

Installation et prise en main de Python

Aymeric Jan (aymeric.jan@univ-paris1.fr)

January 30, 2024

1 Introduction

Python est un des langages de programmation les plus populaires actuellement. Très polyvalent, il permet aussi bien de faire de simples scripts scientifiques que de créer des logiciels ou des application web. Depuis plusieurs années, il est le langage plébiscité pour l'intelligence artificielle (Data Science, Machine Learning, etc...).

2 Installation

2.1 Installation de Python

L'installation de Python est possible sur tous les OS (Windows, MacOS, Linux), pour ce faire, il suffit de télécharger une version compatible depuis le site officiel <https://www.python.org/downloads/>. Bien que les dernières versions possèdent d'intéressantes fonctionnalités, je recommande d'installer la version 3.8.10, disponible ici <https://www.python.org/downloads/release/python-3810/>. Pour ce faire, il suffit de télécharger la version correspondante depuis le tableau en bas de page, en général, il s'agit de la version Windows installer (64-bit). Il est conseillé de redémarrer l'ordinateur pour finaliser l'installation.

2.2 Installation des librairies

De nombreuses librairies sont disponibles en Python, la meilleure manière de les installer est de passer par la commande pip du terminal.

Dans notre cas, nous installerons les librairies suivantes

1. Numpy: Librairie de calcul scientifique.
2. Matplotlib: Librairie de visualisation.
3. Jupyter notebook: Notebook interactifs.

Pour les installer, il suffit d'ouvrir le terminale et de taper les commandes suivantes

```
pip install numpy
pip install matplotlib
pip install notebook
```

Files					
Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		83d71c304acab6c678e86e239b42fa7e	24720640	SIG
XZ compressed source tarball	Source release		d9eee4b20155553830a2025e4dcaa7b3	18433456	SIG
macOS 64-bit Intel installer	macOS	for macOS 10.9 and later	690ddb1be403a7efb202e93f3a994a49	29896827	SIG
macOS 64-bit universal2 installer	macOS	experimental, for macOS 11 Big Sur and later; recommended on Apple Silicon	ae8a1ae082074b260381c058d0336d05	37300939	SIG
Windows embeddable package (32-bit)	Windows		659adf421e90fba0f56a9631f79e70fb	7348969	SIG
Windows embeddable package (64-bit)	Windows		3acb1d7d9bde5a79f840167b166bb633	8211403	SIG
Windows help file	Windows		a06af1ff933a13f6901a75e59247cf95	8597086	SIG
Windows installer (32-bit)	Windows		b355cfc84b681ace8908ae50908e8761	27204536	SIG
Windows installer (64-bit)	Windows	Recommended	62cf1a12a5276b0259e8761d4cf4fe42	28296784	SIG

Figure 1: Installer Python.

3 Jupyter Notebook

Le plus pratique pour les TP/TD est d'utiliser ce qu'on appelle des notebooks. Similaire aux notebooks dans R, ils permettent d'exécuter de manière interactive seulement certaines parties du code. Ces parties de codes sont dans ce qu'on appelle des cellules. Vous trouverez sur le lien ci-après une documentation plus fournie sur les notebooks: <https://docs.jupyter.org/en/latest/>.






Pour démarrer un notebook, il faut ouvrir un terminal de commande et taper la commande suivante

```
jupyter notebook
```

Cela va ensuite ouvrir une page dans le navigateur web pour choisir le notebook à ouvrir.

3.1 Commandes de base sur les notebook

Voici quelques raccourcis très utiles pour les notebooks:

1. Créer une nouvelle cellule: 
2. Exécuter une cellule:  + 
3. Supprimer une cellule:  + 

4 Base en Python

4.1 Affectation et variables

En Python, il n'est pas nécessaire de déclarer le type de la variable, il suffit juste de donner le nom de la variable à gauche du signe `=` et sa valeur à droite. Des exemples sont donnés ci-après.

