Tutorial 16

Código e análise de dados reproduzívies

Como sabemos que o seu código não contém um erro? Como podemos provar que não manipulamos a análise de um paper? Como contribuimos para um projeto comparthilado? A reprodutibilidade de nossos scripts (e os outputs que eles geram) é central para programação e pesquisa. Nesse tutorial, vamos explorar várias ferramentas para apoiar a reprodutibilidade.

1. **Projetos em R** - pastas para organizar os nossos inputs, scripts e outputs
2. **R Markdown** - um outro tipo de script que transforma seu código de R em documentos finais (PDF, Word, HTML)

Projetos em R

Até agora, trabalhamos com arquivos de R individuais. Uma desvantagem disso é que temos que especificar referências longas para os arquivos de inputs e outputs. Isso é chato, porque é fácil cometer um erro e fácil perder seus arquivos. Além disso, quando você manda seu código para outra pessoa, ele não é reproduzível porque a estrutura de pasta deles será diferente da sua.

Projetos em R ajuda resolver este problema. Um projeto em R é simplesmente uma pasta contendo vários arquivos: um arquivo genérico de ".Rproj" que o Rstudio abre, todos os nossos scripts, os nossos arquivos de input, e os nossos outputs. Crucialmente, quando quisemos abrir um ".csv" dentro dessa pasta, não precisamos especificar a referência completa (endereço 'absoluto') do arquivo, apenas o nome do arquivo (endereço 'relativo'). Em termos técnicos, o diretório de trabalho é automaticamente a pasta do projeto atual. Por exemplo podemos substituir:

```
pasta <- "C:\\User\\Name\\Projects\\New\\Political Science\\2018\\Examples\\"
path_file1 <- file.path(pasta, "file1.csv")
dados <- read_csv(path_file1)
```

por:

```
dados <- read_csv("file1.csv")
```

A outra vantagem de trabalhar em projetos é que eles facilitam o uso do controle de versão de Git dentro do Rstudio (vamos discutir isso na próxima aula).

Para abrir um projeto, simplesmente escolhe File -> New Project -> New Directory -> New Project e digite o nome desejado da pasta do projeto. Dado que um projeto é apenas uma pasta, em seguida precisamos abrir um script novo e salvá-lo na mesma pasta. Note que no canto superior direito do RStudio, tem um menu de opções com o seu projeto aberto.

Produzindo 'outputs' profissionais com o R Markdown

Agora, sabemos como a executar um análise de dados na linguagem de R. Mas o nosso objetivo geralmente não é apenas executar algum código, mas produzir uma tabela, um gráfico, um relatório, um documento ou um site. Como podemos comunicar o nosso análise numa forma profissional, documentada e reproduzivel? Vamos usar o R Markdown: um tipo de arquivo compatível com o RStudio e uma linguagem bem fácil e flexível para formatar documentos.

Um script em R tem a extensão de arquivo ".R"; um arquivo de R Markdown tem uma extensão de arquivo ".Rmd". Com o mesmo arquivo ".Rmd" podemos criar três tipos de documentos sem mudar nada: um PDF,

um Documento de Word e um site de HTML. Então podemos compartilhar os nossos resultados com tudo mundo em formatos diversos, evitando a duplicação de códigos e arquivos.

Só uma dica: Todos podem criar outputs em HTML e Word. Para criar PDFs precisa instalar LATEX for Windows Complete.

Na verdade, este tutorial não foi feito por mágica, mas por R Markdown!

Abrindo um arquivo de R Markdown

Em geral, é melhor começar abrindo um 'projeto' em R (veja a discussão anterior). Depois, abrimos um arquivo de R Markdown: Clique File -> New File -> R Markdown. Escolhe 'documento' na esquerda, digite um título e autor, e escolhe o formato *default* para o output final (podemos mudar em qualquer momento). Salve o novo script.

