CREACIÓN DE UN PAQUETE EN R

PALLASCO CATOTA JONATHAN FERNADO

email: jonathan.pallasco@epn.edu.ec

ESCUELA POLITÉCNICA NACIONAL

Quito-Ecuador: 4 de octubre de 2022

Introduccion

En esta presentacion se va a explicar de manera muy detallada como se crea y además se explica las componentes claves que debe contener un nuevo paquete creado en software estadístico R.
Para mantener el ritmo, aprovechamos las comodidades modernas del paquete devtools y el IDE de RStudio.

Instalacion y llamado del paquete devtools

DEVTOOLS

Puede iniciar su nuevo paquete desde cualquier sesión de R activa. No necesita preocuparse por si está en un proyecto nuevo o existente o no. Las funciones que usamos se encargan de esto.

Cargue el paquete devtools como se muestra en el recuadro. Este paquetes la cara pública de un conjunto de paquetes que admiten varios aspectos del desarrollo de paquetes. El más obvio de estos es el paquete usethis, que verá que también se está cargando.

```
# Cargamos el paquete devtools
install.packages( 'devtools' )
```

Instalacion y llamado del paquete devtools

Una vez cargada la libreria se procede a llamarla

```
library(devtools)
Loading required package: usethis
```

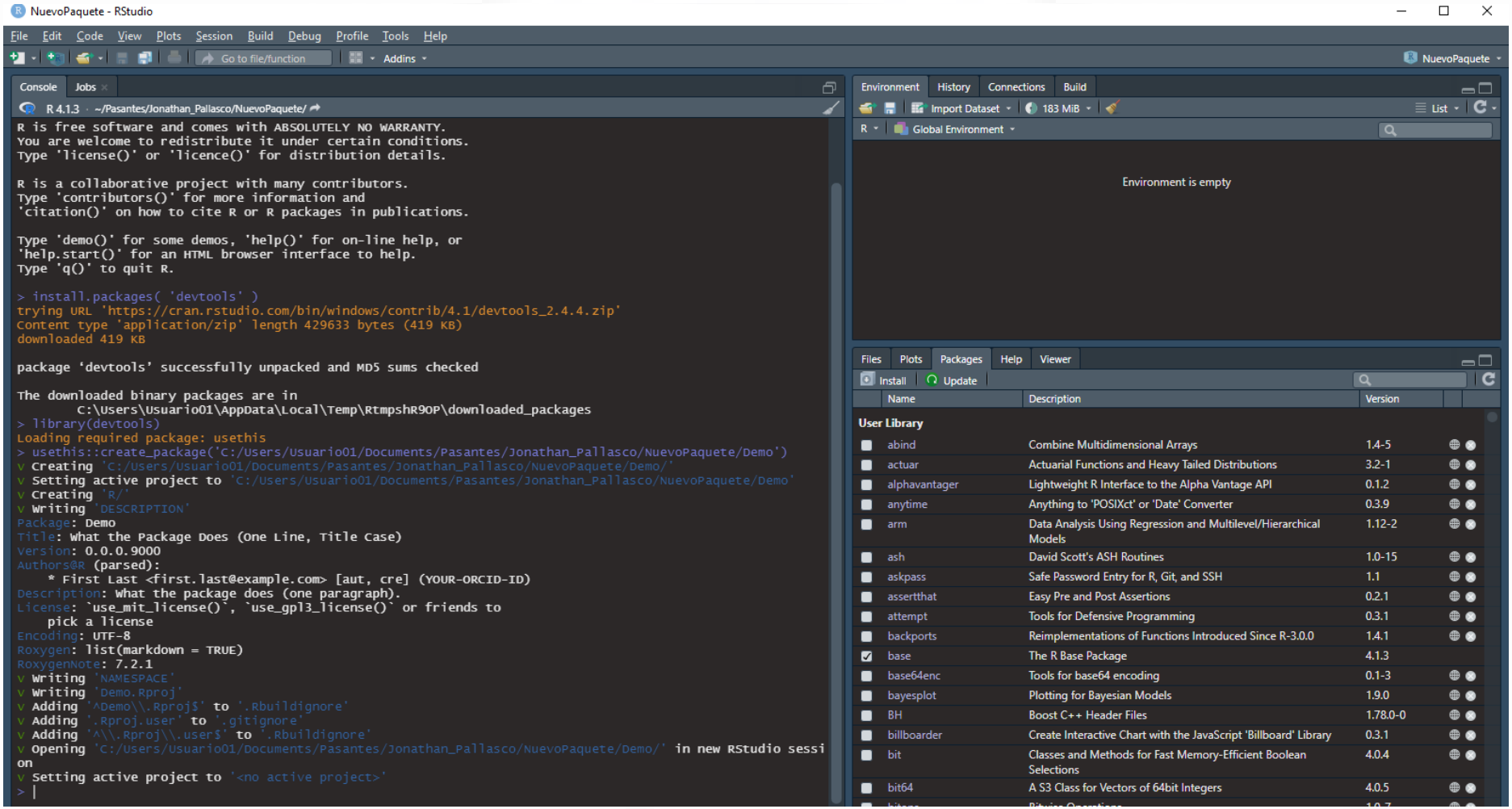
Ahora se utilizan varias funciones de devtools para construir un paquete desde cero, con características que se ven comúnmente en los paquetes lanzados. Las funciones que contienen el paquete **usethis** son de suma importancia y se carga al momento de instalar el paquete **devtools**.