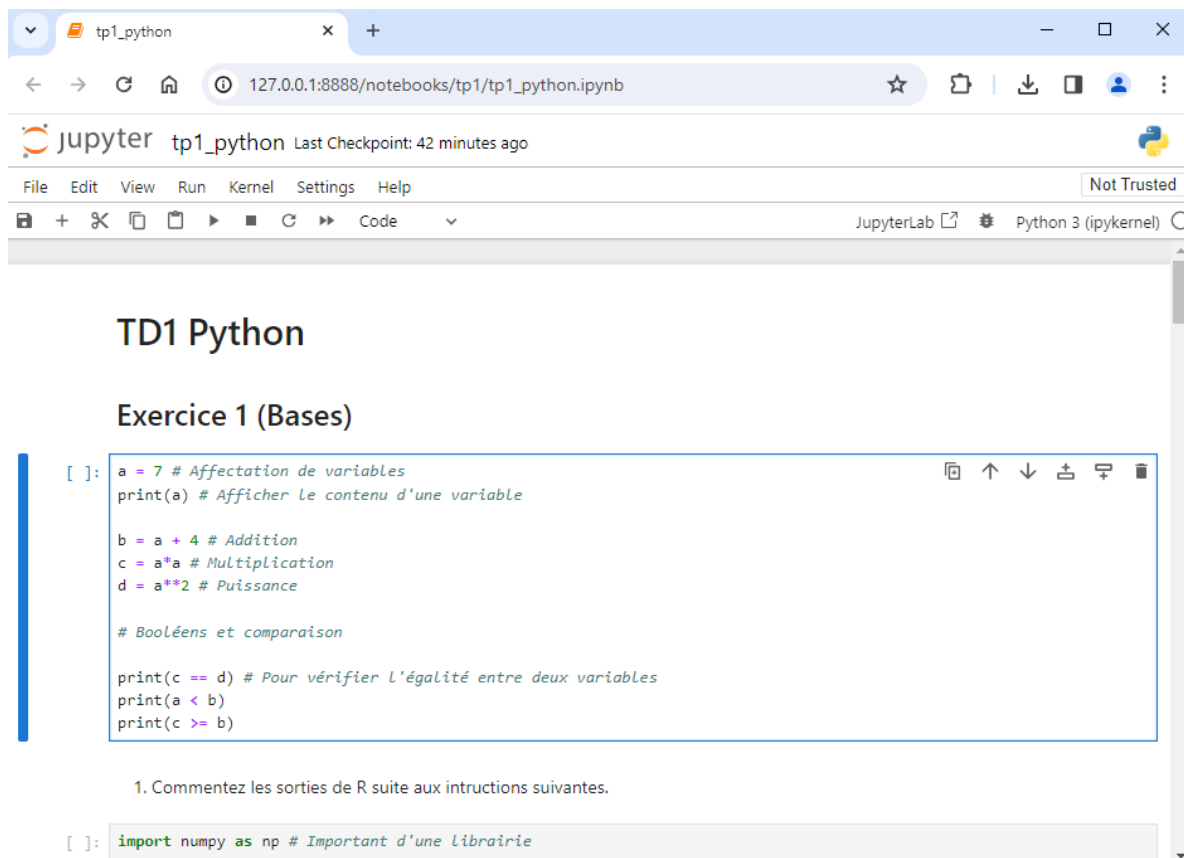


Figure 2: Exemple d'un Jupyter Notebook

```
var_int = 1 ## int
var_float = 1.4242 ## floats
my_string = "python" ## string
my_bool = True
print(var_float)
```

Les opérations de base comme l'addition, la soustraction, la multiplication etc... sont disponibles

```
result_addition = 1 + 2
result_mult = 1*2
result_puissance = 2**3
```

4.2 Boucles for, while et if statements

Il existe deux types de boucles en Python, les `for` et les `while`.

