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] 5.866516

$y
[1] -3.240906

> sapply(lista,sum)

       x1        x2        x3         x         y
15.000000 20.000000 25.000000  5.866516 -3.240906

> 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=
> 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)]

    weight Chick Diet Time
228    240    21    2   14
102     90     9    1   12
520     52    46    4    2
296     73    27    2    6
289    169    26    2   16
528    210    46    4   18
212     58    20    1    6
495    199    43    4   20
401     39    36    3    0
405     98    36    3    8
507     41    45    4    0
14      49     2    1    2
136     72    12    1    8
559    108    49    4    8
38      49     4    1    2
99      68     9    1    6
263     70    24    2   12
504    138    44    4   14
239    108    22    2   12
566    237    49    4   21
89      84     8    1    8
533     66    47    4    4
186     72    17    1    6
531     41    47    4    0
217     98    20    1   16
324    150    29    2   14
274    124    25    2   10
312    156    28    2   14
258     52    24    2    2
146     53    13    1    4
294     46    27    2    2
163    192    14    1   14
```

```
72     157     6    1   21
4       64     1    1    6
185     61    17    1    4
503    127    44    4   12
234     55    22    2    2
535    100    47    4    8
335    115    30    2   12
328    309    29    2   21
75      57     7    1    4
96      42     9    1    0
277    197    25    2   16
502    118    44    4   10
374    146    33    3   18
125    139    11    1   10
530    238    46    4   21
567     41    50    4    0
91     110     8    1   12
250    103    23    2   10
253    145    23    2   16
95     125     8    1   20
7      106     1    1   12
433    192    38    3   16
67     141     6    1   12
466    124    41    4   10
314    207    28    2   18
31     115     3    1   12
113     81    10    1   10
413     41    37    3    0
390     53    35    3    2
363    291    32    3   20
441     89    39    3    8
140    162    12    1   16
292    251    26    2   21
336    122    30    2   14
332     72    30    2    6
35     198     3    1   20
108     41    10    1    0
311    145    28    2   12
207    144    19    1   20
339    157    30    2   20
48     157     4    1   21
544     50    48    4    2
```

```
181     51    16    1   10
367     63    33    3    4
344     73    31    3    6
382    134    34    3   10
512    117    45    4   10
357    107    32    3    8
524    120    46    4   10
573    155    50    4   12
59     220     5    1   20
159     79    14    1    6
65      97     6    1    8
506    146    44    4   18
134     56    12    1    4
242    148    22    2   18
577    264    50    4   20
177     45    16    1    2
15      58     2    1    4
526    156    46    4   14
516    174    45    4   18
254    163    23    2   18
554    322    48    4   21
143    205    12    1   21
571    105    50    4    8
345     85    31    3    8
572    122    50    4   10
431    128    38    3   12
```

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

    Chick Diet Time weight
228    21    2   14    240
```

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

    weight Time Chick Diet
102     90   12     9    1
212     58    6    20    1
14      49    2     2    1
136     72    8    12    1
38      49    2     4    1
```

```
99      68    6     9    1
89      84    8     8    1
186     72    6    17    1
217     98   16    20    1
146     53    4    13    1
163    192   14    14    1
72     157   21     6    1
4       64    6     1    1
185     61    4    17    1
75      57    4     7    1
96      42    0     9    1
125    139   10    11    1
91     110   12     8    1
95     125   20     8    1
7      106   12     1    1
67     141   12     6    1
31     115   12     3    1
113     81   10    10    1
140    162   16    12    1
35     198   20     3    1
108     41    0    10    1
207    144   20    19    1
48     157   21     4    1
181     51   10    16    1
59     220   20     5    1
159     79    6    14    1
65      97    8     6    1
134     56    4    12    1
177     45    2    16    1
15      58    4     2    1
143    205   21    12    1
228    240   14    21    2
296     73    6    27    2
289    169   16    26    2
263     70   12    24    2
239    108   12    22    2
324    150   14    29    2
274    124   10    25    2
312    156   14    28    2
258     52    2    24    2
294     46    2    27    2
234     55    2    22    2
```

