MA0501 – Tarea 1

Diego Alberto Vega Víquez

2025-09-01

Tabla de contenidos

Ejercicio 1

Dado:

$$x = (3, -5, 31, -1, -9, 10, 0, 18) \qquad y = (1, 1, -3, 1, -99, -10, 10, -7)$$

Realice:

1. Introduzca `x` y `y` como vectores en R.

```r
x <- c(3, -5, 31, -1, -9, 10, 0, 18)
y <- c(1, 1, -3, 1, -99, -10, 10, -7)
```

2. Calcule media, varianza, raíz cuadrada y desviación estándar de `y`.

```r
mean(y) ①
var(y) ②
sqrt(y) ③
sd(y) ④
```

① Media de y
② Varianza de y
③ Raíz Cuadrada de y
④ Desviación Estandar de y

```
[1] -13.25
[1] 1236.786
[1] 1.000000 1.000000      NaN 1.000000      NaN      NaN 3.162278      NaN
[1] 35.16796
```

3. Calcule media, varianza, raíz cuadrada y desviación estándar de `x`.

```r
mean(x) ⑤
var(x) ⑥
sqrt(x) ⑦
sd(x) ⑧
```

⑤ Media de x
⑥ Varianza de x
⑦ Raíz Cuadrada de x
⑧ Desviación Estandar de x

```
[1] 5.875
[1] 174.9821
[1] 1.732051      NaN 5.567764      NaN      NaN 3.162278 0.000000 4.242641
[1] 13.22808
```

4. Calcule la correlación entre `x` y `y`.

```r
cor(x,y)
```

```
[1] 0.3659679
```

5. Comando en R para extraer entradas 2 a 7 de `x`.

```
x[2:7]
```

```
[1] -5 31 -1 -9 10  0
```

6. Comando en R para extraer entradas de `y` excepto la 2 y la 7.

```
y[-c(2,7)]
```

```
[1]   1  -3   1 -99 -10  -7
```

7. Comando en R para extraer entradas de `y` menores a -3 o mayores a 10.

```
y[(y < -3)|(y>10)]
```

```
[1] -99 -10  -7
```

8. Comando en R para extraer entradas de `x` mayores a 0 y pares.

```
x[(x>0)&(x%%2==0)]
```

```
[1] 10 18
```

Ejercicio 2

Introduzca en R la matriz 4×3:

```
A = matrix(c(1,2,3,4,5,6,7,8,9,10,11,12), nrow=4, "byrow"="true")
A
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
```

Luego, obtenga algunos elementos de la matriz de la siguiente manera: `A[1,1:3]`, `A[1:4,2]`, `A[3,3]`, `A[11]`, `A[20]`, `A[5,4]`, `A[1,1,1]` y explique qué pasa en cada caso.

```
A[1,1:3]
```

```
[1] 1 2 3
```

El código anterior lo que hace es que retorna la primera fila de la matriz

```r
A[1:4,2]
```

```
[1]  2  5  8 11
```

El código anterior retorna la 2da columna de la matriz

```r
A[3,3]
```

```
[1] 9
```

El código anterior retorna la el valor que se encuentra en la entrada 3,3 de la matriz

```r
A[11]
```

```
[1] 9
```

Aquí lo que pasa es que las matrices se almacenan por columnas: primero todos los elementos de la columna 1, luego los de la 2, y así sucesivamente. A[11] toma el 11.º elemento de ese vector interno, que corresponde a la fila 3, columna 3. Este valor depende de si la matriz se llenó por filas o por columnas.

```r
A[20]
```

```
[1] NA
```

En este caso da vacío pues vea que la matriz tiene $3 \times 4 = 12$ elementos por lo que no tiene sentido consultar la entrada del elemento 20 del vector equivalente a la matriz.

```r
A[5,4]
```

Este código da error debido a que se está solicitanto el valor de la entrada que se encuentra en la 5ta fila y en la 4ta columna, pero la matriz tiene solo 3 columnas y 4 filas.

```r
A[1,1,1]
```

Este código también da error, para explicar esto basta con recordar que que una matriz es una lista de listas, teniendo esto en cuenta se puede entender como un elemento en 2 dimensiones (filas, columnas), pero la consulta que se le está haciendo se pide una 'tercera dimension', para que esto fucione se tendría que trabajar con una lista de listas que tiene listas como contenido, graficamente se debería entender como un cubo en tres simensiones con ancho, largo y profundidad.

