Actividad 2A: Introducción al lenguaje R en Rstudio
Estadística

Luis Fernando Izquierdo Berdugo

22 de Agosto del 2024

1. Inciso 1

Realizar las lecciones de la librería interactiva swirl de r en Rstudio.

- **5: Missing Values**
- **6: Subsetting Vectors**
- **7: Matrices and Data Frames**
- **13: Simulations.**

Entregar en un pdf por lo menos 3 ejemplos de ejercicios realizados para cada una de las diferentes lecciones.

1.1. Missing Values

En esta lección se hicieron ejercicios con Valores Faltantes ("Missing Values"), ya que en el análisis de datos no deben ser ignorados, si no tratados de manera pertinente.

Se crea un vector con valores faltantes NA y se guarda en la variable x y se multiplica por 3

```
x <- c(44, NA, 5, NA)
x * 3
```

Lo cual arroja como resultado

```
[1] 132  NA  15  NA
```

Esto demuestra que los elementos del vector mantienen sus multiplicaciones incluso si existen valores NA

Se crea un vector que contiene 1000 extracciones de una distribución normal estándar y se guarda en la variable y. De igual manera, se crea un vector z que contiene 1000 valores de NA y se seleccionan 100 elementos aleatorios de estos 2000 valores, con la finalidad de no saber cuantos valores de NA se obtendrán y que posiciones ocuparán en el vector final, esto se guardará en la variable `my_data`

```
y <- rnorm(1000)
z <- rep(NA, 1000)
my_data <- sample(c(y,z), 100)
```

Se guardará en la variable `my_na` las posiciones de los NA en la variable creada previamente.

```
my_na <- is.na(my_data)
my_na
```

El resultado será un vector con los mismos elementos que `my_data`, pero tendrán 'TRUE' en todos los elementos que son NA y 'FALSE' en el caso contrario.

```
  [1]  TRUE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
 [13]  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE
 [25] FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE  TRUE
 [37] FALSE  TRUE  TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE
 [49]  TRUE FALSE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE FALSE FALSE
 [61]  TRUE FALSE FALSE FALSE  TRUE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE
 [73]  TRUE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE FALSE
 [85]  TRUE  TRUE FALSE FALSE  TRUE  TRUE FALSE FALSE  TRUE FALSE  TRUE  TRUE
 [97]  TRUE  TRUE  TRUE  TRUE
```

Con el vector anterior, se hará una suma que devolverá la cantidad de NA en el vector original, obteniendo 53 como resultado.

```
sum(my_na)
[1] 53
```

Revisando la información del vector `my_data` podemos observar que esto es correcto:

```
   my_data
  [1]         NA         NA  1.69319278         NA  0.29630046  0.21949467
  [7] -0.24757350 -0.03868213 -0.49302253 -0.31857913  0.75842386         NA
 [13]         NA -0.83557217         NA -1.04568846 -0.29369191  0.39463849
 [19] -1.73389289 -0.71721539 -0.75550600         NA -0.19095228         NA
 [25]  2.01632112  0.29448997         NA         NA         NA         NA
 [31]         NA -0.73673481  1.37232690  0.56827586         NA         NA
 [37]  1.42708829         NA         NA  1.08696507         NA  0.15443125
```

```
 [43]          NA          NA          NA          NA -0.56167247          NA
 [49]          NA -0.96640157  1.05949281          NA          NA          NA
 [55] -0.04851078          NA          NA          NA  0.68065277  1.35801612
 [61]          NA -1.01562402 -1.12056272  0.90403314          NA  0.80507340
 [67]          NA          NA  0.15234025 -0.58383470  0.16378129  0.09480226
 [73]          NA          NA -0.14478870          NA          NA          NA
 [79]  0.24609503          NA          NA -0.39170552          NA -0.24957482
 [85]          NA          NA  0.59403545 -0.04255236          NA          NA
 [91] -1.09251697  1.41977623          NA  0.56409803          NA          NA
 [97]          NA          NA          NA          NA
```

Otro tipo de dato faltante (missing value) es `NaN` que significa "Not a Number", este se puede obtener con una división entre 0 o una resta de Infinitos.

```
    0/0
[1] NaN

    Inf - Inf
[1] NaN
```

