Tutorial

Data Envelopment Analysis with deaR

Version 1.0 (Español)

(Noviembre 2018)

Vicente Coll-Serrano(1)

Rafael Benítez(2)

Vicente J. Bolós(3)

(1) Economía Aplicada. Vicente.Coll@uv.es

Métodos Cuantitativos para la Medición de la Cultura (MC2)

(2) Matemáticas para la Economía y la Empresa. Benitez.Suarez@uv.es

(3) Matemáticas para la Economía y la Empresa. Vicente.Bolos@uv.es

Facultat d'Economia (www.uv.es/economia)

Universitat de València (España)

Índice

1. INTRODUCCIÓN.

deaR es un paquete de R (software libre) que permite ejecutar un amplio y variado número de modelos basados en el Análisis Envolvente de Datos.

Este tutorial está pensado para que los no usuarios de R puedan utilizar el paquete deaR, pero no es una manual de introducción a R[1].

Si eres usuario de R puedes ir directamente a las secciones 5 y 7.

Queremos que **deaR** se convierta en el software referente de aquellos investigadores, profesores, estudiantes y usuarios en general del Análisis Envolvente de Datos. Por esto, realmente agradeceremos cualquier comentario y sugerencia para mejorar **deaR.** También aceptamos sugerencias de modelos DEA y consideraremos su programación en nuevas versiones de **deaR**. En este sentido, nos gustaría anticipar que en una nueva versión de **deaR** se incluirán modelos o características no cubiertos actualmente como: DEA estocástico, Network DEA, índices de Malmquist (otras descomposiciones y bootstraping), valores negativos y extensión de inputs/outputs no deseables a otros modelos en los que ahora no están disponibles, etc.

En breve lanzaremos la versión **deaR Shiny** (es una aplicación web interactiva). Os mantendremos informados cuando la app esté disponible.

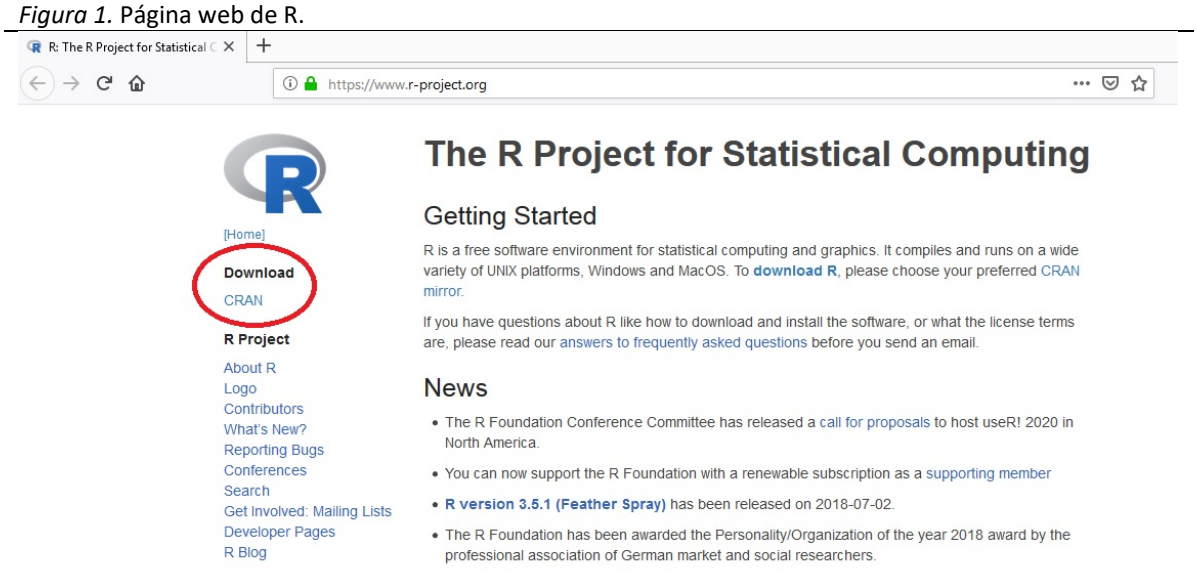
2. DESCARGAR E INSTALAR R Y RStudio.

Para poder utilizar **deaR** el primer paso consiste en descargarnos e instalar R y RStudio.

2.1. Instalación de R.

Para instalar R en nuestro ordenador, vamos a la página web de *R project*: http://www.r-project.org (ver Figura 1).

Figura 1. Página web de R.

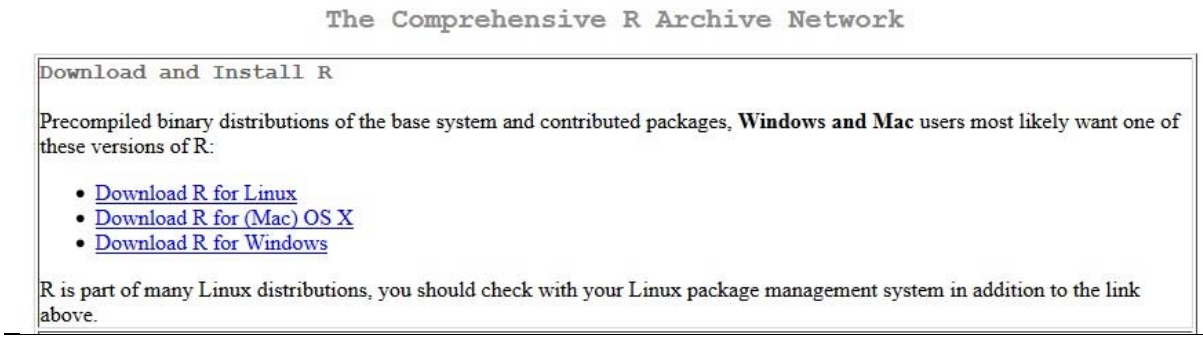


Para descargar R, hacemos clic en *CRAN* y seleccionamos el enlace del *"espejo"* más próximo a nuestra ubicación.

[1] En www.uv.es/vcoll está disponible un ***Curso introductorio de R*** (en español).
https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf (en Inglés).

Ahora, en función del sistema operativo de nuestro ordenador, seleccionamos la opción adecuada (ver Figura 2).

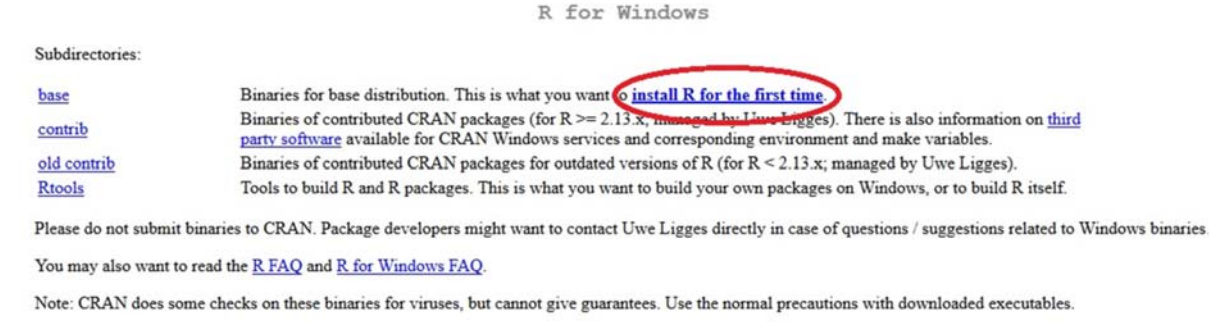
Figura 2. Versiones de R.



a) Instalar R en Windows.

