Documents dynamiques et création de packages

Hadrien Commenges, Timothée Giraud

03/10/2014

Packages et données nécessaires

Packages nécessaires :

- `knitr` pour la création de documents dynamiques
- `reshape2` pour la transposition de tableaux
- `roxygen2` pour la création du *package*
- `devtools` pour le création du *package*

Données nécessaires :

- **PopIdf.csv** : population dans les communes d'Île-de-France entre 1968 et 2010, disponible sur le site de l'Insee
- **NavIdf.csv** : navettes domicile-travail entre les communes d'Île-de-France en 2010, disponible sur le site de l'Insee
- **DistIdf.csv** : distances à vol d'oiseau entre les communes d'Île-de-France en 2010, créée à partir des données GEOFLA disponibles sur le site de l'IGN

La recherche reproductible

Le terme de "recherche reproductible" désigne un changement récent dans la production scientifique: c'est l'idée que le produit final de la recherche ne peut plus être un simple article, mais un article accompagné de tous les éléments nécessaires pour le reproduire, c'est-à-dire principalement le code et les données. Pour que d'autres puissent reproduire le résultat à partir du code et des données, il faut qu'ils soient compréhensibles d'où une nécessité de documenter tous les éléments (code et données) acompagnant l'article.

Plusieurs liens pour approfondir cette question des conditions, des limites et des enjeux de la reproductibilité des résultats scientifiques :

- Plusieurs articles de fond dans le dossier à télécharger contenant le matériel de la séance
- Un article du Monde : http://www.lemonde.fr/sciences/article/2013/07/15/pour-une-recherche-reproductible-publiez-vos
3447825_1650684.html
- Un article du Wall Street Journal : http://online.wsj.com/news/articles/SB10001424052970203764804577059841672541590
- Un exemple sur les archives ouvertes (HAL) : http://hal.archives-ouvertes.fr/hal-00591455

Création de documents dynamiques avec R et markdown

Qu'est-ce qu'un document dynamique ?

L'utilisateur de R peut visualiser des résultats numériques et graphiques, mais il a souvent besoin d'exporter ces sorties pour les intégrer dans des documents rédigés. Deux types de flux de travail (*workflow*) peuvent être distingués. Le plus classique consiste à découpler l'écriture du texte et la production des sorties numériques ou graphiques : l'utilisateur produit un graphique avec R, l'exporte dans un format d'image et l'insère dans un logiciel de traitement de texte avec lequel il écrit le contenu. L'autre façon de procéder, de plus en plus

développée, s'inscrit dans la vague de la *reproducible research*. Elle consiste à combiner l'écriture des contenus textuels et des traitements numériques et graphiques pour générer un document tout-en-un. Ce flux de travail est souvent qualifié de *dynamic report generation* ou parfois (de façon abusive) *literate programming*.

Pour produire un document qui fourmille de traitements et de graphiques, le document dynamique, qui combine l'écriture et les traitements dans un même document, est très pratique. R et RStudio disposent d'outils très pratiques pour cela grâce au *package* `knitr` (voir le site dédié : http://yihui.name/knitr). Dans l'interface de RStudio, le menu `File~>~New file` propose plusieurs solutions : `R Sweave` pour une combinaison avec Latex (le manuel *R et espace* est écrit de la sorte), `R Markdown`, `R HTML` et `R Presentation` pour produire des documents écrits en markdown et en HTML. À la date d'écriture de ces lignes, la version de RStudio la plus récente (version 0.98.1062) permet d'écrire un document en markdown et de compiler le texte et le code pour obtenir une sortie en pdf, en HTML et en docx (Microsoft Word).

Cette première section présente l'utilisation de R avec Markdown pour intégrer les contenus textuels, les scripts et les sorties numériques ou graphiques de ces scripts. Il s'agit d'écrire le texte en Markdown et le code en R, ensuite le *package* `knitr` se charge de tout compiler et de créer le document.

Qu'est-ce que Markdown ?

