El arte de programar en R

Un lenguaje para la estadística

Julio Sergio Santana • Efraín Mateos Farfán

El arte de programar en R: un lenguaje para la estadística

Julio Sergio Santana
Efraín Mateos Farfán

27 de noviembre de 2014

005.1330727 Santana Sepúlveda, Julio Sergio.
S72 *El arte de programa en R: un lenguaje para la estadística* / Julio Sergio Santana Sepúlveda y Efraín Mateos Farfán.. -- México : Instituto Mexicano de Tecnología del Agua. UNESCO. Comité Nacional Mexicano del Programa Hidrológico Internacional, ©2014.
182 p. : il.

ISBN 978- 607-9368-15-9
1. R [Lenguajes de programación] 2. Estadística matemática I. Santana Sepúlveda, Julio Sergio II. Mateos Farfán, Efraín.

Coordinación editorial:
Instituto Mexicano de Tecnología del Agua.

Coordinación de Comunicación,
Participación e Información.

Subcoordinación de Vinculación, Comercialización
y Servicios Editoriales.

Primera edición: 2014.

Ilustración de portada:
© Óscar Alonso Barrón

Paseo Cuauhnáhuac 8532
62550 Progreso, Jiutepec, Morelos
MÉXICO
www.imta.gob.mx

ISBN: 978- 607-9368-15-9

Índice general

Índice de figuras

Prólogo

El arte de programar en R

Un artista, típicamente un pintor, tiene en sus manos un conjunto de recursos artísticos: materiales y herramentales, que al combinarlos de acuerdo con su sensibilidad y habilidades, transforma en una obra de arte, estéticamente atractiva o repulsiva a sus destinatarios. Aunque esencialmente tecnológica, en el caso de la programación, el programador o programadora juega ese mismo papel: los distintos ambientes de programación ponen delante de él/ella un conjunto de recursos que de acuerdo con sus conocimientos, habilidades y sensibilidades, combinará para producir obras: programas y sistemas, que, en la superficie, serán funcionales o no funcionales; pero que, a un nivel más profundo, podrían también ser juzgadas como estéticamente atractivas o repulsivas. Uno de los factores que más decisivamente influyen en el *cómo* un creador combina los elementos a su mano, es el gusto y la pasión que imprime en su tarea. Por consiguiente, si de la lectura de este texto se logra encender en ti, querida lectora o lector, una pasión que te lleve a producir verdaderas obras de arte, los autores estaremos más que satisfechos por haber cumplido el propósito de esta *pequeña* obra.

.

Capítulo 1

Introducción

1.1. ¿Qué es R?

Si este es tu primer acercamiento a R, es muy probable que te cuestiones sobre la ventaja y la utilidad de R sobre otras paqueterías de estadística; como veremos adelante R es más que eso. La intención de este primer capítulo, es responder algunas dudas y animarte a que explores este *software* poderoso, que puede ser aplicado ampliamente en el procesamiento de datos en ciencias.

Empezaremos diciendo que R es un lenguaje de programación interpretado, de distribución libre, bajo Licencia GNU, y se mantiene en un ambiente para el cómputo estadístico y gráfico. Este *software* corre en distintas plataformas Linux, Windows, MacOS, e incluso en PlayStation 3. El término ambiente pretende caracterizarlo como un sistema totalmente planificado y coherente, en lugar de una acumulación gradual de herramientas muy específicas y poco flexibles, como suele ser con otro *software* de análisis de datos. El hecho que R sea un lenguaje y un sistema, es porque forma parte de la filosofía de creación[1], como lo explica John Chambers (Chambers and Hastie [1991]), cito:

> "*Buscamos que los usuarios puedan iniciar en un entorno interactivo, en el que no se vean, conscientemente, a ellos mismos como programadores. Conforme sus necesidades sean más claras y su complejidad se incremente, deberían gradualmente poder profundizar en la programación, es cuando los aspectos del lenguaje y el sistema se vuelven más importantes*."

