Práctica 2. Generación de nuevas variables con R. Ordenación de casos.

Christian J. Acal González y Miguel Ángel Montero Alonso

UNIVERSIDAD DE GRANADA

Práctica 2. Generación de nuevas variables con R. Ordenación de casos.

Christian J. Acal González y Miguel Ángel Montero Alonso

2.1 Generación de nuevas variables

En un fichero de datos podemos llevar a cabo la tarea de generar nuevas variables a partir de las ya existentes. Las nuevas variables pueden ser generadas como resultado de la aplicación de una fórmula en la que interviene, generalmente, una o varias de las variables existentes en dicho fichero (tal y como se vio en el pasado tema) o como resultado de una transformación (recodificación) de los códigos o valores de una de las variables de esa matriz de datos.

En lo que sigue, se toma como referencia el fichero de datos **osteo.sav**.

```
library(foreign)
```

```
## Warning: package 'foreign' was built under R version 4.1.2
```

```
osteo=read.spss("osteo.sav",to.data.frame = TRUE)
```

```
## re-encoding from CP1252
```

```
colnames(osteo) #Muestra todas las variables del fichero
```

```
##  [1] "num"        "edad"       "grupo_edad" "sexo"       "peso"
##  [6] "talla"      "imc"        "tevol"      "tabaco"     "alcohol"
## [11] "ingca"      "acfis"      "retin"      "nefro"      "neuro"
## [16] "hba1c"      "ca"         "p"          "cr"         "pthm"
## [21] "pthi"       "bmdcue"     "szl24"      "sztri"      "szcue"
## [26] "osteo_cue"  "osteo_tri"
```

2.1.1 Cálculo de nuevas variables

Con el fin de recordar el proceso de cálculo de nuevas variables, se procede a determinar la variable *imc*. Esta variable ya se encuentra en el fichero, por lo que servirá de referencia para comprobar que se han hecho bien todos los cálculos. El *imc* es el resultado de dividir el peso entre la altura (expresada en metros) al cuadrado. Por tanto, habrá que seleccionar ambas variables, pasar la talla a metros y realizar la división.

```
attach(osteo)
talla_metros=talla/100
imc_osteo=peso/(talla_metros^2)
```

A continuación, a modo de comprobación, se comparan los primeros seis valores que se acaban de obtener con los primeros seis valores de la variable *imc* que contiene la base de datos **osteo**.

```
head(imc) #Primeros 6 valores "imc" del fichero osteo
```

```
## [1] 23.60816 27.56524 21.10727 24.02381 18.06973 24.63548
```

```
head(imc_osteo) #Primeros 6 valores que se han calculado
```

```
## [1] 23.60816 27.56524 21.10727 24.02381 18.06973 24.63548
```

2.1.2 Recodificación de los valores de las variables

La recodificación tiene utilidad en las ocasiones en las que los datos de las variables cuentan con valores muy dispersos que dificultan su interpretación o sencillamente no se prestan para el análisis estadístico. La principal virtud de este procedimiento radica en la posibilidad de asignar números representativos de cada categoría de acuerdo a las necesidades del estudio, lo que nos permite agrupar valores que pueden no ser consecutivos. Un ejemplo muy simple para aclarar el propósito de la recodificación es el siguiente: supóngase que en la base de datos hay una variable que denota la ciudad del sujeto, pero podría ser interesante disponer de una variable que exprese la comunidad autónoma en la que reside el individuo, y así reducir el número de categorías. En este caso, todos aquellos individuos que sean de Almería, Jaén, Granada, Málaga, Córdoba, Cádiz, Sevilla o Huelva tomarán el valor *Andalucía* en la nueva variable que se genere.

