

Hoja 1 de problemas y prácticas con R

Estadística Computacional I. Grado en Estadística

Departamento de Estadística e Investigación Operativa. Universidad de Sevilla

Contents

1	Ejercicio 1	2
1.1	Solución:	2
2	Ejercicio 2	2
2.1	Solución:	2
3	Ejercicio 3	3
3.0.1	Visualizarlos en dos columnas.	3
3.0.2	Construir el vector	4
3.0.3	Generar el vector	5
3.0.4	Construir el vector	5
3.0.5	Calcular la suma de los valores	6
4	Ejercicio 4	6
4.1	Solución:	6
4.1.1	Determinar las posiciones y valores de los $y_i > 600$	6
4.1.2	Construir una matriz con las posiciones y valores anteriores, y con los valores de x en esas posiciones.	7
4.1.3	Guardar las posiciones como nombres de filas de la matriz anterior.	7
4.1.4	Construir el vector	8
4.1.5	Calcular el número de elementos de y que distan menos de 200 del máximo de y.	9
4.1.6	¿Cuántos elementos de x son pares?	9
4.1.7	Seleccionar los elementos de x en posiciones donde y sea múltiplo de 5.	9
4.1.8	Ordenar los elementos de x según la ordenación creciente de y.	10
5	Ejercicio 5	10
5.1	Solución:	10
6	Ejercicio 6	10
6.1	Solución:	10
7	Ejercicio 7	11
7.1	Solución:	11
8	Ejercicio 8	11
8.1	Solución:	11
9	Ejercicio 9	13
9.1	Solución:	13
9.1.1	Calcular la suma de cada fila, visualizarlo en una nueva columna.	13
9.1.2	Calcular el máximo de cada columna, visualizarlo en una fila nueva.	13

1 Ejercicio 1

1. Crear un vector donde se repitan los códigos provinciales de Andalucía: 10 veces Almería, 10 veces Cádiz, ..., 10 veces Jaén, 15 para Málaga y 18 Sevilla. Permutar aleatoriamente los elementos de dicho vector y calcular la tabla de frecuencias.

1.1 Solución:

```

codigos=c(4,11,14,18,21,23,29,41)
#acceder a elementos:
codigosrepetidos=rep(codigos,c(rep(10,6),15,18))

set.seed(1)
codrepetidospermutados=sample(codigosrepetidos)
codrepetidospermutados

## [1] 29 18 4 18 41 21 11 41 23 23 14 23 29 4 29 18 41 41 21 18 18 29 21 18 11
## [26] 14 29 29 18 29 14 23 23 4 23 41 41 29 23 14 18 41 4 21 11 14 23 29 41 29
## [51] 29 14 41 29 41 11 11 18 11 29 41 29 21 29 41 41 21 11 11 21 41 41 21 41 4
## [76] 14 41 21 18 4 21 14 4 23 41 14 4 11 14 11 4 23 4

table(codrepetidospermutados)

## codrepetidospermutados
## 4 11 14 18 21 23 29 41
## 10 10 10 10 10 10 15 18

Provincias =c("Almería","Cádiz","Jaén","Granada","Córdoba","Huelva","Málaga","Sevilla")
#table(factor(codrepetidospermutados),labels=Provincias)

```

Otra forma, con el paquete tidyverse.

```

set.seed(1)
codigosrepetidos %>%
  sample() %>%
  table()

## .
## 4 11 14 18 21 23 29 41
## 10 10 10 10 10 10 15 18

```

2 Ejercicio 2

2. Con la ayuda de paste, crear un vector de nombres "Caso_1",..., "Caso_30".

2.1 Solución:

```

paste("Caso_",1:30, sep = "")

## [1] "Caso_1" "Caso_2" "Caso_3" "Caso_4" "Caso_5" "Caso_6" "Caso_7"
## [8] "Caso_8" "Caso_9" "Caso_10" "Caso_11" "Caso_12" "Caso_13" "Caso_14"
## [15] "Caso_15" "Caso_16" "Caso_17" "Caso_18" "Caso_19" "Caso_20" "Caso_21"
## [22] "Caso_22" "Caso_23" "Caso_24" "Caso_25" "Caso_26" "Caso_27" "Caso_28"

```

```
## [29] "Caso_29" "Caso_30"
```

```
# Formas equivalentes:
```

```
# paste0("Caso_",1:30)
```

```
# paste("Caso",1:30, sep = "_")
```

