Escuela Politécnica Nacional
Facultad de Ciencias

Manual para crear un nuevo paquete en R

Fecha de valuación : 2020-12-31

Quito - Ecuador

Manual Nuevo Paquete R

Instituto Ecuatoriano de Seguridad Social, IESS

Título: Manual para crear un nuevo paquete en R

Departamento: Dirección Actuarial, de Investigación y Estadística

Dirección: Avenida 10 Agosto y Bogotá, Quito-Ecuador

Versión: c623d7999e5e61411859e190e39e03238e43b3bc40f8b26b46e65c468bef2979

Contenidos

Manual Nuevo Paquete R

Tablas

Manual Nuevo Paquete R

Figuras

1 Introducción

En **R**, la unidad fundamental del código compartible es el paquete. Un paquete agrupa código, datos, documentación y pruebas, y es fácil de compartir con otros. A partir de junio de 2022, había más de 18 000 paquetes disponibles en **CRAN** (**C**omprehensive **R** **A**rchive **N**etwork). La mayor parte de la población que usa el software estadístico **RStudio**, ya sabe cómo trabajar con paquetes de las siguientes maneras:

- Se instala desde CRAN usando la sintaxis **install.packages("Nombre_paquete")**.
- Se lo usa en R mediante **library(Nombre_paquete)**.
- Se accede a las ayudas de las funciones usando **?Nombre_Funcion**.

¿Por qué escribir un paquete? Una razón eficiente es que el usuario tiene un código que desea compartir con otros. Al agrupar su código en un paquete, es fácil que otras personas lo usen porque, al igual que nosotros, ya saben cómo usar los paquetes. Si su código está en un paquete, cualquier usuario de R puede descargarlo, instalarlo y aprender a usarlo fácilmente.

1.1 Objeto del Manual

El objetivo principal de este manual es darles a conocer de manera sencilla y rápida cómo desarrollar paquetes en RStudio, para que pueda escribir los suyos propios, no solo usar los de otras personas.

2 Creación de un paquete R en RStudio

Este capítulo 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.

2.1 Devtools y dependencias

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.

```
# Instalamos el paquete devtools
install.packages( 'devtools' )
library(devtolls)
```

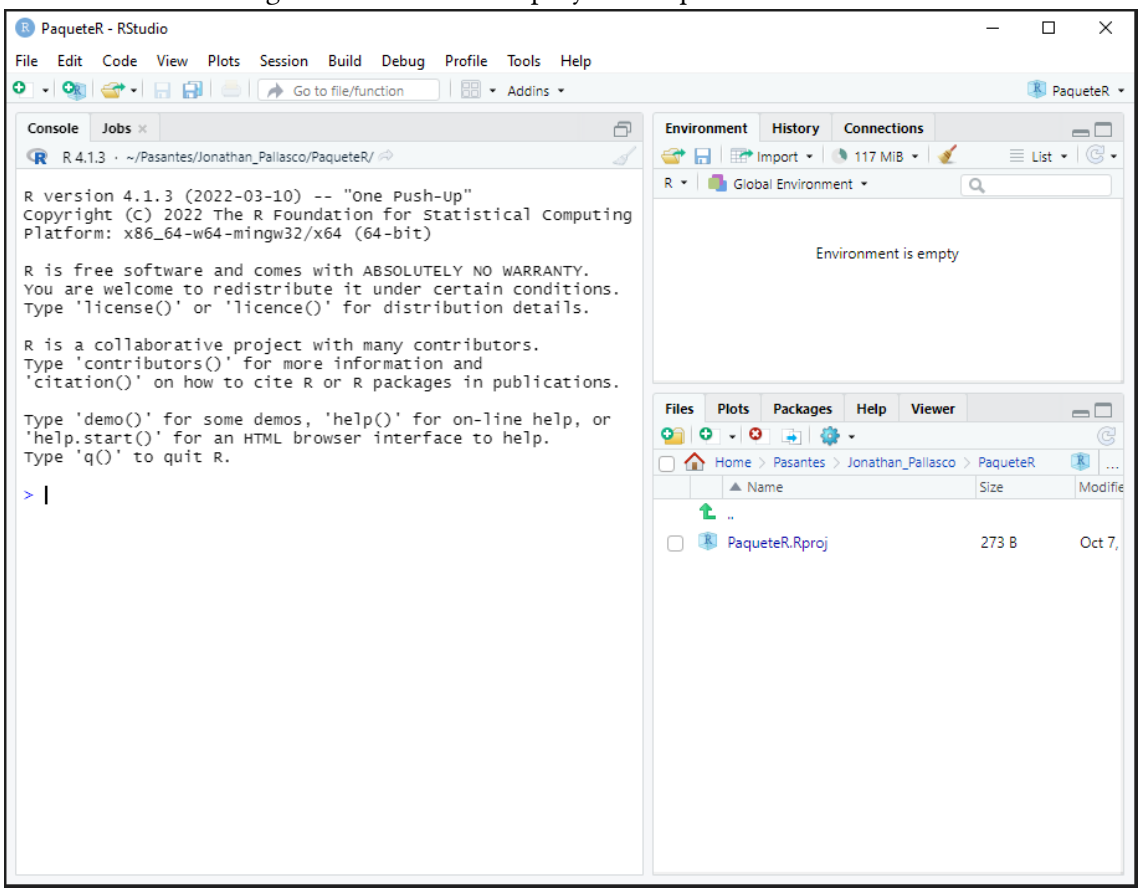
Por otro lado, vamos a usar varias funciones de **devtools**, para construir un paquete desde cero, manteniendo la estructura de los paquetes que se encuentran publicados en CRAN.

2.2 Inicialización de un nuevo paquete R

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.

Ahora empezamos creando un nuevo proyecto en nuestra consola de R Studio, llamemolo al proyecto como **PaqueteR**, y se obtiene algo similar a la figura 2.1. Este proyecto posee un único archivo **PaqueteR.Rproj** que convierte a este directorio en un proyecto de RStudio.

Figura 2.1: Creación del proyecto PaqueteR en R Studio



Elaborado: Pallasco Catota Jonathan Fernando
Fuente: Software estadístico R Studio.

Luego procedemos a ejecutar la función **create_packages()** en la consola del nuevo proyecto para iniciar un nuevo paquete. Cabe recalcar que en esta función es neseario especificar adecadamente la dirección junto con el nombre que le vayamos a dar al paquete; es decir, se ejecuta la siguiente linea de código.

```
# Creamos un nuevo paquete
usethis::create_package( 'C:/Users/Usuario01/Documents/Pasantes/
                         Jonathan_Pallasco/NuevoPaqueteR' )
```

Ahora se nos habre una nueva sección de R, en la cual ya se ha creado nuestro proyecto llamado **NuevoPaqueteR**, como se muestra en la figura 2.2. Además, se puede observar que en este nuevo proyecto se construyó una lista 5 nuevos archivos los cuales se describen a continuación:

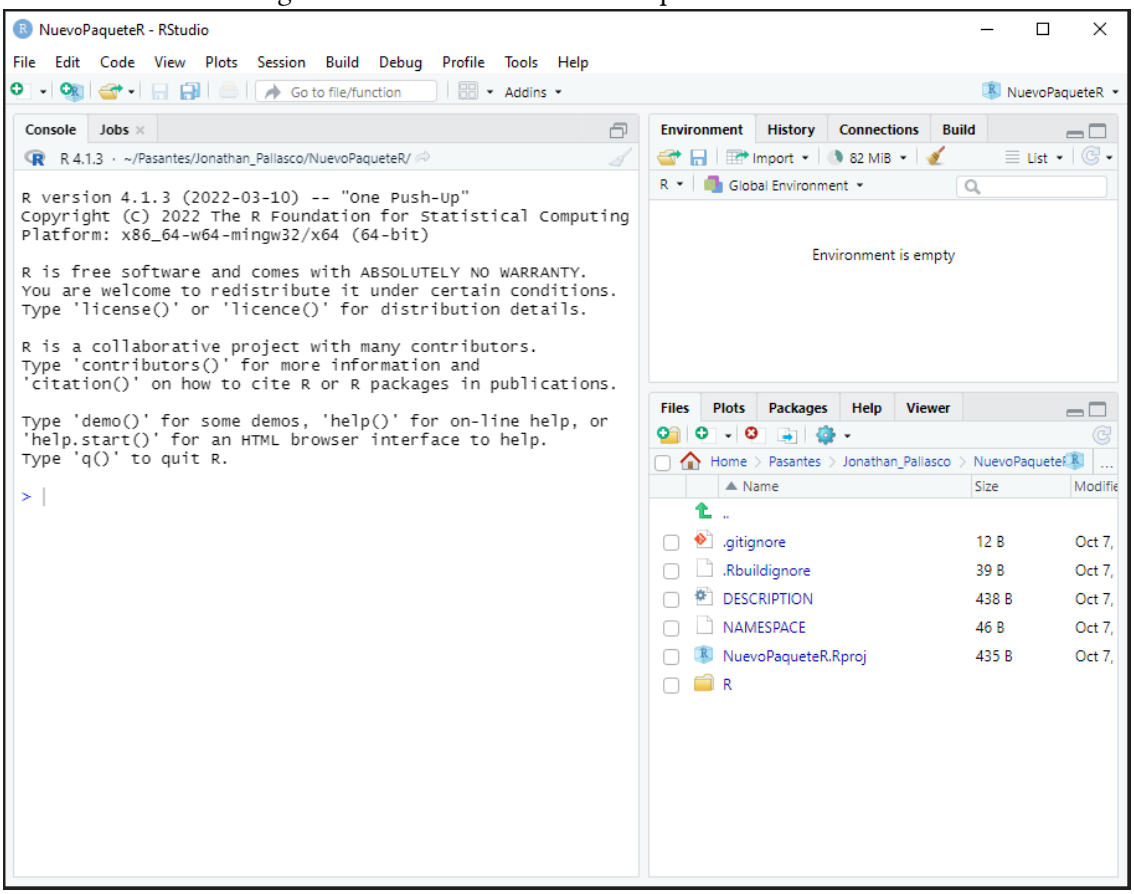
1. **.Rbuildignore** : Enumera los archivos que debemos tener pero que no deben incluirse al compilar el paquete R desde el código fuente.

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. **Directorio R**: Es el extremo comercial de su paquete.

Figura 2.2: Creación del nuevo PaqueteR en R Studio



Elaborado: Pallasco Catota Jonathan Fernando
Fuente: Software estadístico R Studio.

Nota: Probablemente necesitemos volver a llamar **library(devtools)**, porque **create_package()**, utilizo una nueva sección de R en el nuevo paquete.

2.3 Conversión del paquete en un repositorio Git

El directorio **NuevoPaqueteR** es un paquete fuente R y un proyecto RStudio. Ahora lo convertimos también en un repositorio Git. Para ello vamos a usar la función **use_git()**. En esta función, no se especifican los parámetros de entrada.

```
# Creamos el repositorio git
usethis::use_git()
```

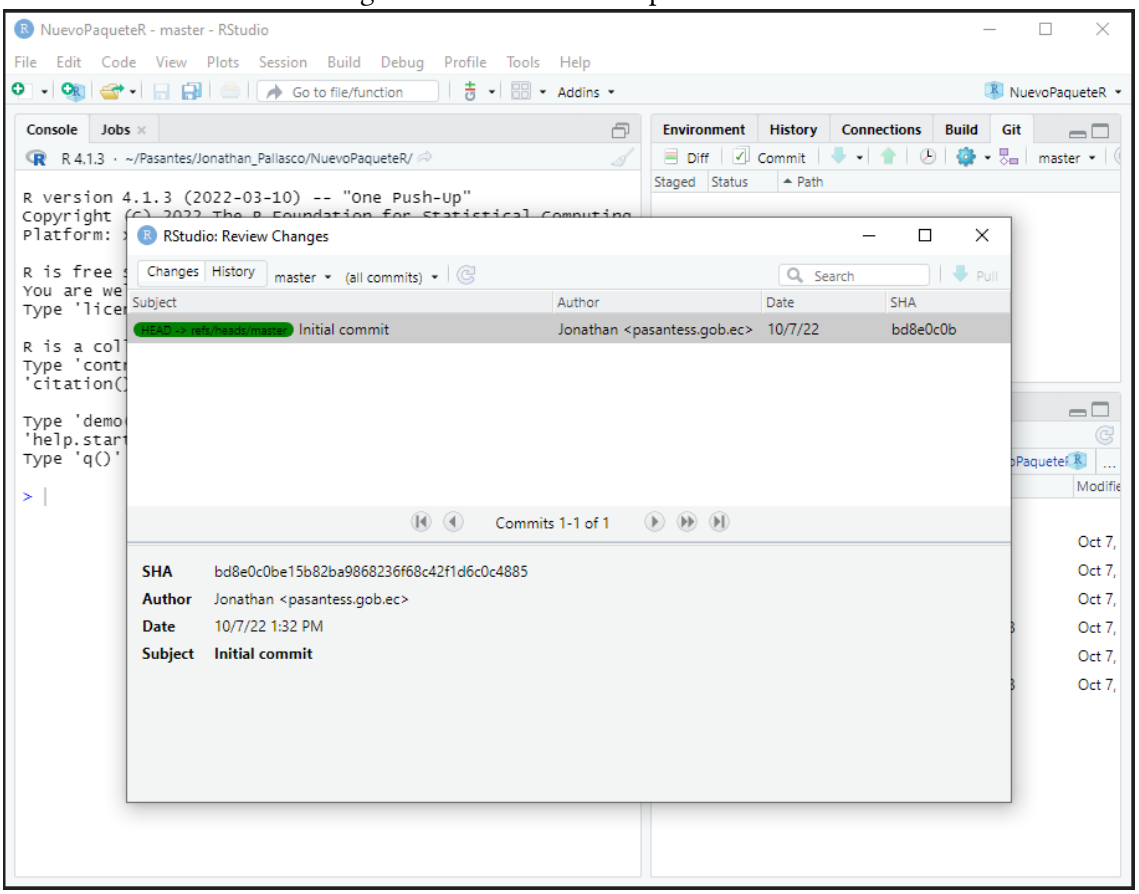
Nota: La función **use_git()** funciona en cualquier proyecto, independientemente de si es un paquete R.

Nos preguntamos: ¿Qué hay de nuevo? Solo la creación de un directorio .git, que está oculto en la mayoría de los contextos, incluido el explorador de archivos de RStudio. Su existencia es evidencia de que, de hecho, hemos inicializado un repositorio de Git aquí.

Si está utilizando RStudio, probablemente solicitó permiso para reiniciarse en este proyecto, lo cual debe hacer. Puede hacerlo manualmente saliendo y luego reiniciando RStudio haciendo doble clic en NuevoPaqueteR.Rproj. Ahora, además de la compatibilidad con el desarrollo de paquetes, tiene acceso a un cliente Git básico en la pestaña Git del panel Entorno/Historial/ Compilación.

Ahora hacemos clic en Historial (el ícono del reloj en el panel de Git) y, si dio su consentimiento, verá una confirmación inicial realizada a través de **use_git()**, como se muestra en la figura 2.3.

Figura 2.3: Creación del repositorio Git



Elaborado: Pallasco Catota Jonathan Fernando
Fuente: Software estadístico R Studio.