Markdown est le plus simple des langages à balises (http://fr.wikipedia.org/wiki/Markdown). Sur des logiciels comme MS Word ou LibreOffice Writer, l'utilisateur voit le résultat final mais les commandes de mise en forme lui sont invisible (logiciels WYSIWYG - *What You See Is What You Get*). Avec Markdown, Latex ou HTML, l'utilisateur écrit et voit les commandes de mise en forme mais pas directement le résultat final.

Trois avantages à utiliser Markdown plutôt que Latex : l'installation est très simple, quel quel soit le système opérationnel ; le langage est très simple et s'apprend en cinq minutes ; la transformation en d'autres formats (pdf, docx, HTML) est très simple. Bref, tout est très simple !

Installation et aide

Pour travailler avec R + Markdown :

- Utiliser RStudio et une version récente de RStudio (version 0.98.1028 actuellement). Ce n'est pas la seule manière, mais RStudio intègre de nombreuses fonctionnalités destinées à cet usage.
- Installer les *packages* suivants : `knitr, yaml, htmltools, caTools, bitops, rmarkdown`. Dans RStudio, en essayant de créer un fichier Markdown pour la première fois (`File > New file > R Markdown`), ces *packages* sont installés automatiquement.
- Créer ou ouvrir un fichier .Rmd (`File > New file > R Markdown`). Le résultat peut être visualisé en HTML, pdf et docx. Le plus simple et rapide est de produire une sortie HTML, pour produire des documents pdf et docx il faut des installations supplémentaires. Pour visualiser, cliquer sur `Knit HTML`

Des références pour la syntaxe Markdown :

- sur RStudio, il y a un onglet **Markdown Quick Reference** en cliquant sur le `?` à côté du bouton `Knit HTML`
- sur le site de RStudio http://rmarkdown.rstudio.com
- sur plusieurs sites web, par exemple http://daringfireball.net/projects/markdown/syntax

La référence pour l'utilisation du *package* `knitr` qui se charge de la compilation est le site de l'auteur du *package* : http://yihui.name/knitr/options

---

Le codage des caractères

Pour éviter des migraines au moment de mettre en forme des textes écrits par plusieurs utilisateurs sur des systèmes opérationnels différents, **il faut absolument utiliser le codage de caractères UTF-8 pour la rédaction du fichier .Rmd** (cf. http://fr.wikipedia.org/wiki/UTF-8). Dans RStudio, ce réglage se fait à `Tools > Global options > General > Default text encoding`.

Les utilisateurs de Windows auront probablement un codage Windows (Windows-1252, ANSI ou ISO-8859-1). Il s'agit d'un standard "universel Windows" qui tend à disparaître (ou qui devrait disparaître si Microsoft se décidait enfin à l'abandonner) au profit du standard vraiment universel qu'est UTF-8 .

Utilisation

Concernant la mise en forme du texte, l'essentiel se trouve dans le **Markdown Quick Reference**.

Concernant l'intégration des traitements et des sorties dans le document, le fonctionnement est le suivant : on crée des *chunks* (des morceaux ?) comme ci-dessous dans lesquels on écrit le code qui doit être interprété par R. On crée le ckunk soit en cliquant sur l'onglet `Chunks > Insert chunk`, soit en tapant `CTRL+ALT+I`, soit en copiant-collant un *chunk* existant.

Le *chunk* sans option se présente ainsi :

```
data(iris)
summary(iris$Sepal.Length)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   4.300   5.100   5.800   5.843   6.400   7.900
```

> Le premier exemple est montré par une capture d'écrant, pour les suivants il faut voir directement le fichier brut au format .Rmd dans le dossier en téléchargement.

Dans ce premier exemple minimal, voici les réglages par défaut de plusieurs arguments qui peuvent être modifiés :

- le *chunk* n'a pas de nom
- le code R est interprété (`eval`)
- le code est affiché dans la sortie (`echo`)
- le résultat numérique ou graphique est affiché dans la sortie (`results` et `fig.show`)
- le résultat des traitements est mis en mémoire cache (`cache`)

Pour la suite, il est conseillable de toujours nommer le *chunk*. Le nom vient juste après la lettre `r`. Si plusieurs *chunks* partagent le même nom, le fichier ne peut pas être compilé.

```
data(iris)
summary(iris$Sepal.Length)
```