Con un bucle for, pero se recomienda no usarlo por ser mucho más lento en general.

```
ve2=c()
for (i in 1:30) {
  ve2[i]= paste("Caso_",i)
}
ve2
```

```
## [1] "Caso_ 1" "Caso_ 2" "Caso_ 3" "Caso_ 4" "Caso_ 5" "Caso_ 6"
## [7] "Caso_ 7" "Caso_ 8" "Caso_ 9" "Caso_ 10" "Caso_ 11" "Caso_ 12"
## [13] "Caso_ 13" "Caso_ 14" "Caso_ 15" "Caso_ 16" "Caso_ 17" "Caso_ 18"
## [19] "Caso_ 19" "Caso_ 20" "Caso_ 21" "Caso_ 22" "Caso_ 23" "Caso_ 24"
## [25] "Caso_ 25" "Caso_ 26" "Caso_ 27" "Caso_ 28" "Caso_ 29" "Caso_ 30"
```

3 Ejercicio 3

3. Generar dos vectores de tamaño 250, seleccionando aleatoriamente números enteros entre 0 y 999, sean x e y los vectores resultantes.

3.0.1 Visualizarlos en dos columnas.

```
set.seed(1357) #semilla del generador
n=250
x=sample(0:999, size = n, replace = T) #Extracción con reemplazamiento.
y=sample(0:999, size = n, replace = T)
xy=cbind(x,y) # Es una matriz
head(xy,10)
```

```
##           x    y
## [1,] 139 894
## [2,] 776 235
## [3,] 537 316
## [4,] 262 963
## [5,] 451  69
## [6,]  89  99
## [7,] 272  70
## [8,] 644 731
## [9,]  81 848
## [10,] 955 220
```

Otro modo, con la librería tidyverse:

```
set.seed(1357) #semilla del generador
n=250
xyt= tibble(
  x= sample(0:999,n, replace = T),
  y= sample(0:999,n, replace = T))
head(xyt,10) # Creo un objeto tibble
```

```
## # A tibble: 10 x 2
##       x     y
##   <int> <int>
## 1   139   894
## 2   776   235
## 3   537   316
## 4   262   963
## 5   451    69
## 6    89    99
## 7   272    70
## 8   644   731
## 9    81   848
## 10  955   220
```

3.0.2 Construir el vector

$y_2 - x_1, \dots, y_{250} - x_{249}$.

Vemos que la y va adelantada un lugar.

```
v2=y[2:n]-x[1:(n-1)]
v2
```

```
## [1] 96 -460 426 -193 -352 -19 459 204 139 -91 735 218 367 795 286
## [16] -137 205 -95 -73 -8 -316 37 -175 -90 -344 0 -160 209 -17 721
## [31] 286 -648 441 -677 492 97 -457 -62 13 402 124 -45 -591 436 288
## [46] 83 -814 688 337 -331 -187 222 222 -590 -165 479 -626 19 -781 -746
## [61] -126 245 359 -253 -76 -614 24 607 -172 -262 385 -504 -556 462 188
## [76] 667 -229 124 -594 -240 -580 -611 -171 -461 -884 365 531 546 196 -303
## [91] 244 -265 633 -400 -828 -548 31 859 -370 48 -446 -65 728 274 14
## [106] -746 54 502 664 -217 164 -286 -15 -87 287 248 -628 550 147 -246
## [121] -36 -361 -510 -84 -293 212 -345 -547 -632 232 127 236 236 -904 81
## [136] 168 92 -200 -349 -215 56 257 756 -278 334 392 -321 -586 -578 -805
## [151] -164 -81 737 269 119 133 44 -221 -679 -229 308 200 653 24 179
## [166] -90 100 517 39 519 -273 -233 712 -40 -255 77 341 -4 226 -13
## [181] -215 -151 -103 26 -788 -401 65 610 46 121 -463 549 -450 -721 -463
## [196] -139 -602 -467 -3 55 76 399 443 -588 -444 608 -101 253 588 -314
## [211] -595 -247 89 -534 466 289 642 -455 -537 71 335 -579 -351 -276 -564
## [226] 777 106 176 -331 -218 57 398 60 -59 125 -212 -500 -322 -99 7
## [241] -710 2 601 -175 -368 -587 -858 541 -42
```

```
# Otra forma
# y[-1]-x[-n]
```