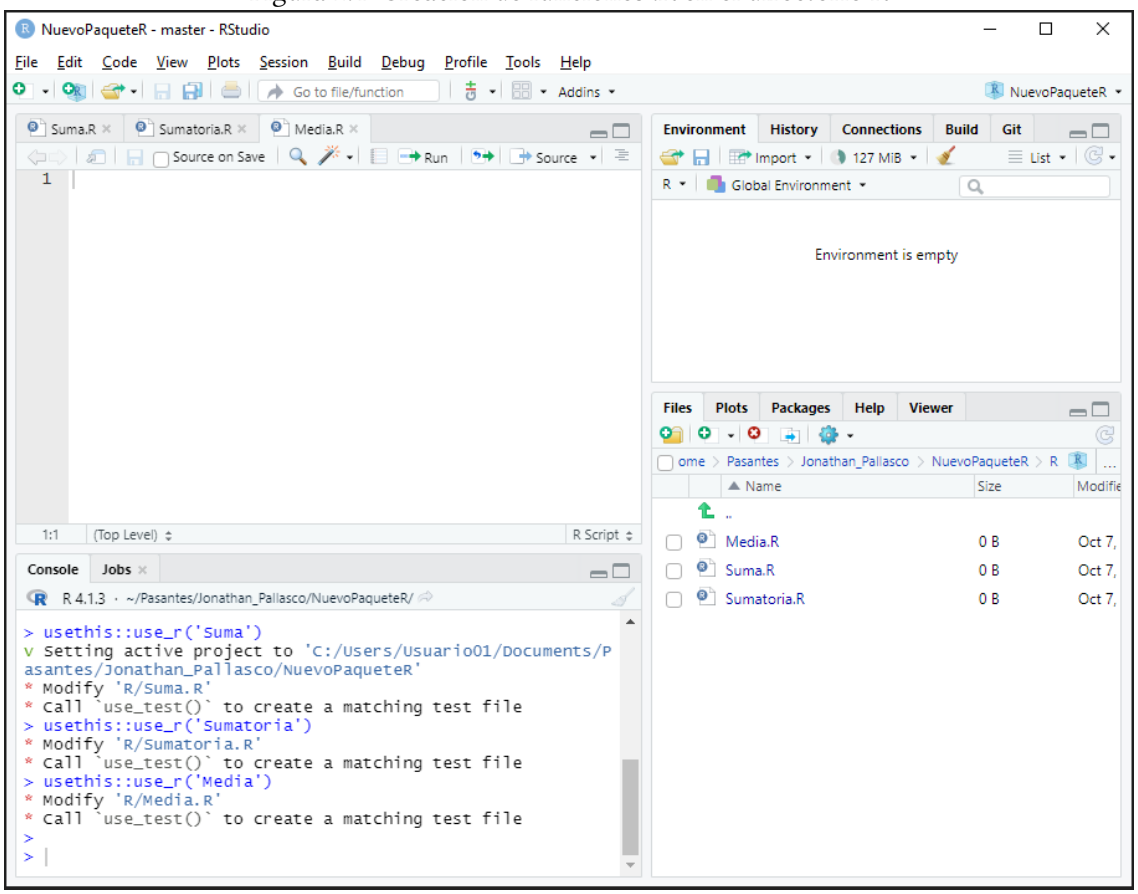
2.4 Funciónes y bases de datos del paquete R

Una vez construido nuestro paquete, nos preguntamos ¿Como se crean las funciones y las bases de datosque estan en nuestro paquete?

Debemos tener en cuenta que las funciones de nuestro paquetes deben ser archivos **.R** y las bases de datos deben de ser de tipo **.rda**. La poblacion que a indagado en los paquetes lanzados al CRAN, deben tener claro que en un paquete no se debe mesclar las fucniones y las bases de datos, es por ello que las funciones deben ser almacenadas en repositorio **R** y las bases de datos deben ser almacenadas en el repositorio **Data**. Como vimos anteriormente el repositorio **R** se crea automáticamente al momento de iniciar nuestro paquete. Ahora el repositorio **Data** no se encuentra en nuestro paquete. Sin embargo, existen funciones del paquete **devtolls** que realiza este trabajo de manera automática, más adelante se vera a detalle su funcionamiento.

2.4.1 Creación de Funciones

Figura 2.4: Creación de funciones .R en el directorio R



Elaborado: Pallasco Catota Jonathan Fernando
Fuente: Software estadístico R Studio.

Como se dijo anteriormente, toda función debe estar en un archivo .R (**NombreFuncion.R)** que se almacena en el directorio **R/** de nuestro paquete. Existe una función del paquete **usethis** que nos crea funciónes automáticamente y la almacena directamente en el directorio **R** de nuestro paquete. La función **use_r()** es la que se encarga de todo este proceso ya que ayuda a crea y/o abrir un script en el repositorio **R**. En esta función el parámetro importante que se debe especificar es el nombre de la función, el mismo que se usa para renombrar al scrip creado.

Tabla 2.1: Código de las funciones del nuevo paquete R

Nombre de la función	Código

Suma.R	`# Funcion suma`
	`Suma <- function(a,b){`
	`  suma <- a + b`
	`  return(suma)`
	`}`

Sumatoria.R	`# Funcion Sumatoria`
	`Sumatoria <- function(vector){`
	`  n <- length(vector)`
	`  sumatoria <- 0`
	`  for (i in 1:n){`
	`  sumatoria <- sumatoria + vector[i]`
	`  }`
	`  return(sumatoria)`
	`}`

Media.R	`# Funcion Media`
	`Media <- function(vector){`
	`  n <- length(vector)`
	`  media <- Sumatoria(vector)/n`
	`  return(media)`
	`}`

Elaborado: Pallasco Catota Jonathan Fernando
Fuente: Software estadístico R Studio.

Para la demostración, se va a crear 3 funciones simples, como se muestra a continuación:

```
# Creamos funciones
usethis::use_r('Suma')
usethis::use_r('Sumatoria')
usethis::use_r('Media')
```

Es evidente que se crearon nuevos scrip en el directorio R de nuestro proyecto, como se muestra en la figura 2.4.

Ahora vamos a escrir el código dentro de cada una de las funciones, de tal manera que cumpla las siguientes especificaciones:

1. **Suma.R:** Calcula la suma de 2 números.
2. **Sumatoria.R:** Calcula la sumatoria de n números.
3. **Media.R:** Calcula la media de un vector de n números.

Notemos que en la tabla 2.1, se puede observar los códigos utilizados para la creación de las funciones:

Es evidente que los códigos son demasiado simples, más adelante veremos la documentación que deben tener las funciones de todos los paquetes lanzados en **CRAN**.

2.4.1.1 Verificación del funcionamiento de las funciones creeadas en el paquete R

Una vez, creadas las funciones, debemos hacer la prueba de las mismas. ¿Cómo hacemos la prueba de manejo **Suma()**, **Sumatoria()** y **Media()?** Si se tratara de un script R normal, podríamos usar RStudio para enviar la definición de la función a R Console y definir **Suma()**, **Sumatoria()** y **Media()** en el entorno global. O tal vez llamaríamos **source('R/Suma.R')**, **source('R/Sumatoria.R')** o **source('R/Media.R')**. Sin embargo, para el desarrollo de paquetes, devtools ofrece un enfoque más sólido.

Llamar a la función **load_all()** es lo único que se debe hacer para poner a **Suma()**, **Sumatoria()** y **Media()** para la experimentación como se muestra en el recuadro.