4.2.1 Boucle for

Une boucle `for` permet de répéter une action un certain nombre de fois défini à l'avance.

```
for ind in range(0, 10):
```

```
print(ind)
```

4.2.2 Boucle while

Une boucle `while` permet de répéter une séquence de code plusieurs fois, tant que la condition est respectée.

```
ind = 0
my_condition = True
while my_condition:
    ind += 1
    my_condition = ind < 3
```

4.2.3 If et else

Il est possible d'ajouter des conditions avec `if` et `else` comme suit:

```
if 3+1 > 2:
    print("It's true")
else:
    print("You are wrong")
```

4.3 Structure de données

Il existe plusieurs types de structures de données en Python, en voici les principales.

4.3.1 Les listes

Une liste est une structure de données contenant une série de valeur ordonnées. Elle se définit à l'aide de crochets `[]`, ses éléments sont séparés par des virgules, et il est possible d'accéder à un élément de la liste à partir de son index en utilisant des crochets également. Il faut noter qu'en Python, les indices commencent à partir de 0.

```
my_list = [2, 3, 5]
print(my_list[2]) # Cela renvoie 5
```

On obtient la taille d'une liste en utilisant la fonction `len`

4.3.2 Les dictionnaires

Un dictionnaire est une structure de donnée où les valeurs sont identifiées par une clé. On utilise les accolades `{}` pour créer un dictionnaire. Chaque élément d'un dictionnaire est donc un couple (clé, valeur).

```
my_dict = {"pi":3.141593, "e":2.718282}
print(my_dict["pi"]) # Cela renvoie 3.141593
```

On peut obtenir la liste des clés d'un dictionnaire grâce à la méthode `keys`. Une méthode s'utilise comme suit: `my_dict.keys()`.

4.4 Les fonctions

En Python, une fonction se définit de la manière suivante

```
def my_func(arg_1, arg_2):  
    var_interne = arg_1 + arg_2  
    return var_interne  
  
my_result = my_func(arg_1, arg_2)
```

Le keyword `def` permet la définition d'une fonction, le nom de celle-ci est suivie de parenthèses contenant les arguments de la fonction séparés par des virgules. Le keyword `return` permet de renvoyer une ou plusieurs variables.

4.5 Les librairies

4.5.1 Numpy

NumPy est la librairie la plus utilisée en Python, elle est fondamentale pour le calcul scientifique. Elle permet notamment de définir des arrays, regroupant les vecteurs et les matrices. On y retrouve également toutes les fonctions liées à l'algèbre linéaire et à l'aléatoire. Plus d'informations sur la documentation (<https://numpy.org/doc/stable/>).

4.5.2 Matplotlib

Matplotlib est une librairie de visualisation. Elle permet la création de graphiques, parfois interactifs. C'est celle-ci que nous utiliserons pour afficher des nombreux résultats. Plus d'informations sur la documentation (<https://matplotlib.org/stable/users/index>).

Installation et prise en main de R

Fabien Navarro

19 janvier 2023

Table des matières

1	Introduction	1
2	Installation	1
3	Environnement de travail R natif	2
3.1	Lancement d’une session R	2
3.2	Premières instructions	2
3.3	Naviguer dans l’environnement	3
3.4	Déclaration	4
3.5	Sauvegarde, chargement et suppression	4
3.6	Installation de packages	5
3.7	Documentation	6
4	L’environnement Rstudio	7

1 Introduction

R est un langage de programmation/logiciel dédié initialement aux statistiques et à la science des données. Il est soutenu par la **R Core Team** et la **R Foundation for Statistical Computing**. Créé par les statisticiens Ross Ihaka et Robert Gentleman, R est couramment utilisé par les *data scientist* et les statisticiens pour l’analyse des données et le développement de logiciels statistiques. De nombreux utilisateurs ont créé des packages pour en augmenter les fonctionnalités. Même si son usage premier reste la statistique, ses nombreuses fonctionnalités permettent son usage au sein d’autres disciplines, comme l’analyse numérique. En effet, R est très populaire, en janvier 2022, il est classé 12e d’après l’indice TIOBE (une certaine mesure de popularité des langages de programmation).

