Análisis y Tratamiento de Datos con R

Con ejemplos e ilustraciones

Primera Edición

Diego Paul Huaraca S.
MS-PLUS, Inc.

MS+
Soluciones
Matemáticas

Un aporte de Source Stat Lab Ecuador a la sociedad.

Índice general

1

Introducción

Hace varios años en el área de la Matemática y la Estadística existía un claro concenso acerca del lenguaje de programación que se debía enseñar en las aulas universitarias, este era `Pascal`. Pues se le consideraba un lenguaje sencillo y al mismo tiempo elegante, en la actualidad son pocas las personas que consideran que Pascal sea adecuado debido a sus deficiencias como: la ausencia de modularidad, la falta de paradigmas de programación (OOP[1]), el desarrollo de software libre, etc.

En la Estadística, aprender a programar es un paso importante para acercarse a la comprensión de la información. Por esta razón, creo firmemente que la programación es una habilidad vital para todos los que trabajamos analizando datos.

En Ecuador, el `lenguaje de programación R` no goza aún de un amplio reconocimiento en las aulas universitarias, sin embargo, mi interés me ha llevado a usar el programa desde el año 2010 por tres propiedades esenciales:

- **Reproducibilidad:** Capacidad para recrear un análisis pasado.
- **Automatización:** Capacidad para volver a crear un análisis cuando han surgido cambios en los datos.
- **Comunicación:** El código es solo texto, por lo que es fácil comunicar. Esto hace simple conseguir ayuda de usuarios de todo el mundo.

1.1. Lenguaje R

Para introducirnos en el *mundo del lenguaje R* debemos tener claro la doble naturaleza de *entorno* y *lenguaje de programación*, pues integra un conjunto de herramientas y un lenguaje de comandos para la manipulación de datos, la realización de cálculos y análisis estadísticos, así como su representación gráfica en alta calidad y la reportería dinámica.

Figura 1.1: Logo del proyecto R

[1] Programación orientada a objetos

El lenguaje R se encuentra enmarcado como un software estadístico flexible, potente y profesional que se distribuye *libremente* bajo licencia GNU (General Public License), y en la actualidad es muy utilizado por la comunidad científica debido a su doble naturaleza, como lenguaje de programación y como entorno.

La libertad de la licencia GNU permite al usuario:

- Ejecutar el programa para cualquier propósito;
- Estudiar el funcionamiento del programa y adaptarlo a las necesidades, pues se dispone del código fuente;
- Redistribuir copias del programa;
- Mejorar el programa y liberar las mejoras al público.

Dadas sus características el lenguaje R tiene gran potencial para ser utilizado en diferentes áreas de la estadística, finanzas, simulación, reportería dinámica, biomatemática, minería de datos, big data, etc., y puede ser instalado en diversas plataformas y sistemas operativos tales como: Windows, Linux, Mac OS X y Unix.

1.2. Historia

R inició como un proyecto experimental para utilizar métodos de Lisp[2] en la construcción de un pequeño banco de pruebas que sirvan para evaluar posibles construcciones de entornos estadísticos. Desde el inicio del proyecto se decidió usar la sintaxis del lenguaje S[3]. Como consecuencia, la sintaxis del lenguaje R es similar al lenguaje S, pero la semántica, que aparentemente es parecida a la de S, en realidad es sensiblemente diferente, sobre todo en los detalles un poco más profundos de la programación.

Ross Ihaka inicia el proyecto R tras haber obtenido acceso a cierta información importante sobre el lenguaje S, misma que fue publicada por John Chambers y Rick Becker. En corto tiempo, Ross nota las similitudes existentes entre S y Scheme[4], en su afán de mostrarse ante Alan Zaslavsky[5] le propone indicar el uso del ámbito léxico para la obtención de variables propias. Sin ninguna copia de Scheme trata de mostrarle usando S, sin embargo sus intentos fracasan debido a las diferencias entre las reglas de asignación que tiene S y Scheme, esto le llevó a darse cuenta que S requería ciertas funcionalidades extras para lograr convertirse en un completo entorno estadístico.

Tiempo más tarde, Ross Ihaka y Robert Gentleman llegan a ser colegas en la Universidad de Auckland[6], y ambos muestran interés en temas de Estadística Computacional. Como profesores del Departamento de Estadística ven la necesidad de mejorar un ambiente del laboratorio de computación e inician su trabajo con la visión de crear un lenguaje similar al S pero con más funcionalidades, mismas que ya fueron identificadas previamente por Ross en su intento de incluir los ámbitos léxicos.

[2]Lenguaje de programación multiparadigma creado en 1958 por el MIT.
[3]Lenguaje de programación estadístico comercial desarrollado en Bell Laboratories.
[4]Lenguaje de programación desarrollado en 1970 por el MIT.
[5]Alan M. Zaslavsky, PhD, profesor de Estadística del Departamento de Cuidado de la Salud en la Escuela Médica de Harvard.
[6]Universidad pública situada en Auckland. Fue fundada en 1883 como parte de la Universidad de Nueva Zelanda.


Figura 1.2: Ross Ihaka & Robert Gentleman

El desarrollo del lenguaje R como tal inició en el año de 1991, el progreso fue bastante bueno por lo que para Agosto de 1993 decidieron colocar algunas copias binarias del lenguaje R en Statlib[7] y a su vez anunciaron al público el lanzamiento de la versión alfa del programa por medio de la lista de noticias de S (*S news mailing list*). De manera sorpresiva un gran número de personas probaron el nuevo lenguaje y ofrecieron su retroalimentación sobre la versión que habían liberado, entre ellos el más persistente fue Martin Machler[8] que los animó a liberar el código fuente de manera que R se distribuya bajo licencia GNU General Public License, por lo cual en Junio de 1995 aparece la primera versión libre (*open source*), el interés por el lenguaje R creció rapidamente al punto que para Marzo de 1996 fue necesario crear la propia lista de noticias y un año más tarde se tuvo que reemplazar por listas específicas: R-announce, R-help & R-devel, esto debido a la gran cantidad de consultas que realizaban los usuarios en varios temas relacionados.

Para a mediados de 1997 se estableció el `R Core Group` o `R Core Team`, un grupo de desarrolladores talentosos y experimentados con permisos para administar el código fuente del lenguaje R, en sus inicios lo conformaban:

- Ross Ihaka
- Robert Gentleman
- Martin Machler
- Doug Bates
- Peter Dalgaard
- Kurt Hornik
- Friedrich Leisch
- Thomas Lumley
- Paul Murrell
- Heiner Schwarte
- Luke Tierney

Entre sus tareas iniciales se encontraba realizar los cambios en el código fuente de acuerdo a los bugs reportados por los usuarios, todas las tareas se realizaban de manera voluntaria pues el grupo se formó sin fines de lucro, en la actualidad lo conforman alrededor de 20 personas que se encuentran en 11 diferentes cuidades del mundo.

Se tuvo que esperar hasta el 29 de Febrero del 2000, para considerar que el software se encontraba completo y estable para liberar la versión 1.0.0. Desde entonces el programa se distribuye gratuitamente a través del repositorio `Comprehensive R Archive Network` (CRAN) en `http://www.r-project.org`, su mantenimiento se encuentra a cargo del grupo R Core Team desde el año 1997 asistido por una gran cantidad de colaboradores internacionales.

[7] Sistema para la distribución de software estadístico vía electrónica.
[8] Profesor de Estadística Computacional, Departamento de Matemática, Escuela Politécnica Federal de Zúrich.

El repositorio CRAN es fundamental para el uso del lenguaje R, pues en el sitio se almacena el ejecutable del programa así como las librerías que permiten ampliar sus capacidades. Con la finalidad de evitar el colapso del *mundo estadístico* no se tiene una única ubicación al que todo mundo tiene acceso, el CRAN se *refleja* en diferentes lugares de todo el mundo, de esta manera como residente de Ecuador podría acceder a una ubicación CRAN en Ecuador[9], mientras que si se encuentra en un país que no tiene repositorio lo más recomendable es acceder a la copia del repositorio de un país cercano. Los países desarrollados como: EEUU, Alemania, Francia tienen múltiples CRAN's para abastecer a los usuarios. *La filosofía básica es elegir un repositorio que se encuentre geográficamente cercano al usuario.*

R es considerado la versión libre del programa comercial S-Plus, el cual fue desarrollado para AT&T Bell Laboratories por John M. Chambers y colaboradores en el año 1988, aunque son evidentes las diferencias entre R y S, la gran mayoría del código escrito para S funciona sin inconvenientes en R. En la actualidad el lenguaje S es distribuido por la empresa TIBCO bajo el nombre de S-PLUS.

El mayor inconveniente para los nuevos usuarios de R es adaptarse a su interfaz gráfica que para muchos es *poco amigable*, en el sentido que se deben tipear las funciones predefinidas o a su vez programar las funciones nuevas a diferencia de programas clásicos como SPSS, STATA, Minitab, etc., en los cuales se tienen botones o ventanas que despliegan opciones de análisis para el usuario.

Dada la popularidad que ha adquirido el lenguaje en los últimos años se han desarrollado varias interfaces gráficas libres GUIS's (Grafical User Interface) con el fin de volver más amigable la interacción con el usuario. Entre las interfaces más populares y utilizadas por la comunidad se encuentran:

- RStudio
- Rattle
- Deducer
- RKWard
- Eclipse
- Knime
- R Analytic Flow
- JGR
- ESS
- Red -R
- R Commander

En el tercer capítulo del libro analizaremos con detalle las interfaces: RStudio y R Analityc Flow.

1.3. Soporte

El lenguaje R al ser un software libre carece de soporte técnico, sin embargo, debido a la gran cantidad de usuarios que ha adquirido en los últimos años han aparecido empresas que proveen varios tipos de soportes bajo pago. Entre las más destacadas se encuentran:

- RStudio, Inc.
- Revolution Analytics, Inc.
- XL - Solutions Corporation.
- Quantide, Inc.
- R-Plus
- Statconn

De entre las empresas antes mencionadas se destaca de sobremanera Revolution Analytics[10], Inc. por haber optimizado el código fuente del lenguaje R lo cual ha permitido que el programa sea multicore, es decir, a través de estas modificaciones se logró aprovechar al máximo la

[9] En Ecuador se cuenta con el CRAN de la Escuela Politécnica del Litoral

[10] Revolution Analytics fue fundada en el año 2007 con la finalidad de dar soporte comercial al software Revolution R, adicionalmente provee componentes tales como: ParallelR, RevoScaleR, RevoDeployR, etc.

funcionalidad de todos los núcleos de los procesadores (16 núcleos para procesadores Core i7, etc.). El impacto inmediato del multicore se ve reflejado en la mayor velocidad de procesamiento de información permitiendo de este modo trabajar con grandes cantidades de datos sin ningún problema, así como también en la reducción del tiempo de ejecución.

Figura 1.3: Revolution R Open

El programa optimizado es 100 % compatible con el software relacionado a R (paquetes, interfaces, etc.) y puede obtenerse desde el sitio web `http://www.revolutionanalytics.com` como `Revolution R Open` de forma gratuita bajo licencia GPL, sin embargo en el caso de requerir soporte se puede adquirir la versión pagada `Revolution R Enterprise`.

1.4. Funcionamiento

El programa R es un lenguaje orientado a objetos (OOP[11]) diseñado en un entorno auténtico bajo el cual esconde su simplicidad y flexibilidad, lo cual permite a sus usuarios añadir funcionalidad mediante la definición de nuevas *funciones*. El término *orientado a objetos* hace referencia a un paradigma de la programación que emplea objetos en sus interacciones y diseño de aplicaciones. R almacena sus variables, datos, funciones, resultados, etc., en la memoria activa del computador en forma de objetos con un nombre específico y pueden ser modificados o manipulados por el usuario mediante operadores y funciones.

El hecho que R sea un lenguaje de programación puede desmotivar a muchos usuarios, los cuales piensan que para iniciarse en el programa se necesita *alma de programador* lo cual no es cierto. Primero R es un lenguaje interpretado similar a Java, y segundo no es un lenguaje compilado a diferencia de C, C++, Fortran, Pascal, ect. sino más bien mediante comandos ingresados por teclado los cuales se ejecutan directamente sin necesidad de construir archivos ejecutables. En su mayoría R se encuentra programado en C++, Python y Fortran, esto implica que R tiene la capacidad de interpretar código externo mediante el uso de ciertas librerías. La utilidad básica de lo anterior se encuentra en elaborar scripts, por ejemplo en C++, que emplean menor tiempo de ejecución.

El programa R incluye 8 bibliotecas o paquetes estándar, sin embargo, las capacidades de R pueden ser ampliadas fácilmente mediante la incorporación de paquetes que se encuentran disponibles en varios repositorios como:

- CRAN
- BioConductor
- Github
- Omegahat
- MRAN
- RForge, entre otros.

[11]La programación OOP está basada en varias técnicas: herencia, clasificación, identidad, polimorfismo y encapsulamiento.