Otra forma:

```
head(cbind(y[-1],x[-n],y[-1]-x[-n]),10)
```

```
##      [,1] [,2] [,3]
## [1,] 235 139 96
## [2,] 316 776 -460
## [3,] 963 537 426
## [4,] 69 262 -193
## [5,] 99 451 -352
## [6,] 70 89 -19
## [7,] 731 272 459
## [8,] 848 644 204
```

```
## [9,] 220 81 139
## [10,] 864 955 -91
```

Con el sistema tidyverse:

```
xyt %>%
  mutate(
    t1=lead(y,n=1),
    t2=x ,
    v2=t1-t2
  ) %>%
pull(v2) %>% # Me quedo con la columna v2 en formato vector
head(10)
```

```
## [1] 96 -460 426 -193 -352 -19 459 204 139 -91
```

3.0.3 Generar el vector

$y_2 - y_1, \dots, y_{250} - y_{249}$.

```
v3=y[-1]-y[-n]
head(v3,10)
```

```
## [1] -659 81 647 -894 30 -29 661 117 -628 644
```

Otra forma

```
head(cbind(y[-1],y[-n],y[-1]-y[-n]),10)
```

```
##      [,1] [,2] [,3]
## [1,] 235 894 -659
## [2,] 316 235 81
## [3,] 963 316 647
## [4,] 69 963 -894
## [5,] 99 69 30
## [6,] 70 99 -29
## [7,] 731 70 661
## [8,] 848 731 117
## [9,] 220 848 -628
## [10,] 864 220 644
```

```
xyt %>%
  mutate(
    t1=lead(y,n=1),
    t2=y ,
    v2=t1-t2
  ) %>%
pull(v2) %>% # Me quedo con la columna v2 en formato vector
head(10)
```

```
## [1] -659 81 647 -894 30 -29 661 117 -628 644
```

3.0.4 Construir el vector

$x_1 + 2x_2 - x_3, x_2 + 2x_3 - x_4, \dots, x_{248} + 2x_{249} - x_{250}$.

```
v4=x[1:248]+2*x[2:249]-x[3:250]
v4
```

```
## [1] 1154 1588 610 1075 357 -11 1479 -149 1774 955 795 899 478 -279 423
```

```
## [16] 1522 316 219 1748 633 1478 1751 1864 2046 558 282 267 1578 483 150
## [31] 1865 -223 1574 1734 -243 1844 1191 552 -272 1150 417 998 1061 508 732
## [46] 2408 949 625 1456 713 580 -316 1826 1148 455 1940 95 944 2187 764
## [61] 551 27 2052 534 554 2229 245 1006 1016 856 970 1504 1092 518 -431
## [76] 601 1343 1528 1385 1429 1012 2086 1041 2072 1441 959 369 110 2398 605
## [91] 671 193 1119 2001 1754 1005 -384 1213 1344 626 1976 579 539 -171 1112
## [106] 2540 1151 246 445 1273 903 538 965 303 -273 1701 1308 77 438 -321
## [121] 1060 1212 1753 1958 1487 920 1165 2177 1357 1242 107 211 1939 1625 918
## [136] 829 1236 2080 883 716 488 -97 1188 967 -448 1404 1396 1183 1566 2137
## [151] 1184 512 192 211 188 1925 562 1358 1092 1111 779 -81 2026 1037 122
## [166] 1276 245 1186 -178 1582 1344 14 1572 1213 767 -153 2123 1053 -3 150
## [181] 2107 998 887 1798 1197 1892 1360 706 -63 1392 536 1005 1686 1851 609
## [196] 823 1096 2026 1632 -112 605 697 977 1688 939 318 264 -86 116 1597
## [211] 1416 540 1471 658 919 -545 1161 2007 958 -136 807 1489 926 2303 710
## [226] 500 -88 1573 1100 152 704 678 714 1508 1458 834 2249 1407 -221 1065
## [241] 2171 509 59 754 1425 2552 49 1590
```

Otra forma:

```
xyt %>%
  mutate(
    t1=x,
    t2=lead(x,n=1), # Empieza por el 2º
    t3=lead(x,n=2), # Empieza por el 3º
    v3= t1+2*t2-t3
  ) %>%
  pull(v3) %>%
  head(10)
```

```
## [1] 1154 1588 610 1075 357 -11 1479 -149 1774 955
```