2 Installation

R est un environnement logiciel libre et *open-source*. Il est écrit principalement en C et Fortran. Des exécutables précompilés sont fournis pour divers systèmes d’exploitation (e.g., Windows et Mac OS) alors que son installation sur d’autres systèmes (e.g., Linux) nécessite une compilation des sources. Il possède une interface de ligne de commande et de nombreuses interfaces graphiques tierces sont

également disponibles, telles que **RStudio**, un environnement de développement intégré gratuit, libre et multiplateforme. L'usage de ce dernier est recommandé dans le cadre de ce cours.

Les fichiers sources associés à une distribution de R donnée peuvent être téléchargées via le CRAN (Comprehensive R Archive Network). Il est recommandé de récupérer les sources de la dernière version (i.e. R 4.1.2). La marche à suivre est la suivante :

- ouvrir le navigateur web de votre choix ;
- accéder à l'URL du CRAN (<https://cran.r-project.org/>) ;
- sous Windows et Mac OS, télécharger l'exécutable correspondant à votre système d'exploitation (i.e., via Download R for Windows/install R for the first time ou Download R for macOS). Sous Linux, les instructions, lignes de commandes à exécuter et dépendances à installer sont également fournies, pour quelques distributions standards (Download R for Linux).

A priori, l'installation après exécution des fichiers téléchargés devrait bien se passer. Si toutefois, vous rencontreriez des problèmes (e.g., messages d'avertissement ou d'erreurs), il est possible que certaines instructions particulières soient nécessaires (dans ce cas voir la FAQ ici <https://cran.r-project.org/bin/windows/base/rw-FAQ.html>).

3 Environnement de travail R natif

3.1 Lancement d'une session R

Pour utiliser R sous Windows, il suffit de cliquer sur l'icône R sur le bureau. Une interface graphique utilisateur (GUI) apparaît, la console R. Sous Linux et MacOS, R peut être lancé soit via l'icône de l'application soit directement depuis un terminal (en tapant R), donnant ainsi accès à la console R directement depuis le terminal.

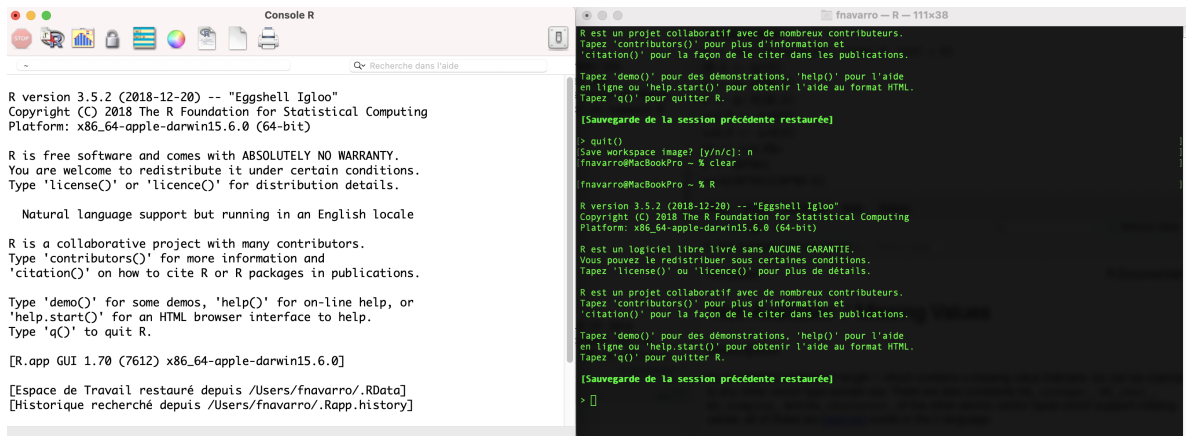


FIGURE 1 – R depuis l'application ou depuis un terminal.