Los paquetes estándar pueden ser visualizados a través del comando:

```
search()

##  [1] ".GlobalEnv"        "package:knitr"     "package:stats"
##  [4] "package:graphics"  "package:grDevices" "package:utils"
##  [7] "package:datasets"  "package:methods"   "Autoloads"
## [10] "package:base"
```

En la actualidad[12] existen 7190 paquetes válidos en el repositorio CRAN, que se encuentran ordenados por fecha de publicación o alfabéticamente agrupadas en diversas líneas de investigación.

1.5. Ventajas

Entre las principales ventajas que posee el software R podemos anotar lo siguiente:

- Al tratarse de un software libre el costo es nulo.
- Se han implementado una gran cantidad de métodos estadísticos desde los más básicos hasta los más avanzados y modernos. Todos los métodos se encuentran organizados en librerías, que se encuentran en constante crecimiento.
- Capacidad para acceder a datos de múltiples formatos. En la actualidad existen varias librerías para leer datos desde SPSS, SAS, STATA, MySQL, Excel, etc.
- Gran capacidad para la manipulación de datos y funciones, así como para la generación de gráficos de alta calidad.
- Facilidad para enlazarse con LaTeX y generar reportes dinámicos.
- Amplia bibliografía tanto en internet como en libros publicados por prestigiosas editoriales como: Springer, Wiley, O'Reilly, Chapman & Hall/CRC, etc.
- Fácil visualización e interpretación de los algoritmos implementados en R con lo cual el usuario puede conocer exactamente lo que el ordenador ejecuta.
- Permite visualizar los algoritmos en él implementados, modificarlos y ajustarlos a nuestras necesidades (esto no es permitido en los softwares licenciados).
- Una gran comunidad de usuarios, lo cual permite obtener ayuda de manera fácil de expertos por medio de la R help mailing list, entre otros.

1.6. Desventajas

Los inconvenientes a los cuales se deben enfrentar los usuarios de R son:

- Al ser un programa libre carece de un departamento de atención al cliente al cual se pueda recurrir en caso de que se reporte un inconveniente con el mismo. Sin embargo, existe una comunidad en crecimiento de usuarios de R que se encuentran dispuestos a colaborar desinteresadamente en la resolución de problemas.

[12] Información obtenida al 21 de Septiembre 2015.

- El software R como tal no dispone de una interfaz amigable para el usuario, las tareas se llevan a cabo a través de líneas de comando lo cual puede resultar difícil para el usuario común. No obstante con el desarrollo de GUI's se ha facilitado en gran medida la experiencia del programa con el usuario común.
- El código en R es interpretado, no compilado, lo cual puede ocasionar una ejecución lenta en ocasiones en las que se realizan simulaciones intensas. Con el fin de remediar lo anterior el grupo R Core Team a partir de la versión 2.14 ha precompilado todas las funciones y librerías de R con el objetivo de acelerar la ejecución.
- R no es particularmente un lenguaje de programación rápido, si a eso sumamos que muchos usuarios escriben pobremente su código, obtenemos como resultado un funcionamiento lento.

2

Instalación y actualización

En esta sección mostraremos los pasos a seguir para la instalación, actualización del programa y sus paquetes. Asumiremos que el usuario se encuentra trabajando sobre un computador con sistema Windows o Mac, sin embargo, no existe mayor diferencia en los pasos a seguir en el caso que el usuario use un sistema operativo o plataforma diferente.

2.1. Instalación de R

La instalación del programa R se describe en los siguientes pasos:

1. Accedemos al sitio web del R Project a través de un navegador tecleando la dirección: `http://www.r-project.org`.

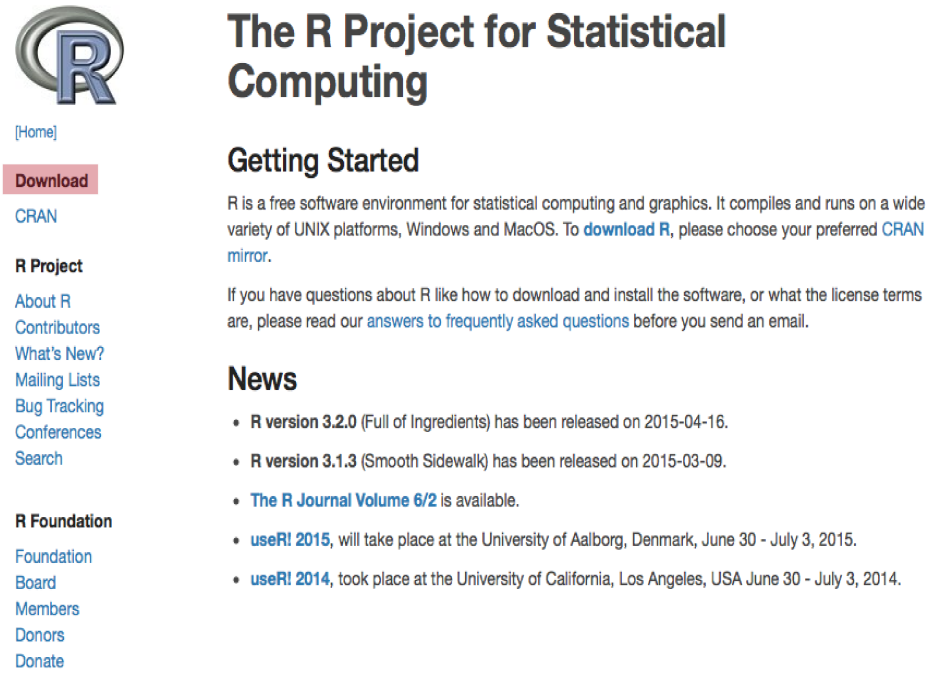


Figura 2.1: Página web R-Project

2. Nos dirigimos al enlace de la CRAN ubicado en la parte superior izquierda y elegimos un repositorio cercano a nuestra localidad. En nuestro caso seleccionamos el repositorio de la Escuela Politécnica de Litoral.

Figura 2.2: Repositorio Espol

3. En la parte superior de la página nos dirigimos al enlace `Download and install R`, seguido elegimos el sistema operativo.

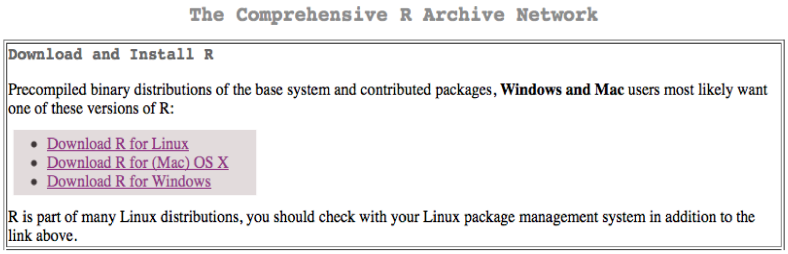


Figura 2.3: Selección Sistema Operativo

4. Elegimos el subdirectorio `base`.

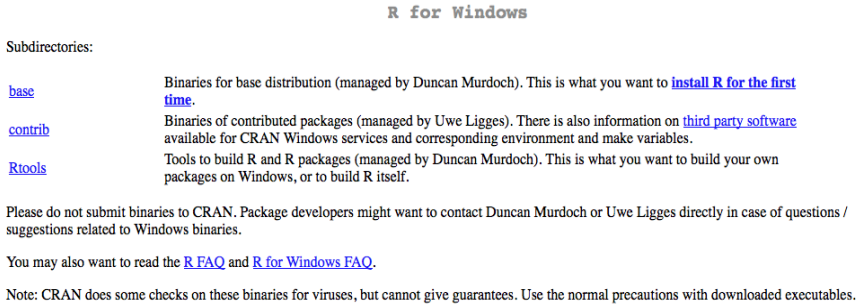


Figura 2.4: Subdirectorios

5. Finalmente, damos click sobre `Download R X.Y.Z for Windows`, con lo cual descargamos la versión de R más reciente.

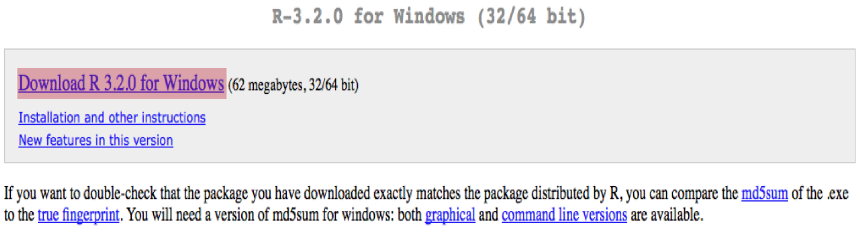


Figura 2.5: Descarga programa

En la actualidad, el software R puede ser instalado incluso sobre la plataforma Android[1], esto permite que muchos usuarios puedan tener acceso al programa en todo momento con tan sólo contar con un smartphone operativo.

[1] `https://play.google.com/store/apps/details?id=com.appsopensource.R&hl=es_419`

2.2. Entorno de trabajo

Es muy importante para el usuario de R, conocer su entorno de trabajo, y es precisamente lo que trataremos en esta sección. Antes de iniciar, debemos aclarar que el presente documento está enfocado en la plataforma `Windows`, sin embargo en los diferentes sistemas operativos no existe mayor diferencia.

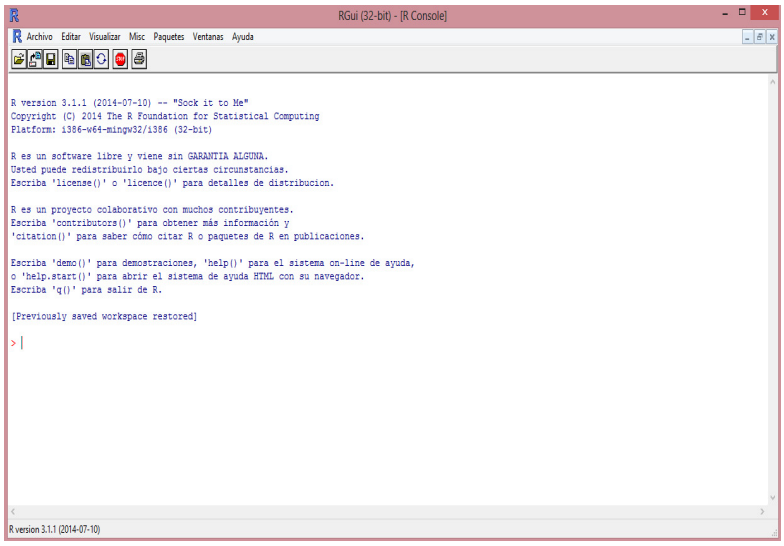


Figura 2.6: Entorno de trabajo de R

Iniciaremos explicando la barra de herramientas, la misma que consta de las secciones: Archivo, Editar, Visualizar, Misc, Paquetes, Ventanas y Ayuda. Todas serán explicadas con detalle a continuación:

- **Archivo:** En esta sección se podrá tratar todo lo relacionado con el manejo de archivos y salir del programa. Los elementos que lo componen son los siguientes:

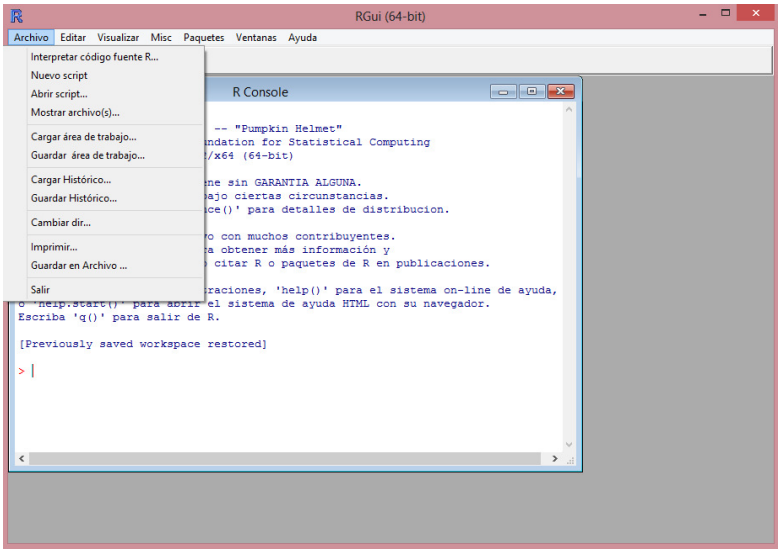


Figura 2.7: Sección Archivo

- **Interpretar código fuente R:** Hay que recordar que R es un lenguaje de programación derivado del S, es por ello que podemos realizar programas en un editor externo,

con esta opción podemos ejecutar dicho programa en la consola siempre y cuando, la extensión de guardado sea del tipo .R.