El primer tipo de recodificación que se va a estudiar va a ser la recodificación entre variable factor a variable factor. La variable *alcohol* indica el nivel de consumo de alcohol en tres valores o códigos (1=No, 2=Moderado, 3=Excesivo), pero se quiere trabajar con una variable (por ejemplo, *alcohol.sn*) que exprese el consumo en dos categorías (0=No consume, 1=Sí consume). La idea es crear un vector vacío donde se va a recodificar los valores correspondientes a los códigos originales de la variable *alcohol*. Posteriormente, el nuevo vector será luego transformado en un factor y será añadido a la base de datos.

```
alcohol.sn=vector() #vector() crea un vector vacío
alcohol.sn[alcohol=="No"]=0 #Los que no beben se les asigna el valor 0
alcohol.sn[alcohol=="Moderado" | alcohol=="Excesivo"]=1 #1 a los que sí beben

alcohol.sn=factor(alcohol.sn,labels=c("No","Sí")) #Los 0 y 1 se transforman en "No" y "Si"
osteo$alcohol.sn=alcohol.sn #Se integra la variable recodificada en el dataframe
colnames(osteo) #Se aprecia que ya aparece la nueva variable en la base (al final)
```

```
##  [1] "num"        "edad"       "grupo_edad" "sexo"       "peso"
##  [6] "talla"      "imc"        "tevol"      "tabaco"     "alcohol"
## [11] "ingca"      "acfis"      "retin"      "nefro"      "neuro"
## [16] "hba1c"      "ca"         "p"          "cr"         "pthm"
## [21] "pthi"       "bmdcue"     "szl24"      "sztri"      "szcue"
## [26] "osteo_cue"  "osteo_tri"  "alcohol.sn"
```

Notar que en el *script* se ha utilizado el operador lógico '|' (se consigue con **Alt Gr+1** en el teclado) para indicar la condición de que todos los pacientes con valor 'moderado' **o** con valor 'excesivo' en la variable *alcohol*, se les asignara el valor 1 en la nueva variable *alcohol.sn*. Por tanto, el operador lógico | se traduce como '**o**'. Si en lugar de querer utilizar una función lógica que verifique una condición u otra como en este caso, se está interesado en emplear una función que cumpla dos condiciones al mismo tiempo, se usa el operador lógico **&** que hace el papel de '**y**'.

Se comprueba que la recodificación se ha realizado de manera satisfactoria (por economía del espacio, solo se muestra los primeros 20 individuos)

```
head(alcohol,20)
```

```
##  [1] Moderado Moderado Moderado Moderado Moderado Moderado Moderado Moderado
##  [9] Moderado Moderado Moderado Moderado Moderado Moderado Moderado Moderado
## [17] No       Moderado Moderado Moderado
## Levels: No Moderado Excesivo
```

```
head(alcohol.sn,20)
```

```
##  [1] Sí Sí Sí Sí Sí Sí Sí Sí Sí Sí Sí Sí Sí Sí Sí Sí No Sí Sí Sí
## Levels: No Sí
```

Otra posibilidad es cambiar el código numérico asignado a las modalidades de una variable de manera que se altere el orden. Por ejemplo, la variable *sexo* tiene como primer registro 'hombre' y después 'mujer'. Esto es fácil de comprobar, ya que basta con ejecutar `osteo$sexo` o, simplemente, `levels(osteo$sexo)` y apreciar el orden en el que aparecen los niveles del factor de la variable. Esto se debe a que cuando se introdujo los datos en el programa estadístico en el que se generó la base de datos, se asignó el valor 1 a 'hombre' y 2 a 'mujer'. Mediante la recodificación podemos cambiar dichos códigos, asignando 2 a 'hombre' y 1 a 'mujer', obteniendo así una nueva variable igual que la original pero alterado el orden de los niveles.

```
levels(sexo) #Para comprobar el orden de los factores de la variable
```

```
## [1] "Hombre" "Mujer"
```

```
sexo2=vector()  #Se crea un vector vacío
sexo2[sexo=="Hombre"]=2 #Asigna el valor 2 a hombres
sexo2[sexo!="Hombre"]=1 #Asigna el valor 1 a mujeres
sexo2=factor(sexo2,labels=c("Mujer","Hombre")) #Transforma el 1 en "Mujer" y 2 en "Hombre"
```

Para comprobar que se ha cambiado el orden, basta con ejecutar el siguiente código y ver el orden que marca la fila *'Levels'*.

```
head(sexo) #Se muestra los primeros 6 valores de la variable original y el orden
```

```
## [1] Hombre Hombre Hombre Hombre Mujer  Mujer
## Levels: Hombre Mujer
```

```
head(sexo2) #Se muestra los primeros 6 valores de la variable recodificada y el orden
```

```
## [1] Hombre Hombre Hombre Hombre Mujer  Mujer
## Levels: Mujer Hombre
```

