S08: Produire des rapports d'analyse

Analyse de données quantitatives avec R

Samuel Coavoux

Knitr

Knitr

Markdown est un langage dit *markup* (HTML, Latex) conçu pour l'écriture web, et notamment pour les blogs. Il a donc vocation à être transformé en HTML. Son principe est d'être facile à lire. Le formatage est réduit au minimum, et toutes les instructions de format sont des signes de ponctuation.
Pandoc est un convertisseur universel de documents. Il est notamment employé pour transformer des documents markdown en d'autres formats, en particulier HTML, docx, et tex/pdf. Pandoc définit sa propre version de markdown, la plus adaptée à l'écriture universitaire (John McFarlane, son créateur, est professeur de philosophie à Berkeley).
Knitr est un package R qui emploie pandoc pour faciliter l'écriture de compte-rendu d'analyse statistique directement dans R.

Schéma

Un document Rmarkdown est composé de:

- un YAML front matter: un ensemble d'options;
- du texte écrit au format markdown;
- du code R écrit soit directement dans le texte, soit dans des *chunks*, des paragraphes séparés.

Knitr prend ce document, exécute l'ensemble du code R et remplace ce code par les résultats, sauvegarde ce document intermédiaire au format markdown, puis execute pandoc, qui transforme ce document au format docx, pdf ou html.

Chunks

Principe

Les chunks sont des paragraphes contenant du code R. Knitr va executer ce code et ajouter le résultat au document avant de le transformer vers son format final.
Syntaxiquement, un chunk est encadré de deux lignes signifiant son début et sa fin. Dans Rstudio, il apparaît avec un fond gris.

```
```{r}
# contenu du chunk
```
```

Pour insérer un chunk dans Rstudio, on emploie le raccourci ctrl+alt+i.

Attributs

La ligne ouvrant le chunk comprend plusieurs informations

```
```{engine titre, opt1="opt1", ...}
```

- engine: il s'agit le plus souvent de r; knitr comprend d'autres langages, comme python;
- titre: il est séparé de engine par une espace; il s'agit du titre du chunk, employé pour le retrouver facilement dans la table des matières (dans la barre d'état de Rstudio);
- options: des couples nom de l'option = valeur de l'option.
```

Chunk options

Il existe de très nombreuses options. Les plus utiles sont:

- eval: TRUE par défaut; le code est évalué;
- echo: TRUE par défaut; le code du chunk est répété dans le document;
- message: TRUE par défaut; renvoie les messages produits par les fonctions;
- warning: TRUE par défaut; renvoie les messages warnings produits par les fonctions du chunk;
- results: "markup" par défaut. Controle la façon dont s'affichent les résultats;
- dpi: 72 par défaut; controle la résolution des images (passer à 300 est habituellement une bonne idée).

Chunk options

Les options de chunk peuvent être précisées à chaque chunk. Si l'on souhaite changer le défaut dans tout le document, il suffit d'ajouter au document un premier chunk les spécifiant avec `knitr::opts_chunk$set()`

````
```{r}
library(knitr)
opts_chunk$set(echo = FALSE, message = FALSE,
               warning = FALSE, dpi = 300)
```
````

Chunk: évaluation

Lorsque l'on appuie sur le bouton knit (ou que l'on emploie la fonction `rmarkdown::render()`), les chunks sont évalués dans l'ordre dans lequel ils apparaissent, et dans un nouvel environnement.
Cela signifie qu'un document rmarkdown doit être **entièrement autosuffisant**. Il doit inclure une importation de données, des fonctions library() pour charger les packages, etc. Lorsque l'on écrit un document markdown, il faut garder à l'esprit que toutes les commande que l'on écrit directement dans la console ne seront pas reproduites lors de la compilation du document.

Chunk: écriture

En pratique, lors de l'écriture d'un document rmarkdown, on peut s'assurer de la cohérence du code en executant les chunks un par un. Pour cela, on peut soit:

- utiliser ctrl+entrée qui envoie la ligne dans la console comme dans un script R normal;
- utiliser ctrl+shift+c qui execute l'ensemble du chunk actuel;
- utiliser ctrl+shift+n qui execute l'ensemble du prochain chunk (en répétant cette commande depuis le début du document, on execute tout);
- cliquer sur la flèche vers le bas, en haut à droite de chaque chunk, pour executer tous les chunks précédents.

