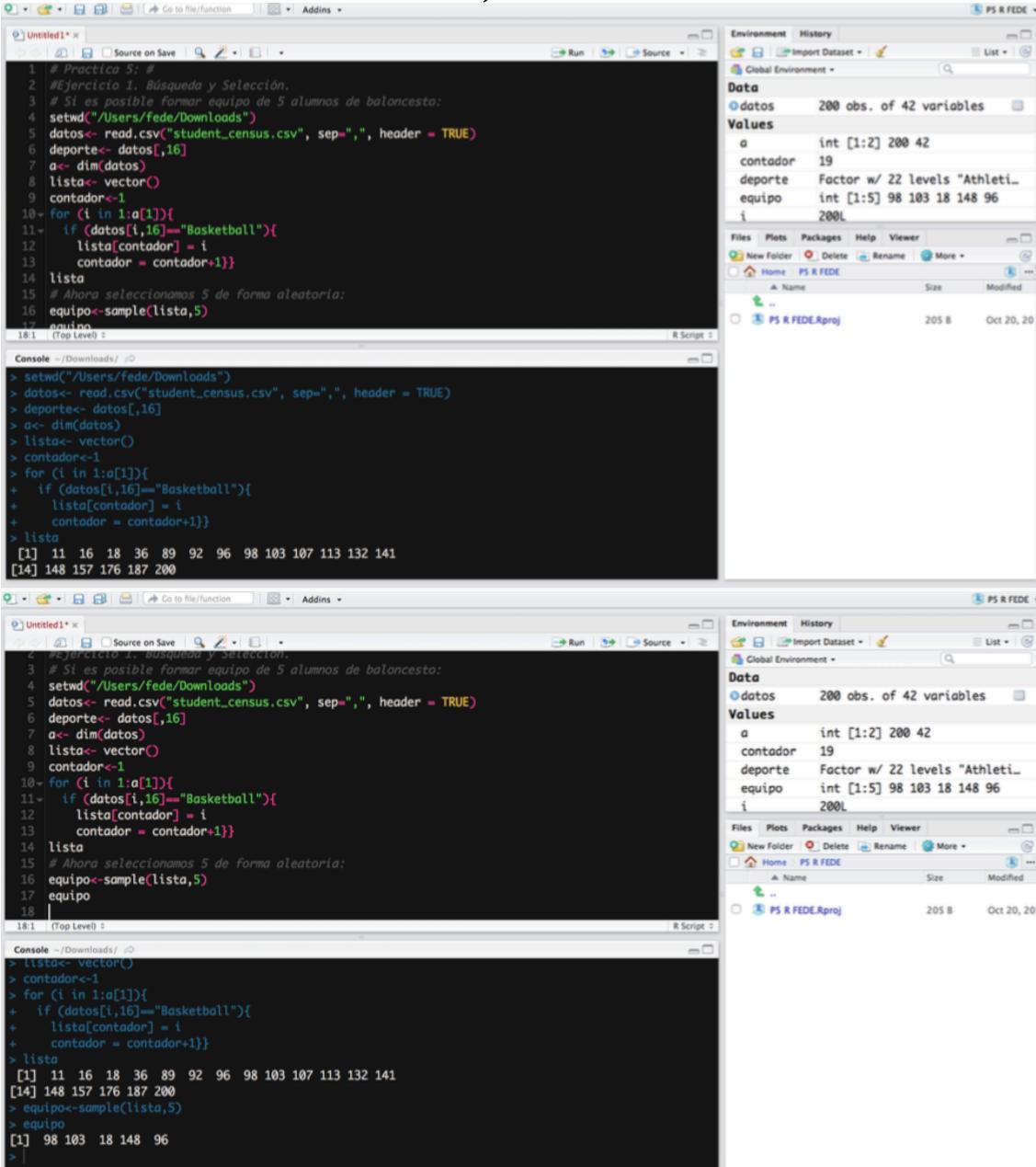


PRACTICA 5 PROGRAMACION EN R:

**Nota: En las capturas de pantalla del programa de incluyen tambien algunas explicaciones al código utilizado.

EJERCICIO 1:

En el primer ejercicio, cargamos los datos del archivo deseado, mediante la función `read.csv()`, y una vez cargado nuestro data set, generaremos una función que coge números aleatorios, en este caso alumnos, con el rango adecuado, en la que el estudiante se unirá al equipo en caso de jugar al baloncesto, y si no, se generarán números aleatorios hasta tener un conjunto de 5.



The image shows two side-by-side sessions of RStudio. Both sessions have the same code and output, demonstrating the execution of a script to select basketball players from a dataset.

```
1 # Practica 5: #
2 #Ejercicio 1. Búsqueda y Selección.
3 # Si es posible formar equipo de 5 alumnos de baloncesto:
4 setwd("~/Users/fede/Downloads")
5 datos<- read.csv("student_census.csv", sep=",", header = TRUE)
6 deporte<- datos[,16]
7 a<- dim(datos)
8 lista<- vector()
9 contador<-1
10 for (i in 1:a[1]){
11   if (datos[i,16]=="Basketball"){
12     lista[contador] = i
13     contador = contador+1}
14 lista
15 # Ahora seleccionamos 5 de forma aleatoria:
16 equipo<-sample(lista,5)
17 equipo
```

Output (from both sessions):

```
[1] 11 16 18 36 89 92 96 98 103 107 113 132 141
[14] 148 157 176 187 200
```

Environment pane (from both sessions):

- Data: datos 200 obs. of 42 variables
- Values:
 - a int [1:2] 200 42
 - contador 19
 - deporte Factor w/ 22 levels "Athleti..."
 - equipo int [1:5] 98 103 18 148 96
 - i 200L
- Files, Plots, Packages, Help, Viewer tabs are visible.

Finalmente obtenemos el output “equipo” como conjunto de 5 alumnos que practican baloncesto.