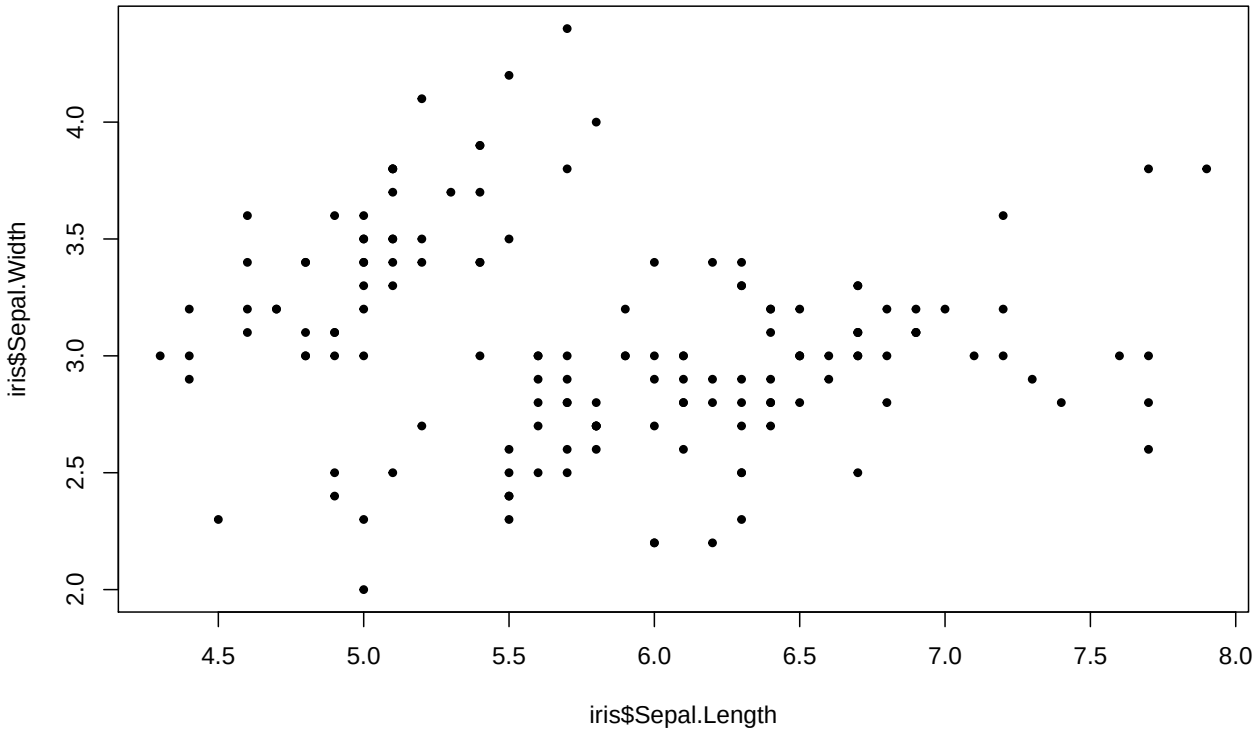
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   4.300   5.100   5.800   5.843   6.400   7.900
```

L'autocomplétion avec la touche `Tab` fonctionne aussi dans ce cadre (d'où l'intérêt d'utiliser RStudio) : se placer dans l'entête du *chunk* après une virgule et taper `Tab` pour afficher tous les arguments.

Pour afficher du code sans l'exécuter (`eval`) :

```
library(spdep)
```

Pour afficher la sortie mais masquer le code (`echo`) :



Pour afficher le code mais masquer la sortie numérique (`results`) :