O conteúdo default contém um exemplo, mas pode apagar tudo exceto o 'header', que parece assim:

```
---
title: "Example Document"
author: "Jonathan"
date: "April 19, 2018"
output: pdf_document
---
```

Misturando código, outputs e documentação/explicação

Trabalhamos com vários tipos de conteúdo dia-a-dia e queremos integrar todos num documento sem confusão: texto básico, código de análise, tabelas, gráficos e equações. Em R Markdown, separamos eles em dois lugares dentro de nosso arquivo ".Rmd" : (i) no corpo do script, onde pode digitar como em Microsoft Word, e (ii) num 'chunk' de código, onde pode digitar código como num script ".R".

Digite no corpo	Digite num 'chunk'
Texto explicativo	Código de R
Equações (na sintaxe do Latex)	Tabelas criados com código de R
Gráficos criados com código de R	

Para inserir um chunk, clique: Insert -> R. Aparece uma caixa cinza onde podemos colocar o nosso código:

```
```r
x <- 1 + 1
```
```

Você pode incluir todas as funções que você conhece nesses chunks. Qualquer objeto criado está disponível para chunks debaixo, mas lembre-se que chunks serão executados na ordem em que aparecem no documento.

Produzindo um documento final

Para transformar o nosso script ".Rmd" para um PDF, clique no botão "Knit". Feito. O documento é salvo na pasta de trabalho (do projeto atual). Para criar um outro formato de documento, clique a flecha ao lado de "Knit" e eschole o formato preferido.

O RStudio abre uma nova janela "R Markdown" que mostra o progresso e descreve os erros.

Formatação de texto básico

Em contraste de Microsoft Word, formatando texto em R Markdown não depende de clicar nos botões - precisamos incluir caracteres no script para indicar a formatação que queremos.

Italic - `*Italic*`
Bold - `**Bold**`

Big Header

```
# Big Header
```

Sub-Header

```
## Sub-Header
```

Sub-Sub-Header

```
### Sub-Sub-Header
```

link - `[link](http://www.google.com)`

- Bullets
  - Sub-Bullets

```
* Bullets
[quatro espaços]    * Sub-Bullets
```

1. Numbered List
    1. Sub-numbered list

```
1. Numbered List
[quatro espaços]    1. Sub-Numbered List
```

Para mais detalhes veja O Cheatsheet.

Equações

Podemos escrever equações com a linguagem de Latex (que vamos aprender a semana que vem). Dentro de uma frase, use `$ equation $`, e para centralizar numa nova linha use `$$ big equation $$`. A sintaxe é assim:

```
$$\alpha^2 + \beta^2 = \chi^2$$
```

$$\alpha^2 + \beta^2 = \chi^2$$

```
$$\frac{\sqrt{1}}{2} * \frac{a}{2b} = \frac{a}{4b}$$
```

$$\frac{\sqrt{1}}{2} * \frac{a}{2b} = \frac{a}{4b}$$

```
$$\sum_0^{10} x = ...$$
```

$$\sum_0^{10} x = ...$$

Mais detalhes aqui.

Exercício

Começando com um novo projeto e um novo arquivo markdown, crie um PDF (ou HTML se não tem Latex) que inclui apenas texto (sem chunks de código) com ao menos cinco das formatações acima e a famosa equação de Einstein.

'In-line' Code: Incluindo valores simples do código

Podemos fazer um análise num chunk de código e atribuir/salvar o resultado como um objeto de R. Por exemplo, calculamos a área de um círculo de raio 20 com o chunk debaixo:

````
```r
raio <- 20
area <- pi * raio^2
```
````

Como incluimos o resultado numérico `area` dentro de uma frase em nosso texto (for do chunk do código)? Podemos usar "in-line code" para exibir um valor simples no documento final. O formato é assim, usando o acento grave:

A área de um círculo de raio 20 é `` `r area` ``.

A área de um círculo de raio 20 é 1256.6370614.

Exercício

O volume de uma esfera é $A = \frac{4}{3}\pi r^3$. Use um chunk para calcular a área de uma esfera de raio 7 e relate o resultado numérico dentro ('in-line') de uma frase explicativa. Crie um documento PDF (ou HTML se não tem Latex instalado).

