

Este tutorial introduce al
participante en el software
estadístico R.



01 – Introducción al Entorno

Versión 1.1 - 20170607

Dr.Ing. Mario José Diván
mjdivan@divsar.com.ar

Índice

Índice	1
Lista de Figuras	1
Introducción	2
Pre-requisitos	2
Iniciando y trabajando en una sesión	2

Lista de Figuras

Figura 1. Sesión iniciada	2
Figura 2. Vista parcial de los comandos ingresados a la sesión	3
Figura 3. Creando un vector	4
Figura 4. Creando vector desde otros vectores	4
Figura 5. Utilizando las funciones integradas	5
Figura 6. Utilización de las funciones como parte de las expresiones	5
Figura 7. Ordenamiento y mediana	5
Figura 8. Uso de secuencias y comentarios	6
Figura 9. Salida de la sesión por R	6
Figura 10. Acceso directo al contenido del vector	7
Figura 11. Suma parcial de los datos del vector	7
Figura 12. Suma parcial del vector excluyendo posiciones	8
Figura 13. Calculo de la media en presencia de valores faltantes	8
Figura 14. Reemplazando los nulos en un vector	8
Figura 15. Conversión entero - caracter	9
Figura 16. Verificación y conversión a numérico	9
Figura 17. Ajuste dinámico del tamaño de los vectores	10
Figura 18. Creando arreglos	10
Figura 19. Verificando los nulos de una matriz	11
Figura 20. Obteniendo submatrices	11
Figura 21. Convirtiendo un vector en matriz	11
Figura 22. Utilizando el vector para completar la matriz	12
Figura 23. Inicializando la matriz con un valor dado	12
Figura 24. Producto matricial	12
Figura 25. Producto externo entre arreglos	13
Figura 26. Diagonal Principal y Matrices	14
Figura 27. Operaciones directas sobre filas y columnas de la matriz	15

R

Introducción al Entorno

Introducción

La presente guía tiene por objetivo guiar al participante dentro de los primeros pasos en el entorno del software R[1]. R es un software estadístico de libre utilización orientado a la generación de gráficos y cómputo estadístico con posibilidad de scripting. El mismo corre sobre una amplia variedad de plataformas Unix, Windows y MacOS. El sitio oficial del proyecto es www.r-project.org, cuenta con una amplia variedad de paquetes para extender su funcionalidad y posibilidades de importar/exportar datos desde una vasta cantidad de orígenes.

Pre-requisitos

Haber culminado el tutorial “Introducción a SparkR con Apache Spark”.-

Iniciando y trabajando en una sesión

1. Inicie R

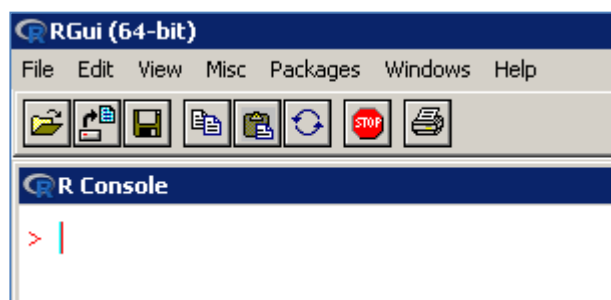


Figura 1. Sesión iniciada

Cada vez que opere con R verá que existe una sesión dentro de la cual Ud. cargará datos, exportará datos, definirá variables o realizará las acciones que crea pertinente. Note que la sesión se indica mediante una línea de comandos identificada por un “>” junto con un cursor titilante a su derecha a la espera de las instrucciones que desee ingresar.