```
summary(iris$Sepal.Length)
```

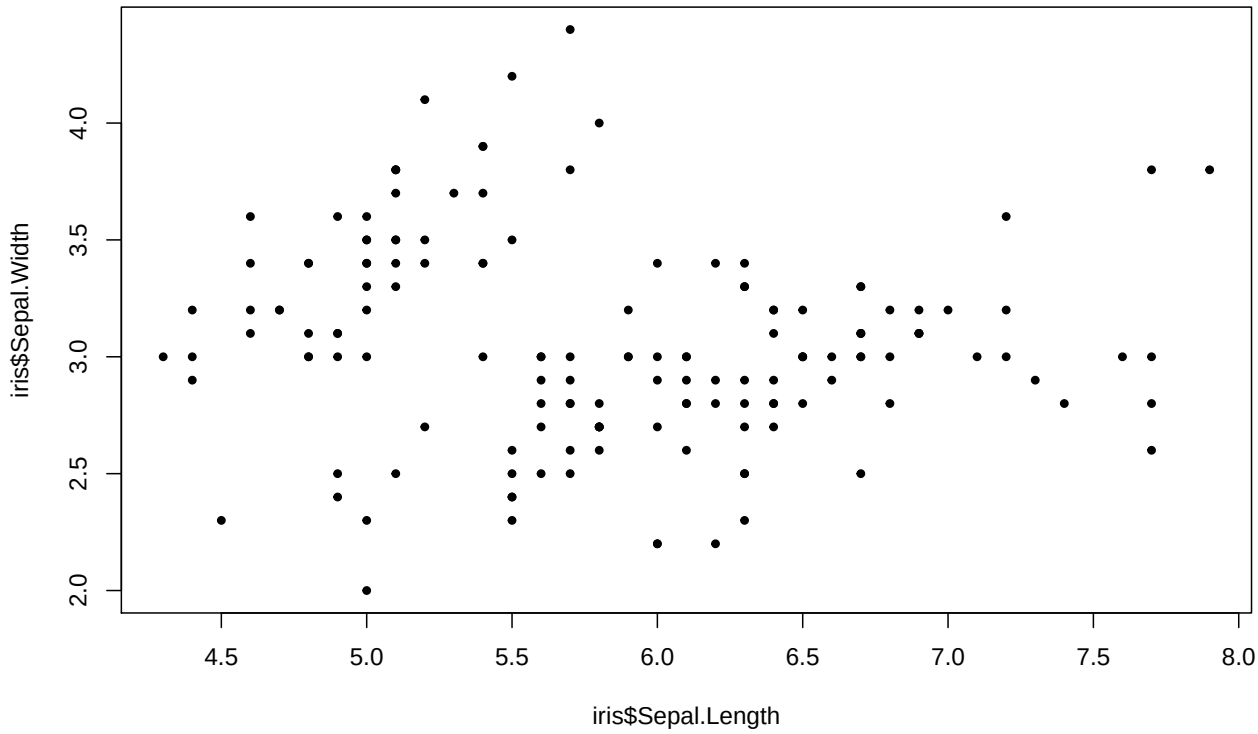
Pour afficher le code mais masquer la sortie graphique (`fig.show`) :

```
plot(iris$Sepal.Length, iris$Sepal.Width, pch = 20)
```

Enfin la question du cache (`cache`) : lors de la rédaction du document, l'utilisateur compile régulièrement le fichier pour voir le rendu final. À chaque compilation, le résultat des traitements est mis par défaut en mémoire cache pour éviter de ré-exécuter des traitements déjà exécutés et diminuer le temps de compilation. C'est très utile mais ça peut être source de bugs.

Toute modification dans l'en-tête du *chunk* (changer le nom du *chunk*, ou changer une option) ou dans le corps du *chunk*, les modifications sont prises en compte et le résultat (un graphique par exemple) écrase la version antérieure. Ceci peut poser problème si on fait une modification dans un autre *chunk* (modification d'une valeur par exemple) qui affecte un graphique créé par ailleurs et mis dans le cache. Dans ce cas, le graphique ne serait pas créé à nouveau et ne prendra pas en compte le modification de la valeur.

```
plot(iris$Sepal.Length, iris$Sepal.Width, pch = 20)
```



On peut forcer à tout ré-exécuter à chaque compilation. Cependant, le plus pratique est de travailler avec l'option par défaut qui met les résultats dans la mémoire cache et de s'assurer de temps en temps que tout fonctionne en faisant une compilation "fraîche" : supprimer les dossiers qui stockent tous ces résultats ("chemin/Dossiercache" et "chemin/Dossierfiles").