Inline code

Outre les chunks, on peut également employer du code R “inline”. La syntaxe à adopter, alors, est de mettre entre backtick r puis la commande à executer. Cela permet en particulier d’intégrer des résultats de calculs directement dans un texte au format markdown.

````
```{r}
x <- rnorm(100)
```

La moyenne de x est `r mean(x)`
````

Markdown

Principe

Markdown est donc un langage markup: il sert à écrire des documents avec une logique “what you see is what you mean” (par opposition aux traitements de texte “what you see is what you get”). Markdown emploie des signes de ponctuation pour donner une signification aux différents éléments d’un document.

Paragraphes

Tous les paragraphes pour lesquels il n'y a aucun marqueur particulier sont des paragraphes normaux. Ils doivent commencer nécessairement en début de ligne (pas d'espace ou de tabulation). Il faut sauter deux lignes pour changer de paragraphe.

```
Voici un paragraphe qui continue ici :
ce ne sont pas deux lignes différentes

Voici un second paragraphe
```

Titres

Un titre est une ligne commençant par un à six #

```
# Voici un titre de niveau 1

## Et un titre de niveau 2
```

Citations

Une citation est un paragraphe qui commence par un $>$

```
> Voici une citation.
```

Il apparaîtra dans un style différent des paragraphes normaux.

> *Voici une citation formattée*

Gras et italique

- Italique (emphasis) : il suffit d'encadrer le passage en italique de * ou de _

```
Voici *un passage en italique* et _un autre_
```

- Gras (strong) : il suffit d'encadrer le passage en italique de ** ou de __

```
Voici **un passage en gras** et __un autre__
```

Listes

Comme en HTML, les listes sont ordonnées ou non-ordonnées.
Les listes ordonnées sont des paragraphes commençant par -, +, ou *

```
+ premier item (on peut remplacer + par - ou *)
+ second item
```

Les listes non-ordonnées commencent par un chiffre et un point (n'importe quel chiffre : la liste est numérotée plus tard)

```
1. premier élément
1. deuxième élément (le 1. deviendra 2.
    à la compilation)
```

Listes et paragraphes

Si un item de liste fait plusieurs lignes, la deuxième ligne doit commencer par une tabulation.
Cet exemple produit un élément de liste :

```
1. premier élément avec un texte
    un peu long
```

Alors que celui-ci produira un élément et un paragraphe indépendant.

```
1. premier élément avec un texte
un peu long
```

Liens

Pour ajouter des liens urls (balise “” en HTML), on met le texte du lien (le texte qui s'affiche et que l'on peut cliquer) entre et l'adresse du lien, immédiatement après, entre (). Par exemple :

```
Chercher sur [google](http://www.google.fr)
```

Il est également possible de donner l'adresse du lien après le paragraphe. Dans ce cas, on mettra le texte du lien entre , suivi d'un renvoi entre

```
Chercher sur [google][1] ou [yahoo][y]

[1]: http://www.google.fr
[y]: http://www.yahoo.fr
```

Images

Les images fonctionnent presque exactement comme les liens. Elles comportent trois éléments : un !, un texte entre (qui correspond à l'attribut alt en HTML), l'adresse de l'image entre () (attribut href)

```
![logo](./logo_ens.png)
```

Comme les liens, on peut renvoyer l'adresse de l'image en dehors du paragraphe

```
![logo][1]

[1]: ./logo_ens.png
```

Notes de bas de page

Il existe deux types de notes, les notes inline et les notes pleines.

```
Texte normal^[texte de la note de bas de page].

Texte normal[^appel_note].

[^appel_note]: texte de la note de bas de page.
```

Références

Pandoc markdown et knitr comprennent les références bibliographiques. Pour cela, il faut lier un fichier au format bibtex (format bibliographique de latex) dans l'option bibliography du YAML front matter. Ensuite, on appel des références de la manière suivante:

```
Dans *La distinction* [@bourdieu1979]
```

Qui se transformera en:

```
Dans *La distinction* (Bourdieu, 1979)
```

Par défaut, les citations sont compilées selon le style Chicago, que l'on peut changer avec l'option `csl`.