Tabelas

Qual é a diferença entre uma tabela num documento, e um data.frame (/tibble) em R? Por nós, nada exceto formatação! Então podemos facilmente transformar nossos data.frames em tabelas para exibir para os leitores. Podemos criar um data.frame usando as nossas ferramentas existentes dentro de um chunk. Depois, temos várias funções que nos ajudam a formatar essas tabelas. Para tabelas simples, podemos usar a função `kable()` (e para tabelas de regressão podemos usar `stargazer()`). Note que não fazemos nada fora do chunk - o R Markdown exibe o código **e** o resultado da execução do código assim que não atribuamos / salve o resultado.

Vamos abrir um banco de dados simples, manipular a estrutura, e mandar para `kable()` para criar uma tabela.

```
library(tidyverse)
```

```
## Registered S3 methods overwritten by 'ggplot2':
##   method         from
##   [.quosures     rlang
##   c.quosures     rlang
##   print.quosures rlang
```

```
## Registered S3 method overwritten by 'rvest':
##   method            from
##   read_xml.response xml2
```

```
## -- Attaching packages ---------------------------------------------------------------- tidyverse 1.
```

```
## v ggplot2 3.1.1       v purrr   0.3.2
## v tibble  2.1.1       v dplyr   0.8.0.1
## v tidyr   0.8.3       v stringr 1.4.0
## v readr   1.3.1       v forcats 0.4.0
```

```
## -- Conflicts ------------------------------------------------------------------------ tidyverse_conflicts
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(knitr)

file1 <- "https://raw.githubusercontent.com/leobarone/ifch_intro_r/master/data/bf_amostra_hv.csv"
dados <- read.table(file1, header = T, sep = ",")

dados %>% group_by(uf) %>%
  summarize(mean_valor=mean(valor)) %>%
  kable()
```

uf	mean_valor
AC	170.0000
AL	182.0000
AM	172.0000
BA	129.4000
CE	193.0000
ES	124.0000
GO	209.0000
MA	177.5000
MG	222.6667
MT	85.0000
PA	169.3333
PB	191.2500
PE	166.0000
PI	167.0000
PR	511.0000
RJ	145.0000
RN	215.3333
RS	124.0000
SP	143.5000
TO	124.0000

Podemos especificar os parâmetros de `kable()` para controlar a formatação do resultado. Por exemplo:

```
dados %>% group_by(uf) %>%
  summarize(mean_valor=mean(valor)) %>%
  kable(caption="A minha tabela",align="cr",digits=1,col.names=c("Estado","Valor Media"), format.args=li
```