3.0.5 Calcular la suma de los valores

$$\frac{1}{(x_i + y_i)}$$

```
sum(1/(x+y))
```

```
## [1] 0.3663756
```

4 Ejercicio 4

4.Continuando con los vectores x e y anteriores:

4.1 Solución:

4.1.1 Determinar las posiciones y valores de los $y_i > 600$.

```
cbind(which(y>600), y[which(y>600)]) %>%
  head(10)
```

```
##      [,1] [,2]
## [1,]    1 894
## [2,]    4 963
## [3,]    8 731
## [4,]    9 848
## [5,]   11 864
```

```
## [6,] 12 952
## [7,] 13 652
## [8,] 14 657
## [9,] 15 910
## [10,] 18 780
```

Sistema tidyverse:

```
xyt %>%
  mutate(positcion = row_number()) %>%
  filter(y>600) %>%
  select(positcion , y) %>%
  head(10)
```

```
## # A tibble: 10 x 2
##   positcion     y
##   <int> <int>
## 1      1     894
## 2      4     963
## 3      8     731
## 4      9     848
## 5     11     864
## 6     12     952
## 7     13     652
## 8     14     657
## 9     15     910
## 10    18     780
```

4.1.2 Construir una matriz con las posiciones y valores anteriores, y con los valores de x en esas posiciones.

```
xyt %>%
  mutate(positcion = row_number()) %>%
  filter(y>600) %>%
  select(positcion , x) %>%
  as.matrix() %>%
  head(10)
```

```
##   positcion  x
## [1,]      1 139
## [2,]      4 262
## [3,]      8 644
## [4,]      9  81
## [5,]     11 217
## [6,]     12 434
## [7,]     13 290
## [8,]     14 115
## [9,]     15  42
## [10,]    18 106
```

4.1.3 Guardar las posiciones como nombres de filas de la matriz anterior.

```
xyt %>%
  mutate(positcion = row_number()) %>%
  filter(y>600) %>%
```

```

select(posicion ,y, x) %>%
as.matrix()->xytmat

rownames(xytmat)= xytmat[,1]
xytmat=xytmat[,-1]
xytmat %>% head()

```

```

##      y    x
## 1  894 139
## 4  963 262
## 8  731 644
## 9  848  81
## 11 864 217
## 12 952 434

```

4.1.4 Construir el vector

$|x_1 - \bar{x}|^{1/2}, \dots, |x_n - \bar{x}|^{1/2}$.

```
(xmedia = mean(x))
```

```
## [1] 477.988
```

```
abs(x-xmedia)^(1/2)
```

```

## [1] 18.4116268 17.2630241 7.6819268 14.6965302 5.1949976 19.7227787
## [7] 14.3522820 12.8845644 19.9245577 21.8406044 16.1551230 6.6323450
## [13] 13.7108716 19.0522440 20.8803257 0.1095445 9.8494670 19.2869904
## [19] 2.6434826 18.7353142 9.8482486 21.9091762 18.4665102 19.1575573
## [25] 12.9233123 19.7227787 14.5941084 11.8738368 13.9288190 19.3646069
## [31] 9.1097750 16.2791892 21.4939061 22.8037716 6.3236066 18.3844500
## [37] 21.9775340 16.1241434 16.4920587 19.9997000 12.4904764 16.7924983
## [43] 11.4460474 7.8732458 9.8988888 13.7844840 22.4947105 15.8110088
## [49] 3.6072150 10.3446605 16.5223485 13.9995714 16.9996471 22.3161825
## [55] 12.7666754 4.4707941 17.2630241 20.1987128 18.5205831 17.0297387
## [61] 17.5495869 11.5753186 13.0762380 21.2840786 18.9996842 12.3698019
## [67] 18.6282581 20.6394767 14.3531181 7.9992500 4.2440547 8.4859885
## [73] 12.1660182 13.4159606 18.6544365 20.9281628 12.7676153 15.5888422
## [79] 16.1867847 13.9646697 14.9335863 12.9619443 22.4279290 6.6341541
## [85] 22.4947105 7.7451921 9.9492713 16.1551230 5.8299228 22.7378979
## [91] 21.0235106 3.9984997 13.7472906 19.2096851 19.6471881 9.7985713
## [97] 14.8319924 19.8239249 18.2759952 4.2440547 4.2412262 17.6638614
## [103] 20.3466951 11.7893172 16.5827621 20.9287362 21.0953075 15.9996250
## [109] 16.1860434 8.3658831 10.4408812 12.9995385 13.2660469 10.1483004
## [115] 19.7734165 15.2311523 19.3393899 15.1323495 20.8803257 14.8992617
## [121] 19.0259822 18.1937352 14.0004286 21.6104604 18.2486164 11.4460474
## [127] 8.0007500 17.1759134 21.0953075 5.9989999 5.2903686 19.4419135
## [133] 8.0630019 22.2937659 8.7184861 4.4707941 8.6030227 15.9690952
## [139] 17.4359399 16.1551230 12.0410963 17.6348519 17.2912695 12.0005000
## [145] 15.5880724 18.7879749 21.3310103 10.4408812 15.2647306 18.6550797
## [151] 17.8609070 13.9638104 17.2912695 18.6812205 15.2639444 8.3658831
## [157] 19.8749088 15.7793536 17.0590738 8.3059015 4.1245606 13.7836135
## [163] 13.6377417 21.7947700 17.4925127 14.7644167 9.5922886 18.8145688
## [169] 9.7474099 19.8491310 21.0002857 11.7468294 14.9327827 18.9212050
## [175] 11.0899955 12.0410963 14.9662286 22.7158975 18.9469787 17.2912695
## [181] 1.4184499 22.6055745 11.2688952 14.6632875 19.2876126 10.8172085

