l'on soumet au logiciel sont parfois très longs. De plus, il faut souvent avoir recours à des essais (notamment pour le réglage des paramètres) avant d'obtenir ce que l'on souhaite. Il est donc très utile d'écrire les programmes dans un document à part et de "naviguer" ensuite entre les deux applications par des copier-coller ou comme nous le verrons plus tard, en faisant du clique-bouton.

La méthode la plus courante est d'utiliser un éditeur de texte/codes. La procédure consiste ensuite à sélectionner la partie du programme que l'on veut soummettre, puis l'envoyer vers la console. Sous Windows, **R** contient son propre éditeur : pour cela, aller dans "Fichier", "nouveau script" et une fenêtre apparaît dans laquelle vous pouvez écrire vos instructions. Pour exécuter ensuite le code dans la console, il suffit de sélectionner les parties du code qui nous intéressent, de faire un clique-droit à la souris et appuyer sur "exécuter le code", ce qui revient à utiliser le raccourci Ctrl + **R**.

En général, un fichier qui contient du code **R** est enregistré en utilisant l'extension **.R**. Aussi, dans le cours, lorsque vous disposerez de fichiers avec cette extension, ceci signifiera qu'il s'agit de fichiers contenant du code **R**. Il est possible d'exécuter l'ensemble du contenu de fichiers de codes **R** avec la commande suivante :

```
source("nom_du_fichier.R")
```

L'éditeur de code fourni par défaut est toutefois très limité pour un éditeur de codes. C'est pour cela qu'il est préférable, d'utiliser des éditeurs de codes spécifiques, qui fournissent une batterie d'applications complémentaires et facilement accessibles.

Pour ce cours, parmi les éditeurs présentés, nous recommandons d'utiliser **RStudio**.

4.4.1 L'incontournable RStudio

RStudio a l'avantage d'être multiplateforme (Windows, Linux, macOS). Il est en fait une interface agréable qui réunit dans une même fenêtre : la console **R**, un éditeur de texte, l'historique des commandes et des objets créés et l'interface graphique. **RStudio** a l'avantage de faire de **R** un logiciel plus convivial un peu comme **STATA** ou **SAS**. Il est téléchargeable sur le site R Studio et nous vous proposons de voir la vidéo suivante qui montre les avantages de **RStudio** (Vidéo) sur l'éditeur de texte par défaut de **R**.

Sachant que cette vidéo a été réalisée il y a quelques années, vous verrez que **R** Studio a depuis intégré de nouvelles cordes à son arc. Notamment, il a intégré des outils récents comme *Sweave*, *R Markdown*, *Notebook* qui sont des outils permettant de faire du reporting ou *Shiny* qui permet de faire des applications Web de manière interractive.

Les employés de **RStudio** sont à l'origine de nombreuses nouvelles applications. Pour financer ces activités de Recherche et Développement, vous constaterez que **RStudio** propose certains de ses services payants.

Compléments :

Pour découvrir l'outil **R** Markdown, vous pouvez consulter le document suivant écrit par Sophie Lamarre, une collègue statisticienne (lien de la présentation) ou bien ce chapitre d'un cours de Julien Barnier (lien du chapitre). Nous aurons l'occasion d'utiliser cet outil dans le cours, notamment pour rendre les devoirs et projets.

4.4.2 Alternatives à RStudio

Les éditeurs suivants peuvent être des alternatives à **RStudio** bien que nous conseillons fortement d'utiliser **RStudio** en priorité. Il n'est donc pas nécessairement utiles d'installer les éditeurs suivants à moins que pour une raison particulière, vous souhaitiez le faire.

Tinn-R Il est l'éditeur de texte recommandé après **RStudio** pour les utilisateurs de Windows. Il est téléchargeable sur le site suivant : sourceforge. L'installation de cet éditeur est détaillée dans la vidéo "Tinn-R: éditeur de texte pour R", disponible dans l'espace de cours (Tinn-R a connu quelques changements depuis ces dernières années et il se peut que certains élements de la vidéo soient obsolètes. Cependant, la procédure devrait être très ressemblante).

