Rmd: Code Chunks

Table of Contents

En plus du texte libre au format Markdown, un document R Markdown contient, comme son nom l'indique, du code R. Celui-ci est inclus dans des blocs (chunks) délimités par la syntaxe suivante :

```
```{r}
x <- 1:5
```
```

On peut insérer un code chunks R à l'aide de la barre d'outils RStudio (le bouton Insert) ou du raccourci clavier Ctrl + Alt + I (Cmd + Option + I sur macOS).

Utilité

On peut faire beaucoup de choses dans un bloc de code:

- On peut produire une sortie texte, des tableaux ou des graphiques.
- On a un contrôle précis sur toutes ces sorties via des options de bloc, qui peuvent être fournies entre accolades (entre `{r et } ). Par exemple, vous pouvez choisir de masquer la sortie de texte via l'option **chunk results = 'hide'**, ou de régler la hauteur de la figure à 4 pouces via fig.height = 4 . Les options de chunk sont séparées par des virgules, par exemple:`

```
{r, chunk-label, results='hide', fig.height=4}
```

La valeur d'une option de bloc peut être une expression R arbitraire, ce qui rend les options de bloc extrêmement flexibles. Par exemple, l'option chunk **eval** détermine s'il faut évaluer (exécuter) un morceau de code et vous pouvez évaluer de manière conditionnelle un morceau avec une variable définie précédemment, par exemple:

```
 ```{r}
 # execute code if the date is later than a specified day
 do_it = Sys.Date() > '2018-02-14'
```
```

```
```
{r, eval=do_it} x = rnorm(100)
```
```

Options de code shunks

Il existe un grand nombre d'options de chunks dans knitr:

- **eval**: eval s'il faut évaluer un morceau de code.
- **echo**: Indique si le code source doit être répercuté dans le document de sortie (quelqu'un peut ne pas préférer lire votre code source intelligent, mais uniquement les résultats).
- **results**: lorsque réglé sur **hide** , la sortie de texte sera masquée; lorsqu'il est défini sur **asis** , la sortie de texte est écrite “telle qu'elle”, par exemple, on peut écrire du texte brut de Markdown à partir de code R (comme cat('**Markdown** is cool.\n')). Par défaut, la sortie de texte sera encapsulée dans des éléments in extenso (généralement des blocs de code simples).
- **collapse**: Indique si le texte et le code source doivent être fusionnés en un seul bloc de code dans la sortie. C’est surtout esthétique: **collapse=TRUE** rend la sortie plus compacte, car le code source R et sa sortie texte sont affichés dans un seul bloc de sortie. La valeur par défaut **collapse=FALSE** signifie que les expressions R et leur sortie texte sont séparées en différents blocs.
- **warning, message and error** : Indique si les avertissements, les messages et les erreurs doivent être affichés dans le document de sortie. Si **error=FALSE** est définit, **rmarkdown::render()** sera interrompu en cas d'erreur dans un code chunk et l'erreur sera affichée dans la console R. De même, lorsque **warning=FALSE** ou **message=FALSE**, ces messages seront affichés dans la console R.
- **include**: Indique si un élément de code doit être inclus dans le document de sortie. Lorsque **include=FALSE**, tout ce code est exclu de la sortie, mais il sera toujours évalué si **eval=TRUE**. Si on essaye de définir **echo=FALSE** , **results='hide'** , **warning=FALSE** et **message=FALSE** , utiliser simplement la seule option, **include=FALSE** au lieu de supprimer différents types de texte individuellement.
- **cache**: Indique s'il faut activer la mise en cache. Si la mise en cache est activée, le même fragment de code ne sera pas évalué lors de la prochaine compilation du document (si le fragment de code n'a pas été modifié), ce qui peut faire gagner du temps.
- **fig.width** et **fig.height**: taille graphique des tracés R en pouces. Les tracés R dans les fragments de code sont d'abord enregistrés via un périphérique graphique en knitr , puis écrits dans des fichiers. On peut spécifier les deux options ensemble dans une seule option de bloc fig.dim , par exemple, fig.dim = c(6, 4) signifie fig.width = 6 et fig.height = 4 .
- **out.width** et **out.height**: taille de sortie des tracés R dans le document de sortie. Ces options peuvent redimensionner les images. On peut utiliser des pourcentages. Par exemple, out.width = '80%' signifie 80% de la largeur de la page.
- **fig.align**: alignement des chunks. Cela peut être 'left' , 'center' ou 'right' .
- **dev** : Le périphérique graphique pour enregistrer les graphes R. En règle générale, il s'agit de **pdf** pour la sortie LaTeX et de **png** pour la sortie HTML, mais on peut utiliser d'autres périphériques, tels que **svg** ou **jpeg**.
- **fig.cap** : La légende de la figure.
- **child** : On peut inclure un document enfant dans le document principal. Cette option prend un chemin vers un fichier externe.