2. Pruebe los siguientes **operadores**, ingresándolos mediante línea de comandos:

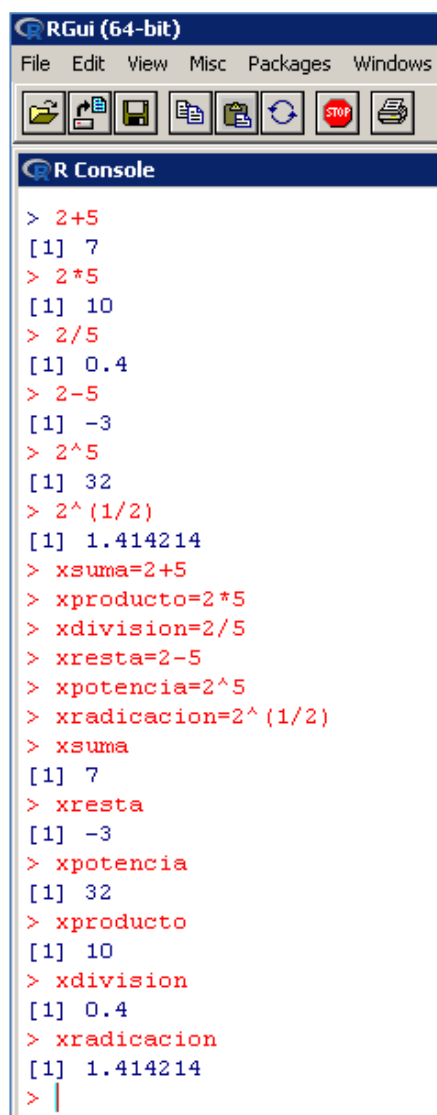
Operador	Descripción	Interpretación	Ejemplo en R
+	Suma	2+5	2+5
-	Resta	2-5	2-5
/	División	$\frac{2}{5}$	2/5
*	Producto	2*5	2*5
^	Potencia	2 ⁵	2^5
	Radicación (Exponente Fraccional)	$\sqrt[2]{2}$	2^(1/2)

3. Ahora bien, normalmente necesitará almacenar el resultado en alguna variable, motivo por el cual pruebe almacenar los resultados de las operaciones del punto anterior en las variables: `xsuma`, `xresta`, `xdivision`, `xproducto`, `xpotencia` y `xradicacion` respectivamente. La forma general para asignar el resultado de una operación a la variable es como sigue:

<variable_nombre> = <operación>

`xsuma = 2+5`

4. Ahora bien, ha asignado el resultado de una operación a la variable pero no ha visualizado el valor almacenado en la variable. Si desea ver por consola el resultado de la variable, simplemente ingrese su nombre y presione ENTER.



```

RGui (64-bit)
File Edit View Misc Packages Windows

R Console

> 2+5
[1] 7
> 2*5
[1] 10
> 2/5
[1] 0.4
> 2-5
[1] -3
> 2^5
[1] 32
> 2^(1/2)
[1] 1.414214
> xsuma=2+5
> xproducto=2*5
> xdivision=2/5
> xresta=2-5
> xpotencia=2^5
> xradicacion=2^(1/2)
> xsuma
[1] 7
> xresta
[1] -3
> xpotencia
[1] 32
> xproducto
[1] 10
> xdivision
[1] 0.4
> xradicacion
[1] 1.414214
>

```

Figura 2. Vista parcial de los comandos ingresados a la sesión

5. Claro que cuando almacena resultados en una variable es porque posiblemente desee utilizarlo en lo inmediato ¿podría utilizar una variable como parte de una

operación? En efecto, suponga que deseamos obtener como resultado la variable `x` elevada a la 4:

- a. Escriba: **`xradicacion^4`**
- b. Verá que el resultado es 4 ¿por qué?
 - i. `xradicacion` es el resultado de $\sqrt[2]{2} = 2^{1/2}$
 - ii. Al elevarlo a la 4 la expresión queda $2^{(\frac{1}{2}) * 4} = 2^2 = 4$
 - iii. Como puede apreciar, la variable es empleada naturalmente como parte de la expresión!