Ejercicio 3

Investigue para qué sirven los comandos de R `as.matrix(...)` y `as.data.frame(...)`. Explique y dé un ejemplo de cada uno.

`as.matrix(...)` es una función de R que tiene como objetivo convertir su argumento en una matriz, lo que dice su documentación es que "intenta" convertir su argumento en una matriz como la del ejemplo anterior.

`as.data.frame(...)` es una función que convierte su argumento en un objeto del tipo Data Frame, intuitivamente este objeto se puede entender como una tabla o base de dacos con columnas y filas, como una hoja de excel.

Ejercicio 4

Introduzca usando código R (no archivos) en un DataFrame la siguiente tabla de datos:

Peso	Edad	Nivel Educativo
76	25	Lic
67	23	Bach
55	19	Bach
57	18	Bach
87	57	Dr
48	13	MSc

```r
peso <- c(76,67,55,57,87,48)
edad <- c(25,23,19,18,57,13)
nivel <- c('Lic','Bach','Bach','Bach','Dr','MSc')

matriz <- list('Peso' = peso,
               'Edad' = edad,
               `Nivel Educativo` = nivel)
df <- as.data.frame(matriz)
df
```

```
  Peso Edad Nivel.Educativo
1   76   25             Lic
2   67   23            Bach
3   55   19            Bach
4   57   18            Bach
5   87   57              Dr
6   48   13             MSc
```

Ejercicio 5

En muchas ocasiones nos interesa hacer referencia a determinadas partes o componentes de un vector.
Defina el vector:

$$x = (2, -5, 4, 6, -2, 8)$$

Luego, a partir de este vector defina instrucciones en R para generar los siguientes vectores:

- $y = (2, 4, 6, 8)$, así definido y es el vector formado por las componentes positivas de x.
- $z = (-5, -2)$, así definido z es el vector formado por las componentes negativas de x.
- $v = (-5, 4, 6, -2, 8)$, así definido v es el vector x eliminada la primera componente.
- $w = (2, 4, -2)$, así definido w es el vector x tomando las componentes con índice impares, es decir, $x[1] = 2$, $x[3] = 4$ y $x[5] = -2$.

```r
x <- c(2, -5, 4, 6, -2, 8)                    ①
y <- x[x > 0]                                 ②
z <- x[x < 0]                                 ③
v <- x[-1]                                    ④
w <- x[(1:length(x)) %% 2 == 1]               ⑤
```

① Vector x con valores positivos y negativos.
② y: elementos positivos de x.
③ z: elementos negativos de x.
④ v: todos los elementos de x excepto el primero.
⑤ w: elementos en posiciones impares de x.

Ejercicio 6

Cargue la tabla de datos que está en el archivo `SAheartv.csv` y realice:

1. Calcule la dimensión de la tabla de datos.
2. Despliegue las primeras 3 columnas de la tabla.
3. Ejecute `summary()` y `str()` de los datos.
4. Usando `cor()` de R calcule la correlación entre las variables `tobacco` y `alcohol`.
5. Calcule la suma de las columnas con variables cuantitativas (numéricas).
6. Calcule para todas las variables cuantitativas presentes en `SAheart.csv`: mínimo, máximo, media, mediana y para la variable `chd` determine la cantidad de **Sí** y **No**.

```r
library(readr)
df <- read_delim(
  "~/Library/CloudStorage/OneDrive-UniversidaddeCostaRica/EMat/MA-0501/Recursos/Datos/Para la p
  delim = ";", escape_double = FALSE, trim_ws = TRUE
)                                                                                    ①

dim(df)                                                                              ②
df[1:3]                                                                              ③
summary(df) ; str(df)                                                                ④
cor(df$tobacco, df$alcohol)                                                          ⑤

cuantitativas <- df[sapply(df, is.numeric)]                                          ⑥
colSums(cuantitativas)                                                               ⑦

data.frame(
  Minimo  = sapply(cuantitativas, min),
  Maximo  = sapply(cuantitativas, max),
  Media   = sapply(cuantitativas, mean),
  Mediana = sapply(cuantitativas, median)
)                                                                                    ⑧

table(as.factor(df$chd))                                                             ⑨
```

