Um curso intensivo de pesquisa reprodutível em R

October 14, 2016 in r, programming tips

Fonte: http://t-redactyl.io/blog/2016/10/a-crash-course-in-reproducible-research-in-r.html

Tempos atrás, escrevi em um post uma introdução sobre pesquisas reproduzíveis *em Python* (veja aqui: http://t-redactyl.io/blog/2016/10/a-crash-course-in-reproducible-research-in-python.html). Embora os princípios de reprodutibilidade permaneçam os mesmos, não importa a linguagem que se esteja usando, existem algumas bibliotecas e ferramentas específicas que o R tem que são diferentes do Python. Neste post, mostrarei como conduzir uma análise reprodutível em R e, assim como fiz com o Python, pode-se observar que em R também é simples!

Recapitulando: o que é pesquisa reprodutível?

Há um tempo atrás, tive a sorte de me apresentar na PyCon Australia, bem aqui na minha cidade natal de Melbourne. Minha palestra foi sobre pesquisa reprodutível, um conceito cada vez mais importante em ciência e análise de dados à medida que projetos se tornam mais complexos e os datasets se tornam maiores (veja neste artigo, por exemplo, como um número surpreendentemente alto de documentos genéticos contém erros por conta da forma em que o Excel interpreta os nomes dos genes!).

Devo confessar que reprodutibilidade não mereceu atenção de minha parte durante a maior parte da minha carreira acadêmica. Para ser honesto, quando comecei a fazer programação estatística, o básico era tão volumoso que a ideia de o tornar reproduzível parecia muito além do que eu conseguiria fazer! Eu também me sentia um pouco sobrecarregado por todas as coisas que eu precisava aprender quando comecei a tentar conduzir pesquisas reproduzíveis. Parecia haver tantos detalhes na web sobre Git, Github, Python, R, RStudio, etc. que realmente eu não sabia por onde começar.

Com o tempo, percebi que o básico é realmente muito simples e exige apenas um bom conhecimento introdutório das ferramentas que você usará. Neste post, tentei desmistificar o processo e a implementação de pesquisas reproduzíveis, além de mostrar o quanto facilita sua vida tornar os projetos colaborativos ou até mesmo retomar às suas análises meses depois!

O que torna uma análise reproduzível?

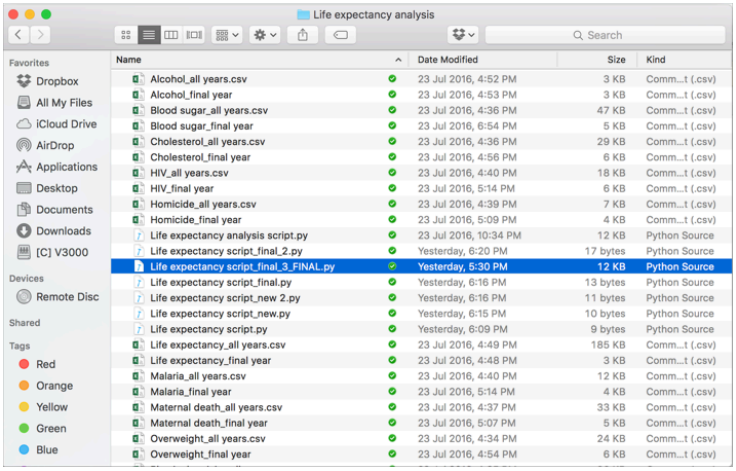
Encontrei uma pequena citação no *CRAN-R Project* que define o propósito da pesquisa reproduzível desta forma:

> *O objetivo da pesquisa reproduzível é vincular instruções específicas à análise de dados e dados experimentais para que os estudos possam ser recriados,* melhor compreendidos e verificados.

Esta é uma ótima visão geral de alto nível sobre *reprodutibilidade*, mas é difícil entender como isso se relaciona com o nosso trabalho diário. Em vez disso, pode ser útil começar com um exemplo de uma análise *irreprodutível*.