EJERCICIO 2:

Se pide con los datos del ejercicio anterior crear un data frame con las variables Height, Arm.Span y Foot.Size, realizar correlaciones de estas variables dos a dos y finalmente crear un histograma con la información de los resultados por provincia:

The figure shows two RStudio sessions side-by-side.

Top Session:

```
14 lista
15 # Ahora seleccionamos 5 de forma aleatoria:
16 equipo<-sample(lista,5)
17 equipo
18 #Ejercicio 2. Ordenación y Regresión.
19 #Filtrar en un data.frame las variables Height, Arm.Span, Foot.Size y realizar correlaciones dos
20 #a dos.
21 height<-datos$Height
22 arm_span<-datos$Arm.Span
23 foot_size<-datos$Foot.Size
24 df1<-data.frame(height,arm_span,foot_size)
25 df1
26 # correlacion dos a dos:
27 correlacion<-cor(df1)
28 correlacion
29
30
```

Console output:

```
28:12 | (Top Level) | R Script |
```

```
195   158   162   25
196   175   152   27
197   154   157   26
198   180   188   28
199   187   182   27
200   177   172   26
> correlacion<-cor(df1)
> correlacion
```

	height	arm_span	foot_size
height	1.0000000	0.7498349	0.5006804
arm_span	0.7498349	1.0000000	0.4615200
foot_size	0.5006804	0.4615200	1.0000000

> |

Bottom Session:

```
18 #Ejercicio 2. Ordenación y Regresión.
19 #Filtrar en un data.frame las variables Height, Arm.Span, Foot.Size y realizar correlaciones dos
20 #a dos.
21 height<-datos$Height
22 arm_span<-datos$Arm.Span
23 foot_size<-datos$Foot.Size
24 df1<-data.frame(height,arm_span,foot_size)
25 df1
26 # correlacion dos a dos:
27 correlacion<-cor(df1)
28 correlacion
29 # ordenamos por altura y provincia:
30 (ii<-order(height, arm_span, foot_size))
31 cbind(height, arm_span, foot_size)[ii,]
32 plot(datos$Province)
```

Console output:

```
33:1 | (Top Level) | R Script |
```

```
> (ii<-order(height, arm_span, foot_size))
 [1] 113 183 144 179 66 10 64 71 154 62 194 63 169
 [14] 29 139 115 100 197 18 163 89 88 48 153 51 79
 [27] 171 140 50 132 5 192 164 195 134 142 123 116 186
 [40] 77 173 148 157 123 182 178 92 96 31 172 189 146
 [53] 137 68 159 14 161 37 90 21 49 136 162 3 170
 [66] 147 85 126 130 2 44 45 124 36 27 38 119 81
 [79] 82 188 86 166 42 84 117 16 93 112 175 22 20
 [92] 99 13 168 107 43 83 111 8 55 107 143 125 60
[105] 1 34 190 104 177 155 110 131 69 180 114 40 33
[118] 91 106 12 61 32 94 35 58 80 185 181 156 72
[131] 184 149 54 102 151 109 108 76 87 4 28 158 30
[144] 19 145 167 152 41 196 105 176 73 6 141 122 121
```

Environment pane (Top Session):

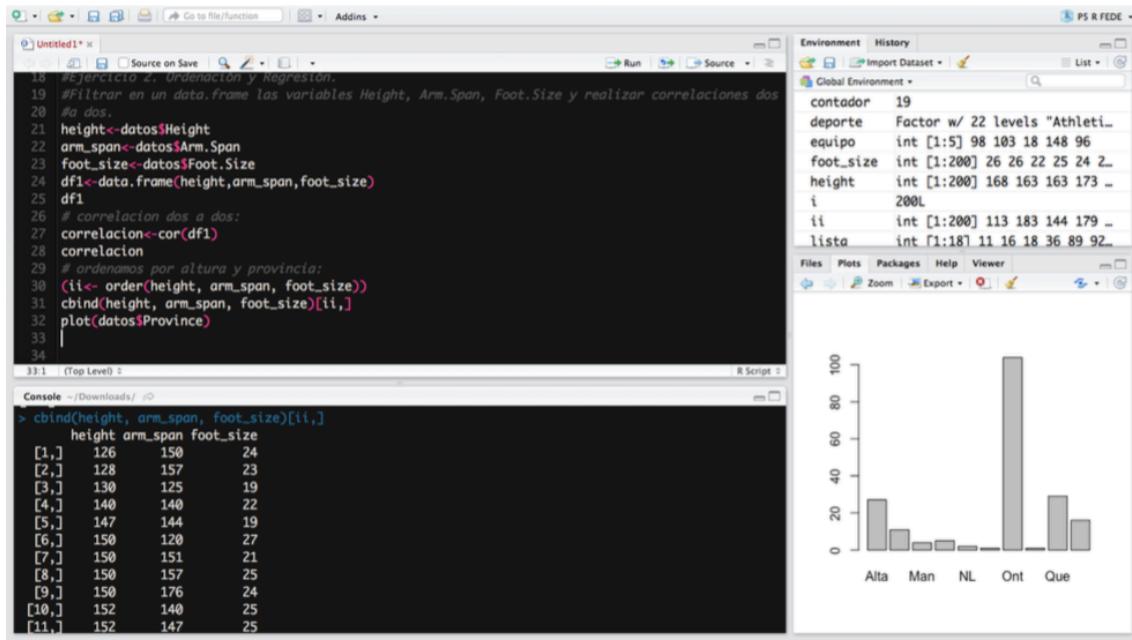
- correlacion num [1:3, 1:3] 1 0.75 0.50...
- datos 200 obs. of 42 variables
- df1 200 obs. of 3 variables
- foot.size 200 obs. of 1 variable
- matriz 200 obs. of 3 variables
- tabla_corr_num [1:3, 1:3] 1 0.75 0.50...