El tercer caso que se va a considerar es el referido a la agrupación de valores numéricos. Cuando el número de valores distintos de una variable es elevado (por ejemplo, *edad*, *peso*,...) puede ser atrayente (según el análisis) dividir el rango total de los datos en intervalos facilitando el trato y el manejo de la variable. En estos casos lo que se tiene al final son datos agrupados en intervalos. A modo de ejemplo, se va a agrupar los datos referidos al peso en intervalos de 10 kg.

```
peso.agrupado=vector() #Se crea un vector vacío
peso.agrupado[peso<=50]=1
peso.agrupado[peso>50 & peso<=60]=2
peso.agrupado[peso>60 & peso<=70]=3
peso.agrupado[peso>70 & peso<=80]=4
peso.agrupado[peso>80 & peso<=90]=5
peso.agrupado[peso>90]=6

peso.agrupado=factor(peso.agrupado,labels=c("<=50","(50,60]","(60,70]","(70,80]",
                                            "(80,90]",">90"))

head(peso.agrupado)
```

```
## [1] (70,80] (80,90] (60,70] (60,70] (50,60] (60,70]
## Levels: <=50 (50,60] (60,70] (70,80] (80,90] >90
```

```
head(peso)
```

```
## [1] 72.3 82.5 61.0 67.0 51.0 61.5
```

2.2 Manejo de datos faltantes en R

Otro aspecto importante en lo que a la manipulación de datos en R se refiere, es el tema de **datos faltantes** o **valores perdidos**. En una investigación es muy habitual encontrase que no se sepa la respuesta de un individuo sobre una variable, especialmente en situaciones donde la información se recoge a través de un cuestionario en el que el encuestado no quiere o no sabe responder a una pregunta. Esta información desconocida es lo que se conoce en estadística como valores faltantes o valores perdidos. En la práctica anterior se adelantó que los datos faltantes en R son denotados como **NA** (ver definición de los factores). A modo de ilustración, se aprecia que la variable *pthi* presenta una serie de valores perdidos.

```
pthi
```

```
##  [1] 49.0   NA 54.3   NA 49.8 59.9 20.9   NA   NA 10.0   NA 80.0 43.0   NA   NA
## [16] 59.7   NA 58.7 22.1 45.0 24.7 38.9   NA 34.6   NA   NA   NA 66.7 56.1   NA
## [31]   NA   NA   NA 54.7   NA 44.4 87.2 46.2 58.3 68.8 77.2 53.1 53.0 55.6 71.3
## [46] 73.4 68.5 63.4 25.6 45.7 40.4 79.9 42.1 38.6 64.2 34.7   NA 82.4 59.6 66.1
## [61] 47.8   NA   NA 35.2 57.1 97.0 88.1 89.3 40.2 36.0 39.0 76.8 35.9 59.9 34.0
## [76] 63.5 42.6 46.8 91.2 69.6 37.3 62.9 31.8   NA 36.1   NA 53.6   NA   NA 56.8
## [91]   NA   NA   NA 57.0
```

Si se combinan la función **which()** que indica las posiciones de un objeto que cumplen una determinada condición y la función **is.na()** que señala qué elementos del vector son perdidos, se puede averiguar las posiciones del vector que presentan valores no conocidos. Las funciones que comienzan por **is.** permiten comprobar si un objeto considerado pertenece a un cierto tipo. Si devuelve TRUE (verdadero) se debe a que coincide con el tipo cuestionado; en caso contrario devolverá FALSE (falso). Las más usuales son *is.character, is.logical, is.numeric ,is.integer,is.double,is.vector, is.matrix,is.data.frame,is.list, is.factor* e *is.na*.

```
is.na(pthi)
```

```
##  [1] FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE  TRUE FALSE  TRUE FALSE
## [13] FALSE  TRUE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
## [25]  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE
## [37] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [49] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
## [61] FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [73] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
## [85] FALSE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE FALSE
```

```
which(is.na(pthi))  #Observaciones que son valores perdidos en pthi
```

```
##  [1]  2  4  8  9 11 14 15 17 23 25 26 27 30 31 32 33 35 57 62 63 84 86 88 89 91
## [26] 92 93
```