Table 3: A minha tabela

Estado	Valor Media
AC	170.0
AL	182.0
AM	172.0
BA	129.4
CE	193.0

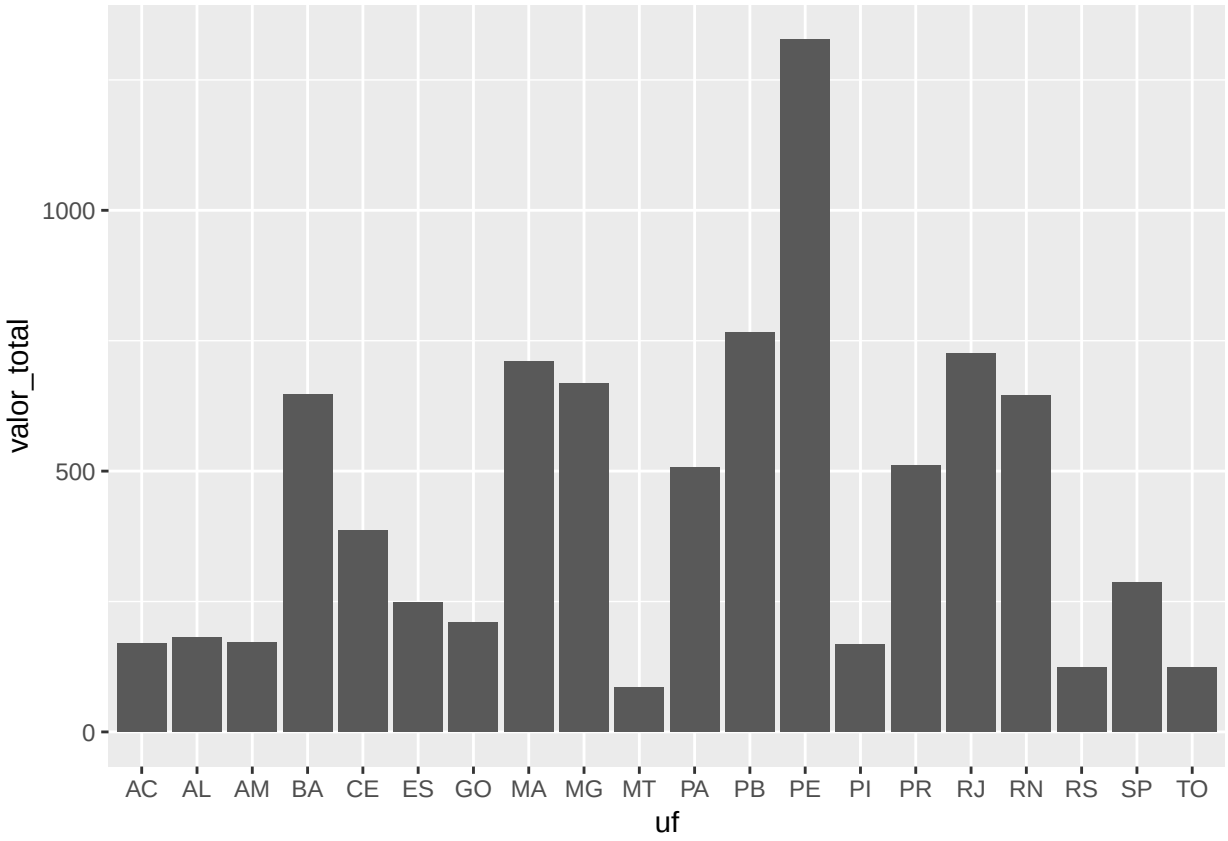
Estado	Valor Media
ES	124.0
GO	209.0
MA	177.5
MG	222.7
MT	85.0
PA	169.3
PB	191.2
PE	166.0
PI	167.0
PR	511.0
RJ	145.0
RN	215.3
RS	124.0
SP	143.5
TO	124.0

Note: Para gerar tabelas em PDF (com Latex) temos que inserir a opção

Gráficos

Os gráficos seguem a mesma lógica que usamos para tabelas. Criamos o nosso gráfico e manda para um `plot/ggplot`.

```
dados %>% group_by(uf) %>%
  summarize(valor_total=sum(valor)) %>%
  ggplot() +
  geom_col(aes(x=uf,y=valor_total))
```



