



Introducción a R

MATERIAL EXTRA

¿QUÉ ES R Y RSTUDIO?, PRIMEROS PASOS E IMPORTACIÓN DE DATOS

Material práctico

Autor:

Econ. Alexis Adonai Morales Alberto

2025

Índice

| | |
|---|-----------|
| ¿Qué es R? | 2 |
| ¿Qué es RStudio? | 3 |
| ¿De donde se puede descargar R y Rstudio? | 5 |
| Iniciando en R y RStudio | 6 |
| R | 6 |
| RStudio | 6 |
| ¿Cómo se crea un proyecto en RStudio? | 7 |
| ¿Cómo se crea un Script? | 8 |
| ¿Cómo instalar y nombrar una paquetería? | 9 |
| Tipo de operaciones | 11 |
| Suma | 11 |
| Resta | 12 |
| Multiplicación | 12 |
| División | 13 |
| Variables | 13 |
| Vectores | 14 |
| Extraer un elemento de un vector | 15 |
| Matrices | 15 |
| Ejemplos | 15 |
| Operaciones de matrices | 17 |
| Suma | 17 |
| Resta | 17 |
| Multiplicación | 17 |
| Transposición, determinante e inversa de una matriz | 18 |
| Transposición | 18 |

| | |
|--|-----------|
| Determinante | 18 |
| Inversa | 19 |
| Marco de datos “data frame” | 20 |
| ¿Cómo se pueden extraer los elementos? | 20 |
| Listas | 21 |
| Carga de bases de datos desde archivos | 23 |
| Preparación para importar y exportar datos | 23 |
| setwd() | 23 |
| Almacenando la base dentro de un proyecto | 24 |
| Importación de datos | 25 |
| Comando read.delim(“clipboard”) | 26 |
| Lectura de archivos CSV. | 28 |
| Cargando bases CSV con el espacio previamente establecido | 28 |
| Uso de botonera | 29 |
| Lectura de archivos DTA. | 30 |
| Cargando bases DTA con el espacio previamente establecido | 30 |
| Uso de botonera | 31 |
| Lectura de archivos SPSS. | 32 |
| Cargando bases SAV (SPSS) con el espacio previamente establecido | 32 |
| Uso de botonera | 34 |
| Lectura de archivos EXCEL. | 34 |
| Librería readxl. | 35 |
| Librería openxlsx. | 36 |
| Lectura de archivos DBF | 36 |
| Cargando bases DBF con el espacio previamente establecido | 37 |
| Exportación de datos | 37 |
| Creado data.frame para la elaboración de ejemplos | 38 |
| Exportar en salida .csv | 39 |

| | |
|-------------------------------------|----|
| Exportar en salida .DTA | 39 |
| Exportar en salida .SAV | 39 |
| Exportar en salida .RData | 40 |

¿Qué es R?

R es un lenguaje estadístico computacional, aunque también está catalogado como de programación. Nació como una reimplementación del lenguaje libre S.

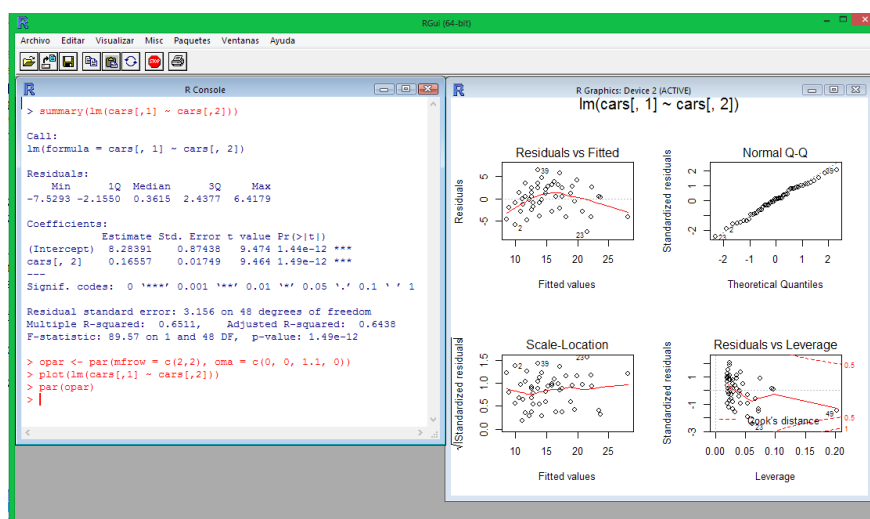
Apareció en 1993 en Austria y hoy en día es una herramienta utilizada por estudiosos de la estadística, economía, actuaría, matemáticas, biología y diversas áreas que requieran del análisis de datos.

Figura 1: Logo de r-project.



Su implementación es diversa, pues con el paso del tiempo sus aplicaciones han alcanzado desde realizar gráficos simples o avanzados, modelos de predicción, cartografía o simple análisis estadístico (ya sea descriptivo o inferencial). Además, también ha tenido desarrollo en la administración de bases de datos, tal como la manipulación, limpieza y ordenamiento de datos. Ha tenido desarrollos como Rmarkdown, Quatro o Shiny que permiten realizar tareas como la creación de documentos de texto (como este que estás leyendo) hasta aplicaciones interactivas como *Web applications*.

Figura 2: Ejemplo 1



¿Qué es RStudio?

RStudio es un entorno o interfaz de desarrollo integrado para el lenguaje de programación de R, dedicado a la computación estadística, gráficos y reportes. En el mismo, se incluye una consola, editor de sintaxis y algunos espacios para visualizar los gráficos, utilerías y herramientas de R.

Fue lanzado por primera vez como software libre en Febrero del 2011 por Joseph J. Allaire mediante una estructura de Java, C++ y JavaScript¹.

Figura 3: Logo de RStudio

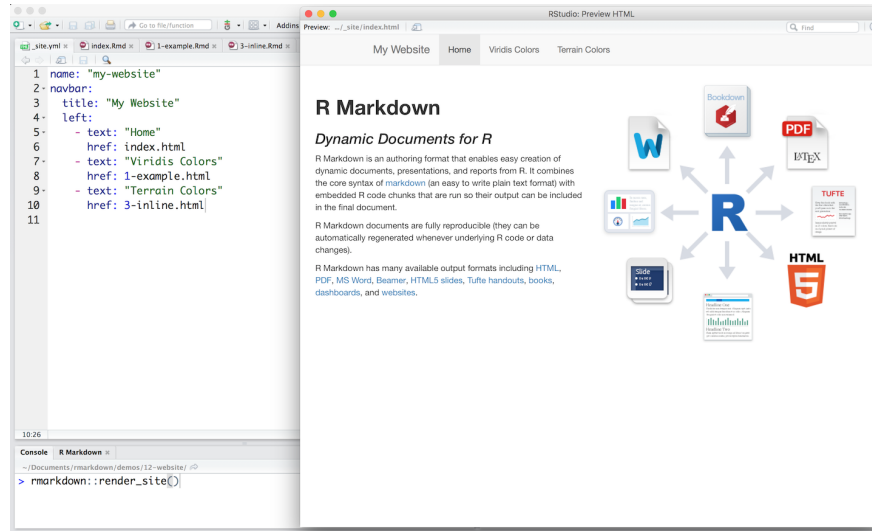


Su uso es dinámico, puesto a que aparte de poder enlazar el lenguaje de R, se pueden elaborar otra clase de documentos (cómo RMardown, Shiny, Quatro) en lo cuales se

¹Hay que recordar que estos lenguajes mencionados se utilizan para el desarrollo de aplicaciones de computación, en este caso se usaron para desarrollar el entorno de RStudio.

pueden realizar reportes combinando LaTeX o la asociación de otros programas y/o lenguajes de computo, tales como Python, SQL, etc. Es por ello que es una interfaz que ha cobrado mucho uso en diferentes áreas de estudio.

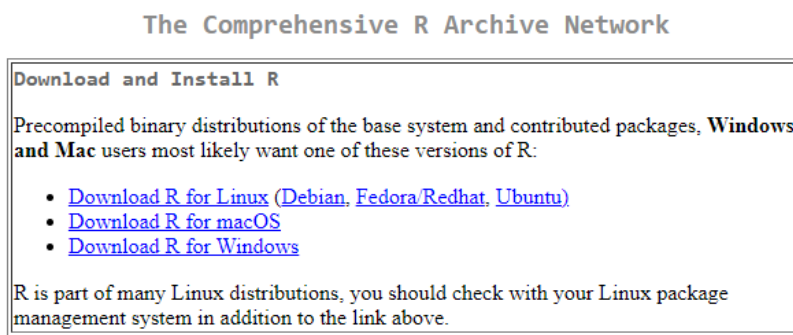
Figura 4: Ejemplo 2



¿De donde se puede descargar R y Rstudio?

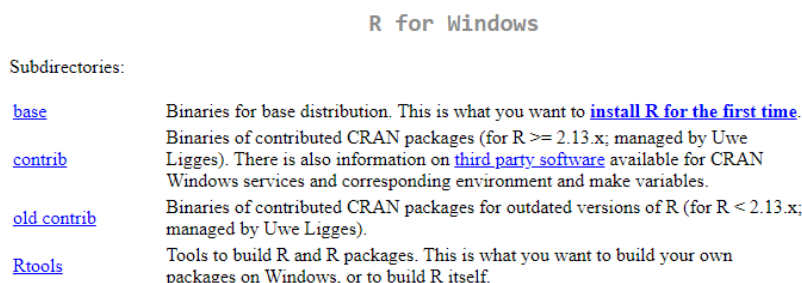
Para descargar **R-project** debes de ingresar a la siguiente dirección: <https://cloud.r-project.org/> donde ahí tendrás tres opciones para diferentes sistemas operativos: Linux, macOS y Windows.

Figura 5: Descargas para diferentes S.O.