- **Nuevo script:** Si lo ejecutamos, se nos abrirá un editor de lenguaje R para crear script que luegos podemos llamar desde la consola, el tipo de archivo que podemos generar son R o S.
- **Abrir script:** Los script que hemos generados o que tengamos del tipo R o S podemos abrirlos en un editor para poder editarlos.
- **Mostrar archivo(s):** Siver para abrir y poder visualizar o editar cualquier archivo, es interesante para los que tengan relación con el lenguaje R o S. Se diferencia con la opción Abrir script de que éste no puede editarlos directamente, simplemente visualizarlos. Se pueden abrir varios archivos a la vez.
- **Cargar área de trabajo:** Pues como su nombre indica, sirve para cargar un área de trabajo que hayamos configurado y guardado previamente, es útil por ejemplo, cuando hemos definido el tipo de letra, el espacio de trabajo, los colores, etc, y queremos usarlo. La extensión común es .RData aunque también puede cargarse entornos de trabajo con formato antiguo tipo .rda.
- **Guardar área de trabajo:** Cuando se haya configurado el entorno de consola de R, colores, tipo de letra, tamaño, etc, podemos guardarlo para cargarlo posteriormente en futuras aplicaciones ya que el propio programa R no guarda, de momento, dicha configuración. El formato del entorno de trabajo es .RData.
- **Cargar Histórico:** Podemos cargar el archivo de comandos que se hayan ejecutado en una sesión previamente guardada. El formato de salida es .history.
- **Guardar Histórico:** Con él, podemos guardar los comandos que hayamos instroducido por consola en una sesión. El formato de guardado es .history.
- **Cambiar dir:** Podemos configurar el directorio de trabajo que está definido por defecto cuando se instaló el programa.
- **Imprimir:** Podemos configurar e imprimir el entorno de trabajo de R, la consola.
- **Guardar en archivo:** Se guardará todo lo que se haya escrito por la consola en formato .txt que después podremos recuperar.
- **Salir:** Sirve para salir del programa R, antes nos preguntará si queremos guardar el área de trabajo.

- **Editar:** Es la sección encargada de editar la consola de R, así como de configurar el entorno de trabajo.

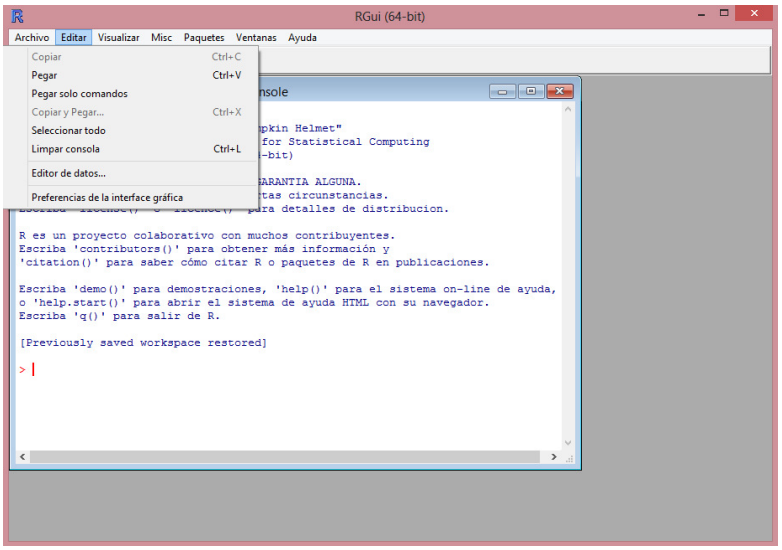


Figura 2.8: Sección Editar

- **Copiar:** Podemos copiar al portapapeles el comando o sentencia o lo que queramos de la consola. Para acceder de forma rápida por teclado se debe pulsar: Ctrl + C.
- **Pegar:** Podemos pegar en la consola aquello que tengamos en el portapapeles. Para acceder de forma rápida por teclado se debe pulsar: Ctrl + V.
- **Pegar solo comandos:** La diferencia básica con la opción Pegar es que con esta opción, sólo se pegarán en la consola aquello que sea comandos para ejecutarse.
- **Copiar y Pegar:** Todo aquello que copiemos, directamente se pegará, de forma inmediata, en la consola. Para acceder de forma rápida por teclado se debe pulsar: Ctrl + X.
- **Seleccionar todo:** Como su nombre indica, se selecciona todo lo que esté presente en la consola.
- **Limpiar consola:** Se borrarán todo lo que esté presente en la consola, pero ojo, no se borrarán las variables y estructuras definidas. Es útil cuando tenemos en la consola mucha información que ya no es útil. Para acceder de forma rápida por teclado se debe pulsar: Ctrl + L.
- **Editor de datos:** Podemos definir datos (vectores, estructuras, funciones, etc) y guardarlos para posteriormente llamarlos en la consola. Estos datos estarán definidos en la consola cuando se guarde.
- **Preferencias de la interfaz gráfica:** En esta opción es donde podremos configurar todo lo relacionado a la visualización de texto reflejado en la consola: tamaño de letra, colores, tipo de letra, etc. También, se puede configurar el entorno de trabajo, para multiventana o una única ventana.

- **Visualizar:** Activa la visualización de algunos iconos en la parte superior tales como: Abrir script, Cargar área de trabajo, etc., mismas que ya se detallaron anteriormente.

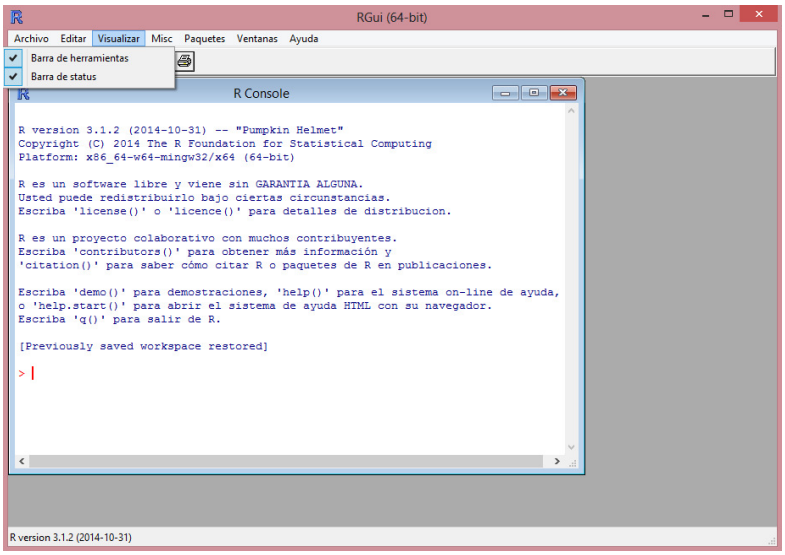


Figura 2.9: Sección Visualizar

- **Misc:** Esta es la sección denominada misceláneas, donde hay más de un control interesante.

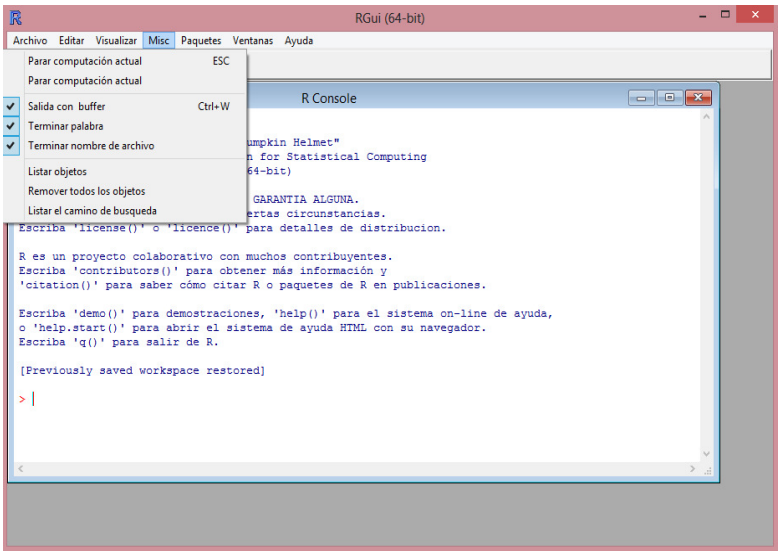


Figura 2.10: Sección Misc

- **Parar computación actual:** Este control es muy interesante y útil, con él, podremos parar cualquier archivo, sentencia o código que esté ejecutando la consola R. Para acceder de forma rápida por teclado se debe pulsar: ESC.
- **Salida con buffer:** Para acceder de forma rápida por teclado se debe pulsar: Ctrl + W.
- **Terminar palabra:** Es una ayuda interactiva, que nos ayuda a completar las palabras mientras estamos escribiendo en caso que la consola las reconozca.
- **Terminar nombre de archivo:** Realiza la misma función que la opción Terminar palabra pero en archivos.
- **Listar objetos:** Se nos mostrará por consola los objetos que hasta en este momento hemos definido en la consola.

- **Remover todos los objetos:** Como su nombre indica, elimina de memoria todos los objetos que hayamos definido (datos, variables, matrices, vectores, etc) en la consola de R. Cuando pulsemos sobre dicha opción, el programa, nos preguntará si realmente queremos borrarlos.
- **Listar el camino de búsqueda:** Nos ofrece por consola las librerias y complementos que tenemos instalados en R.

- **Paquetes:** Dedicado al manejo de las librerías que posee el programa R.

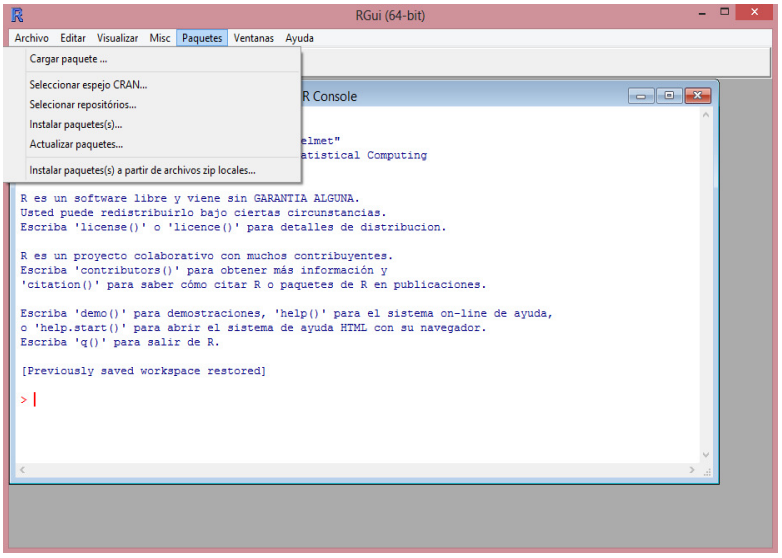


Figura 2.11: Sección Paquetes

- **Cargar paquete:** Con él, podemos cargar en la consola cualquier librería que tengamos instalada.
- **Seleccionar espejo CRAN:** Sirve para configurar el servidor de librerías.
- **Seleccionar repositórios:** En consola nos mostrará los repositorios que tenemos instalados y nos pedirá cual usar para la sesión activa.
- **Instalar paquetes(s):** Podremos actualizar o instalar librerías nuevas en red, para ello, debemos elegir un servidor.
- **Actualizar paquetes(s):** Podemos actualizar las librerías que tengamos instaladas en caso de haber una actualización reciente por red, para ello, debemos seleccionar un servidor
- **Instalar paquetes(s) a partir de archivos zip locales:** En caso de haber descargado una librería y tenerla en nuestro ordenador, podemos instalarlo mediante esta opción.

- **Ventanas:** Permite configurar la visualización de las ventanas: consola, área de gráficos y editor de script en modo vertical, horizontal, etc.

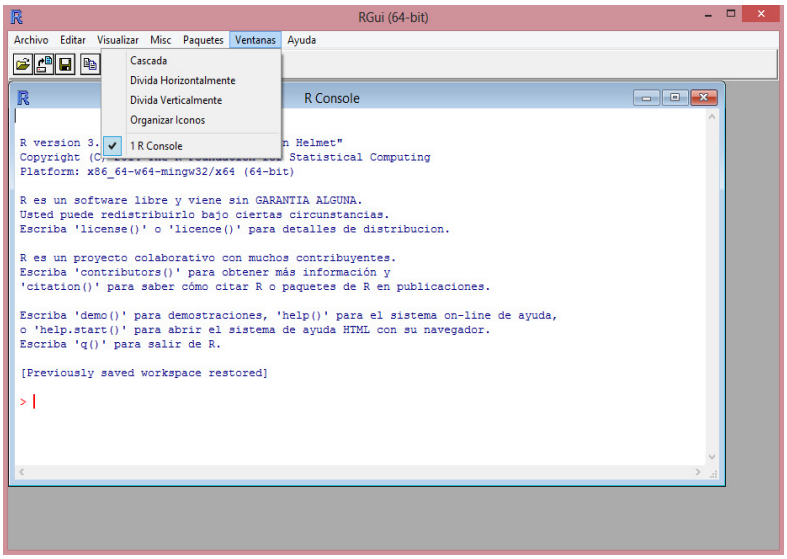


Figura 2.12: Sección Ventanas

- **Ayuda:** Es un apartado que debemos tenerlo siempre presente, ya que en él se dispone al usuario, los manuales de utilización y específicos de R.