```



```
## [187] 19.1052872  9.4333451 14.7305126 16.5223485 16.0003750 14.0352414
## [193] 16.7932129 17.8328910 13.7117468 14.1770237 11.5330828 14.0716737
## [199] 19.7233871  9.6947408 21.7942194  4.9002041  8.7171096 11.4460474
## [205] 12.8456997 16.4617132 18.9733497 18.7879749 19.3387694  7.6803646
## [211] 18.6550797  1.9969977 10.9539034 13.1153345 17.0583704 10.5824383
## [217] 21.8629367 20.8089404 13.5281928 15.8741299 17.9718669 13.9288190
## [223] 14.6291490  9.4346171 20.5429307 20.3466951 12.6486363 16.6729721
## [229] 18.1111016 15.4592367 17.1460783  4.7945803  9.3801919  8.8888694
## [235] 17.6638614 12.2886940 10.5835722 22.2937659 13.6743556 18.1104390
## [241] 18.2759952 15.1990789 20.4691964 12.6486363 12.5702824 18.9212050
## [247] 20.1249099 20.6878708 21.3544375 12.2469588
```

4.1.5 Calcular el número de elementos de y que distan menos de 200 del máximo de y.

```
length(which(abs(y-max(y))<200))
```

```
## [1] 45
```

```
xyt %>%
  mutate(
    distancia=abs(y-max(y))
  ) %>%
  filter(distancia<200) %>%
  count()
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1     45
```

```
# summarise(
#   cuantos = n() )
```

4.1.6 ¿Cuántos elementos de x son pares?

```
length(which((x %% 2)==0))
```

```
## [1] 129
```

```
xyt %>%
  filter((x %% 2)==0) %>%
  count()
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1    129
```

```
# La expresión `x %% 2` es módulo. Calcula cual es el resto de la división.
```

4.1.7 Seleccionar los elementos de x en posiciones donde y sea múltiplo de 5.

```
xyt %>%
  mutate( Resto=(y %% 5)==0 ) %>%
  filter(Resto==TRUE) %>%
  pull(x)
```

```
## [1] 776 272 955 42 106 958 819 645 337 103 16 140 634 228 458 70 768 298 130
## [20] 740 981 522 418 217 995 462 64 923 408 116 809 458 622 409 495 953 570 124
## [39] 333 254 994 119 179 989 760 796 666 609 118 826 0 672 239 557 629 975 318
## [58] 50
```

```
# Otra forma:
# xyt %>% filter ((y%%5) ==0) %>% pull(x)
```

Otra forma

```
x[(y%%5)==0]
```

```
## [1] 776 272 955 42 106 958 819 645 337 103 16 140 634 228 458 70 768 298 130
## [20] 740 981 522 418 217 995 462 64 923 408 116 809 458 622 409 495 953 570 124
## [39] 333 254 994 119 179 989 760 796 666 609 118 826 0 672 239 557 629 975 318
## [58] 50
```