En nuestro caso seleccionaremos Windows, posterior a ello, tendremos una ventana como la siguiente:

Figura 6: Descarga para Windows

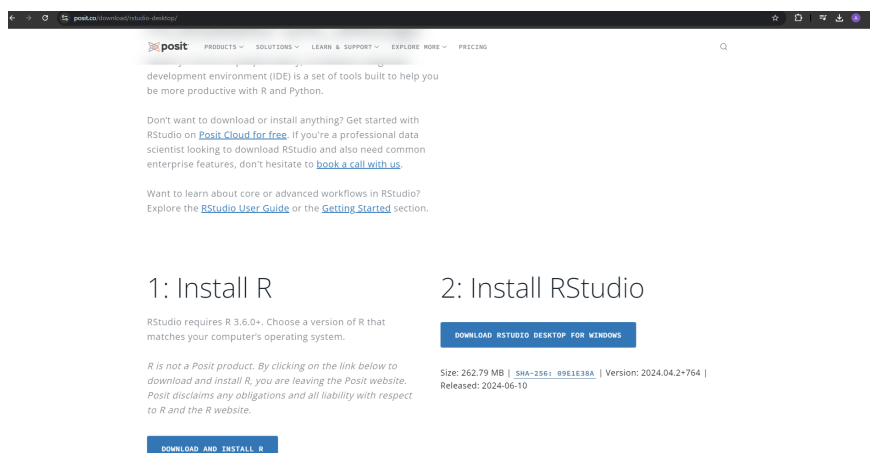


Posterior a ello, seleccionamos **install R for the first time** y en automático comenzará la descarga del archivo .exe con el cual podrás instalar el lenguaje de R. Además de instalar R, en el caso de Windows hay que complementar con RTools, esta herramienta permite compilar paquetes que provengan de otras estructuras (como C++), es por ello que también se requiere.

Para proceder a instalar Rtools, es necesario dirigirse al sitio que aparece en la imagen anterior y seleccionar *Rtools installer* y posteriormente realizar el proceso de instalación.

Para descargar **RStudio** nos dirigimos al siguiente enlace: <https://posit.co/download/rstudio-desktop/> donde podremos encontrar la página que enlazará con el descargador de RStudio en la versión de escritorio gratuita.

Figura 7: Descarga de RStudio para escritorio



Iniciando en R y RStudio

En esta sección podrás encontrar todo lo relacionado con los primeros pasos de R y RStudio, divididos en dos subtemas, en los cuales se harán diversas explicaciones del uso de este lenguaje.

R

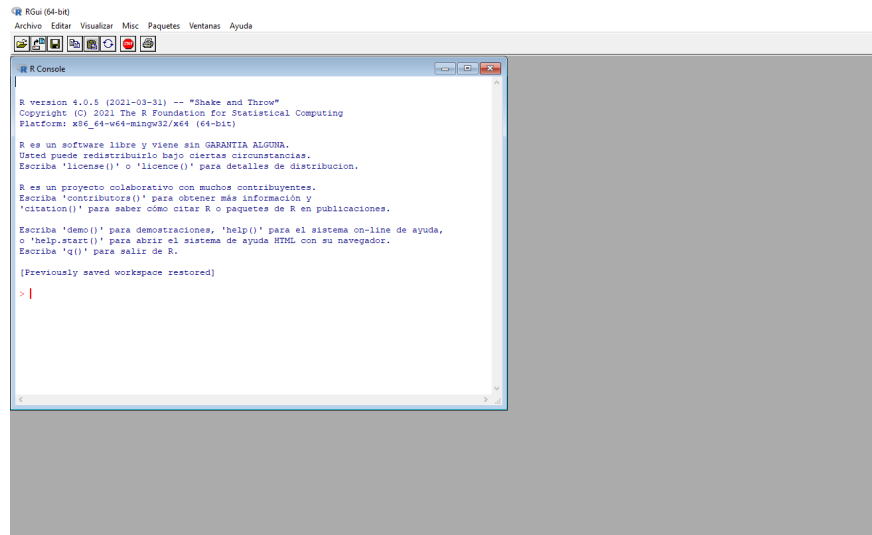
Una vez instalado R, Rtoos al ejecutarlo encontrarás un interfaz muy similar a lo que es CMD o *símbolo de sistema*, en el cual solo se pueden ejecutar códigos del propio lenguaje, sin poder elaborar algún archivo de *memoria* o de serie de comandos que se puedan ejecutar automáticamente, es por ello que R suele ser un poco tedioso, ya que solo se puede escribir una vez el código y si este resulta estar mal, tendrás que regresar y ver si permite corregir dicho código, o en su defecto, reescribir el código.

RStudio

RStudio tiene una facilidad de manejo, puesto a que su interfaz permite una interacción más eficiente para administrar el trabajo, pues la pantalla se divide en diversos espacios, los cuales son:

- **Script:** Es un archivo con la extensión **.R** en el cual contiene líneas de comando, en el cual se pueden escribir diversas instrucciones y ejecutar, pero que permite la corrección inmediata del código sin necesidad de regresarse o de reescribir todo el código.

Figura 8: Consola de R



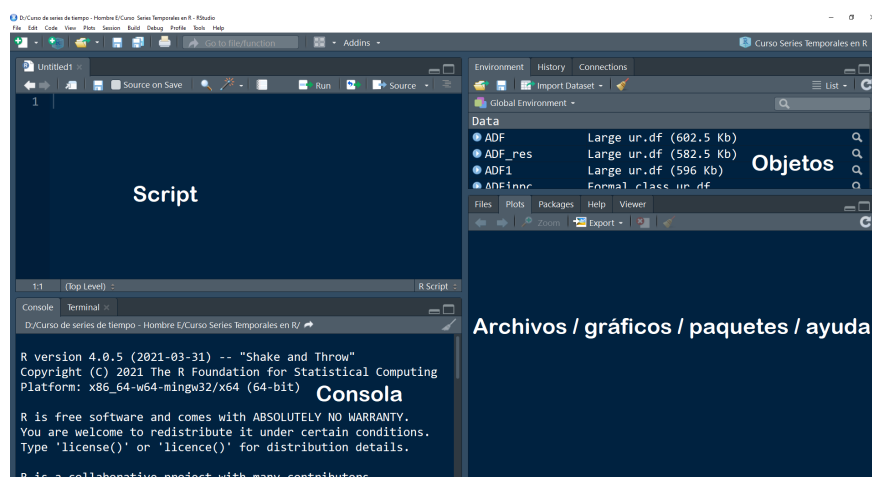
- **Objetos:** Es un espacio donde se podrán visualizar diversos objetos, tales como: Dataframe, vectores de valores, series de tiempo, objetos de clase, logicos, etc.
- **Consola:** Espacio donde se refleja el proceso de ejecución de los comandos del lenguaje, es decir, es tal cual la interfaz que se visualiza en **R**.
- **Archivos, gráficos, paquetes y ayuda:** Es una espacio donde se pueden visualizar los archivos de donde se enruta el espacio de trabajo, también se reflejan los gráficos una vez ejecutados, además podemos observar si las paqueterías instaladas están en funcionamiento o simplemente no están instaladas y por último, la ventana de ayuda, visualiza la estructura o documentos de los comandos o paqueterías de R.

¿Cómo se crea un proyecto en RStudio?

Para poder elaborar un proyecto, en el cual se incluyan archivos **.R** (Script), bases de datos con extensiones **.xlsx**, **.dta**, **.csv** o imágenes (**.png**) se requiere la siguiente ruta de RStudio. Para ello nos vamos a **“File > New project”**:

Posterior a ello, se elegirá **“New Directory > New Project”**, donde nombraremos el directorio y enrutaremos a una carpeta donde solo contendrá el proyecto (lo anterior para facilidad de manipulación).

Figura 9: Interfaz de RStudio



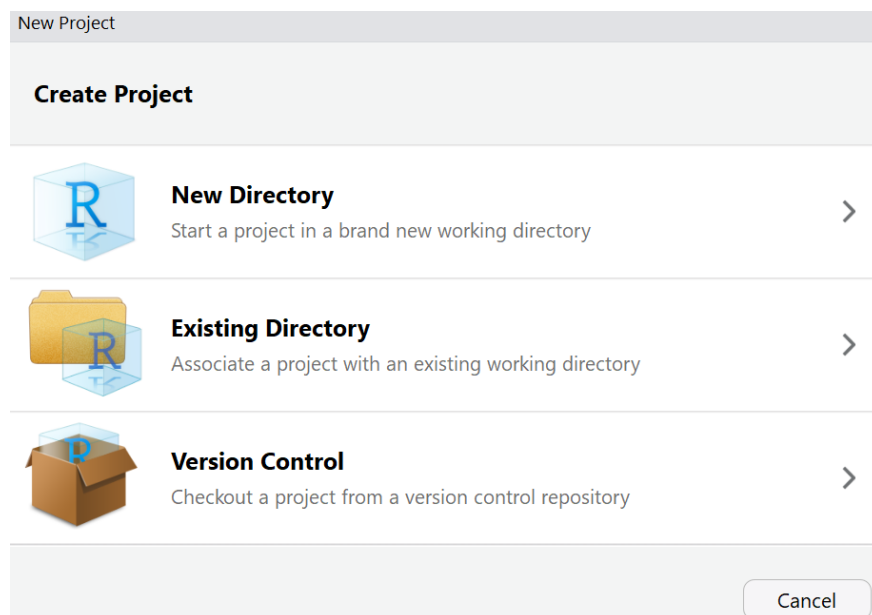
¿Cómo se crea un Script?

Una vez ya elaborado el proyecto, tendremos la siguiente visualización:

Después solo se presionará `ctrl + shift + N` y desplegará una ventana nueva con el Script:

Finalmente para ejecutar una línea de código, debes posicionarte al final del comando y presionar `ctrl + enter`.

Figura 10: Nuevo proyecto



¿Cómo instalar y nombrar una paquetería?