6. La forma de indicar **rangos** en R es similar al empleado en Excel, es decir:
`<nro inicial>:<nro final>`

Escriba **`1:7`** en consola y presione ENTER

7. R opera sobre **estructuras denominadas**, esto es que existe un nombre asignado a la variable, vector u otra estructura a través del cual referirla. Para crear un vector a partir de una secuencia de números, puede utilizar la función "**`c`**" que concatena los argumentos llevándola a la forma vectorial:

`x = c(10.4, 5.6, 3.1, 6.4, 21.7)`

```
> x = c(10.4, 5.6, 3.1, 6.4, 21.7)
> x
[1] 10.4 5.6 3.1 6.4 21.7
```

Figura 3. Creando un vector

8. Note que puede generar otras estructuras denominadas a partir de otra estructura denominada, es decir, pruebe escribir la siguiente expresión en su sesión:

`nuevo = c(x, 0, x)`

```
> nuevo = c(x, 0, x)
> nuevo
[1] 10.4 5.6 3.1 6.4 21.7 0.0 10.4 5.6 3.1 6.4 21.7
> |
```

Figura 4. Creando vector desde otros vectores

9. En un lenguaje guiado por scripting e interactivo las funciones con rutinas conocidas e integradas son esenciales. ¿Cómo invocarlas? Se escribe el nombre de la función deseada y a continuación, entre paréntesis, van separado por "coma" los argumentos. Por ejemplo, pruebe los ejemplos esgrimidos en la figura 5 en su sesión aprovechando los vectores `x` y `nuevo`:

```

> max(nuevo)
[1] 21.7
> min(nuevo)
[1] 0
> length(x)
[1] 5
> length(nuevo)
[1] 11
> mean(nuevo)
[1] 8.581818
> sd(nuevo)
[1] 7.154274

```

Figura 5. Utilizando las funciones integradas

Note que la primera función (**max**) toma el vector **nuevo** y retorna el mayor valor (21.7). En forma similar, la segunda función (**min**) toma el vector **nuevo** y calcula el valor mínimo dentro del mismo (0). La función **length** calcula la dimensionalidad del vector, por ejemplo, note que para el vector “x” es 5, mientras que para el vector “nuevo” es 11. La función **mean** calcula la media aritmética del vector numérico, mientras que **sd** ¿Qué será?

10. Suponga que desea ayuda sobre alguna de las funciones que está empleando, escribiendo el comando **help(<nombre>)** se abre el navegador con la ayuda local sobre la función. Por ejemplo, escriba **help(sd)** para saber de qué se trata dicha función.
11. ¿Cómo verificar si **sd** es la desviación estándar? Porque el cuadrado de la desviación es la varianza pero para ello debiéramos utilizar como parte de una expresión lo cual es totalmente posible. De este modo, pruebe:

```

> var(nuevo)
[1] 51.18364
> sd(nuevo)^2
[1] 51.18364

```

Figura 6. Utilización de las funciones como parte de las expresiones

12. La mediana es una medida importante porque nos separa la serie de datos en dos partes iguales, ahora bien, supone un ordenamiento de los valores. Note que sencillez implica en R el ordenamiento y cálculo de la mediana asociada con un vector:

```

> median(nuevo)
[1] 6.4
> sort(nuevo)
[1] 0.0 3.1 3.1 5.6 5.6 6.4 6.4 10.4 10.4 21.7 21.7

```

Figura 7. Ordenamiento y mediana

13. La generación de secuencias es una alternativa útil, (por ejemplo, para asociar a las escalas de los gráficos). En R puede generar una secuencia desde -5 a 5 con saltos de 0.2, o bien si no se indicare el salto, se asumirá que es “1” por defecto. Ejecute los siguientes comandos en su sesión:

```

> secu=seq(-5,5,by=0.5) #Genero secuencia de -5 a 5 con saltos 0.5
> secu
[1] -5.0 -4.5 -4.0 -3.5 -3.0 -2.5 -2.0 -1.5 -1.0 -0.5 0.0 0.5 1.0 1.5 2.0
[16] 2.5 3.0 3.5 4.0 4.5 5.0
> length(secu)
[1] 21
> secu1=seq(-5,5)
> secu1
[1] -5 -4 -3 -2 -1 0 1 2 3 4 5

```

Figura 8. Uso de secuencias y comentarios

Note que los comentarios pueden indicarse en la misma línea que la expresión, pero solo luego de indicar el símbolo “#”. Esto representa una alternativa muy útil a la hora de documentar las operaciones.