4.1.8 Ordenar los elementos de x según la ordenación creciente de y.

```
xyt %>%
  arrange(y) %>%
  pull(x) %>%
  head(10)
```

```
## [1] 711 333 769 676 116 471 790 825 125 416
```

5 Ejercicio 5

5. Calcular $1 + (1 + 2) + \dots + (1 + 2 + 3 + \dots + 10)$.

5.1 Solución:

```
# cumsum(1:10) Cada uno de los sumandos del ejercicio
cumsum(1:10) %>%
  sum()
```

```
## [1] 220
```

```
# Otra forma:
# sum(cumsum(1:10))
```

6 Ejercicio 6

6. Calcular:

$1 + (2/3) + (2/3)(4/5) + (2/3)(4/5)(6/7) + \dots + ((2/3)(4/5)(6/7)\dots(38/39))$.

6.1 Solución:

```
v6=c(1,seq(2,38,by=2)/seq(3,39, by=2))
v6
```

```
## [1] 1.0000000 0.6666667 0.8000000 0.8571429 0.8888889 0.9090909 0.9230769
## [8] 0.9333333 0.9411765 0.9473684 0.9523810 0.9565217 0.9600000 0.9629630
## [15] 0.9655172 0.9677419 0.9696970 0.9714286 0.9729730 0.9743590
```

```
cumprod(v6) %>%
  sum()
```

```
## [1] 6.976346
```

7 Ejercicio 7

7. Construir una matriz $n \times n$ con 0 en la diagonal, +1 en la mitad triangular superior y -1 en la mitad triangular inferior.

7.1 Solución:

```
m1<-diag(0,10,10)
m1[lower.tri(m1)] <- -1
m1[upper.tri(m1)] <- 1
m1
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    0    1    1    1    1    1    1    1    1    1
## [2,]   -1    0    1    1    1    1    1    1    1    1
## [3,]   -1   -1    0    1    1    1    1    1    1    1
## [4,]   -1   -1   -1    0    1    1    1    1    1    1
## [5,]   -1   -1   -1   -1    0    1    1    1    1    1
## [6,]   -1   -1   -1   -1   -1    0    1    1    1    1
## [7,]   -1   -1   -1   -1   -1   -1    0    1    1    1
## [8,]   -1   -1   -1   -1   -1   -1   -1    0    1    1
## [9,]   -1   -1   -1   -1   -1   -1   -1   -1    0    1
## [10,]  -1   -1   -1   -1   -1   -1   -1   -1   -1    0
```

8 Ejercicio 8

8. Construir una matriz con la tabla de multiplicar.

8.1 Solución:

```
numeros=c(1:10)
m=matrix(1:10,nrow=10,ncol=10)
m[1,]=numeros

diag(m)=diag(m)^2

m[-c(1,2),2]=2*m[-c(1,2),2]
m[-c(1,3),3]=3*m[-c(1,3),3]
m[-c(1,4),4]=4*m[-c(1,4),4]
m[-c(1,5),5]=5*m[-c(1,5),5]
m[-c(1,6),6]=6*m[-c(1,6),6]
m[-c(1,7),7]=7*m[-c(1,7),7]
m[-c(1,8),8]=8*m[-c(1,8),8]
m[-c(1,9),9]=9*m[-c(1,9),9]
m[-c(1,10),10]=10*m[-c(1,10),10]
m
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
```

```
## [1,] 1 2 3 4 5 6 7 8 9 10
## [2,] 2 4 6 8 10 12 14 16 18 20
## [3,] 3 6 9 12 15 18 21 24 27 30
## [4,] 4 8 12 16 20 24 28 32 36 40
## [5,] 5 10 15 20 25 30 35 40 45 50
## [6,] 6 12 18 24 30 36 42 48 54 60
## [7,] 7 14 21 28 35 42 49 56 63 70
## [8,] 8 16 24 32 40 48 56 64 72 80
## [9,] 9 18 27 36 45 54 63 72 81 90
## [10,] 10 20 30 40 50 60 70 80 90 100
```