Les options de chinks dans knitr peuvent être trés puissantes. Par exemple, on peut créer des animations à partir d'une série de tracés dans un morceau de code.

Il existe une option de bloc facultative qui ne prend aucune valeur, à savoir le libellé du bloc. Ce devrait être la première option dans l'en-tête de chunk. Les étiquettes de chunks sont principalement utilisées dans les noms de fichiers de parcelles et de cache. Si l'étiquette d'un chunk est manquante, une valeur par défaut de la forme **unnamed-chunk-i** sera générée, où i est incrémental.

N'utiliser que des caractères alphanumériques (az , AZ et 0-9) et des tirets (-) dans les libellés, car ils ne sont pas des caractères spéciaux et fonctionneront sûrement pour tous les formats de sortie. D'autres caractères, en particulier les espaces et les tirets bas, peuvent causer des problèmes dans certains paquets, tels que bookdown.

Si une option donnée doit souvent être définie sur une valeur dans plusieurs fragments de code, on peut la définir globalement dans le premier fragment de code du document, par exemple,

```
 ```{r, setup, include=FALSE}
 knitr::opts_chunk$set(fig.width = 8, collapse = TRUE)
 ```
```

Outre les fragments de code, on peut également insérer des valeurs d'objets R en ligne dans le texte. Par exemple:

```
```    {r}
x = 5 # radius of a circle
```
For a circle with the radius `rx` , its area is `r pi * x^2` .
```

Chiffres

Par défaut, les chiffres produits par le code R seront placés immédiatement après le bloc de code à partir duquel ils ont été générés. Par exemple:

```
```{r}
plot(cars, pch = 18)
```
```

On peut fournir une légende sous fig.cap dans les options chunk. Si le format de sortie du document prend en charge l'option **fig_caption: true** (par exemple, le format de sortie **rmarkdown::html_document**), les tracés R seront placés dans des environnements de figures. Dans le cas d'une sortie PDF, ces chiffres seront automatiquement numérotés.

Les documents PDF sont générés via les fichiers LaTeX générés à partir de R. Markdown. Un fait très surprenant pour les débutants de LaTeX est que les figures flottent par défaut: même si on génére un tracé dans un bloc de code sur la première page, l'environnement de la figure entière peut passer à la page suivante. C'est comme ça que LaTeX fonctionne par défaut. Il a tendance à faire flotter des

chiffres en haut ou en bas des pages. Bien que cela puisse être ennuyeux. Il est préfrable de ne pas essayer désespérément de positionner les figures «correctement» pedant la pahese conception, mais d'ajuster les positions une fois le contenu terminé à l'aide de l'option **fig.pos chunk** (par exemple, fig.pos = 'h').

Pour placer plusieurs figures côte à côte à partir du même bloc de code, on peut utiliser l'option **fig.show='hold'** avec l'option **out.width**. Par exemple avec deux parcelles ayant chacune une largeur de 50% .

```
par ( mar = c ( 4 , 4 , 0.2 , 0.1 ))
plot (cars, pch = 19 )
plot (pressure, pch = 17 )
```

Pour inclure un graphique qui n'est pas généré à partir de code R, utiliser la fonction **knitr::include_graphics()** , qui permet de mieux contrôler les attributs de l'image que la syntaxe Markdown `![alt text or image title](path/to/image)` (par exemple, on peut spécifier la largeur de l'image via **out.width**).

```
```{r, out.width='25%', fig.align='center', fig.cap='...'}
knitr::include_graphics('images/hex-rmarkdown.png')
```
```