14. R permite manipular también vectores lógicos. Los valores lógicos surgen de las operaciones lógicas expuestas en la siguiente tabla:

Operador	Concepto	Ejemplo en R
<	Menor	2 < 5
<=	Menor o igual	2 <= 5
>	Mayor	2 > 5
>=	Mayor o igual	2 >= 5
==	Igual a	2 == 5
!=	Distinto de	2 != 5
&	Y lógica	2<5 & 5>2
	O lógica	2<5 2>5
!	Negación	!(2<5)

```

> 2<5
[1] TRUE
> 2<=5
[1] TRUE
> 2==5
[1] FALSE
> 2!=5
[1] TRUE
> 2>5
[1] FALSE
> 2>=5
[1] FALSE
> 2<5 & 5>2
[1] TRUE
> 2<5 & 2>5
[1] FALSE
> 2<5 | 2>5
[1] TRUE
> !(2<5)
[1] FALSE

```

Figura 9. Salida de la sesión por R

Como anteriormente se disponía de vectores numéricos sobre los que se calculaba la media aritmética, entre otras operaciones, aquí puede también

generar vectores con los valores lógicos resultantes de las operaciones. Pruebe el siguiente ejemplo:

```
> vlogico=c(2<5,2<=5,2==5,2!=5,2>5,2>=5,2<5 & 5>2, 2<5 | 2>5,!(2<5))
> vlogico
[1] TRUE TRUE FALSE TRUE FALSE FALSE TRUE TRUE FALSE
```

15. Los **valores faltantes** son uno de los cuatro problemas fundamentales asociados con el manejo de datos. Cuando un elemento no está disponible, o bien, se trata de un valor faltante, es representado en "R" mediante la constante **NA**. La siguiente instrucción crea un vector con 5 valores y 1 valor faltante, para luego utilizar la función **is.na** para verificar la presencia de valores faltantes dentro de cada elemento del vector:

```
> vfaltante=c(seq(1,10,by=2),NA); is.na(vfaltante)
[1] FALSE FALSE FALSE FALSE FALSE TRUE
> vfaltante
[1] 1 3 5 7 9 NA
> is.na(vfaltante)
[1] FALSE FALSE FALSE FALSE FALSE TRUE
```

16. Ahora bien, ¿Qué ocurre si es necesario acceder a una posición particular de un vector dado? Para ello, referimos **<nombre_vector>[posición]**, pruebe los siguientes comandos:

```
> vfaltante
[1] 1 3 5 7 9 NA
> vfaltante[2]
[1] 3
> vfaltante[2]+vfaltante[3]
[1] 8
> vlogico
[1] TRUE TRUE FALSE TRUE FALSE FALSE TRUE TRUE FALSE
> !vlogico[1]
[1] FALSE
```

Figura 10. Acceso directo al contenido del vector



IMPORTANTE

Tenga en cuenta que la posición inicial del vector es contabilizada a partir de 1 (uno).

17. Note que es posible sumar parcialmente un vector empleando el acceso directo soportado por rangos, ejecute el siguiente comando:

```
> sum(x[1:3])
[1] 19.1
> x
[1] 10.4 5.6 3.1 6.4 21.7
```

Figura 11. Suma parcial de los datos del vector

18. Alternativamente, podríamos indicar que sume la totalidad del vector excluyendo las posiciones 1 a 3, pruebe el siguiente comando:


```
> sum(x[-(1:3)])
[1] 28.1
```

Figura 12. Suma parcial del vector excluyendo posiciones

Note que el 28.1 surge de la suma de 6.4 y 21.7 correspondientes a la posición 4 y 5 (Ver figura 11), que son las posiciones remanentes una vez excluidas las posiciones de 1 a 3.

