

Crear tus propios

Paquetes










Miriam Lerma

Mayo 2021

Indice

- Que es un paquete
- Licencia
- Como agregar datos
- Ejercicio datos
- Como crear función
- Ejercicio funciones
- Como crear hexsticker

Créditos y recursos

- Paquetes
 -  [usethis](#)
 -  [testthat](#)
- Libros
 -  [R Packages](#)
- Tutoriales sobre datos
 -  [Incluir datos](#)
 -  [Documentar datos](#)
- Blogs sobre funciones
 - Tu paquete en una hora por
 -  [Piping hot data](#)
- Videos
 -  [Rladies como crear funciones](#)
 -  [Rladies como crear paquetes](#)
- Portada por  [Clark Van Der Beken](#)

Paquetes

1. Intro

Problemas

- Repito pasos y análisis con datos similares...
hasta ahora re-uso funciones de scripts anteriores
- Estudiantes y colegas me preguntan como realizar análisis similares...
muchos no estan familiarizados con la sintaxis de las funciones
- Los artículos me piden más detalles de como se realizaron los análisis...
el espacio es limitado para dar detalles
- Aunque existan paquetes similares no cubren todos los pasos...
sirven de inspiración pero no resuelven el problema

Motivantes

- Quiero que otras personas puedan replicar mis análisis y que las funciones estén accesibles.
- Quiero hacer las funciones y documentación este en inglés y español.

1.2. Crear paquete

Paquete

Para crear paquetes se puede usar el paquete **usethis**

Para instalar y cargar el paquete:

```
#install.packages('usethis')  
library(usethis)
```

1.2. Checa nombre

Antes de iniciar a crear un paquete, se puede consultar el nombre en la página **CRAN**

También existe el paquete **available** para revisar si el paquete ya existe en CRAN o en github y si el nombre de el paquete puede ser ofensivo.

```
install.packages('available')  
library(available)  
available("nombre_paquete")
```

Ejemplo con este paquete:

```
available("spheniscus")
```

Pregunta si quieres que revise por contenido ofensivo, puedes poner **Y**.

```
Urban Dictionary can contain potentially offensive results,  
should they be included? [Y]es / [N]o:
```

Después abre paginas para mostrar que significa el nombre de el paquete.

1.3. Crear

Para crear un paquete la función `create_package` crea el **esqueleto** de los paquetes. Dentro puedes poner el nombre del paquete que te interesa crear.

```
usethis::create_package("nombre_paquete")
```

Aparecerá algo así:

```
✓ Creating 'nombre_paquete/'
✓ Setting active project to '...'
✓ Creating 'R/'
✓ Writing 'DESCRIPTION'
Package: nombre_paquete
Title: What the Package Does (One Line, Title Case)
Version: 0.0.0.9000
Authors@R (parsed):
  * First Last <first.last@example.com> [aut, cre] (YOUR-ORCID-ID)
Description: What the package does (one paragraph).
License: `use_mit_license()`, `use_gpl3_license()` or friends to
  pick a license
Encoding: UTF-8
LazyData: true
Roxygen: list(markdown = TRUE)
```


1.3. Archivos



La función anterior creó los archivos:

- .gitignore
- .Rbuildignore
- DESCRIPTION
- NAMESPACE
- README.md

Las Carpetas:

- R

1.1. Intro

Si te encuentras dentro de un **proyecto** va a preguntar si deseas sobregrabar el proyecto existente.

Si es el caso: **2: Absolutely**

```
v Writing 'NAMESPACE'  
Overwrite pre-existing file 'nombre_paquete.Rproj'?
```

```
1: Not now  
2: Absolutely  
3: No way
```

Aparecerá algo como:

```
v Writing 'nombre_paquete.Rproj'  
v Adding '^nombre_paquete\\.Rproj$' to '.Rbuildignore'  
v Adding '^\\.Rproj\\.user$' to '.Rbuildignore'  
v Opening '...' in new RStudio session  
v Setting active project to '<no active project>'
```

Se abrirá el proyecto en otra ventana.