Por esta razón, en lugar de pensar de R como un sistema estadístico, es preferible verlo como un ambiente en el que se aplican técnicas estadísticas. Por ejemplo, en este libro nos inclinaremos hacia el lado de la programación (lenguaje) más que tocar los aspectos estadísticos. Esto con la finalidad de ampliar la gamma de aplicaciones en el tratamiento de datos.

[1]Desde la codificación del lenguaje S, lenguaje progenitor de R, como se verá en la sección siguiente.

1.2. Historia de R y S

R fue creado en 1992 en Nueva Zelanda por Ross Ihaka y Robert Gentleman (Ihaka [1998]). La intención inicial con R, era hacer un lenguaje didáctico, para ser utilizado en el curso de Introducción a la Estadística de la Universidad de Nueva Zelanda. Para ello decidieron adoptar la sintaxis del lenguaje S desarrollado por Bell Laboratories. Como consecuencia, la sintaxis 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.

A modo de broma Ross y Robert, comienzan a llamar "R" al lenguaje que implementaron, por las iniciales de sus nombres, y desde entonces así se le conoce en la muy extendida comunidad amante de dicho lenguaje. Debido a que R es una evolución de S, a continuación daremos una breve reseña histórica de este lenguaje, para entender los fundamentos y alcances de R.

S es un lenguaje que fue desarrollado por John Chambers y colaboradores en Laboratorios Bell (AT&T), actualmente Lucent Tecnologies, en 1976. Este lenguaje, originalmente fue codificado e implementado como unas bibliotecas de FORTRAN. Por razones de eficiencia, en 1988 S fue reescrito en lenguaje C, dando origen al sistema estadístico S, Versión 3. Con la finalidad de impulsar comercialmente a S, Bell Laboratories dio a StatSci (ahora Insightful Corporation) en 1993, una licencia exclusiva para desarrollar y vender el lenguaje S. En 1998, S ganó el premio de la *Association for Computing Machinery* a los Sistemas de *Software*, y se liberó la versión 4, la cual es prácticamente la versión actual.

El éxito de S fue tal que, en 2004 *Insghtful* decide comprar el lenguaje a *Lucent* (*Bell Laboratories*) por la suma de 2 millones de dólares, convirtiéndose hasta la fecha en el dueño. Desde entonces, *Insightful* vende su implementación del lenguaje S bajo el nombre de S-PLUS, donde le añade un ambiente gráfico amigable. En el año 2008, TIBCO compra *Insightful* por 25 millones de dólares y se continúa vendiendo S-PLUS, sin modificaciones. R, que define su sintaxis a partir de esa versión de S, no ha sufrido en lo fundamental ningún cambio dramático desde 1998.

Regresemos ahora al lenguaje que nos ocupa: R. Luego de la creación de R (en 1992), se da un primer anuncio al público del software R en 1993. En el año de 1995 Martin Mächler, de la Escuela Politécnica Federal de Zúrich, convence a Ross y Robert a usar la Licencia GNU para hacer de R un software libre. Como consecuencia, a partir de 1997, R forma parte del proyecto GNU.

Con el propósito de crear algún tipo de soporte para el lenguaje, en 1996 se crea una lista pública de correos; sin embargo debido al gran éxito de R, los creadores fueron rebasados por la continua llegada de correos. Por esta razón, se vieron en la necesidad de crear, en 1997, dos listas de correos, a saber: R-help y R-devel, que son las que actualmente funcionan para responder las diversas dudas que los usuarios proponen en muy diversos asuntos relativos al lenguaje. Además se consolida el grupo núcleo de R,donde se involucran personas asociadas con S-PLUS, con la finalidad de administrar el código fuente de R.

Fue hasta febrero de 29 del 2000, que se considera al software completo y lo

suficientemente estable, para liberar la versión 1.0.

Más información acerca de la historia de este lenguaje se puede obtener en Ihaka [1998].