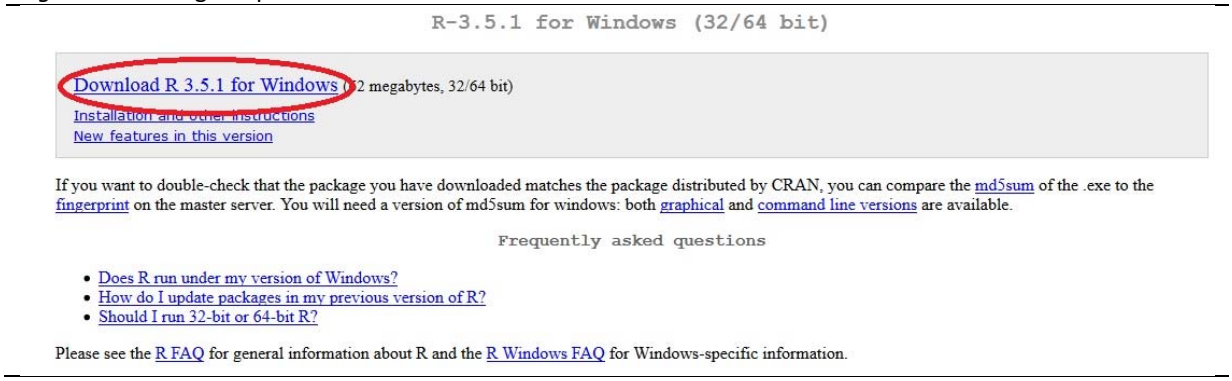
Al hacer clic sobre *Download R for Windows* iremos a la página que se reproduce más abajo. Hacemos clic sobre *install R for the first time* (ver Figura 3).

Figura 3. R para Windows.



En la siguiente ventana, hacemos clic sobre ***Download R 3.5.1 for Windows*** y guardamos el archivo de instalación (ver Figura 4).

Figura 4. Descargar R para Windows.

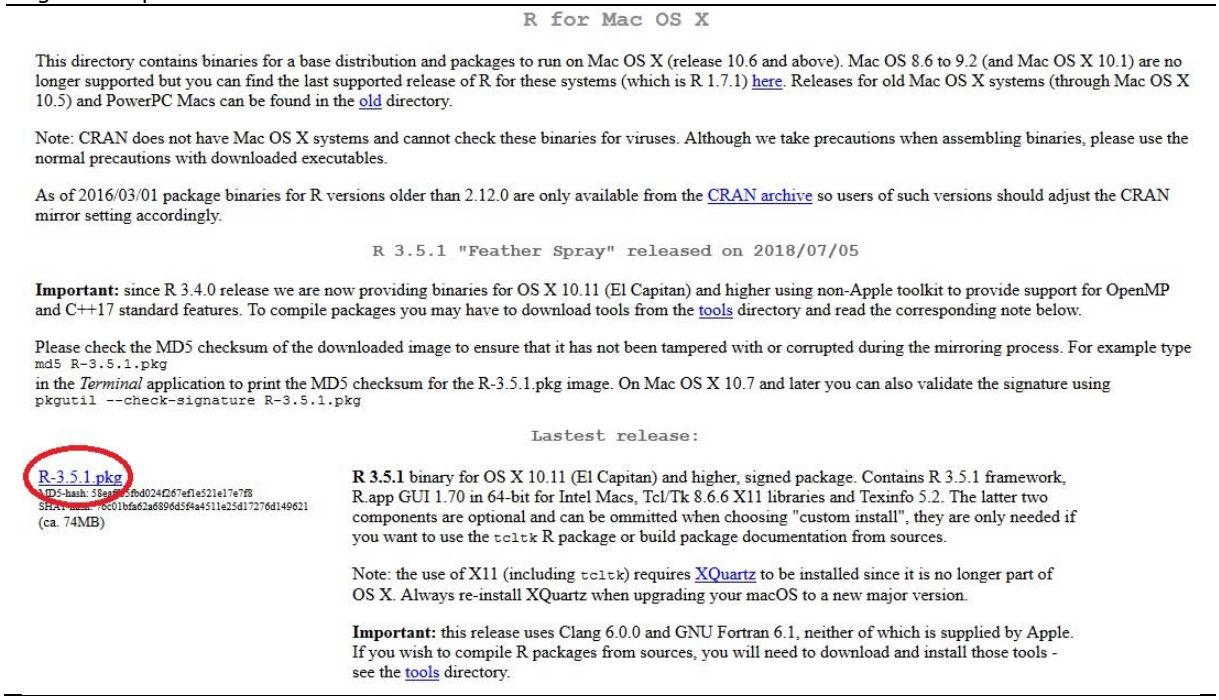


Abrimos el archivo descargado (haciendo doble clic sobre fichero) para instalar R.

b) Instalar R en Mac.

Al hacer clic sobre *Download R for (Mac) OS X* iremos a la página que se reproduce más abajo. Hacemos clic sobre ***R-3.5.1.pkg*** para descargarnos el fichero de instalación (ver Figura 5).

Figura 5. R para Mac.



R for Mac OS X

This directory contains binaries for a base distribution and packages to run on Mac OS X (release 10.6 and above). Mac OS 8.6 to 9.2 (and Mac OS X 10.1) are no longer supported but you can find the last supported release of R for these systems (which is R 1.7.1) here. Releases for old Mac OS X systems (through Mac OS X 10.5) and PowerPC Macs can be found in the old directory.

Note: CRAN does not have Mac OS X systems and cannot check these binaries for viruses. Although we take precautions when assembling binaries, please use the normal precautions with downloaded executables.

As of 2016/03/01 package binaries for R versions older than 2.12.0 are only available from the CRAN archive so users of such versions should adjust the CRAN mirror setting accordingly.

R 3.5.1 "Feather Spray" released on 2018/07/05

Important: since R 3.4.0 release we are now providing binaries for OS X 10.11 (El Capitan) and higher using non-Apple toolkit to provide support for OpenMP and C++17 standard features. To compile packages you may have to download tools from the tools directory and read the corresponding note below.

Please check the MD5 checksum of the downloaded image to ensure that it has not been tampered with or corrupted during the mirroring process. For example type `md5 R-3.5.1.pkg` in the *Terminal* application to print the MD5 checksum for the R-3.5.1.pkg image. On Mac OS X 10.7 and later you can also validate the signature using `pkgutil --check-signature R-3.5.1.pkg`

Lastest release:

R-3.5.1.pkg
MD5-hash: 58ea[illegible]5fbd024f267ef1e521e17e7f8
SHA1-hash: 76c01bfa62a6896d5f4a4511e25d17276d149621
(ca. 74MB)

R 3.5.1 binary for OS X 10.11 (El Capitan) and higher, signed package. Contains R 3.5.1 framework, R.app GUI 1.70 in 64-bit for Intel Macs, Tcl/Tk 8.6.6 X11 libraries and Texinfo 5.2. The latter two components are optional and can be ommitted when choosing "custom install", they are only needed if you want to use the `tcltk` R package or build package documentation from sources.

Note: the use of X11 (including `tcltk`) requires XQuartz to be installed since it is no longer part of OS X. Always re-install XQuartz when upgrading your macOS to a new major version.

Important: this release uses Clang 6.0.0 and GNU Fortran 6.1, neither of which is supplied by Apple. If you wish to compile R packages from sources, you will need to download and install those tools - see the tools directory.