Esta falta de información a veces provoca que ciertas funciones no puedan ser ejecutadas a causa del mismo. En cambio, también existen funciones que ignoran internamente estos valores y realizan las operaciones sobre las observaciones que sí presentan un valor conocido. Para ilustrar este comentario, se va a proceder a calcular la media de la variable *pthi* mediante dos funciones distintas. La explicación de estas funciones se ignora puesto que serán introducidas en futuras prácticas.

```
mean(pthi)
```

```
## [1] NA
```

```
summary(pthi)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   10.00   39.60   54.30   53.93   65.15   97.00      27
```

```
detach(osteo) #Anula la funcion attach(osteo)
```

En las salidas anteriores, se puede apreciar como la primera función devuelve un valor **NA**, ya que no es capaz de calcular la media en presencia de valores perdidos. Por su parte, la segunda función sí permite el cálculo (entre otras) de la media, pero indica que hay 27 datos faltantes que han sido eliminados a la hora de determinar la media. En consecuencia, se han tenido en cuenta 67 (94 sujetos menos 27 valores perdidos) observaciones válidas para el cálculo de la media en la segunda función.

Debido a esta circunstancia, hay que tener sumo cuidado en el análisis de datos cuando se está trabajando en presencia de valores faltantes. Muchos investigadores que no tienen experiencia en el análisis de datos o que no son estadísticos, suelen suprimir estas observaciones para eliminar el problema. Esta opción es altamente peligrosa y bajo ningún concepto debería realizarse. La única posibilidad válida para subsanar el problema de la presencia de valores perdidos en el análisis de datos es mediante el proceso de imputación. Este proceso consiste en asignar un valor a cada dato faltante. Detrás de este campo, existe una teoría muy densa con distintas perspectivas que se escapan de los objetivos del curso. En lo que respecta al curso, el alumnado deberá utilizar las funciones que se irán comentando a lo largo de las explicaciones de los temas con el fin de evitar los problemas que puedan surgir.

2.3 Ordenar casos

Finalmente, se estudiará el procedimiento de **ordenar casos** con el fin de completar la formación sobre el manejo de una matriz de datos en R. Hasta ahora, se han estudiado otra serie de técnicas cómo seleccionar casos, dividir archivo, importar/exportar una base de datos, etc., pero aún no se ha tratado otra técnica realmente interesante en el estudio estadístico cómo es la de ordenar los casos (filas) según una o varias variables de la matriz de datos. En el primer tema, se detalló como se podían ordenar los valores de una variable mediante la función **sort()**. No obstante, esta acción no es útil cuando se está trabajando con una base de datos, ya que ordenar los valores de una variable con la función 'sort' no implica ordenar automáticamente los valores del resto de variables. Con el fin de ilustrar lo comentado, a continuación se crea una base de datos compuesta por tres variables: *edad*, *peso* y *altura*; y únicamente 7 individuos para facilitar el aprendizaje.

```
edad=c(26,32,29,26,27,25,27)
peso=c(64.75,55.25,61.20,58.75,63.30,59.80,51.30)
altura=c(1.80,1.65,1.71,1.62,1.71,1.83,1.63)
datos_original=data.frame(edad,peso,altura)
datos_original
```

```
##   edad  peso altura
## 1   26 64.75   1.80
## 2   32 55.25   1.65
## 3   29 61.20   1.71
## 4   26 58.75   1.62
## 5   27 63.30   1.71
## 6   25 59.80   1.83
## 7   27 51.30   1.63
```

Si se ordena la variable *edad* y se introduce en la base de datos (ver siguiente salida), se puede apreciar que los valores del resto de variables permanecen inamovibles, alterando así la información real de los individuos. Por ejemplo, el individuo más joven de la base de datos, el cual tiene 25 años, pesa 59.80 kg y mide 1.83m, ocupa la sexta posición del fichero de datos original. La información de este individuo (25-59.80-1.83) debería ocupar la primera posición del fichero después de la ordenación. Sin embargo, aunque en el nuevo fichero sí aparece la *edad* correctamente, en las variables *peso* y *altura* no se han modificado el orden de los valores con respecto a la ordenación de la *edad*.

```
attach(datos_original)
edad=sort(edad)
datos_ordenados=data.frame(edad,peso,altura)
datos_ordenados
```

```
##   edad  peso altura
## 1   25 64.75   1.80
## 2   26 55.25   1.65
## 3   26 61.20   1.71
## 4   27 58.75   1.62
## 5   27 63.30   1.71
## 6   29 59.80   1.83
## 7   32 51.30   1.63
```