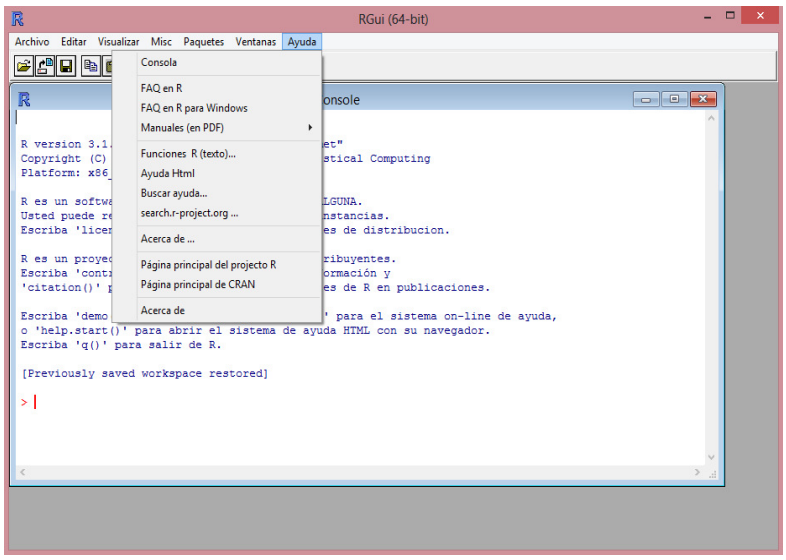


Figura 2.13: Sección Ayuda

- **Console:** Nos mostrará en una ventana los atajos por teclado que posee la consola.
- **FAQ en R:** Ayuda en html que se carga desde nuestro ordenador, donde nos ofrece las preguntas frecuentes que se suelen hacer los usuarios al utilizar R. No hace falta estar conectado a Internet.
- **FAQ en R para Windows:** Preguntas frecuentes para los usuarios de Windows.
- **Manuales en PDF:** Una recopilación de los manuales para el manejo del programa, esta opción es bastante interesante.
- **Funciones R(texto):** Es una opción donde podremos introducir sentencias de comando para obtener ayuda sobre ellas, es bastante útil.
- **Ayuda Html:** Se nos abrirá en el navegador una ayuda interactiva, no es necesario estar conectado a Internet.

- **Html search page:** Consiste en una página html donde podremos buscar las instrucciones que queramos consultar.
- **Página principal R:** Es un enlace directo a la web del proyecto R.
- **Página principal de CRAN:** Enlace directo al directorio de librerias de R.
- **Sobre:** Indica información de la compilación que se tenga instalada de R.

2.3. Instalación de paquetes

Como habíamos señalado en la sección anterior, una de las principales ventajas de R es su capacidad para incrementar su funcionalidad mediante la incorporación de nuevos paquetes o librerías.

Para entender de mejor manera la necesidad de emplear paquetes en R usaremos la siguiente metáfora: Imaginemos que tener instalado el programa en el computador es como haber adquirido un auto nuevo el cual cumple con ciertas funciones principales y no trae contratiempos, una vez que lo empezamos a rodar nos vamos dando cuenta de la necesidad de mejorar el rendimiento de ciertos mecanismos ya existentes del auto debido que lo vamos exponiendo a diferentes ambientes y condiciones. Lo anterior pasa exactamente con R, cuando descargamos e instalamos el sotfware contamos con ciertas características y paquetes básicos pero existen miles de paquetes adicionales que pueden ser agregados para lograr mejorar su funcionalidad y a la vez realizar cosas estupendas.

2.3.1. Repositorio CRAN

La instalación de paquetes en R desde el repositorio CRAN es sencillo, una vez conocido el nombre del paquete a ser instalado basta con introducir el siguiente comando en la consola:

```
install.packages("nombre_paquete")
```

Ciertos paquetes de R requieren la instalación de otros paquetes adicionales debido que entre estos comparten algunas funciones (*paquetes sugeridos*) para evitar tener inconvenientes con la instalación de un paquete se aconseja adicionar el parametro siguiente:

```
install.packages("nombre_paquete", dependencies=TRUE)
```

En el caso que se desee instalar un lista de n paquetes planteamos la siguiente solución:

```
paquetes <- c("pckg_1","pckg_2", ... , "pckg_n")
lapply(seq_along(paquetes), function(i){
      install.packages(paquetes[[i]], dependencies=TRUE)
      })
```

Una segunda alternativa al momento de instalar los paquetes de R consiste en obtener el archivo zipeado (.zip) desde el repositorio CRAN e instalarlo manualmente a través del menú del programa.

Una vez instalado un paquete el paso siguiente consiste en cargar las funciones y datos del mismo dentro del área de trabajo, para esto tenemos dos comandos útiles: `library()` y `require()`, sin embargo lo más recomendable es utilizar el primero de ellos como se detalla a continuación:

```
library('translate2R')

## Error in library("translate2R") :
## there is no package called 'translate2R'

require('translate2R')

## Loading required package: translate2R
## Warning message:
## In library(package, lib.loc=lib.loc, character.only=TRUE,
##          logical.return=TRUE,
##         : there is no package called 'translate2R'
```

Para mostrar la diferencia entre de los comandos anteriores trataremos de cargar el paquete `translate2R`, el mismo que no ha sido descargado previamente. Al emplear `library` observamos que nos arroja un error, mientras que al emplear `require` únicamente tenemos una advertencia. En el caso puntual que nos encontremos trabajando en un script la segunda opción nos causaría serios problemas dado que permite continuar con la ejecución del proceso sin que el usuario se percate, mientras que la primera opción alerta al usuario. Por todo lo antes mencionado la gran mayoría de usuarios R prefieren usar el comando `library`.

En el caso que se desee cargar n paquetes se tiene la siguiente solución:

```
paquetes <- c("pckg_1","pckg_2", ... , "pckg_n")
lapply(paquetes, FUN=library, character.only=TRUE)
```

Si deseamos visualizar el listado completo de los paquetes instalados podemos recurrir al comando `library()`. Además, si deseamos conocer el lugar en el cual se instalan los paquetes podemos hacer lo siguiente:

```
.libPaths()

## [1] "/Library/Frameworks/R.framework/Versions/3.1/Resources/library"
```

En algunas ocasiones los usuarios desean descargar los paquetes de R en una carpeta específica (C:/pckg_down) e instalar los paquetes en otra carpeta diferente (D:/pckg_inst), a continuación mostramos una posible solución al problema:

```
install.packages("pckgs", destdir="C:/pckg_down", lib="D:/pckg_inst")
library("pckgs",lib.loc="D:/pckg_inst")
```

Conocer la totalidad de paquetes que se encuentran albergados en la CRAN, y a su vez verificar cuáles de ellos se encuentran válidos y disponibles para su uso es posible mediante los siguientes comandos:

```
installed.packages()
available.packages()
```

En el caso que se requiera conocer los paquetes que se encuentra cargados en el área de trabajo emplearemos el siguiente comando:

```
sessionInfo()

## R version 3.1.3 (2015-03-09)
## Platform: x86_64-apple-darwin13.4.0 (64-bit)
## Running under: OS X 10.10.5 (Yosemite)
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] knitr_1.9
##
## loaded via a namespace (and not attached):
## [1] evaluate_0.5.5 formatR_1.0    highr_0.4      stringr_0.6.2
## [5] tools_3.1.3
```

El comando anterior imprime información acerca de la versión y plataforma del ejecutable de R que se encuentra utilizando, adicionalmente muestra información de la localidad y de los paquetes adjuntos o cargados.

Para ocasiones en las cuales se desee eliminar un paquete previamente instalado en R tenemos dos alternativas: la primera consiste básicamente en eliminar la carpeta que contiene dicho paquete del repositorio del computador mientras que la segunda alternativa consiste en emplear el siguiente comando:

```
remove.packages("nombre_paquete", "directorio")
```

donde:

- **nombre_ paquete:** Nombre o vector de nombres de los paquetes a ser eliminados.
- **directorio:** Es un vector de caracteres que proporcionan la dirección del directorio del cual se eliminará el paquete. Si se omite dicho parámetro se asume por default la dirección obtenida por .libPaths().

2.3.2. Repositorios externos

Habíamos comentado sobre la existencia de varios repositorios adicionales al CRAN de entre los cuales destacan R-Forge[2], BioConductor[3] y Omegahat[4], dentro de estos repositorios se pueden encontrar paquetes que no se encuentran en el CRAN, o bien, versiones más actualizadas de los mismos.

Para instalar los paquetes desde los repositorios externos antes mencionados basta la siguiente sentencia:

[2]Sistema para el desarrollo y versionamiento de paquetes en R.
[3]Subproyecto dedicado a la investigación en bioestadística.
[4]Subproyecto dedicado al desarrollo de interfaces para la programación estadística basado en Java.

```
install.packages("pckgname", repos="http://r-forge.r-project.org")
install.packages("pckgname", repos="http://www.omegahat.org/R")
install.packages("pckgname", repos="http://www.bioconductor.org/
                                    packages/2.10/bioc")
```

Algunos de los paquetes se encuentran en formato binario, para su instalación basta adicionar el parámetro `type="source"`.

2.3.3. Github

GitHub es un repositorio de archivos y proyectos el cual emplea un sistema de control de versiones conocido como Git. Github fue lanzado a inicios del 2010, el código alojado en el repositorio se almacena de forma pública aunque también se puede almacenar de forma privada siempre y cuando se tenga una cuenta de pago.

Figura 2.14: Logo Github

La instalación y sincronización de Git & Github con RStudio se realiza de la siguiente manera:

1. Descargar e instalar la plataforma específica de Git[5] (no Github) con las opciones por default.

2. Configurar Git con los comandos globales a través de la versión bash, esto permitirá que Git tenga conexión con los repositorios de Github para el envió/recepción de archivos, teclear lo siguiente:

   git config —global user.name ”nombre de la cuenta de Github”
   git config —global user.email ”Github-email@something.com”

3. Por último, abrir RStudio y configurar la ruta del ejecutable de Git:
   Ir a Tools >Options >Git/SVN.

En la actualidad gran cantidad de usuarios R han alojado sus proyectos en el sistema de control de versiones GitHub, Inc. por lo cual se hace necesario contar con una solución al momento de instalar algún paquete alojado en dicho sistema. Nuestra solución consiste en lo siguiente:

```
install.packages('devtools', dependencies=TRUE)
devtools::install_github("rstudio/rmarkdown")
```

Iniciamos instalando el paquete `devtools` el cual nos provee la función `install_github`, esta última es la encargada de acceder al sistema Github e instalar el paquete deseado, para esta ocasión hemos seleccionado el paquete `rmarkdown` alojado en el repositorio Github dentro del proyecto `rstudio`.

[5] http://www.git-scm.com/

En el caso que se requiera información acerca de un paquete previamente instalado podemos recurrir a:

```
library(help='pckgname')
```

También podemos enlistar todas las funciones que han sido implementadas dentro de un paquete como muestra el siguiente ejemplo:

```
library(foreign)
ls('package:foreign')

##  [1] "data.restore"  "lookup.xport"  "read.arff"     "read.dbf"
##  [5] "read.dta"      "read.epiinfo"  "read.mtp"      "read.octave"
##  [9] "read.S"        "read.spss"     "read.ssd"      "read.systat"
## [13] "read.xport"    "write.arff"    "write.dbf"     "write.dta"
## [17] "write.foreign"
```

Si deseamos conocer los parámetros de cada una de las funciones implementadas dentro de un paquete podemos recurrir a lo siguiente:

```
lsf.str('package:foreign')
```

Cuando un paquete es empleado dentro de una investigación surge la necesidad de citar a los autores del paquete, lo anterior lo solventamos mediante el comando `citation`.

```
citation("foreign")

## To cite package 'foreign' in publications use:
##
##  R Core Team (2014). foreign: Read Data Stored by Minitab, S, SAS,
##  SPSS, Stata, Systat, Weka, dBase, .... R
##  package version 0.8-61. http://CRAN.R-project.org/package=foreign
##
## A BibTeX entry for LaTeX users is
##
##  @Manual{,
##    title = {foreign: Read Data Stored by Minitab, S, SAS, SPSS,
##  Stata, Systat, Weka,dBase, ...},
##    author = {{R Core Team}},
##    year = {2014},
##    note = {R package version 0.8-61},
##    url = {http://CRAN.R-project.org/package=foreign},
##  }
```

2.4. Actualización de paquetes

En el caso que se desee actualizar todos los paquetes previamente instalados contamos con el comando `update.packages`, el mismo que revisa las actualizaciones de los paquetes. El anterior comando nos preguntará si deseamos actualizar uno por uno los paquetes, lo cual es tedioso pues si contamos con un gran número de paquetes nos demandará bastante tiempo aprobar la actualización de los mismos. Para subsanar lo anterior basta añadir el parámetro `ask = TRUE`.

```
update.packages(ask=TRUE)
```

En el caso que se desee conocer únicamente los paquetes que fueron instalados previamente y en la actualidad constan con un versionamiento emplearemos lo siguiente:

```
old.packages()
```

El comando anterior nos mostrará la versión del paquete que ha sido instalada y la última versión disponible del mismo.