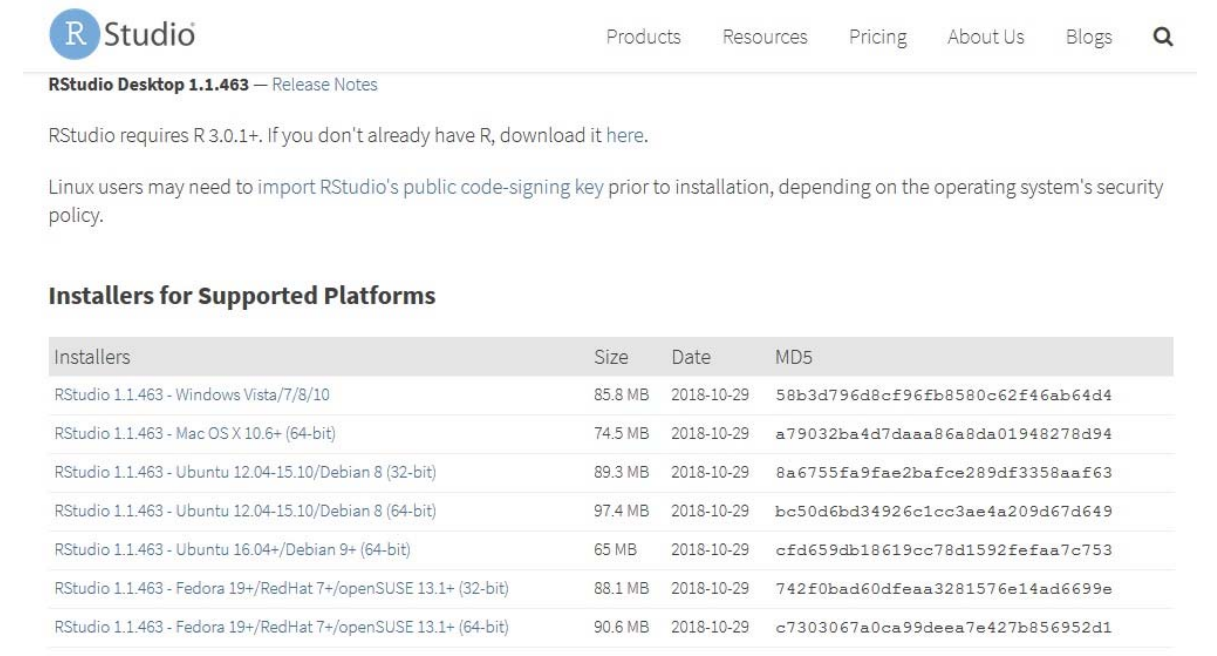
Abrimos *R-3.5.1.pkg* y seguimos las instrucciones para instalar R.

2.2. Instalar RStudio.

Una vez que hemos instalado R, descargamos RStudio desde el siguiente enlace (ver Figura 6):

https://www.rstudio.com/products/rstudio/download/#download

Figura 6. Descargar RStudio.



RStudio Products Resources Pricing About Us Blogs

RStudio Desktop 1.1.463 — Release Notes

RStudio requires R 3.0.1+. If you don't already have R, download it here.

Linux users may need to import RStudio's public code-signing key prior to installation, depending on the operating system's security policy.

Installers for Supported Platforms

Installers	Size	Date	MD5
RStudio 1.1.463 - Windows Vista/7/8/10	85.8 MB	2018-10-29	58b3d796d8cf96fb8580c62f46ab64d4
RStudio 1.1.463 - Mac OS X 10.6+ (64-bit)	74.5 MB	2018-10-29	a79032ba4d7daaa86a8da01948278d94
RStudio 1.1.463 - Ubuntu 12.04-15.10/Debian 8 (32-bit)	89.3 MB	2018-10-29	8a6755fa9fae2bafce289df3358aaf63
RStudio 1.1.463 - Ubuntu 12.04-15.10/Debian 8 (64-bit)	97.4 MB	2018-10-29	bc50d6bd34926c1cc3ae4a209d67d649
RStudio 1.1.463 - Ubuntu 16.04+/Debian 9+ (64-bit)	65 MB	2018-10-29	cfd659db18619cc78d1592fefaa7c753
RStudio 1.1.463 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (32-bit)	88.1 MB	2018-10-29	742f0bad60dfeaa3281576e14ad6699e
RStudio 1.1.463 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (64-bit)	90.6 MB	2018-10-29	c7303067a0ca99deea7e427b856952d1

Para descargar el fichero ejecutable, seleccionamos la opción según nuestro sistema operativo:

- RStudio 1.1.463 - Windows Vista/7/8/10

- RStudio 1.1.463 - Mac OS X 10.6+ (64-bit)

Primero, guardamos el fichero. A continuación, lo abrimos para instalar RStudio. Seguimos las instrucciones de instalación.

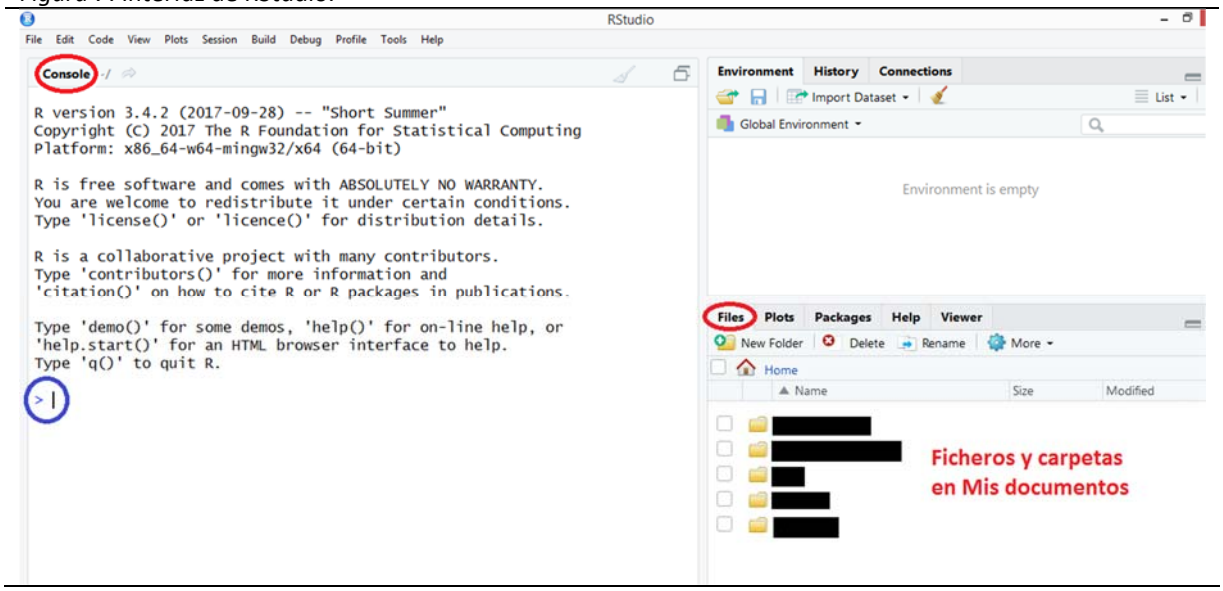
3. INICIAR RStudio

En general, trabajamos con la interfaz de RStudio antes que con la de R porque la primera es "más amigable".

Para iniciar RStudio, hacemos clic en el icono de RStudio: 

Al abrir RStudio deberíamos ver algo parecido a la Figura 7:

Figura 7. Interfaz de RStudio.



Por defecto, la consola se encuentra en el panel izquierdo. Primero aparece un texto informativo y después el prompt del sistema ("**>**"). ¿Vemos el cursor intermitente? Aquí es donde R espera que le demos instrucciones. Para ejecutar las instrucciones y obtener el resultado pulsamos *Enter*.