CREACIÓN DEL PAQUETE :

Para empezar la creación de un nuevo paquete en R, debemos llamar a la función create_package() del paquete **usethis**, para inicializar un nuevo paquete en un directorio de su computadora. Esta función crea automáticamente ese directorio.

```
usethis::create_package('path/Nuevo_Paquete')
```

Vizualizacion en R



Archivos que contiene el proyecto

En este nuevo directorio, existe un proyecto RStudio (El nuevo paquete). Aquí hay una lista de 6 archivo:

1. **.Rbuildignore** : Enumera los archivos que debemos tener pero que no deben incluirse al compilar el paquete R desde el código fuente.
2. **.gitignore**: Anticipa el uso de Git e ignora algunos archivos estándar detrás de escena creados por R y RStudio. Incluso si no planea usar Git, esto es inofensivo.
3. **DESCRIPTION**: Proporciona metadatos sobre su paquete.
4. **NAMESPACE**: Declara las funciones que su paquete exporta para uso externo y las funciones externas que su paquete importa de otros paquetes.
5. **R/directorio**: Es el extremo comercial de su paquete.
6. **regexcite.Rproj**: Es el archivo que convierte a este directorio en un proyecto de RStudio.

Conversión a un repositorio Git

El directorio **Nombre_paquete** es un paquete de R y un proyecto RStudio. Ahora se debe convertir en un repositorio Git, usando la funcion **use_git()**. Cabe recalcar que, use_git() funciona en cualquier proyecto, independientemente de si es un paquete R.

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

```
> usethis::use_git()
v Setting active project to 'C:/Users/Usuario01/Documents/Pasantes/Jonathan_Pallasco/NuevoPaquete/Demo'
v Initialising Git repo
v Adding '.Rhistory', '.Rdata', '.httr-oauth', '.DS_Store' to '.gitignore'
There are 5 uncommitted files:
* '.gitignore'
* '.Rbuildignore'
* 'Demo.Rproj'
* 'DESCRIPTION'
* 'NAMESPACE'
```