Para finalizar la sección dejamos un comando útil para los desarrolladores que deseen conocer los autores de los paquetes, así como también las personas que dan mantenimiento a los mismos:

```
lapply(dir(R.home("library")), packageDescription)
```

2.5. Actualización de R

Mantener actualizado R y sus paquetes es relativamente fácil a través del tecleo de los siguientes comandos en la consola, los mismos que se detallan a continuación:

```
install.packages('installr', dependencies = TRUE)
library('installr')
updateR()
```

Iniciamos instalando el paquete **installr**, acto seguido procedemos a cargar el paquete a través del comando `library`, y finalmente, mediante el comando `updateR` verificamos la versión más reciente del programa y descargamos la misma.

En el caso que se desee conocer la versión de R que se encuentra instalada podemos teclear en la consola el comando: `R.version.string` o `R.version`

```
R.version.string

## [1] "R version 3.1.3 (2015-03-09)"
```

```
R.version

##                _
## platform       x86_64-apple-darwin13.4.0
## arch           x86_64
## os             darwin13.4.0
## system         x86_64, darwin13.4.0
## status
## major          3
## minor          1.3
## year           2015
## month          03
## day            09
```

```
## svn rev        67962
## language       R
## version.string R version 3.1.3 (2015-03-09)
## nickname       Smooth Sidewalk
```

2.6. Obteniendo ayuda

Existen algunas funciones en R que nos facilitan la vida al momento de indagar sobre la funcionalidad de ciertos comandos con tan sólo tener disponible una conexión de Internet. Por ejemplo, si nos encontramos interesados en obtener información acerca de la función `seq()` podemos escribir en la consola lo siguiente:

```
help(seq)
```

Una forma alternativa es

```
?seq
```

Para el caso que se encuentren trabajando con funciones especificadas por caracteres especiales, el argumento debería ir entre comillas, con el fin de transformarlo en una *cadena de caracteres*:

```
help("[[")
```

Se pueden utilizar tanto comillas simples como dobles sin que esto conlleve problemas a posteriori.

Adicionalmente, si nos encontramos interesados en buscar información relacionada al término "normal", o algún otro término en específico podemos teclear lo siguiente:

```
help.search("normal")
help.search("termino_especifico")
```

En el caso que se requiera un manual de ayuda completo en formato HMTL, empleamos el comando:

```
help.start()
```

Por último, existen circunstancias en las cuales se hace indispensable conocer todas las funciones relacionadas a un nombre en específico. Primero observamos un ejemplo en el cual se obtienen todas las funciones relacionadas con el término **mean**.

```
apropos("mean")

##  [1] ".colMeans"     ".rowMeans"     "colMeans"      "kmeans"
##  [5] "mean"          "mean.Date"     "mean.default"  "mean.difftime"
##  [9] "mean.POSIXct"  "mean.POSIXlt"  "rowMeans"      "weighted.mean"
```

Ahora observamos un ejemplo que obtiene todas las funciones que poseen la letra w en sus nombres.

```
apropos("w$")

##  [1] ".__C__raw"          "all.equal.raw"      "ar.yw"
##  [4] "as.data.frame.raw" "as.raw"             "charToRaw"
##  [7] "dev.new"            "file.show"          "grepRaw"
## [10] "is.raw"             "layout.show"        "n2mfrow"
## [13] "new"                "nrow"               "NROW"
## [16] "pat_brew"           "pat_rnw"            "plot.new"
## [19] "plot.window"        "rainbow"            "raw"
## [22] "row"                "show"               "tryNew"
## [25] "url.show"           "View"               "window"
```

Para finalizar observamos un ejemplo que encuentra todos las funciones que tiene un número entre el 4 y 8.

```
apropos("[4-8]")

##  [1] ".__C__S4"        ".TAOCP1997init" "asS4"           "enc2utf8"
##  [5] "fixPre1.8"       "Harman74.cor"   "intToUtf8"      "isS4"
##  [9] "seemsS4Object"   "state.x77"      "utf8ToInt"
```

2.7. Mostrando ejemplos

La mayoría de funciones incluyen ejemplos que pueden ser ejecutados y facilitan al usuario comprender de mejor manera su uso. Para ejecutar los ejemplos usaremos el comando `example`, como se muestra a continuación:

```
example(mean)

##
## mean> x <- c(0:10, 50)
##
## mean> xm <- mean(x)
##
## mean> c(xm, mean(x, trim = 0.10))
## [1] 8.75 5.50

example(range)

##
## range> (r.x <- range(stats::rnorm(100)))
## [1] -2.863284  2.589062
##
## range> diff(r.x) # the SAMPLE range
## [1] 5.452346
##
## range> x <- c(NA, 1:3, -1:1/0); x
## [1]   NA    1    2    3 -Inf  NaN  Inf
##
## range> range(x)
## [1] NA NA
```

```
##
## range> range(x, na.rm = TRUE)
## [1] -Inf  Inf
##
## range> range(x, finite = TRUE)
## [1] 1 3
```

3

Entornos de desarrollo

Un entorno de desarrollo integrado, también conocido como IDE (Integrated Development Enviroment) es un programa informático compuesto por un conjunto de herramientas de programación que contiene: un editor, un compilador, un depurador y un constructor de interfaz gráfica (GUI), el mismo que viene empaquetado como un programa `aplicación` que facilita de sobre manera la realización de operaciones al usuario mediante una serie de menús o mediante interacción con los objetos gráficos que aparecen en pantalla, a través de periféricos como: el ratón y teclado.

Los IDE's fueron desarrollados con el fin de proveer al usuario un marco de trabajo más amigable, a continuación describimos las IDE's más populares para los usuarios:

- RStudio
- R Analytic Flow

3.1. RStudio

RStudio es un entorno IDE de código abierto lanzado en Febrero 2011 el cual nos ofrece un marco de trabajo más amigable con el software R y lo podemos encontrar para todas las plataformas (Windows, Mac, Linux) así como también puede ser ejecutado a través de un navegador web[1].

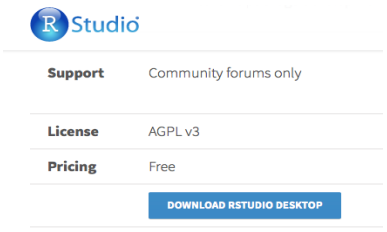


Figura 3.1: Descarga RStudio

RStudio puede ser obtenido libremente desde su página web `http://www.rstudio.org/`. Una vez obtenido el archivo ejecutable la instalación se la realiza de manera simple e intuitiva.

Una forma de descarga e instalación más técnica de RStudio puede realizarse a partir del siguiente script:

[1]Opción válida para la versión `server`.

```
# descarga e instala el paquete installr
install.packages("installr")
# cargamos el paquete installr
library(installr)
# instalamos RStudio IDE
install.RStudio()
```

RStudio ofrece una amplia integración con ficheros de diversos formatos: R scripts (.R), Markdown (.md), LaTeX (.Rnw) entre otros. La facilidad en la generación de documentos dinámicos con `RStudio` y `knitr` han hecho que el programa se convierta en la IDE preferida por muchos usuarios de R.

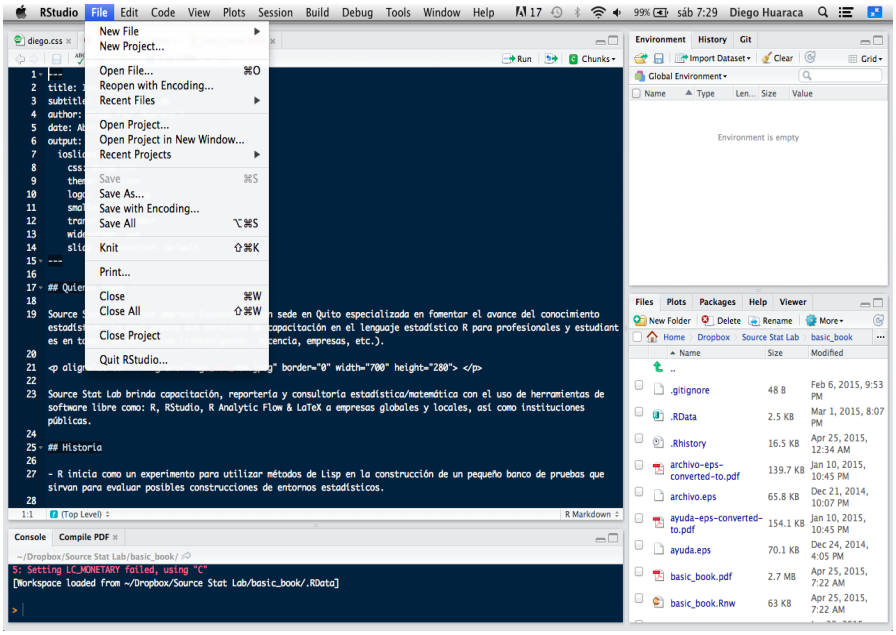


Figura 3.2: Interfaz de RStudio

El programa se encuentra organizado en cuatro ventanas de trabajo distintas:

- **Editor de código fuente:** Se encuentra en la zona superior izquierda, esta ventana nos permite abrir y editar ficheros con código R.
- **Consola:** Se ubica en la zona inferior izquierda, esta ventana es también conocida como consola y nos permite ejecutar comandos de R.
- **Navegador de objetos:** La zona superior derecha posee dos ventanas auxiliares:
  - **Workspace:** En esta ventana se enlistan todos los objetos creados en memoria.
  - **History:** En esta ventana se almacena el histórico de las líneas de código que han sido ejecutadas en R.
- **Visualización e información:** Esta última ventana ubicada en la zona inferior derecha se encuentra conformada por 4 ventanas auxiliares:
  - **Files:** Provee el acceso al árbol de directorios y ficheros del disco duro.
  - **Plots:** Ventana auxiliar en la cual aparecen los gráficos creados en la consola.

- **Packages:** Esta ventana facilita la administración de los paquetes de R instalados en el computador.
- **Help:** Esta última ventana nos ayuda en la búsqueda de información respecto a un comando en específico.

RStudio ofrece varios mecanismos para controlar varios aspectos de la evaluación durante una sesión. La función `options( )` es empleada para compartir los valores de parámetros entre las funciones.

3.1.1. Ancho de impresión

Existen ocasiones en las cuales el usuario desea controlar el ancho de impresión de los resultados que se muestran en la pantalla, como primer paso para modificar el ancho de impresión debemos obtener el parámetro actual mediante:

```
getOption("width")

## [1] 75
```

Una vez conocido el ancho de pantalla actual procedemos a modificar el mismo cambiando el valor del parámetro `width`, de la siguiente manera:

```
options(width=40)
rnorm(10)

##  [1] -0.85051649  0.11849963  1.62168905
##  [4]  1.09786611 -1.06605821 -0.04856583
##  [7] -0.27299846 -0.44725027  0.30085759
## [10]  1.43912640
```

```
options(width=55)
rnorm(10)

##  [1] -0.50210704  0.78492102  0.30163276  1.40569874
##  [5]  0.03484832 -1.17678809  0.87013044  0.40936701
##  [9]  0.19438312 -0.73466261
```

```
options(width=70)
rnorm(10)

##  [1] -0.9041028 -0.7925827  0.2517137 -0.3882113  0.4077788 -1.3609977
##  [7]  0.3372508 -2.5411556  1.3064928 -0.8011971
```

3.1.2. Prompt

Para los usuarios que deseen cambiar el simbolo `>` del prompt o interpretador por otro símbolo diferente como: `− >` o por un nombre, tenemos el siguiente código:

```
options(prompt="->")
options(prompt="diego >")
```

3.1.3. Decimales

Una preocupación adicional para los usuarios es la cantidad de decimales con la cual se muestran los resultados, dicha cantidad de decimales puede ser modificada y debe encontrarse en el rango de 1 a 22.

```
getOption("digits")

## [1] 7
```

R por default muestran los resultados con 7 decimales, sin embargo los mismos pueden ser modificados como se muestra a continuación:

```
options(digits=2)
rnorm(3)

## [1] 0.259 1.319 0.039
```

```
options(digits=5)
rnorm(3)

## [1]  1.69690  0.94364 -0.97631
```

```
options(digits=10)
rnorm(3)

## [1]  2.0575569605 -0.8964782298  0.4606099029
```

Existen opciones adicionales que pueden ser modificadas de acuerdo a las necesidades que tenga el usuario, para ver el listado completo de opciones podemos teclear en la consola el comando:

```
help(options)
```

3.1.4. Respaldando información

Un tema importante dentro del análisis de datos es el respaldo de información que se pueda dar sobre ciertos resultados obtenidos, en este punto R consta de dos comandos muy útiles: `save()` & `load()`.