Environment pane (Bottom Session):

- contador int [1:2] 19 19
- deporte Factor w/ 22 levels "Athleti..."
- equipo int [1:5] 98 103 18 148 96
- foot_size int [1:200] 26 26 22 25 24 2...
- height int [1:200] 168 163 163 173 1...
- i 200L
- ii int [1:200] 113 183 144 179 1...
- lista int [1:18] 11 16 18 36 89 92...

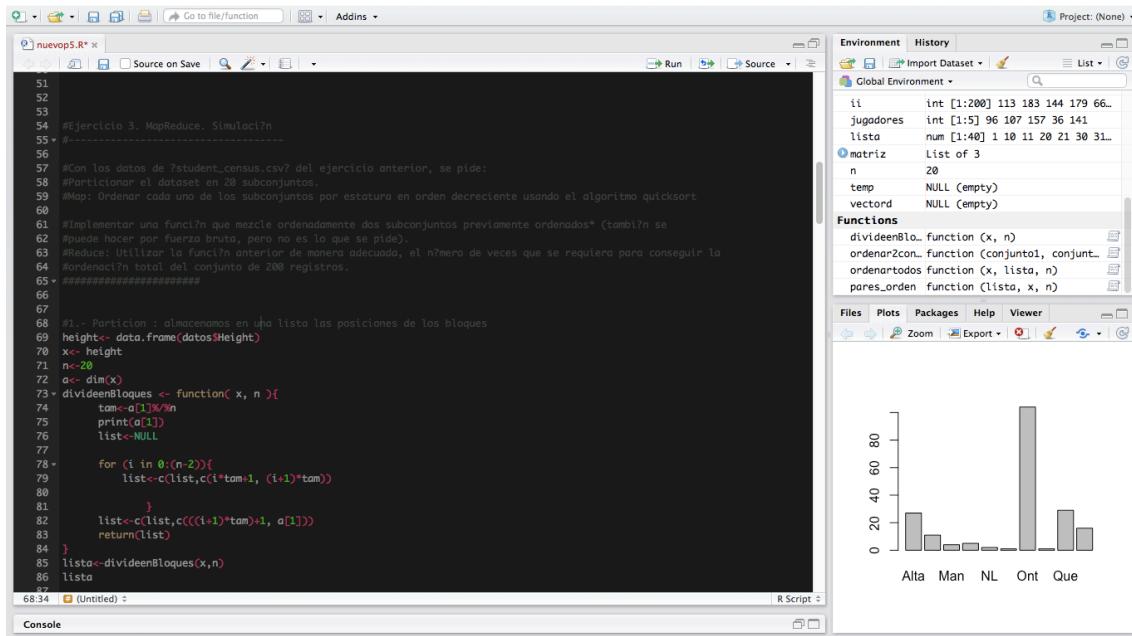
Plots pane (Bottom Session):

Province	Count
Alta	~25
Man	~10
NL	~5
Ont	~10
Que	~15



EJERCICIO 3:

En primer lugar, creamos una función que nos devuelva una lista que delimita donde empieza y acaba cada bloque:



Mostramos la lista:

```

nuevop5.R* ~ [1] > source("nuevop5.R")
[1] "nuevop5.R"
[2] "divideenBloques <- function(x, n) {
[3]   tam<-a[1]%
[4]   print(a[1])
[5]   list<-NULL
[6]
[7]   for (i in 0:(n-2)){
[8]     list<-c(list,c(i*tam+1, (i+1)*tam)))
[9]
[10]   }
[11]   list<-c(list,c(((i+1)*tam)+1, a[1]))
[12]   return(list)
[13] }
[14] lista<-divideenBloques(x,n)
[15] lista
[16] > "
[17] > divideenBloques <- function(x, n){
[18]   tam<-a[1]%
[19]   print(a[1])
[20]   list<-NULL
[21]
[22]   for (i in 0:(n-2)){
[23]     list<-c(list,c(i*tam+1, (i+1)*tam))
[24]
[25]   }
[26]   list<-c(list,c(((i+1)*tam)+1, a[1]))
[27]   return(list)
[28] }
[29] lista<-divideenBloques(x,n)
[30] lista
[31] [1] 200
[32] > lista
[33] [1] 1 10 11 20 21 30 31 40 41 50 51 60 61 70 71 80 81 90 91
[34] [20] 100 101 110 111 120 121 130 131 140 141 150 151 160 161 170 171 180 181 190
[35] [39] 191 200
[36] > "
[37] > "

```

El siguiente paso consiste en ordenar cada bloque del vector de mayor o menor con la función Quicksore pero no me funciona y utilizamos una función alternativa que creamos y ejecutamos:

```

nuevop5.R* ~ [1] > source("nuevop5.R")
[1] "nuevop5.R"
[2] "divideenBloques <- function(x, n) {
[3]   tam<-a[1]%
[4]   print(a[1])
[5]   list<-NULL
[6]
[7]   for (i in 0:(n-2)){
[8]     list<-c(list,c(i*tam+1, (i+1)*tam)))
[9]
[10]   }
[11]   list<-c(list,c(((i+1)*tam)+1, a[1]))
[12]   return(list)
[13] }
[14] lista<-divideenBloques(x,n)
[15] lista
[16] > "
[17] > divideenBloques <- function(x, n){
[18]   tam<-a[1]%
[19]   print(a[1])
[20]   list<-NULL
[21]
[22]   for (i in 0:(n-2)){
[23]     list<-c(list,c(i*tam+1, (i+1)*tam))
[24]
[25]   }
[26]   list<-c(list,c(((i+1)*tam)+1, a[1]))
[27]   return(list)
[28] }
[29] lista<-divideenBloques(x,n)
[30] lista
[31] [1] 200
[32] > lista
[33] [1] 1 10 11 20 21 30 31 40 41 50 51 60 61 70 71 80 81 90 91
[34] [20] 100 101 110 111 120 121 130 131 140 141 150 151 160 161 170 171 180 181 190
[35] [39] 191 200
[36] > "
[37] > "