19. Se sabe que el vector **vfaltante** (Ver paso 15) presenta valores faltantes por lo que al intentar calcular una media aritmética, R nos retornará **NA**. Ahora bien, la siguiente instrucción capitaliza lo visto hasta el momento para calcular la media aritmética sobre un vector excluyendo los valores ausentes como sigue:

```
> vfaltante
[1] 1 3 5 7 9 NA
> mean(vfaltante)
[1] NA
> mean(vfaltante[!is.na(vfaltante)])
[1] 5
```

Figura 13. Calculo de la media en presencia de valores faltantes

Note que el primer comando expone que la posición sexta del vector posee un valor faltante. De este modo, al intentar calcular la media aritmética retorna “NA”. Ahora bien, note que la última expresión obtiene la media aritmética de los valores no nulos, es decir 5 surge de $(1+3+5+7+9)/5$ ¿Cómo es posible?

- La evaluación de las expresiones se dan de derecha a izquierda en las asignaciones y de adentro hacia fuera en las expresiones. De este modo en la tercera instrucción se resuelve en el siguiente orden:
 - Se consulta los valores faltantes del vector **vfaltante**, lo cual retornará las posiciones que contienen valor faltante,
 - Ahora bien, se desea calcular la media sobre los valores no nulos, por lo que El operador “!” se utiliza para que retorne las posiciones no nulas del vector **vfaltante** (**!is.na(vfaltante)**)
 - La expresión **mean(!is.na(vfaltante))** calculará la media sobre el conjunto de valores no nulos del vector **vfaltante**.

20. Ahora bien, suponga que desea valor los nulos del vector por el promedio de la serie de datos con valor ¿Cómo hacerlo?

```
> mean(vfaltante[!is.na(vfaltante)]);
[1] 5
> vfaltante[is.na(vfaltante)]
[1] NA
> vfaltante[is.na(vfaltante)]=mean(vfaltante[!is.na(vfaltante)])
> vfaltante
[1] 1 3 5 7 9 5
> mean(vfaltante)
[1] 5
```

Figura 14. Reemplazando los nulos en un vector

La primera expresión muestra el cálculo de la media aritmética para los valores del vector *vfaltante*. La segunda expresión expone las posiciones sin valor (en este caso solo una). La tercera expresión reemplaza las posiciones sin valor del vector *vfaltante*, por la media aritmética calculada para las posiciones con valor del mismo vector. De hecho, note que la instrucción siguiente muestra nuevamente el vector *vfaltante* pero en esta oportunidad actualizada, aspecto que se confirma al utilizar la función *mean* que calcula la media directamente sin arrojar error.

21. Una utilidad importante de conocer cuando se manejan datos de distintos tipos es como convertir por ejemplo, caracteres a enteros o viceversa.

```
> vfaltante_char=as.character(vfaltante)
> vfaltante_char
[1] "1" "3" "5" "7" "9" "5"
> vfaltante_char=as.character(vfaltante)
> as.integer(vfaltante_char)
[1] 1 3 5 7 9 5
```

Figura 15. Conversión entero - caracter

La función *as.character* permite convertir el vector *vfaltante* entero a caracter, por ello al mostrarse se exponen entre comillas dobles. Si se deseara convertir el vector caracter a entero, se utilizaría la función *as.integer*.

22. ¿Qué ocurre si la conversión es requiere a un tipo con decimales y no entero? En ese caso utilizaremos *as.numeric* en lugar de entero, pudiendo verificar si el argumento es o no numérico (real o entero) mediante *is.numeric*. Pruebe el siguiente ejemplo:

```
> is.numeric(vfaltante_char)
[1] FALSE
> as.numeric(vfaltante_char)
[1] 1 3 5 7 9 5
> is.numeric(as.numeric(vfaltante_char))
[1] TRUE
```

Figura 16. Verificación y conversión a numérico

Note que empleando *is.character(<argumento>)* o *is.integer(<argumento>)* es posible verificar si el argumento es entero o caracter.