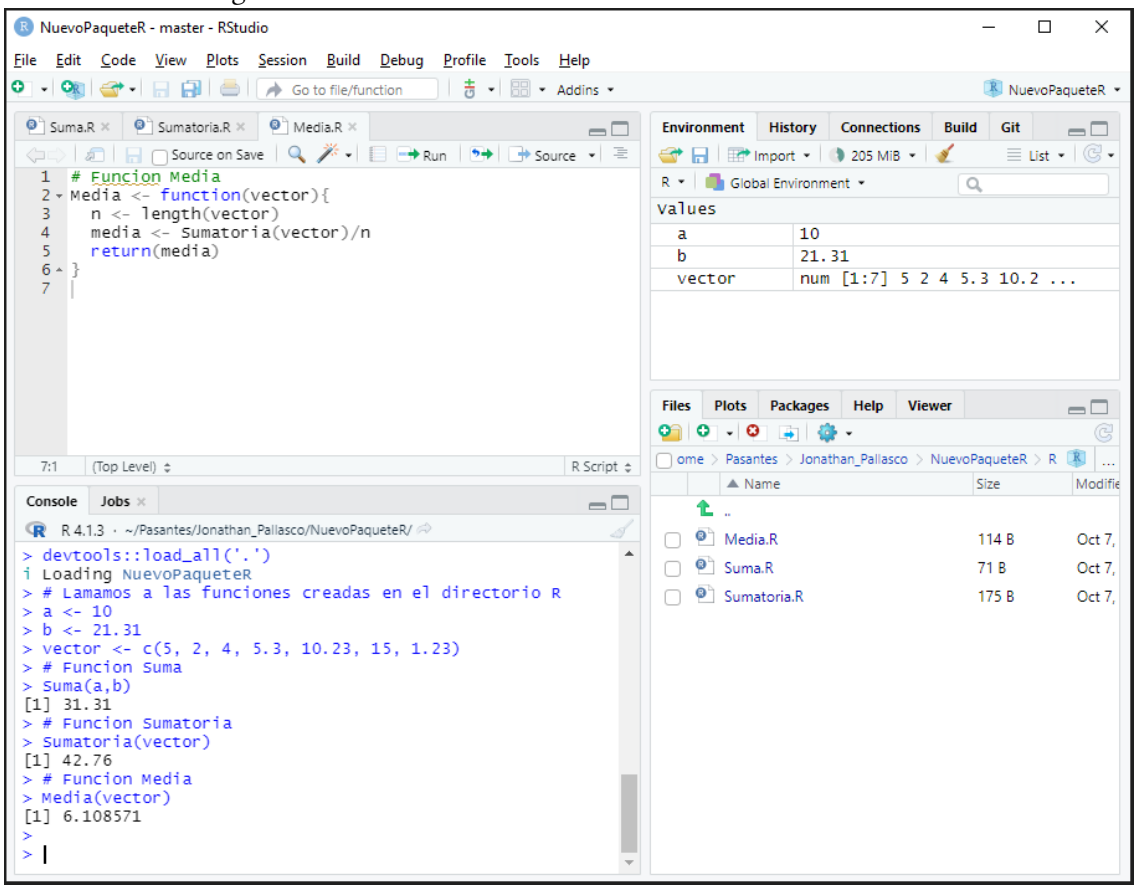
```
# Lamamos a las funciones creadas en el directorio R
devtools::load_all('.')
}
```

Simplemente debemos llamar a **Suma()**, **Sumatoria()** y **Media()?** para ver cómo funciona. El llamado se lo realiza de la siguiente manera,

```
# Lamamos a las funciones creadas en el directorio R
```

```
a <- 10
b <- 21.31
vector <- c(5, 2, 4, 5.3, 10.23, 15, 1.23)
# Funcion Suma
Suma(a,b)
# Funcion Sumatoria
Sumatoria(vector)
# Funcion Media
Media(vector)
```

Figura 2.5: Prueba de las funciones creadas en el directorio R



Elaborado: Pallasco Catota Jonathan Fernando
Fuente: Software estadístico R Studio.

En la figura 2.5, se observa que se carga el paquete **NuevoPaqueteR**, por ende se cargaron las funciones antes mensionadas. Además se observa en la sonsola que estan funcionando correctamente. Ahora verifiquemos de manera manueal que esta correcto el resultado, entonces se procede a efectuar todas las operaciones antes mensionadas, lo cual coinciden con el resultado de las funciones creadas

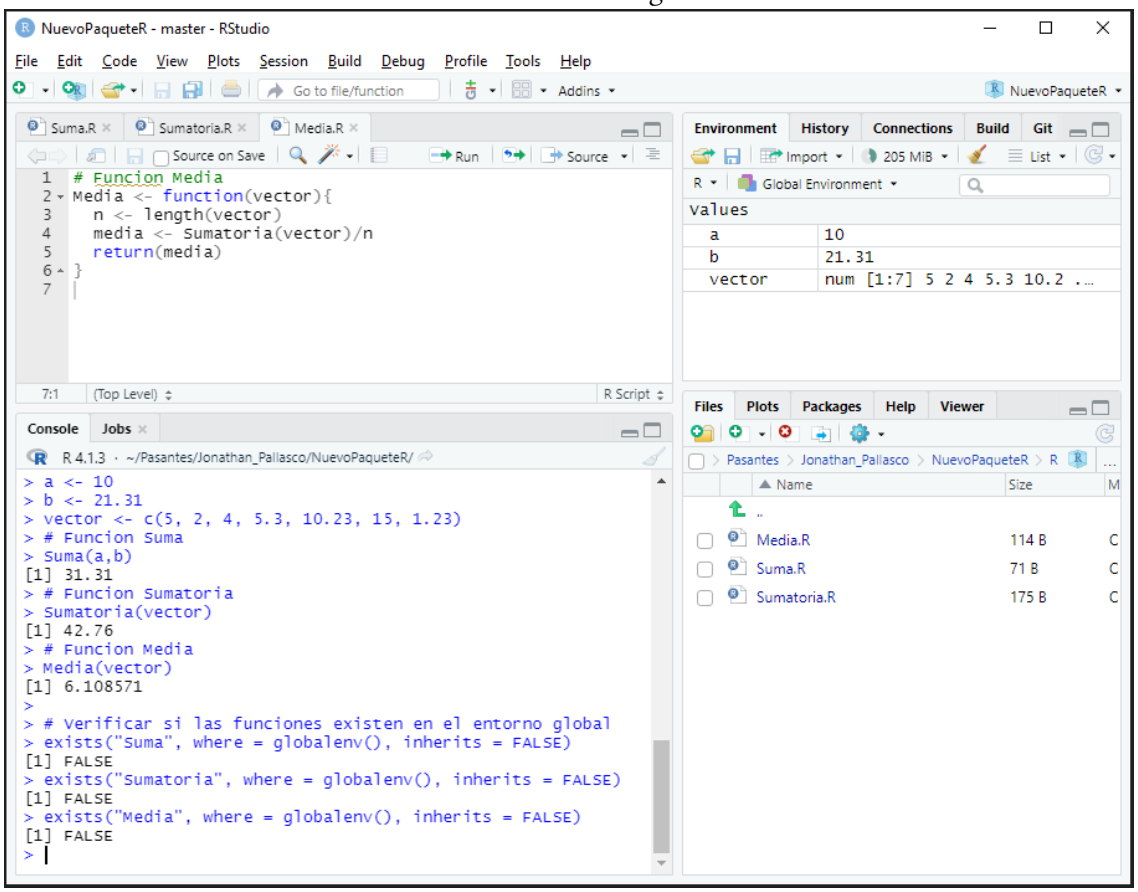
$$\text{Suma} = 10 + 21{,}31 = 31{,}31$$

$$\text{Sumatoria} = 5 + 2 + 4 + 5{,}3 + 10{,}23 + 15 + 1{,}23 = 42{,}76$$

$$\text{Media} = \frac{5 + 2 + 4 + 5{,}3 + 10{,}23 + 15 + 1{,}23}{7} = 10{,}1085$$

```
# Verificar si las funciones existen en el entorno global
exists("Suma", where = globalenv(), inherits = FALSE)
exists("Sumatoria", where = globalenv(), inherits = FALSE)
exists("Media", where = globalenv(), inherits = FALSE)
```

Figura 2.6: Prueba de verificamos para la existencia de las funciones en el entorno global



Elaborado: Pallasco Catota Jonathan Fernando
Fuente: Software estadístico R Studio.

Tenga en cuenta que **load_all()** ha hecho que las funciones esté disponible, aunque no existe en el entorno global, para ello verificamos si existen en el entorno global con la función **exists()** como se muestra en el recuadro y además en la figura 2.6. Como se

