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

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. En este capítulo analizaremos dos IDE's de gran importancia y utilidad en el mundo de los usuarios de R:

- RStudio
- R Analytic Flow

1.1. 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].

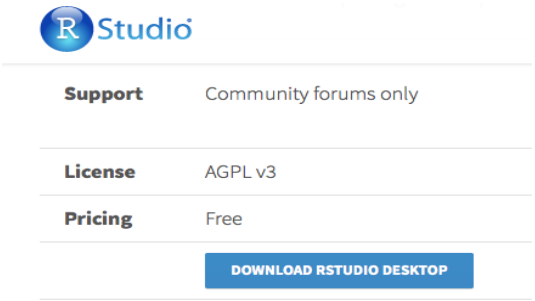

Figura 1.1: Descarga RStudio

Con el propósito que el usuario realice un trabajo rápido y eficiente, RStudio incorpora atajos de teclado que reducen la dependencia del ratón, se recomienda a los nuevos usuarios hacer uso de los mismos y adquirir este buen hábito.

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

Para acceder al listado de atajos desde RStudio se debe presionar la combinación de teclas[2]:

Sistema	Combinación
Windows / Linux	`Alt + Shift + K`
Mac OS	`Option + Shift + K`

1.1.1. Instalación y actualización

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 se realiza a partir del siguiente script:

```
# descargar e instalar el paquete installr
install.packages("installr")
# cargar el paquete installr
library(installr)
# instalamos RStudio IDE
install.RStudio()
```

La actualización del programa se realiza desde el menú: `Help` > `Check for Updates`.

1.1.2. Funcionamiento

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.

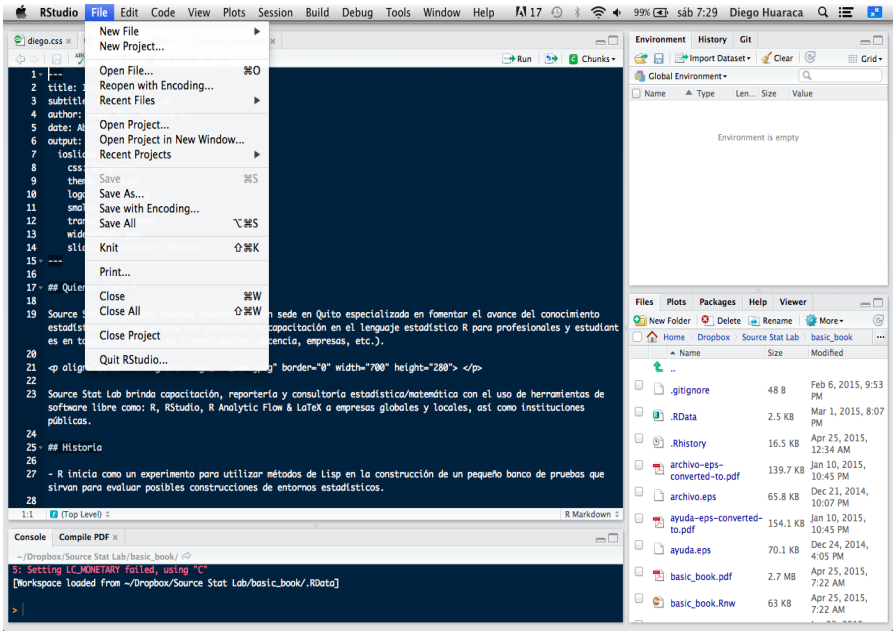

Figura 1.2: Interfaz de RStudio

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

[2]En caso de requerir el listado completo de atajos recomendamos visitar la página: `https://support.rstudio.com/hc/en-us/articles/200711853-Keyboard-Shortcuts`.

- **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.

1.1.3. 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]  0.9436435 -0.2694918 -0.8745993
##  [4] -1.0675457  1.1465734  0.2706766
##  [7] -1.3196751 -0.2825003 -0.1271204
## [10] -0.9769779
```

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

##  [1] -0.7653821  1.5872601  0.4362222 -0.2266527
##  [5]  2.2721490  0.6638876 -0.5819160  1.5610229
##  [9]  0.5421298  2.1252262
```

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

##  [1] -0.7032691  0.5356480 -1.6006910  0.2348602  0.9416207  2.4057097
##  [7]  1.0292868  1.5119428 -1.0026021  1.0182127
```

1.1.4. 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 >")
```

1.1.5. 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] -1.339  0.027 -0.946
```

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

## [1]  0.28953 -0.24557  0.44907
```

