Práctica 3: Listas y data frames

A continuación proponemos distintas tareas relacionadas con la creación y manipulación de listas y data frames. Crea un fichero script con el código que permita resolverlas, incluyendo en el mismo los comentarios que estimes oportunos. Este script deberás enviarlo a través de PRADO siguiendo las instrucciones proporcionadas en la tarea allí creada.

1. Crea un objeto de tipo lista con estas tres componentes: $x1 = (1, 2, 3, 4, 5)$, $x2 = (2, 3, 4, 5, 6)$ y $x3 = (3, 4, 5, 6, 7)$. A partir de ella resuelve las siguientes tareas:

   a) Crea un vector `x` con una muestra de 10 números aleatorios de una distribución uniforme en el intervalo (0,1). Añade dicho vector como una nueva componente a la lista anterior.

   b) Crea un vector `y` con una muestra de 10 números aleatorios de una distribución normal estándar. Añade dicho vector como una nueva componente a la lista anterior.

   c) Utiliza la función `lapply` para calcular la suma de cada componente de la lista. Observa qué tipo de objeto devuelve. Después prueba con la variante `sapply`, ¿qué diferencia observas entre las dos funciones?

   d) Escribe el siguiente código:

   `reg<-lm(y~x)`

   [1] y utiliza una función adecuada para confirmar que `reg` es un objeto de tipo lista.

   e) Utiliza una función adecuada para obtener qué tipo de objetos constituyen las componentes de `reg`.

   f) Crea una matriz que contenga por columnas las componentes `residuals` y `fitted.values` del objeto `reg`, además de los vectores `x` e `y`. Añade nombres a las columnas de dicha matriz.

```
> lista<-list(x1=1:5,x2=2:6,x3=3:7)
> x<-runif(10)
> lista$x<-x
> y<-rnorm(10)
> lista$y<-y
> lapply(lista,sum)
```

[1] La función `lm` permite estimar un modelo de regresión lineal. En este caso sería de regresión lineal simple ($y_i = \beta_0 + \beta_1 x_i + \epsilon_i$, $i = 1, \ldots, n$) con los elementos de `x` como observaciones de la variable explicativa, y los de `y` como observaciones de la variables de respuesta.

```
$x1
[1] 15

$x2
[1] 20

$x3
[1] 25

$x
[1] 5.518912

$y
[1] 0.9460225
```

```
> sapply(lista,sum)

        x1         x2         x3          x          y
15.0000000 20.0000000 25.0000000  5.5189122  0.9460225
```

```
> reg<-lm(y~x)
> is.list(reg) # typeof(reg)

[1] TRUE
```

```
> lapply(reg,class)

$coefficients
[1] "numeric"

$residuals
[1] "numeric"

$effects
[1] "numeric"

$rank
[1] "integer"

$fitted.values
[1] "numeric"
```

```
$assign
[1] "integer"

$qr
[1] "qr"

$df.residual
[1] "integer"

$xlevels
[1] "list"

$call
[1] "call"

$terms
[1] "terms"   "formula"

$model
[1] "data.frame"
```

```
> matriz<-cbind(reg$residuals,reg$fitted.values,x,y)
> colnames(matriz)<-c('residuals','fitted','x','y')
```

2. A veces los datos que tenemos para un análisis estadístico corresponden a datos agregados en forma de tabla de frecuencias. Crea un data frame con nombre `datos` con los datos que aparecen a continuación:

x_i	y_i	n_i
1.2	15	12
1.8	18	23
2.2	10	5
2.5	12	9
1.1	16	11

Se trata de datos agregados donde las dos primeras columnas corresponden a los valores que se observan en una muestra de dos variables estadísticas, (x_i, y_i), y la última columna contiene la frecuencia absoluta (n_i), esto es, el número de veces que se observa el par (x_i, y_i). De este modo el tamaño de la muestra (n) es la suma de dichas frecuencias absolutas. A partir de dicho data frame realiza las siguientes tareas:

a) Calcula el tamaño de la muestra.

b) Calcula las media aritméticas de las observaciones de las variables $\bar{x}$ e $\bar{y}$, así como las cuasivarianzas, s_x^2 y s_y^2.

c) Crea un segundo data frame con nombre `datos.n` que recoja las n observaciones individuales por filas, esto es, repitiendo las filas de `datos` tantas veces como indique la columna de la frecuencia absoluta.

d) A partir del data frame `datos.n` calcula de nuevo las medias aritméticas y las cuasivarianzas (usando `mean` y `var`, respectivamente) y comprueba el resultado anterior con los datos agregados.