Para instalar una paquetería en R, solo se debe seguir el siguiente comando (la librería **tseries** es un ejemplo):

```
# install.packages("tseries", dependencies = TRUE)

# sirve como comentario, para replicarlo,
# escribe en el script sin este simbolo "#"
```

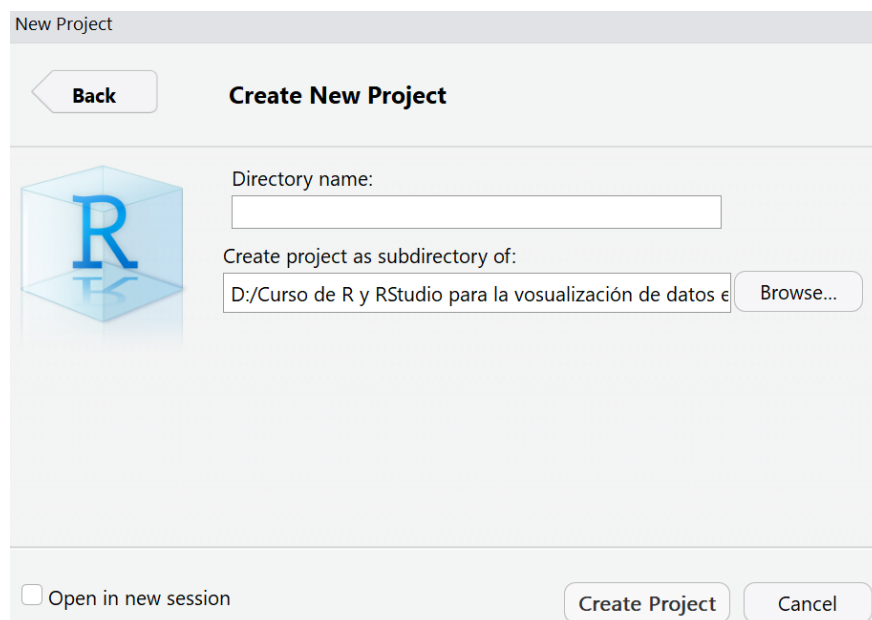
El comando se divide en: **install.packages("librería a elegir o instalar", dependencies=TRUE)** donde *dependencies* al ser verdadero se indica que instalará aquellos paquetes que se utilizan en el paquete que se desea instalar, por ejemplo el paquete **tseries** necesita **quantmod**, **xts** y algunas otras librerías extras para que funcionen algunos comandos y/o funciones.

De igual manera para instalar varias paqueterías en una sola línea se necesita crear un vector con los nombres de las paqueterías para posteriormente instalarlas, por ejemplo:

```
# paquetes<-c("ggplot2", "dplyr")
# install.packages(paquetes, dependencies=TRUE)
```

Para llamar una librería se utiliza lo siguiente:

Figura 11: Nombramiento y enrutamiento del proyecto



```
library(tseries)
```

Pero de igual manera, podemos llamar diversas librerías de forma simultanea con la ayuda de la función **libraries** que se encuentra en la librería *easypackages*:

```
library(easypackages)
```

```
paquetes <- c("tidyverse", "urca")  
libraries(paquetes)
```

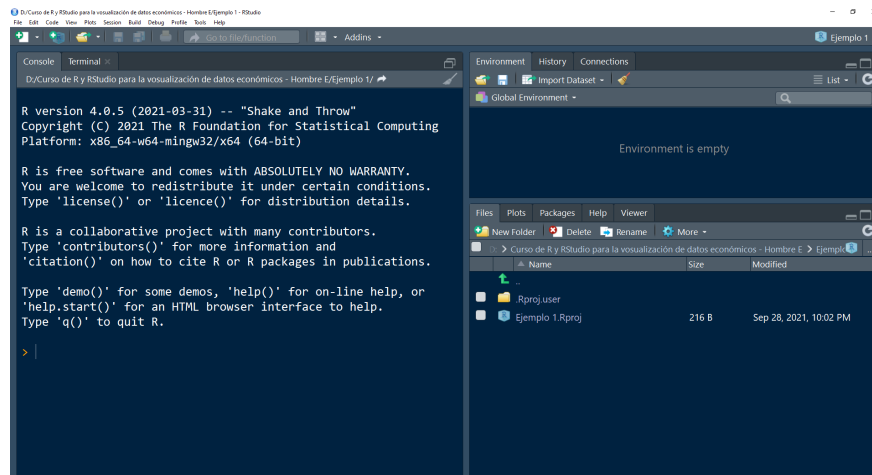
Por último, podemos llamar funciones sin la necesidad de cargar la paquetería completa, para ello utilizamos dobles puntos (:), pero la estructura es: *paquete::función*, tal como se mostrará a continuación.

```
x<-seq(1, 20, 1)
```

```
urca::summary(x)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.     
##      1.00   5.75   10.50   10.50   15.25   20.00
```

Figura 12: Proyecto nuevo



Tipo de operaciones

En R existen diversas operaciones, tales como: Suma, resta, multiplicación y división. Además de otro tipo de funciones así como valores lógicos o de caracteres.

Suma

Para realizar una suma, es muy sencillo, simplemente se debe ocupar el operador “+” y con ello se ejecuta la suma, por ejemplo:

```
5+2
```

```
## [1] 7
```

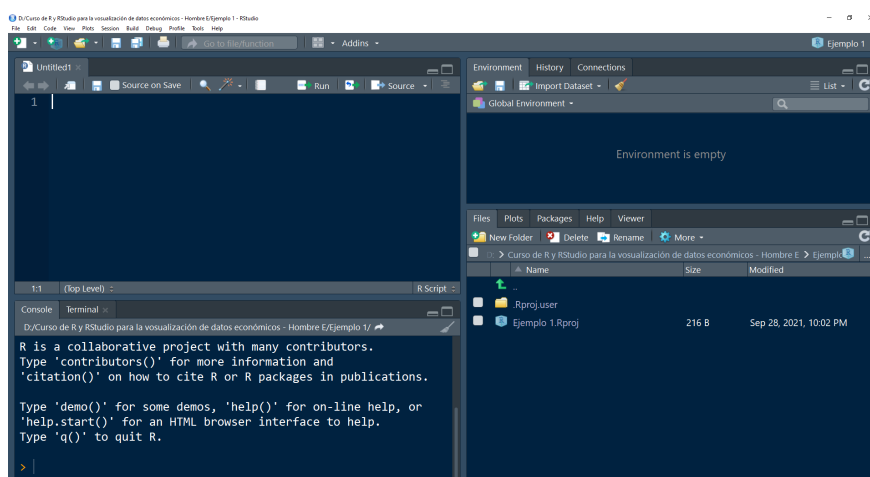
```
10+16
```

```
## [1] 26
```

```
1256+2378
```

```
## [1] 3634
```

Figura 13: Nuevo script



Resta

Para realizar una resta, solo se tiene que utilizar el simbolo “-” para indicar la operación, por ejemplo:

```
5-2
```

```
## [1] 3
```

```
15-34
```

```
## [1] -19
```

```
9873-3091
```

```
## [1] 6782
```

Multiplicación

Cuando se requiere efectuar una mutiplicación, es necesario utilizar el símbolo “*”, con ello se puede realizar la operación, por ejemplo:

```
6*12
```

```
## [1] 72
```

```
10*10
```

```
## [1] 100
```

```
35*89
```

```
## [1] 3115
```

Sin embargo cuando se trata de matrices, el operador cambia a `“%%”`.

División

Por último, para poder operar una división solo basta con utilizar `“/”`, ejemplo:

```
100/10
```

```
## [1] 10
```

```
25/195
```

```
## [1] 0.1282051
```

```
1/3
```

```
## [1] 0.3333333
```

Variables

Dentro de R, existen diversos tipos de objetos, los cuales son diversos. Pero también se pueden asignar variables, donde se otorga un valor a una letra o nombre en específico, para realizar lo anterior se utiliza el operador `<-`, por ejemplo:

```
# Variables numéricas
```

```
x<-67
```

```
x
```

```
## [1] 67
```



```
5★x-43
```

```
## [1] 292
```

```
# Variables cualitativas
```

```
pais<-"Mexico"
pais
```

```
## [1] "Mexico"
```

Vectores

Este tipo de datos, son arreglos ordenados en los cuales se puede almacenar información de tipo numérico (variables cuantitativas), alfanumérico (variables cualitativa) o un valor lógico (Falso y verdadero ó **TRUE y FALSE**), pero que no son mezclas de estos. En R, para poder crear dicho vector, se utiliza el símbolo **c()** y que significa **concatenar o combinar**.

Esta función almacena la información de forma vectorial, por ende se acostumbra a etiquetarlo con un nombre corto y representativo del tipo de datos que contiene, ejemplo:

```
edad<-c(25,40,39,46,10,11,14,23,43)
deporte<-c(TRUE, TRUE, FALSE, FALSE, TRUE, TRUE, TRUE,
            FALSE, FALSE)
comic_fav<-c(NA,"Superman", "Batman", NA, "Shazam",
             "Batman", "Superman", NA, "GreenLanter")
edad
```

```
## [1] 25 40 39 46 10 11 14 23 43
```

```
deporte
```

```
## [1] TRUE TRUE FALSE FALSE TRUE TRUE TRUE FALSE FALSE
```

```
comic_fav
```

```
## [1] NA           "Superman"    "Batman"      NA            "Shazam"
## [6] "Batman"     "Superman"    NA            "GreenLanter"
```

Extraer un elemento de un vector

Para extraer un dato que se encuentre dentro de un vector, se hace uso del corchete “[]” y dentro del mismo solo se coloca la posición en donde se encuentra el dato, ejemplo:

```
edad[5]
```

```
## [1] 10
```

En el ejemplo anterior se observa que el dato número 5, el valor es de 10.

Ahora, si deseamos encontrar el dato número 2 y 9 del vector *comic_fav* solo tenemos que utilizar **[c(2,8)]** para realizar esta búsqueda, ejemplo:

```
comic_fav[c(2,9)]
```

```
## [1] "Superman" "GreenLanter"
```

Por último, si se busca exceptuar un dato dentro de un vector solo se debe poner el valor como **[-a]** donde *a* es la posición del dato a omitir, ejemplo:

```
deporte[-5]
```

```
## [1] TRUE TRUE FALSE FALSE TRUE TRUE FALSE FALSE
```

Matrices

Las matrices son arreglos rectangulares de filas y columnas con información numérica, alfanumérica o lógica. Para crear una matriz, se utiliza la función **matriz()**.

Ejemplos

Vamos a contruir diversas matrices, con la finalidad de demostrar que hay diferentes maneras de declarar una matriz.