```
335    115   12    30    2
328    309   21    29    2
277    197   16    25    2
250    103   10    23    2
253    145   16    23    2
314    207   18    28    2
292    251   21    26    2
336    122   14    30    2
332     72    6    30    2
311    145   12    28    2
339    157   20    30    2
242    148   18    22    2
254    163   18    23    2
401     39    0    36    3
405     98    8    36    3
374    146   18    33    3
433    192   16    38    3
413     41    0    37    3
390     53    2    35    3
363    291   20    32    3
441     89    8    39    3
367     63    4    33    3
344     73    6    31    3
382    134   10    34    3
357    107    8    32    3
345     85    8    31    3
431    128   12    38    3
520     52    2    46    4
528    210   18    46    4
495    199   20    43    4
507     41    0    45    4
559    108    8    49    4
504    138   14    44    4
566    237   21    49    4
533     66    4    47    4
531     41    0    47    4
503    127   12    44    4
535    100    8    47    4
502    118   10    44    4
530    238   21    46    4
567     41    0    50    4
466    124   10    41    4
```

```
544     50    2    48    4
512    117   10    45    4
524    120   10    46    4
573    155   12    50    4
506    146   18    44    4
577    264   20    50    4
526    156   14    46    4
516    174   18    45    4
554    322   21    48    4
571    105    8    50    4
572    122   10    50    4
```

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

```
    weight Time Chick Diet
108     41    0    10    1
96      42    0     9    1
177     45    2    16    1
14      49    2     2    1
38      49    2     4    1
181     51   10    16    1
146     53    4    13    1
134     56    4    12    1
75      57    4     7    1
212     58    6    20    1
15      58    4     2    1
185     61    4    17    1
4       64    6     1    1
99      68    6     9    1
136     72    8    12    1
186     72    6    17    1
159     79    6    14    1
113     81   10    10    1
89      84    8     8    1
102     90   12     9    1
65      97    8     6    1
217     98   16    20    1
7      106   12     1    1
91     110   12     8    1
31     115   12     3    1
95     125   20     8    1
```

```
125    139   10    11    1
67     141   12     6    1
207    144   20    19    1
72     157   21     6    1
48     157   21     4    1
140    162   16    12    1
163    192   14    14    1
35     198   20     3    1
143    205   21    12    1
59     220   20     5    1
294     46    2    27    2
258     52    2    24    2
234     55    2    22    2
263     70   12    24    2
332     72    6    30    2
296     73    6    27    2
250    103   10    23    2
239    108   12    22    2
335    115   12    30    2
336    122   14    30    2
274    124   10    25    2
253    145   16    23    2
311    145   12    28    2
242    148   18    22    2
324    150   14    29    2
312    156   14    28    2
339    157   20    30    2
254    163   18    23    2
289    169   16    26    2
277    197   16    25    2
314    207   18    28    2
228    240   14    21    2
292    251   21    26    2
328    309   21    29    2
401     39    0    36    3
413     41    0    37    3
390     53    2    35    3
367     63    4    33    3
344     73    6    31    3
345     85    8    31    3
441     89    8    39    3
405     98    8    36    3
```

```
357    107    8    32    3
431    128   12    38    3
382    134   10    34    3
374    146   18    33    3
433    192   16    38    3
363    291   20    32    3
507     41    0    45    4
531     41    0    47    4
567     41    0    50    4
544     50    2    48    4
520     52    2    46    4
533     66    4    47    4
535    100    8    47    4
571    105    8    50    4
559    108    8    49    4
512    117   10    45    4
502    118   10    44    4
524    120   10    46    4
572    122   10    50    4
466    124   10    41    4
503    127   12    44    4
504    138   14    44    4
506    146   18    44    4
573    155   12    50    4
526    156   14    46    4
516    174   18    45    4
495    199   20    43    4
528    210   18    46    4
566    237   21    49    4
530    238   21    46    4
577    264   20    50    4
554    322   21    48    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
554    322   21    48    4
328    309   21    29    2
363    291   20    32    3
59     220   20     5    1
```