Tableau

Il existe de nombreuses manières de faire des tableaux, qui nous concernent peu parce que nous allons les produire automatiquement. Voir l'aide en ligne de pandoc pour les syntaxes précises: `http://pandoc.org/MANUAL.html#tables`

Mathématiques

Markdown supporte l'écriture de formules mathématiques avec la syntaxe de latex. Tout ce qui se trouve entouré d'un signe $ est considéré comme une formule mathématique ; et de deux signes $$, comme une équation écrite sur une ligne à part

```
L'erreur-standard est $z\frac{\sigma}{\sqrt{n}}$
```

YAML front matter

YAML front matter

Le YAML front matter est la première partie d'un document Rmarkdown. Elle est automatiquement générée lorsque l'on crée un nouveau document en employant New File -> Rmarkdown/Rnotebook dans Rstudio. On peut également l'éditer à la main.
Cette partie commence et se termine par une ligne composée seulement de trois tirets. Entre ces deux lignes, le code doit respecter les contraintes du format YAML. On y trouve d'abord des métadonnées sous la forme de couples `clé: valeur`.

```
---
title: "Titre de mon analyse"
author: "Samuel Coavoux"
---
```

YAML front matter

Outre le format clé: valeur, on peut écrire des listes. Le format le plus simple de liste dans YAML est le suivant:

```
author:
- "Pierre Bourdieu"
- "Jean-Claude Passeron"
```

YAML front matter

Le front matter contient également des instructions de formatage pour knitr. La plus importante est l'option `output` qui est nécessaire pour pouvoir compiler le document directement dans rstudio, en cliquant sur knit. Les principales possibilités sont:

- html_document: le résultat est une page HTML, publiable sur le web;
- word_document: le résultat est un document au format docx;
- pdf_document: le résultat est un document pdf, en passant par latex;
- beamer_presentation: le résultat est un fichier de slides au format pdf, en passant par beamer (classe de latex).

YAML front matter

Souvent, ces formats sont eux-mêmes suivis d'options spécifiques, qui varient d'un format à l'autre. Elles se définissent dans une liste après le format.

```
---
output:
  pdf_document:
    toc: true
---
```

YAML front matter

On trouve enfin des options globales, qui s'appliquent quel que soit le format de sortie. Les plus importantes sont celles qui concernent la bibliographie employée:

```
---
bibliography: "/chemin/vers/fichier/biblio.bib"
csl: /chemin/vers/fichier/style/bibliographique
---
```

Adapter les résultats pour publication

Rmarkdown et résultats formattés

On peut employer Rmarkdown avec les mêmes packages que l'on emploie avec des scripts. Sans personnalisation, les résultats sont un peu bruts: chaque ligne de résultat commence par ##, les résultats sont présentés dans une fixed-font width, et sont exactement ceux de la console R.
On peut au contraire utiliser Rmarkdown pour produire des résultats publiables. Dans ce cas, la plupart du temps, le code sera caché (avec `echo=FALSE`), et l'on veillera à ce que chaque chunk produise un retour correctement formatté.

Graphiques

Pour les graphiques, aucune personnalisation n'est nécessaire. Les graphiques produits dans un chunk prendront sa place dans le document.
Il est cependant conseillé de ne produire qu'un seul graphique par chunk, pour simplifier la mise en page. Il existe des fonctions pour combiner plusieurs graphiques en un si nécessaire (en base-R, cf. l'argument `mfrow` de la fonction `par()`; avec ggplot, cf. `grid.arrange()` dans le package `grid.extra`).
L'option de chunk `fig.cap` permet de donner un titre au graphique produit. (`fig.cap = "Titre du graphique"`).