e) La tipificación de los datos es una práctica habitual y requerida en algunas técnicas estadísticas. Consiste en una transformación del tipo $z_i = (x_i - \bar{x})/s_x$, de modo que la media de los z_i es 0 y su cuasi-varianza es 1. Añadir dos columnas al final del data frame `datos.n` con los valores tipificados de las variables x e y. Realiza esta tarea de dos formas, primero utilizando la función `transform` y luego utilizando `within`.

```
> xi<-c(1.2,1.8,2.2,2.5,1.1)
> yi<-c(15,18,10,12,6)
> ni<-c(12,23,5,9,11)
> datos<-data.frame(xi,yi,ni)
> n<-sum(ni) ; n

[1] 60

> mx<-sum(ni*xi)/n ; mx

[1] 1.69

> my<-sum(ni*yi)/n ; my

[1] 13.63333

> s2x<-sum(ni*(xi-mx)^2)/(n-1) ; s2x

[1] 0.2405763

> s2y<-sum(ni*(yi-my)^2)/(n-1) ; s2y

[1] 20.20226
```

```
> datos.n<-datos[rep(1:nrow(datos), datos$ni),1:2]
> mx<-mean(datos.n$xi); mx

[1] 1.69

> my<-mean(datos.n$yi) ; my

[1] 13.63333

> s2x<-var(datos.n$xi) ; s2x

[1] 0.2405763

> s2y<-var(datos.n$yi) ; s2y

[1] 20.20226

> datos.n1<-transform(datos.n,zx=(xi-mx)/sqrt(s2x),zy=(yi-my)/sqrt(s2y))
> head(datos.n1)

     xi yi         zx        zy
1   1.2 15 -0.9990097 0.3040623
1.1 1.2 15 -0.9990097 0.3040623
1.2 1.2 15 -0.9990097 0.3040623
1.3 1.2 15 -0.9990097 0.3040623
1.4 1.2 15 -0.9990097 0.3040623
1.5 1.2 15 -0.9990097 0.3040623

> datos.n2<-within(datos.n,{zx<-(xi-mx)/sqrt(s2x);zy<-(yi-my)/sqrt(s2y)})
> head(datos.n2)

     xi yi        zy         zx
1   1.2 15 0.3040623 -0.9990097
1.1 1.2 15 0.3040623 -0.9990097
1.2 1.2 15 0.3040623 -0.9990097
1.3 1.2 15 0.3040623 -0.9990097
1.4 1.2 15 0.3040623 -0.9990097
1.5 1.2 15 0.3040623 -0.9990097
```

3. En este ejercicio vamos a realizar varias manipulaciones sobre el data frame `ChickWeight`

del paquete *datasets*. Comienza escribiendo `help(ChickWeight)` y descubre el tipo de datos que contiene el data frame. Después resuelve las siguientes tareas:

a) Imprime en la venta de la consola las primeras 5 filas del data frame `ChickWeight` y las 3 últimas, utilizando para ello las funciones `head` y `tail`, respectivamente.

b) Imprime la estructura del objeto `ChickWeight`.

c) Realiza un resumen descriptivo numérico elemental de todas las variables del data frame con `summary`.

d) Realiza el mismo tipo de resumen pero ahora solo de la variable `weight` para los distintos niveles del factor dieta, usando la función `tapply`. Almacena el resultado en un objeto con nombre `peso.dieta`. ¿Qué tipo de objeto es `peso.dieta`?[2]

e) Crea un data frame (`peso.dieta.2`) colocando por columnas el resumen obtenido del peso para cada tipo de dieta. Cada columna tendrá como nombre el de la correspondiente medida descriptiva ("Min.", "1st Qu.", etc.).

f) La función `aggregate` permite resumir columnas de un data frame para cada uno de los niveles de un factor[3]. Utiliza esta función para realizar el mismo resumen que realizaste antes en el objeto `peso.dieta`. ¿Qué tipo de objeto devuelve esta función? Vuelve a crear el data frame `peso.dieta.2` con la estructura especificada antes a partir del objeto que devuelve `aggregate`.

g) Crea un data frame (`Chick100`) con una submuestra de los datos contenidos en `ChickWeight` seleccionando aleatoriamente (sin reemplazo) 100 filas[4].

h) Muestra el data frame `Chick100` con sus columnas permutadas aleatoriamente[5].

i) Muestra el data frame `Chick100` con sus columnas por orden alfabético.

j) Muestra los datos del data frame `Chick100` ordenados según la variable `Diet` (orden ascendente). Observa que cómo trata R los empates en dicha ordenación. Repite la operación rompiendo los empates de acuerdo al valor en la variable `Weight`.

k) Extrae del data frame `Chick100` una submuestra conteniendo solo una observación para cada tipo de dieta (variable `Diet`), en concreto la que corresponda al mayor valor de la variable `weight`. [Sugerencia: ordena las filas del data frame según `weight` en orden descendente, después puedes usar la función

[2]Si inspeccionas la ayuda de la función `tapply` entenderás mejor el resultado.