23. Es posible **crear un objeto** (por ejemplo, unvector) **vacío y luego ajustar su tamaño** en forma dinámica ¿Cómo dinámico? Se crea el objeto vacío indicándole el tipo, y luego se asigna un valor dado a una posición específica, si esta no existe, el vector extiende su tamaño en forma automática. Escriba los siguientes comandos en su sesión de R:

```

> nro = numeric();
> nro
numeric(0)
> nro[3]=25.3
> nro
[1] NA NA 25.3
> nro[5]=78.7
> nro
[1] NA NA 25.3 NA 78.7
> mean(nro[!is.na(nro)])
[1] 52
> nro[is.na(nro)]=mean(nro[!is.na(nro)])
> nro
[1] 52.0 52.0 25.3 52.0 78.7

```

Figura 17. Ajuste dinámico del tamaño de los vectores

24. En R se gestionan los arreglos bidimensionales como sinónimo de matrices.

```

> miarreglo=array(1:20,dim=c(5,4))
> miarreglo
      [,1] [,2] [,3] [,4]
[1,]    1    6   11   16
[2,]    2    7   12   17
[3,]    3    8   13   18
[4,]    4    9   14   19
[5,]    5   10   15   20
> miarreglo[1,]
[1] 1 6 11 16
> miarreglo[,1]
[1] 1 2 3 4 5
> miarreglo[1,2]
[1] 6

```

Figura 18. Creando arreglos

Note que la función *array* permite crear un arreglo 5x4 (indicado en *dim* - dimensión-) e inicializar los valores con una secuencia del 1 al 20, como puede observarse en la anterior figura. Adicionalmente, si se desea referir a una fila completa, puede escribirse **miarreglo[<fila>,]**, o bien, si se desea referirse a una columna completa puede escribirse **miarreglo[,<columna>]** como expone el ejemplo anterior. Adicionalmente, para acceder al contenido de una posición particular indicamos fila y columna conjuntamente como se expone para la fila 1 columna 2.

25. Podría preguntarse si siempre que se crea un arreglo deberá indicar valores, la respuesta es no. La inicialización de la matriz es opcional, lo que debiera indicar es la dimensionalidad de la misma.

```

> miarreglo2=array(dim=c(3,3))
> miarreglo2
      [,1] [,2] [,3]
[1,]    NA    NA    NA
[2,]    NA    NA    NA
[3,]    NA    NA    NA
> miarreglo2[3,3]=15.75
> miarreglo2
      [,1] [,2] [,3]
[1,]    NA    NA    NA
[2,]    NA    NA    NA
[3,]    NA    NA 15.75
> is.na(miarreglo2[,1])
[1] TRUE TRUE TRUE
> is.na(miarreglo2[,3])
[1] TRUE TRUE FALSE

```

Figura 19. Verificando los nulos de una matriz

La primera instrucción de la figura 19 crea una matriz 3x3, la cual se completa con nulos al no indicar inicialización. Note que del mismo modo que se leía de una posición dada, puede escribirse en la misma posición. De este modo, se pueden emplear las funciones vistas hasta el momento, por ejemplo, para conocer si en una fila o columna dada existen nulos.

26. La siguiente instrucción expone cómo es posible obtener una sub matriz a partir de otra.

```

> miarreglo2[2:3,2:3]
      [,1] [,2]
[1,]    NA    NA
[2,]    NA 15.75
> miarreglo2
      [,1] [,2] [,3]
[1,]    NA    NA    NA
[2,]    NA    NA    NA
[3,]    NA    NA 15.75

```

Figura 20. Obteniendo submatrices

Note que para referir a una sub matriz, se indican los rangos de la sección asociados con la matriz principal de la cual se desean recuperar los valores.