Tableaux

Pour les tableaux, l'opération est un peu plus compliquée que pour les graphiques. Par défaut, les tableaux (objets de classe table ou data.frame) ne sont pas très faciles à lire. On utilise habituellement un package qui transforme ces tableaux au format R en des objets au format markdown, html, ou latex (selon le type de document que l'on produit).
Le point commun de tous ces packages est qu'il nécessite l'option de chunk `results = "asis"`. Habituellement, on met cette option par défaut dans le premier chunk avec `opts_chunk$set(results = "asis)`.

Présenter des modèles: stargazer

Il existe de nombreux packages pour présenter des tableaux. Pour les résultats de modèles de régression, le plus souvent utilisé est `stargazer`.
`stargazer(modele)` s'adapte au type de modèle (selon la fonction par laquelle il a été construit). `?stargazer_models` liste les modèles compatibles.
L'argument `type` permet de contrôler si la sortie doit être "latex" (par défaut), "html" (lorsque l'on produit un document html) ou "text" (utile lorsque l'on utilise stargazer dans un script plutôt que dans un document Rmarkdown).

Présenter un modèle linéaire: stargazer

Avec un modèle linéaire, stargazer renvoie par défaut un tableau contenant les coefficients, leur erreur standard, le résultat d'un test de nullité sous la forme d'étoiles, le nombre d'observations, le R^2, le R^2 ajusté. Cf. les options de stargazer pour personnaliser cette sortie. Habituellement, ça n'est pas nécessaire.

Présenter un modèle logistique: stargazer

Pour les modèles logit, stargazer renvoie également les coefficients et leur erreur standard, le résultat d'un test de nullité sous la forme d'étoiles, la log-vraisemblance et l'AIC.
Souvent, on souhaitera transformer les coefficients en odds ratio. Il est alors préférable de rapporter l'intervalle de confiance de l'OR.
On peut spécifier les odds.ratio en modifiant l'argument `coef` et l'intervalle de confiance avec `ci = TRUE` et l'argument `ci.custom`.
Dans ce cas, mieux vaut rapporter les p.values à la main également.
On peut utiliser ce code de la slide suivante pour cela.

```
sg_or <- function(mod){
  OR <- lapply(mod, function(x) {
    as.data.frame(questionr::odds.ratio(x)))
  }
  re <- list(
    or = lapply(OR, `[[`, "OR"),
    ci = lapply(OR, `[`, 2:3),
    p  = lapply(OR, `[[`, "p")
  )
  return(re)
}
x <- sg_or(mod)
stargazer(mod, type = "text", ci = TRUE, coef = x$or, ci.cu
```

Présenter plusieurs modèles

stargazer accepte également plusieurs modèles en arguments. Dans ce cas, il produit un seul tableau mêlant ces modèles, en confondant les lignes pour lesquels les prédicteurs sont les mêmes.

Personnaliser un tableau produit par stargazer

- column.labels : vecteur caractère avec le nom de chacun des modèles (lorsque plusieurs modèles sont présentés);
- covariate.labels : vecteur caractère spécifiant le nom de chacune des variables (par défaut, le nom du vecteur);
- dep.var.labels : un vecteur caractère avec le nom de la variable modélisée (par défaut, le nom du vecteur);
- model.names : vecteur logique indiquant s'il faut indiquer le type de modèle (OLS, logit, probit, etc.);
- no.space : vecteur logique controlant les saut de lignes entre deux variables (FALSE par défaut).

Présenter un tableau croisé

Pour les autres types de tableau, on peut utiliser une fonction du package `knitr` appellée `kable()`.

```
kable(cprop(table(df$var1, df$var2)),
      cap = "Titre du tableau")
```

Il existe également d'autres packages spécialisés, dont `pander` et `xtable`. `kable` et `pander` produisent des sorties au format markdown, qui ne supportent pas les tables complexes (par exemple, avec des cellules fusionnées entre deux lignes ou deux colonnes). Dans ce cas, on peut écrire ces tables directement en latex avec `xtable`.

Écrire des slides

Articulation scripts/knitr