3.1. Crear un script.

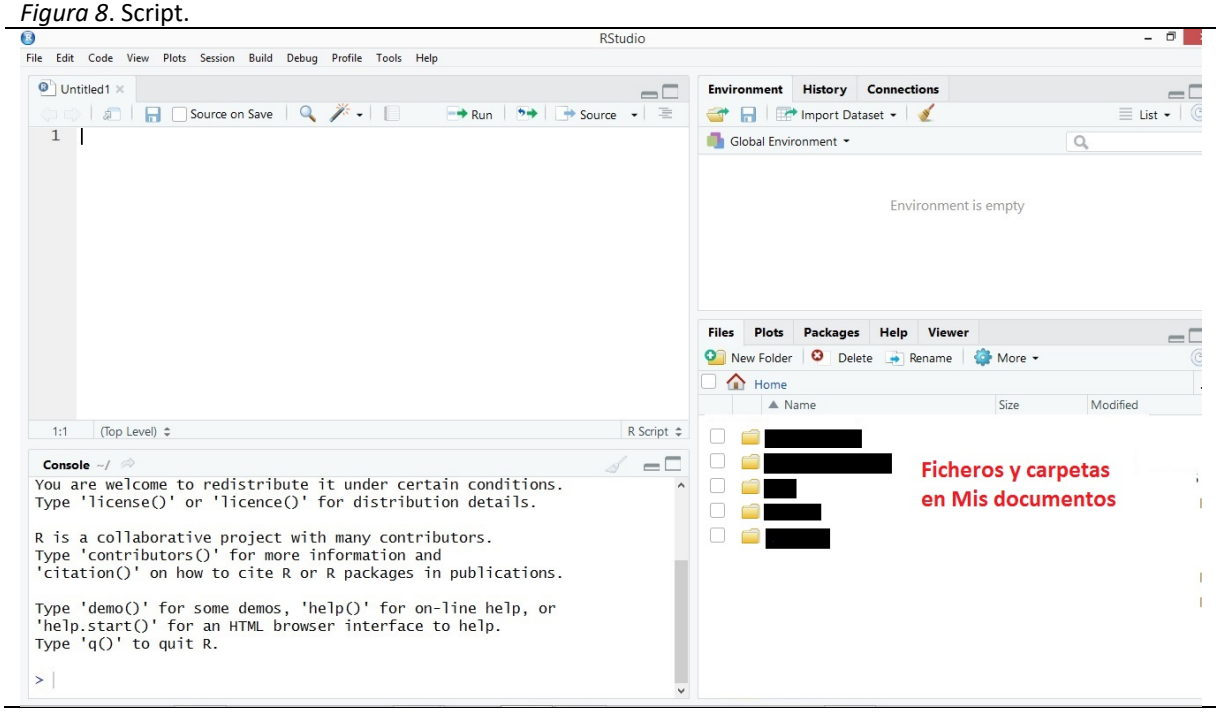
Trabajar en la *Consola* es muy limitado ya que en la *Consola* las instrucciones generalmente se escriben y ejecutan una por una. Lo habitual es trabajar con **scripts** o ficheros de instrucciones. Estos ficheros tienen la extensión "*.R*".

Para crear un script, seleccionamos *File > New File > R Script*

Ahora el panel del script se sitúa en la parte superior-izquierda de RStudio y la *Consola* en el panel inferior-izquierdo. Por defecto, el nombre del nuevo script es "*Untitled1*" (ver Figura 8).

En el script podemos escribir las instrucciones línea por línea. Las instrucciones las podemos ejecutar una a una; también podemos seleccionar todas (o algunas de) las instrucciones y ejecutar la selección. Para ejecutar una instrucción o selección de instrucciones hacemos clic en 

Figura 8. Script.



4. Como crear y trabajar con proyectos en RStudio

Una vez hemos arrancado RStudio, deberíamos establecer el directorio de trabajo para indicar a R y RStudio donde se encuentran nuestros datos, scripts, etc. Sin embargo, en nuestra opinión, la mejor opción es crear un proyecto. Al crear un proyecto todos los ficheros (de datos, scripts, etc.) y directorios quedan vinculados directamente al proyecto. Es decir, todo el trabajo que realicemos en un proyecto estará auto-contenido en el directorio del proyecto. De esta forma, fácilmente podemos compartir nuestro proyecto con alguien más o podemos copiar y pegar nuestro proyecto en otro ordenador, etc.

Nota importante:

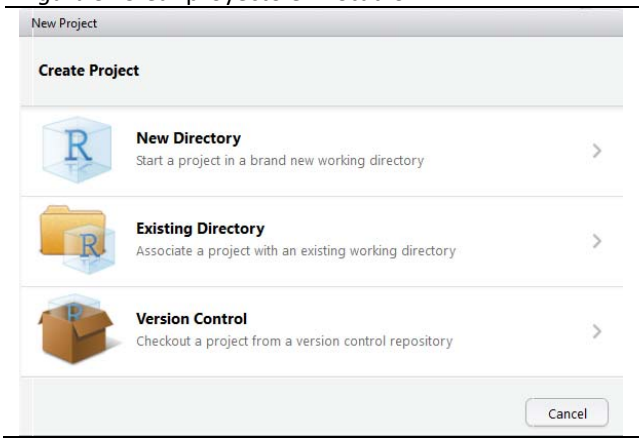
Creamos un proyecto para cada actividad/trabajo (por ejemplo, para cada artículo/paper).

Para organizar correctamente un proyecto, recomendamos crear un directorio específico para los datos, resultados, documentos de texto, etc.

4.1. Crear un proyecto.

Para crear un proyecto en RStudio, seleccionamos *File* > *New project*... Se abrirá una ventana similar a la que se muestra en la siguiente Figura.

Figura 9. Crear proyecto en RStudio.



Para crear un proyecto en un nuevo directorio, hacemos clic en el botón *New Directory*. Seguidamente, seleccionamos el tipo de proyecto, en nuestro caso *New Project*. Ahora, asignamos un nombre al directorio (carpeta) que se va a crear y que al mismo tiempo será el nombre del proyecto de R. Para terminar, hacemos clic en el botón *Create Project*. Al seguir este proceso se habrá creado una carpeta en *Documentos* y dentro encontraremos el archivo: "*nombre_carpeta.Rproj*".

Para crear un proyecto en una carpeta que ya existe, hacemos clic en el botón *Existing Directory* y después seleccionamos el directorio o carpeta ayudándonos del botón *Browse…*. Una vez elegida la carpeta, clicamos en el botón *Create Project*.

4.2. Abrir un proyecto.

Para abrir un proyecto, hacemos doble clic sobre el archivo con extensión "*.Rproj*". También podemos abrir el proyecto desde el menú de RStudio: *File > Open Project…*

Ventaja de los proyectos: cualquier fichero que guardemos trabajando en un proyecto se guardará en la carpeta del proyecto.

4.3. Información adicional.

Aquí os dejamos la dirección de dos lecturas cortas sobre los proyectos, la segunda incluye información sobre cómo crear proyectos.

- https://www.r-bloggers.com/managing-projects-using-rstudio/
- https://www.ssc.wisc.edu/sscc/pubs/RFR/RFR_Projects.html#projects