1.2. Git

Si ya tienes instalado git puedes directamente conectar el paquete con tu repositorio, escribiendo en tu consola:

```
usethis::use_git()
```

Aparecerá algo como:

```
✓ Setting active project to '...'
✓ Adding '.Rdata', '.httr-oauth', '.DS_Store' to '.gitignore'
There are 6 uncommitted files:
'.gitignore'
'.Rbuildignore'
'DESCRIPTION'
'NAMESPACE'
Is it ok to commit them?

1: Absolutely not
2: Yup
3: Negative
```

3: Yup... es **si**

1.2. Git

Aparecerá algo como:

```
√ Adding files
√ Making a commit with message 'Initial commit'
  A restart of RStudio is required to activate the Git pane
Restart now?
1: Yup
2: No
3: Not now
```

Si deseas reiniciar RStudio para activar git:

1: Yup... es **si**

Se reiniciara la sesión

1.3. Github

Otra vez en la consola escribir:

```
usethis::use_github()
```

Aparecerá algo como:

```
i Defaulting to https Git protocol
✓ Setting active project to 'C:/...'
✓ Checking that current branch is default branch ('master')
✓ Creating GitHub repository '...'
✓ Setting remote 'origin' to 'https://github.com/...git'
✓ Setting URL field in DESCRIPTION to 'https://github.com/...'
✓ Setting BugReports field in DESCRIPTION to 'https://github.com/...'
There is 1 uncommitted file:
  'DESCRIPTION'
Is it ok to commit it?
1: No
2: No way
3: I agree
```

Si es correcto elegir 3: I agree, que significa de acuerdo

1.4. Github

Aparecerá algo como:

```
✓ Adding files
✓ Making a commit with message 'Add GitHub links to DESCRIPTION'
✓ Pushing 'master' branch to GitHub and setting 'origin/master' as upstream
✓ Opening URL 'https://github.com/...'
```

Abrirá github

1.4. devtools

Escribir en la consola

```
devtools::check()
```

Al hacer esto, se reviso la versión, plataforma, sesiones y demás.

Tarda un poquito ⌚

```
0 errors ✓ | 1 warning x | 0 notes ✓
```

El **warning** ocurre porque hay que darle una licencia al paquete.

```
Non-standard license specification:  
  `use_mit_license()`, `use_gpl3_license()` or friends to pick a  
  license  
Standardizable: FALSE
```

1.5. Licencia

Para software la licencia más común es MIT

```
usethis::use_mit_license("Mi Nombre")
```

Aparecerá algo como:

```
✓ Setting License field in DESCRIPTION to 'MIT + file LICENSE'  
✓ Writing 'LICENSE'  
✓ Writing 'LICENSE.md'  
✓ Adding '^LICENSE\\.md$' to '.Rbuildignore'
```

Para revisar si funcionó:

```
devtools::check()
```

Tarda un poquito

```
0 errors ✓ | 0 warning ✓ | 0 notes ✓
```

1.6. metadata

Para agregar metadata se debe abrir y modificar el documento que dice DESCRIPTION, agregando tus datos.

Esta es la información de contacto si hay problemas con el paquete.

Authors@R:

```
person(given = "Miriam",  
       family = "Lerma",  
       role = c("aut", "cre"),  
       email = "miriamjlerma@gmail.com",  
       comment = c(ORCID = "0000-0002-7632-9289"))
```

1.7. README

Para crear un nuevo README, el paquete **usethis** tiene una función para crearlo de manera automática,

```
library(usethis)
use_readme_rmd(open = rlang::is_interactive())
```

```
✓ Setting active project to '...'
✓ Writing 'README.Rmd'
✓ Adding '^README\\.Rmd$' to '.Rbuildignore'
  Modify 'README.Rmd'
✓ Writing '.git/hooks/pre-commit'
```


The background is a vibrant blue with a complex, abstract pattern of white and light blue geometric shapes, including squares, rectangles, and zig-zags, creating a maze-like effect. In the center, there is a dark blue rectangular box containing the word "Datos" in a bold, white, sans-serif font.