Escribe la primera función

Toda funcion esta en un archivo .R (**Nombre_Funcion.R)**, y se va a encontrar almacenada en la carpeta **R/** del proyecto.
La función **use_r()** del paquete **usethis**, crea y/o abre un script a continuación R/

```
usethis::use_r('suma')
```

```
> usethis::use_r("Funcion1")
v Setting active project to 'C:/Users/Usuario01/Documents/Pasantes/Jonathan_Pallasco/NuevoPaquete/Demo'
* Modify 'R/Funcion1.R'
* Call `use_test()` to create a matching test file
```

Una vez, creada una funcion simple, debemos llamar a tolas las funciones, para poner a **suma** a disposición para la experimentación, utilizando la funcion **load_all()** del paquete **devtools**.

```
devtools::load_all('.')
```

Creación y compilación de funciones

```
> devtools::load_all(".")
i Loading Demo
> suma(10,20)
[1] 30
> suma(10,1.40)
[1] 11.4
> suma(12.34,1.87)
[1] 14.21
```

Tenemos evidencia empírica informal que la función **suma()** funciona correctamente. Pero, ¿cómo podemos estar seguros de que todas las partes móviles del paquete **Demo()** todavía funcionan?.
Usar la función **check()** del paquete **devtools** es una forma conveniente de verificar que las partes moviles del paquete funcionen sin salir de su sesión R.

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

```
-- R CMD check results ------------------------------------------------------- Demo 0.0.0.9000 ----
Duration: 15.6s

> checking DESCRIPTION meta-information ... WARNING
  Non-standard license specification:
    `use_mit_license()`, `use_gpl3_license()` or friends to pick a
    license
  Standardizable: FALSE

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

METADATOS DEL PAQUETE

Hay dos archivos importantes que proporcionan metadatos sobre su paquete, estos son los archivos: **DESCRIPTION** y **NAMESPACE**. **DESCRIPTION** proporciona metadatos generales sobre el paquete y **NAMESPACE** describe qué funciones usa de otros paquetes y expone al mundo.

DESCRIPTION

Todo paquete debe tener un DESCRIPTION. De hecho, es la característica que define a un paquete y posee varios campos directos.

- **Title:** Se describe en una línea el paquete y, a menudo, se muestra en una lista de paquetes. Debe ser texto sin formato (sin marcado), en mayúsculas como un título, y NO terminar en un punto. Sea breve: los listados a menudo truncarán el título a 65 caracteres.
- **Description:** Es más detallado que el título. Puede usar varias oraciones, pero está limitado a un párrafo. Si su descripción abarca varias líneas, cada línea no debe tener más de 80 caracteres de ancho.

METADATOS DEL PAQUETE

Si planea enviar su paquete a CRAN, tanto los **Title** como **Description** son una fuente frecuente de rechazos, asi se presenta algunas sugerencias:

a) Coloque los nombres de los paquetes, el software y las API de R entre comillas simples.
b) No incluya el nombre del paquete, especialmente en Title.
c) No comience con 'Un paquete para...' o 'Este paquete hace...'.

- **Authors:** Para identificar al autor del paquete y a quién contactar si algo sale mal, use el campo Authors@R. Este campo es inusual porque contiene código R ejecutable en lugar de texto sin formato. Para ello se utiliza la función **person()** y posee 4 parametros de entrada:

```
person(given, family, email, role, comment)
```

El **nombre**, especificado por los dos primeros argumentos, given y family. El **email**, es la dirección del correo, y además es la que utiliza CRAN para informarle si es necesario reparar su paquete para permanecer en CRAN.

METADATOS DEL PAQUETE

El **role** es el cargo de las personas entre las mas importantes:

- cre Creador o mantenedor.
- aut Autores, aquellos que han hecho contribuciones significativas al paquete.
- ctb Contribuyentes, aquellos que han hecho contribuciones más pequeñas.
- cph Titular de derechos de autor. Esto se utiliza si los derechos de autor pertenecen a alguien que no sea el autor, normalmente una empresa (es decir, el empleador del autor).
- fnd Financiador, las personas u organizaciones que han proporcionado apoyo financiero para el desarrollo del paquete.

El **comment** es argumento opcional, el mas relevante es el ORCID (es un código alfanumérico, no comercial, que identifica de manera única a científicos y otros autores académicos.)

METADATOS DEL PAQUETE

- **Url**: El URLcampo se usa comúnmente para anunciar el sitio web del paquete y para vincular a un repositorio de fuente pública, donde ocurre el desarrollo. Las URL múltiples se separan con una coma.
- **BugReportses**: Es la URL donde se deben enviar los informes de errores, por ejemplo, como problemas de GitHub.
- **Encoding**: Describe la codificación de caracteres de los archivos en todo el paquete. Encoding: UTF-8 es ahora la codificación de texto más utilizada
- **Collate**: Controla el orden en que se obtienen los archivos R.
- **Versiones** Realmente importante como una forma de comunicar dónde se encuentra su paquete en su ciclo de vida y cómo evoluciona con el tiempo
- **LazyData** Es relevante si su paquete pone los datos a disposición del usuario. Si especifica LazyData: true, los conjuntos de datos se cargan de forma diferida, lo que hace que estén disponibles de forma más inmediata, es decir, los usuarios no tienen que utilizar data().

NAMESPACE

El archivo **NAMESPACE** se parece un poco al código R. Cada línea contiene una **directiva** : S3method(), export(), exportClasses(), etc. Cada directiva describe un objeto R y dice si se exporta desde este paquete para que lo usen otros, o si se importa desde otro paquete para usarlo localmente.
En total, hay ocho directivas de espacio de nombres. Cuatro describen las exportaciones:

1. **export():** Funciones de exportación (incluidos los genéricos S3 y S4).
2. **exportPattern():** Exporta todas las funciones que coinciden con un patrón.
3. **exportClasses()** y **exportMethods():** Funciones de exportación (incluidos los genéricos S3 y S4).
4. **S3method():** exportar métodos S3.

METADATOS DEL PAQUETE

Y cuatro describen las importaciones:

1. **import():** Importa todas las funciones de un paquete.
2. **importFrom():** Importar funciones seleccionadas (incluidos los genéricos de S4).
3. **importClassesFrom()** y **importMethodsFrom():** Importar clases y métodos S4.

CÓDIGO R

El primer principio de hacer un paquete es que todo el código R va en el R/directorio. Y se debe organizar las funciones en archivos, mantener un estilo consistente.

FUNCIONES .R

Organizar funciones en archivos: La única regla es que el paquete debe almacenar sus definiciones de función en scripts R, es decir, archivos con extensión .R, que se almacenan en el R/directorio. El nombre del archivo debe ser significativo y transmitir qué funciones se definen dentro. Si bien se puede organizar funciones en archivos como desee.

Sugerencias

- No colocar todas las funciones en un archivo.
- No colocar cada función en su propio archivo separado.

COMPONENTES DE UN PAQUETE

Sin embargo, si una función es muy grande o tiene mucha documentación, puede tener sentido darle su propio archivo, con el nombre de la función; es decir, un solo archivo .R contendrá múltiples definiciones de funciones como se muestra en el cuadro 1.

Cuadro: Definiciones de funciones

Principio Organizador	**Descripción**
Una función	Define exactamente una función que no es particularmente grande, pero no encaja naturalmente en ningún otro archivo .R
Función principal más ayudantes	Define el orientado al usuario, un método y ayudantes privados.
Familia de funciones	Define una familia de funciones para rectangular" listas anidadas, todas documentadas juntas en un gran tema de ayuda, además de ayudantes privados

Fuente: `https://r-pkgs.org/Code.html`

Elaborado: Pallasco Catota Jonathan F.

COMPONENTES DEL PAQUETE

Documentación de funciones: La documentación es uno de los aspectos más importantes de un buen paquete: ¡sin ella, los usuarios no sabrán cómo usar su paquete! La documentación también es útil para el autor en el futuro (para que recuerde lo que se suponía que debían hacer sus funciones) y para los desarrolladores que deseen ampliar el paquete.

Una forma estándar de documentar un paquete donde cada tema de documentación corresponde a un archivo .Rd en el directorio man/. Estos archivos usan una sintaxis personalizada, basada libremente en LaTeX, que se procesan en HTML, texto sin formato o pdf, según sea necesario, para su visualización. No vamos a utilizar estos archivos directamente. En su lugar, usaremos el paquete roxygen2 para generarlos a partir de comentarios con formato especial. Hay algunas ventajas de usar roxygen2:

COMPONENTES DEL PAQUETE

- El código y la documentación se entremezclan para que cuando modifique su código, sea fácil recordar actualizar también su documentación.
- Puede usar Markdown, en lugar de aprender una nueva sintaxis de formato de texto.

Roxigen2: Para comenzar, trabajaremos con el flujo de trabajo básico de roxygen2 y analizaremos la estructura general de los comentarios de roxygen2 que se organizan en bloques y etiquetas.

- **El flujo de trabajo de la documentación**
  El flujo de trabajo de la documentación comienza cuando agrega comentarios de roxygen, comentarios que comienzan con ', a su archivo de origen. Aquí hay un ejemplo simple:

COMPONENTES DEL PAQUETE