3.2 Premières instructions

Une fois lancée, la console de R peut recevoir une instruction de l'utilisateur. L'invite de commande est représentée par le symbole **>**. Chaque instruction doit être validée par **Entrée** pour être exécutée. Suite à une instruction, différents types de réponses de l'interpréteur de commandes sont possibles :

Le résultat

```
1+1
#> [1] 2
```

Un message d'avertissement (ou *Warning*)

```
c(1,2)+c(1,2,3)
#> Warning in c(1, 2) + c(1, 2, 3): longer object length is not a multiple of
#> shorter object length
#> [1] 2 4 4
```

Une erreur

```
1+"un"
#> Error in 1 + "un": non-numeric argument to binary operator
```

Une demande de complétion d'instruction, représentée par le symbole `+` (généralement issue de l'oubli d'une parenthèse). Dans ce cas, il faut soit compléter l'instruction soit sortir (en tapant sur la touche `esc` dans la console).

Et parfois...  lors d'un crash d'une session (par exemple dû à la saturation de la mémoire vive).

Les majuscules et minuscules sont différenciées. Les instructions commençant par le symbole `#` ne sont pas interprétées, ce sont des commentaires.

3.3 Naviguer dans l'environnement

Les données, les variables déclarées et les instructions utilisées pourront être enregistrées à l'endroit où R a été installé, ou à la même position que celle du chemin d'accès du répertoire (ou du fichier) depuis lequel R a été ouvert (ou encore à l'endroit de votre choix en spécifiant le chemin désiré). La commande `getwd()` permet d'obtenir le chemin associé au répertoire courant, `setwd()` permet de le modifier.

```
getwd()
#> [1] "/Users/fnavarro/Documents/P1/P12021/S2/L2MN/lab/intro"
```

```
setwd("somewhere/on/my/laptop/")
```

Attention : les chemins sont définis à l'aide du symbole `/` et non pas `\`. En effet, le symbole *backslash* a une signification lorsqu'il est présent dans une chaîne de caractère. Par exemple, `\n` indique un saut de ligne. C'est pourquoi un double backslash est requis pour obtenir le symbole désiré. Par exemple,

```
cat("retour chariot \nAfficher le symbole : \\")
#> retour chariot
#> Afficher le symbole : \
```

Les commandes `dir` et `list.files/list.dirs` permettent d'afficher le contenu d'un répertoire (en produisant un vecteur de caractères des noms de fichiers ou de répertoires).

```
dir()
#> [1] "images" "introR_MN_L2MIASHS.pdf" "introR_MN_L2MIASHS.rmd"
#> [4] "introR_MN_L2MIASHS.tex" "references.bib" "template-latex.tex"
```



```
#> [7] "tp1_MN_L2MIASHSpdf.pdf" "tp1_MN_L2MIASHSpdf.rmd" "tp1_MN_L2MIASHSpdf.tex"
#> [10] "x.RData"
```

Il est possible de ne considérer que les fichiers d'un format donné et/ou comportants un *pattern* spécifique. Par exemple

```
list.files(path = '.', pattern = "L2MIASH.*")
#> [1] "introR_MN_L2MIASHS.pdf" "introR_MN_L2MIASHS.rmd" "introR_MN_L2MIASHS.tex"
#> [4] "tp1_MN_L2MIASHSpdf.pdf" "tp1_MN_L2MIASHSpdf.rmd" "tp1_MN_L2MIASHSpdf.tex"
```

3.4 Déclaration

La déclaration peut-être effectuée via le symbole `=` ou le symbole `<-`. Il est préférable d'utiliser `<-`. Lorsque le langage R (et S avant lui) a été créé, `<-` était le seul choix d'opérateur d'affectation. C'est un héritage du langage APL (sur les claviers APL il y avait une touche sur le clavier avec le symbole `<-`), où la notation fléchée était utilisée pour distinguer l'affectation (assigner la valeur 3 à x) de l'égalité (x est-il égal à 3?). Cependant, de nombreux langages (comme le C, par exemple) utilisent `=` pour l'affectation, de sorte que les débutants utilisant R considèrent souvent la notation fléchée comme peu pratique, et sont enclins à utiliser `=`. Mais R utilise `=` dans un autre but : associer des arguments de fonction à des valeurs (comme dans `pnorm(1, sd = 2)`, pour fixer l'écart type à 2). R a ajouté en 2001 la possibilité d'utiliser `=` comme opérateur d'affectation, en partant du principe que l'intention (affectation ou association) est généralement claire en fonction du contexte. Ainsi,

```
x = 1
```