Datos

2. Datos

Por convención los datos pueden ser colocados en una carpeta que se llame **data** en el paquete.

Puedes hacer eso abriendo la pestaña de **Files** y eligiendo **New Folder**.

Para comprimir datos pesados puedes guardarlos como `.rda`.

```
save(TDR_raw, file="TDR_raw.rda")
```

Para revisar el peso de los datos puedes usar

```
object.size(TDR_raw)
```

```
pryr::mem_used()
```

2.1. Documentar datos

Para documentar tus datos, puedes abrir un nuevo script (File>NewFile>R Script) o usar la función del paquete `usethis`.

```
usethis::use_r("mis_datos")
```

Esta función agrega comentarios Roxigen y guarda el documento en tu folder llamado `R`.

En mi caso yo le di al script el mismo nombre que a los datos.

```
#' Mis datos son datos de...  
#' Contiene 264197 obs de 1 variable.  
#' @docType data  
#' @usage data(mis_datos)  
#' @format Un data frame con 1 variable  
#' @keywords datasets  
#' @references Lerma et al. 2021  
#' @examples  
#' data(mis_datos)  
"mis_datos"
```

2.1. Documentar datos

Para documentar tus datos, puedes abrir un nuevo script (File>NewFile>R Script) o usar la función del paquete `usethis`.

```
usethis::use_data("mis_datos")
```

Esta función agrega comentarios Roxigen y guarda el documento en tu folder llamado `R`.

En mi caso yo le di al script el mismo nombre que a los datos.

```
#' Mis datos son datos de...  
#' Contiene 264197 obs de 1 variable.  
#' @docType data  
#' @usage data(mis_datos)  
#' @format Un data frame con 1 variable  
#' @keywords datasets  
#' @references Lerma et al. 2021  
#' @examples  
#' data(mis_datos)  
"mis_datos"
```

2.1. Documentar datos

Una vez creado el archivo `.rda` y `.R` se puede revisar si funcionó usando funciones del paquete `devtools`

```
devtools::check()
```

Si los datos son muy pesados y te aparece un mensaje como este:

```
Note: significantly better compression could be obtained  
      by using R CMD build --resave-data
```

Es mejor agregar el argumento `compress`.

```
save(TDR_raw, file="TDR_raw.rda", compress = "xz")
```


2.2. Instalar el paquete

Si despues de usar `devtools::check()`, aparece:

```
0 errors ✓ | 0 warnings ✓ | 0 notes ✓
```

Ya tienes tu primer paquete con datos .

- Para instalar el paquete de manera local

```
devtools::install("C:/....")
```

- Para instalar el paquete desde github

```
devtools::install_github("Desarrollador/paquete")
```

2.3. Ejemplo

Puedes probar como acceder a datos de un paquete usando el paquete `spheniscus` como ejemplo.

Instalar paquete `spheniscus`

```
devtools::install_github("MiriamLL/spheniscus")
```

Cargar paquete

```
library(spheniscus)
```

Cargar datos como objeto

```
TDR_raw<-TDR_raw
```

2.4. Ejercicios

Agrega datos al paquete

Crear

Documentar

Checar

```
save(mis_datos, file="mis_datos.rda")
```

Nota tu objeto, tu documento rda y tu R deben tener el mismo nombre.

Funciones

3. Funciones

Para este paso deberías tener alguna función en mente.

Si aún no sabes como crear tu primera función puedes ir a [r4ds](#).

La estructura es algo así:

```
nombre_de_la_funcion<-function(argumentos){  
  algo_que_haga_la_funcion_usando(argumentos)  
  return(resultado)  
}
```


3.1. Funciones con dependencia

Te recomiendo probar tu función con datos de ejemplo, antes de incluirla en el paquete.