[3]Por ejemplo supongamos un data frame `df` con varias columnas donde la segunda de ellas es un factor, si queremos calcular las medias de la primera columna para los distintos niveles del factor entonces podríamos escribir `aggregate(df[,1],by=list(df[,2]),mean)`

[4]Seleccionar aleatoriamente observaciones de una muestra forma parte de técnicas estadísticas y del Machine Learning como el bootstrap y validación cruzada (*cross-validation*).

[5]Esta operación es necesaria en técnicas estadísticas relacionadas con big data y Machine Learning como *random forest*.

`duplicated`[6] aplicada a columna `Diet` para quedarse solo con la primera observación correspondiente a cada tipo de dieta.]

```
> head(ChickWeight)

  weight Time Chick Diet
1     42    0     1    1
2     51    2     1    1
3     59    4     1    1
4     64    6     1    1
5     76    8     1    1
6     93   10     1    1

> tail(ChickWeight)

    weight Time Chick Diet
573    155   12    50    4
574    175   14    50    4
575    205   16    50    4
576    234   18    50    4
577    264   20    50    4
578    264   21    50    4

> str(ChickWeight)

Classes 'nfnGroupedData', 'nfGroupedData', 'groupedData' and 'data.frame':
578 obs. of  4 variables:
 $ weight: num  42 51 59 64 76 93 106 125 149 171 ...
 $ Time  : num  0 2 4 6 8 10 12 14 16 18 ...
 $ Chick : Ord.factor w/ 50 levels "18"<"16"<"15"<..: 15 15 15 15 15 15 15 15 15 15
 $ Diet  : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 1 1 1 1 1 ...
 - attr(*, "formula")=Class 'formula'  language weight ~ Time | Chick
  .. ..- attr(*, ".Environment")=<environment: R_EmptyEnv>
 - attr(*, "outer")=Class 'formula'  language ~Diet
  .. ..- attr(*, ".Environment")=<environment: R_EmptyEnv>
 - attr(*, "labels")=List of 2
  ..$ x: chr "Time"
  ..$ y: chr "Body weight"
```