4.4. Cómo crear un proyecto.

Paso 1. Abrir RStudio

Paso 2. Seleccionar *File > New Project*

Paso 3. Seleccionar *New Directory*

Paso 4. *Project Type*: seleccionar *New Project*

Paso 5. *Directory name*: *Paper_1*

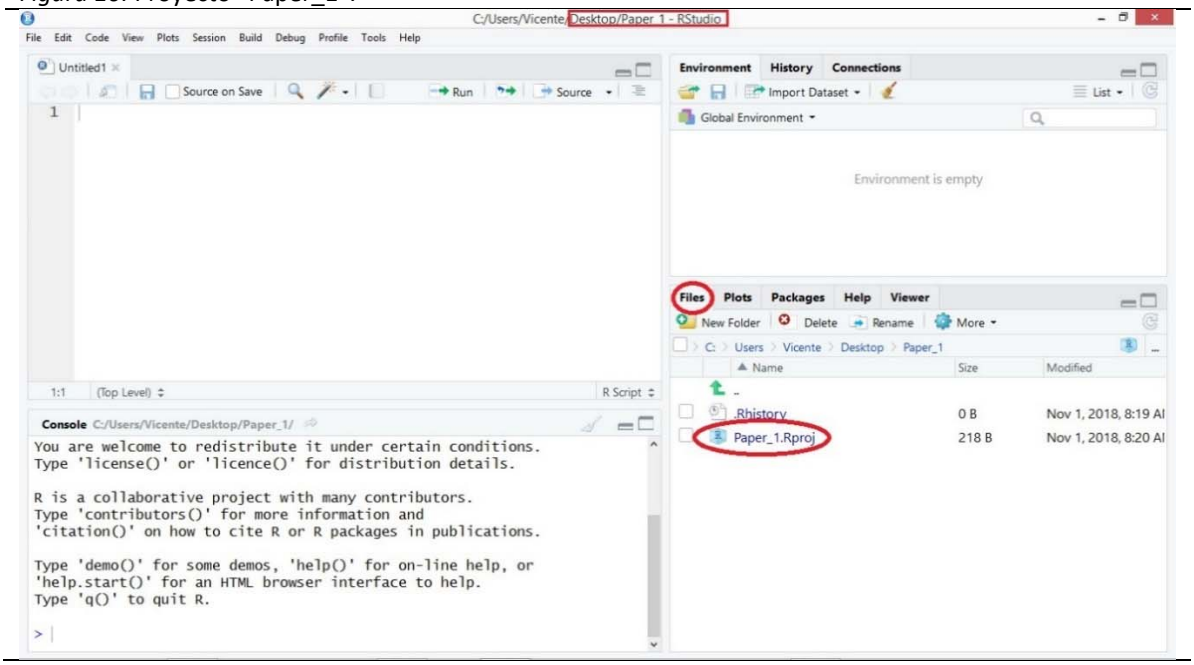
Paso 6: *Create project as subdirectory of*: seleccionar con el browse la ruta donde se creará el proyecto. Por ejemplo, seleccionar el escritorio.

Paso 7. Hace clic sobre *Create Project.*

Resultado: En el escritorio se habrá creado una carpeta con el nombre "*Paper_1*". Dentro de esta carpeta habrá dos ficheros (ver Figura 10):

- **Paper_1** (type: R Project)
- .Rhistoy (type: RHISTOY file)

Figura 10. Proyecto "Paper_1".



Nota importante:

La sesión de trabajo con un proyecto se abrirá siempre

haciendo doble clic sobre el fichero R Project

o

File > Open Project y seleccionando el proyecto

Se recomienda no trabajar simultáneamente con varios proyectos.

5. Instalar y cargar deaR

Abrir RStudio o el proyecto "*Paper_1*".

Seleccionar *File > New File > R script*

5.1. Instalar deaR.

Para instalar **deaR**, escribir en el script:

install.packages("deaR")

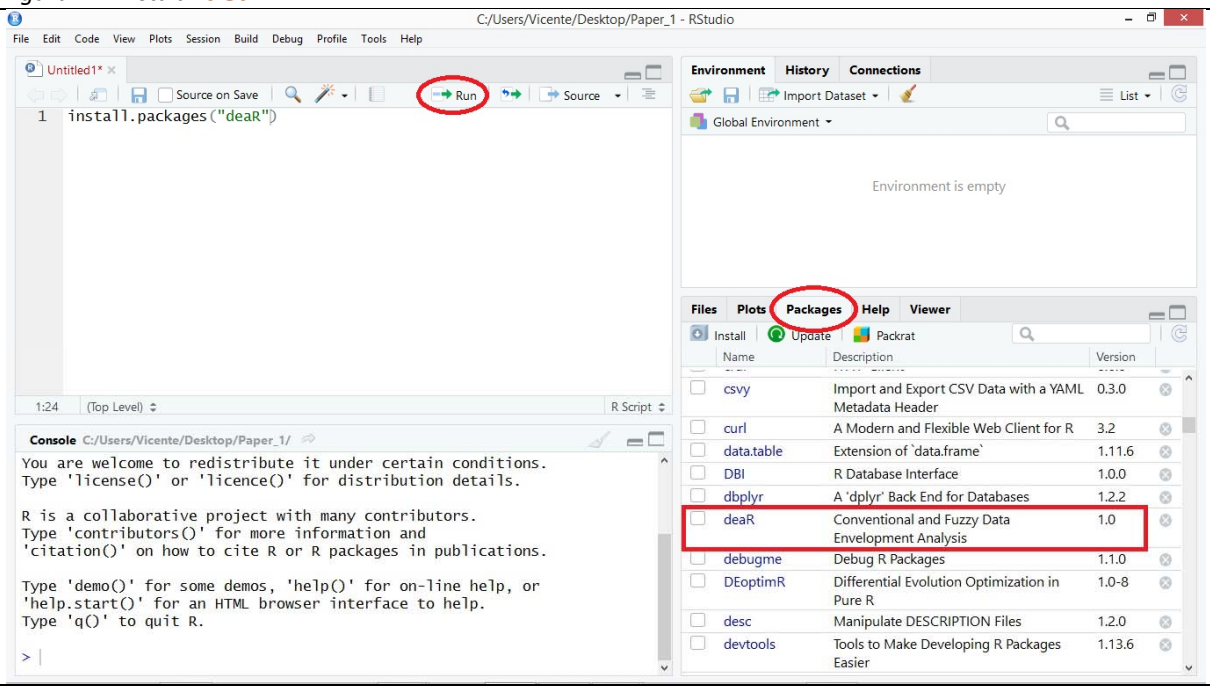
y ejecutar la instrucción: hacer clic en el botón Run: Run (ver Figura 11).

En la pestaña *Packages* situada en la ventana inferior derecha de RStudio, aparece el listado de todos los paquetes que el usuario tiene instalados en R. Ahora debería aparecer **deaR**.

Nota importante:

Solo hay que instalar **deaR** una vez. Las nuevas versiones de **deaR** las podemos obtener actualizando el paquete.

Figura 11. Instalar **deaR**.



5.2. Cargar deaR.

Una vez instalado **deaR**, o cualquier otro paquete de R, para usarlo primero hay que cargarlo (ver Figura 12). Para ello, escribimos en el script:

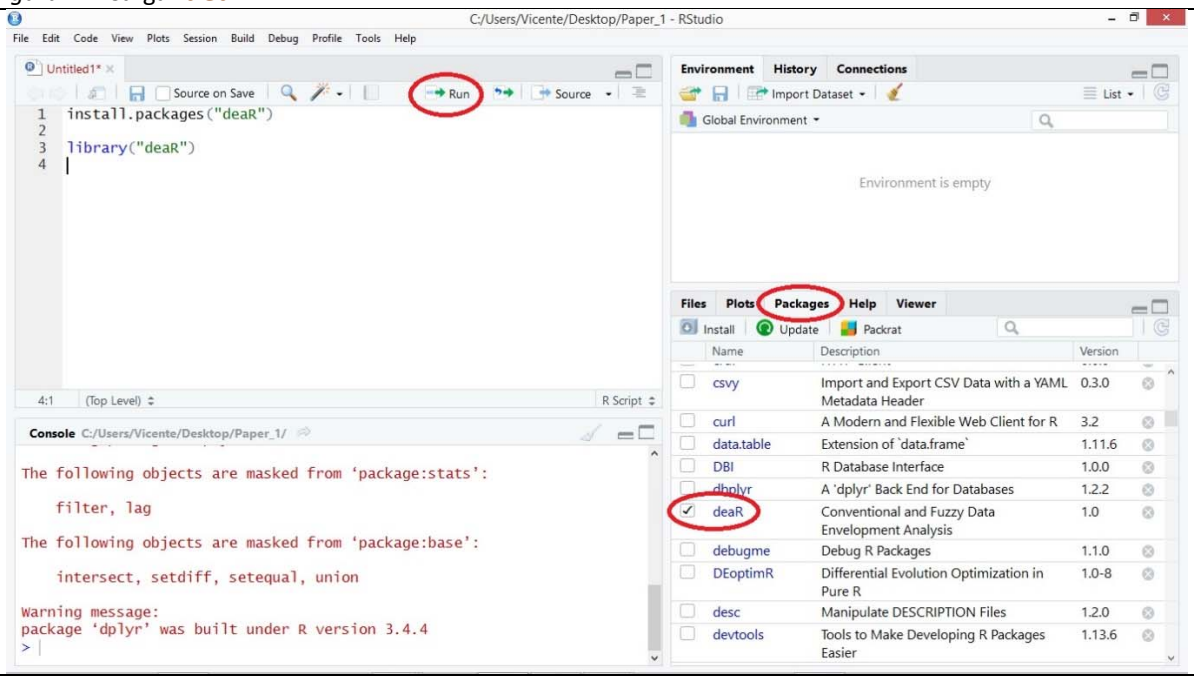
library("deaR")

y ejecutamos la instrucción haciendo clic en el botón *Run*: 

Nota importante:

Es necesario cargar **deaR** en cada sesión de trabajo.