① Cargar la base de datos
② Dimensión de la tabla
③ Despliegue de las primeras 3 columnas de la tabla
④ Resumen (summary()) y estructura (str()) de los datos
⑤ Calcular correlación entre tobacco y alcohol
⑥ Filtrar variables cuantitativas
⑦ Calcular la suma de las columnas numéricas
⑧ Calcular mínimo, máximo, media y mediana para variables cuantitativas
⑨ Contar cantidad de Sí y No en la variable chd

```
[1] 462  10
# A tibble: 462 x 3
     sbp tobacco   ldl
   <dbl>   <dbl> <dbl>
```

```
 1   160   12     5.73
 2   144    0.01  4.41
 3   118    0.08  3.48
 4   170    7.5   6.41
 5   134   13.6   3.5
 6   132    6.2   6.47
 7   142    4.05  3.38
 8   114    4.08  4.59
 9   114    0     3.83
10   132    0     5.8
# i 452 more rows
      sbp           tobacco             ldl           adiposity
 Min.   :101.0   Min.   : 0.0000   Min.   : 0.980   Min.   : 6.74
 1st Qu.:124.0   1st Qu.: 0.0525   1st Qu.: 3.283   1st Qu.:19.77
 Median :134.0   Median : 2.0000   Median : 4.340   Median :26.11
 Mean   :138.3   Mean   : 3.6356   Mean   : 4.740   Mean   :25.41
 3rd Qu.:148.0   3rd Qu.: 5.5000   3rd Qu.: 5.790   3rd Qu.:31.23
 Max.   :218.0   Max.   :31.2000   Max.   :15.330   Max.   :42.49
   famhist              typea         obesity         alcohol
 Length:462         Min.   :13.0   Min.   :14.70   Min.   :  0.00
 Class :character   1st Qu.:47.0   1st Qu.:22.98   1st Qu.:  0.51
 Mode  :character   Median :53.0   Median :25.80   Median :  7.51
                    Mean   :53.1   Mean   :26.04   Mean   : 17.04
                    3rd Qu.:60.0   3rd Qu.:28.50   3rd Qu.: 23.89
                    Max.   :78.0   Max.   :46.58   Max.   :147.19
      age            chd
 Min.   :15.00   Length:462
 1st Qu.:31.00   Class :character
 Median :45.00   Mode  :character
 Mean   :42.82
 3rd Qu.:55.00
 Max.   :64.00
spc_tbl_ [462 x 10] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
 $ sbp      : num [1:462] 160 144 118 170 134 132 142 114 114 132 ...
 $ tobacco  : num [1:462] 12 0.01 0.08 7.5 13.6 6.2 4.05 4.08 0 0 ...
 $ ldl      : num [1:462] 5.73 4.41 3.48 6.41 3.5 6.47 3.38 4.59 3.83 5.8 ...
 $ adiposity: num [1:462] 23.1 28.6 32.3 38 27.8 ...
```

```
 $ famhist  : chr [1:462] "Present" "Absent" "Present" "Present" ...
 $ typea    : num [1:462] 49 55 52 51 60 62 59 62 49 69 ...
 $ obesity  : num [1:462] 25.3 28.9 29.1 32 26 ...
 $ alcohol  : num [1:462] 97.2 2.06 3.81 24.26 57.34 ...
 $ age      : num [1:462] 52 63 46 58 49 45 38 58 29 53 ...
 $ chd      : chr [1:462] "Si" "Si" "No" "Si" ...
 - attr(*, "spec")=
  .. cols(
  ..   sbp = col_double(),
  ..   tobacco = col_double(),
  ..   ldl = col_double(),
  ..   adiposity = col_double(),
  ..   famhist = col_character(),
  ..   typea = col_double(),
  ..   obesity = col_double(),
  ..   alcohol = col_double(),
  ..   age = col_double(),
  ..   chd = col_character()
  .. )
 - attr(*, "problems")=<externalptr>
[1] 0.2008134
      sbp   tobacco       ldl adiposity     typea   obesity   alcohol       age
 63907.00   1679.67   2190.03  11737.91  24534.00  12032.38   7874.51  19781.00
          Minimo Maximo      Media Mediana
sbp       101.00 218.00 138.326840 134.000
tobacco     0.00  31.20   3.635649   2.000
ldl         0.98  15.33   4.740325   4.340
adiposity   6.74  42.49  25.406732  26.115
typea      13.00  78.00  53.103896  53.000
obesity    14.70  46.58  26.044113  25.805
alcohol     0.00 147.19  17.044394   7.510
age        15.00  64.00  42.816017  45.000

 No  Si
302 160
```