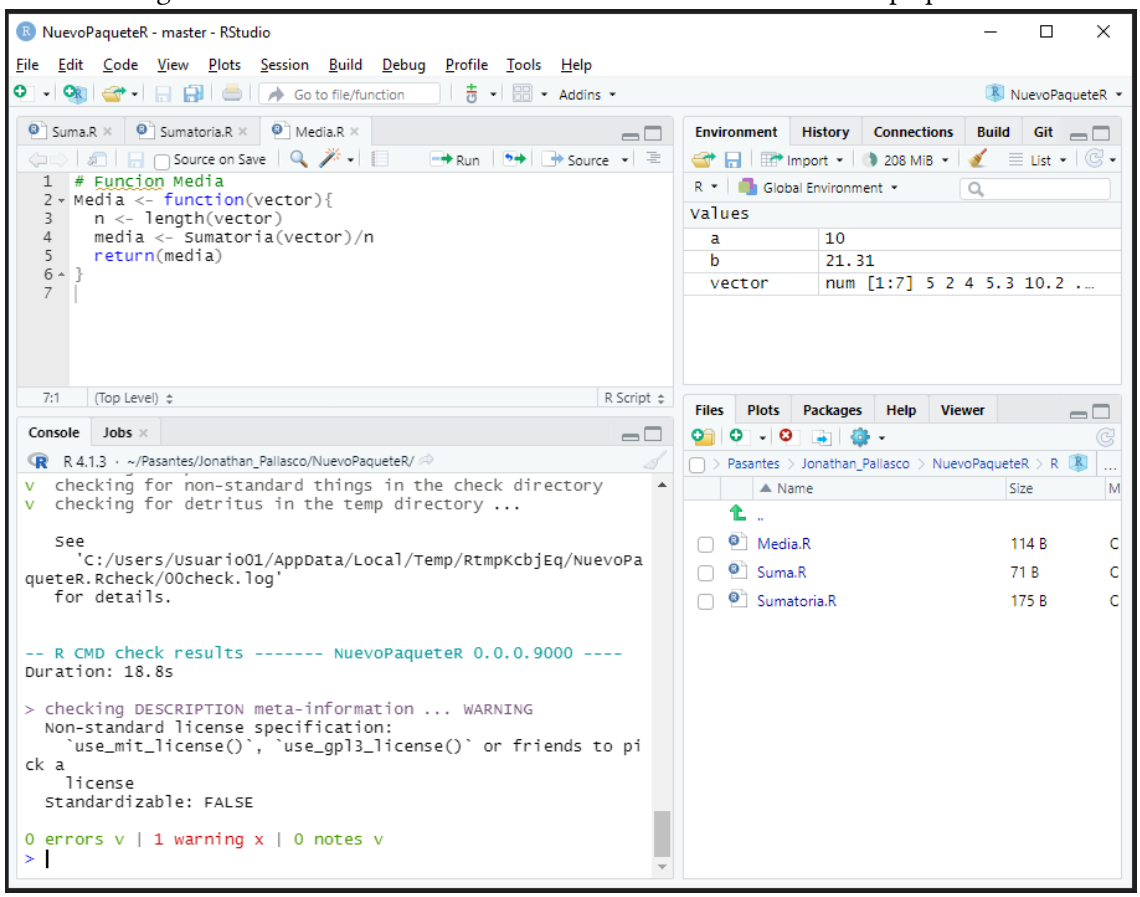
ve FALSE en las 3 funciones, entonces concluimos que las funciones no existen en el entorno global, por otro lado si fuese TRUE, entonces eso indica que todavía está usando un flujo de trabajo orientado a secuencias de comandos y obteniendo sus funciones.

La **load_all()** simula el proceso de creación, instalación y conexión del paquete **NuevoPaqueteR**. A medida que su paquete acumula más funciones, algunas exportadas, otras no, algunas de las cuales se llaman entre sí, algunas de las cuales llaman a funciones de los paquetes de los que depende, **load_all()** le brinda una idea mucho más precisa de cómo se está desarrollando el paquete que las funciones de conducción de prueba definidas en el entorno mundial. También **load_all()** permite una iteración mucho más rápida que construir, instalar y adjuntar el paquete.

2.4.1.2 Verificación del funcionamiento del paquete R

Tenemos evidencia empírica informal que **Suma()**, **Sumatoria()** y **Media()** funcionan. Pero, ¿cómo podemos estar seguros de que todas las partes móviles del paquete **NuevoPaqueteR** todavía funcionan? Verificar esto puede parecer una tontería, después de una adición tan pequeña, pero es bueno establecer el hábito de verificar esto con frecuencia.

Figura 2.7: Prueba de verificación de funcionamiento del nuevo paquete R



Elaborado: Pallasco Catota Jonathan Fernando
Fuente: Software estadístico R Studio.

Una forma conveniente de verificar esto es usando la función **check()** sin salir de su sesión R. La sintaxis para ejecutar la función es la siguiente:

```
# Verificar que el paquete funciona
devtools::check()
```

Tenga en cuenta que **check()** produce una salida bastante voluminosa, optimizada para el consumo interactivo. Lo interceptamos en la figura 2.7 y solo revelamos un resumen. !Esto puedo tomar pocos minutos no se desespere!.

Podemos ver que no existe errores en el paquete, lo cual es bueno para nosotros como programadores.

2.4.2 Creación de Bases de Datos

Suele ser útil incluir datos en un paquete. Si el objetivo principal de un paquete es distribuir funciones útiles, los conjuntos de datos de ejemplo facilitan la redacción de documentación excelente. Estos conjuntos de datos se pueden crear a mano para proporcionar casos de uso atractivos para las funciones del paquete.

La ubicación más común para los datos del paquete es el repositorio **data/**. Cada archivo que se encuentre en este directorio debe ser de tipo **.rda**. La forma más fácil de lograr esto es utilizando la función **use_data()**, en esta función es necesario especificar el nombre de la data que desea guardar, para la demostración vamos a crear una base de datos llamado familia con 3 variables:

1. **Id:** Id de la familia.
2. **Personas:** Cantidad de personas que conforman una familia.
3. **Tipo:** Tipo de Familia según la cantidad de personas que la conforman (Familia Normal, Familia Media, Familia Numerosa)

Esta base de datos es inventada, pues posee datos aleatorios de los numeros de personas en cada familia. Además el codigo que se presenta, depende del programador, en esta ocación se usa una data table, como se muestra en el siguiente código:

```
# Creacion de la base de datos Familias
install.packages("data.table")
library(data.table)
Familia <- as.data.table( c(1:200) )
setnames(Familia, c('Id'))
Familia <- Familia[ , Personas := sample(1:10, size = 200,
                                         replace = TRUE) ]
Familia[ Personas <= 4, Tipo := 'Familia Normal' ]
```

```
Familia[ Personas >= 4 & Personas <= 6 ,
         Tipo := 'Familia Media' ]
Familia[ Personas >= 7 & , Tipo := 'Familia Numerosa' ]
```

A continuación en la tabla 2.2, se encuentra los primeros datos de la base creada.

Tabla 2.2: Primeros datos de la base creada de las Familias.

ID Familia	Personas	Tipo
1	8	Familia Numerosa
2	8	Familia Numerosa
3	8	Familia Numerosa
4	3	Familia Normal
5	4	Familia Media
6	3	Familia Normal
7	8	Familia Numerosa
8	4	Familia Media
9	2	Familia Normal
10	1	Familia Normal

Elaborado: Pallasco Catota Jonathan Fernando
Fuente: Software estadístico R Studio.

Ahora simplemente debemos guardarlo en el directorio **data** usando la función **use_data()**, como se muestra a continuación:

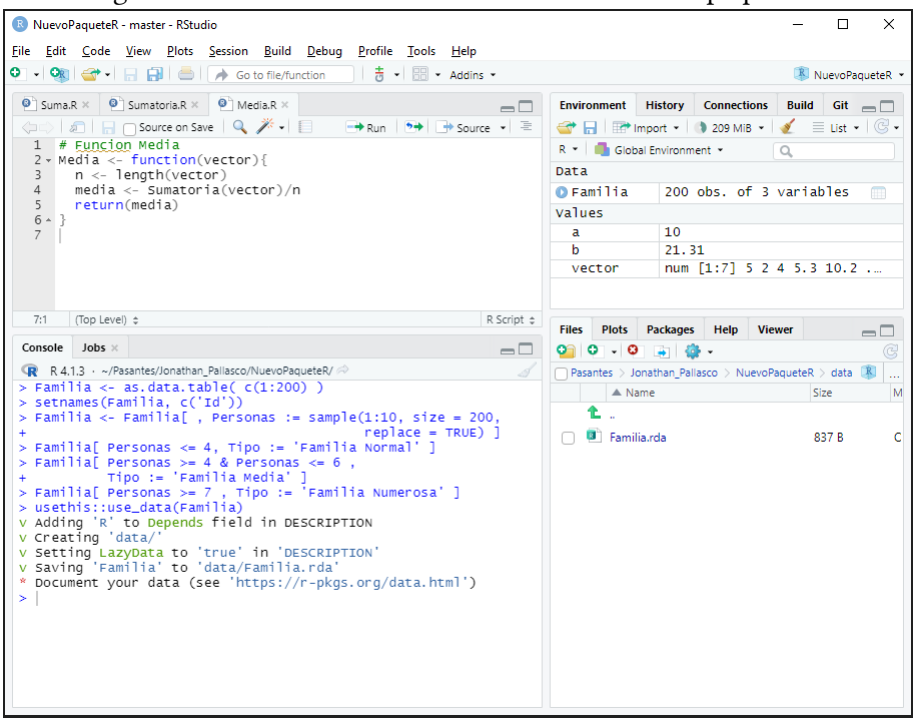
```
# Creacion de la base de datos Familias
usethis::create_data(Famila)
```

El fragmento anterior crea **data/Familia.rda** dentro de la fuente del paquete **NuevoPaqueteR** y agrega **LazyData: true** en el archivo **DESCRIPTION**. Esto hace que el objeto R **Familia** esté disponible para los usuarios de **NuevoPaqueteR** a través de **NuevoPaqueteR::Familia**

En la figura 2.8 lo antes dicho. Ahora si queremos llamar a la base de datos **Familia**, debemos nuevamente cargar la función **load_all()** y podemos utilizarla a nuestra conveniencia.