Uma *análise irreprodutível*, em minha opinião, é qualquer análise que você ou outra pessoa tenha dificuldades significativas para assumir e continuar

trabalhando. Imaginemos que fiz uma análise há seis meses, onde usei dados da OMS para prever a expectativa de vida média em cerca de 150 países. Eu abro a pasta Dropbox com todos os meus arquivos:



Não tenho ideia por onde começar. Não apenas existem várias versões do script de análise (sem um indicador claro para o arquivo final), como também há mais de 30 conjuntos de dados diferentes. Quando eu abro o que parece ser a versão mais recente do script de análise (como visto aqui), não há nenhuma indicação real sobre o duais foram a análise final e os resultado, o que torna realmente difícil continuar de onde parei.

Buscando uma nova forma para o desenvolvimento de projetos compartilhados e consistentes, e para que isso seja possível, é preciso responder a 5 perguntas sobre seu projeto:

1. *O que fiz*?
2. *Por que fiz*?
3. *Como configurei o ambiente no momento da análise*?
4. *Quando fiz alterações no projeto e quais foram elas*?
5. *Quem precisa acessá-lo e como pode faze-lo*?

Como expliquei no post na linguagem Python (e no adendo ao final deste post), utilizo o *Git* e o *Github* para rastrear minhas alterações e compartilhar meus projetos com colaboradores, por isso utilizarei tais ferramentas nesta postagem. A linguagem R possui ferramentas específicas para os pontos de 1 a 3.

O que fiz?

Como eu falei no post de Python, um dos maiores problemas que você enfrentará para se lembrar do que fez em sua análise é se você fez as coisas manualmente. Assim como no Python, o R tem ótimas funcionalidades para baixar dados de fontes on-line e limpar esses dados depois de importados.

É fácil acessar dados estruturados de fontes on-line e importá-los para R como *dataframes*. Abaixo, criei uma função para baixar o arquivo .csv que contém os dados da expectativa de vida sobre os quais falei na última postagem. Como você pode ver, você só precisa usar alguns comandos (os comandos getURL e

textConnection do pacote RCurl). Além de ser fácil, é completamente reproduzível!

```
install.packages("RCurl"); library(RCurl)
dataImport <- function(dataurl) {
  url <- dataurl
  dl <- getURL(dataurl, ssl.verifyhost=FALSE, ssl.verifypeer=FALSE)
  read.csv(textConnection(dl), header=T)
}
life <-
dataImport("http://apps.who.int/gho/athena/data/xmart.csv?target=GH
O/WHOSIS_000001,WHOSIS_000015&profile=crosstable&filter=COUNTRY:*&x
-sideaxis=COUNTRY;YEAR&x-topaxis=GHO;SEX")
```

O R também tem grande funcionalidade para limpeza de *datasets*. Criei uma função curta usando apenas comandos do pacote "*base*" do R para selecionar o subconjunto apropriado de colunas e linhas em nosso novo *dataset* importado, assim como renomear as colunas restantes.

```
cleaningData <- function(data, startrow, columnyear, year,
colsToKeep, columnNames) {
  df <- data[c(startrow:nrow(data)) & data[[columnyear]] == year, ]
  df <- df[ , colsToKeep]
  names(df) <- columnNames
  df
}

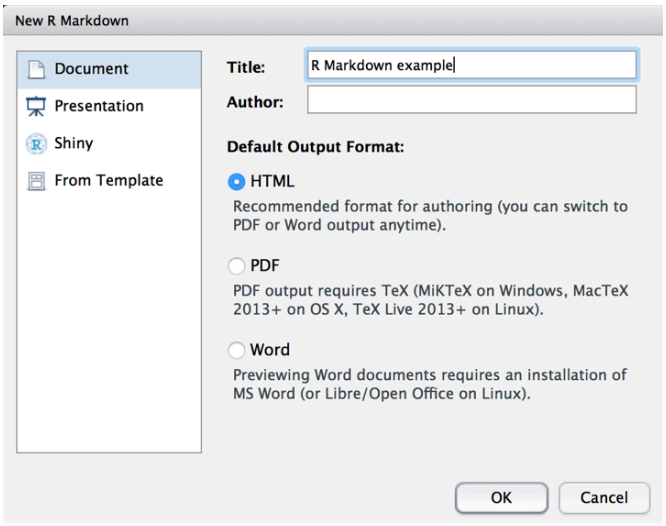
life <- cleaningData(life, 2, "X.1", " 2015", c("X", "X.1",
"Life.expectancy.at.birth..years."),
                     c("Country", "Year", "LifeExpectancy"))
```