Exercício

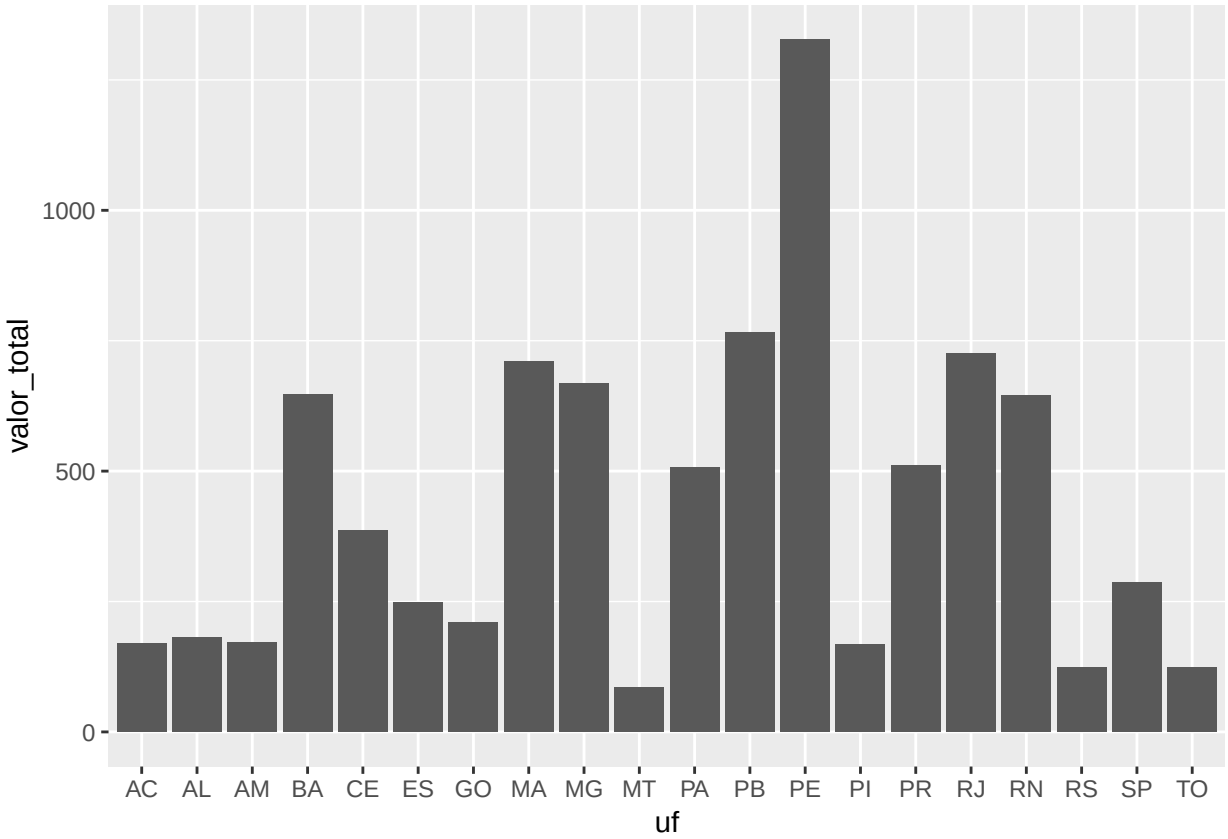
Usando o link acima para os dados de Bolsa família, criar uma tabela dos cinco maiores valores recebidos, formate-lo com um título, nomes de colunas e o número apropriado de dígitos, e 'Knit'.

Parâmetros de Chunks

Agora você pode produzir quase todo tipo de conteúdo. Mas agora estamos provavalmente produzindo *demais* conteúdo para nosso documento - por exemplo, o chunk acima produziu linhas de código e um gráfico. Para tutoriais, podemos querer incluir o código no documento final (como aqui), mas, para os relatórios, queremos focar no gráfico. Como controlamos quais tipos de conteúdo são traduzidos de chunks para o documento final? Usamos parâmetros no começo de cada chunk.

Por exemplo, `echo=F` vai prevenir o código bruto aparecendo no documento final:

```
```{r, echo=F}
dados %>% group_by(uf) %>%
  summarize(valor_total=sum(valor)) %>%
  ggplot() +
  geom_col(aes(x=uf,y=valor_total))
```
```



Pode usar `eval=F` para prevenir a execução do código, por exemplo se você quiser criar um tutorial.

Também, normalmente queremos definir `warning=F` e `message=F` para prevenir a exibição de mensagems de pacotes.

Mais um parâmetro útil: às vezes nosso código dura muito tempo para executar e podemos gravar os objetos criados dentro de um chunk se especificamos `cache=TRUE`. Mas tome cuidado que todo o seu código é atualizada se muda uma linha acima de chunk com cache.

Dado que queremos especificar estas opções para cada chunk, podemos usar um 'shortcut' para definir os defaults. Num chunk preliminar, colocamos por exemplo:

```
```{r, echo=FALSE}
knitr::opts_chunk$set(echo = F, warning=F, message=F)
```
```

Exercício

Vamos limpar e organizar o script com quem você está trabalhando para criar um documento PDF (ou HTML se não tem Latex instalado) profissional. Especifique parâmetros de cada chunk para tirar todos os mensagens, warnings e código bruto do documento final. O seu PDF deve conter apenas o seu texto explicativo, 'in-line' código, e tabelas/gráficos.