Tableaux

La méthode la plus simple pour inclure des tableaux consiste à utiliser **knitr::kable()** , qui permet de créer des tableaux pour les sorties HTML, PDF et Word. Les légendes de table peuvent être incluses en transmettant une caption à la fonction, par exemple,

```
```{r tables-mtcars}
knitr::kable(iris[1:5, ], caption = 'A caption')
```
```

Les tables dans des formats de sortie autres que LaTeX seront toujours placées après le bloc de code. Pour les formats de sortie LaTeX / PDF, les tableaux ont le même problème que les figures: ils peuvent flotter. Pour éviter ce problème, utiliser le package longTable de LaTeX, qui peut décomposer des tables sur plusieurs pages. Ceci peut être réalisé en ajoutant **\usepackage{longtable}** au préambule LaTeX et en transmettant **longtable = TRUE à kable()**.

Pour un contrôle plus avancé du style des tableaux, il est recommandé d'utiliser le package **kableExtra**, qui fournit des fonctions permettant de personnaliser l'apparence des tableaux PDF et HTML. Le formatage des tableaux peut s'avérer une tâche très compliquée, notamment lorsque certaines cellules s'étendent sur plus d'une colonne ou d'une ligne. C'est encore plus compliqué lorsqu'on doit considérer différents formats de sortie. Par exemple, il est difficile de faire fonctionner un tableau complexe à la fois pour la sortie PDF et HTML.

Autres moteurs linguistiques

Un fait moins connu sur R Markdown est que de nombreux autres langages sont également pris en charge, tels que Python, Julia, C ++ et SQL. Le support provient du paquetage **knitr** , qui a fourni un grand nombre de moteurs de langage . Les moteurs de langage sont essentiellement des fonctions enregistrées dans l'objet **knitr::knit_engine**. On peut lister les noms de tous les moteurs disponibles via:

```
names (knitr::knit_engines$get())
## [1] "awk" "bash" "coffee"
## [4] "gawk" "groovy" "haskell"
## [7] "lein" "mysql" "node"
## [10] "octave" "perl" "psql"
## [13] "Rscript" "ruby" "sas"
## [16] "scala" "sed" "sh"
## [19] "stata" "zsh" "highlight"
## [22] "Rcpp" "tikz" "dot"
## [25] "c" "fortran" "fortran95"
## [28] "asy" "cat" "asis"
## [31] "stan" "block" "block2"
## [34] "js" "css" "sql"
## [37] "go" "python" "julia"
## [40] "theorem" "lemma" "corollary"
## [43] "proposition" "conjecture" "definition"
## [46] "example" "exercise" "proof"
## [49] "remark" "solution"
```

La plupart des moteurs ont été documentés au chapitre 11 de Xie (2015). Les moteurs du theorem solution ne sont disponibles que lorsque vous utilisez le package bookdown , le reste étant livré avec le package knitr . Pour utiliser un moteur de langue différent, changer le nom de la langue dans l'en-tête du bloc de r en nom du moteur, par exemple,

````
```{python}
x = 'hello, python world!'
print(x.split(' '))
```
````

Pour les moteurs qui utilisent des interpréteurs externes tels que python , perl et ruby , les interpréteurs par défaut sont obtenus à partir de Sys.which() , c'est-à-dire en utilisant l'interpréteur trouvé via la variable d'environnement PATH du système. Pour utiliser un autre interpréteur, spécifier son chemin dans l'option **chunk engine.path**. Par exemple, pour utiliser Python 3 au lieu du Python 2 par défaut, et en supposant que Python 3 se trouve dans /usr/bin/python3:

````
```{python, engine.path = '/usr/bin/python3'}
import sys
print(sys.version)
```
````

On peut également modifier les interpréteurs de moteur globalement pour plusieurs moteurs, par

exemple,

```
knitr::opts_chunk $set
(engine.path=list(python='~/anaconda/bin/python',ruby='/usr/local/bin/ruby')
)
```

La plupart des moteurs exécutent chaque bloc de code dans une nouvelle session distincte (via un appel system() dans R), ce qui signifie que les objets créés en mémoire dans un bloc de code précédent ne seront pas directement disponibles pour les derniers morceaux de code. Par exemple, si on crée une variable dans un bloc de code bash, on ne peut pas l'utiliser dans le prochain bloc de code bash . Actuellement, les seules exceptions sont r , python et julia . Seuls ces moteurs exécutent du code dans la même session tout au long du document. Pour clarifier, tous les r code r sont exécutés dans la même session R, tous les code pythonsont exécutés dans la même session Python, etc., mais la session R et la session Python sont indépendantes.