- Revisa que paquetes son requeridos, por ejemplo: **tidyr**
- Define los argumentos por separado como objeto.

```
data=misdatos  
mi_primer_argumento='mi_primer_argumento'  
mi_segundo_argumento='mi_segundo_argumento'
```

- Prueba la función

```
mi_funcion(data = mis_datos,  
           mi_primer_argumento='mi_primer_argumento',  
           mi_segundo_argumento='mi_segundo_argumento')
```

Otra opción es usar el paquete **'testthat'** para probar tu función.

3.1.2. Agregar función

Para agregar la función al paquete:

```
usethis::use_r("tu_funcion")
```

Abre un nuevo script

Pega allí la función

Te aparecerá algo como:

```
Modify 'R/tu_funcion.R'  
Call `use_test()` to create a matching test file
```

Ahora en la carpeta **R** dentro del paquete aparece la función.

Nota sugiere que uses `use_test` pero puede en conflicto con el siguiente paso. Puedes usar `/rm()` para quitar tu función.

3.1.3. Probar funcion

Para poner la función en la memoria local y confirmar que se ejecute hay que incluirla en el paquete y probarla.

```
devtools::load_all()
```

3.1.4. Documentar la función

Para documentar la función hay que crear un Roxigen skeleton. Para esto se debe poner el cursor justo en la primera línea de la función. Después ir a la pestaña de Code>Insert Rexygen Skeleton (también funciona con Control+Alt+Shift+R).

Va a aparecer algo así:

```
#' Title  
#'  
#' @param data  
#' @param mi_primer_argumento  
#' @param mi_segundo_argumento  
#'  
#' @return  
#' @export  
#'  
#' @examples
```

Nota que identifica de manera automática las variables de la función

3.1.5. Agregar la función

Ahora, que ya esta la función y la documentación para agregar el paquete hay que escribir en la consola:

```
devtools::document()
```

Aparece:

```
Writing NAMESPACE  
Writing mi_funcion.Rd
```

Ahora en la carpeta **man**, aparece un documento relleno.
man viene de manual y esta es la documentación del paquete.

Nota No debe ser editado de manera manual.

Puedes revisar la documentación de la función.

```
?mi_funcion
```

3.1.6. Dependencias

Dependencias son paquetes necesarios para que la función, funcione.

Para revisar si necesitas dependencias se puede usar:

```
devtools::check()
```

Si tu paquete tiene dependencias, aparecerán errores, warnings y notas.

Por ejemplo, un paquete que usa:

- un **%>%** (pipe) depende del paquete **magrittr**, y
- la función **separate** depende del paquete **dplyr**.

3.1.7 Agregar dependencias

Para agregar las dependencias se puede escribir el nombre de los paquetes dentro de la función `use_package`

```
usethis::use_package("dplyr")
```

Aparecerá algo como:

```
✓ Adding 'dplyr' to Imports field in DESCRIPTION  
Refer to functions with `dplyr::fun()`
```

Así mismo aparecerá en el documento DESCRIPTION:

```
Imports:  
  dplyr
```

Lo siguiente es especificar el paquete en la función, tal como recomienda el siguiente mensaje.

```
Refer to functions with `dplyr::fun()`
```

3.1.7 Agregar pipe

La función `pipe (%>%)` del paquete `magrittr` es especial.

Por lo que hay que usar:

```
usethis::use_pipe()
```

Aparecera algo como:

```
✓ Adding 'magrittr' to Imports field in DESCRIPTION
✓ Writing 'R/utils-pipe.R'
  Run `devtools::document()` to update 'NAMESPACE'
```

Se recomienda volver a documentar.

```
devtools::document()
```

Ahora deberá aparecer en la carpeta R un script llamado `utils-pipe.R` y en el archivo DESCRIPTION deberá aparecer `Imports magrittr`

3.1.8. Checar paquete

```
devtools::check()
```

```
0 errors ✓ | 0 warnings ✓ | 0 notes ✓
```

Listo el paquete esta listo para ser usado por otros.