Asimismo, si se intenta ordenar las otras variables mediante la misma técnica (ver siguiente salida), el fichero de datos resultante tendría todas las variables ordenadas de menor a mayor, pero no correspondería con la información real de los individuos. Se puede apreciar que la información de la primera fila es 25-51.30-1.62, que queda muy lejos de la información verdadera 25-59.80-1.83.

```
peso=sort(peso)
altura=sort(altura)
datos_ordenados=data.frame(edad,peso,altura)
datos_ordenados
```

```
##   edad  peso altura
## 1   25 51.30   1.62
## 2   26 55.25   1.63
## 3   26 58.75   1.65
## 4   27 59.80   1.71
## 5   27 61.20   1.71
## 6   29 63.30   1.80
## 7   32 64.75   1.83
```

Para subsanar este problema, hay que recurrir a la función **order()**. Esta función devuelve una permutación que reorganiza su primer argumento en orden ascendente (por defecto) o descendente, rompiendo los lazos con otros argumentos. Esta función se sitúa entre corchetes en la posición filas del fichero, es decir, `nombre.fichero[order(),]`. Seguidamente se muestra como ordenar los valores del fichero descrito anteriormente en función de la variable edad, tanto en forma ascendente como descendente.

```
#Se vuelven a introducir los datos originales para evitar problemas de solapamiento
edad=c(26,32,29,26,27,25,27)
peso=c(64.75,55.25,61.20,58.75,63.30,59.80,51.30)
altura=c(1.80,1.65,1.71,1.62,1.71,1.83,1.63)

datos_original=data.frame(edad,peso,altura)
datos_original
```

```
##   edad  peso altura
## 1   26 64.75   1.80
## 2   32 55.25   1.65
## 3   29 61.20   1.71
## 4   26 58.75   1.62
## 5   27 63.30   1.71
## 6   25 59.80   1.83
## 7   27 51.30   1.63
```

```
datos.ordenados_edad=datos_original[order(edad),]
datos.ordenados_edad
```

```
##   edad  peso altura
## 6   25 59.80   1.83
## 1   26 64.75   1.80
## 4   26 58.75   1.62
## 5   27 63.30   1.71
## 7   27 51.30   1.63
## 3   29 61.20   1.71
## 2   32 55.25   1.65
```

```
datos_original[order(edad,decreasing=TRUE),] #Para ordenar de mayor a menor
```

```
##   edad  peso altura
## 2   32 55.25   1.65
## 3   29 61.20   1.71
## 5   27 63.30   1.71
## 7   27 51.30   1.63
## 1   26 64.75   1.80
## 4   26 58.75   1.62
## 6   25 59.80   1.83
```

También es posible ordenar el fichero según los valores de dos o más variables. Dentro de la función 'order' se indicarán todas las variables separadas por coma y sin necesidad de usar la función *concatenar*. Con esto, R ordena primero los casos respecto a la primera variable, después, para cada valor de esta primera variable, ordena los casos respecto a la segunda variable, y así sucesivamente. Por ejemplo, si se introducen *edad* y *peso* (en este orden, y especificando el orden ascendente), los casos quedarán ordenados por *edad*, y dentro de cada *edad*, quedarán ordenados por *peso*. En el objeto creado en la salida anterior de R, `datos.ordenados_edad`, la segunda observación corresponde con el paciente 1 (26 años, 64.75kg y 1.80m), mientras que la tercera fila denota la información del paciente 4 (26 años, 58.75kg y 1.62m). Si se introduce en la ordenación la variable *peso*, estos individuos alterarán la posición en el fichero de datos, puesto que el paciente 4 pesa menos que el paciente 1.

```
datos.ordenados_edad_peso=datos_original[order(edad,peso),]
datos.ordenados_edad_peso
```

```
##   edad  peso altura
## 6   25 59.80   1.83
## 4   26 58.75   1.62
## 1   26 64.75   1.80
## 7   27 51.30   1.63
## 5   27 63.30   1.71
## 3   29 61.20   1.71
## 2   32 55.25   1.65
```

```
detach(datos_original)  #Anula la funcion attach(datos)
```