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

## [1]  0.2126794562  0.9549453993 -0.9188249907
```

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)
```

1.1.6. 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)
```

1.2. R Analytic Flow

R Analytic Flow (RAF) es una interfaz gráfica de usuario desarrollado por Ryota Suzuki[3], 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.

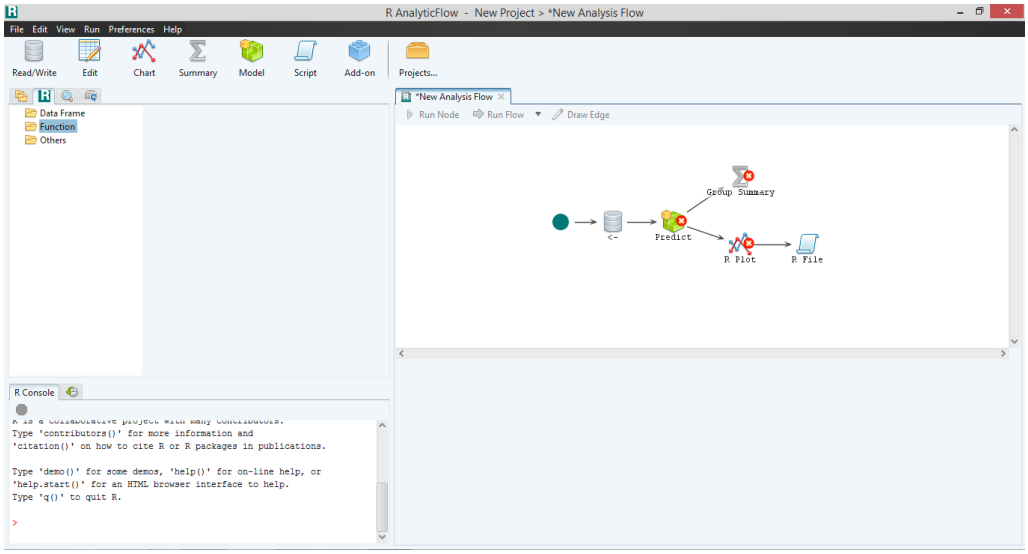

Figura 1.3: R Analytic Flow

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]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.

1.2.1. 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.

1.2.2. 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.

2

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 realizadas por el usuario, la posibilidad de encapsular fragmentos de código que puedan ser ejecutados en múltiples ocasiones y con diferentes parámetros da una gran ventaja a R sobre el resto de programas estadísticos.

2.1. Creación

Las funciones dentro de R, son tratadas como cualquier otro objeto. El comando `function()` tiene como tarea crear funciones a partir de un número establecido de argumentos o valores iniciales necesarios para la ejecución. La declaración de una función consta de cuatro partes:

1. **Nombre**: Carácter asignado para el llamado a la función, mismo que debe cumplir con ciertas condiciones:

   - Iniciar con una letra,
   - No tener espacios en blanco, ni símbolos reservados.

Recomendados	No permitidos
MiFuncion	Mi-Funcion, Mi+Funcion, MiFunción, mi:funcion
Codigo_01	Código #1, 1 Código, Codigo.0.1, codigo$

   En el nombre de una función si se puede emplear el signo de punto (.), sin embargo no es muy aconsejable su uso debido que el mismo está reservado para la definición de métodos.

2. **Argumentos**: Son un serie de valores que recibe la función para operar y lograr obtener un resultado. Pueden existir funciones que carezcan de argumentos, siempre y cuando las acciones a realizarse no dependan de ningún valor enviado por el usuario.

   Vale la pena aclarar que el orden en que se pasen los valores de los argumentos corresponde con la ubicación de estos en la declaración de la función.

3. **Código**: Conjunto de instrucciones necesarias para alcanzar un resultado.

4. **Resultado**: Objeto devuelto tras la ejecución de la función.

```
NombreFuncion <- function (Argumento1,..., ArgumentoN)
{
 expresión 1
 .......     # Código
 expresión N
 return(val) # Resultado
 }
```

Procedemos a crear nuestra primera función:

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

La función `fr` recibe como argumento un objeto x (vector) y arroja como resultado el objeto x multiplicado por 2.

```
fr(2)

## [1] 4

fr(c(3,5))

## [1]  6 10
```

En las líneas anteriores observamos que las palabras `function` y `return` son *reservadas* (palabras propias del lenguaje R). Vale hacer énfasis en que toda función inicia con la especificación de su nombre y termina con el comando `return` que especifica el resultado que debe mostrarse como salida.

2.1.1. Componentes

Todas las funciones en R constan de tres partes fundamentales en la etapa de ejecución:

- **body**: Corresponde al código que se encuentra dentro de la función, es la encargada de realizar las operaciones para alcanzar el resultado deseado.

  ```
  body(fr)

  ## {
  ##     res <- 2 * x
  ##     return(res)
  ## }
  ```

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

  ```
  formals(fr)

  ## $x
  ```

- **environment**: Localización de las variables de la función dentro de la jerarquía de ambientes.

```
environment(fr)

## <environment: R_GlobalEnv>
```

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

Para evidenciar la inclusión de dos o más argumentos, crearemos una función que nos calcule la hipotenusa de un triángulo rectángulo, la cual constará de dos argumentos (catetos):

```
hipotenusa <- function(x,y){ # x, y son los catetos
   h <- sqrt(x^2 + y^2) # Se calcula la hipotenusa
   return(h) # Devuelve el resultado
}
```

La salida de la función corresponde a la hipotenusa del triángulo rectángulo, contrastamos los resultados:

```
hipotenusa(4,3)

## [1] 5

hipotenusa(12, 16)

## [1] 20
```

2.1.2. Estructuras de control

Estructura IF

La estructura de control condicional o selectiva `if` nos permite ejecutar un bloque de código siempre y cuando se cumpla con la condición impuesta por el usuario. Su sintaxis es la siguiente:

```
if(condición){
   ....
   código
   ....
}
```

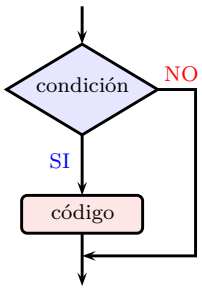

Figura 2.1: Estructura IF

2.2. 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.

2.3. Funciones genéricas

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

2.4. Lexical Scoping

2.4.1. Name Masking

2.4.2. Function vs Variables

2.4.3. Fresh Start

2.4.4. Dynamic Lookup