

Rmarkdown avec conditions pour créer des pdf et html en différentes langues et avec des images

Sébastien Rochette - StatnMap

13 mars, 2017

Table des matières

1	“Knit” avec plusieurs options de rendu	1
1.1	Une liste non exhaustive	2
1.2	Fonctions pour créer un rendu spécifique selon le format html ou pdf	2
2	Choisir une langue pour votre document	4
2.1	Inclure du texte dans des <code>chunks</code> au format ‘asis’	4
2.2	Inclure des sorties R dans les “chunks” conditionnels	5
2.3	Images et chemins sous conditions	6
3	Deux images côte-à-côte et centrées sur la page	7
4	Versions formateur et participants	8
4.1	Créer la feuille de route de la formation	8
4.2	Extraire le code R du fichier rmarkdown	8
5	Essayez vous-même ce script !	8
5.1	Quelques limites	9
6	Astuces supplémentaires	9

 Cette version est la version française.

Retourner sur le site Internet :

StatnMap

Formation et consultance

[ACCUEIL](#) | [BLOG](#) | [FORMATIONS](#) | [CONTACT](#)

Cette page est présentée *ici sur mon site Internet*

1. “Knit” avec plusieurs options de rendu

Mes documents rmarkdown sont créés en différentes langues (français et anglais). Je fais aussi en sorte que le rendu final soit similaire entre une sortie pdf et une sortie html. De cette manière, je peux utiliser la sortie html pour la version électronique de mes formations et proposer une version papier à partir du pdf.

Par ailleurs, j’ai aussi une version (partielle) des codes R pour les étudiants et une (qui marche) pour le formateur. Ces codes R sont produits à partir de la feuille de route rmarkdown qui passe par la commande “purl”.

Cela signifie que j’ai plusieurs options qui sont incluses dans un unique document rmarkdown. Utiliser un unique document me permet d’apporter les modifications et mises à jour de mes formations dans le même fichier. Je n’ai pas besoin de me balader entre différents documents pour m’assurer que les mises à jour ont été apportées partout dans les feuilles de route et les scripts R.

En règle générale, j’utilise un script R dit “*run-all*” qui permet de choisir les différentes options pour

produire les différentes versions html, pdf, script R étudiant et formateur (que l'on peut mettre dans une boucle "for").

1.1. Une liste non exhaustive

Dans cet article, je liste différentes astuces que je dois utiliser pour rendre ces différentes versions possibles ainsi que quelques bonus:

- Utilisation des options des "chunks" pour produire les textes en fonction de la langue choisie,
- Utilisation de fonctions pour formater ou colorer du texte et son arrière-plan, de manière similaire en pdf ou en html,
- Afficher des images dans le texte en fonction de la langue choisie,
 - Afficher deux images côte à côte et centrée sur la page en html et en pdf,
- Colorer l'arrière plan d'une zone de code créée avec un environnement "verbatim",
- Sortir d'un fichier rmarkdown le code R des "chunks" pour vos formations, avec une version différente entre le formateur et les participants.

Je modifierai la liste si je fais face à de nouvelles difficultés ou si je trouve de meilleures manières de faire les choses.

Les fichiers nécessaires à la création de cet article sont [ici sur mon github](#).

```
packageVersion("knitr")
```

```
## [1] '1.15.1'
```

1.2. Fonctions pour créer un rendu spécifique selon le format html ou pdf

Les couleurs de texte ou de fond ne se codent pas de la même manière en html ou en latex. J'ai donc créé des fonctions spécifiques qui reconnaissent le format de sortie et ajoute le code de formatage adéquat:

- `colFmt` est utilisé pour colorer du texte en utilisant les noms des couleurs reconnues en latex et html comme "blue", "red", ...
- `styleFmt` est utilisé en combinaison avec la définition de styles dans un fichier à part.
- `beginStyleFmt` et `endStyleFmt` sont une variante séparée de `styleFmt` qui peut s'utiliser lorsqu'il y a des caractères spéciaux dans le texte (comme underscore) qui peuvent être mal interprétés avec latex.
L'option `type` permet notamment d'utiliser `div` au lieu de `span` pour permettre d'afficher du texte sur plusieurs lignes. De manière similaire avec latex, si `type == "div"` est défini, la commande utilisera une ouverture du format avec `begin` et une fermeture avec `end`.
- `beginVerbatim` et `endVerbatim` sont spécifiques au format "verbatim". Ces fonctions permettent d'encadrer une zone de code avec une couleur de fond, ce qui n'est pas implémenté par défaut avec latex notamment.

```
# Simple color Format functions
colFmt <- function(x, colorname, type = "span") {
  outputFormat <- knitr::pandoc_to()
  if (outputFormat %in% c('latex', 'beamer'))
    paste0("\\textcolor{" , colorname, "}{" , x, "}")
  else if (outputFormat == 'html')
    paste0("<" , type, " class=" , colorname, ">" , x, "</" , type, ">")
  else
    x
}

# Style format function, not specific for textcolor
styleFmt <- function(x, textstyle, type = "span") {
  outputFormat <- knitr::pandoc_to()
  if (outputFormat %in% c('latex', 'beamer'))
```

```

    paste0("\\", textstyle, "{", x, "}")
  else if (outputFormat == 'html')
    paste0("<", type, " class=", textstyle, ">", x, "</", type, ">")
  else
    x
}
# Style format options for longer text output
beginStyleFmt <- function(textstyle, type = "span") {
  outputFormat <- knitr::pandoc_to()
  if (outputFormat %in% c('latex', 'beamer')) {
    if (type == "div") {
      cat("\\begin{", textstyle, "}\n", sep = "")
    } else {
      paste0("\\", textstyle, "{")
    }
  } else if (outputFormat == 'html') {
    if (type == "div") {
      cat("<", type, " class=", textstyle, ">", sep = "")
    } else {
      paste0("<", type, " class=", textstyle, ">")
    }
  } else {
    ""
  }
}
endStyleFmt <- function(textstyle, type = "span") {
  outputFormat <- knitr::pandoc_to()
  if (outputFormat %in% c('latex', 'beamer')) {
    if (type == "div") {
      cat("\n\\end{", textstyle, "}", sep = "")
    } else {
      paste0("}")
    }
  } else if (outputFormat == 'html') {
    if (type == "div") {
      cat("</", type, ">", sep = "")
    } else {
      paste0("</", type, ">")
    }
  } else {
    ""
  }
}
# Specific to verbatim environment
# to be able to show you the 'asis' chunks with background color.
beginVerbatim <- function() {
  outputFormat <- knitr::pandoc_to()
  if (outputFormat %in% c('latex', 'beamer')) {
    cat("\\begin{blueShaded}\n\\begin{verbatim}\n")
  } else if (outputFormat == 'html') {
    cat("<div class='blueShaded'>\n```\n", sep = "")
  } else {
    ""
  }
}
endVerbatim <- function() {
  outputFormat <- knitr::pandoc_to()

```

```

if (outputFormat %in% c('latex', 'beamer')) {
  cat("\\end{verbatim}\\end{blueShaded}")
} else if (outputFormat == 'html') {
  cat("\n```\n</div>", sep = "")
} else {
  ""
}
}
}

```

Pour définir les différents styles LaTeX nécessaires au bon fonctionnement des fonctions ci-dessus, vous devez créer un fichier LaTeX qui doit être appelé dans l'en-tête YAML de votre fichier Rmd de cette façon: `in_header: header.tex`. Ce fichier doit inclure la définition de toutes les couleurs et format que vous souhaitez utiliser avec les fonctions précédentes. Par exemple ici :

```

\usepackage{color}
\definecolor{blueSeb}{RGB}{30,115,190}
\definecolor{advertcolor}{HTML}{FF8929}

\definecolor{backcolor}{RGB}{235, 235, 235}
\newcommand{\mybox}[1]{\par\noindent\colorbox{backcolor}
  {\parbox{\dimexpr\textwidth-2\fbboxsep\relax}{#1}}}

\newcommand{\advert}[1]{\textit{\advertcolor{#1}}}
\newcommand{\codecommand}[1]{\texttt{\colorbox{backcolor}{#1}}}