Figura 12. Cargar **deaR**.



6. Guardar el script y cerrar la sesión de trabajo.

Para guardar el script, seleccionamos *File > Save as...*

Ahora, nombramos el fichero (por ejemplo: *sesion_1*) y hacemos clic en el botón *Save*. Por defecto el script se guardará en el proyecto de trabajo (“*Paper_1”*). Esta es una ventaja de trabajar con proyectos.

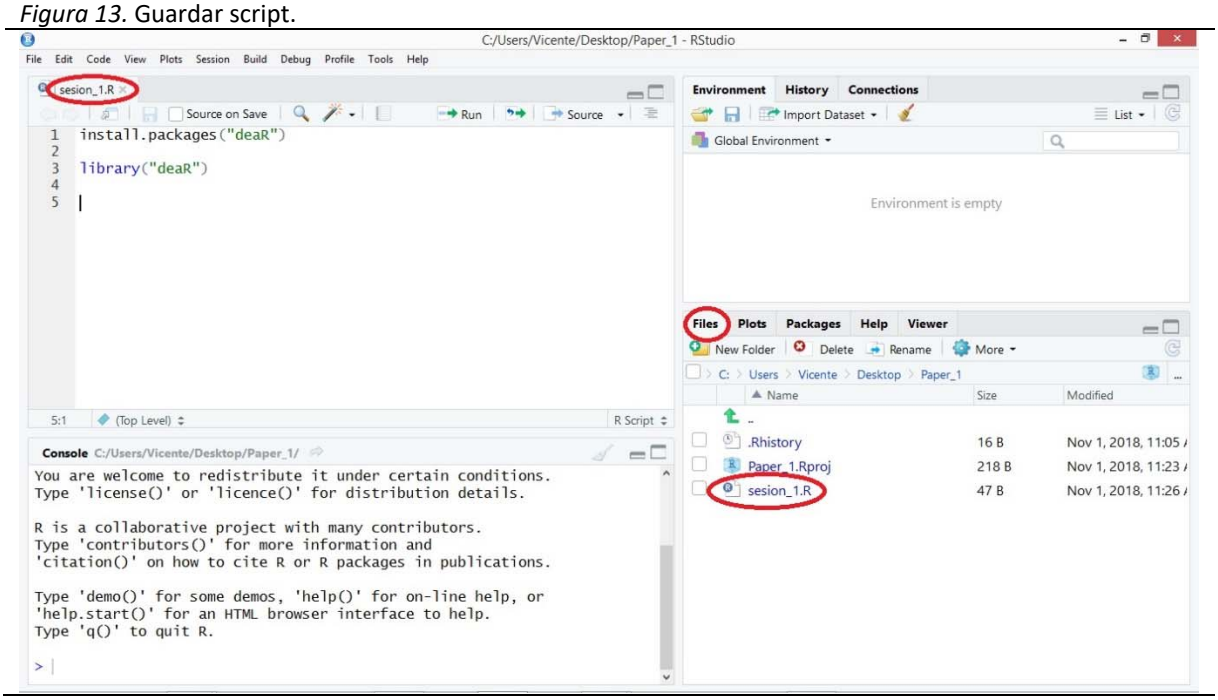
Observad que ahora el nombre del documento R (script) ha cambiado de “*Untitled1*” a “*sesion_1.R*” (ver Figura 13). Este fichero también aparece en el listado de ficheros que forman parte del proyecto “*Paper_1”.*

Nota: Hasta ahora, el contenido de este primer documento/script de R es irrelevante. Lo que es importante recordar es que en un script podemos guardar todas las instrucciones.

Nota importante:

Asignar nombres a los scripts que sean ilustrativos de su contenido.

Figura 13. Guardar script.



Si queremos cerrar el proyecto, pero permanecer en Rstudio: *File > Close project*

Si queremos cerrar el proyecto y salir de Rstudio*: File > Quit Session*

En este punto, vamos a **guardar *"sesion_1.R"*, cerrar el proyecto *"Paper_1"* y salir de RStudio.**

7. Análisis envolvente de datos con deaR.

Comenzamos abriendo el proyecto "*Paper_1*". Para ello, hacer doble clic sobre el fichero "*Paper_1.Rproj*". Se abrirá RStudio y el proyecto. Recuerda que también puedes abrir primero RStudio y después el proyecto (*File > Open Project*).

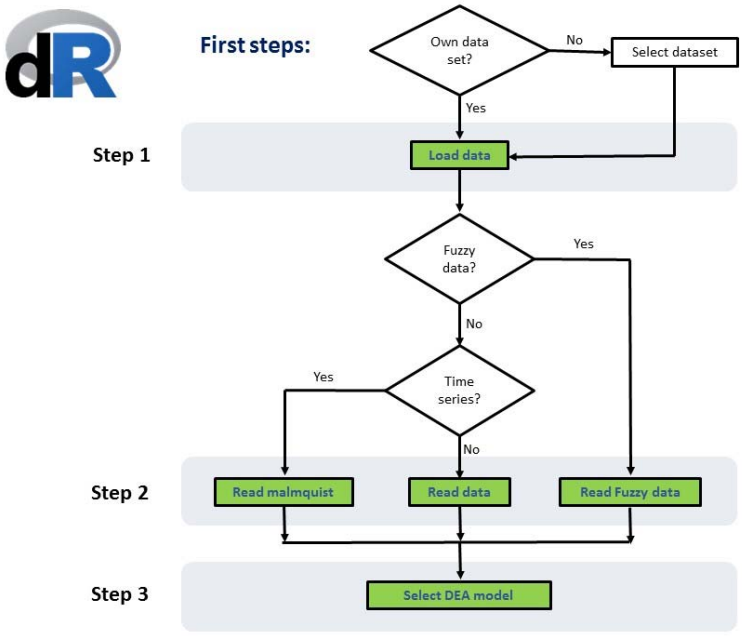
A continuación, creamos un nuevo script (*File > New File > R Script*). En este script escribimos la instrucción para cargar **deaR:**

library("deaR")

(Nota: Ejecutamos la instrucción con Run).

En el siguiente diagrama de flujo (ver Figura 14) podemos ver los pasos a seguir para realizar cualquier análisis DEA (Data Envelopment Analysis; en español, Análisis Envolvente de Datos) con **deaR**.

Figura 14. Pasos para utilizar **deaR**.