Ejercicio 7

Programe en R una función que genere 200 números al azar entre 1 y 500 y calcule cuántos están entre 50 y 450, ambos inclusive.

```
num.aleatorios <- function() {
  numeros <- runif(200, min = 1, max = 500)                    ①
  bool <- numeros<=450 & numeros>=50                           ②
  return(sum(bool))                                            ③
}
# Prueba
num.aleatorios()
```

① Genera 200 números aleatorios entre 1 y 500 (decimales)
② Verifica si están entre 50 y 450 inclusive
③ Cuenta cuántos cumplen la condición

```
[1] 166
```

Ejercicio 8

Desarrolle una función que calcule el costo de una llamada telefónica que ha durado t minutos, sabiendo que:

- Si $t < 1$: costo $= 0.4$ USD.
- Si $t \geq 1$: costo $= 0.4 + (t - 1)/4$ USD.

La función debe recibir el valor de t.

```
costo.llamada <- function(t) {
  if (t < 1) {
    costo <<- 0.4
  } else {
    costo <<- 0.4 + (t - 1)/4
  }
  return(costo)
}
costo.llamada(2)
```

```
[1] 0.65
```

```r
costo.llamada(.2)
```

```
[1] 0.4
```

Ejercicio 9

Desarrolle una función que reciba una matriz cuadrada A de tamaño $n \times n$ y calcule su traza, es decir, la suma de los elementos de la diagonal.

Ejemplo:

$$\begin{pmatrix} 9 & 3 & 4 \\ 1 & 3 & -1 \\ 4 & 12 & -2 \end{pmatrix}$$

Traza $= 10$.

```r
traza <- function(matriz) {
  suma <- sum(diag(matriz))
  return(suma)
}
# Prueba
A = matrix(c(9,3,4,1,3,-1,4,12,-2), nrow=3, byrow=TRUE)
traza(A)
```

```
[1] 10
```

Ejercicio 10

Escriba una función que genere los n primeros términos de la serie de Fibonacci.

```r
n.fibonacci <- function(n) {

  if (n == 1) return(0)                                                    ①
  if (n == 2) return(c(0, 1))

  if (n>=3) {
    serie <- c(0, 1)                                                       ②
    while (length(serie) < n) {
      serie <- c(serie, serie[length(serie)] + serie[length(serie) - 1])
```

```
    }
    return(serie)
  }
}
# Prueba
n.fibonacci(13)
```

① Casos base

② Generación para n >= 3

```
 [1]   0   1   1   2   3   5   8  13  21  34  55  89 144
```

Ejercicio 11

Escriba una función que retorne el mayor número entero cuyo cuadrado no exceda de x, donde x es un número real que se recibe como parámetro, utilizando `while`.

```
entero.cuadrado <- function(x) {
  if (x <= 0) {
    return(NA)
  } else {
    n <- 0
    while ((n+1)^2 <= x) {
      n <- n+1
    }
    return(n)
  }
}
# Prueba
entero.cuadrado(27)
```

```
[1] 5
```

Ejercicio 12

Cree un Data Frame con diez alumnos con su edad, año de nacimiento y número de teléfono. Deberá aparecer el nombre de la columna (`edad`, `año de nacimiento`, `teléfono`) y el nombre de la fila, que será el nombre del alumno.

```r
edad <- c(sample(17:25, 10, TRUE))

nums <- sample(60000000:89999999, 10)
telefonos <- paste0(
  "+506 ",
  substr(nums, 1, 4), " ",
  substr(nums, 5, 8)
)

alumnos <- as.data.frame(list(Edad = edad,
                  Año.Nacimiento = 2025-edad,
                  Teléfono = telefonos), row.names = c("Ana María","Carlos","Sofía","Javier","
alumnos
```

```
          Edad Año.Nacimiento       Teléfono
Ana María   20           2005 +506 6589 3453
Carlos      23           2002 +506 7664 2698
Sofía       25           2000 +506 6127 2371
Javier      25           2000 +506 8192 3890
Mariana     24           2001 +506 8556 1306
Luis        22           2003 +506 6032 8310
Fernanda    22           2003 +506 8764 0749
Pablo       20           2005 +506 8697 1484
Camila      24           2001 +506 6340 7817
Andrés      19           2006 +506 8295 8991
```