Por que fiz?

Como o Python, o R possui seu próprio ferramental para programação estatística literária chamada R Markdown. Assim como nos notebooks Juypter, você pode escrever trechos de texto markdown dentro do código R, o que significa que você pode criar para sua análise, anotações significativas e fáceis de ler. Você também pode incluir resultados, tabelas e gráficos, o que permite criar relatórios e outros documentos a partir de um script R Markdown autocontido. Na verdade, Mauricio e eu escrevemos nosso livro sobre gráficos em ggplot2 usando inteiramente o Markdown!

Como faço para configurar um documento R Markdown?

Os documentos Markdown podem ser criados dentro do RStudio (como grande parte da melhor funcionalidade R!). Para abrir um novo documento R Markdown, basta escolher "R Markdown" como o tipo ao criar um novo arquivo. Você será solicitado a dar um título ao seu documento R Markdown; Eu chamei este de "Exemplo de R Markdown". Em seguida, clique em "OK" para inicializar o novo documento.

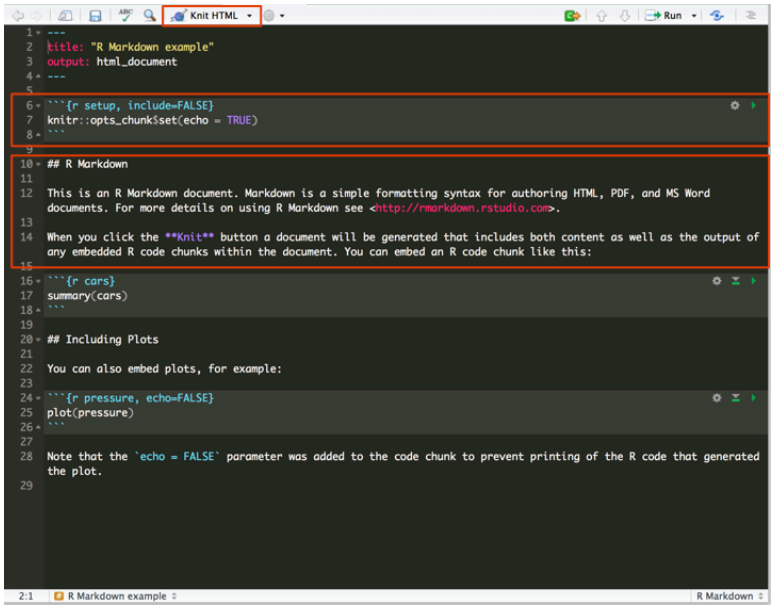


Os documentos R Markdown começam automaticamente com um *modelo*. Como você pode ver, existem dois tipos de código dentro de um documento R Markdown. O código é colocado dentro de blocos (chunks), que são delimitados por backticks (```) e {r}.

```
```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```
```

Há disponíveis muitas opções que permitem personalizar como o código será apresentado e executado dentro dos fragmentos (chunks), mas o padrão é que eles simplesmente executem qualquer código R que estiver dentro deles. Fora dos blocos, qualquer texto escrito é reconhecido como markdown.

Para executar um documento R Markdown, pressione o botão "Knit HTML" na parte superior do documento. Isso renderiza o markdown, executa o código R e libera um documento HTML.



Pode-se exportar documentos R Markdown em vários formatos, usando a função *knit* dentro do pacote *knitr*. Para isso, utilizamos o arquivo Markdown R como entrada (`teste_html.Rmd`), especificamos o nome do arquivo renderizado e o formato desejado como saída. Aqui, o documento de exemplo estou renderizando como um arquivo regular de markdown (`R Markdown example.md`):