Emacs Speaks Statistics Il existe un mode spécifique à Emacs appelé ESS ("Emacs Speaks Statistics") qui fournit une "bonne" interface entre les programmes statistiques et les méthodes statistiques. Ce module de Emacs est réalisé pour fournir une assistance à la programmation statistique interactive et à l'analyse des données. Toutefois, son utilisation est plutôt recommandé à des usages qui sont déjà habitués à travailler avec des éditeurs de texte dédiés à la programmation avancée. Le site officiel de ESS est abrité par l'Université de Washington http://ess.r-project.org/. Vous y trouverez le logiciel et la documentation. L'utilisation d'ESS est recommandée pour les utilisateurs de Linux car au niveau de la gestion des commandes, certains pré-requis de Linux peuvent s'avérer très utile... Toutefois, il s'utilise très bien sur Windows aussi.

Atom Pour les utilisateurs de macOS, il peut s'avérer intéressant.

D'autres éditeurs existent (voir une liste non exhaustive sur ce lien)

4.5 R Commander

Comme vous avez pu le constater, **R** est un logiciel de programmation où il faut taper toutes les commandes dans la console pour obtenir les résultats et graphiques que l'on souhaite. Grâce à la librairie **Rcmdr**, il est possible de créer une interface qui permet de faire de **R** un logiciel "clique-bouton" pour l'analyse de données basique (un peu comme SPSS ou le module SAS Enterprise Guide). Vous trouverez une description détaillée de ce package dans l'article de John Fox (McMaster University), The R Commander: A Basic-Statistics GUI for R. Depuis peu, l'interface facilite également l'accès à **R** Markdown pour faciliter la création de rapports qui contiennent à la fois du texte et des résultats/graphiques statistiques.

Pour installer le package, télécharger-le depuis le CRAN comme expliqué dans la section 3.6.1. Ce package dépend de nombreux autres packages et lorsque vous allez le charger, il vous sera nécessaire d'installer d'autres packages (**zoo**, **rgl**, **colorspace**, etc.), cette opération pouvant se faire automatiquement en suivant les instructions qui vous seront données. Si des erreurs se produisent, cela peut s'expliquer par la dépendance de certains packages pré-installés (comme **foreign** ou **rpart**). Dans ce cas, il faudra faire l'installation manuelle de ces deux packages.

Pour lancer le module :

```
require("Rcmdr")
```

Dans ce cours, nous n'utiliserons pas **R** commander pour la raison que l'objectif de ce cours est que vous appreniez les fonctionnalités du logiciel. Ce module peut s'avérer être intéressant pour des personnes qui ne souhaitent pas apprendre le langage **R**, mais uniquement utiliser les outils statistiques (calculer une moyenne, faire un histogramme, etc.) et dans ce cas, on peut comprendre que la connaissance du langage **R** n'est pas indispensable.

Remarque : dans des versions précédentes de **RStudio**, la package **Rcmdr** n'était pas nécessairement compatible. Depuis, **RStudio** a pris en compte ces défauts et le package Rcmdr peut être utiliser conjointement avec **RStudio**.

5 La table diamants

Les chapitres qui suivent vont présenter les fonctionnalités de **R**. Au cours de chacun d'entre eux, nous utiliserons le jeu de données **diamants**, qui étudie les caractéristiques de diamants. Chaque ligne du tableau représente donc un diamant et les variable observées sont :

- **price**, prix en dollar ($326–$18,823),
- **carat**, poids du diamant (0.2–5.01),
- **cut**, qualité de la forme (*Fair*, *Good*, *Very Good*, *Premium*, *Ideal*),
- **color**, couleur du diamant, de *J* (mauvaise) à *D* (bonne),
- **clarity**, une mesure de pureté (*I1* (mauvaise), *SI1*, *SI2*, *VS1*, *VS2*, *VVS1*, *VVS2*, *IF* (la meilleure)),
- **x**, taille en mm (0–10.74),
- **y**, diamètre en mm (0–58.9),
- **z**, profondeur en mm (0–31.8).

Avant chaque début de chapitre, vous devrez donc créer un nouveau répertoire de travail et exécuter l'instruction suivante pour charger les données :

```
load(file("http://www.thibault.laurent.free.fr/cours/Ressource/diamants.RData"))
```