Para crear una matriz cuadrada:

```
matriz1<-matrix(c(11,21,31,12,22,32,13,23,33), nrow = 3, ncol = 3)
```

```
matriz1
```

```
##      [,1] [,2] [,3]
## [1,]   11   12   13
## [2,]   21   22   23
## [3,]   31   32   33
```

Para crear una matriz cuadrada de números consecutivos

```
matriz2<-matrix(1:9, nrow=3, ncol=3)

matriz2
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

Otra forma de plantear matrices, es mediante vectores y con la función **cbind()** la cual une columnas, ejemplo:

```
vector1<-c(34,56,22)
vector2<-c(22,67,98)
vector3<-c(12,45,76)

matriz3<-matrix(cbind(vector1,vector2, vector3), nrow = 3, ncol = 3)

matriz3
```

```
##      [,1] [,2] [,3]
## [1,]   34   22   12
## [2,]   56   67   45
## [3,]   22   98   76
```

Ahora bien, si lo que se busca es acomodar los vectores por fila, se hace uso de **rbind()**, ejemplo:

```
matriz4<-matrix(rbind(vector1, vector2, vector3), nrow = 3, ncol = 3)

matriz4
```

```
##      [,1] [,2] [,3]
## [1,]   34   56   22
## [2,]   22   67   98
## [3,]   12   45   76
```

Operaciones de matrices

Para realizar diversas operaciones de matrices, se debe tomar en cuenta que las matrices a operar tienen que ser cuadráticas (mismas filas y columnas), ya que de no ser así las operaciones tal vez no podrás realizarse.

Como ejemplos, se utilizarán las matrices 3 y 4 de los ejemplos anteriores.

Suma

```
matriz5<- matriz3+matriz4
```

```
matriz5
```

```
##      [,1] [,2] [,3]
## [1,]   68   78   34
## [2,]   78  134  143
## [3,]   34  143  152
```

Resta

```
matriz6<- matriz3-matriz4
```

```
matriz6
```

```
##      [,1] [,2] [,3]
## [1,]    0  -34  -10
## [2,]   34    0  -53
## [3,]   10   53    0
```

Multiplicación

```
matriz7<- matriz3%*%matriz4
```

```
matriz7
```

```
##      [,1] [,2] [,3]
## [1,] 1784  3918  3816
## [2,] 3918  9650 11218
## [3,] 3816 11218 15864
```

Transposición, determinante e inversa de una matriz

En esta sección se demuestra como se pueden realizar estas operaciones en R.

Transposición

Se utiliza para cambiar de lugares de las filas por columnas y viceversa en una matriz, en R, para realizar dicha acción se utiliza **t()**, ejemplo:

```
matriz4
```

```
##      [,1] [,2] [,3]
## [1,]   34   56   22
## [2,]   22   67   98
## [3,]   12   45   76
```

```
matrizt<-t(matriz4)
```

```
matrizt
```

```
##      [,1] [,2] [,3]
## [1,]   34   22   12
## [2,]   56   67   45
## [3,]   22   98   76
```

Determinante

Se utiliza para obtener una forma multilineal alternada de un cuerpo. Esta definición indica una serie de propiedades matemáticas y generaliza el concepto de determinante haciéndolo aplicable en numerosos campos, para realizar dicha operación en R, se utiliza **det()**, ejemplo:

```
matriz3
```

```
##      [,1] [,2] [,3]
## [1,]   34   22   12
## [2,]   56   67   45
## [3,]   22   98   76
```

```
matrizdet<-det(matriz3)
```

```
matrizdet
```

```
## [1] -496
```

Inversa

La inversa de una matriz se utiliza por la necesidad de *dividir* matrices, ya que en el álgebra lineal o matricial, no existe tal operación, es por ello que se genera el concepto de *inversa*.

Para realizar este proceso se requiere:

- Definir una matriz $m=n$ elementos, que en pocas palabras, es cuadrática.
- Definir una matriz de diagonales (matriz de identidad), esto se realiza con el comando **diag(1,nrow=n)**, se debe definir la cantidad de 1 en la diagonal conforme al valor de la matriz cuadrática a invertir.
- Se utiliza el comando **solve()** para resolver la matriz y obtener la inversa.

Ejemplo:

```
matriz3 #Es cuadrática 3X3
```

```
##      [,1] [,2] [,3]
## [1,]   34   22   12
## [2,]   56   67   45
## [3,]   22   98   76
```

```
I<-diag(1, nrow = 3)
```

```
I
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
```

```
matriz3_inv<-solve(matriz3, I)
```

```
matriz3_inv
```

```
##           [,1]      [,2]      [,3]
## [1,] -1.375000  1.000000 -0.375000
## [2,]  6.584677 -4.677419  1.729839
## [3,] -8.092742  5.741935 -2.108871
```

Marco de datos “data frame”

El marco de datos o *data frame* es uno de los objetos más utilizados porque permite agrupar varios vectores con información de diferente tipo (numérico, alfanumérico o lógico) en un mismo objeto, la única restricción es que todos los elementos deben contener el mismo espacio o dimensión.

Para generar un marco de datos en R, se utilizará la función **data.frame**, ejemplo:

```
datos<-data.frame(edad,deporte,comic_fav)
```

```
datos
```

```
##   edad deporte  comic_fav
## 1   25     TRUE    <NA>
## 2   40     TRUE  Superman
## 3   39    FALSE   Batman
## 4   46    FALSE    <NA>
## 5   10     TRUE   Shazam
## 6   11     TRUE   Batman
## 7   14     TRUE  Superman
## 8   23    FALSE    <NA>
## 9   43    FALSE GreenLanter
```

¿Cómo se pueden extraer los elementos?

Para poder recuperar las variables (columnas) contenidas en el marco de datos (*data frame*), se pueden usar diferentes operadores: \$, corchetes simples [] o corchetes dobles [[]].

Ejemplo:

```
datos$deporte # Con $
```

```
## [1]  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE FALSE FALSE
```

```
datos[,2] # Con corchete
```

```
## [1] TRUE TRUE FALSE FALSE TRUE TRUE TRUE FALSE FALSE
```

```
datos[["deporte"]] # Con doble corchete
```

```
## [1] TRUE TRUE FALSE FALSE TRUE TRUE TRUE FALSE FALSE
```

Listas

Las listas son otro tipo de objeto muy usado para almacenar objetos de diferente tipo, aquí ya no importa la dimensión. Para poder crear una lista en R, solo basta con usar **list()**, ejemplo:

```
lista<-list(E1=datos, E2=matriz7, E3=vector1)
```

```
lista
```

```
## $E1
##   edad deporte  comic_fav
## 1   25     TRUE      <NA>
## 2   40     TRUE    Superman
## 3   39    FALSE     Batman
## 4   46    FALSE      <NA>
## 5   10     TRUE     Shazam
## 6   11     TRUE     Batman
## 7   14     TRUE    Superman
## 8   23    FALSE      <NA>
## 9   43    FALSE GreenLanter
##
## $E2
##      [,1] [,2] [,3]
## [1,] 1784 3918 3816
## [2,] 3918 9650 11218
## [3,] 3816 11218 15864
##
## $E3
## [1] 34 56 22
```

De igual forma, para extraer elementos se hace uso de los mismo operadores que con el *data frame*:


```
lista$E1 #Con $
```

```
##      edad deporte      comic_fav
## 1      25      TRUE      <NA>
## 2      40      TRUE      Superman
## 3      39     FALSE      Batman
## 4      46     FALSE      <NA>
## 5      10      TRUE      Shazam
## 6      11      TRUE      Batman
## 7      14      TRUE      Superman
## 8      23     FALSE      <NA>
## 9      43     FALSE GreenLanter
```

```
lista[2] #Con un solo []
```

```
## $E2
##      [,1] [,2] [,3]
## [1,] 1784 3918 3816
## [2,] 3918 9650 11218
## [3,] 3816 11218 15864
```

```
lista[[3]] #Con dos [[]]
```

```
## [1] 34 56 22
```

Carga de bases de datos desde archivos

En esta sección se mostrará cómo se pueden importar y exportar datos en el espacio de trabajo de R, mediante algunas librerías tales como: **foreign**.

Es importante señalar que se pueden cargar datos de diferentes tipos de archivos o extensiones, pueden ser varios, tales como: **CSV**; que es un tipo de datos separados por comas, o bien, también se pueden cargar datos con el formato **.DTA** que refiere a un espacio de almacenamiento de Stata.

Además de las dos extensiones mencionadas, R permite realizar lecturas de extensiones **.SAV** pertenecientes a una memoria de almacenamiento de datos de SPSS.

Y por último, tenemos un tipo de archivo que tiene la extensión de **.DBF** es el formato de datos utilizado originalmente por el producto *dBase*, siendo en la actualidad el formato más comúnmente utilizado en DBMS (Sistema de Gestión de Base de Datos), para computadoras personales.

Preparación para importar y exportar datos

Es importante tener en cuenta que hay muchas maneras de importar los datos en R, en las cuales RStudio también juega un papel destacado, puesto a que con esta interfaz podemos cargar datos con el uso de la botonera en vez de programar el código.

Para exportar datos, es necesario contar con un espacio o **ruta** definido, puesto a que se utilizarán funciones de escritura donde se podrán almacenar los archivos exportados en diferentes formatos. Cabe recalcar que solo se puede realizar mediante comando y no hay una función dentro de la botonera de RStudio que permita realizar dicha acción.

Es por ello que a continuación se mostrarán las formas en las cuales podrás enrutar un espacio de trabajo (en caso de no contar con un proyecto) o bien, se mostrará una forma sutil en la cual se pueden cargar y exportar las bases de una forma muy simple.

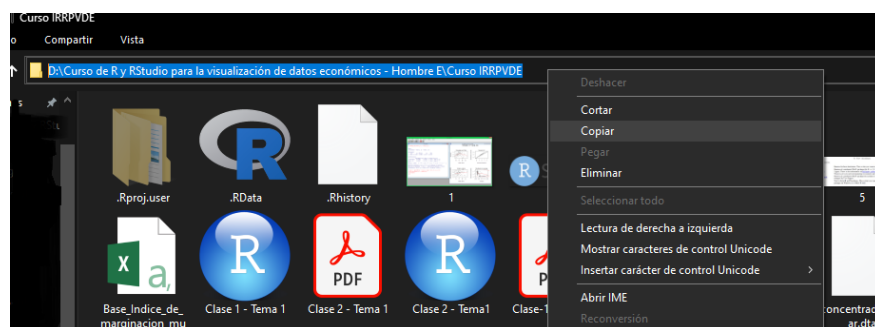
setwd()

La función **setwd()** permite enrutar un espacio de trabajo o una carpeta de donde se harán lectura de diversos archivos que contenga este espacio, con ello podremos realizar carga de datos de una forma más útil.

Para realizar lo anterior es necesario conocer la ruta de la carpeta donde se contienen los archivos, en el caso de Windows en muy sencillo podemos obtener dicha ruta, solo basta con dar click derecho y seleccionar “**copiar dirección**”.

Posterior a ello, se utilizará el comando en R:

Figura 14: Copiando ruta