El primero de ellos permite almacenar en disco los objetos que desee el usuario (almacenamiento parcial), dicho comando puede ser configurado de tal manera que almacene todos los objetos que se encuentra válidos en el área de trabajo.

```
# si deseamos guardar el objeto "datos_banco" con el nombre "base"
save(datos_banco, file = "base.RData")
# para el caso que se desee almacenar todos los objetos con el nombre "info"
save(list = ls(all = TRUE), file = "info.RData")
```

El segundo comando nos va a permitir cargar los objetos guardados en el área de trabajo actual o en un ambiente determinado.

```
# cargamos el objeto base en el area de trabajo
load("base.RData")
# ahora cargamos "info" en un ambiente determinado "env"
load("info.RData", envir = env)
```

3.2. R Analytic Flow

R Analytic Flow (RAF) es una interfaz gráfica de usuario desarrollado por Ryota Suzuki[2], que facilita el análisis de datos a través de diagramas de flujo. El software se encuentra bajo licencia BSD & GPL, por lo cual puede obtenerse de forma gratuita a través de la página web de Ef-prime, Inc. `http://www.ef-prime.com` para las plataformas: Windows, Mac OS X y Linux.

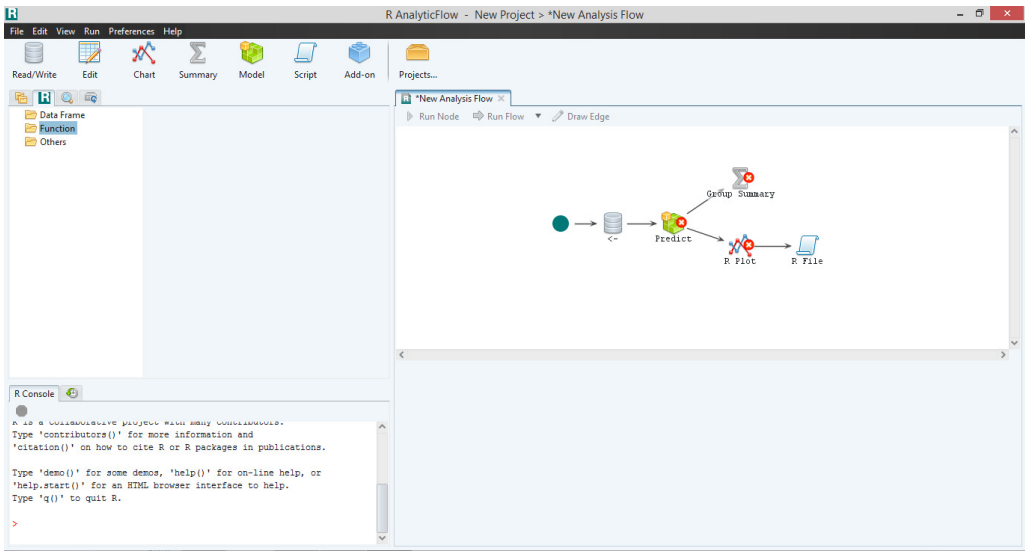


Figura 3.3: R Analytic Flow

R Analytic Flow permite insertar código de R enmarcado dentro de nodos con la capacidad de ejecutar diferentes rutinas a partir de determinadas conexiones entre nodos.

3.2.1. Ventajas

A continuación enumeramos algunas de las ventajas que posee R Analytic Flow:

1. Facilita ejecutar procesos a través de flujos.
2. Fácil implementación de tareas en cada nodo.
3. Reduce la complejidad a la hora de programar varias funciones que se relacionen entre sí.
4. El número de usuarios que usan R Analytic Flow va en aumento debido a las facilidades que presenta.

[2]Ryota Suzuki es un desarrollador de software orientado al análisis de datos, fundó con sus amigos la empresa Ef-prime, Inc. en Tokyo, además es el creador del paquete *pvclust* de R.

3.2.2. Desventajas

Algunas de las desventajas por las cuales los usuarios no usan R Analytic Flow:

1. Escasa documentación sobre el manejo de la interfaz.
2. El código fuente se encuentra administrado únicamente por Ef-prime, Inc. Esto impide que se pueda seguir optimizando la interfaz con mayor rapidez.

4

Estructura de datos

En las secciones anteriores mencionamos que R es un lenguaje de programación orientado a objetos, por lo que cualquier cosa que exista en él tales como: variables, datos, funciones, etc. son objetos. En este capítulo trataremos los diferentes tipos de objetos que R crea y manipula, incluso revisaremos estructuras de almacenamiento de datos más complejas construidas a partir de otras más sencillas.

Durante una sesión de trabajo los objetos que se crean son almacenados por nombre, para enlistar dichos objetos es necesario recurrir al comando `ls()` o su equivalente `objects()`. La colección de objetos almacenados se denomina espacio de trabajo (*workspace*).

```
# Inicialmente el area de trabajo se encuentra vacia
ls()

## character(0)

# Generamos 2 vectores
x_vec <- c(1, 2, 3)
y_vec <- c(4, 5, 6)
# Los vectores creados ya se encuentran en el area de trabajo
objects()

## [1] "x_vec" "y_vec"
```

Vale la pena advertir que el operador de asignación '<-', no es equivalente al operador habitual '='que se reserva para otro propósito, sino que consiste en dos caracteres '<'(*menor que*) y '-'(*guión*) que obligatoriamente deben ir unidos, además deben apuntar al objeto que recibe el valor de la expresión. Los usuarios principiantes de R suelen confundir estos dos conceptos por lo que en ciertas ocasiones presentan conflictos de interpretabilidad.

La asignación puede realizarse también mediante la función `assign()`, la cual es una forma equivalente a la asignación anterior.

```
# Generamos 2 vectores nuevos
assign("x_new", c(2, 4, 6))
assign("y_new", c(3, 5, 7))
# Revisamos sus elementos
x_new
```

```
## [1] 2 4 6

y_new

## [1] 3 5 7

# Verificamos los objetos del espacio de trabajo
ls()

## [1] "x_new" "x_vec" "y_new" "y_vec"
```

En ocasiones los usuarios de R suelen realizar las asignaciones apuntando la flecha a la derecha '->', realizando el cambio obvio en la asignación. Esto no es muy recomendable debido que en la actualidad la tendencia es estandarizar la forma de escritura en R.

```
# Creamos un nuevo vector
c(5,10,15) -> x_mul
# Revisamos los elementos del vector
x_mul

## [1]  5 10 15
```

Los nombres que se les asigna a los objetos en R pueden ser de cualquier longitud, y a su vez pueden combinar letras, números y caracteres especiales (coma, punto, guión bajo, etc.), la única exigencia al momento de asignar un nombre a un objeto es que el mismo inicie con una letra (R diferencia mayúsculas de minúsculas). La construcción explícita de un objeto nos proporcionará un mejor entendimiento de su estructura, y nos permitirá ahondar en algunas nociones mencionadas previamente.

Las estructuras de datos en R pueden ser organizados por su dimensionalidad (1 dimensión, 2 dimensiones o n-dimensiones), así como también por su tipo (homogéneo, heterogéneo) lo anterior da lugar a 6 tipos de estructuras que se resumen a continuación:

Dimensión	**1-d**	**2-d**	**n-d**
Homogéneo	Vector	Matriz	Array
Heterogéneo	Lista	Data Frame / Data Table	

Cuadro 4.1: Estructura de datos

En la actualidad ha tomado fuerza el uso de los data tables a diferencias de los data frames, esto debido a las diversas facilidades que poseen los primeras sobre la manipulación de grandes cantidades de información (incorporación del almacenamiento en disco).

4.1. Vectores

Es la estructura más simple de R que sirve para almacenar un conjunto de valores del mismo o diferente tipo llamados *elementos*. Existen 6 tipos de elementos que R puede almacenar dentro de un vector:

- logical
- double
- integer
- complex

- character
- raw

4.1.1. Creación

Vectores homogéneos

Para la creación de vectores recurriremos a una función interna de R, dicha función es conocida con *concatenación* y es denotada por la letra `c()`. Iniciamos mostrando al lector la creación de vectores donde todos sus elementos son del mismo tipo.

```
# Podemos crear vectores logicos tecleando TRUE y FALSE (o T & F).
logi_var <- c(TRUE, FALSE, TRUE, FALSE)
logi_var

## [1]  TRUE FALSE  TRUE FALSE

# Usamos el sufijo L, para diferenciar un numero entero
int_var <- c(2L, 4L, 7L, 5L)
int_var

## [1] 2 4 7 5

# Usamos el simbolo . para notar los decimales
dbl_var <- c(2.3, 6.8, 4.1)
dbl_var

## [1] 2.3 6.8 4.1

# Los caracteres deben ir entre comillas
chr_var <- c("statistical", "model", "test")
chr_var

## [1] "statistical" "model"       "test"
```

La función `vector` permite crear vectores de un tipo y longitud determinada, a continuación mostramos unos ejemplos:

```
# creamos un vector numerico de longitud 5
vector("numeric", 5)

## [1] 0 0 0 0 0

# creamos un vector logico de longitud 5
vector("logical", 5)

## [1] FALSE FALSE FALSE FALSE FALSE

# creamos un vector de caracteres de longitud 5
vector("character", 5)

## [1] "" "" "" "" ""
```

Observamos que la función `vector` genera vectores con sus valores por default, es decir, para el caso de vectores númericos los inicializa en 0, para el caso de vectores lógicos los inicializa en

FALSE, mientras que para los vectores de caracteres los inicializa en vacíos.

Todo vector consta de dos argumentos: `mode` & `length`. El primero de ellos especifica el tipo de elementos que almacena el vector, mientras que el segundo argumento especifica la longitud o número de elementos que tiene dicho vector.

```
# creamos un vector numerico
var <- c(3, 6, 8, 9)
# verificamos el tipo de vector
mode(var)

## [1] "numeric"

# mostramos la longitud del vector
length(var)

## [1] 4
```

Vectores heterogéneos

El usuario tiene toda la facilidad de crear un vector con diferentes tipos de elementos, sin embargo, debido al conflicto que se genera internamente por la *coerción* de los distintos elementos tenemos la siguiente jerarquía:

−						+
logical	<	**integer**	<	**double**	<	**character**

Cuadro 4.2: Flexibilidad de los elementos

La coerción se produce automáticamente, por tanto, al momento de crear un vector es importante revisar el tipo de elemento de mayor flexibilidad con el fin de conocer a que tipo de vector coercionará el resultado, revisamos algunos ejemplos:

```
var1 <- c(TRUE, 3L, FALSE, 5L)
mode(var1)

## [1] "numeric"

var2 <- c(2+5i, 6L, 8.14701)
mode(var2)

## [1] "complex"

var3 <- c(FALSE, 4L, 3.67012, -4+9i, "model")
mode(var3)

## [1] "character"
```

Posiblemente al lector le llamó la atención el primer ejemplo `var1`, debido que el mismo contiene elementos `logical` & `integer` y al momento de conocer el tipo de vector obtenemos `numeric` en lugar de `logical`. Esto se debe básicamente a la manera como R almacena y reconoce los diferentes elementos, la siguiente tabla resume lo expuesto.

Tipo	Modo	Almacenamiento
logical	logical	logical
integer	numeric	integer
double	numeric	double
complex	complex	complex
character	character	character
raw	raw	raw

Cuadro 4.3: Tipos de vectores

Dentro de la coerción un evento importante se produce cuando un vector logical coerciona a un vector integer o double, pues en estos casos el TRUE se convierte en 1 y el FALSE en 0.

```
# creamos un vector logico
vec <- c(TRUE, FALSE, TRUE, TRUE, FALSE)
# coercionamos a numerico
as.numeric(vec)

## [1] 1 0 1 1 0
```

Para la coerción contamos con varias funciones que nos facilitan el trabajo tales como: `as.character()`, `as.double()`, `as.integer()`, `as.logical()`.