1.2. Subsetting Vectors

En esta lección se observó como extraer elementos de un vector basado en condiciones específicas. Se inicio con un vector x preestablecido:

```
    x
 [1]          NA          NA  0.2647045          NA -1.3982708          NA          NA
 [8]          NA -0.2940722  1.1899344          NA -0.4697394 -0.2826327 -0.4219074
[15]          NA          NA          NA -0.9979604 -0.4339687          NA          NA
[22]          NA  1.8487704  2.0333438  0.5989772          NA  1.4430054 -2.0621025
[29] -0.3483517          NA -0.7848110          NA  0.2051519  0.7314924          NA
[36]          NA -1.8909420 -0.5852233          NA          NA
```

Se hace la selección específica de los elementos 1 a 10 del vector x:

```
    x[1:10]
 [1]          NA          NA  0.2647045          NA -1.3982708          NA          NA
 [8]          NA -0.2940722  1.1899344
```

Se usa la función `is.na` como parte de una selección que solamente devolverá los NA del archivo.

```
    x[is.na(x)]
[1] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
```

De igual manera, se hizo la parte contraria del ejemplo anterior y se depositó en la variable y

```
y <- x[!is.na(x)]
y
 [1]  0.2647045 -1.3982708 -0.2940722  1.1899344 -0.4697394 -0.2826327 -0.4219074
 [8] -0.9979604 -0.4339687  1.8487704  2.0333438  0.5989772  1.4430054 -2.0621025
[15] -0.3483517 -0.7848110  0.2051519  0.7314924 -1.8909420 -0.5852233
```

Se hizo la selección para los valores de y mayores a 0.

```
y[y>0]
[1] 0.2647045 1.1899344 1.8487704 2.0333438 0.5989772 1.4430054 0.2051519
[8] 0.7314924
```

Una parte importante fue excluir los **NA** ya que estos no son valores, entonces la evaluación de la expresión **NA** > 0 da como resultado **NA**

```
x[x>0]
 [1]        NA        NA 0.2647045        NA        NA        NA        NA
 [8] 1.1899344        NA        NA        NA        NA        NA        NA
[15]        NA 1.8487704 2.0333438 0.5989772        NA 1.4430054        NA
[22]        NA 0.2051519 0.7314924        NA        NA        NA        NA
```

El proceso de los mayores a 0 se puede ejecutar con una sola ejecución:

```
x[!is.na(x) & x>0]
[1] 0.2647045 1.1899344 1.8487704 2.0333438 0.5989772 1.4430054 0.2051519
[8] 0.7314924
```

En R se puede usar índices negativos para excluir valores.

```
x[c(-2,-10)]
 [1]        NA  0.2647045        NA -1.3982708        NA        NA        NA
 [8] -0.2940722        NA -0.4697394 -0.2826327 -0.4219074        NA        NA
[15]        NA -0.9979604 -0.4339687        NA        NA        NA  1.8487704
[22]  2.0333438  0.5989772        NA  1.4430054 -2.0621025 -0.3483517        NA
[29] -0.7848110        NA  0.2051519  0.7314924        NA        NA -1.8909420
[36] -0.5852233        NA        NA

x[-c(2,10)]
 [1]        NA  0.2647045        NA -1.3982708        NA        NA        NA
 [8] -0.2940722        NA -0.4697394 -0.2826327 -0.4219074        NA        NA
[15]        NA -0.9979604 -0.4339687        NA        NA        NA  1.8487704
[22]  2.0333438  0.5989772        NA  1.4430054 -2.0621025 -0.3483517        NA
[29] -0.7848110        NA  0.2051519  0.7314924        NA        NA -1.8909420
[36] -0.5852233        NA        NA
```

Se pueden crear vectores con elementos nombrados

```
vect <- c(foo = 11, bar = 2, norf = NA)
vect
```

```
foo  bar norf
 11    2   NA
```

De igual manera, se pueden pasar nombre a un vector creado previamente

```
vect2 <- c(11, 2, NA)
names(vect2) <- c("foo", "bar", "norf")
```

1.3. Matrices and Data Frames

Las matrices y data frames son tipos de datos rectangulares"lo cual significa que se usan para guardar datos tabulares con líneas y columnas. La principal diferencia es que las matrices solamente pueden contener un tipo de dato y los data frames consisten de registros con tipos de datos diferentes. Se puede crear un vector de la forma tradicional y después asignarle una dimensión para convertirlo en una matriz.

```
my_vector <- 1:20
my_vector
```

```
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