7.1. Importar datos en RStudio.

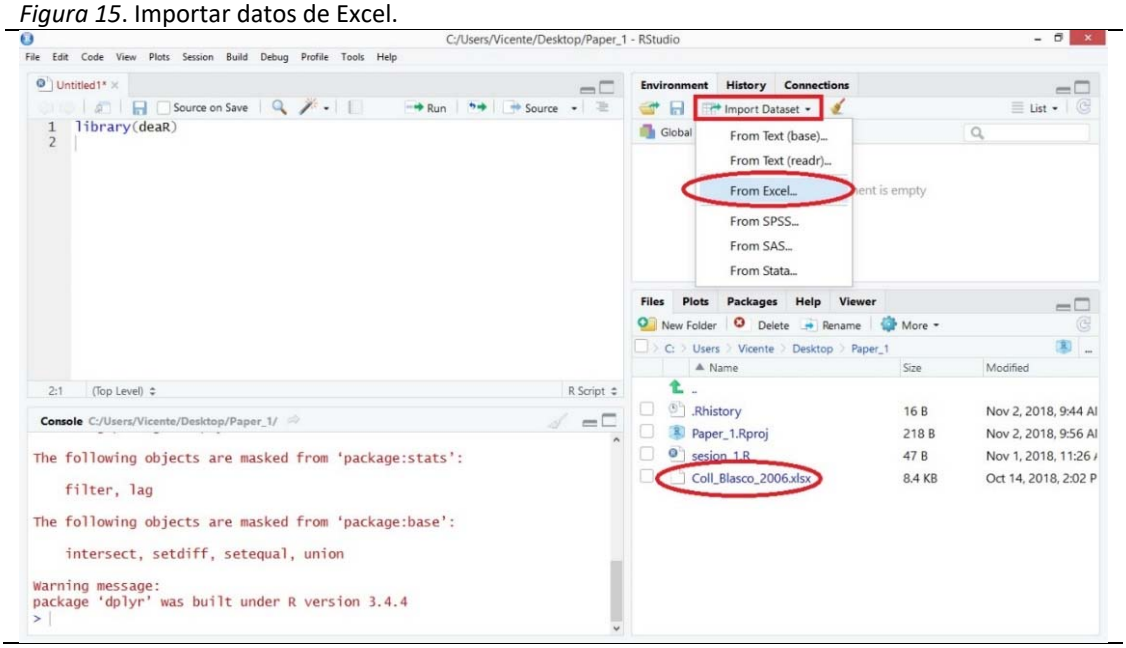
El primer paso consiste en cargar (importar) los datos que vamos a utilizar para analizar la eficiencia utilizando el análisis envolvente de datos. El usuario no familiarizado con R puede utilizar la opción *Import dataset*, es muy sencilla de utilizar. Vamos a verlo con el Ejemplo 1.

Ejemplo 1. Cargar en R datos de un fichero Excel:

1. Descarga los datos del ejemplo desde www.uv.es/vcoll/Coll_Blasco_2006.xlsx y guarda el fichero en la carpeta del proyecto "*Paper_1*".

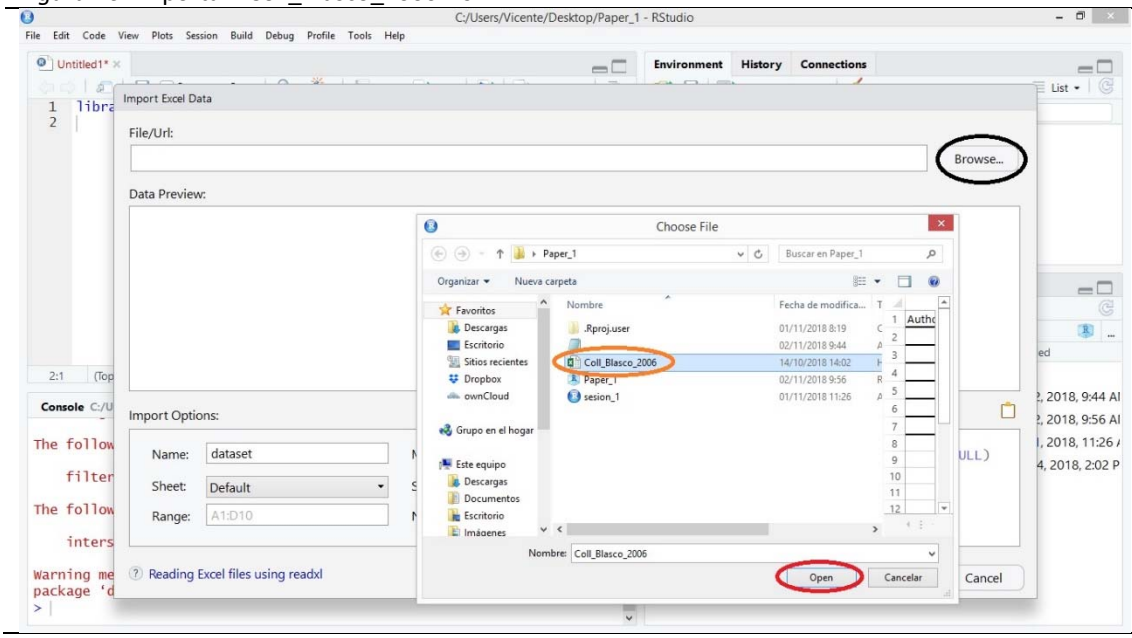
2. Del menú Environment situado en el panel superior izquierdo, seleccionamos la opción: *Import Dataset < From Excel* (ver Figura 15).

Figura 15. Importar datos de Excel.



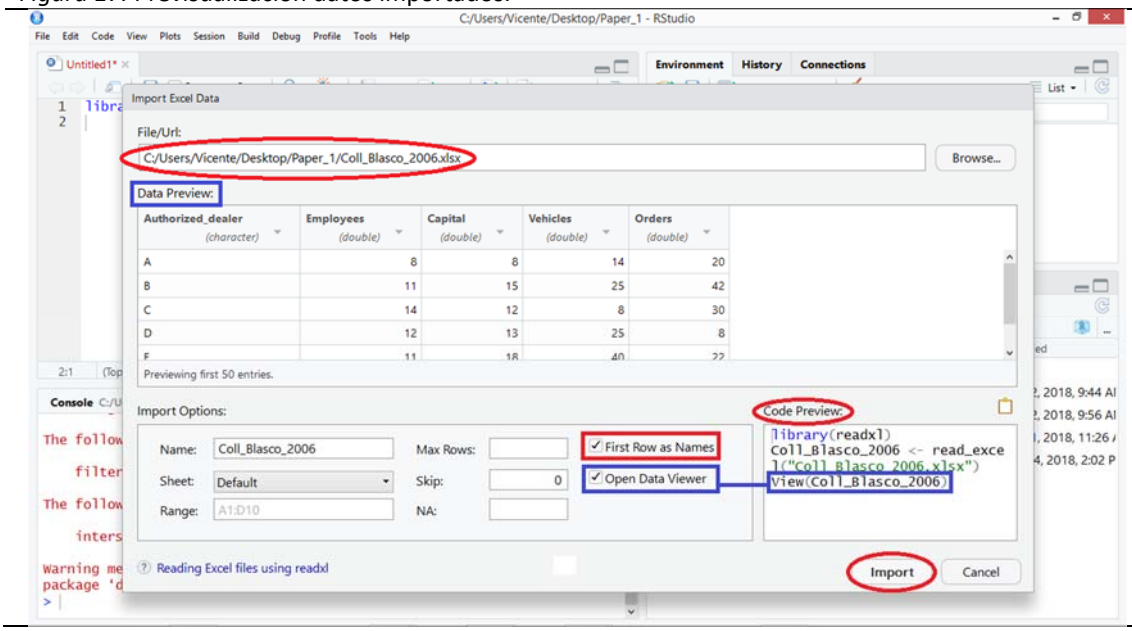
3. Se abre la ventana *Import Excel Data*. Hacemos clic sobre el botón *Browse* para elegir el fichero Excel ("*Coll_Blasco_2006.xlsx*")[2] y clicamos sobre *Open* (ver Figura 16).

Figura 16. Importar "Coll_Blasco_2006.xlsx".



4. Ahora podemos ver una previsualización de los datos "*Coll_Blasco_2006.xlsx*" y las opciones de importación que se han utilizado. En la parte inferior izquierda de esta ventana se muestra el código R utilizado por *Import Excel Data* (ver Figura 17). Como está seleccionada la opción *Open Data Viewer*[3], se abrirá el fichero de datos cuando terminemos de importarlos. Para ello, hacemos clic en *Import*.