Almacenamiento de vectores

Una característica importante de los vectores es que se encuentran almacenados de forma plana incluso cuando se anidan vectores como se observa:

```
c(2L, c(4L, c(6L, 8L)))

## [1] 2 4 6 8

# obtenemos el mismo resultado mediante
c(2L, 4L, 6L, 8L)

## [1] 2 4 6 8
```

4.1.2. Valores perdidos

En ocasiones puede que no todos los elementos de un vector sean conocidos por diferentes motivos, en este caso nos hacen falta dichos elementos, lo cuales se denominan datos perdidos o faltantes. R permite que el usuario establezca dichos valores perdidos a través del término NA.

```
#Generamos un vector cuyo tercer elemento es un dato perdido
vec <- c(2, 3, NA, 8)
vec

## [1]  2  3 NA  8
```

Debido a los distintos tipos de elementos que existen en R, se han creado terminologías para los datos perdidos de acuerdo al tipo de elemento, es por ello que tenemos:

```
#NA para elementos numericos
NA_real_

## [1] NA

#NA para elementos enteros
NA_integer_

## [1] NA

#NA para elementos caracteres
NA_character_

## [1] NA

#NA para elementos complejos
NA_complex_

## [1] NA
```

Lo anterior evitará que coercionen los elementos de un vector por algún motivo. Para conocer si un vector contiene valores perdidas usaremos la función `is.na( )`.

```
# creamos dos vectores
x <- c(2, 4, 7, 9)
y <- c(3, NA, 8, 13)
# evaluamos si existen valores perdidos
is.na(x)

## [1] FALSE FALSE FALSE FALSE

is.na(y)

## [1] FALSE  TRUE FALSE FALSE
```

4.1.3. Eliminación

En el caso que se desee eliminar ciertas variables u objetos innecesarios del área de trabajo usamos los comandos `rm()` o `remove()`.

```
# Mostramos los objetos actuales
ls()

##  [1] "chr_var"  "dbl_var"  "int_var"  "logi_var" "var"      "var1"
##  [7] "var2"     "var3"     "vec"      "x"        "x_mul"    "x_new"
## [13] "x_vec"    "y"        "y_new"    "y_vec"

# Iniciamos eliminando los vectores: chr_var, dbl_var, int_var, logi_var
remove(chr_var, dbl_var, int_var, logi_var)
# Mostramos los objetos restantes
ls()

##  [1] "var"   "var1"  "var2"  "var3"  "vec"   "x"     "x_mul" "x_new"
##  [9] "x_vec" "y"     "y_new" "y_vec"
```

Cuando el usuario desea eliminar los objetos cuyos nombres cumple con algún patrón en común, usamos lo siguiente:

```
# Eliminamos las variables: var, var1, var2, var3
rm(list = ls(pattern = 'var'))
# Revisamos los objetos restantes
ls()

## [1] "vec"   "x"     "x_mul" "x_new" "x_vec" "y"     "y_new" "y_vec"
```

Finalmente, para eliminar todos los objetos del área de trabajo usaremos:

```
# Elimina todos los objetos existentes
rm(list=ls())
# Verificamos el area de trabajo
ls()

## character(0)
```

4.1.4. Modificación

Los vectores en R son almacenados como arreglos lineales, la ventaja de aquello es que sus elementos se encuentran indexados.

```
# creamos un vector
vec <- c(2, 4, 6, 8, 10)
vec

## [1]  2  4  6  8 10

# accedemos al elemento de la posicion 3
vec[3]

## [1] 6

# visualizamos los elementos de la posicion 2 y 5
vec[c(2, 5)]

## [1]  4 10
```

Añadiendo elementos

Sabemos que el tamaño de un vector es determinado en su creación, por lo que si se desea añadir o eliminar elementos es necesario reasignar el vector.

```
# agregamos los elementos 15, 18 al final
vec <- c(vec, c(15, 18))
vec

## [1]  2  4  6  8 10 15 18
```

```
# agregamos el valor de -1 al inicio
vec <- c(-1, vec)
vec

## [1] -1  2  4  6  8 10 15 18
```

En el caso que se desee añadir elementos en posiciones intermedias del vector es necesario conocer las posiciones de sus elementos.

```
# agregamos el elemento -4 en la posicion 5
vec <- c(vec[1:4], -4, vec[5:8])
vec

## [1] -1  2  4  6 -4  8 10 15 18
```

Eliminando elementos

Para eliminar ciertos los elementos de un vector usaremos subíndices negativos, lo anterior nos permite excluir los elementos deseados como se muestra a continuación:

```
# eliminamos el valor de la posicion 2
vec <- vec[-2]
vec

## [1] -1  4  6 -4  8 10 15 18

# eliminamos los elementos de las posiciones 4, 7
vec <- vec[-c(4, 7)]
vec

## [1] -1  4  6  8 10 18
```

Modificando elementos

Para modificar los elementos de un vector es suficiente conocer la posición de los mismos como se muestra en el siguiente ejemplo:

```
# modificamos el primer elemento por 0
vec[1] <- 0
# visualizamos el vector modificado
vec

## [1]  0  4  6  8 10 18

# modificamos las posiciones 2 y 5
vec[c(2, 5)] <- c(5, 9)
vec

## [1]  0  5  6  8  9 18
```

Generación de secuencias

Ahora mostramos algunos operadores que son útiles para crear vectores. Iniciamos con el operador ':', el cual genera un vector que abarca un rango de números con salto uniforme de amplitud 1.

```
1:12

##  [1]  1  2  3  4  5  6  7  8  9 10 11 12

7:-5

##  [1]  7  6  5  4  3  2  1  0 -1 -2 -3 -4 -5
```

El segundo operador que revisaremos es `seq( )`, mismo que tiene tres parámetros que controlan el inicio, final y salto de la secuencia.

```
seq(from = 1, to = 12, by = 1)

##  [1]  1  2  3  4  5  6  7  8  9 10 11 12

seq(from = -2, to = 3, by = 0.7)

## [1] -2.0 -1.3 -0.6  0.1  0.8  1.5  2.2  2.9

seq(from = -4, to = 4, length = 12)

##  [1] -4.0000000000 -3.2727272727 -2.5454545455 -1.8181818182
##  [5] -1.0909090909 -0.3636363636  0.3636363636  1.0909090909
##  [9]  1.8181818182  2.5454545455  3.2727272727  4.0000000000
```

Por último, la function `rep( )` nos permite repetir un objeto un número de veces especificado, a continuación mostramos algunos ejemplos:

```
rep(3, 8)

## [1] 3 3 3 3 3 3 3 3

rep(c(3, 6, 9), 2)

## [1] 3 6 9 3 6 9

rep(1:2, 4)

## [1] 1 2 1 2 1 2 1 2

rep(c(2, 4, 6), each = 3)

## [1] 2 2 2 4 4 4 6 6 6

rep(c("Source", "Stat", "Ecuador"), 2)

## [1] "Source"  "Stat"    "Ecuador" "Source"  "Stat"    "Ecuador"
```

4.1.5. Operaciones

Ahora revisaremos algunas operaciones comúnes relacionadas con los vectores. Cubriremos las operaciones aritméticas y lógicas más utilizadas.

Operación	Descripción
a + b	Suma
a - b	Resta
a * b	Multiplicación
x / y	División
x ∧ y	Potencia
sqrt(x)	Raíz cuadrada
abs(x)	Valor absoluto
exp(x)	Exponencial
log(x, base=n)	Logaritmo en base n
factorial(x)	Factorial
x %% y	Módulo
x %/% y	División entera
x == y	Test de igualdad
x != y	Test de desigualdad
x <= y	Test menor o igual que
x >= y	Test mayor o igual que
x && y	Conjunción para escalares
x || y	Disyunción para escalares
x & y	Conjunción para vectores
x | y	Disyunción para vectores
!x	Negación

Cuadro 4.4: Operaciones básicas

Operaciones aritméticas

Recordemos que R es un lenguaje funcional, por lo que cualquier operador es una función. Veamos un ejemplo:

```
# suma de 2 elementos
3 + 5

## [1] 8

# Division entera
8 %/% 3

## [1] 2

# Logaritmo en base 3
log(24, base=3)

## [1] 2.892789261

# otra alternativa
"+" (3, 5)
```

```
## [1] 8

"%/%" (8, 3)

## [1] 2

"log" (24, 3)

## [1] 2.892789261

# creamos dos vectores
x <- c(1, 3, 5, 7, 9)
y <- c(2, 4, 6, 8, 10)
# sumamos los vectores
x + y

## [1]  3  7 11 15 19

# segunda alternativa para la suma
"+" (x, y)

## [1]  3  7 11 15 19
```

Cuando aplicamos una operación a dos vectores se requiere que tengan la misma longitud, caso contrario, R *recicla* o repite los elementos con el fin que los vectores tengan la misma longitud.

```
# suma de vectores longitud 2 y 3
c(2, 4) + c(3, 6, 9)

## Warning in c(2, 4) + c(3, 6, 9): longer object length is not a multiple of shorter
object length

## [1]  5 10 11

# suma de vectores longitud 3 y 6
c(1, 2, 3) + c(1, 2, 3, 4, 5, 6)

## [1] 2 4 6 5 7 9
```

El primer ejemplo muestra una advertencia debido que el vector más pequeño no es multiplo del vector más grande, lo cual no ocurre en el segundo ejemplo. Observamos que en ambos casos el vector más pequeño fue reciclado para que se realice la operación.

Si el usuario se encuentra familiarizado con el álgebra lineal se verá sorprendido al ver lo que sucede cuando se multiplica dos vectores.

```
# calculamos el producto de los vectores x, y
x*y

## [1]  2 12 30 56 90

# otra alternativa para la multiplicacion
"*" (x, y)

## [1]  2 12 30 56 90
```

El ejemplo anterior muestra que el producto entre vectores de igual longitud se da entre elementos de la misma posición. En el caso que se multiplique un vector por un escalar el resultado es el siguiente:

```
# Producto
2 * c(1, 3, 5, 7)

## [1]  2  6 10 14

c(3, 4) * c(1, 3, 5, 7)

## [1]  3 12 15 28

# Division
c(2, 4, 6, 8) / 2

## [1] 1 2 3 4

c(24, 16, 15, 8) / c(4, 8, 3, 2)

## [1] 6 2 5 4
```

Operaciones lógicas

Ahora revisaremos las expresiones lógicas o booleanas[1], las cuales son utilizadas con frecuencia en distintos lenguajes de programación para verificar si un objeto cumplen ciertas condiciones deseadas.

```
# verificamos la siguientes expresiones
2 == 6

## [1] FALSE

7 <= 10

## [1] TRUE

(1 < 3) * (4 >= 2)

## [1] 1

(8 >= 5) == 1

## [1] TRUE

# generamos un vector entre 1 y 9
x <- seq(1,9)
# verificamos los elementos mayores a 5
x > 5

## [1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
```

[1] George Boole (1815–1864)

Las funciones `any( )` y `all( )` nos reportan cuando al menos uno o todos sus argumentos son verdaderos.

```
# verificamos si algun elemento es mayor a 5
any(x>5)

## [1] TRUE

# verificamos si todos los elementos son mayores a 5
all(x>5)

## [1] FALSE
```

4.1.6. Atributos

Todos los objetos en R tienen una lista de atributos, misma que es utilizada para almacenar la metadata[2] de los objetos

4.2. Factores

Son variables en R que tienen un número limitado de valores diferentes, dichas variables se conocen como variables categóricas a menudo; por ejemplo: el conjunto de observaciones acerca del color de ojos de un grupo de personas.

```
ojos <- c("negro", "azul", "negro", "verde", "verde", "cafe", "negro")
table(ojos)

## ojos
##  azul  cafe negro verde
##     1     1     3     2
```

4.2.1. Uso de factores

El uso más importante de los factores se encuentra en el modelamiento estadístico; dado que ciertas variables categóricas ingresan a los modelos de manera diferente a las variables continuas. Los factores en R se almacenan como un vector de valores enteros con un conjunto correspondiente de valores de caracteres para usar cuando aparezca el factor.

4.3. Matrices

Colección de datos a los que se accede por varios índices enteros (dimensiones).

4.4. Listas

Colección ordenada de objetos, en la que los elementos pueden ser de distinto tipo.

[2]La Metadata es data que describe otra data. Es información que describe el contenido un archivo u objeto.

4.5. Arrays

Un arreglo (array) de datos es un objeto que puede ser concebido como una matriz multidimensional (hasta 8 dimensiones). Una ventaja de este tipo de objeto es que sigue las reglas que hemos descrito para las matrices. La sintaxis para definir un arreglo es

```
array(data, dim)
```

Las componentes data y dim deben presentarse como una sola expresión, por ejemplo

```
c(2,4,6,8,10)
c(2,3,2)
x <- array (1:24, c(3,4,2))
```

produce un arreglo tridimensional: la primera dimensión tiene tres niveles, la segunda tiene cuatro y la tercera tiene dos. Al imprimir el arreglo R comienza con la dimensión mayor y va bajando hacia la dimensión menor, imprimiendo matrices bidimensionales en cada etapa.

4.6. Data Frames

Tipo particular de listas de gran utilidad para el trabajo estadístico.

4.7. Data Table

El paquete `data.table` fue creado por Matt Dowle conjuntamente con grupo de contribuidores y fue publicado en el año 2015. El paquete ofrece una versión mejorada de los `data.frame`, a continuación enumeramos las mejoras:

1. Rápida agregación para datos de gran tamaño (por ejemplo: 100 GB en RAM).
2. Lectura rápida y amigable para archivos a través de la función `fread`.
3. Añade, modifica y elimina variables sin utilizar copias en absoluto.
4. Rápido ordenamiento de variables: hacia adelante, hacia atrás.