```
dim(my_vector) <- c(4,5)
my_vector
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    2    6   10   14   18
[3,]    3    7   11   15   19
[4,]    4    8   12   16   20
```

Se confirma que sea una matriz y se guarda en la variable `my_matrix`

```
class(my_vector)
```

```
[1] "matrix" "array"
```

```
my_matrix <- my_vector
```

Si a una matriz se le pasa un vector de caracteres o strings, la matriz se convertirá automáticamente en puros caracteres, esto es una coerción implícita.

```
    patients <- c("Bill", "Gina", "Kelly", "Sean")
    cbind(patients, my_matrix)
     patients
[1,] "Bill"   "1" "5" "9"  "13" "17"
[2,] "Gina"   "2" "6" "10" "14" "18"
[3,] "Kelly"  "3" "7" "11" "15" "19"
[4,] "Sean"   "4" "8" "12" "16" "20"
```

Para estos casos es mejor usar un data frame:

```
    my_data <- data.frame(patients, my_matrix)
    my_data
  patients X1 X2 X3 X4 X5
1     Bill  1  5  9 13 17
2     Gina  2  6 10 14 18
3    Kelly  3  7 11 15 19
4     Sean  4  8 12 16 20
```

La adición de nombre a las columnas se hace con la función `colnames`

```
    cnames <- c("patient", "age", "weight", "bp", "rating", "test")
    colnames(my_data) <- cnames
    my_data
   patient age weight bp rating test
1    Bill   1      5  9     13   17
2    Gina   2      6 10     14   18
3   Kelly   3      7 11     15   19
4    Sean   4      8 12     16   20
```

1.4. Simulations

R puede simular correctamente números aleatorios y funciones existentes comunes. Con la función `sample` se puede simular números aleatorios, por ejemplo, cuatro tiros de un dado de 6 caras.

```
    sample(1:6, 4, replace = TRUE)
[1] 4 6 1 5
```

El primer argumento indica los elementos de donde se puede elegir información, el segundo argumento indica la cantidad de tiros que se harán y el tercer argumento indica si la muestra debe ser con reemplazo, que cada que se tira una opción, esta vuelve a estar disponible para el siguiente tiro.

La función `sample` también se puede usar para reordenar los elementos de un vector. En el ejemplo siguiente el vector **`LETTERS`** incluye las letras del abecedario en orden.

```
    sample(LETTERS)
 [1] "O" "J" "K" "U" "I" "G" "B" "M" "A" "V" "X" "S" "Q" "Y" "D" "E" "L" "N" "W"
[20] "C" "T" "F" "H" "R" "P" "Z"
```

De igual manera, la función puede tomar valores de probabilidad para cada opción específica. Se puede simular tiros de una moneda cargada para la cara, tomando en cuenta que 0 es cruz y 1 es cara.

```
    flips <- sample(c(0,1), 100, replace=TRUE,  prob = c(0.3,0.7))
    flips
  [1] 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 0 0 1 1 0 0 0 1 0 0 1 1 1 1 0 1 1 1 1 1
 [39] 0 0 0 1 0 0 0 1 0 1 0 0 1 0 1 0 1 1 1 1 1 0 1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 1
 [77] 0 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 0 0 1 0 1 0 1

    sum(flips)
[1] 66
```

Como se puede observar en la suma del vector, se obtuvieron 66 tiros de cara, lo cual indica que si se añadieron correctamente las probabilidades de 0.3 para la cruz y 0.7 para la cara.

La variable aleatoria binomial representa el número de éxitos en un número de tiros independientes. A continuación se generarán 100 casos de 1 tiro de una moneda cargada (como en el ejercicio anterior), donde el éxito será que se obtenga cara.

```
    flips2 <- rbinom(100, size = 1, prob = 0.7
    flips2
  [1] 1 0 0 1 1 1 0 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 0 1 1 0 1 0 1 1 1 1 1 0
 [39] 0 1 1 0 1 0 0 1 1 0 0 0 1 0 0 0 1 1 1 1 0 0 1 0 1 1 1 1 0 1 1 1 0 1 1 1 1 0
 [77] 1 1 1 0 1 1 1 1 1 0 1 0 1 0 1 0 1 1 1 1 0 1 1 1

    sum(flips2)
[1] 68
```

También se puede simular la distribución estándar. Por predeterminado esta vendrá con media de 0 y desviación estándar de 1.