signifie clairement "assigner 1 à x ", alors que

```
f(x = 1)
```

signifie clairement "appeler la fonction `f`, en donnant la valeur 1 à l'argument x ".

Il existe une situation où une ambiguïté peut se produire : si vous voulez assigner une variable pendant un appel de fonction. Une façon de le faire dans les versions modernes de R est :

```
f(x <- 1)
```

ce qui signifie "assigner 3 à x , et appeler `f` avec le premier argument défini à la valeur 1" (un autre moyen étant possible avec la syntaxe suivante, `f((x=1))`). Il s'agit donc d'une question de préférence.

3.5 Sauvegarde, chargement et suppression

Par défaut, R conserve en mémoire les objets créés. Lors de la fermeture d'une session (e.g., avec la commande `quit` ou l'alias de cette dernière `q` ou encore via le bouton dédié de la barre d'outils de l'interface R), ces derniers peuvent être sauvegardés dans un fichier `.RData` comportant une image de la session qui sera chargée à la prochaine ouverture. Il est également possible de sauvegarder, à tout moment, l'environnement de travail via la commande `save.image`. La commande `save` permet de ne sauvegarder que certains objets (dans le répertoire courant, ou ailleurs en spécifiant le chemin désiré).

```
x <- 1
save(x, file = "x.RData")
```

```
save(x, file = "/somewhere/else/x.RData")
```

Le chargement d'un fichier `.RData` s'effectue via la commande `load`.

```
load(file = "x.RData")
```

La commande `ls` renvoie les objets déclarés.

```
z <- y <- x
ls()
#> [1] "x" "y" "z"
```

La commande `remove` (ou son alias `rm`) permet de supprimer (définitivement) un objet de l'environnement de travail actuellement actif,

```
rm(y)
ls()
#> [1] "x" "z"
```

ou de la totalité des objets déclarés

```
rm(list = ls())
ls()
#> character(0)
```

3.6 Installation de packages

La distribution courante de R contient un certain nombre de packages installés en même temps que le logiciel. La liste de ce dernier peut être obtenue via la commande `installed.packages`. `install.packages` permet l'installation d'un package additionnel. La liste des packages est accessible sur le site du CRAN (ou via la documentation) .

Les commandes `library` et `require` chargent et attachent les packages additionnels (à ceux de la version native) que l'on souhaiterait utiliser.

```
library("splines")
```

`require` est conçu pour être utilisé à l'intérieur d'autres fonctions ; il retourne `FALSE` et donne un avertissement (plutôt qu'une erreur comme le fait `library` par défaut) si le package n'a pas été préalablement installé.