La syntaxis básica de data.table no es difícil de dominar debido que los autores se preocuparon en reducir el tiempo que le toma al usuario programar:

$$\text{DT[where, select | group by]}$$

Existe una similitud entre la syntaxis de SQL y R, misma que se resume a continuación:

SQL:	where	select	group by
R:	i	j	by

4.7.1. Creación

En las siguientes líneas revisamos como crear un objeto data.table:

```
library(data.table)
set.seed(12345)
base <- data.table(B1=1:12, B2=LETTERS[1:4], B3=round(rnorm(3), 4),
                   B4=c(2L, 5L, 8L))
dim(base)

## [1] 12  4
```

4.7.2. Filtrado

Filtrado de filas

Para seleccionar las filas de un data.table se debe especificar la posición de la fila:

```
base[3, ]

##    B1 B2      B3 B4
## 1:  3  C -0.1093  8

base[9:12, ]

##    B1 B2      B3 B4
## 1:  9  A -0.1093  8
## 2: 10  B  0.5855  2
## 3: 11  C  0.7095  5
## 4: 12  D -0.1093  8

base[c(3, 9, 12), ]

##    B1 B2      B3 B4
## 1:  3  C -0.1093  8
## 2:  9  A -0.1093  8
## 3: 12  D -0.1093  8
```

Para filtrar variables que cumplen una cierta condición realizamos lo siguiente:

```
base[B4==5L]

##    B1 B2     B3 B4
## 1:  2  B 0.7095  5
## 2:  5  A 0.7095  5
## 3:  8  D 0.7095  5
## 4: 11  C 0.7095  5

base[B2=="D"]

##    B1 B2      B3 B4
## 1:  4  D  0.5855  2
## 2:  8  D  0.7095  5
## 3: 12  D -0.1093  8

base[B2 %in% c("A","B")]

##    B1 B2      B3 B4
## 1:  1  A  0.5855  2
## 2:  2  B  0.7095  5
## 3:  5  A  0.7095  5
## 4:  6  B -0.1093  8
## 5:  9  A -0.1093  8
## 6: 10  B  0.5855  2
```

Filtrado de columnas

Ahora revisaremos como seleccionar ciertas variables de la base inicial

```
base[,B2]

##  [1] "A" "B" "C" "D" "A" "B" "C" "D" "A" "B" "C" "D"
```

La selección de dos o más variables se realizada anteponiendo `.()` o su equivalente `list()`

```
base[,.(B2,B3)]

##     B2      B3
##  1:  A  0.5855
##  2:  B  0.7095
##  3:  C -0.1093
##  4:  D  0.5855
##  5:  A  0.7095
##  6:  B -0.1093
##  7:  C  0.5855
##  8:  D  0.7095
##  9:  A -0.1093
## 10:  B  0.5855
## 11:  C  0.7095
## 12:  D -0.1093
```

Calculamos el promedio de la variable B3.

```
base[,mean(B3)]

## [1] 0.3952333333
```

```
base[,list(SUMA=sum(B4), DESVIACION=sd(B3))]

##    SUMA   DESVIACION
## 1:   60 0.3763551643
```

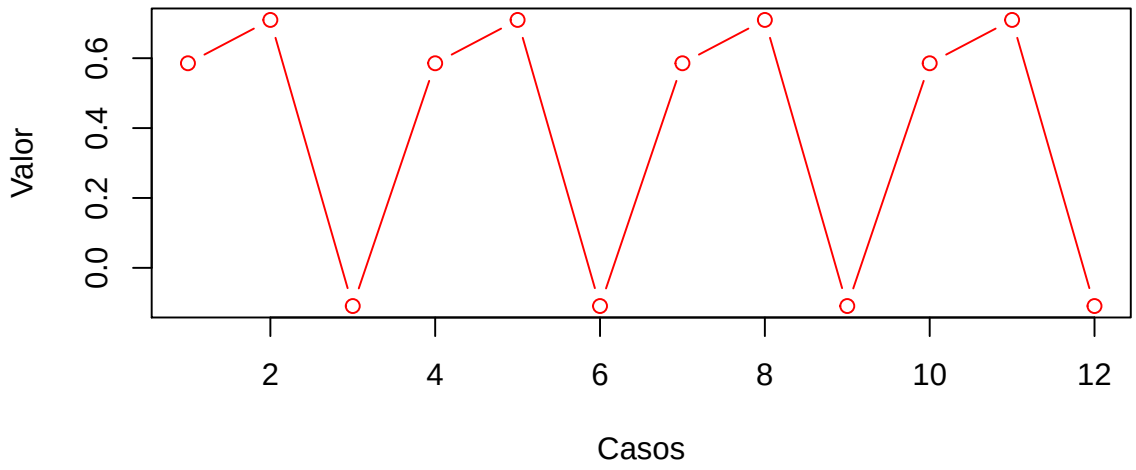
Reciclado de resultados

```
base[,.(B2, Sd.B3 = sd(B3))]

##     B2        Sd.B3
##  1:  A 0.3763551643
##  2:  B 0.3763551643
##  3:  C 0.3763551643
##  4:  D 0.3763551643
##  5:  A 0.3763551643
##  6:  B 0.3763551643
##  7:  C 0.3763551643
##  8:  D 0.3763551643
##  9:  A 0.3763551643
## 10:  B 0.3763551643
## 11:  C 0.3763551643
## 12:  D 0.3763551643
```

```
base[,{print(B1)
     plot(B3, xlab='Casos', ylab='Valor', col='red', type='b')
     NULL}]

##  [1]  1  2  3  4  5  6  7  8  9 10 11 12
```




```
## NULL
```

5

Funciones

En los capítulos anteriores ya hemos hecho uso de una variedad de funciones que vienen con R. En este capítulo abordaremos el diseño y construcción de funciones propias para el usuario, la posibilidad de programar de una manera sencilla una serie de análisis que puedan ser ejecutados de manera sucesiva da una gran ventaja a R sobre el resto de programas estadísticos.

5.1. Creación

El comando `function()` tiene como tarea crear funciones a partir de ciertos argumentos[1].

```
NombreFuncion <- function (Argumento1,..., ArgumentoN)
{
 expresión 1
 ...
 expresión N
 }
```

Procedemos a crear nuestra primera función:

```
f <- function(x){
   return(2*x)
}
```

La función f recibe un objeto x y arroja como resultado el objeto x multiplicado por 2.

```
f(2)

## [1] 4

f(c(3,5))

## [1]  6 10
```

Las funciones en R también son objetos, de esta manera podemos trabajar al igual que lo hacíamos con otros tipos de objetos.

[1] Valores iniciales necesarios para la ejecución de las funciones.

5.1.1. Componentes

Todas las funciones en R constan de tres partes fundamentales:

- body(): Corresponde al código que se encuentra dentro de la función.

```
body(f)

## {
##     return(2 * x)
## }
```

- formals(): Corresponde a la lista de argumentos que controlan la llamada a la función.

```
formals(f)

## $x
```

- environment(): Localización de las variables de la función.

```
environment(f)

## <environment: R_GlobalEnv>
```

Si el ambiente no se visualiza significa que la función fue creada en el ambiente global.

Para evidencia la inclusión de dos o más argumentos dentro de una función, crearemos una función que nos calcule la hipotenusa de un triángulo rectánculo, dicha función consta de dos argumentos (catetos):

```
hipotenusa <- function(x,y){
   h <- sqrt(x^2 + y^2)
   return(h)
}
```

La salida de la función es la hipotenusa del triángulo rectangulo, contrastamos los resultados:

```
hipotenusa(4,3)

## [1] 5

hipotenusa(12, 16)

## [1] 20
```

5.1.2. Estructuras de control

Las estructuras de control

5.2. Funciones primitivas

Existe un excepción para la regla que todas las funciones tienen 3 componentes; las funciones primitivas tales como: `any`, `sum`, `for`, etc. se encuentran construidas en `C`, `C++` o `Phyton`, por lo cual no cumple con lo antes mencionado.

```
# Funcion any
any

## function (..., na.rm = FALSE)  .Primitive("any")

body(any)

## NULL

formals(any)

## NULL

environment(any)

## NULL
```

En el caso que el usuario desee conocer todas las funciones primitivas del paquete `base`, lo puede hacer por medio de la siguiente línea de comando:

```
ls("package:base")
```

Las funciones primitivas son creadas únicamente con el R Core Team, y en la actualidad se hace lo posible para no crear más funciones primitivas a excepción que no exista otra opción.

5.3. Funciones genéricas

Hemos observado que una gran mayoría de las funciones usadas hasta el momento se encuentran albergadas dentro de los paquetes base.

5.4. Lexical Scoping

5.4.1. Name Masking

5.4.2. Function vs Variables

5.4.3. Fresh Start

5.4.4. Dynamic Lookup

6

Manipulación de datos

6.1. Operadores de encadenamiento

El desarrollo de los operadores de encadenamiento inicia el 17 de Enero de 2012 a partir de la inquietud colocada por el usuario anónimo *user4* en el sitio web `Stack Overflow`[1], la misma trataba de averiguar la posible implementación de los operadores del lenguaje $F\#$[2] (F Sharp) en R.

Ben Bolker respondío el mismo día dando lo que podríamos considerar el primer operador en R:

```
"%>%" <- function(x,f) do.call(f,list(x))
16 %>% sqrt

## [1] 4
```

Para Octubre de 2013 aparece el primer operador de encadenamiento como parte del paquete `dplyr` desarrollado por Hadley Wickham. Este operador fue denominado `chain` (%.%), la idea detrás de la introducción del operador fue simplificar la notación con el fin de aplicar varias funciones al mismo tiempo a un `data.frame`.

Rutina con encadenamiento

```
mtcars %.% group_by(carb, cyl) %.%
   select(mpg, disp, hp) %.%
   summarise(
      mean_mpg = mean(mpg, na.rm = TRUE),
      mean_disp = mean(disp, na.rm = TRUE),
      mean_hp = mean(hp, na.rm = TRUE)
   )

## Source: local data frame [9 x 5]
## Groups: carb
##
##   carb cyl mean_mpg mean_disp mean_hp
```

[1] Sitio web desarrollado por Jeff Attwood muy utilizado por una comunidad de desarrolladores informáticos, en la cual se pueden encontrar soluciones a problemas de programación en diferentes lenguajes.

[2] Lenguaje de programación multiparadigma de código abierto para la plataforma .NET

```
## 1    1   4    27.58     91.38    77.4
## 2    1   6    19.75    241.50   107.5
## 3    2   4    25.90    116.60    87.0
## 4    2   8    17.15    345.50   162.5
## 5    3   8    16.30    275.80   180.0
## 6    4   6    19.75    163.80   116.5
## 7    4   8    13.15    405.50   234.0
## 8    6   6    19.70    145.00   175.0
## 9    8   8    15.00    301.00   335.0
```

Rutina sin encadenamiento

```
summarise(select(group_by(mtcars, carb, cyl), mpg, disp, hp),
          mean_mpg = mean(mpg, na.rm = TRUE),
          mean_disp = mean(disp, na.rm = TRUE),
          mean_hp = mean(hp, na.rm = TRUE))

## Source: local data frame [9 x 5]
## Groups: carb
##
##   carb cyl mean_mpg mean_disp mean_hp
## 1    1   4    27.58     91.38    77.4
## 2    1   6    19.75    241.50   107.5
## 3    2   4    25.90    116.60    87.0
## 4    2   8    17.15    345.50   162.5
## 5    3   8    16.30    275.80   180.0
## 6    4   6    19.75    163.80   116.5
## 7    4   8    13.15    405.50   234.0
## 8    6   6    19.70    145.00   175.0
## 9    8   8    15.00    301.00   335.0
```

En Diciembre de 2013, Stefan Bache propone una respuesta alternativa original para la inquietud colocada en `Stack Overflow`.

```
`%>%` <- function(e1, e2){
   cl <- match.call()
   e  <- do.call(substitute, list(cl[[3]], list(. = cl[[2]])))
   eval(e)
}
```

Un ejemplo rápido del uso para el operador propuesto por Bache es el siguiente:

```
mtcars %>%
  subset(., cyl == 8, select = -vs) %>%
  colMeans(.)

##            mpg            cyl           disp             hp
##  15.1000000000   8.0000000000 353.1000000000 209.2142857143
##           drat             wt           qsec             am
##   3.2292857143   3.9992142857  16.7721428571   0.1428571429
##           gear           carb
##   3.2857142857   3.5000000000
```

Stefan continuó trabajando con la finalidad de mejorar el funcionamiento de su operador de encadenamiento al punto que implementó el paquete `magrittr` que incluída al operador $\%>\%$.

El paquete `dplyr` estuvo siendo desarrollado en paralelo al trabajo de Bache. Finalmente, en Abril de 2014 el paquete `dplyr` incorporó el operador de `magrittr` sustituyendo al operador original de Hadley $\%.\%$ debido que el primero es más fácil de escribir y posee desarrollo más minucioso.

Desde Agosto de 2014, RStudio incorporó un acceso directo para el operador $\%>\%$ por medio de la combinación de teclas: Ctrl+Shift+M.