1.3. Formato del código en el texto

Con el propósito de facilitar la lectura del presente texto, el código del lenguaje se diferencía en párrafos especiales, que han sido construidos con la ayuda del *software* knitr, gentilmente creado por Xie [2013][2]. A continuación se muestra un fragmento de código con las explicaciones correspondientes.

```
# Este es un comentario; en R los comentarios empiezan
# a partir del caracter '#'.
# --------------------
# En seguida asignaremos mediante código de R el valor 2014 a una
# variable llamada 'x':
x <- 2014
# Ahora se imprimirá el valor de la variable dos veces, la primera
# vez se hará de manera explícita por medio de la función print(),
# como sigue:

print(x)

## [1] 2014

# ... en seguida se imprimirá de manera implícita, simplemente
# 'tecleándola' en la consola:

x

## [1] 2014

# Finalmente haremos una multiplicación de x por 2

2*x

## [1] 4028

# Notemos que las impresiones o resultados de estas operaciones
# aparecen como comentarios, pero iniciados con '##' y con
# una tipografía diferente que los comentarios usuales.
```

[2]La fuente de su trabajo se puede encontrar en http://yihui.name/knitr/ y en http://cran.r-project.org/web/packages/knitr/index.html.

1.4. Algunas características importantes de R

El sistema R esta dividido en dos partes conceptuales: 1) El sistema base de R, que es el que puedes bajar de CRAN[3]; y, 2) en todo lo demás. La funcionalidad de R consta de paquetes modulares. El sistema base de R contiene el paquete básico que se requiere para su ejecución y la mayoría de las funciones fundamentales. Los otros paquetes contenidos en la "base" del sistema incluye a ***utils***, ***stats, datasets, graphics, grDevices, grid, tools, parallel, compiler, splines, tcltk, stats4***.

La capacidad de gráficos de R es muy sofisticada y mejor que la de la mayoría de los paquetes estadísticos. R cuenta con varios paquetes gráficos especializados, por ejemplo, hay paquetería para graficar, crear y manejar los *shapefiles*[4], para hacer contornos sobre mapas en distintas proyecciones, graficado de vectores, contornos, etc. También existen paqueterías que permiten manipular y crear datos en distintos formatos como netCDF, Matlab, Excel entre otros. Cabe señalar que, además del paquete base de R, existen más de 4000 paquetes en CRAN (http://cran.r-project.org), que han sido desarrollados por usuarios y programadores alrededor del mundo, esto sin contar los paquetes disponibles en redes personales.

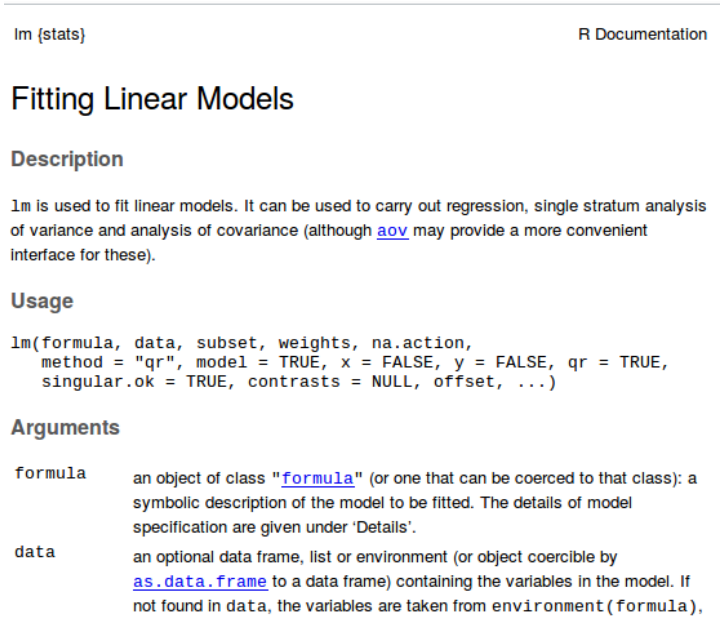
R es muy útil para el trabajo interactivo, pero también es un poderoso lenguaje de programación para el desarrollo de nuevas herramientas, por ejemplo rclimdex, cliMTA-R, etc. Otra ventaja muy importante es que tiene una comunidad muy activa, por lo que, haciendo las preguntas correctas rápidamente encontrarás la solución a los problemas que se te presenten en el ámbito de la programación con R. Estas características han promovido que el número de sus usuarios en el área de las ciencias se incremente enormemente.