```
setwd("E:/GEM/Materiales extra/Introducción a R/Bases")
```

```
# Se reemplaza \ por /
```

Con lo anterior ya tendremos la facilidad de utilizar los comandos base de “**read.csv()**” y solo tendremos que poner dentro del parentesis el nombre del archivo con la extensión .csv.

Almacenando la base dentro de un proyecto

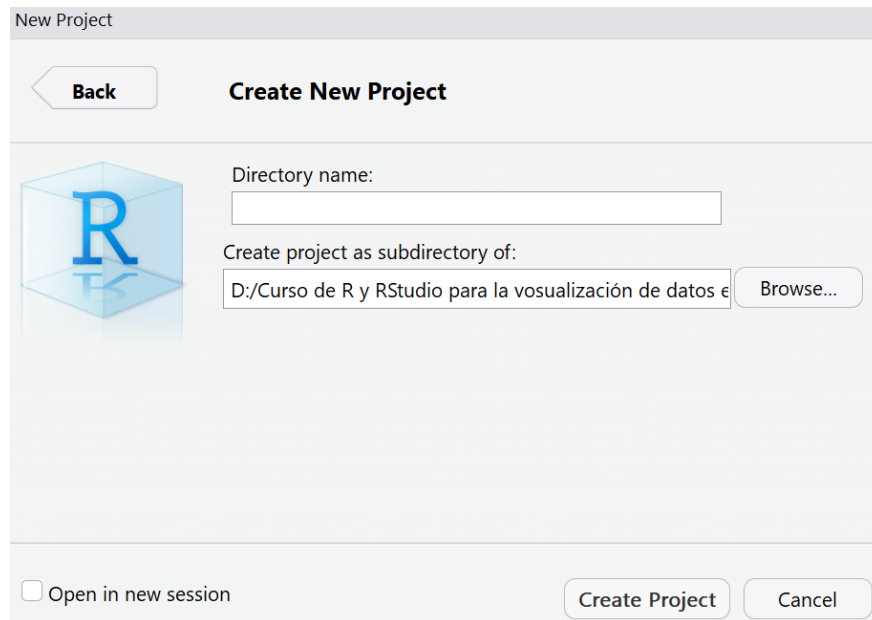
La otra manera de preparar todo, es mediante la creación de un proyecto, donde en este se contenga todas las bases de datos a usar dentro de un análisis. Para esto se crea un proyecto nuevo, con una ruta y nombre en específico.

Posteriormente, se guardarán las bases dentro de la carpeta donde se encuentra el proyecto:

Una vez realizado lo anterior, cuando se este trabajando en RStudio, en la parte de “**Files**” se podrán visualizar los archivos.

Y nuevamente para llamar a una base, solo se ocuparán los comandos pertenecientes al tipo de extensión que se requieren.

Figura 15: Proyecto nuevo



Importación de datos

A continuación, se mostrarán en diversos subtemas los tipos de archivo que se pueden cargar usando RStudio, así mismo como el método de carga ya sea mediante un espacio previamente definido o con el uso de la botonera.

Figura 16: Carpeta del proyecto con las bases de datos

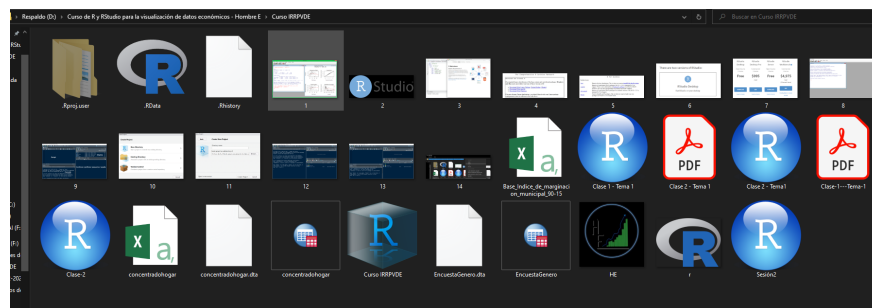
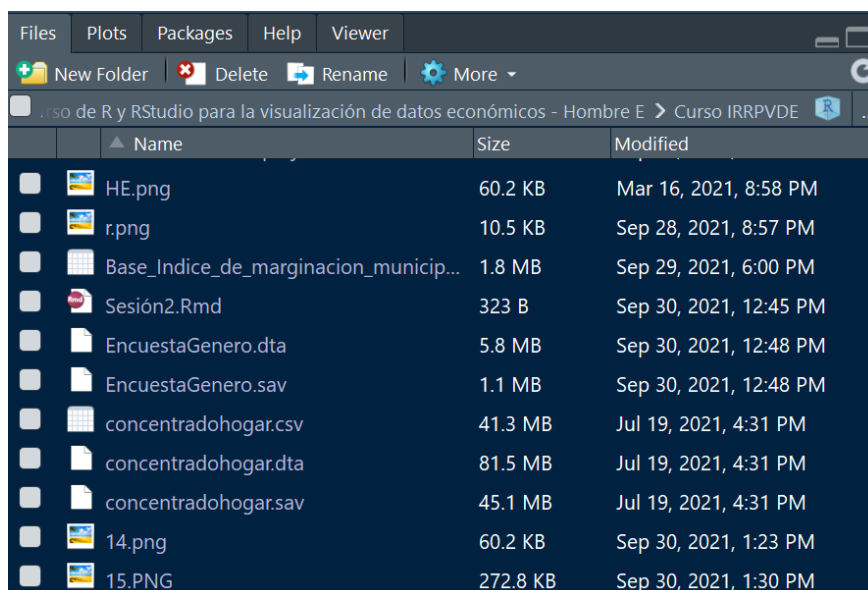


Figura 17: Archivos



Comando `read.delim("clipboard")`

Una manera muy sencilla de cargar una base de datos, es mediante el comando **`read.delim("clipboard")`** con este comando podemos (literalmente) copiar y pegar datos traídos de una hoja de excel, de un archivo de Stata o SPSS.

Para realizar este paso, lo primero que se debe hacer es abrir el archivo de donde queremos sustraer los datos y copiar todo lo que queremos ingresar a R (con todo y cabeceras):

Posteriormente de copiar los datos, se utiliza el comando:

```
base<-read.delim("clipboard")
```

```
head(base)
```

```
##          GM IND0A100 LUG_NAC LUGAR_EST  AÑO
## 1 Muy bajo          - 2408.00    11.00 2015
## 2 Muy bajo      7.69 2409.00         - 2010
## 3 Muy bajo          - 2419.00    11.00 2005
## 4 Muy bajo          - 2408.00    11.00 2000
## 5 Muy bajo          - 2393.00         - 1995
## 6 Muy bajo          - 2341.00     9.00 1990
```

La desventaja de utilizar este tipo de comando es que en ocasiones (dependiendo del formato de los datos), tomará los valores “pegados” como caracteres.

Figura 18: Copiando datos

| R | S | T | U | V | W | X | |
|--------|----------|----------|---------|-----------|------|---|--|
| IM | GM | IND0A100 | LUG_NAC | LUGAR_EST | AÑO | | |
| -1.676 | Muy bajo | - | 2408 | 11 | 2015 | | |
| -1.768 | Muy bajo | 7.69 | 2409 | - | 2010 | | |
| -1.831 | Muy bajo | - | 2419 | 11 | 2005 | | |
| -1.871 | Muy bajo | - | 2408 | 11 | 2000 | | |
| -1.735 | Muy bajo | - | 2393 | - | | | |
| -1.833 | Muy bajo | - | 2341 | - | | | |
| -1.256 | Muy bajo | - | 2229 | - | | | |
| -1.262 | Muy bajo | 13.411 | 2202 | - | 2010 | | |
| -1.234 | Muy bajo | - | 2188 | - | | | |
| -1.141 | Bajo | - | 2104 | - | | | |
| -0.698 | Bajo | - | 1799 | - | | | |
| -0.972 | Bajo | - | 1975 | - | | | |
| -0.754 | Bajo | 19.152 | 1836 | - | | | |
| -1.045 | Bajo | - | 2090 | - | | | |
| -1.105 | Bajo | - | 2057 | - | | | |
| -0.811 | Bajo | - | 1870 | - | | | |
| -0.875 | Bajo | - | 1935 | - | | | |
| -1.035 | Bajo | 15.982 | 2056 | - | | | |
| -0.957 | Bajo | - | 1953 | - | | | |
| -1.129 | Muy bajo | - | 2153 | - | | | |
| -0.565 | Bajo | - | 1669 | - | | | |
| -1.156 | Muy bajo | - | 2171 | - | | | |
| -0.463 | Medio | 22.445 | 1591 | - | | | |
| -1.136 | Bajo | - | 2122 | - | | | |
| -1.245 | Bajo | 13.603 | 2194 | - | | | |
| -1.124 | Bajo | - | 2091 | - | | | |
| -0.895 | Bajo | - | 1910 | - | | | |
| -0.62 | Medio | - | 1730 | - | | | |
| -1.306 | Muy bajo | - | 2239 | - | | | |
| -0.985 | Bajo | - | 1982 | - | 1995 | | |