```

Screenshot of RStudio showing the code for the `pares_orden` function and a histogram.

```

100 if (x[j, 1] < x[j+1, 1])
101 {
102     temp = x[j, ]
103     x[j, ]=x[j+1,]
104     x[j+1, ]=temp
105 }
106 }
107 }
108 }
109 }
110 return(x)
111 }
112 a<- pares_orden(lista, x, n)
113
109.3 [pares_orden(lista, x, n) :]

Console ~/Downloads/ ~
> x<- data.frame(height, list(1:200))
> pares_orden<-function (lista, x, n){
+   for (l in 0:(n-1)){
+     init<-lista[1*(l+2)+1]
+     fin<-lista[1*(l+2)+2]
+     for (l in (init:fin))
+     {
+       for (j in (init:(fin-1)))
+       {
+         if (x[j, 1] < x[j+1, 1])
+         {
+           temp = x[j, ]
+           x[j, ]=x[j+1,]
+           x[j+1, ]=temp
+         }
+       }
+     }
+   }
+   return(x)
+ }
> a<- pares_orden(lista, x, n)

```

Global Environment:

- ii int [1:200] 113 183 144 179 66...
- jugadores int [1:5] 96 107 157 36 141
- lista num [1:40] 1 10 11 20 21 30 31...
- matriz List of 3
- n 20
- temp NULL (empty)
- vectorx NULL (empty)

Functions:

- divideenBlo... function (x, n)
- ordenar2con... function (conjunto1, conjunt...
- ordenardatos function (x, lista, n)
- pares_orden function (lista, x, n)

Files Plots Packages Help Viewer

Ordenamos los dos conjuntos:

Screenshot of RStudio showing the code for the `ordenar2conjuntos` function and a histogram.

```

117 conjunto1<- c(lista[1],lista[2], 1]
118 conjunto2<- c(lista[3],lista[4], 1)
119 vectorx<-NULL
120 ordenar2conjuntos<- function(conjunto1, conjunto2)
121 {
122   vectorx<- c(1:length(conjunto2)>length(conjunto1)))
123   contador<- 1
124   contador2<- 1
125   for (i in 1:(length(conjunto1)))
126   {
127     while (contador2<=(length(conjunto2)))
128     {
129       if (conjunto1[i] >= conjunto2[contador2])
130       {
131         vectorx[contador]<- conjunto1[i]
132         contador<-contador+1
133         break
134       }
135       if (conjunto1[i] <= conjunto2[contador2])
136       {
137         vectorx[contador]<- conjunto2[contador2]
138         contador<-contador+1
139         contador2 = contador2+1
140       }
141     }
142     if (conjunto1[length(conjunto1)] > conjunto2[length(conjunto2)])
143     {
144       vectorx[length(vectorx)]<-conjunto2[length(conjunto2)]
145     }
146     else
147     {
148       vectorx[length(vectorx)]<-conjunto1[length(conjunto1)]
149     }
150   }
151   return(vectorx)
152 }
153 ordenar2conjuntos(conjunto1, conjunto2)

152.1 [ (Untitled) :]

Console

```

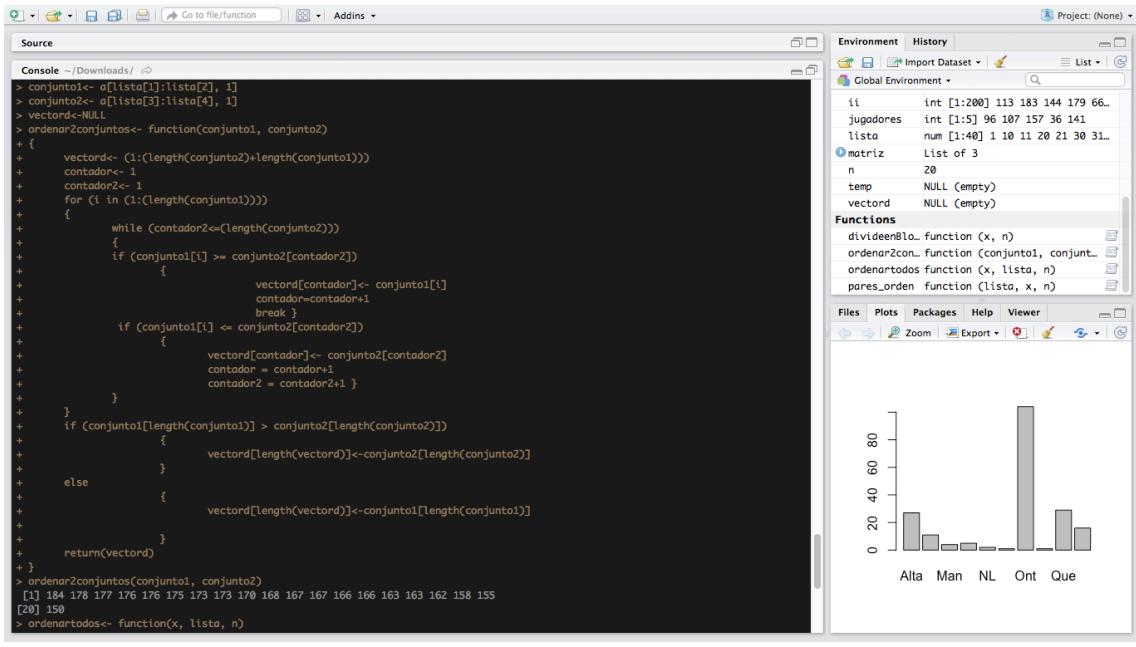
Global Environment:

- ii int [1:200] 113 183 144 179 66...
- jugadores int [1:5] 96 107 157 36 141
- lista num [1:40] 1 10 11 20 21 30 31...
- matriz List of 3
- n 20
- temp NULL (empty)
- vectorx NULL (empty)