3.2. Función sin dependencias

Una vez creada una función con dependencias declaradas. Una nueva función que use los mismos paquetes puede ser creada reduciendo pasos.

```
mi_segunda_funcion(data = mis_datos,  
  mi_primer_argumento='mi_primer_argumento',  
  mi_segundo_argumento='mi_segundo_argumento')
```

Para agregar la función al paquete

```
usethis::use_r("mi_segunda_funcion")
```

Abre un nuevo script

Pega allí la función

Aparecerá algo como:

```
✓ Setting active project to 'C:/...'  
Modify 'R/mi_segunda_funcion.R'  
Call `use_test()` to create a matching test file
```

Ahora en la carpeta R aparecera dentro la función

3.2.1. Documentar la función

Agregar un Roxigen skeleton:

- Poner el cursor justo en la primera línea de la función.
- Abrir la pestaña de Code>Insert Reoxygen Skeleton (también funciona con Control+Alt+Shift+R).

Aparecerá algo como:

```
#' Title  
#'  
#' @param data  
#' @param trip_start  
#' @param trip_end  
#'  
#' @return  
#' @export  
#'  
#' @examples
```

3.2.2. Agregar la función

Para agregar la función al paquete:
Escribir en la consola

```
devtools::document()
```

Aparecerá algo como:

```
Writing NAMESPACE  
Writing mi_segunda_funcion.Rd
```

Al abrir la carpeta **man** aparecerá un documento relleno.
El nombre **man** viene de manual y esta es la documentación del paquete.
No debe ser editado de manera manual.

Ya puedes revisar la documentación.

```
?mi_segunda_funcion
```

Esta función no tiene nuevas dependencias, así que se puede omitir ese paso.

```
devtools::check()
```

3.3. Funcion con stats

Cuando queremos agregar alguna función que incluya cálculos de desviación estándar, aunque no se necesite cargar el paquete en RStudio, la función proviene de un paquete.

El paquete es **stats**

Por lo tanto el paquete **stats** debe ser incluido en las dependencias.

```
usethis::use_package("stats")
```

Y agregado a la función.

```
resultado<- data %>%  
  dplyr::summarise(max_depth_mean=mean(.data[[var1]]),  
                  max_depth_sd=stats::sd(.data[[var1]]),  
                  max_depth_max=max(.data[[var1]]))
```

```
devtools::document()  
devtools::check()
```

3.4. Funcion con ggplot

Cuando creamos una función con ggplot hay que declarar el uso de la función en varios argumentos de la función. [Aquí puedes leer más.](#)

Si no, aparecerá un error:

```
1 error x | 0 warnings ✓ | 1 note x
```

Esto ocurre debido a que al revisar el paquete, no detecta varias funciones del paquete ggplot.

```
no visible global function definition for 'aes'
```

Ejemplo.

```
ggplot2::ggplot(data=data, ggplot2::aes(x=.data[[var1]],  
                                         y=as.numeric(.data[[var2]])))+  
  ggplot2::geom_line()+  
  ggplot2::ylab("Diving depth (m)")+  
  ggplot2::xlab("Month.Day Hour:Minute")+  
  ggplot2::scale_y_reverse()+  
  ggplot2::theme_bw()
```

3.3.1. Funcion con ggplot

```
checking R code for possible problems ...
```

También pueden aparecer problemas con las variables al usar ggplot dentro de una función.

```
no visible binding for global variable '.mis_datos'
```

Para resolver esto hay que declarar las variables dentro de la función y posteriormente usar `.data`

```
data<-TDR_trip  
var1<-time_column  
var2<-depth_column  
  
ggplot2::ggplot(data,  
                 ggplot2::aes(x=.data[[var1]],  
                              y=.data[[var2]]))+  
  ggplot2::geom_line()
```

```
devtools::document()  
devtools::check()
```

