R PARA CIÊNCIA DE DADOS: PACOTES

C. Lente + Curso-R

2018-02-05

Nossa motivação

"Por que vamos aprender a fazer um pacote se scripts funcionam bem o suficiente?" e "Por que divulgar meus pacotes em algum serviço open source?".

Porque:

- Compartilhar código é sempre uma boa ideia para que a comunidade se beneficie dos avanços individuais de seus membros
- Para sermos desenvolvedores melhores, precisamos receber ajuda e sugestões de utros desenvolvedores mais experientes
- É muito mais fácil usar controle de versão e integração contínua se você estiver programando um pacote
- Reprodutibilidade
- Reprodutibilidade
- Reprodutibilidade

PACOTES

Primeiros passos

O que é um pacote

Em R, pacotes são a unidade fundamental de código compartilhável. Usando pacotes podemos distribuir funcionalidades ao mesmo tempo que garantimos (o máximo possível de) reprodutibilidade.

```
install.packages("abjutils")
devtools::install_github("decryptr/decryptr")
```

Um pacote é uma coleção de código, dados, documentação e testes que qualquer pessoa pode instalar em sua máquina. Se quisermos criar apenas um conjunto de funções que provavelmente não serão utilizadas por muitas pessoas, podemos subir esse pacote para o GitHub e mantê-lo lá somente para garantir controle de versão.

Mas se quisermos que o máximo número possível de pessoas tenha acesso ao nosso pacote, pode ser que precisemos subi-lo para o CRAN (Comprehensive R Archive Network). Neste caso precisaremos criar teste e documentação (em inglês) para nosso pacote.

Nome

Suponha que você criou uma função incrível que você quer compartilhar com o mundo. Ela tem várias funções auxiliares e um comportamento complexo o suficiente para depender de uma documentação... Você precisa criar um pacote.

Mas qual deve ser o seu nome?

> “There are only two hard things in Computer Science: cache invalidation and naming things.” --- Phil Karlton

Os melhores nomes são simples e descritivos. Pense em algo que possa ser procurado facilmente no Google e que, preferencialmente, seja em inglês (a menos que você não ache que pessoas de outro país usarão o seu pacote).

Evite usar letras maiúsculas ou mesmo números pois isso pode confundir os usuários. Você definitavente ganha pontos extras se você conseguir inserir alguma brincadeira com a letra R no nome (`stringr`, `decryptr`, `plyr`, `purrr`, etc.).

O esqueleto do pacote

Para criar o esqueleto de um pacote (um diretório com toda a estrutura necessária), basta rodar `devtools::create()` com o caminho e o nome do seu pacote.

```
devtools::create("caminho_para_dir/pacoter")
```

```
#> ✓ Writing 'pacoter.Rproj'
#> ✓ Adding '.Rproj.user' to '.gitignore'
#> ✓ Adding '^pacoter\\.Rproj$', '^\\.Rproj\\.user$' to '.Rbuildignore'
```

O comando acima criará um diretório chamado `pacoter` com os arquivos e pastas abaixo:

```
list.files("caminho_para_dir/pacoter")
```

```
#> [1] "DESCRIPTION"   "NAMESPACE"     "pacoter.Rproj" "R"
```

Automaticamente `devtools::create()` gera um arquivo `pacoter.Rproj`, permitindo que acessemos o projeto do pacote. `DESCRIPTION` é um arquivo que contém a diversas informações sobre o nosso pacote, enquanto `R` é o

A pasta R

Na pasta `R`, podemos colocar quantos arquivos R quisermos. Todos eles serão colados juntos e carregados ao mesmo tempo para o usuário, então é fortemente recomendado que você divida os diverentes conjuntos de funções do seu pacote em arquivos distintos.

Para propósito de exemplo, criaremos duas funções `tira_media()` e `conta_itens()`, cada uma em seu arquivo `media.R` e `conta.R`.

```
tira_media <- function(x, rm_na = TRUE) {
  purrr::reduce(x, sum, na.rm = rm_na)/conta_itens(x, rm_na)
}

conta_itens <- function(x, compact) {
  if (compact) { x <- purrr::discard(x, is.na) }
  length(x)
}
```

Assim que tivermos os dois arquivos prontos, podemos carregar as funções do nosso para testá-lo manualmente com `devtools::load_all()`.