2.4 Librerías en R

Una de las ventajas de R es que permite al usuario trabajar con multitud de librerías, también denominadas **paquetes** (de aquí en adelante se usarán ambos términos indistintamente), las cuales contienen subprogramas que aumentan su funcionalidad. El abanico de posibilidades que ofrece el uso de varios paquetes sobre unos datos de partida permite enriquecer su análisis y, por supuesto, las conclusiones pertinentes. En la siguiente dirección web (conocida como Cran de R) se puede encontrar todas las librerías ordenadas alfabéticamente

que están disponibles en R: Link para acceder a las librerías de R. Si se pincha en cualquier librería, se accede a otra dirección web que contiene la información del paquete seleccionado y donde se ubica un manual de referencia en formato .pdf que explica todas las funciones de las que dispone la propia librería. Otro enlace de interés es r-bloggers.com. Este portal es un agregador de contenido del blog aportado por blogueros que escriben sobre R (en inglés) que ayuda a los bloggers y usuarios de R a conectarse y utilizar distintas librerías de R.

Algunas librerías se cargan por defecto, como es el caso de las librerías básicas (base, graphics, stats, etc.). Las que vengan instaladas por defecto (depende de la versión) no necesitan ser cargadas. Otras librerías pueden cargarse si el usuario lo solicita mediante la función **library()** indicando el nombre del paquete dentro de los paréntesis (sin comillas). Hay que notar que es altamente probable que el paquete que se vaya a utilizar no se encuentre instalado en R. En cuyo caso, habrá que instalar la librería que se vaya a utilizar. Para instalar un paquete se puede utilizar la función **install.packages()** escribiendo entre comillas el nombre del paquete, o bien, dirigirse al menú **Packages** del panel 2, hacer click en el botón **Install** e introducir el nombre del paquete en el recuadro **Packages (separate multiple with space or comma):** que aparecerá. En esta ventana también se puede indicar el destinto dónde se guardará el paquete y el lugar de dónde se descargará, pero se recomienda no alterar estas opciones. Una vez instaladas, ya se pueden cargar con la función library. Se invita al usuario a instalar las librerías `foreign`, `ggplot2` y `readxl` que se utilizarán a lo largo del presente curso.

2.4.1 Instalación de la librería BioestadisticaR

Hay librerías que no están ubicados en la red CRAN de R. A modo de ejemplo, se va a proceder a instalar el paquete **BioestadisticaR**, el cual se encuentra en un repositorio local (en este caso, en la plataforma PRADO) y que será utilizado en futuras prácticas. Para su instalación, es necesario descargar previamente el archivo "**zip**" disponible en la plataforma docente (como se muestra en la Figura 1).

Figure 1: Figura 1. Descarga de la Librería BioestadisticaR en formato **zip**.

Una vez realizado este paso, para instalar dicho paquete en *R Studio* habrá que dirigirse a la opción del menú **Tools→Install Packages** que se muestra en la Figura 2.

Una vez activada la opción del menú, hay que indicar que el paquete se instalará desde un fichero local (Figura 3) e indicar la ubicación del archivo que se ha descargado de PRADO. Al hacerlo, comenzará la instalación.

Finalmente, es necesario habilitar el paquete para poder utilizar las funciones que se encuentran en él. Para ello, se debe marcar la casilla correspondiente en la pestaña **Packages** (Figura 4) del panel 2.