Ejercicio 13

Desarrolle una función en **R** que reciba un **DataFrame** y retorne la cantidad de entradas que son divisibles entre 3.

```r
divisibles.3 <- function(df) {

  datos <- unlist(df)                                                    ①
  datos <- datos[!is.na(as.numeric(datos))]
  datos <- as.numeric(datos)

  suma <- sum(datos %% 3 == 0)                                           ②
```

```r
  return(suma)
}
# Prueba
divisibles.3(data.frame(
  a = c(3, 4, 9),
  b = c(6, 7, 12)
))
```

① Convertir a vector numérico y descartar NAs
② Contar cuántos son divisibles entre 3

```
[1] 4
```

Ejercicio 14

Desarrolle una función en **R** que reciba un **DataFrame** y dos números de columna y que retorne en una lista:

- El nombre de las variables correspondientes a las columnas.
- La covarianza entre dichas variables.
- La correlación entre dichas variables.

```r
nombre.cov.corr <- function(df, x, y) {
  if (x <= ncol(df) & y <= ncol(df)) {
    col1 <- as.numeric(df[[x]])
    col2 <- as.numeric(df[[y]])
    return(list(
      `Nombre de variables` = c(colnames(df)[x], colnames(df)[y]),
      'Covarianza' = cov(col1, col2),
      'Correlación' = cor(col1, col2))
    )
  }
}
# Prueba
nombre.cov.corr(alumnos, 1, 2)
```

```
$`Nombre de variables`
[1] "Edad"           "Año.Nacimiento"
```

```
$Covarianza
[1] -4.711111

$Correlación
[1] -1
```

Ejercicio 15

Programe la siguiente función recursiva:

$$U(n) = \begin{cases} 5 & \text{si } n = 0 \\ -5 & \text{si } n = 1 \\ 2 & \text{si } n = 2 \\ 4U_{n-1} - 15U_{n-2} + U_{n-3} & \text{si } n \geq 3 \end{cases}$$

```r
U <- function(n) {
  if (n == 0) {
    return(5)
  } else if (n == 1) {
    return(-5)
  } else if (n == 2) {
    return(2)
  } else if (n >= 3) {
    return(4 * U(n - 1) - 15 * U(n - 2) + U(n - 3))
  }
}
# Prueba
U(5)
```

```
[1] -50
```

Ejercicio 16

El **Algoritmo de Luhn** es una fórmula de suma de verificación utilizada para validar una diversidad de números de identificación.
Por ejemplo, para validar si un número es o no un número de tarjeta de crédito válido.

Este algoritmo es muy simple: nos dice que, dado un número que contenga solamente dígitos $[0-9]$,

una tarjeta de crédito es válida si y solo si, obteniendo la reversa de este número, la suma especial de sus dígitos debe ser un múltiplo de 10, es decir, que la suma módulo 10 debe ser igual a cero.

La suma especial se realiza de la siguiente forma:

1. Invertir el número.
2. Si la posición es impar, sumar el dígito tal cual.
3. Si la posición es par, multiplicar el dígito por 2 y sumar los dígitos del resultado (o restar 9 si es mayor o igual a 10).
4. Verificar que la suma total sea un múltiplo de 10.

Ejemplo 1

Número: *49927398716*

a) Multiplicamos por 2 los dígitos que ocupan las posiciones pares empezando por el final (reversando el número):

$$(1 \times 2) = 2, \quad (8 \times 2) = 16, \quad (3 \times 2) = 6, \quad (2 \times 2) = 4, \quad (9 \times 2) = 18$$