```
    rnorm(10)
 [1] -0.3323882 -1.3480719  0.1648232  0.5571825  0.8155565 -0.3548919 -0.2589720
 [8] -1.4874112 -0.9435833  0.0736770
```

Esto generó 1o numeros aleatorios de una distribución estándar normal. Si se quiere generar lo mismo pero con media de 100 y desviación estándar de 25, se haría lo siguiente:

```
    rnorm(10, mean = 100, sd = 25)
 [1] 101.92227  91.57630  95.95732  96.44125  94.78948  90.61180 106.88557
 [8]  83.26655 121.27102 101.99961
```

Otro ejemplo de función que se puede generar con R es la distribución de Poisson.

```
rpois(5, lambda = 10)
```

```
[1] 12 10 11 12 10
```

Se puede usar la función `replicate` para hacer esta operación 100 veces. Esta función creará una matriz donde cada columna tendrá 5 números aleatorios generados por una distribución de Poisson con media de 10.

```
my_pois <- replicate(100, rpois(5, 10))
my_pois
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
[1,]   13    9    9    8   11    6   14    6    3    12    13    14    13     5
[2,]    9    5   11   17    6   16    9   13   15     7     6     9     9     6
[3,]   13    9   14   13    8   10   11    5    5     5     9    14     5     9
[4,]   14   10    9   11    6   11   11   10   13     8     7     8    12     8
[5,]    7    9    8   10    6    9    8    3   13     7    11    10     6     9
     [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25] [,26]
[1,]     7     7     6     5     8    10    14    12     9     9    12     9
[2,]    12    11    11    16    14    19     8    10    15    10    10     4
[3,]    12     7    10     8     6     9     7    14     7     8     9     4
[4,]     7     8     6    14     7    13     8    19    15    14    10     8
[5,]     7     9    10     5    12    18     9    11     8    10    12    10
     [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35] [,36] [,37] [,38]
[1,]    10    14    10    14    11    10     8     6     9    12    15    12
[2,]     3     8    13    11    15    15     7     7    13    10    16     8
[3,]    13    11    11     7    16     7     8     4     9     8     6    11
[4,]     7    11    21     9    13     4    13    14    15    11    11     7
[5,]    12     4    10     7    11    11     6    12     9    12     8    13
     [,39] [,40] [,41] [,42] [,43] [,44] [,45] [,46] [,47] [,48] [,49] [,50]
[1,]     9    12    11    16     7    13    12    13    10    10     7     9
[2,]     5    16     8     6    13    10    12    12    12    13     8     5
[3,]    10     9     9    10    14    10     8    10     8    12    10    13
[4,]    16    13     5    11     8    10     7    13     9    10    10    11
[5,]     6    16     4    12     7    12     7    10     7    20     9    12
     [,51] [,52] [,53] [,54] [,55] [,56] [,57] [,58] [,59] [,60] [,61] [,62]
[1,]     9     5    10    13     5    10    10    10    14     5    12     9
[2,]     8     8     8    16     8     9     7     8     5    10    11    19
[3,]    13     6    10    12     5     7    12     6     6    16    10    13
[4,]    12    12     8     7    13    14    12     9    14     6     5     9
[5,]     7    10    13    12     6     9     7     7    12    13    17    11
     [,63] [,64] [,65] [,66] [,67] [,68] [,69] [,70] [,71] [,72] [,73] [,74]
[1,]     1     9    12    12    13     2     8    12     6     8    11    14
[2,]    14    11    13     6     8    12    12    11    12    13     9    11
```