```

De même, pour le rendu html, les fonctions doivent être utilisées avec un fichier d'accompagnement css, déclaré dans l'en-tête YAML de votre fichier Rmd de cette façon: `css: style.css`. Par exemple ici :

```

.advert {
  color: #FF8929;
  font-style: italic;
}
.codecommand {
  background-color:#E0E0E0;
  font-family: Courier New, Courier, monospace;
}
.Shaded {
  background-color:#E0E0E0;
  font-family: Courier New, Courier, monospace;
}

```

2. Choisir une langue pour votre document

Si vous voulez produire vos document rmarkdown dans différentes langues sans utiliser différents fichiers, voici mes astuces.

```

```{r R-Lang, echo=TRUE, eval=FALSE}
Choose the language at the beginning of your script or knit from external file
lang <- c("EN", "FR")[1]
```

```

2.1. Inclure du texte dans des chunks au format 'asis'

Dans les chunks au format 'asis', avec l'option `echo=TRUE`, il est possible d'écrire votre texte en utilisant la syntaxe rmarkdown. Cela produit (partiellement) le format correct comme si vous n'étiez pas dans un chunk. L'intérêt ici étant de pouvoir utiliser la commande `eval` pour afficher ou non le texte selon la langue désirée.

```

<!-- english -->
```{asis, eval=(lang == "EN"), echo=TRUE}
For titles and subtitles when at the beginning
This above chunk allow for markdown syntax

And for titles in the middle if there is space line above
And you can use it for normal text or _italic_ or **other**
```

<!-- french -->
```{asis, eval=(lang == "FR"), echo=TRUE}
Pour les titres et sous-titres au début du "chunk"
Le "chunk" ci-dessus permet d'utiliser la syntaxe markdown

Et les titres en milieu de "chunk" s'il y a une ligne vide avant
Et vous pouvez utiliser le texte normal, _italic_ ou **autre**
```

```

2.1.1 Pour les titres et sous-titres au début du “chunk”

Le “chunk” ci-dessus permet d’utiliser la syntaxe markdown

2.1.2 Et les titres en milieu de “chunk” s’il y a une ligne vide avant

Et vous pouvez utiliser le texte normal, *italic* ou **autre**

2.2. Inclure des sorties R dans les “chunks” conditionnels

```

```{r CodeInChunkFR, echo=(lang == "FR"), eval=(lang == "FR"), results='asis'}
cat(
'### Exemple de titre dans un "chunk"
Exemple de sous-titre dans un "chunk"
- ', styleFmt("cat", "codecommand"), ' doit être utilisé avec l\'option ',
styleFmt("sep = \"\\\"\", \"codecommand\"), ' sinon un espace sera ajouté
devant les dièses des "## sous-titres", ce qui empêche markdown de les
reconnaître comme tel.
- ', styleFmt("eval", "codecommand"), ' peut être utilisé avec un objet R
pour spécifier si le texte doit être affiché, selon le choix de la langue.
Ici here ', styleFmt("lang", "codecommand"), ' est défini comme ', lang, '\n',
'- Les doubles espaces fonctionnent normalement pour produire un nouveau
paragraphe, mais lorsqu\'on utilise une fonction R dans le code, il faut
parfois utiliser "\\\"\\n".
_(Notez le nombre de backslash que j\'ai dû utiliser pour afficher "\\\"\\n"
correctement...)_.
+ ', styleFmt("Par exemple, si j\'utilise ma fonction de formatage,
les espaces suivants ne fonctionnent pas:", "advert"),
' ',
' + Cette phrase ne sera pas sur une nouvelle ligne.',
sep = "")
```

```

2.2.1 Exemple de titre dans un “chunk”

2.2.1.1 Exemple de sous-titre dans un “chunk”

- `cat` doit être utilisé avec l'option `sep = ""` sinon un espace sera ajouté devant les dièses des “## sous-titres”, ce qui empêche markdown de les reconnaître comme tel.
- `eval` peut être utilisé avec un objet R pour spécifier si le texte doit être affiché, selon le choix de la langue. Ici here `lang` est défini comme FR
- Les doubles espaces fonctionnent normalement pour produire un nouveau paragraphe, mais lorsqu'on utilise une fonction R dans le code, il faut parfois utiliser “\n”. (*Notez le nombre de backslash que j'ai dû utiliser pour afficher “\n” correctement...*).
- *Par exemple, si j'utilise ma fonction de formatage, les espaces suivants ne fonctionnent pas:* +
Cetle phrase ne sera pas sur une nouvelle ligne.