[6]Esta función aplicada a un vector devuelve (`duplicated(v)`) un vector lógico (del mismo tamaño del original, `v`) indicando con `TRUE` las posiciones del vector que contienen el mismo valor. De este modo `!duplicated(v)` se puede usar como filtro para seleccionar las filas del data frame en este ejercicio eliminando las duplicaciones.

```
 - attr(*, "units")=List of 2
  ..$ x: chr "(days)"
  ..$ y: chr "(gm)"

> summary(ChickWeight)

     weight           Time           Chick     Diet
 Min.   : 35.0   Min.   : 0.00   13     : 12   1:220
 1st Qu.: 63.0   1st Qu.: 4.00   9      : 12   2:120
 Median :103.0   Median :10.00   20     : 12   3:120
 Mean   :121.8   Mean   :10.72   10     : 12   4:118
 3rd Qu.:163.8   3rd Qu.:16.00   17     : 12
 Max.   :373.0   Max.   :21.00   19     : 12
                                 (Other):506

> peso.dieta<-tapply(ChickWeight$weight,ChickWeight$Diet,summary)
> class(peso.dieta);mode(peso.dieta);is.list(peso.dieta)

[1] "array"
[1] "list"
[1] TRUE

> peso.dieta.2<-data.frame(matrix(unlist(peso.dieta),
+                     nrow=length(peso.dieta),byrow=TRUE))
> colnames(peso.dieta.2)<-names(peso.dieta[[1]])
> peso.dieta.2

  Min. 1st Qu. Median     Mean 3rd Qu. Max.
1   35   57.75   88.0 102.6455  136.50  305
2   39   65.50  104.5 122.6167  163.00  331
3   39   67.50  125.5 142.9500  198.75  373
4   39   71.25  129.5 135.2627  184.75  322

> peso.dieta.2<-aggregate(ChickWeight$weight,
+                     by=list(ChickWeight$Diet),summary)
> peso.dieta.2

  Group.1   x.Min. x.1st Qu. x.Median   x.Mean x.3rd Qu.   x.Max.
1       1  35.0000   57.7500  88.0000 102.6455  136.5000 305.0000
2       2  39.0000   65.5000 104.5000 122.6167  163.0000 331.0000
3       3  39.0000   67.5000 125.5000 142.9500  198.7500 373.0000
4       4  39.0000   71.2500 129.5000 135.2627  184.7500 322.0000
```

```
> class(peso.dieta.2)

[1] "data.frame"

> Chick100<-ChickWeight[sample(1:nrow(ChickWeight),100),]
> # columnas permutadas aleatoriamente
> p<-ncol(Chick100)
> Chick100[,sample(1:p,p,replace=FALSE)]

    Diet Chick Time weight
395    3    35   12    201
550    4    48   14    170
177    1    16    2     45
220    1    20   21    117
163    1    14   14    192
149    1    13   10     67
224    2    21    6     86
41     1     4    8     74
288    2    26   14    147
141    1    12   18    185
325    2    29   16    187
457    3    40   16    215
353    3    32    0     41
231    2    21   20    318
380    3    34    6     85
549    4    48   12    154
211    1    20    4     54
36     1     3   21    202
422    3    37   18    157
501    4    44    8    103
123    1    11    6     84
357    3    32    8    107
65     1     6    8     97
468    4    41   14    153
330    2    30    2     48
83     1     7   20    288
137    1    12   10     88
153    1    13   18     81
382    3    34   10    134
439    3    39    4     61
227    2    21   12    217
167    1    14   21    266
```

```
373    3    33   16    151
118    1    10   20    120
115    1    10   14     96
510    4    45    6     78
165    1    14   18    248
426    3    38    2     49
396    3    35   14    238
57     1     5   16    197
513    4    45   12    135
188    1    17   10     89
219    1    20   20    115
389    3    35    0     41
78     1     7   10    112
484    4    42   21    281
261    2    24    8     66
362    3    32   18    263
239    2    22   12    108
86     1     8    2     50
367    3    33    4     63
511    4    45    8     98
393    3    35    8    123
462    4    41    2     51
242    2    22   18    148
256    2    23   21    175
272    2    25    6     78
489    4    43    8    131
376    3    33   21    147
265    2    24   16     72
150    1    13   12     71
160    1    14    8    101
106    1     9   20    100
568    4    50    2     54
281    2    26    0     42
432    3    38   14    154
293    2    27    0     39
342    3    31    2     53
438    3    39    2     50
418    3    37   10     83
524    4    46   10    120
443    3    39   12    130
303    2    27   20    185
517    4    45   20    197
```

```
332    2    30    6     72
556    4    49    2     53
108    1    10    0     41
61     1     6    0     41
413    3    37    0     41
213    1    20    8     65
234    2    22    2     55
386    3    34   18    294
315    2    28   20    212
262    2    24   10     68
394    3    35   10    158
307    2    28    4     58
421    3    37   16    135
571    4    50    8    105
198    1    19    2     48
111    1    10    6     63
226    2    21   10    163
14     1     2    2     49
112    1    10    8     74
190    1    17   14    103
547    4    48    8    104
494    4    43   18    198
28     1     3    6     67
2      1     1    2     51
578    4    50   21    264
175    1    15   14     68

> # columnas por orden alfabético
> Chick100[1,order(names(Chick100))]

    Chick Diet Time weight
395    35    3   12    201

> # ordenacion segun diet
> Chick100[order(Chick100$Diet),]

    weight Time Chick Diet
177     45    2    16    1
220    117   21    20    1
163    192   14    14    1
149     67   10    13    1
41      74    8     4    1
```

```
141    185   18    12    1
211     54    4    20    1
36     202   21     3    1
123     84    6    11    1
65      97    8     6    1
83     288   20     7    1
137     88   10    12    1
153     81   18    13    1
167    266   21    14    1
118    120   20    10    1