```
class (base$IM)
```

```
## [1] "NULL"
```

```
class (base$GM)
```

```
## [1] "character"
```

Lectura de archivos CSV.

Para cargar una base de datos con la extensión o formato CSV, se utiliza el comando **read.csv("nombre.csv")** con lo cual podremos cargar los datos de dicha base en esta extensión, sin embargo hay dos maneras de realizarla: 1) Generando el espacio de trabajo o enrutando y 2) Con el uso de la botonera.

Cargando bases CSV con el espacio previamente establecido

Para cargar una base de esta manera, se debe realizar lo que previamente se realizó en este documento (ya sea usando `setwd()` o un proyecto con lo archivos a cargar), entonces, lo único que se debe realizar es simplemente asignarle un nombre a la base y cargar el archivo:

```
base1<-read.csv("Bases/MarginaciónMun.csv",
               check.names = F)
head(base1[1:3])
```

```
##      CVE_ENT      ENT CVE_MUN
## 1      1 Aguascalientes    1001
## 2      1 Aguascalientes    1001
## 3      1 Aguascalientes    1001
## 4      1 Aguascalientes    1001
## 5      1 Aguascalientes    1001
## 6      1 Aguascalientes    1001
```

```
class(base1)
```

```
## [1] "data.frame"
```

```
base2<-read.csv("Bases/concentradohogar.csv")
head(base2[1:3])
```

```
##      folioviv foliohog ubica_geo
## 1 100013605      1      1001
## 2 100013606      1      1001
## 3 100017801      1      1001
## 4 100017802      1      1001
## 5 100017803      1      1001
## 6 100017804      1      1001
```

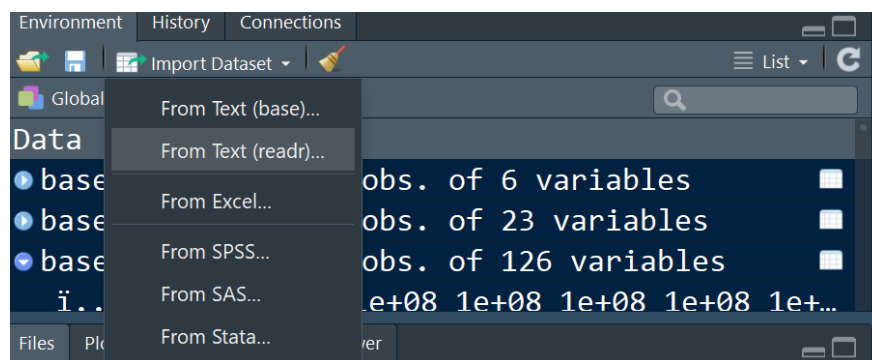
```
class (base2)
```

```
## [1] "data.frame"
```

Uso de botonera

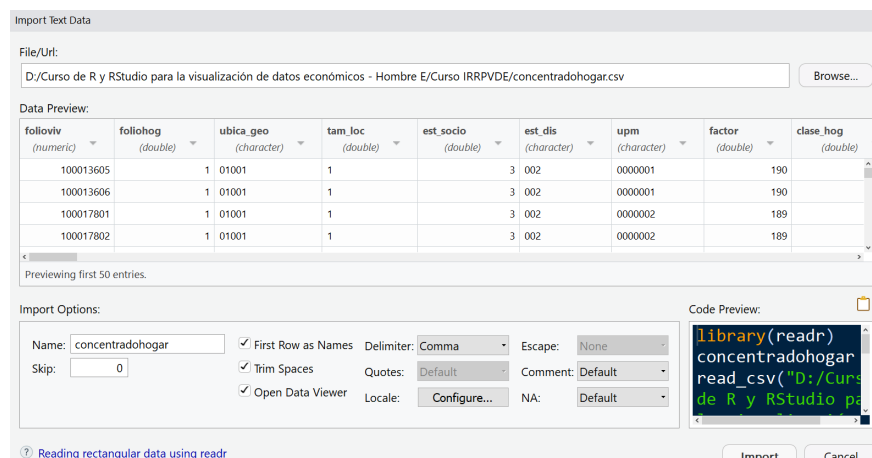
Otra manera de cargar la base, es mediante la botonera de RStudio. Para ello, tendremos que irnos la parte de “**import Dataset**” y dar click en “**From text**”:

Figura 19: Importando datos



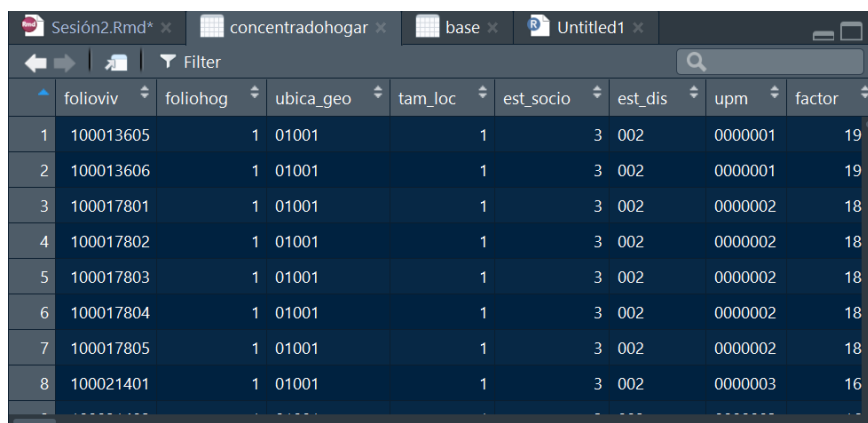
Posterior a ello, aparecera un menu, en el cual vamos a buscar la ruta del archivo:

Figura 20: Importando datos



Una vez seleccionado el archivo y al ejecutar, se desplegará una ventana como la siguiente:

Figura 21: Datos importados



| | folioviv | foliohog | ubica_geo | tam_loc | est_socio | est_dis | upm | factor |
|---|-----------|----------|-----------|---------|-----------|---------|---------|--------|
| 1 | 100013605 | 1 | 01001 | 1 | 3 | 002 | 0000001 | 19 |
| 2 | 100013606 | 1 | 01001 | 1 | 3 | 002 | 0000001 | 19 |
| 3 | 100017801 | 1 | 01001 | 1 | 3 | 002 | 0000002 | 18 |
| 4 | 100017802 | 1 | 01001 | 1 | 3 | 002 | 0000002 | 18 |
| 5 | 100017803 | 1 | 01001 | 1 | 3 | 002 | 0000002 | 18 |
| 6 | 100017804 | 1 | 01001 | 1 | 3 | 002 | 0000002 | 18 |
| 7 | 100017805 | 1 | 01001 | 1 | 3 | 002 | 0000002 | 18 |
| 8 | 100021401 | 1 | 01001 | 1 | 3 | 002 | 0000003 | 16 |

Lectura de archivos DTA.

Para cargar una base de datos con la extensión o formato DTA, se utiliza el comando **read_dta("nombre.dta")** de la librería **haven** con lo cual podremos cargar los datos de dicha base en esta extensión, sin embargo hay dos maneras de realizarla: 1) Generando el espacio de trabajo o enrutando y 2) Con el uso de la botonera.

Cargando bases DTA con el espacio previamente establecido

Para cargar una base de esta manera, se debe realizar lo que previamente se realizó en este documento (ya sea usando `setwd()` o un proyecto con lo archivos a cargar), entonces, lo único que se debe realizar es simplemente asignarle un nombre a la base y cargar el archivo:

```
library(haven)

baseE<-read_dta("Bases/EncuestaGenero.dta")
head(baseE[1:3])
```

```
## # A tibble: 6 x 3
##   con1   edo   muni
##   <dbl> <dbl> <dbl>
## 1     1     2     2
## 2     2     2     2
## 3     3     2     2
## 4     4     2     2
## 5     5     2     2
## 6     6     2     2
```

```
class(baseE)
```

```
## [1] "tbl_df"      "tbl"        "data.frame"
```

```
base2d<-read_dta("Bases/concentradohogar.dta")
head(base2d[1:3])
```

```
## # A tibble: 6 x 3
##   folioviv foliohog ubica_geo
##   <chr>    <chr>    <chr>
## 1 0100013605 1      01001
## 2 0100013606 1      01001
## 3 0100017801 1      01001
## 4 0100017802 1      01001
## 5 0100017803 1      01001
## 6 0100017804 1      01001
```

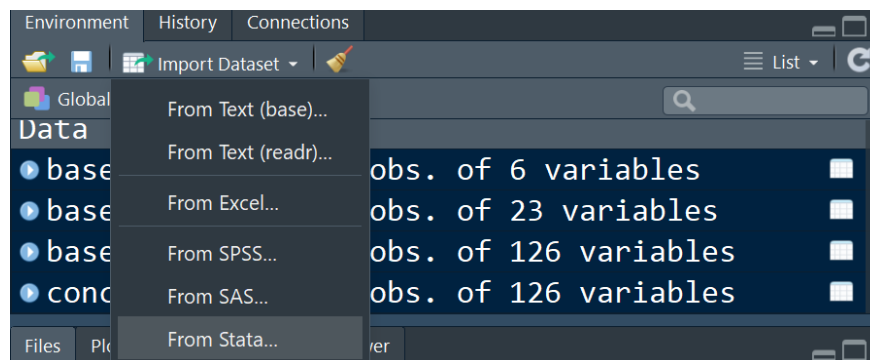
```
class(base2d)
```