Al ser *software* libre lo hace un lenguaje atractivo, debido a que no hay que preocuparse por licencias y cuenta con la libertad que garantiza GNU. Es decir con R se tiene la libertad de: 1) correrlo para cualquier propósito, 2) estudiar como trabaja el programa y adaptarlo a sus necesidades, pues se tiene acceso al código fuente, 3) redistribuir copias, y 4) mejorar el programa y liberar sus mejoras al público en general.

Es importante mencionar que, debido a su estructura, R consume mucho recurso de memoria, por lo tanto si se utilizan datos de tamaño enorme, el programa se alentaría o, en el peor de los casos, no podría procesarlos. En la mayoría de los casos, sin embargo, los problemas que pudieran surgir con referencia a la lentitud en la ejecución del código, tienen solución, principalmente teniendo cuidado de vectorizar el código; ya que esto permitiría particionarlo y aprovechar en procesamiento paralelo en equipos con multi-núcleos.

[3]Por sus siglas en inglés: *The Comprehensive R Archive Network*. Su página *Web* es: http://cran.r-project.org/.

[4]Formato común de sistemas de información geográfica (GIS), introducido por la compañía ESRI en su sistema de *software* comercial ArcGIS.



lm {stats} R Documentation

Fitting Linear Models

Description

`lm` is used to fit linear models. It can be used to carry out regression, single stratum analysis of variance and analysis of covariance (although `aov` may provide a more convenient interface for these).

Usage

```
lm(formula, data, subset, weights, na.action,
   method = "qr", model = TRUE, x = FALSE, y = FALSE, qr = TRUE,
   singular.ok = TRUE, contrasts = NULL, offset, ...)
```

Arguments

`formula`	an object of class "`formula`" (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details'.
`data`	an optional data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`,

Figura 1.1: El sistema de ayuda del lenguaje R

1.5. Ayuda en R

R cuenta con una muy buena ayuda en el uso de funciones de manera muy similar al *man* de UNIX. para obtener información de cualquier función en específico, por ejemplo *lm,* el comando es:

```
help(lm)

# Una forma abreviada sería

?lm # -- comentario
```

El código anterior, muestra dos formas de invocar la ayuda en el intérprete de R, la función `help()` y el operador `'?'`. En ambos casos el resultado es el mismo. Un fragmento de la salida de ayuda que ofrece el sistema para ese tema (`lm`) se muestra en la Fig. 1.1. Aparte de esto, el lenguaje omite interpretar, en un renglón, cualquier texto que siga al símbolo `'#'`; esta es la provisión del lenguaje para incorporar comentarios en el código.

Cuando se desea información sobre caracteres especiales de R, el argumento se debe encerrar entre comillas sencillas o dobles, con la finalidad que lo identifique como una cadena de caracteres. Esto también es necesario hacer para unas cuantas palabras con significado sintáctico incluyendo al `if`, `for`, y `function`. Por ejemplo:

```
help("[[")

help('if')
```

Por otra parte, el sistema puede mostrar un listado de contenidos acerca de algún tópico cualquiera invocando la función `help.search()`, que abreviadamente se invoca con `??`. Por ejemplo, si se desea saber acerca del tópico *split*, lo que puede incluir, funciones, paquetes, variables, etc., se hace de la siguiente manera:

```
help.search("split")

# O abreviadamente:
??"split"
```

Además de esto, existen foros muy activos en diversos temas de R (*Mailing Lists*), donde seguramente podrás encontrar las respuestas apropiadas. La dirección de esos foros la puedes se encontrar en http://www.r-project.org[5].