A pasta R (cont.)

Algumas recomendação sobre como organizar seu código:

- Evite usar `.` no nome das suas funções (hoje em dia usar `_` é muito mais comum)
- Use nomes descritivos para as funções, pois isso facilita a manutenção e o uso do pacote
- Tente se limitar a 80 caracteres por linha porque isso permite que seu código caiba confortavelmente em qualquer tela
- Não use `library()` ou `require()`, pois isso **vai** causar problemas (use a notação `pacote::função()` como no slide anterior)
- Nunca use `source()`, todo o código já será carregado automaticamente
- Prefira manter os arquivos em inglês ou em ASCII (sem acentos) para que seu pacote possa ser submetido no CRAN; se precisar escapar strings, dê uma olhada no add-on `abjutils::escape_unicode()`

Saíndo do básico

Metadados

Agora que já aprendemos a parte mais essencial, podemos passar para os detalhes que diferenciam um pacote ruim de um pacote bom. No arquivo `DESCRIPTION` devemos adicionar o título do nosso pacote, a sua versão, seus autores e contribuidores, e sua descrição.

```
Title: What the Package Does (one line, title case)
Version: 0.0.0.9000
Authors@R: person("First", "Last", email = "first.last@example.com", role = c
Description: What the package does (one paragraph).
```

É aqui também que colocaremos todas as dependências. No caso do `pacoter` existe penas uma (o pacote `purrr`), mas poderíamos ter muito mais.

```
Imports:
  purrr,
  dplyr,
  decryptr
Remotes,
  decryptr/decryptr
```

Documentação

Se quisermos adicionar uma documentação para o nosso pacote (as instruções que aparecem quando vamos usar uma função ou o documento mostrado quando rodamos `?função()`) precisamos usar um comentário especial: `#'`.

```
#' Take the mean of a vector
#'
#' @param x A numeric vector or list
#' @param rm_na Whether or not to remove NAs before taking the mean
#' @return A numeric value
#'
#' @examples
#' tira_media(c(1, 2, 3, 4, NA, 6))
#'
#' @export
```

Quando tivermos terminado, basta rodar `devtools::document()`.

Note que apenas as funções com `@export` serão visíveis para o usuário final do pacote. Não se esqueça de exportar todas (e somente) as funções públicas!

Testes

Se quisermos verificar que todo o pacote continua funcionando mesmo depois fazer alguma alteração, precisamos de testes automatizados. Para isso, basta rodar `devtools::use_testthat()` e depois `devtools::use_test("nome_do_teste")`.

```
test_that("taking the mean works", {
  expect_equal(tira_media(c(1, 2, 3, 4, NA, 6)), 3.2)
  expect_equal(tira_media(c(1, 2, 3, 4, 6)), 3.2)
})

test_that("rm_na works as expected", {
  expect_output(tira_media(c(1, 2, NA), rm_na = FALSE), NA)
  expect_equal(tira_media(c(NA, NA, NA)), NaN)
})
```

Com o pacote `testthat` podemos criar quantos arquivos de testes quisermos, cada um com um número ilimitado de testes e sub-testes. Quando tivermos todos os testes prontos, basta rodar `devtools::test()`.

Conclusão

Agora que aprendemos a criar um pacote, podemos usar a função `devtools::check()`. Ela carrega todo o código, gera toda a documentação e executa todos os testes, verificando em todo passo se tudo está funcionando como o esperado.

Se você seguiu todos os passos corretamente até agora, é provável que ao executar essa função você tenha recebido um *NOTE*. Ela pede que você adicione uma licença ao seu pacote, um documento que indique para outros usuários que direitos e deveres eles têm em relação ao copyright do pacote.

O jeito mais simples de adicionar uma licença é com a função `devtools::use_mit_license()`, permitindo que qualquer um use e modifique o pacote desde que não ganhem dinheiro com isso e mantenham seu nome na licença.

E isso é tudo! Agora só resta subir este pacote para um repositório do GitHub (ou equivalente) para que qualquer pessoa possa usá-lo também. Caso você queira submeter o seu pacote para o CRAN, use a função `devtools::submit_cran()` para auxiliá-lo neste processo.

OBRIGADO!