```
[3,]    13     8     7     9     8    12    14    13    14    15     9     8
[4,]    11     8     9    12    10    10     9     9     6     8    11     7
[5,]     6     7     6    16    10     5     8     5     7    13    11     8
     [,75] [,76] [,77] [,78] [,79] [,80] [,81] [,82] [,83] [,84] [,85] [,86]
[1,]    11    14     5    13    10    11     7     8    17     9    12     9
[2,]    15    28    10    15    16    12    10     9    16     8    11     7
[3,]     6     7    10    12     7     5    17    19    14    16    12    12
[4,]    10     8     5    10    11    10    10    12    14     9     6     9
[5,]    15    12     8    17     9     5     7    11     5     6    12     5
     [,87] [,88] [,89] [,90] [,91] [,92] [,93] [,94] [,95] [,96] [,97] [,98]
[1,]    13     8     9    13     7     9    12     8     9     5    11    16
[2,]     5    10    14    17     5    11     7     8    14     8    13     6
[3,]    11     7     5    11    14     8    11    10     5    11    13     7
[4,]    18     7    11    13     6    12     8     5    12     9    15     7
[5,]     9     6    11    10    10     8    10     9     9    19    15     4
     [,99] [,100]
[1,]     5      9
[2,]    21     12
[3,]    14     11
[4,]     5     12
[5,]     7     13
```

Se puede obtener el promedio de cada columna en la matriz usando la función `colMeans`, de igual manera se puede generar su histograma con la función `hist`

```
cm <- colMeans(my_pois)
hist(cm)
```

El histograma generado es el siguiente:

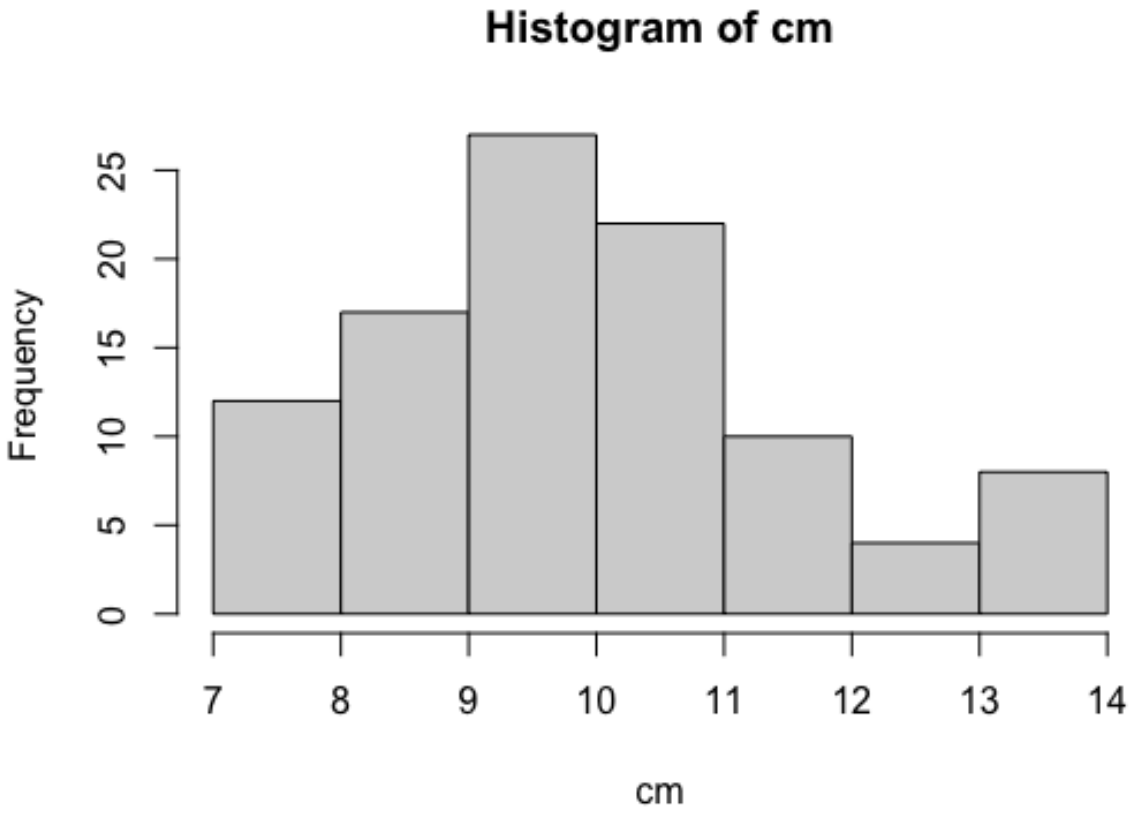

Figura 1: Histograma del promedio de las columnas del vector `my_pois`