[5]Una interfaz en la *Web* al grupo básico de R, conocido como *R-help*, se puede encontrar en http://dir.gmane.org/gmane.comp.lang.r.general.

Capítulo 2

Los datos y sus tipos

Todas las cosas que manipula R se llaman objetos. En general, éstos se construyen a partir de objetos más simples. De esta manera, se llega a los objetos más simples que son de cinco clases a las que se denomina *atómicas* y que son las siguientes:

- `character` (cadenas de caracteres)
- `numeric` (números reales)
- `integer` (números enteros)
- `complex` (números complejos)
- `logical` (lógicos o booleanos, que sólo toman los valores True o False)

En el lenguaje, sin embargo, cada uno de estas clases de datos no se encuentran ni se manejan de manera aislada, sino encapsulados dentro de la clase de objeto más básica del lenguaje: el `vector`. Un `vector` puede contener cero o más objetos, pero todos de la misma clase. En contraste, la clase denominada `list`, permite componer objetos también como una secuencia de otros objetos, pero, a diferencia del `vector`, cada uno de sus componentes puede ser de una clase distinta.

2.1. Los datos numéricos

Probablemente el principal uso de R es la manipulación de datos numéricos. El lenguaje agrupa estos datos en tres categorías, a saber: `numeric, integer` y `complex`, pero cuando se introduce algo que puede interpretar como un número, su inclinación es tratarlo como un dato de tipo `numeric`, es decir, un número de tipo real, a no ser que explícitamente se indique otra cosa. Veamos algunos ejemplos:

```
x <- 2  # Se asigna el valor 2 a x
print(x)  # Se imprime el valor de x

## [1] 2

class(x)  # Muestra cuál es la clase de x

## [1] "numeric"

x <- 6/2  # Se asigna el valor de la operación dividir 6/2 a x
print(x)

## [1] 3

class(x)

## [1] "numeric"
```

Aparentemente las dos asignaciones que se hacen, mediante el operador de asignación, `<-`, a la *variable* `x`, es de los enteros 2 y 3 respectivamente. Sin embargo, al preguntar, mediante la *función* `class()`, cuál es la clase de `x`, la respuesta es `numeric`, esto es, un número real. Para asignar explícitamente un entero, `integer`, a una variable, se agrega la letra L al final del número, como sigue:

```
x <- 23L; print(x)

## [1] 23

class(x)

## [1] "integer"
```

Aquí la variable `x` tendrá como valor el entero 23. Como una nota adicional del lenguaje, nótese que se han escrito dos expresiones de R en un mismo renglón. En este caso, las expresiones se separan mediante ';'.

Para lograr que una expresión, como la operación de división `6/2`, arroje como resultado un entero, se tiene que hacer una *conversión*; ello se logra mediante la función `as.integer`, como sigue:

```
x <- as.integer(6/2); print(x)

## [1] 3

class(x)

## [1] "integer"
```

Por su parte, los números complejos, `complex` en el lenguaje, tienen una sintaxis muy particular; misma que se tiene que emplear para indicar explícitamente que un número introducido corresponde a ese tipo:

```
x <- 21 + 2i
y <- 2i + 21 # El mismo valor que x
z <- -1 + 0i # Corresponde a -1
tt <- sqrt(z) # raíz cuadrada de -1
print(x); print(y); print(z); print(tt)

## [1] 21+2i
## [1] 21+2i
## [1] -1+0i
## [1] 0+1i

class(tt)

## [1] "complex"
```

En los ejemplos anteriores a la variable `tt` se le asigna el resultado de una función, `sqrt()`, que es la raíz cuadrada de el número `-1`. Nótese que ésta es la forma correcta de calcular esa raíz, por ejemplo, `sqrt(-1)`, hubiera arrojado como resultado un error.