2.3. Images et chemins sous conditions

2.3.1 Quand vos images sont dans un dossier défini et s'affichent dans un paragraphe différent du texte

```
```{asis, eval=(lang == "EN")}
 Directory for english version
```


```{asis, eval=(lang == "FR")}
 Fichier pour la version française
```
```

2.3.2 Quand vos images sont dans un dossier avec condition de langue et sont affichées en ligne dans le texte

Seul le code du dernier point est affiché ici mais vous pourrez retrouver le code des autres dans le fichier original rmarkdown qui a servi à créer cette page.

```
figWD <- paste0("./myfigdirectory", lang)
```



- Image sans condition
-  Pas de condition mais permet de changer la taille de l'image (si `out.width` ne fait pas partie des options générales de knitr définie au début de votre script).

```
- `r
  if (lang == "EN") {
    knitr::include_graphics(paste0(figWD, '/gb.png'), dpi = 400)
  } else if (lang == "FR") {
    knitr::include_graphics(paste0(figWD, '/fr.png'), dpi = 400)
  }
`r
  if (lang == "EN") {
    "Conditionnal image: English flag shown if 'lang' is 'EN'"
  } else if (lang == "FR") {
    "Image avec condition : Le drapeau français est affiché si 'lang' est 'FR'"
  }
`r
```

-  Image avec condition : Le drapeau français est affiché si ‘lang’ est ‘FR’

3. Deux images côte-à-côte et centrées sur la page

Le problème lorsqu'on veut afficher deux images côte-à-côte avec `"knitr::include_graphics"` à partir d'un même "chunk", c'est que l'option `fig.align = 'center'` s'applique séparément aux deux images, ce qui les fait apparaître l'une sous l'autre, dans deux paragraphes séparés:



Ici, j'utilise deux fonctions à placer avant et après le "chunk" avec `results='asis'` et un espace entre les images :

```
beginCentering <- function() {  
  outputFormat <- knitr::pandoc_to()  
  if (outputFormat %in% c('latex', 'beamer'))  
    cat("\n\\begin{center}\\n", sep = "")  
  else if (outputFormat == 'html')  
    cat('\n<p style="text-align: center">\n', sep = "")  
  else  
    ""  
}  
endCentering <- function() {  
  outputFormat <- knitr::pandoc_to()  
  if (outputFormat %in% c('latex', 'beamer'))  
    cat("\n\\end{center}\\n", sep = "")  
  else if (outputFormat == 'html')  
    cat("\n</p>\n", sep = "")  
  else  
    ""  
}
```

Le "chunk" s'écrit donc ainsi :

```
```{r, echo=FALSE, out.width='25%', fig.align='default', results='asis'}  
beginCentering()
knitr::include_graphics(paste0(figWD, "/fr.png"))
cat("\vspace{1cm}")
cat(" ")
knitr::include_graphics(paste0(figWD, "/gb.png"))
endCentering()
```
```



4. Versions formateur et participants

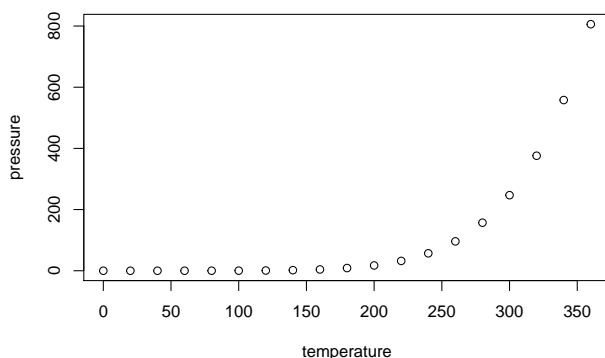
4.1. Créer la feuille de route de la formation

C'est assez simple. Vous devez seulement définir la bonne option dans la commande `pur1` selon que le “chunk” est pour la version formateur ou participant. Personnellement, j'utilise `eval=TRUE` avec la version formateur qui permet de montrer aux participants les figures que les étudiants doivent cibler. Et j'utilise `eval=FALSE` pour la version étudiante qui n'est pas censée fonctionner.