27. Note que el punto 24 utiliza la función array para generar una matriz a partir de la secuencia 1:20, por lo que en carácter transitivo puede deducirse que mediante la misma función se podría generarse una matriz a partir de un vector. Ejecute el las siguientes líneas de código:

```

> mivec=c(10.4,3.4,7,9.8,10,11)
> mivec
[1] 10.4  3.4  7.0  9.8 10.0 11.0
> mivec_arr=array(mivec,dim=c(3,2))
> mivec_arr
      [,1] [,2]
[1,] 10.4  9.8
[2,]  3.4 10.0
[3,]  7.0 11.0

```

Figura 21. Convirtiendo un vector en matriz

Note que la primera instrucción genera un vector numérico que es mostrado por la segunda instrucción. Luego, se emplea la función *array* empleando como entrada el vector (mivec) y definiendo como dimensión 3x2.

28. Ahora bien ¿Qué sucedería si en lugar de indicar dimensión 3x2 se indicare 3x3? Pruébalo y verifíquelo con los siguientes comandos:

```
> mivec_arr2=array(mivec,dim=c(3,3))
> mivec_arr2
      [,1] [,2] [,3]
[1,] 10.4  9.8 10.4
[2,]  3.4 10.0  3.4
[3,]  7.0 11.0  7.0
```

Figura 22. Utilizando el vector para completar la matriz

De este modo, puede apreciarse que los valores del vector respecto a su orden original, es empleado para completar la matriz guiándose por sus columnas (primero la columna 1, luego la 2, y así sucesivamente). En este caso, al culminarse los valores del vector para la matriz, se reinicia el contador del vector a su primera posición y continuará completando la matriz desde el primer valor del vector y del mismo modo en forma reiterativa.

29. El siguiente código permite inicializar la matriz con un valor dado, muy útil en cálculos matriciales por ejemplo.

```
> unos=array(1,dim=c(3,3))
> unos
      [,1] [,2] [,3]
[1,]    1    1    1
[2,]    1    1    1
[3,]    1    1    1
```

Figura 23. Inicializando la matriz con un valor dado

30. Si desea un producto matricial, el operador a utilizar es `%*%`:

```
> m31=array(1,dim=c(3,1))
> m31
      [,1]
[1,]    1
[2,]    1
[3,]    1
> mivec_arr2
      [,1] [,2] [,3]
[1,] 10.4  9.8 10.4
[2,]  3.4 10.0  3.4
[3,]  7.0 11.0  7.0
> mivec_arr2 %*% m31
      [,1]
[1,] 30.6
[2,] 16.8
[3,] 25.0
```

Figura 24. Producto matricial

Note que la primera instrucción crea una matriz 3x1 con unos dentro. La idea es realizar el producto matricial respecto de la matriz mivec_arr2 (Ver punto 28) la cual es 3x3, lo que arrojaría finalmente una matriz resultante 3x1 ($3 \times 3 * 3 \times 1$).



Importante

Si en lugar de emplear `mivec_Arr2 %*% m31` utilizare `mivec_Arr2 * m31`, el software retornaría un error similar a “Error in mivec_arr2 * m31 : non-conformable arrays” ¿Por qué? Porque el operador “*” intenta realizar un producto elemento a elemento, y dado que las matrices contienen dimensiones diferentes $(3 \times 3) * (3 \times 1)$ no es posible obtener una matriz resultante 3×3 por lo que faltan elementos. Por ese motivo se señala que los arreglos no son adecuados entre sí.

31. El producto externo se calcula mediante `%o%`, o bien, `outer(vector1,vector2)`. Analicemos su comportamiento con el siguiente ejemplo:

```
> x=1:4
> x %o% x
      [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    2    4    6    8
[3,]    3    6    9   12
[4,]    4    8   12   16
> y=2:6
> x %o% y
      [,1] [,2] [,3] [,4] [,5]
[1,]    2    3    4    5    6
[2,]    4    6    8   10   12
[3,]    6    9   12   15   18
[4,]    8   12   16   20   24
> outer(x,y)
      [,1] [,2] [,3] [,4] [,5]
[1,]    2    3    4    5    6
[2,]    4    6    8   10   12
[3,]    6    9   12   15   18
[4,]    8   12   16   20   24
```