También, existe un valor numérico especial, `Inf`, que representa el infinito y que puede resultar en algunas expresiones, por ejemplo:

```
x <- 1/0  # División por cero
x

## [1] Inf

# Tambien dividir un número por Inf da cero:
y <- 1/Inf
y

## [1] 0
```

Finalmente, algunas operaciones pueden resultar en algo que no es un número, esto se representa por el valor `NaN`. Veamos un ejemplo:

```
x <- 0/0
x

## [1] NaN
```

2.2. Vectores

Se ha dicho con anterioridad que las clases atómicas de datos no se manejan de manera individual. En efecto, en todos los ejemplos anteriores, el lenguaje ha creado implícitamente vectores de longitud 1, y son esos los que se han asignado a las variables. Tomemos el caso más sencillo:

```
x <- 2  # Se asigna el valor 2 a x
print(x)  # Se imprime el valor de x

## [1] 2
```

Aquí, la impresión del valor de `x` tiene una forma muy particular: "`[1] 2`". El '`[1]`' que precede al valor, indica que se trata del primer elemento y único, en este caso, del vector que se muestra.

Hay diversas maneras de crear vectores de otras longitudes, que, como se ha dicho antes, son secuencias de objetos de la misma clase atómica. En las siguientes secciones se verán algunos casos.

2.2.1. El uso de la función `c()` para crear vectores

La primer manera de crear vectores es a partir de los elementos individuales que compondrán el vector. Para esto se utiliza la función `c()` como se muestra a continuación.

```
c(4,2,-8)  # Creación de un vector sin asignarlo a una variable

## [1]  4  2 -8

## -----------------
## Distintas formas de asignar un vector a una variable
u <- c(4,2,-8) # Usando el operador <-
c(4, 2, -8) -> v # Usando el operador ->
# Usando la función assign:
assign("w", c(4, 2, -8))
p = c(4, 2, -8) # Usando el operador =
print(u); print(v); print(w); print(p)

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

La función `c()` sirve para concatenar varios elementos del mismo tipo. En todos los ejemplos mostrados, la impresión del vector se hace en un renglón que comienza con el símbolo '`[1]`', indicando con ello que el primer elemento del renglón corresponde al primer elemento del vector.

Un caso muy particular de asignación, es el de la función `assign()`. A diferencia de los otros casos vistos en el ejemplo anterior, el nombre de la variable aparece entre comillas.

Más adelante, en la página 20, se verá como la función `c()` también se puede utilizar para la creación de vectores a partir de otros vectores.

2.2.2. Creación de vectores a partir de archivos de texto - la función `scan()`

Otra manera de crear un vector es a partir de un archivo de texto. Sea, por ejemplo, el caso del archivo UnVec.txt, que se contiene la siguiente información:

```
  12 15.5 3.1
  -2.2 0 0.0007
```

Supóngase ahora que a partir de esos datos se quiere crear un vector. Para eso se usa la función `scan()`, como se muestra a continuación:

```
vec <- scan("UnVec.txt")
print(vec)

## [1] 12.0000 15.5000  3.1000 -2.2000  0.0000  0.0007
```

Desde luego que hay otras funciones para lectura de archivos, más complejas, pero baste por el momento con el uso de esta función, tal como se muestra, para permitir la creación de un vector a partir de los datos contenidos en un archivo de texto. Por el ahora, la única nota adicional es que la función `scan()` ofrece la posibilidad de indicarle explícitamente el tipo de vector que se quiere crear. Así por ejemplo, la creación de un vector de enteros se hace de la siguiente manera:

```
vec <- scan("IntVec.txt", integer())
print(vec); class(vec) # El vector y su clase

## [1]   4   3  -2   1   0 200  -8  20
## [1] "integer"
```

Por supuesto que en este caso, se debe prever que el archivo leído contenga datos que puedan ser interpretados como números enteros.

La función inversa, en este caso, de la función scan(), es la función write. Así, un vector cualquiera fácilmente se puede escribir en un archivo de texto, como se muestra a continuación:

```
vv <- c(5, 6.6, -7.7)
write(vv, "OtroArchivo.txt")
```