Básicos de R

Paula Vargas Pellicer

10/02/2022

Fundamentos del lenguaje R

Hay una enorme variedad de cosas que R puede hacer, y una de las partes más difíciles de aprender R es encontrar tu camino alrededor del mismo, no existe un orden obvio en el que diferentes personas querrán aprender los diferentes componentes del lenguaje R. Yo sugiero trabajar el material en el orden presentado, porque las secciones sucesivas se basan en los conocimientos adquiridos en las secciones anteriores.

El indicador de pantalla > es una invitación para poner a trabajar a R.

Cálculos

Puedes usar una línea de comando como calculadora así:

```
log(42/7.3)
```

Cuando una o más líneas están incompletas (por ejemplo, con una coma final, un operador o con más paréntesis a la izquierda que los paréntesis derechos, lo que implica que seguirán más paréntesis derechos), R espera la continuación y aparecerá un +

```
5+6+3+6+4+2+4+8+
```

Ese + no es una suma aritmética.

Si quieres terminar la línea agrega lo que falta

```
3+2+7
```

o presiona Esc

Se pueden colocar dos o más expresiones en una sola línea siempre que estén separadas por punto y coma:

```
2+3; 5*7; 3-7
```

Para números muy grandes o muy pequeños, R usa exponentes por “default”:

```
0.00000005
```

Números complejos

Los números complejos constan de una parte real y una parte imaginaria, que se identifican con i minúscula así:

```
z <- 3.5-8i
```

Las funciones trigonométricas, logarítmicas, exponenciales, raíces cuadradas e hiperbólicas elementales son todas implementado para valores complejos. Las siguientes son las funciones especiales de R que puede usar con números complejos. Determinar la parte real:

```
re(z)

## Error in re(z): could not find function "re"
```

Determinar la parte imaginaria:

```
Im(z)
```

Redondear

Se pueden realizar fácilmente varios tipos de redondeo (redondeo hacia arriba, redondeo hacia abajo, redondeo al entero más cercano): Para redondear hacia abajo:

```
floor(5.7)
```

hacia arriba

```
ceiling(5.7)
```

Se puede redondear al entero más cercano sumando 0.5 al número y luego usando ´*floor()*´. Hay una función incorporada para esto, pero podemos escribir fácilmente una propio para introducir la noción de escritura de funciones. Llamémosla ´rounded´

```
rounded<- function(x) floor(x+0.5)
```

Una vez definida la función, la podemos usar

```
rounded(5.7)
rounded(5.4)
```

La parte difícil es decidir cómo quieres redondear los números negativos, porque el concepto de arriba y abajo es más sutil (recuerda que –5 es un número mayor que –6). Necesitas pensar, en cambio, si quieres redondear hacia cero o lejos de cero. Para números negativos, redondear hacia arriba significa redondear hacia cero así que no te sorprendas cuando el valor de la parte positiva sea diferente:

```
ceiling(-5.7)
floor(-5.7)
```

o simplemente puede eliminar la parte decimal del número usando la función ´*trunc()*´, que devuelve enteros al truncar los valores en x hacia cero

```
trunc(5.7)
trunc(-5.7)
```

Hay una función de R llamada *´round()´* que se puede usar especificando los lugares decimales en el segundo argumento:

```
round(5.7,0)
round(5.8765, 2)
round(-5.766,1)
```

El número de lugares decimales no es lo mismo que el número de dígitos significativos. Puedes controlar el número de dígitos significativos en un número usando la función *´signif()´*. Toma un número grande como 12 345 678

```
signif(12345678,4)
signif(12345678,5)
signif(12345678,6)
```

Aritmética

Como ya lo vimos, el indicador de pantalla en R es una calculadora completamente funcional. Puedes sumar y restar usando los símbolos + y -, mientras que la división se logra con una barra inclinada / y la multiplicación se realiza mediante el uso de un asterisco *

```
7 + 3 - 5 * 2
```

Observa que en este ejemplo la multiplicación (5 × 2) se realiza antes que las sumas y restas. La raíz cuadrada o cúbica usan el símbolo de intercalación ^ y se hacen antes de la multiplicación o división

```
3^2/2
```

La función 'log()' hace el logaritmo natural de la base e (e = 2.718 282), para lo cual la función antilogaritmo es *'exp()'*:

```
log(10)
exp(1)
```

tambien se puede hacer en base 10

```
log10(6)
```

Los registros a otras bases son posibles pero engorrosos, pues hay que proporcionar a la función de registro un segundo argumento que es la base de los registros que deseas usar Supon que deseas hacer el logaritmo en base 3 de 9

```
log(9,3)
```

Las funciones trigonométricas en R miden ángulos en radianes. Un círculo mide 2π radianes, y esto es 360 ∘ , por lo que un ángulo recto (90 ∘) es π/2 radianes. R conoce el valor de π como 'pi'

```
pi
sin(pi/2) #seno de un angulo recto
cos(pi/2) # coseno de un angulo recto
```

Ojo, el coseno de un ángulo recto no resulta exactamente cero, aunque el seno resultó como exactamente 1. El e-017 significa 'veces 10^{-17}'. Si bien este es un número muy pequeño, claramente no es exactamente cero.