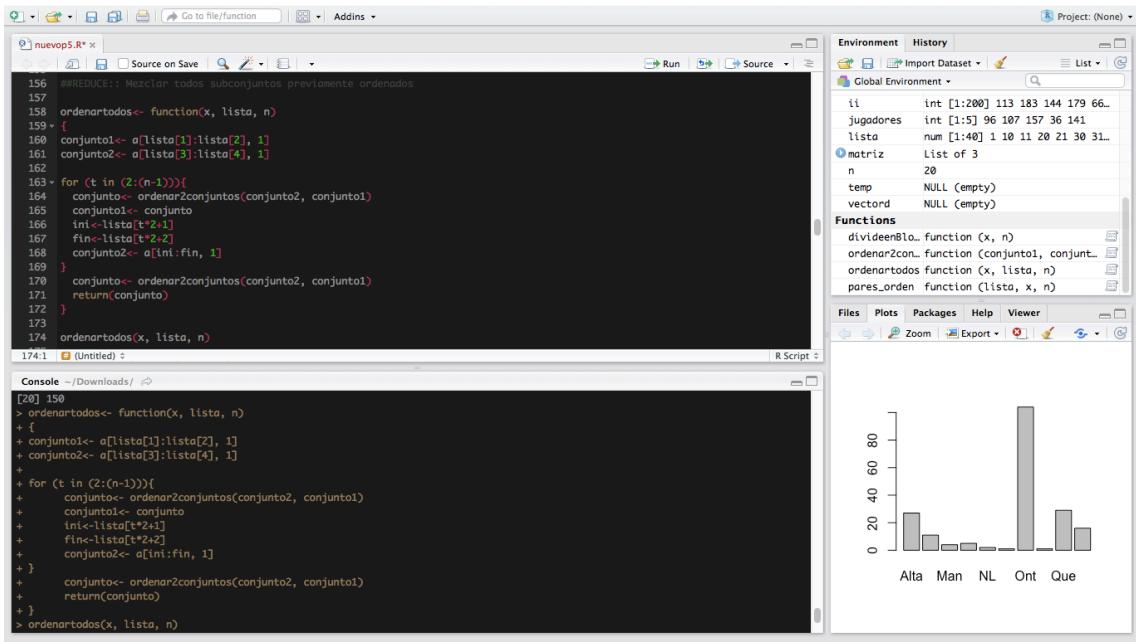
Functions:

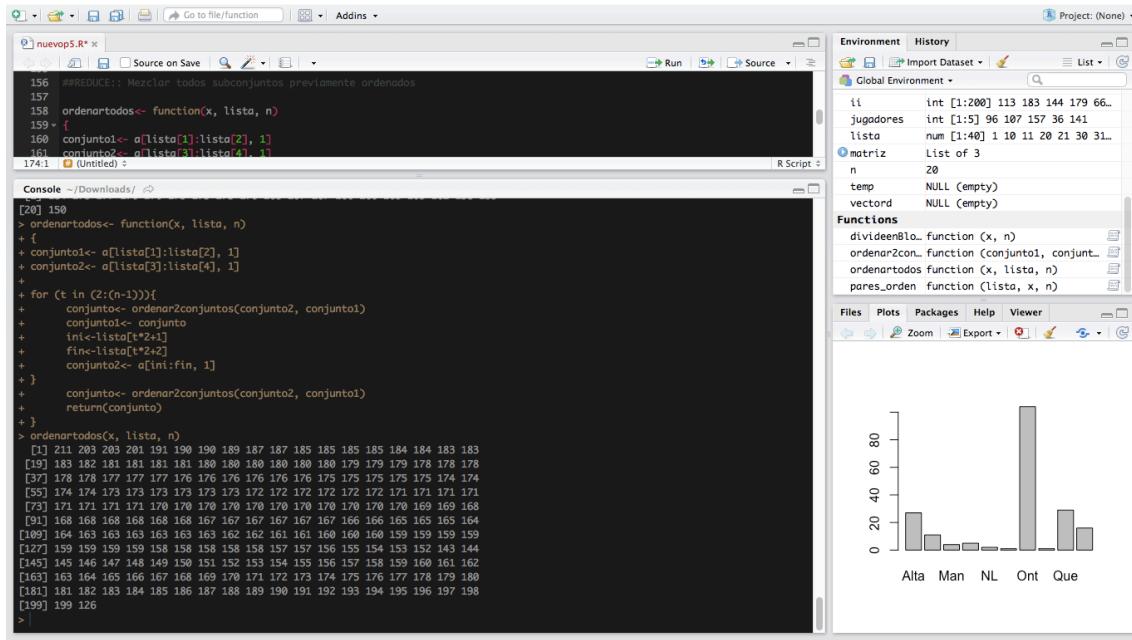
- divideenBlo... function (x, n)
- ordenar2con... function (conjunto1, conjunt...
- ordenardatos function (x, lista, n)
- pares_orden function (lista, x, n)

Files Plots Packages Help Viewer



Finalmente aplicamos el la técnica Reduce, para ordenar las alturas de mayor a menor, donde la nueva función “ordenartodos” donde en cada iteración de irán ordenando las alturas del conjunto1 en la vuelta anterior, hasta que todas las alturas estén ordenadas. Obteniendo:

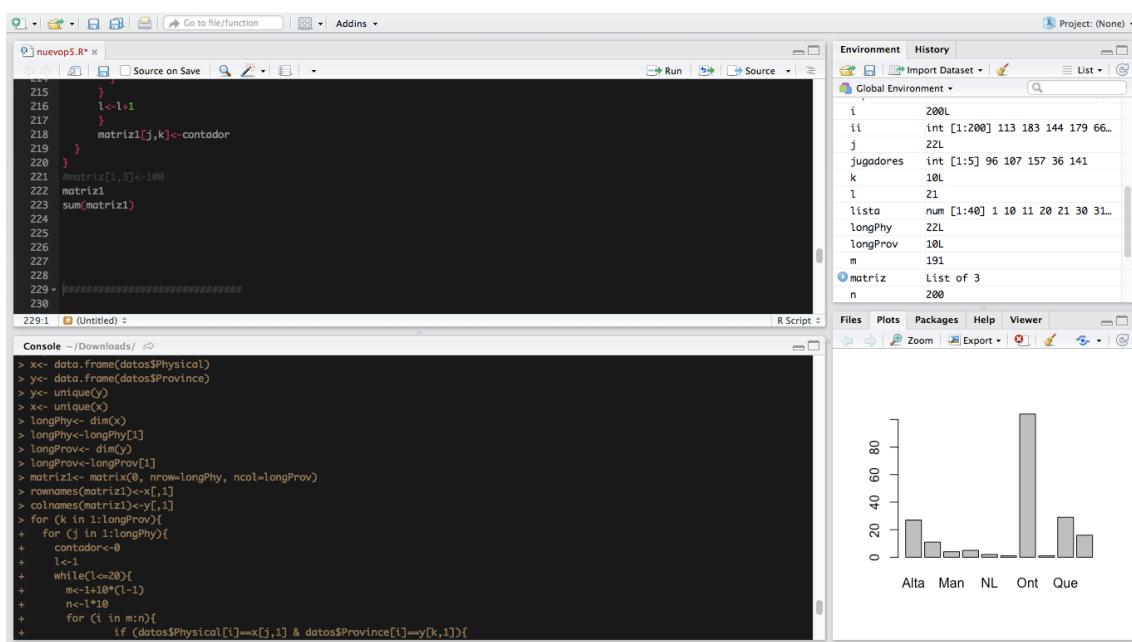
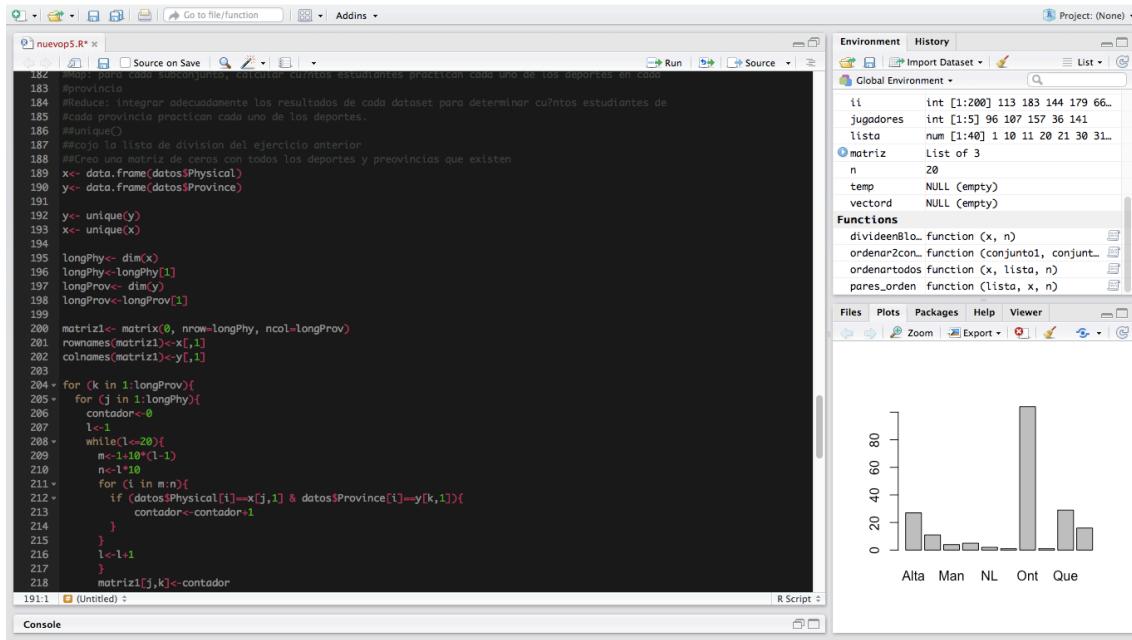


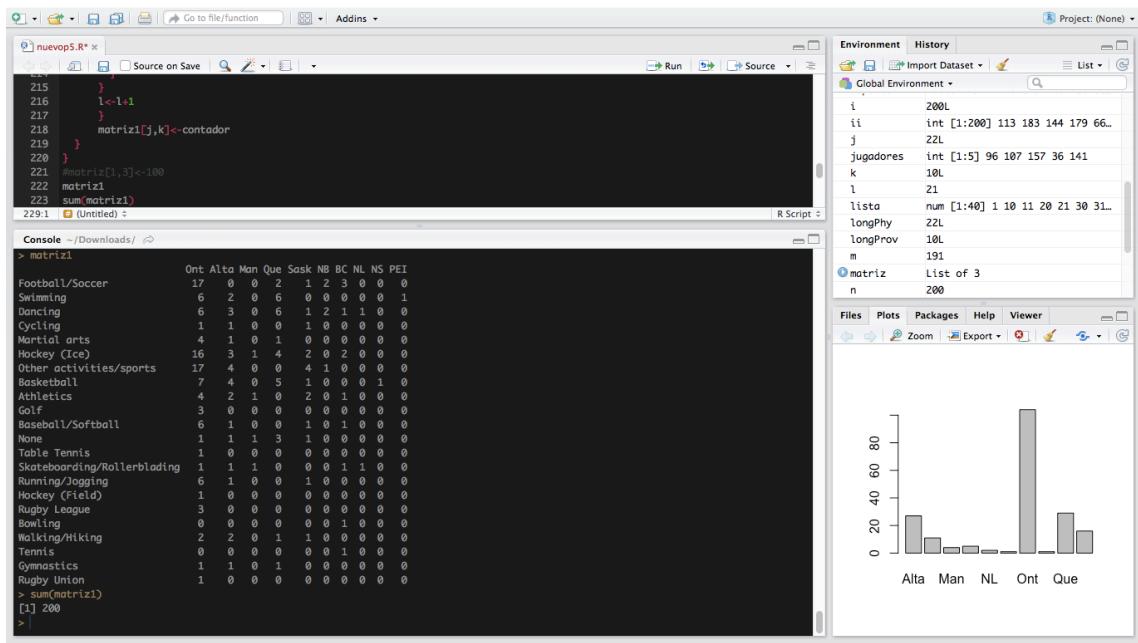


EJERCICIO 4:

Clasificación Con los datos de “student_census.csv” del ejercicio anterior, se pide: Partitionar el dataset en 20 subconjuntos. Map: para cada subconjunto, calcular cuántos estudiantes practican cada uno de los deportes en cada provincia Reduce: integrar adecuadamente los resultados de cada dataset para determinar cuántos estudiantes de cada provincia practican cada uno de los deportes.