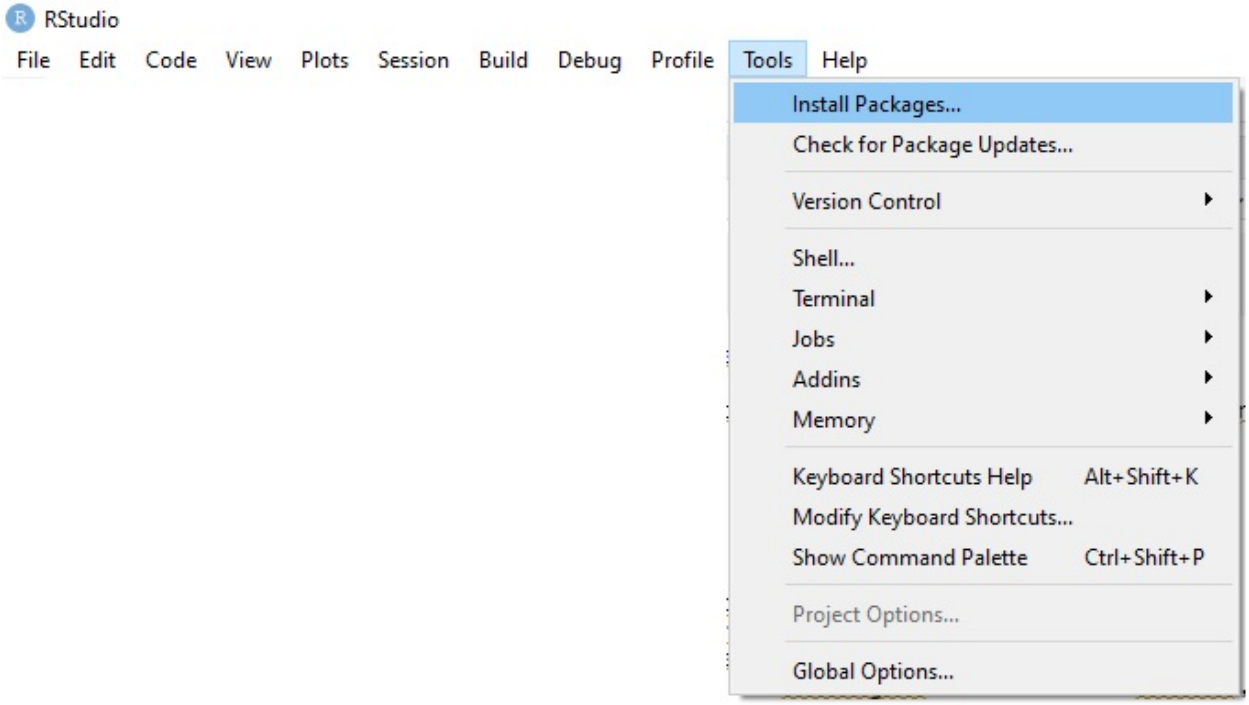

Figure 2: Figura 2. Opción de menú en R Studio para cargar paquetes

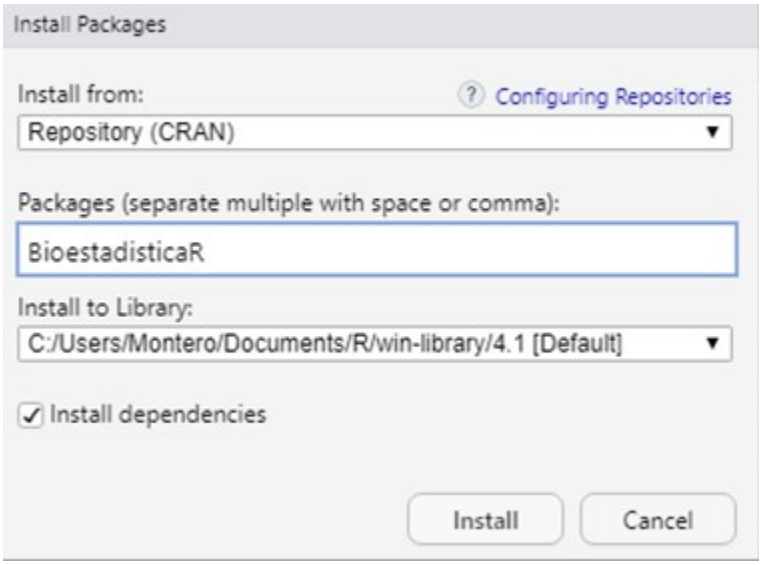

Figure 3: Figura 3. Ventana que permite seleccionar paquetes desde un archivo .zip

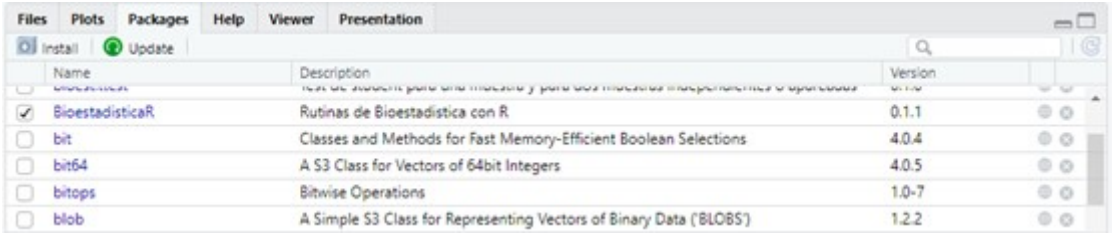

Figure 4: Figura 4. Pestaña de R Studio que permite ver los paquetes instalados y seleccionar los que se desea habilitar