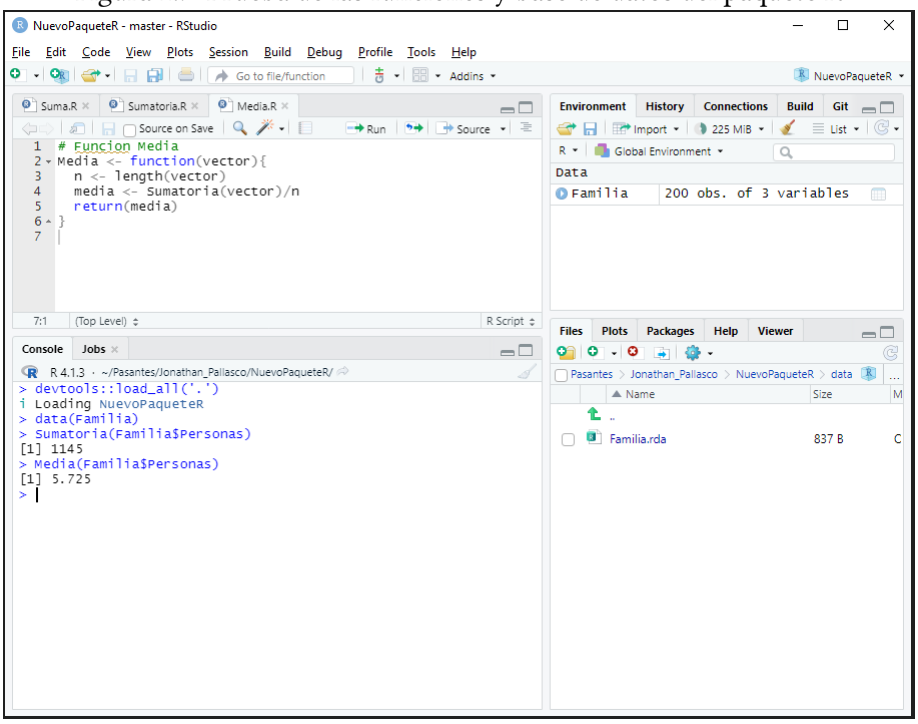
Veamos un ejemplo de como lo podemos utilizar a la base de datos junto con las funciones antes creadas, para ello usamos el código del recuadro y también lo podemos evidenciar en la figura 2.9.

Figura 2.8: Creación de la base de Datos Familia del paquete R



Elaborado: Pallasco Catota Jonathan Fernando
Fuente: Software estadístico R Studio.

Figura 2.9: Prueba de las funciones y base de datos del paquete R



Elaborado: Pallasco Catota Jonathan Fernando
Fuente: Software estadístico R Studio.

```
# Prueba de la base de datos y funciones
devtools::load_all('.')
data(Familia)
Sumatoria(Familia$Personas)
Media(Familia$Personas)
```

Se corrobora que la base de datos también esta funcionando correctamente, al igual que las funciones más adelante se hablará de la documentación de las mismas.

2.5 Metadatos del paquete

Existen 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.

En esta sección, se da a conocer la estructura básica de estos archivos y algunas de sus aplicaciones simples: como el nombre, título de su paquete y quién lo escribió.

2.5.1 DESCRIPTION

Todo paquete debe tener un **DESCRIPTION**. De hecho, es la característica que define a un paquete (RStudio y devtools consideran cualquier directorio que lo contenga **DESCRIPTION** como un paquete). Para comenzar, la función **usethis::create_package()** agrega automáticamente un archivo **DESCRIPTION** básico. Esto le permitirá comenzar a escribir el paquete sin tener que preocuparse por los metadatos hasta que lo necesite.

Este archivo utiliza un formato de simple llamado DCF, el formato de control de Debian. Donde cada línea consta de un nombre de campo y un valor, separados por dos puntos. Cuando los valores abarcan varias líneas, deben usar sangria y además posee varios campos directos.

2.5.1.1 Title y Description

Los **Title** como **Description** son una fuente frecuente de rechazos por razones no cubiertas por el R CMD check. Es por esa razon que se presentan los principales consejos que deben tener estos dos campos.

- **Title:** Se describe en una línea el paquete. 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. Sangría las líneas siguientes con 4 espacios.

Otas sugerencias

- Coloque los nombres de los paquetes, el software y las API de R entre comillas simples.
- No incluya el nombre del paquete, especialmente en Title, que a menudo tiene el prefijo del nombre del paquete.
- En la descripción no se debe comenzar con: *Un paquete para* o *Este paquete hace*.

Ejemplo:

```
Title: Nuevo Paquete R de Funciones Simples
Description: Utlizar funciones simples a menudos se complican
    demasida dada esta circunstancia se emplean 3 funciones:
    'Suma', 'Sumatoria', 'Media' para realizar calculos rapidos
    sin la necesidad de escribir varias lineas de codigo.
```

Nota: Un buen título y descripción son importantes, especialmente si planea lanzar su paquete a CRAN, porque aparecen en la página de descarga de CRAN.

2.5.1.2 Author: ¿Quién eres?

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 5 argumentos de entrada:

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

Donde:

1. **Nombre:** Esta especificado por los dos primeros argumentos, given y family (Nombre, Apellido).
2. **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.
3. **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.

4. **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.)

Ejemplo:

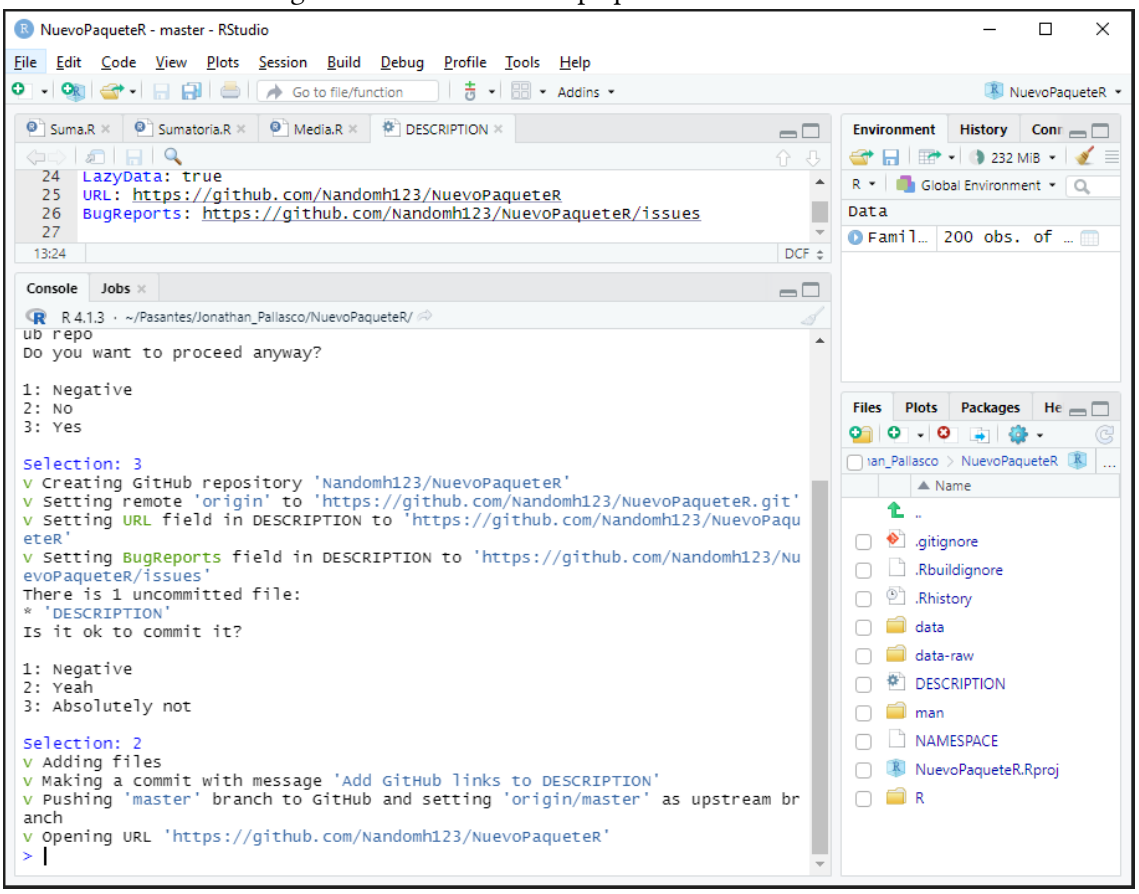
```
# Unico autor
Authors@R:
    person("Jonathan", "Pallasco",
           email = "jonathan.pallasco@epn.edu.ec",
           role = c("aut", "cre"),
           comment = c(ORCID = "0000-0003-2524-9390")

# Varios autores
Authors@R:
    c(person("Jonathan", "Pallasco",
           email = "jonathan.pallasco@epn.edu.ec",
           role = c("aut", "cre"),
           comment = c(ORCID = "0000-0003-2524-9390"),
      person("Miguel", "Flores",
           email = "jonathan.pallasco@epn.edu.ec",
           role = c("aut", "cre"),
           comment = c(ORCID = "0000-0002-7742-1247"))
```

La primera parte dice que Jonathan Pallasco es tanto el mantenedor (cre) como el autor (aut) y que su dirección de correo electrónico es jonathan.pallasco@epn.edu.ec, con id de investigador: 0000-0002-7742-1247 (ORCID).

2.5.1.3 Url y BugReports

Es una buena idea enumerar otros lugares donde las personas pueden obtener más información sobre su paquete. El campo **URL** 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.

Figura 2.10: Conexión del paquete R con el GitHub



Elaborado: Pallasco Catota Jonathan Fernando
Fuente: Software estadístico R Studio.

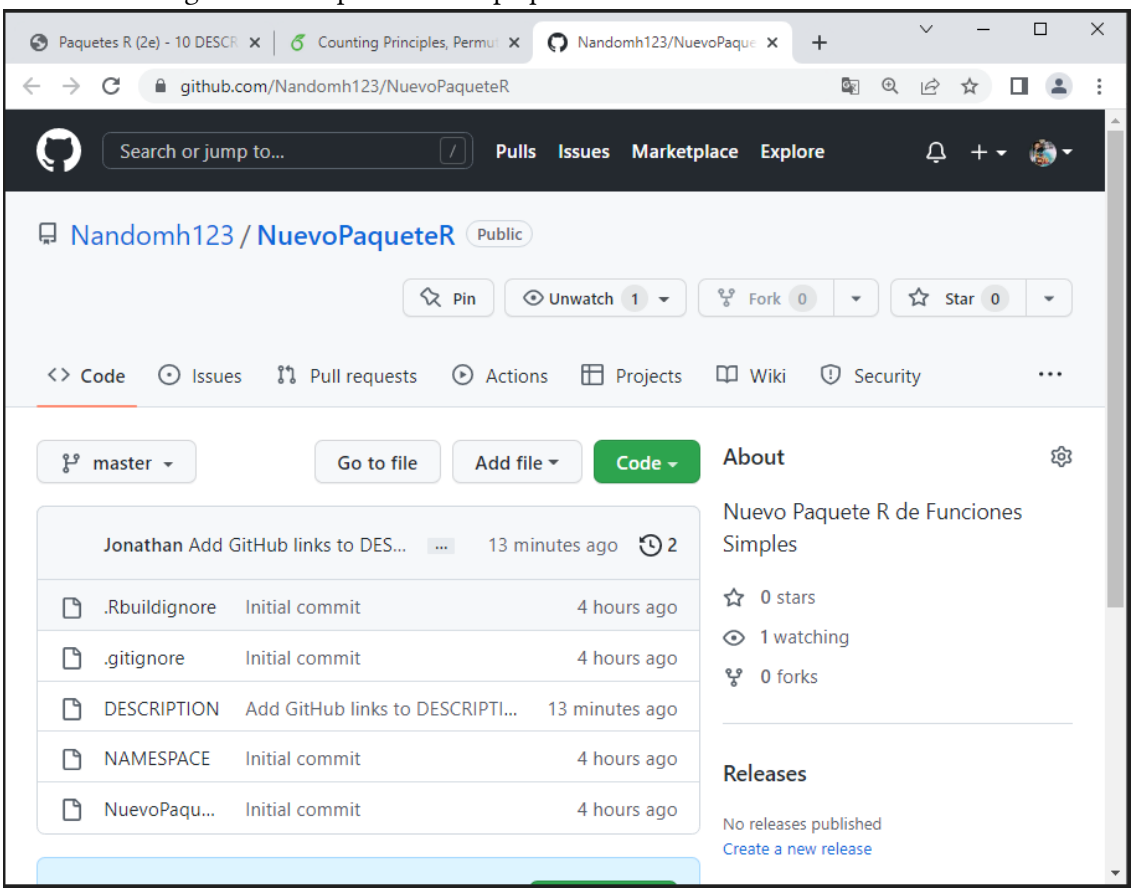
Si usa la función **use_github()** para conectar su paquete local a un repositorio remoto de GitHub, se completará automáticamente URLy BugReports. Si un paquete ya está conectado a un repositorio remoto de GitHub, simplemente debemos usar la función **use_github_links()** simplemente agregar los enlaces en el archivo **DESCRIPTION**.

por otro lado como aun no tenemos conectado nuestro paquete al GitHub, debes usar la función **use_github()** y automáticamente se crean los enlaces de **Url** y **BugReports** como se muestra a continuación.

```
93 # Obtencion de las Urls
94 usethis::use_github()
```

Ademas, en la figura 2.10, se puede vizualizar que se crearon las Urls de **Url** y **BugReports** respectivamente. En adicional, se crea automaticamente el repositorio de nuestro paquete R en nuestro usuario de GitHub personal, dependiendo del cuál este conectado tu máquina. En la figura 2.11, se observa el repositorio de del paquete R como repositorio de GitHub.

Figura 2.11: Repositorio del paquete R en nuestro usuario de GitHub



Elaborado: Pallasco Catota Jonathan Fernando
Fuente: Software estadístico R Studio.

2.5.1.4 Otros campos

Algunos otros campos del archivo **DESCRIPTION** son muy utilizados y vale la pena conocerlos:

- **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. Esto solo importa si su código tiene efectos secundarios; más comúnmente porque está usando S4.
- **Version:** 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().

2.5.1.5 Campos personalizados

Si desea enviar a CRAN, le recomendamos que cualquier nombre de campo personalizado comience con Config/. Presentamos un ejemplo de esto anteriormente, donde **Config/Needs/website** se usa para registrar paquetes adicionales necesarios para construir el sitio web de un paquete.