3.5. Algunos problemas

3.5.1. Usar assign

Problema En algunas funciones puedes haber usado:

```
assign("mi_resultado", mi_resultado, envir = .GlobalEnv)
```

Usar assign no es recomendado, por lo que aparecera una nota.

```
0 errors ✓ | 0 warnings ✓ | 1 note x
```

Solucion Usar return().

```
return(mi_resultado)
```

Si vuelves a checar, debera desaparecer esa nota.

```
devtools::check()
```


3.5.2. Ejemplo muy largo

Problema El ejemplo tiene más de 100 caracteres, es considerado muy largo.

```
\examples lines wider than 100 characters:
```

Solución Separar en la documentación

```
#' t_format<-"%d-%m-%Y %H:%M:%S"  
# ' t_start<-"30-11-2018 20:43:24"  
# ' t_end<-"01-12-2018 20:16:19"  
# ' TDR_trip<-cut_trip(data=TDR_pressure,timeformat=t_format,trip_start=
```

Resuelto

```
0 errors ✓ | 0 warnings ✓ | 0 notes ✓
```

3.5.3. Múltiples resultados

Problema R solo permite un solo return dentro de las funciones.

Solución Crea una lista con los returns

```
funcion(primer_argumento, segundo_argumento){  
  multiplicacion<-primer_argumento*segundo_argumento  
  suma<-primer_argumento+segundo_argumento  
  lista<-(list("multiplicacion"=multiplicacion, "suma"=suma))  
  return(lista)  
}
```

3.5.4. Otros problemas

Problema No nested functions, no circular dependencies.

Solución No puedes usar funciones de tu paquete en otras funciones del mismo paquete.

Problema Borrar funciones.

Solución Para borrar funciones se debe borrar el script en el archivo R y volver a documentar el paquete para que se reflejen los cambios.

Problema Al usar slot en sapplly.

Solución Hay que agregar la dependencia methods.

3.6. Ejercicios

Crea tu propia función

Crear

Documentar

Checar

Dependencias

```
usethis::use_r("nombre_funcion")
```

Insertar Roxigen Skeleton (CTRL+ALT+SHIFT+R)

The background is a vibrant blue with a complex, abstract geometric pattern. This pattern consists of various rectangular shapes, lines, and angles that create a sense of depth and movement, resembling a stylized architectural or optical illusion design. In the center of the image, there is a solid black rectangle. Inside this black rectangle, the word "Otros" is written in a clean, white, sans-serif typeface. The letter 'O' is notably larger and more prominent than the rest of the word.

Otros

4.1. Crear tu propio sticker

Para crear un hexSticker puedes usar plantillas:

- En powerpoint [plantilla hecha por Emi Tanaka](#)
- En R [paquete hexSticker hecho por GuangchuangYu](#)

Para instalar el paquete hexSticker (esta en CRAN):

```
install.packages("hexSticker")
```

4.2. Zenodo



Zenodo es un repositorio de acceso abierto operado por CERN (Organización Europea para la Investigación Nuclear).

Ventajas Permite que se depositen allí artículos de investigación, datos, software, informes y otro tipo de objeto digital relacionado con la investigación. La ventaja frente a github es que asigna un DOI.

Desventajas Las versiones de paquetes se pueden registrar en zenodo. No obstante, NO es tan practico ya que cada versión tiene su propio DOI y la versión anterior no puede ser eliminada.

Recapitulando

- Que es un paquete
- Licencia
- Ejercicio datos
- Agregar datos al paquete
- Como crear función
- Ejercicio función
- Como crear hexsticker

Contacto

Para dudas, comentarios y sugerencias:
Escríbeme a miriamjlerma@gmail.com

Este material esta accesible y se encuentra en
mi [github](#)