4	9	9	2	7	3	9	8	7	1	6
4	18	9	4	7	6	9	16	7	2	6

b) Sumamos los dígitos que ocupaban las posiciones impares con los dígitos de los productos obtenidos:

$$6 + (2) + 7 + (1 + 6) + 9 + (6) + 7 + (4) + 9 + (1 + 8) + 4 = 70$$

O equivalentemente, sumando y restando 9 en los productos mayores o iguales a 10:

$$6 + (2) + 7 + (16 - 9) + 9 + (6) + 7 + (4) + 9 + (18 - 9) + 4 = 70$$

c) Si el resto de dividir 70 entre 10 es igual a 0, el número es válido:

$$70 \bmod 10 = 0$$

Ejemplo 2

Número de tarjeta: 4539319503436467

a) Primero se toman todos los números en las posiciones pares empezando desde la derecha, es decir, reversando el número:

$$4\_3\_3\_3\_9\_0\_4\_6\_6\_$$

b) Luego se deben cambiar por el doble de ese número; si el número obtenido es mayor a 9, entonces se le resta 9 (o se suman los dígitos):

$$8\_6\_6\_9\_0\_8\_3\_3\_$$

c) Finalmente se suman todos los dígitos:

$$8+5+6+9+6+1+9+5+0+3+8+3+3+4+3+7=80$$

Como el número obtenido es divisible entre 10, entonces es un número de tarjeta válido.

Programe en **R** las funciones necesarias para implementar el **Algoritmo de Luhn**.

```r
# Intercala elementos de dos vectores alternando posiciones
intercalar <- function(primero, segundo) {
  total <- length(primero) + length(segundo)
  mezcla <- vector("list", total)                                    ①
  mezcla[seq(1, total, 2)] <- as.list(primero)                       ②
  mezcla[seq(2, total, 2)] <- as.list(segundo)                       ③
  unlist(mezcla, use.names = FALSE)
}

# Si el valor (resultado de doblar un dígito) es >= 10, restar 9 (suma de dígitos).
ajustar_mayor_9 <- function(valor) {
  if (valor >= 0 & valor <= 18) {
    if (valor >= 10) return(valor - 9)
    else return(valor)
  }
}

# Devuelve TRUE si 'numero' pasa la validación, FALSE en caso contrario.
Algoritmo.Luhn <- function(numero) {

  digitos <- as.numeric(strsplit(as.character(numero), "")[[1]])     ④
```

```r
  n <- length(digitos)

  digitos_inv <- digitos[n:1]                                                    ⑤

  # Separar posiciones impares y pares (desde la derecha, ya invertido)
  digitos_impares <- digitos_inv[(1:n) %% 2 == 1]                                ⑥
  digitos_pares_doblados <- digitos_inv[(1:n) %% 2 == 0] * 2                     ⑦

  intercalados <- intercalar(digitos_impares, digitos_pares_doblados)            ⑧

  ajustados <- sapply(intercalados, ajustar_mayor_9)                             ⑨
  suma_total <- sum(ajustados)

  return(suma_total %% 10 == 0)                                                  ⑩
}

# Prueba
Algoritmo.Luhn(49927398716)
Algoritmo.Luhn(4539319503436467)
```

① Preasignación (lista para permitir longitudes distintas)
② Posiciones impares
③ Posiciones pares
④ Convertir número a vector de dígitos
⑤ Invertir para procesar desde el dígito de la derecha
⑥ se quedan igual
⑦ se doblan
⑧ Intercalar: impares, luego pares doblados (manteniendo el orden original de tu código)
⑨ Ajustar los valores >= 10 restando 9 y sumar
⑩ Válido si la suma es múltiplo de 10

```
[1] TRUE
[1] TRUE
```

Ejercicio 17

La **Factorización Prima** es el proceso de descomponer un número en sus factores primos.
El algoritmo consiste en dividir entre números primos hasta que el cociente sea 1.

Por ejemplo, para obtener los factores primos de 36 se hace:

$$36 \div 2 = 18; \quad 18 \div 2 = 9; \quad 9 \div 3 = 3; \quad 3 \div 3 = 1$$

Entonces los factores primos son: $2, 2, 3, 3$ y la factorización prima de $36 = 2 \times 2 \times 3 \times 3$.

Programe en **R** una función recursiva que reciba un valor y retorne una lista con los factores primos de dicho valor.

Los factores primos de un número entero son los números primos divisores exactos de ese número entero.
El proceso de búsqueda de esos divisores se denomina factorización de enteros, o factorización en números primos.