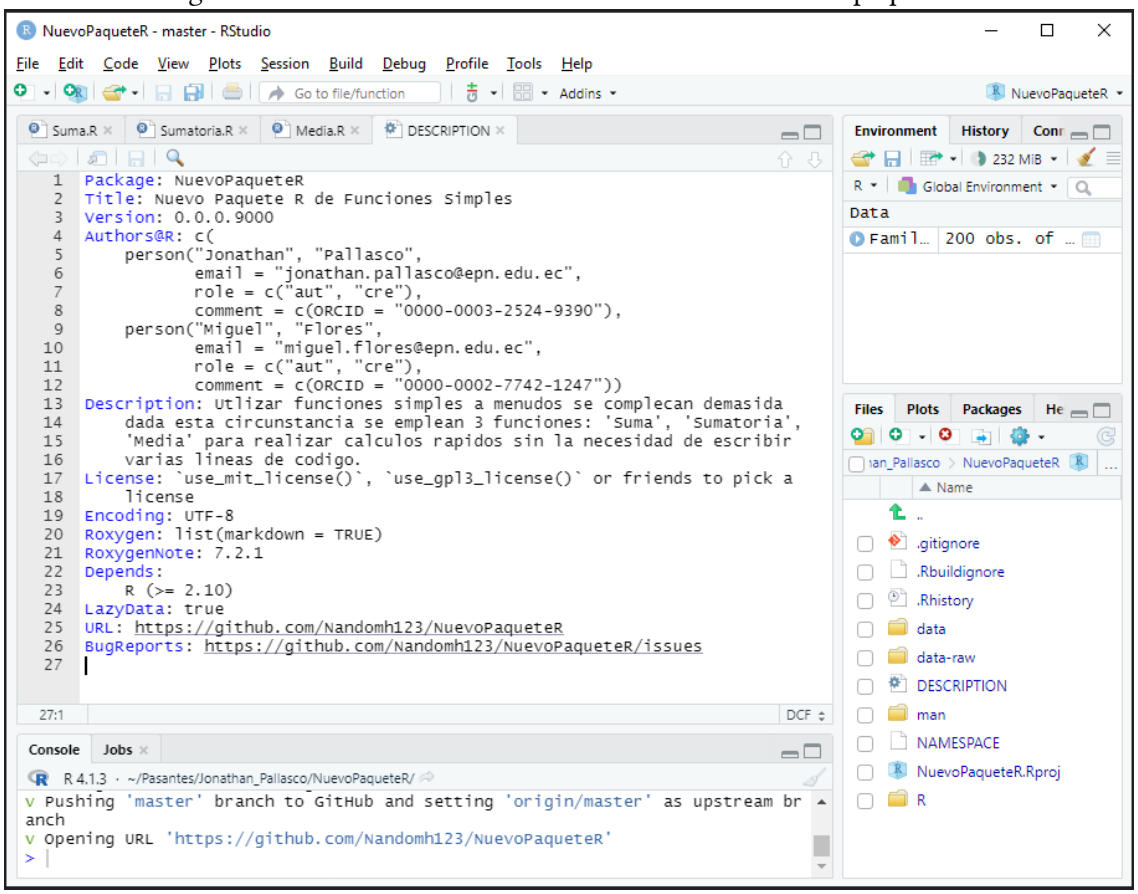
Puede notar que **create_package()** escribe dos campos más que aún no hemos discutido, relacionados con el uso del paquete **roxygen2** para la documentación.

```
95 Roxygen: list(markdown = TRUE)
96 RoxygenNote: 7.2.1
```

2.5.1.6 Archivo DESCRIPTION final

Una vez visto todos los campos y usando las funciones necesarias, junto con ejemplos para llenar algunos de los campos se presenta en la figura 2.12, una versión final del archivo **DESCRIPTION**, usando las sugerencias establecidas.

Figura 2.12: Versión final del archivo **DESCRIPTION** del paquete R



Elaborado: Pallasco Catota Jonathan Fernando
Fuente: Software estadístico R Studio.

2.5.2 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.

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.

No recomiendo escribir estas directivas a mano. En el siguiente capitulo se aprenderá cómo generar el archivo **NAMESPACE** con **roxygen2**. Hay tres ventajas principales al usar **roxygen2**:

- Las definiciones de espacios de nombres se encuentran junto a su función asociada, por lo que cuando lee el código, es más fácil ver lo que se importa y exporta.
- **Roxygen2** abstrae algunos de los detalles de **NAMESPACE**. Solo necesita aprender una etiqueta, **@export** que generará automáticamente la directiva correcta para funciones, métodos S3, métodos S4 y clases S4.
- **Roxygen2** hace ordenado al archivo **NAMESPACE**. No importa cuántas veces lo use **@importFrom(foo, bar)**, solo obtendrá un **@importFrom(foo, bar)** en su archivo NAMESPACE. Esto facilita adjuntar directivas de importación a cada función que las necesite, en lugar de tratar de administrarlas en un lugar central.

3 Documentación de las funciones y bases de datos

En este capitulo se aprenderá sobre la documentación de funciones y bases de datos, a la que se accede mediante **?** o **help()**. Base R proporciona una forma estándar de documentar un paquete donde cada función y bases de datos se documenta en un tema, 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.

3.1 Documentación de las funciones

En esta sección nos centraremos en la documentación de funciones, pero las mismas ideas se aplican a la documentación de conjuntos de datos, clases y genéricos y paquetes. Para ello se va a ser uso del paquete de R **roxigen2**.

3.1.1 Conceptos basicos del paquete R roxigen2

Para comenzar, trabajaremos con el flujo de trabajo básico de **roxigen2** y analizaremos la estructura general de los comentarios de **roxigen2** que se organizan en bloques y etiquetas.

3.1.1.1 El flujo de trabajo de la documentación

Comenzamos el flujo de trabajo de la documentación agregando comentarios de **roxygen** encima de su función. Los comentarios de roxygen son comentarios que comienzan con comilla simple '. A continuación se presenta unu ejemplo de una función con comentarios roxigen.

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

3.1.1.2 Comentarios, bloques y etiquetas de roxygen2

Como ya sabemos la estructura del flujo de trabajo básico, hablemos un poco más sobre la sintaxis. Los comentarios de **roxygen2** comienzan con #' y todos los comentarios de **roxygen2** que preceden a una función se llaman colectivamente un bloque . Los bloques se dividen en etiquetas , que parecen **@tagName** y **@tagValue**, y el contenido de una etiqueta se extiende desde el final del nombre de la etiqueta hasta el comienzo de la siguiente etiqueta. Un bloque puede contener texto antes de la primera etiqueta que se llama **introducción** . Por defecto, cada bloque generará un único tema de documentación , es decir, un archivo **.Rd** en el directorio **man/**.

A continuación presentamos algunas de las etiquetas más utilizadas: **@param**, **@return**, **@seealso**, **@example**, **@export**, etc.

3.1.1.3 Título, descripción, detalles

La **introducción** proporciona un título, una descripción y, opcionalmente, detalles para la función:

1. El **título** Está tomado de la primera oración. Debe escribirse en mayúsculas y minúsculas, no terminar en un punto y debe ir seguido de una línea en blanco. El título se muestra en varios índices de funciones y es lo que el usuario verá normalmente cuando explore múltiples funciones.
2. La **descripción** está tomada del siguiente párrafo. Se muestra en la parte superior de la documentación y debe describir brevemente las características más importantes de la función.
3. Los **detalles** adicionales son cualquier cosa después de la descripción. Los detalles son opcionales, pero pueden tener cualquier longitud, por lo que son útiles si desea profundizar en algún aspecto importante de la función.

A continuación se presenta un ejemplo de como se debe documentar la descripción, no es necesario agregar etiquetas como: **@title**, **@description**, etc.

Ejemplo:

```
#' Add
#'
#' Add together two numbers.
#'
#' This function is set to quickly calculate basic
#' operations.
```

3.1.1.4 Argumentos

3.2 Documentación de las bases de datos

4 Lista de acrónimos y abreviaturas

4.1 Acrónimos

IESS: Instituto Ecuatoriano de Seguridad Social.

SGO: Seguro General Obligatorio.

DAIE Dirección Actuarial, de Investigación y Estadística del IESS.

OIT: Organización Internacional del Trabajo.

4.2 Abreviaturas y símbolos

Seguro IVM: Seguro de invalidez, vejez y muerte del IESS.

PEA: Población económicamente activa.

SBU: salario básico unificado establecido por el Ministerio de Trabajo.

USD: dólares de los Estados Unidos de Norteamérica, como unidad monetaria.

Bibliografía