Python

Le moteur python est basé sur le package réticulé (J. Allaire, Ushey et Tang 2018) , ce qui permet d'exécuter tous les fragments de code Python dans la même session Python. Pour exécuter un certain morceau de code dans une nouvelle session Python, utiliser l'option **chunk python.reticulate=FALSE**.

Scripts de shell

On peut également écrire des scripts Shell dans R Markdown si le système peut les exécuter (l'exécutable bash ou sh devrait exister). Ce n'est généralement pas un problème pour les utilisateurs Linux ou macOS.

```
```{bash}
echo "Hello Bash!"
cat flights1.csv flights2.csv flights3.csv > flights.csv
```
```

Les scripts shell sont exécutés via la fonction system2() de R. En gros, knitr transmet un bloc de code à la commande bash -c pour l'exécuter.

SQL

Le moteur sql utilise le package DBI pour exécuter des requêtes SQL, imprimer leurs résultats et éventuellement affecter ces résultats à un cadre de données.

Pour utiliser le moteur sql , il faut d’abord établir une connexion DBI à une base de données (généralement via la fonction DBI::dbConnect()). On peut utiliser cette connexion dans un bloc sql via

l'option de connection . Par exemple:

```
```{r}
library(DBI)
db = dbConnect(RSQLite::SQLite(), dbname = "sql.sqlite")
```
```{sql, connection=db} SELECT * FROM trials
```
```

Par défaut, les requêtes SELECT affichent les 10 premiers enregistrements de leurs résultats dans le document. Le nombre d'enregistrements affichés est contrôlé par l'option **max.print** , elle-même dérivée de l'option globale **knitr sql.max.print** (par exemple, **knitr::opts_knit$set(sql.max.print = 10)**). Par exemple, le bloc de code suivant affiche les 20 premiers enregistrements:

```
```{sql, connection=db, max.print = 20}
SELECT * FROM trials
```
```

On peut retirer la limite pour les enregistrements à afficher via **max.print=-1** ou **max.print=NA**.

Par défaut, le moteur sql inclut une légende indiquant le nombre total d'enregistrements affichés. On peut remplacer cette légende à l'aide de l'option de bloc **tab.cap**. Par exemple:

```
```{sql, connection=db, tab.cap="My Caption"}
SELECT * FROM trials
```
```

On peut spécifier qu'on ne veut aucune légende via **tab.cap=NA**.

Pour affecter les résultats de la requête SQL à un objet R en tant que cadre de données, utiliser l'option **output.var**, par exemple,

```
```{sql, connection=db, output.var="trials"}
SELECT * FROM trials
```
```

Lorsque les résultats d'une requête SQL sont affectés à un bloc de données, aucun enregistrement ne sera imprimé dans le document (imprimer manuellement le bloc de données dans un fragment R ultérieur).

Pour lier les valeurs de variables R dans des requêtes SQL, préfacer les références de variables R avec un ? . Par exemple:

```
 ```{r}
 subjects=10
 ```
 ```{sql, connection=db, output.var="trials"}
 SELECT * FROM trials WHERE subjects >= ?subjects
 ```
```

Lorsqu'on a plusieurs chunks SQL, il peut être utile de définir une option par défaut pour le chunks de

connection dans le chunk setup, de sorte qu'il ne soit pas nécessaire de spécifier la connexion sur chaque chunk individuel:

```
```{r setup}
library(DBI)
db=dbConnect(RSQLite::SQLite(), dbname="sql.sqlite")
knitr::opts_chunk$set(connection = "db")
```
```

l'option de connection doit être une chaîne nommant l'objet de connexion (pas l'objet lui-même). Une fois défini, on peut exécuter des fragments SQL sans spécifier de connexion explicite:

```
```{sql}
SELECT * FROM trials
```
```