4.1.1 Exemple d'exercice

Reproduisez la figure suivante à partir du jeu de données “cars”.

```
##      speed      dist
## Min.   : 4.0    Min.   :  2
## 1st Qu.:12.0    1st Qu.: 26
## Median :15.0    Median : 36
## Mean   :15.4    Mean   : 43
## 3rd Qu.:19.0    3rd Qu.: 56
## Max.   :25.0    Max.   :120
```



4.2. Extraire le code R du fichier rmarkdown

Après extraction du code R de ce fichier rmarkdown avec l'option “teacher” et la commande `pur1` vous obtiendrez le script R suivant :

```
# A script for teacher with solutions -----
summary(cars)
# Plot pressure against temperature
plot(pressure)
```

Et ce script R si l'extraction se fait avec l'option “student” :

```
# A script for students without solutions -----
summary(cars)
# Plot pressure agains temperature
plot(...)
```

5. Essayez vous-même ce script !

Tous les fichiers nécessaire à la création de cette page son disponibles sur [mon github](#). Toutes les sorties en français et en anglais, en pdf et en html, ainsi que les scripts R formateur et participants sont aussi dans le dossier. Utilisez le fichier *"Run_all_and_choose_options.R"* pour reproduire tous les fichiers par vous-même.

5.1. Quelques limites

Une limite à l'utilisation des balises “\begin” et “\end” avec `lateX` est que `pandoc` ne convertit pas la syntaxe `markdown` incluse dans cet environnement. Elle est considérée comme du code verbatim. Peut-être que je trouverai un moyen de passer outre... Par ailleurs, une des limites de ce type de script multilingue `rmarkdown`, est que la coloration syntaxique dans `Rstudio` n'est plus très utile... Le script utilisé pour générer cette page peut être difficile à lire, en particulier parce que j'ai voulu vous afficher les “chunks” utilisés en entier. De plus, les chunks “asis” ne permettent pas d'utiliser l'option “verbatim”, ce qui m'a obligé à ajouter une couche de code ici.

Si vous avez des meilleures manières de produire le code de cette page, vous pouvez apporter vos commentaires ici ou sur [github](#).

Cela dit, si vos fichiers `rmarkdown` commencent à être trop volumineux, n'hésitez pas à créer des fichiers enfants...

6. Astuces supplémentaires

Le template `rmarkdown` par défaut pour `lateX` charge un certain nombre de packages et définit un certain nombre d'options pour la mise en forme du texte. Si vous ne voulez pas recréer vous-même un template, vous pouvez définir, comme ici, un fichier `tex` à charger dans l'en-tête avec l'option `YAML` “`in_header`”. Toutefois, vous pourrez rencontrer quelques messages d'erreur dus à des incompatibilités d'options. Dans le fichier “`header_tips.tex`” que vous trouverez sur le [github](#), vous trouverez différentes modifications du template pour `pdf` :

- Trouver quelle langue est chargée par `R` avec le package `babel` et définir de nouvelles commandes en fonction de la langue choisie. C'est particulièrement utile dans le cadre de ce template multilingue qui utilise l'option “`lang`” dans l'en-tête `YAML` que j'ai paramétré ici en fonction de la langue.
- Modifier le format des titres de sections pour être colorés et définis comme vous le souhaitez, en utilisant le package “`titlesec`” qui nécessite d'annuler temporairement les options proposées par `rmarkdown`.
- Modifier la couleur d'arrière-plan des zones de code verbatim

Plus d'astuces en lien avec [R](#), les modèles et les trucs spatialisés sur mon site Internet

Cette page est présentée [ici](#) sur mon site Internet

StatnMap

Formation et consultation