Por ejemplo, para 60 el proceso es el siguiente:

- $60/2 = 30$
- $30/2 = 15$
- $15/3 = 5$
- $5/5 = 1$

Por tanto, el resultado es: **(2 2 3 5)**.

```
primos_menores_a <- function(n) {
  if (n <= 2) return(integer(0))                                    ①
  primos <- rep(TRUE, n - 1)                                        ②
  limite <- entero.cuadrado(n)

  for (p in 2:limite) {
    if (primos[p - 1]) {
      primos[seq(p * p, n - 1, by = p) - 1] <- FALSE
    }
  }

  which(primos) + 1                                                 ③
}
```

```r
# Recursiva auxiliar
.factoriza_rec <- function(n, primos, i = 1L, acc = integer(0)) {
  if (n == 1) {
    return(acc)
  }
  if (i > length(primos) || primos[i]^2 > n) {
   return(c(acc, n))                                                    ④
  }
  p <- primos[i]
  if (n %% p == 0) {                                                    ⑤
    return(.factoriza_rec(n / p, primos, i, c(acc, p)))
  } else {                                                              ⑥
    return(.factoriza_rec(n, primos, i + 1L, acc))
  }
}

factorizacion.prima <- function(n) {
  if (n < 2) return(list(factores = integer(0)))
  ps <- primos_menores_a(n)
  list(factores = .factoriza_rec(n, ps))
}

factorizacion.prima(76)
```

① No hay primos menores que 2
② índice 1 representa el número 2
③ Ajuste porque el índice 1 corresponde al 2
④ Si ya no hay más primos o p^2 > n, lo que queda es primo
⑤ Misma i para capturar potencias de p
⑥ Pasar al siguiente primo

```
$factores
[1]  2  2 19
```

Ejercicio 18

La **Conjetura de Goldbach** dice que todo número par mayor a 2 se puede expresar como la suma de 2 números primos.

Por tanto, desarrolle un sistema que reciba un número (por teclado) y retorne una lista con los 2 números primos que componen la suma.
Se puede asumir que el número ingresado siempre es mayor a 2 y es par.

Debe programar en **R** lo siguiente:

a) `es_primo`: determina si un valor es primo. Debe programarla con recursión.

b) `valores`: obtiene en una lista los 2 números que conforman la suma. Debe programarla con recursión.

c) `resultado`: para un valor dado despliega la lista con los números primos.

```r
es_primo <- function(n, d = 2) {
  if (n < 2) return(FALSE)                                    ①
  if (d * d > n) return(TRUE)                                 ②
  if (n %% d == 0) return(FALSE)                              ③

  siguiente <- if (d == 2) 3 else d + 2                       ④
  es_primo(n, siguiente)
}

valores <- function(n, a = 2) {
  b <- n - a
  if (es_primo(a) && es_primo(b)) {
    return(list(a = a, b = b))
  }
  siguiente <- if (a == 2) 3 else a + 2
  valores(n, siguiente)
}

resultado <- function(x = NULL) {
  if (is.null(x)) {
    entrada <- readline("Ingrese un número par (>2): ")
    x <- as.integer(entrada)
  }
  return(valores(x))
}
```

① 0 y 1 no son primos

② no hay divisores hasta √n

③ divisor encontrado

④ si d == 2, saltar a 3; luego avanzar de 2 en 2 (solo impares)

Ejercicio 19

Desarrolle una función en **R** que reciba un Data Frame y dos números de columna y que retorne en una lista:

- El nombre de las variables correspondientes a las columnas.
- La covarianza entre esas dos variables.
- La correlación entre esas dos variables.

Pruebe la función usando el archivo `EjemploEstudiantes.csv`.

```
nombre.cov.corr <- function(df, x, y) {
  if (x <= ncol(df) & y <= ncol(df)) {
    col1 <- as.numeric(df[[x]])
    col2 <- as.numeric(df[[y]])
    return(list(
      `Nombre de variables` = c(colnames(df)[x], colnames(df)[y]),
      'Covarianza' = cov(col1, col2),
      'Correlación' = cor(col1, col2))
    )
  }
}
# Prueba
df <- read_delim("~/Library/CloudStorage/OneDrive-UniversidaddeCostaRica/EMat/MA-0501/Recursos/
    delim = ";", escape_double = FALSE, trim_ws = TRUE)

nombre.cov.corr(df,3,5)
```

```
$`Nombre de variables`
[1] "Ciencias" "Historia"

$Covarianza
[1] -857.6889

$Correlación
```

```
[1] -0.7290368
```