L'instruction `sessionInfo()` permet d'obtenir des informations sur la session : la version de R, du système d'exploitation, son architecture, l'encodage utilisé, les packages chargés et/ou attachés de la session.

```
sessionInfo()
#> R version 3.5.2 (2018-12-20)
#> Platform: x86_64-apple-darwin15.6.0 (64-bit)
#> Running under: macOS 10.16
#>
#> Matrix products: default
#> BLAS: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRblas.0.dylib
#> LAPACK: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRlapack.dylib
```

```

#>
#> locale:
#> [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
#>
#> attached base packages:
#> [1] stats      graphics  grDevices  utils      datasets  methods    base
#>
#> other attached packages:
#> [1] gasper_1.1.0.9000
#>
#> loaded via a namespace (and not attached):
#> [1] Rcpp_1.0.4.6      RSpectra_0.16-0  pillar_1.4.7      compiler_3.5.2
#> [5] tools_3.5.2       digest_0.6.25    evaluate_0.14     lifecycle_0.2.0
#> [9] tibble_3.0.6      gtable_0.2.0     lattice_0.20-38   pkgconfig_2.0.2
#> [13] rlang_0.4.10      Matrix_1.3-4     DBI_1.0.0         yaml_2.2.0
#> [17] xfun_0.28         fastmap_1.1.0    stringr_1.4.0     dplyr_1.0.4
#> [21] knitr_1.31        generics_0.0.2   vctrs_0.3.6       grid_3.5.2
#> [25] tidyselect_1.1.0  glue_1.4.2       R6_2.3.0          rmarkdown_2.11
#> [29] ggplot2_3.3.3     purrr_0.3.4      magrittr_1.5       scales_1.0.0
#> [33] ellipsis_0.3.0    htmltools_0.5.2  assertthat_0.2.0  colorspace_1.4-0
#> [37] stringi_1.2.4     munsell_0.5.0    crayon_1.3.4

```

3.7 Documentation

Pour avoir accès à l'aide associée à une fonction, la commande `help` ou le symbole `?` peuvent être utilisées.

```

help(help)
?help

```

Cette commande peut également être utilisé pour avoir accès à la documentation d'un package donné (et de toutes les fonctions qui le compose).

```

help(package = "MASS")

```

Une recherche par expression régulière est également possible, via la commande `apropos`.

```

apropos("sum")
#> [1] ".colSums"          ".rowSums"
#> [3] ".tryResumeInterrupt" "colSums"
#> [5] "contr.sum"         "cumsum"
#> [7] "format.summaryDefault" "print.summary.table"
#> [9] "print.summary.warnings" "print.summaryDefault"
#> [11] "rowsum"            "rowsum.data.frame"
#> [13] "rowsum.default"    "rowSums"
#> [15] "sum"               "summary"
#> [17] "Summary"           "summary.aov"
#> [19] "summary.connection" "summary.data.frame"
#> [21] "Summary.data.frame" "summary.Date"

```

```
#> [23] "Summary.Date"           "summary.default"
#> [25] "Summary.difftime"       "summary.factor"
#> [27] "Summary.factor"         "summary.glm"
#> [29] "summary.lm"             "summary.manova"
#> [31] "summary.matrix"         "Summary.numeric_version"
#> [33] "Summary.ordered"        "summary.POSIXct"
#> [35] "Summary.POSIXct"        "summary.POSIXlt"
#> [37] "Summary.POSIXlt"        "summary.proc_time"
#> [39] "summary.srcfile"        "summary.srcref"
#> [41] "summary.stepfun"        "summary.table"
#> [43] "summary.warnings"       "summaryRprof"
```

Par ailleurs, R dispose également d'une documentation en ligne très riche, accessible via la commande `help.start()`. Le manuel *An Introduction to R* constitue un bon point de départ pour se familiariser avec R.

Assez rapidement, on est amené à écrire des scripts (suite d'instructions R), puis des fonctions et il est préférable de travailler avec un éditeur de texte ou un environnement de développement intégré (IDE). R dispose de son propre éditeur de texte. Cependant, l'usage de **RStudio** est recommandé dans le cadre de ce cours.

4 L'environnement Rstudio

RStudio est un IDE pour R. Il est disponible en deux formats : **RStudio Desktop** une application de bureau ordinaire tandis que **RStudio Server** fonctionne sur un serveur distant et permet d'accéder à **RStudio** à l'aide d'un navigateur web. De nombreux autres IDE ou éditeurs standards supportent R (e.g., **Emacs**, **Vim**, **Kate**, **Notepad++**, **Jupyter Notebook**,...). Certaines fonctionnalités spécifiques à **Rstudio** en font un IDE particulièrement attractif :

- accès local à **RStudio** ;
- coloration syntaxique, complétion de code et indentation intelligente ;
- exécution du code R directement à partir de l'éditeur de sources ;
- accédez rapidement aux définitions des fonctions ;
- visualisez les modifications de contenu en temps réel avec l'éditeur visuel Markdown ;
- gérez facilement plusieurs répertoires de travail à l'aide de projets ;
- aide et documentation R intégrées ;
- débogueur interactif pour diagnostiquer et corriger les erreurs ;
- outils étendus de développement de *packages*, d'application web **shiny**, de page web standards, ... ;
- versionner du code avec **git** ;
- interpréter du code **python** ou compiler du code C/C++ (via les package **reticulate** et **Rcpp**) ;
- des outils de reporting via **Rmarkdown** ou **Notebook** ;
- ...