Módulos y cocientes enteros

Los cocientes enteros y los residuos se obtienen usando la notación %/% (porcentaje, división, porcentaje) y %% (porcentaje, porcentaje) respectivamente. Supongamos que queremos saber la parte entera de una división: digamos, cuántos 13s hay en 119

```
119 %/% 13
```

Ahora supongamos que quisiéramos saber el resto (lo que queda cuando 119 se divide entre 13): en matemáticas esto se conoce como módulo:

```
119 %% 13
```

Módulo es relativamente útil para probar si los números son pares o impares: los números impares tienen módulo 2 valor 1 y los números pares tienen módulo 2 valor 0

```
9%%2
8%%2
```

Del mismo modo, se puede usar módulo para probar si un número es un múltiplo exacto de algún otro número. Por ejemplo, si 15 421 es un múltiplo de 7 (que lo es)

```
15421 %% 7 == 0
```

el uso de '==' para probar la igualdad

Tabla1. Listado de algunas funciones

Function	Meaning
`log(x)`	log to base e of x
`exp(x)`	antilog of x (e^x)
`log(x,n)`	log to base n of x
`log10(x)`	log to base 10 of x
`sqrt(x)`	square root of x
`factorial(x)`	$x! = x \times (x-1) \times (x-2) \times \cdots \times 3 \times 2$
`choose(n,x)`	binomial coefficients $n!/(x!\ (n-x)!)$
`gamma(x)`	$\Gamma(x)$, for real x $(x–1)!$, for integer x
`lgamma(x)`	natural log of $\Gamma(x)$
`floor(x)`	greatest integer less than x
`ceiling(x)`	smallest integer greater than x
`trunc(x)`	closest integer to x between x and 0, e.g. trunc(1.5) = 1, trunc(–1.5) = –1; trunc is like floor for positive values and like ceiling for negative values
`round(x, digits=0)`	round the value of x to an integer
`signif(x, digits=6)`	give x to 6 digits in scientific notation
`runif(n)`	generates n random numbers between 0 and 1 from a uniform distribution
`cos(x)`	cosine of x in radians
`sin(x)`	sine of x in radians
`tan(x)`	tangent of x in radians
`acos(x), asin(x), atan(x)`	inverse trigonometric transformations of real or complex numbers
`acosh(x), asinh(x), atanh(x)`	inverse hyperbolic trigonometric transformations of real or complex numbers
`abs(x)`	the absolute value of x, ignoring the minus sign if there is one

Nombrar y asignar variables

Hay tres cosas importantes que recordar al seleccionar nombres para las variables en R: - Los nombres de variables en R distinguen entre mayúsculas y minúsculas, por lo que y no es lo mismo que Y. - Los nombres de variables no deben comenzar con números (p. ej., 1x) o símbolos (p. ej., %x). - Los nombres de variables no deben contener espacios en blanco (utiliza, por ejemplo ´back.pay´ no ´back pay´).

Por practicidad haz los nombres de tus variables lo más cortos posible. Los objetos obtienen valores en R por asignación. Esto se logra con la flecha <- o con =, sin embargo, el igual puede llevar a ambigüedades

```
x<-5
y=4
```

Operadores

Varios de estos operadores tienen un significado diferente dentro de las fórmulas de un modelo. Por ejemplo, * indica los efectos principales más una interacción (en lugar de multiplicación), : indica la interacción entre dos variables (en lugar de generar una secuencia) y ^ significa todas las interacciones hasta la potencia indicada (en lugar de elevar a la potencia). Pero esto ya lo veremos mas adelante

Vectores enteros

Los vectores enteros existen para poder ser trasladados a otros lenguajes de programación. Ten cuidado. No intente cambiar la clase de un vector utilizando la función *'integer()*'

```
x<- c(5,3,7,8)
is.integer(x)
is.numeric(x)
x<- integer(x)

## Error in integer(x): invalid 'length' argument
```

Aplicarle la función *'integer()*' reemplaza todos sus números con ceros; definitivamente no es lo que pretendías.

Primero haz el objeto numérico, luego convierta el objeto en entero usando la función *´as.integer()´* como ésta

```
x <- c(5,3,7,8)
x <- as.integer(x)
is.integer(x)
```

Factores

son variables categóricas que tienen un número fijo de niveles. Un ejemplo simple de un factor podría ser una variable denominada género con dos niveles: 'femenino' y 'masculino'. Si tuvieras tres hembras y dos machos, podría crear el factor como ste:

```
gender <- factor(c("female", "male", "female", "male", "female"))
class(gender)
```

Pongamos un ejemplo de una base de datos de R.

```
data <- iris
head(data)
```

Esta base de datos contiene variables de respuesta continua y variable explicativa categórica, la cual es un factor. En el modelado estadístico, los factores están asociados a análisis de varianza (todas las variables explicativas son categóricas) y análisis de covarianza (algunas de las variables explicativas son categóricas y otras son continuas)

A menudo querrás comprobar que una variable es un factor (especialmente si los niveles del factor son números en lugar de caracteres):

```
is.factor(data$Species)

levels(data$Species)

nlevels(data$Species)

length(levels(data$Species))
```

De forma predeterminada, los niveles de los factores se tratan en orden alfabético. Si quieres cambiar esto (para ordenar las barras de un gráfico de barras), entonces escribe los niveles de los factores en el orden que deseas que se utilicen y proporciona este vector como el segundo argumento de la función de factor.

```
tapply(data$Petal.Length, data$Species,mean)

data$Species <-
factor(data$Species,levels=c("virginica","versicolor","setosa"))
```