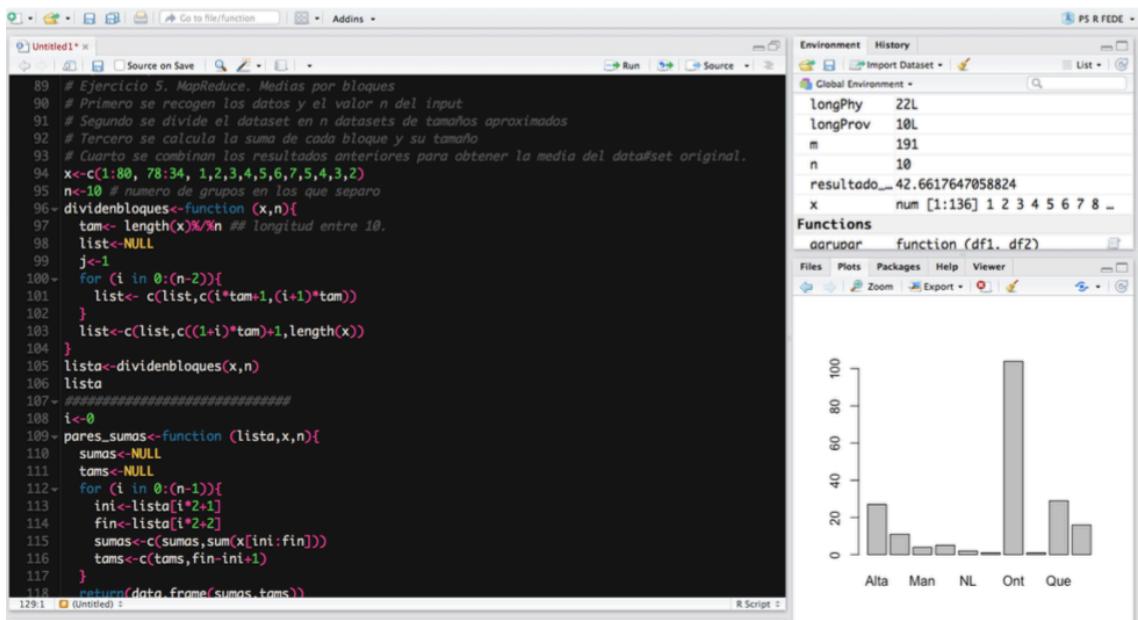
1. Creamos un dataframe con las variables deportes y provincias y creamos dos vectores con las mismas variables y los datos no repetidos con la función “unique()”.
2. Creamos matriz de ceros que tenga como numero de filas el numero de deportes diferentes existentes en el data frame y numero de columnas el numero de provincias del mismo.
3. Se va haciendo un conteo, mediante un “contador” creado, de los individuos que realizan cada deporte en las distintas provincias.
4. Devuelve la matriz de la clasificación por provincias de los deportes practicados por cada persona incluida en el data set.