Rstudio fournit une interface qui réunit notamment, au sein d'une même fenêtre : une console R, un éditeur de texte, l'historique des instructions effectuées, une interface graphique, la documentation, un accès au shell du système, ...

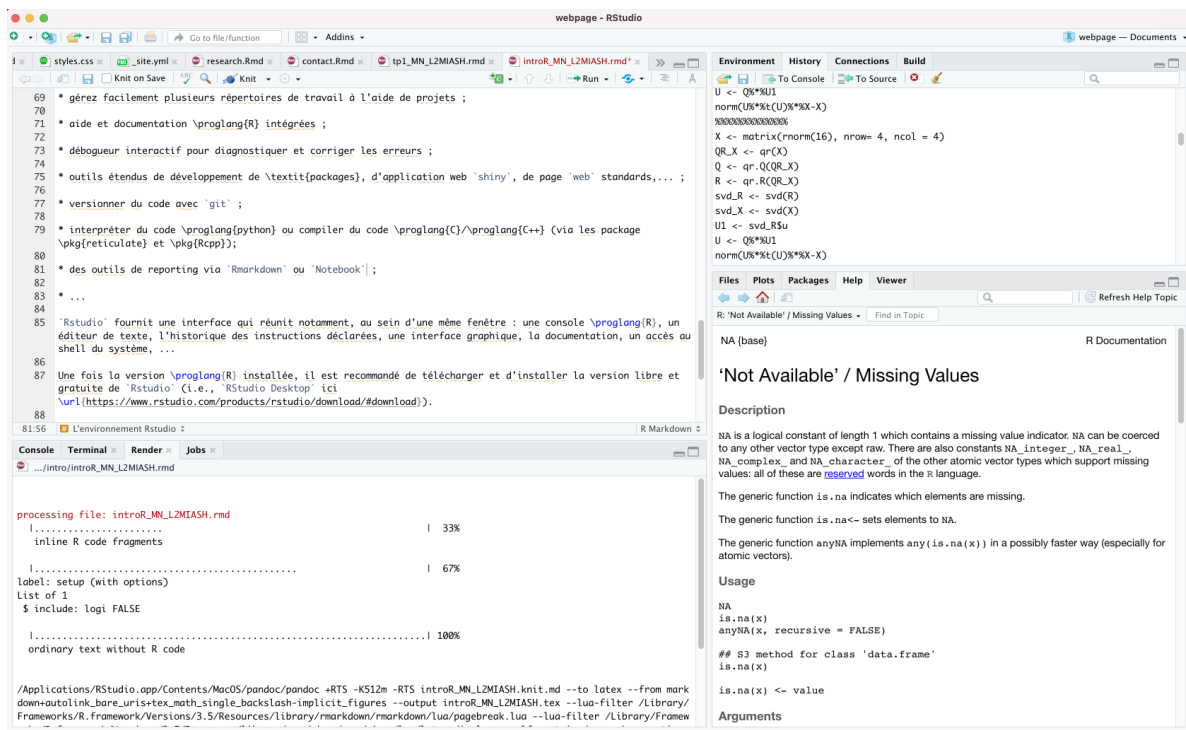


FIGURE 2 – Interface ‘Rstudio’.

Une fois la version R installée, il est recommandé de télécharger et d’installer la version libre et gratuite RStudio Desktop (ici <https://www.rstudio.com/products/rstudio/download/#download>).