Figura 25. Producto externo entre arreglos

Note que el vector “x” se compone de los siguientes 4 elementos 1, 2, 3 y 4; mientras que el vector “y” se compone de los siguientes 5 elementos 2, 3, 4, 5 y 6. De este modo, el producto externo de “`x %o% y`” (o bien, `outer(x,y)`) arroja como resultado una matriz 4x5 (la llamaremos “A”), en donde cada elemento se genera como:

$$A(\text{fila}, \text{columna}) = x(\text{índice_fila}) * y(\text{índice_columna})$$

Es decir:

$$A(2,3) = x(2) * y(3) = 2 * 4 = 8$$

$$A(4,5) = x(4) * y(5) = 4 * 6 = 24$$

32. Crear una matriz indicando el valor de la diagonal principal (por ejemplo, para crear una matriz identidad) es un aspecto importante, al igual que poder recuperar los elementos de la diagonal principal de una matriz. Este aspecto es lo que abordan los siguientes comandos:

```
> diag(1,3,3)
      [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
> diag(2,3,3)
      [,1] [,2] [,3]
[1,]    2    0    0
[2,]    0    2    0
[3,]    0    0    2
> mivec_arr2
      [,1] [,2] [,3]
[1,] 10.4  9.8 10.4
[2,]  3.4 10.0  3.4
[3,]  7.0 11.0  7.0
> diag(mivec_arr2)
[1] 10.4 10.0  7.0
```

Figura 26. Diagonal Principal y Matrices

La primera instrucción tiene la forma **diag(elemento, filas, columnas)** y permite crear una matriz indicando en la diagonal principal el valor de “elemento”. Note que la segunda instrucción, en lugar de crear una matriz identidad, crea una matriz con “2” en su diagonal. Ahora bien, la misma instrucción puede utilizarse para recuperar los elementos de la diagonal principal en una matriz creada, y a ello hace referencia la última instrucción. Más aún, si usted escribe **diag(<escalar>)** por ejemplo: **diag(5)** dará como resultado una matriz identidad 5x5 .

Operación Matricial	Concepto	Ejemplo en R
crossprod(A,B)	Producto cruzado entre matrices “A” y “B”	crossprod(mivec_arr2,mivec_arr2)
t(a)	Traspuesta de una matriz “A”	t(mivec_arr2)
solve(A)	Obtiene la inversa de “A”	solve(5*diag(5))
cbind(A,B, ...)	Combina matrices incorporando las columnas de “B” a continuación de las de “A” y así sucesivamente (Deben contener igual número de filas).	cbind(mivec_arr2,diag(3))
rbind(A, B, ...)	Combina matrices incorporando las filas de “B” a continuación de las de “A” y así sucesivamente (Deben contener igual número de columnas).	rbind(mivec_arr2,diag(3))

rowMeans(A)	Calcula la media aritmética por cada fila, para los valores de las distintas columnas de la matriz	<code>rowMeans(mivec_arr2)</code>
rowSums(A)	Calcula la suma por fila, para los valores en las distintas columnas de la matriz.	<code>rowSums(mivec_arr2)</code>
colMeans(A)	Calcula la media aritmética por cada columna, para los valores de las distintas filas de la matriz.	<code>colMeans(mivec_arr2)</code>
colSums(A)	Calcula la suma por cada columna, para los valores de las distintas filas de la matriz.	<code>colSums(mivec_arr2)</code>

```

> mivec_arr2
      [,1] [,2] [,3]
[1,] 10.4  9.8 10.4
[2,]  3.4 10.0  3.4
[3,]  7.0 11.0  7.0
> rowMeans(mivec_arr2)
[1] 10.200000  5.600000  8.333333
> rowSums(mivec_arr2)
[1] 30.6 16.8 25.0
> colMeans(mivec_arr2)
[1]  6.933333 10.266667  6.933333
> colSums(mivec_arr2)
[1] 20.8 30.8 20.8

```

Figura 27. Operaciones directas sobre filas y columnas de la matriz



Recomendación

Finalizado el presente tutorial, se recomienda avanzar sobre el tutorial R. 02 - Lectura de datos