```
install.packages("knitr"); library(knitr)
setwd("~/Downloads")
install.packages("knitr"); library(knitr)
knit("~/Downloads/teste_html.Rmd",
     output = "~/Downloads/R Markdown example.md")
```

E é simples assim ! Você pode ver facilmente com é possível integrar a programação estatística literária ao fluxo de trabalho usual do RStudio.

Como configurei o ambiente no momento da análise?

Similar ao Python, os scripts em R podem "*dar erro*" porque está se usando a versão de uma biblioteca errada ou porque duas bibliotecas diferentes instaladas globalmente não são compatíveis juntas. O R tem uma solução semelhante ao *virtualenvs*[1] do Python, chamado **packrat**, que permite acompanhar as dependências de suas análises.

Packrat é uma biblioteca R que permite criar um tipo especial de diretório que funciona de maneira muito semelhante a um *virtualenv*. Enquanto se está no diretório **packrat**, todas as bibliotecas instaladas ficam **isoladas**: *essas bibliotecas só estão disponíveis para seu projeto específico e seu projeto não pode acessar nenhuma biblioteca instalada fora dele*.

As pastas do packrat são capazes de manter o controle de suas dependências devido à presença de um **arquivo de bloqueio**, que simplesmente controla quais bibliotecas você instalou e suas versões (muito parecido com o arquivo de requisitos congelados no *virtualenvs*).

Como faço para configurar um arquivo packrat?

Mostrarei como configurar um diretório packrat como parte de um projeto do RStudio, mas há também um ótimo tutorial do RStudio sobre como fazê-lo independentemente de um projeto.

O primeiro passo é instalar globalmente o packrat na sua máquina.

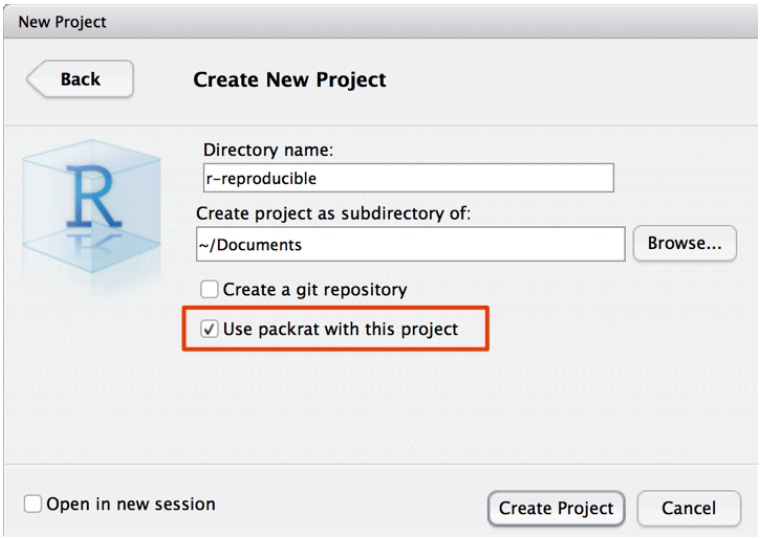
```
install.packages("packrat")
```

---

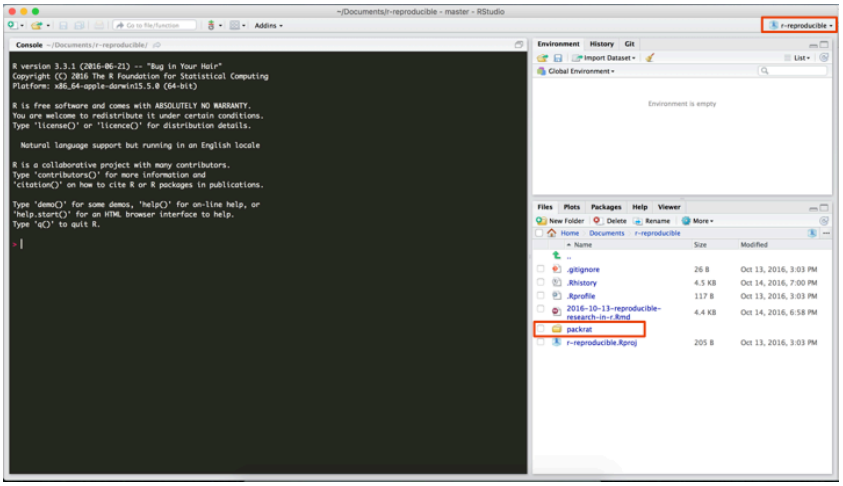
[1] O virtualenv permite que várias versões de uma mesma biblioteca possam conviver no mesmo computador, sem conflitos. Ao contrário do que o nome induz a pensar, um virtualenv não é uma VM (máquina virtual). Fonte: http://blog.aprendapython.com.br/articles/entendendo-virtualenv-1fr69/.

Depois de fazer isso, reinicie o RStudio. Esta deve ser a única vez será necessário fazer isso. Quando voltar ao RStudio, crie um novo projeto do RStudio em um novo diretório (instruções sobre como fazer isso aqui).

Neste ponto, será solicitado um nome ao novo projeto e, o mais importante, indicar se deseja **torná-lo um projeto packrat**. Como se pode ver é tão simples quanto selecionar uma caixa para inicializar o diretório do packrat!



Após criar o novo projeto do RStudio com o packrat, temos uma nova pasta em nosso diretório chamada "packrat". Entre outras coisas, é aqui que reside o arquivo de bloqueio que controla todos os pacotes que você instala.



Instalar bibliotecas é super simples - enquanto estivermos em nosso projeto, podemos usar o comando install.packages e manteremos a instalação limitada a este projeto específico.

Vamos instalar o ggplot2:

```
install.packages("ggplot2"); library(ggplot2)
```

Se você realizou estas operações até aqui, pode ver que, mesmo que tenha a biblioteca ggplot2 instalada, o R ignora isso e faz uma instalação “limpa”. Isso ocorre porque ele não pode acessar a sua versão do ggplot2 instalada globalmente e precisa instalá-la do zero.

Os projetos do RStudio com o packrat são configurados com snapshots automáticos, o que significa que o packrat rastreia todas as alterações que você faz nas dependências do seu projeto sem precisar explicá-las por conta própria.

Finalmente, compartilhar ou retomar ao projeto packrat em uma máquina diferente é realmente fácil. O RStudio faz praticamente todo o trabalho pesado, então tudo que se precisa fazer é instalar o diretório do projeto na nova máquina e abrir o projeto no RStudio. Todas as dependências do projeto serão instaladas automaticamente, deixando tudo pronto para abrir o projeto e iniciar de onde parou.

Na minha opinião, a maneira mais simples de compartilhar um diretório de projeto RStudio é copiando-o (pushing) para um repositório remoto em algo como o Github, e clonando-o em outra máquina quando se precisar trabalhar nele. Isso também tem a vantagem adicional de permitir que você acompanhe as alterações no seu projeto.

Espero que este tutorial tenha mostrado as mudanças necessárias em seu fluxo tradicional de trabalho de análise em R para garantir que o trabalho seja reproduzível. A implementação dessas etapas significa que as pessoas terão prazer em desenvolver seus projetos pelos próximos anos!

Adendo: Como fazer o controle básico de versão com o Git e o Github

As primeiras coisas que se precisa fazer são:
1. Verifique se o Git está instalado na máquina (aqui está um guia para verificar em Mac e outro para verificar em Windows),
2. Criar uma conta do Github, caso não tiver uma e, em seguida,
3. Configurar a autenticação do Github na máquina.

Depois de configurar o Git e o Github, a primeira coisa a fazer é criar um repositório remoto para um novo projeto (veja aqui um excelente guia sobre como fazer isso). Você então precisa clonar uma cópia local deste repositório para o computador, desta forma.
Assim que tivermos nosso repositório local, podemos começar a trabalhar em nosso novo projeto em R. Digamos que criamos um novo arquivo R script! E começamos a trabalhar nele.
A qualquer momento (quanto mais frequente, melhor), podemos fazer um **commit**. Para fazer isso, abra o terminal e navegue até o repositório local para este projeto.
Em seguida, verificamos quais arquivos o Git está **acompanhando** (*tracking*) (ou não - *untracking*) digitando:

```
git status
```

```
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
```

```
  (use "git push" to publish your local commits)
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working
directory)

     [31mmodified:   2015-11-11-understanding-object-orientated-
programming-in-python.ipynb [m
     [31mmodified:   2016-06-01-web-scraping-in-python.ipynb [m
     [31mmodified:   2016-10-04-reproducible-research-in-
python.ipynb [m

Untracked files:
  (use "git add <file>..." to include in what will be committed)

     [31m.DS_Store [m
     [31m.ipynb_checkpoints/ [m
     [31mUntitled.ipynb [m
no changes added to commit (use "git add" and/or "git commit -a")
```

Isso nos informa que temos vários arquivos que já estamos pedindo ao Git para rastrear e que tem algumas alterações não confirmadas (as "Mudanças não testadas para confirmação" - ‘Changes not staged for commit’), bem como uma que não estamos pedindo ao Git para acompanhar (os 'arquivos não rastreados' - ‘Untracked files’). Os arquivos que o Git está rastreando são aqueles que já foram confirmados e que o Git verifica se alguma alteração foi feita desde o último **commit**.
Vamos confirmar nossas alterações mais recentes no arquivo "2016-10-04-reproducible-research-in-python.ipynb".

```
!git add '2016-10-04-reproducible-research-in-python.ipynb'
```

O Git agora colocou esse arquivo em uma fila para ser confirmado, mas precisa que nós confirmemos explicitamente o arquivo com uma mensagem.

```
!git commit -m "Completed the final section of the blog post"

[master 8f544a0] Completed the final section of the blog post
 1 file changed, 5 insertions(+), 6 deletions(-)
```

O que este comando significa é que nós confirmamos (comitted) nossas alterações no repositório local. O que precisamos agora é levar (push) nossas alterações para o repositório remoto. Praticamente toda vez que eu carrego para o repositório remoto, eu estou leevando para o **repo** de origem na *ramificação* **branch master**, o que significa que eu uso o comando abaixo:

```
!git push origin master

Counting objects: 8, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (8/8), done.
Writing objects: 100% (8/8), 9.43 KiB | 0 bytes/s, done.
Total 8 (delta 4), reused 0 (delta 0)
remote: Resolving deltas: 100% (4/4), completed with 1 local
objects. [K
```

```
To https://github.com/t-redactyl/Blog-posts.git
   83df6d4..8f544a0  master -> master
```

No entanto, se você tiver uma estrutura de **repo** mais complicada que contenha ramificações, será necessário adequar sua mensagem de carregar (push) para garantir que você se confirmou (comitted) no local correto (consulte esta postagem para obter mais detalhes).

Isso resume meu guia introdutório sobre como implementar uma pesquisa reproduzível em R - espero que se possa ver como algumas modificações simples seus fluxos de trabalho (e de seus colaboradores), pode garantir que que se está construindo projetos que sejam possíveis de revisar nos próximos anos!