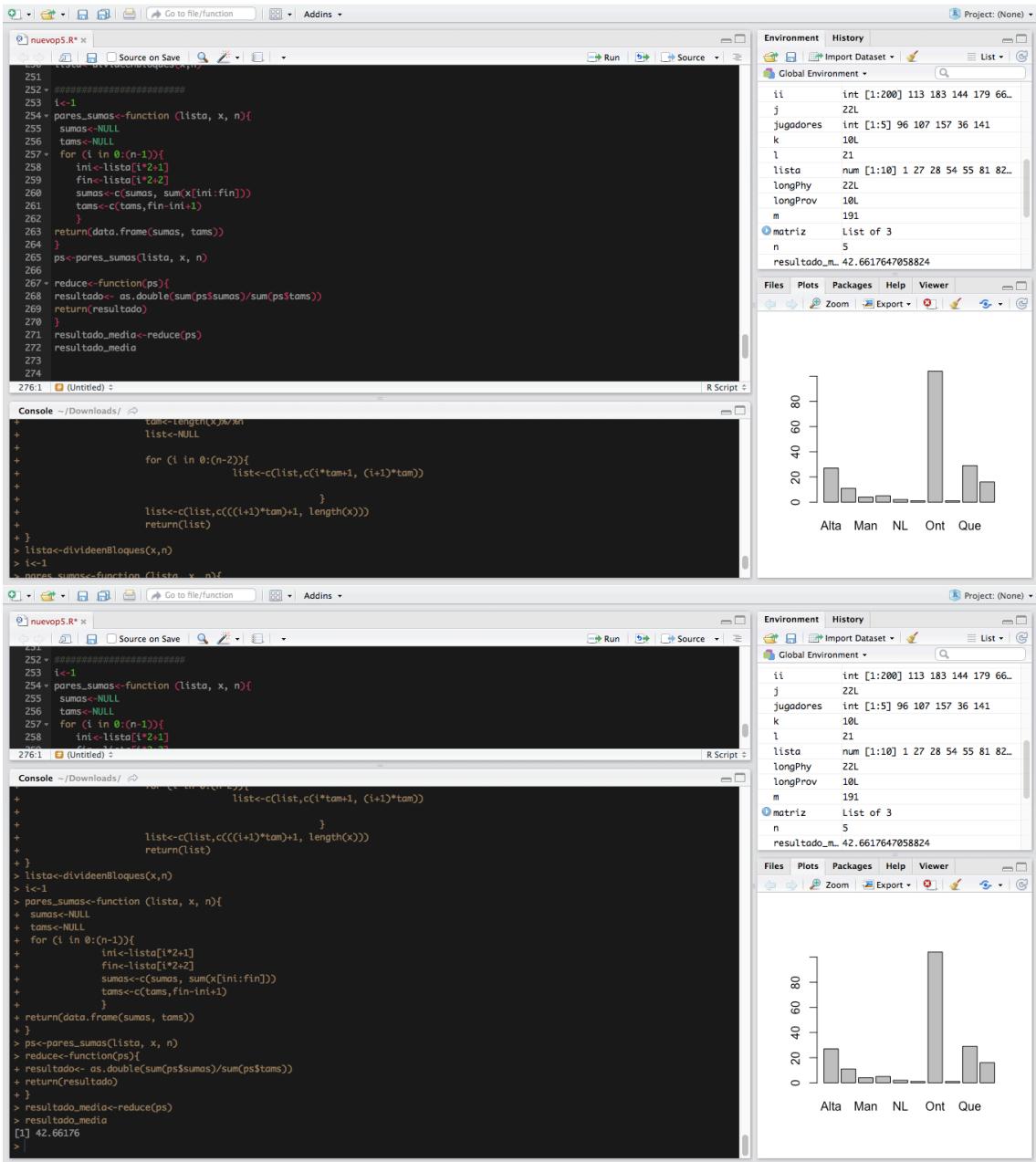




EJERCICIO 5:

Implementar una función que reciba como entrada un dataset de números y un número natural. La función debe devolver la media de los números almacenados en el dataset.

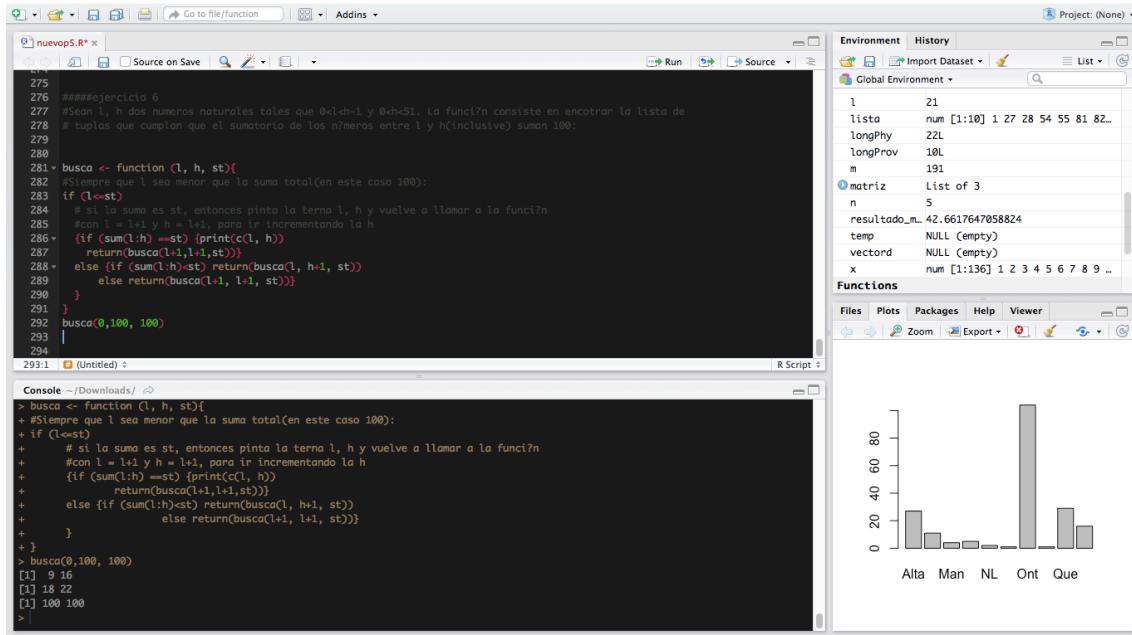




Finalmente obtenemos la media de los datos almacenados en nuestro data set (creado esta vez aleatoriamente).

EJERCICIO 6:

Observad este código Haskell: > $[(l,h) | h <-[0..51], l <-[0..h-1], \text{sum}[l..h]==100]$ Intentar adivinar cuál es el propósito del código y qué resultado obtiene. Observar esta solución recursiva en R y decidir cuál es el propósito del código: busca <- function (l, h, st){ if ($l \leq st$) {if ($\text{sum}(l:h) == st$) {print(c(l, h)) return(busca(l+1,l+1,st))} else {if ($\text{sum}(l:h)$



Interpretación:

Se trata de una función condicionada a que l y h son dos números naturales tales que h se encuentra entre 0 y 51 y l entre 0 y h-1. La condición que satisface la función del enunciado es que todos los números comprendidos entre l y h ambos inclusive tienen que sumar 100. La función que ejecutamos buscar los números tales que cumplan la anterior afirmación.

La condición “if” (el segundo if) indica que si la suma es st entonces se pintar la terna l, h y se vuelve a llamar a la función. Con l=l+1 y h = l+1 para poder buscar otras tuplas que cumplan tal condición empezando a comprobar por la siguiente posición de l y siendo h =l+1 para poder ir incrementando h en las siguientes vueltas.

La condición “else” (el primer else) se ejecuta cuando la suma l y h sea menor (tercer if) que st, entonces se sigue incrementando el valor de h volviendo a llamar a la función. En caso contrario (segundo else) y la suma l y h sea mayor que st es que no se ha encontrado una tupla que cumpla la condición sum(l:h)=st, en este caso de vuelve a llamar a la función para buscar una nueva tupla.