```r
#' Add together two numbers
#'
#' @param x A number.
#' @param y A number.
#' @return The sum of `x` and `y`.
#' @examples
#' add(1, 1)
#' add(10, 1)
add <- function(x, y) {
  x + y
}
```

Ahora que comprende el flujo de trabajo básico, hablemos un poco más sobre la estructura y la sintaxis que se utiliza. Los comentarios de roxygen2 comienzan con ' y todos los comentarios de roxygen2 que preceden a una función se llaman colectivamente un bloque .

COMPONENTES DEL PAQUETE

Los bloques se dividen en etiquetas , que parecen @tagName tagValue, entre las mas importantes estan las siguientes: @params, @returns, @seealso, @examples y @export. El contenido de una etiqueta se extiende desde el final del nombre de la etiqueta hasta el comienzo de la siguiente etiqueta.

Título y descripción:

Utilice la primera línea de la documentación de su función para proporcionar un título conciso que describa la función, el conjunto de datos o la clase. Los títulos deben usar mayúsculas y minúsculas pero no terminar con un punto (.).

No es necesario utilizar las etiquetas explícitas @title o @description, excepto en el caso de la descripción si se trata de varios párrafos o si incluye un formato más complejo, como una lista con viñetas.

COMPONENTES DEL PAQUETE

Una vez conocida la estructura de la documentaci'on de una función se procede a crear su archivo .Rd en el directorio man, la cual se crea de manera automatica usando la siguientes sintaxis

```
devtools::document()
roxygen2::roxygenise()
```

Bases de datos .rda La ubicación más común para los datos del paquete es directorio data/. Se recomienda que cada archivo en este directorio sea un .rda. La forma más fácil de lograr esto es usar la función use_data("Nuevos_datos").
Imaginemos que estamos trabajando en un paquete llamado "Nuevo". El fragmento anterior crea data/Nuevos_datos.rda dentro de la fuente del paquete "Nuevoz agrega LazyData: true en el archivo DESCRIPTION. Esto hace que el objeto de R Nuevos_datos esté disponible para los usuarios de Nuevo a través de Nuevo::Nuevos_datos,

Four Basic Counting Principles

Cabe recalcar que la anterior función es algo que el mantenedor ejecuta una vez (o cada vez que necesitan actualizar Nuevos_Datos). Este es un código de flujo de trabajo y no debería aparecer en el directorio de R, del paquete fuente.
De igual manera para la documentación de las bases de datos es identica a la documentación de las funciones, las cuales ya se explicaron anteriormente.