Otro caso:

```
matriz8 = function(n,k){
  m = matrix(rep(1,n*k),nrow = n,ncol = k)
  for (i in 1:n) {
    for (j in 1:k) {
      m[i,j] = i*j
    }
  }
  return(m)
}
matriz8(10,10)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,] 1 2 3 4 5 6 7 8 9 10
## [2,] 2 4 6 8 10 12 14 16 18 20
## [3,] 3 6 9 12 15 18 21 24 27 30
## [4,] 4 8 12 16 20 24 28 32 36 40
## [5,] 5 10 15 20 25 30 35 40 45 50
## [6,] 6 12 18 24 30 36 42 48 54 60
## [7,] 7 14 21 28 35 42 49 56 63 70
## [8,] 8 16 24 32 40 48 56 64 72 80
## [9,] 9 18 27 36 45 54 63 72 81 90
## [10,] 10 20 30 40 50 60 70 80 90 100
```

Otra forma:

```
i=1:9
j=1:9

A=outer(i,j,"*")
colnames(A)=paste("i",1:9,sep="")
rownames(A)=1:9
A
```

```
##      *1 *2 *3 *4 *5 *6 *7 *8 *9
## 1  1  2  3  4  5  6  7  8  9
## 2  2  4  6  8 10 12 14 16 18
## 3  3  6  9 12 15 18 21 24 27
## 4  4  8 12 16 20 24 28 32 36
## 5  5 10 15 20 25 30 35 40 45
## 6  6 12 18 24 30 36 42 48 54
## 7  7 14 21 28 35 42 49 56 63
## 8  8 16 24 32 40 48 56 64 72
## 9  9 18 27 36 45 54 63 72 81
```

Con el sistema tidyverse:

```
tabla1=tibble(f1=1:10) %>%
  mutate(
    tabla1= f1*1,
    tabla2=f1*2,
    tabla3=f1*3,
    tabla4=f1*4,
    tabla5=f1*5,
    tabla6=f1*6,
    tabla7=f1*7,
    tabla8=f1*8,
    tabla9=f1*9,
    tabla10=f1*10
  )
```

9 Ejercicio 9

9. Construir una matriz 6x9 con enteros aleatorios en 1, ..., 10.

9.1 Solución:

```
set.seed(12345)
matriz=matrix(data = sample(1:10,size = 54,replace = T),nrow = 6,ncol = 9)
matriz
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
## [1,]    3    6    7    3    4    9   10   10    3
## [2,]   10    6    6    9    9    5   10    8    6
## [3,]    8    7    1    4    9    3    3    9    3
## [4,]   10   10    4   10    4    1    3    8    7
## [5,]    8    1    8    7    8    1    3    4   10
## [6,]    2    8   10    2    6    5    1    2    7
```

9.1.1 Calcular la suma de cada fila, visualizarlo en una nueva columna.

```
colSums(matriz)

## [1] 41 38 36 35 40 24 30 41 36

m2=cbind(matriz,sumas=rowSums(matriz))
m2
```

```
##
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] sumas
## [1,]    3    6    7    3    4    9   10   10    3    55
## [2,]   10    6    6    9    9    5   10    8    6    69
## [3,]    8    7    1    4    9    3    3    9    3    47
## [4,]   10   10    4   10    4    1    3    8    7    57
## [5,]    8    1    8    7    8    1    3    4   10    50
## [6,]    2    8   10    2    6    5    1    2    7    43
```

9.1.2 Calcular el máximo de cada columna, visualizarlo en una fila nueva.

```
maximos=c(
  max(matriz[,1]),
```

```

max(matriz[,2]),
max(matriz[,3]),
max(matriz[,4]),
max(matriz[,6]),
max(matriz[,7]),
max(matriz[,8]),
max(matriz[,9]))
mat_max=rbind(matriz,maximos)

## Warning in rbind(matriz, maximos): number of columns of result is not a multiple
## of vector length (arg 2)

mat_max

##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
##      3    6    7    3    4    9   10   10    3
##     10    6    6    9    9    5   10    8    6
##      8    7    1    4    9    3    3    9    3
##     10   10    4   10    4    1    3    8    7
##      8    1    8    7    8    1    3    4   10
##      2    8   10    2    6    5    1    2    7
## maximos 10   10   10   10    9   10   10   10   10

```

9.1.3 Calcular el producto matricial de A por su traspuesta.

```

matriz%*% t(matriz)

##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,] 409 414 277 304 248 250
## [2,] 414 559 380 451 396 293
## [3,] 277 380 319 335 257 201
## [4,] 304 451 335 455 336 257
## [5,] 248 396 257 336 368 252
## [6,] 250 293 201 257 252 287

```