115     96   14    10    1
165    248   18    14    1
57     197   16     5    1
188     89   10    17    1
219    115   20    20    1
78     112   10     7    1
86      50    2     8    1
150     71   12    13    1
160    101    8    14    1
106    100   20     9    1
108     41    0    10    1
61      41    0     6    1
213     65    8    20    1
198     48    2    19    1
111     63    6    10    1
14      49    2     2    1
112     74    8    10    1
190    103   14    17    1
28      67    6     3    1
2       51    2     1    1
175     68   14    15    1
224     86    6    21    2
288    147   14    26    2
325    187   16    29    2
231    318   20    21    2
330     48    2    30    2
227    217   12    21    2
261     66    8    24    2
239    108   12    22    2
242    148   18    22    2
256    175   21    23    2
272     78    6    25    2
```

```
265     72   16    24    2
281     42    0    26    2
293     39    0    27    2
303    185   20    27    2
332     72    6    30    2
234     55    2    22    2
315    212   20    28    2
262     68   10    24    2
307     58    4    28    2
226    163   10    21    2
395    201   12    35    3
457    215   16    40    3
353     41    0    32    3
380     85    6    34    3
422    157   18    37    3
357    107    8    32    3
382    134   10    34    3
439     61    4    39    3
373    151   16    33    3
426     49    2    38    3
396    238   14    35    3
389     41    0    35    3
362    263   18    32    3
367     63    4    33    3
393    123    8    35    3
376    147   21    33    3
432    154   14    38    3
342     53    2    31    3
438     50    2    39    3
418     83   10    37    3
443    130   12    39    3
413     41    0    37    3
386    294   18    34    3
394    158   10    35    3
421    135   16    37    3
550    170   14    48    4
549    154   12    48    4
501    103    8    44    4
468    153   14    41    4
510     78    6    45    4
513    135   12    45    4
484    281   21    42    4
```

```
511     98    8    45    4
462     51    2    41    4
489    131    8    43    4
568     54    2    50    4
524    120   10    46    4
517    197   20    45    4
556     53    2    49    4
571    105    8    50    4
547    104    8    48    4
494    198   18    43    4
578    264   21    50    4
```

```
> # ordenacion segun diet y weight
> Chick100[order(Chick100$Diet,Chick100$weight),]
```

```
    weight Time Chick Diet
108     41    0    10    1
61      41    0     6    1
177     45    2    16    1
198     48    2    19    1
14      49    2     2    1
86      50    2     8    1
2       51    2     1    1
211     54    4    20    1
111     63    6    10    1
213     65    8    20    1
149     67   10    13    1
28      67    6     3    1
175     68   14    15    1
150     71   12    13    1
41      74    8     4    1
112     74    8    10    1
153     81   18    13    1
123     84    6    11    1
137     88   10    12    1
188     89   10    17    1
115     96   14    10    1
65      97    8     6    1
106    100   20     9    1
160    101    8    14    1
190    103   14    17    1
78     112   10     7    1
```

```
219    115   20    20    1
220    117   21    20    1
118    120   20    10    1
141    185   18    12    1
163    192   14    14    1
57     197   16     5    1
36     202   21     3    1
165    248   18    14    1
167    266   21    14    1
83     288   20     7    1
293     39    0    27    2
281     42    0    26    2
330     48    2    30    2
234     55    2    22    2
307     58    4    28    2
261     66    8    24    2
262     68   10    24    2
265     72   16    24    2
332     72    6    30    2
272     78    6    25    2
224     86    6    21    2
239    108   12    22    2
288    147   14    26    2
242    148   18    22    2
226    163   10    21    2
256    175   21    23    2
303    185   20    27    2
325    187   16    29    2
315    212   20    28    2
227    217   12    21    2
231    318   20    21    2
353     41    0    32    3
389     41    0    35    3
413     41    0    37    3
426     49    2    38    3
438     50    2    39    3
342     53    2    31    3
439     61    4    39    3
367     63    4    33    3
418     83   10    37    3
380     85    6    34    3
357    107    8    32    3
```

```
393    123    8    35    3
443    130   12    39    3
382    134   10    34    3
421    135   16    37    3
376    147   21    33    3
373    151   16    33    3
432    154   14    38    3
422    157   18    37    3
394    158   10    35    3
395    201   12    35    3
457    215   16    40    3
396    238   14    35    3
362    263   18    32    3
386    294   18    34    3
462     51    2    41    4
556     53    2    49    4
568     54    2    50    4
510     78    6    45    4
511     98    8    45    4
501    103    8    44    4
547    104    8    48    4
571    105    8    50    4
524    120   10    46    4
489    131    8    43    4
513    135   12    45    4
468    153   14    41    4
549    154   12    48    4
550    170   14    48    4
517    197   20    45    4
494    198   18    43    4
578    264   21    50    4
484    281   21    42    4

> # submuestra conteniendo solo una observación para cada tipo de dieta
> new<-Chick100[rev(order(Chick100$weight)),]
> new[!duplicated(new$Diet),]

    weight Time Chick Diet
231    318   20    21    2
386    294   18    34    3
83     288   20     7    1
484    281   21    42    4
```