```
## [1] "tbl_df"      "tbl"        "data.frame"
```

Uso de botonera

Otra manera de cargar la base, es mediante la botonera de RStudio. Para ello, tendremos que irnos la parte de **“import Dataset”** y dar click en **“From Stata”**:

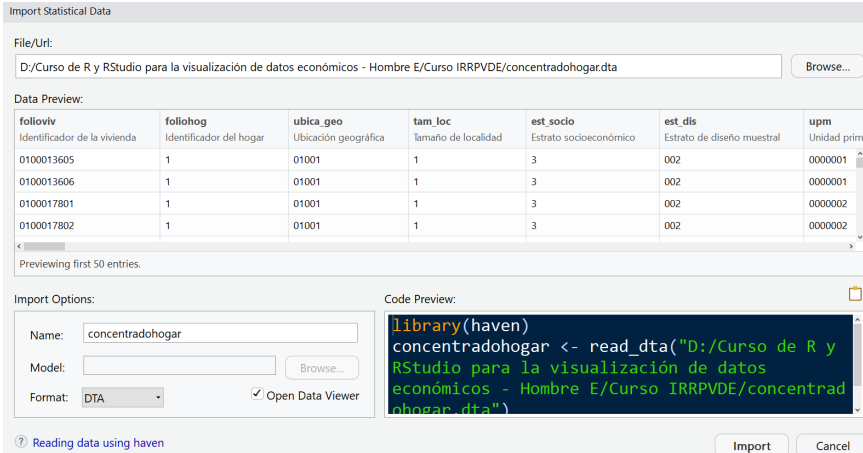
Figura 22: Importando datos



Posterior a ello, aparecerá un menu, en el cual vamos a buscar la ruta del archivo:

Una vez seleccionado el archivo y al ejecutar, se desplegará una ventana como la siguiente:

Figura 23: Importando datos



Import Statistical Data

File/Url:
D:/Curso de R y RStudio para la visualización de datos económicos - Hombre E/Curso IRRPVDE/concentradohogar.dta

Data Preview:

| folio_viv | folio_hog | ubica_geo | tam_loc | est_socio | est_dis | upm |
|------------------------------|-------------------------|----------------------|---------------------|------------------------|----------------------------|-----------------|
| Identificador de la vivienda | Identificador del hogar | Ubicación geográfica | Tamaño de localidad | Estrato socioeconómico | Estrato de diseño muestral | Unidad primaria |
| 0100013605 | 1 | 01001 | 1 | 3 | 002 | 0000001 |
| 0100013606 | 1 | 01001 | 1 | 3 | 002 | 0000001 |
| 0100017801 | 1 | 01001 | 1 | 3 | 002 | 0000002 |
| 0100017802 | 1 | 01001 | 1 | 3 | 002 | 0000002 |

Previewing first 50 entries.

Import Options:

Name: concentradohogar

Model:

Format: DTA

☒ Open Data Viewer

Code Preview:

```
library(haven)
concentradohogar <- read_dta("D:/Curso de R y
RStudio para la visualización de datos
económicos - Hombre E/Curso IRRPVDE/concentrad
ohogar.dta")
```

Reading data using haven

Import Cancel

Lectura de archivos SPSS.

Para cargar una base de datos con la extensión o formato DTA, se utiliza el comando **read_sav("nombre.sav")** de la librería **haven** con lo cual podremos cargar los datos de dicha base en esta extensión, sin embargo hay dos maneras de realizarla: 1) Generando el espacio de trabajo o enrutando y 2) Con el uso de la botonera.

Cargando bases SAV (SPSS) con el espacio previamente establecido

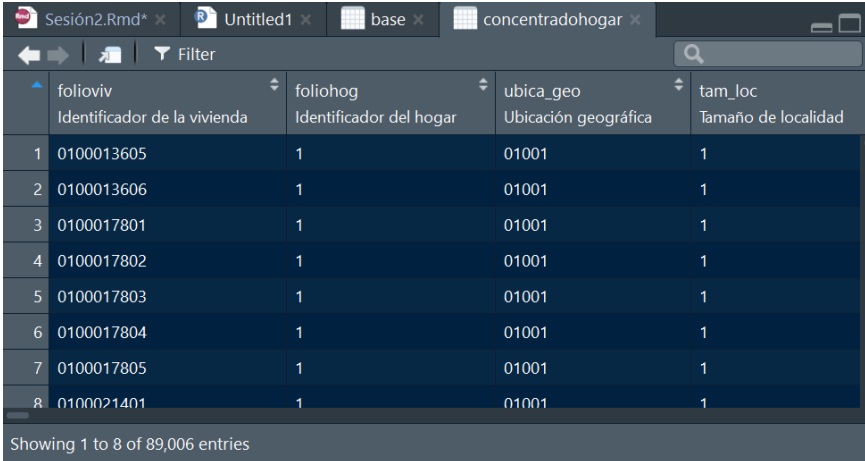
Para cargar una base de esta manera, se debe realizar lo que previamente se realizó en este documento (ya sea usando `setwd()` o un proyecto con lo archivos a cargar), entonces, lo único que se debe realizar es simplemente asignarle un nombre a la base y cargar el archivo:

```
library(haven)

baseS<-read_sav("Bases/EncuestaGenero.sav")
head(baseS[1:3])
```

```
## # A tibble: 6 x 3
##   con1   edo   muni
##   <dbl> <dbl> <dbl>
## 1     1     2     2
## 2     2     2     2
## 3     3     2     2
## 4     4     2     2
```

Figura 24: Datos importados



| | folioviv Identificador de la vivienda | foliohog Identificador del hogar | ubica_geo Ubicación geográfica | tam_loc Tamaño de localidad |
|---|--|-------------------------------------|-----------------------------------|--------------------------------|
| 1 | 0100013605 | 1 | 01001 | 1 |
| 2 | 0100013606 | 1 | 01001 | 1 |
| 3 | 0100017801 | 1 | 01001 | 1 |
| 4 | 0100017802 | 1 | 01001 | 1 |
| 5 | 0100017803 | 1 | 01001 | 1 |
| 6 | 0100017804 | 1 | 01001 | 1 |
| 7 | 0100017805 | 1 | 01001 | 1 |
| 8 | 0100021401 | 1 | 01001 | 1 |

```
## 5      5      2      2
## 6      6      2      2
```

```
class(baseS)
```

```
## [1] "tbl_df"      "tbl"        "data.frame"
```

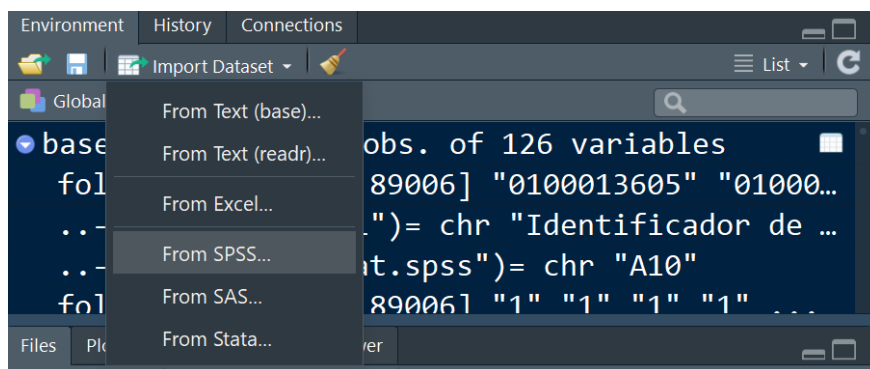
```
base2s<-read_sav("Bases/concentradohogar.sav")
head(base2s[1:3])
```

```
## # A tibble: 6 x 3
##   folioviv foliohog ubica_geo
##   <chr>      <chr>    <chr+lbl>
## 1 0100013605 1      01001 [Ags., Aguascalientes]
## 2 0100013606 1      01001 [Ags., Aguascalientes]
## 3 0100017801 1      01001 [Ags., Aguascalientes]
## 4 0100017802 1      01001 [Ags., Aguascalientes]
## 5 0100017803 1      01001 [Ags., Aguascalientes]
## 6 0100017804 1      01001 [Ags., Aguascalientes]
```

```
class(base2s)
```

```
## [1] "tbl_df"      "tbl"        "data.frame"
```

Figura 25: Importando datos

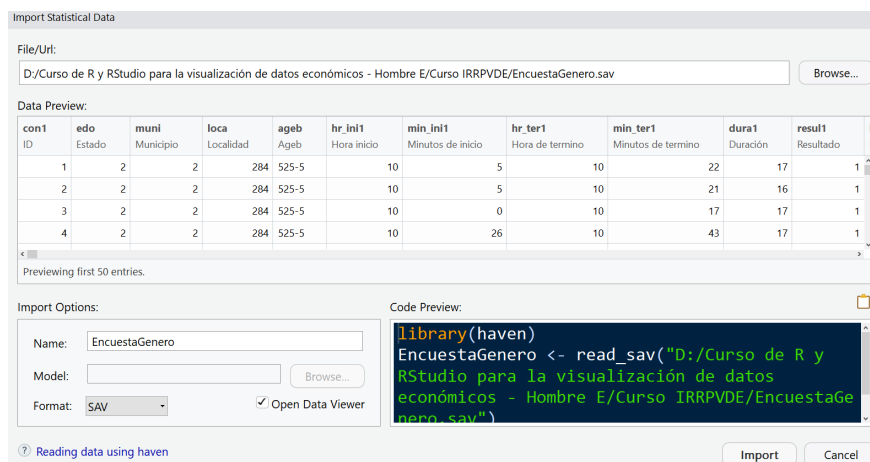


Uso de botonera

Otra manera de cargar la base, es mediante la botonera de RStudio. Para ello, tendremos que irnos la parte de **“import Dataset”** y dar click en **“From SPSS”**:

Posterior a ello, aparecera un menu, en el cual vamos a buscar la ruta del archivo:

Figura 26: Importando datos

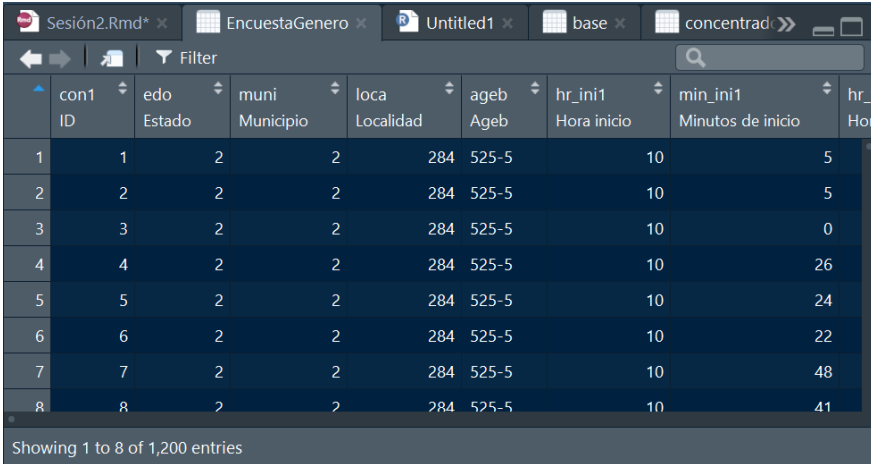


Una vez seleccionado el archivo y al ejecutar, se desplegará una ventana como la siguiente:

Lectura de archivos EXCEL.

Para la lectura de archivos en Excel (extensión .xlsx), se debe contar con dos paqueterias: **readxl** y **openxlsx**. Además se debe trabajar con un espacio definido, es

Figura 27: Datos importados



| | con1 ID | edo Estado | muni Municipio | loca Localidad | ageb Ageb | hr_ini1 Hora inicio | min_ini1 Minutos de inicio | hr_t Hora |
|---|---------|------------|----------------|----------------|-----------|---------------------|----------------------------|-----------|
| 1 | 1 | 2 | 2 | 284 | 525-5 | 10 | | 5 |
| 2 | 2 | 2 | 2 | 284 | 525-5 | 10 | | 5 |
| 3 | 3 | 2 | 2 | 284 | 525-5 | 10 | | 0 |
| 4 | 4 | 2 | 2 | 284 | 525-5 | 10 | | 26 |
| 5 | 5 | 2 | 2 | 284 | 525-5 | 10 | | 24 |
| 6 | 6 | 2 | 2 | 284 | 525-5 | 10 | | 22 |
| 7 | 7 | 2 | 2 | 284 | 525-5 | 10 | | 48 |
| 8 | 8 | 2 | 2 | 284 | 525-5 | 10 | | 41 |

Showing 1 to 8 of 1,200 entries

decir, ya debe existir un enrutamiento de una carpeta donde se contengan los archivos con la extensión **xlsx**.

Librería readxl.

Para ejemplificar el uso de **readxl**, se tiene que utilizar el comando **read_excel**("nombre-xlsx") y posteriormente cargará la base de datos en este formato:

```
# install.packages("readxl",
#                   dependencies=TRUE)

library(readxl)

basexlsx<-read_excel("Bases/concentradohogar.xlsx")

head(basexlsx[1:3])
```

```
## # A tibble: 6 x 3
##   folioviv foliohog ubica_geo
##   <dbl>     <dbl>     <dbl>
## 1 100013605         1         1001
## 2 100013606         1         1001
## 3 100017801         1         1001
## 4 100017802         1         1001
## 5 100017803         1         1001
## 6 100017804         1         1001
```

```
class (basexlsx)
```

```
## [1] "tbl_df"      "tbl"        "data.frame"
```

Librería openxlsx.

Para ejemplificar el uso de **openxlsx**, se tiene que utilizar el comando **read_excel**("nombre-xlsx") y posteriormente cargará la base de datos en este formato:

```
# install.packages("readxl",
#                   dependencies=TRUE)

library(openxlsx)

basexlsx2<-read.xlsx("Bases/MarginaciónMun.xlsx")

head(basexlsx2[1:3])
```

```
##      CVE_ENT      ENT CVE_MUN
## 1          1 Aguascalientes    1001
## 2          1 Aguascalientes    1001
## 3          1 Aguascalientes    1001
## 4          1 Aguascalientes    1001
## 5          1 Aguascalientes    1001
## 6          1 Aguascalientes    1001
```

```
class (basexlsx2)
```

```
## [1] "data.frame"
```

Lectura de archivos DBF

Para poder leer esta clase de archivos (en DBF) es necesario contar con la paquetería **foreign**, donde en ella se puede usar el comando **read.dbf** y en seguida se escribe el nombre del archivo en dicha extensión para poder realizar la carga del mismo.

Cargando bases DBF con el espacio previamente establecido

Para cargar una base en dicha extensión, no se puede usar la botonera, por lo que la definición de un espacio es la única manera, para ello se debe realizar lo que previamente se realizó en este documento (ya sea usando `setwd()` o un proyecto con lo archivos a cargar), entonces, lo único que se debe realizar es simplemente asignarle un nombre a la base y cargar el archivo:

```
library(foreign)

base3e<-read.dbf("Bases/concentradohogar.dbf")
head(base3e[1:3])
```

```
##      folioviv foliohog ubica_geo
## 1 0100013605      1      01001
## 2 0100013606      1      01001
## 3 0100017801      1      01001
## 4 0100017802      1      01001
## 5 0100017803      1      01001
## 6 0100017804      1      01001
```

```
class(base3e)
```

```
## [1] "data.frame"
```

Exportación de datos

En este tema, se mostrarán las formas en como se pueden exportar bases de datos en formato **data.frame** solo en las siguientes extensiones: **.csv**, **.dta**, **.sav** y **RData**. Para elaborar este proceso, se requiere definir un espacio de trabajo, donde se almacenen las bases exportadas, para ello se recomienda elaborar una carpeta dentro del proyecto, por lo que con R podemos realizarlo de una manera sencilla.

Para crear una carpeta, basta y sobra con usar **dir.create** y con la / se indica el nombre de dicha carpeta, tal y como se mostrará a continuación:

```
dir.create("Bases/bases_exportadas")
```

Ahora, para cambiar el directorio a esa carpeta y ahí almacenar las bases importadas, se utilizará **setwd** enseguida de **paste0** junto con **getwd** para copiar la ruta actual del proyecto y al último solo se anexa **/bases_exportadas** y como resultado todo lo que se exporte se encontrará en dicha carpeta.

Creado data.frame para la elaboración de ejemplos

Supongamos que hay una encuesta de 20 personas, donde las preguntas se encuentran en una escala likert (datos de 1 a 5), la información de almacena como vectores y al final se elabora un **data.frame** de dichos datos:

```
ID<-c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20)
Nombre<-c("Juan", "Carlos", "Susana", "Javier", "Valeria",
           "Alexis", "Alfredo", "Erandy", "Melissa", "Jose",
           "Diana", "Alin", "Andrea", "Luis", "Raymundo",
           "Samuel", "Armando", "Magdalena", "Joselin", "Ray")
Preguntal<-c(1,4,5,2,3,1,1,2,4,5,3,1,2,5,3,1,5,3,5,3)
Pregunta2<-c(4,4,5,2,3,1,2,5,2,3,1,4,2,5,3,1,2,5,4,3)
Pregunta3<-c(2,3,5,1,4,2,2,1,3,5,5,3,1,4,2,2,4,5,3,1)

Encuesta<-data.frame(cbind(ID, Nombre, Preguntal,
                           Pregunta2, Pregunta3))
```

Encuesta

| ## | ID | Nombre | Preguntal | Pregunta2 | Pregunta3 |
|-------|----|-----------|-----------|-----------|-----------|
| ## 1 | 1 | Juan | 1 | 4 | 2 |
| ## 2 | 2 | Carlos | 4 | 4 | 3 |
| ## 3 | 3 | Susana | 5 | 5 | 5 |
| ## 4 | 4 | Javier | 2 | 2 | 1 |
| ## 5 | 5 | Valeria | 3 | 3 | 4 |
| ## 6 | 6 | Alexis | 1 | 1 | 2 |
| ## 7 | 7 | Alfredo | 1 | 2 | 2 |
| ## 8 | 8 | Erandy | 2 | 5 | 1 |
| ## 9 | 9 | Melissa | 4 | 2 | 3 |
| ## 10 | 10 | Jose | 5 | 3 | 5 |
| ## 11 | 11 | Diana | 3 | 1 | 5 |
| ## 12 | 12 | Alin | 1 | 4 | 3 |
| ## 13 | 13 | Andrea | 2 | 2 | 1 |
| ## 14 | 14 | Luis | 5 | 5 | 4 |
| ## 15 | 15 | Raymundo | 3 | 3 | 2 |
| ## 16 | 16 | Samuel | 1 | 1 | 2 |
| ## 17 | 17 | Armando | 5 | 2 | 4 |
| ## 18 | 18 | Magdalena | 3 | 5 | 5 |
| ## 19 | 19 | Joselin | 5 | 4 | 3 |
| ## 20 | 20 | Ray | 3 | 3 | 1 |

Exportar en salida .csv

Para elaborar dicha salida, se requiere un **data.frame** ó **marco de datos**, ya sea de alguno que se haya manipulado previamente o bien, el vacío manual de datos y almacenados en dicho formato.

Una vez contado con lo anterior, se procede a exportar en csv con la función **write.csv**, la cual solicita el nombre del **data.frame** que se desea exportar y *file* para la salida con el nombre deseado del formato **.csv**, a continuación se muestra dicho proceso.

```
write.csv(Encuesta, file = "bases_exportadas/Encuesta.csv")
```

Exportar en salida .DTA

Para elaborar dicha salida, se requiere un **data.frame** ó **marco de datos**, ya sea de alguno que se haya manipulado previamente o bien, el vacío manual de datos y almacenados en dicho formato.

Una vez contado con lo anterior, se procede a exportar en .DTA con la función **write_dta** de la librería **haven**, la cual solicita el nombre del **data.frame** que se desea exportar y *la ruta* para la salida con el nombre deseado del formato **.dta**, a continuación se muestra dicho proceso.

```
library(haven)

write_dta(Encuesta, "bases_exportadas/Encuesta.dta")
```

Exportar en salida .SAV

Para elaborar dicha salida, se requiere un **data.frame** ó **marco de datos**, ya sea de alguno que se haya manipulado previamente o bien, el vacío manual de datos y almacenados en dicho formato.

Una vez contado con lo anterior, se procede a exportar en .DTA con la función **write_sav** de la librería **haven**, la cual solicita el nombre del **data.frame** que se desea exportar y *la ruta* para la salida con el nombre deseado del formato **.sav**, a continuación se muestra dicho proceso.

```
library(haven)

write_sav(Encuesta, "bases_exportadas/Encuesta.sav")
```

Exportar en salida .RData

Para elaborar dicha salida, se requiere un **data.frame** ó **marco de datos**, ya sea de alguno que se haya manipulado previamente o bien, el vacío manual de datos y almacenados en dicho formato.

Una vez contado con lo anterior, se procede a exportar en .RData con la función **save** de la librería **base**, la cual solicita el nombre del **data.frame** que se desea exportar y *la ruta* para la salida con el nombre deseado del formato **.RData**, a continuación se muestra dicho proceso.

```
save(Encuesta, file = "bases_exportadas/Encuesta.RData")
```