Figura 17. Previsualización datos importados.

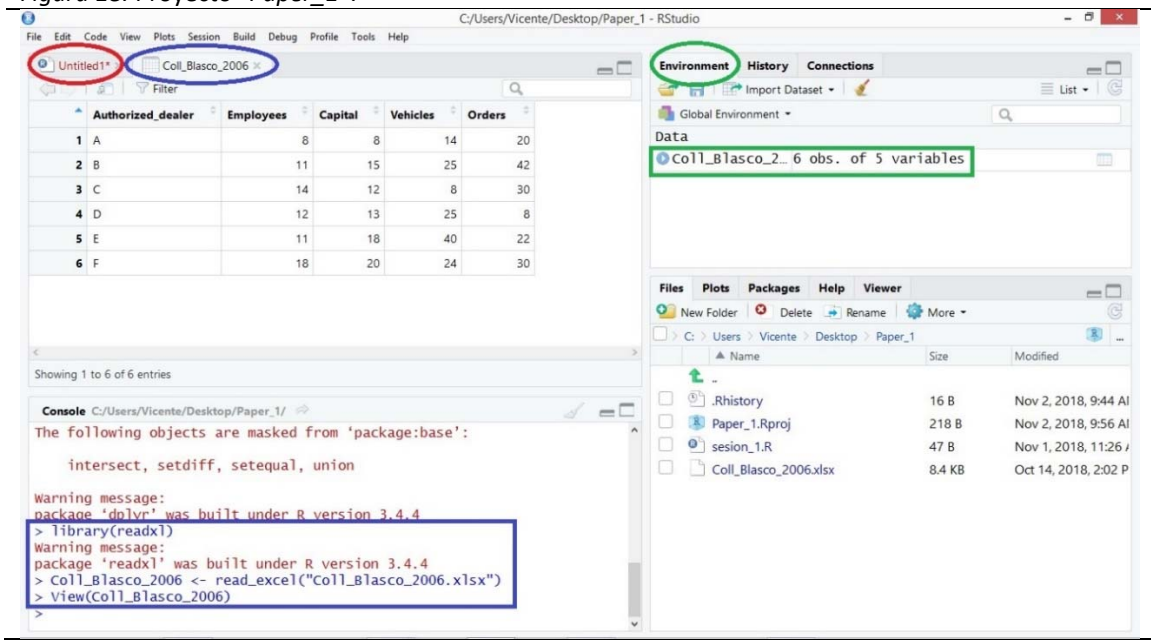


[2] Coll-Serrano, V.; Blasco-Blasco, O. (2006). Evaluación de la Eficiencia mediante el Análisis Envolvente de Datos. Introducción a los Modelos Básicos. www.eumed.net/libros/2006c/197/

[3] Corresponde al código R: **View(Coll_Blasco_2006)**

5. Al hacer clic en *Import* hemos regresado a la ventana principal del proyecto "*Paper_1*". En una nueva hoja aparecen los datos importados (tiene el mismo nombre que el dataset) (ver Figura 18). Además, en el menú *Environment* (situado en el panel superior derecho) se listan los objetos[4] que vamos creando en R. Ahora mismo tenemos sólo un objeto: "*Coll_Blasco_2006*". De hecho, este objeto es un dataframe que contiene 6 observaciones de 5 variables.

Figura 18. Proyecto "*Paper_1*".



En la *Consola* podemos ver el código utilizado para importar los datos (ver Figura 18):

- **library(readxl)** → carga el paquete readxl
- **Coll_Blasco_2006 <- read_excel("Coll_Blasco_2006.xlsx")** → la función read_excel() (del paquete readxl) lee el fichero de datos y asigna (<-) los datos al objeto "*Coll_Blasco_2006*".

  Muy importante:

  Terminología R: **A <- B** Esto significa que B (lo que sea B; un dataset, el resultado de una operación matemática, una función, etc.) es asignado (<-) al objeto A.

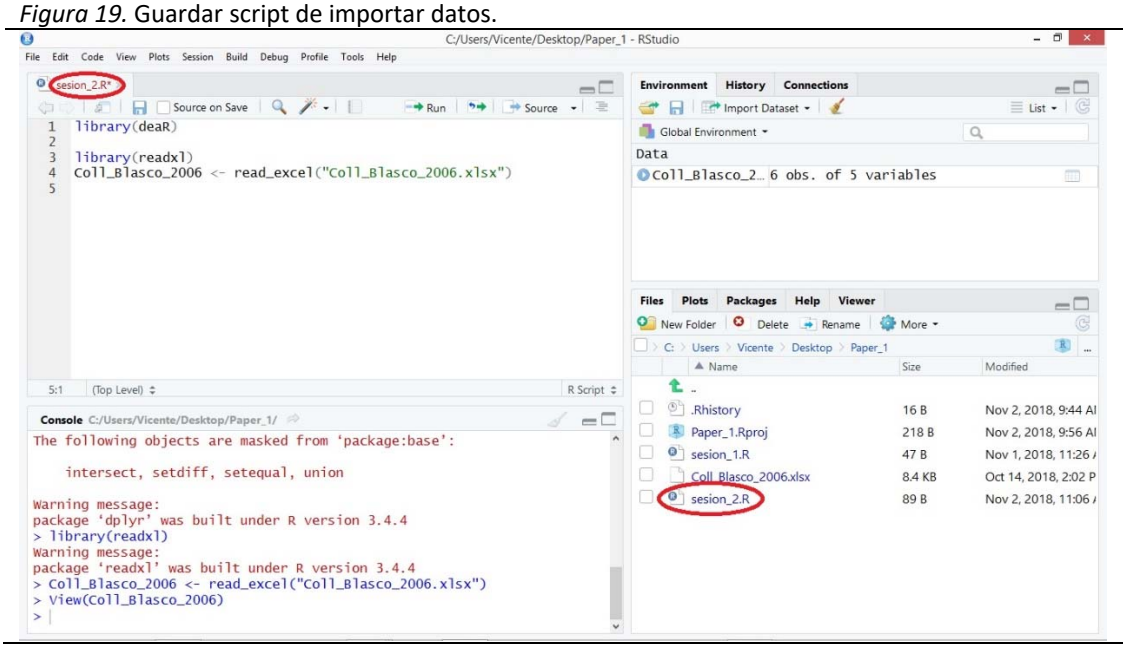
- **View(Coll_Blasco_2006)** → visualiza (View) el objeto "*Coll_Blasco_2006*".

6. Copiamos el código anterior en el script "*Untitled1*" y lo guardamos con el nombre "*sesion_2*".

Nuestro proyecto debería parecerse a la Figura que se encuentra más abajo (Figura 19).

[4] Todo en R es un objeto. Un objeto puede ser un dataset, una función, una instrucción, una matriz, etc.

Figura 19. Guardar script de importar datos.



En posteriores sesiones ya no será necesario realizar los pasos 1 a 5 porque tenemos el código en el fichero "*sesion_2.R*". Para importar nuevamente los datos de "*Coll_Blasco_2006.xlsx*" sólo tendremos que ejecutar el código de "*sesion_2.R*".

---

deaR también dispone de un importante número de datasets. Estos datos proceden de artículos ya publicados y son utilizados para replicar los resultados de los artículos. Pensamos que esto es un valor añadido que ofrece **deaR** a los investigadores y usuarios de DEA porque es una ayuda importante para los procesos de enseñanza-aprendizaje de la metodología DEA. Podemos consultar la estructura y fuente de los datos haciendo uso de la ayuda de **deaR**[5].

Veamos como cargar un dataset en el Ejemplo 2.

Ejemplo 2. Cargar datos de **deaR**.

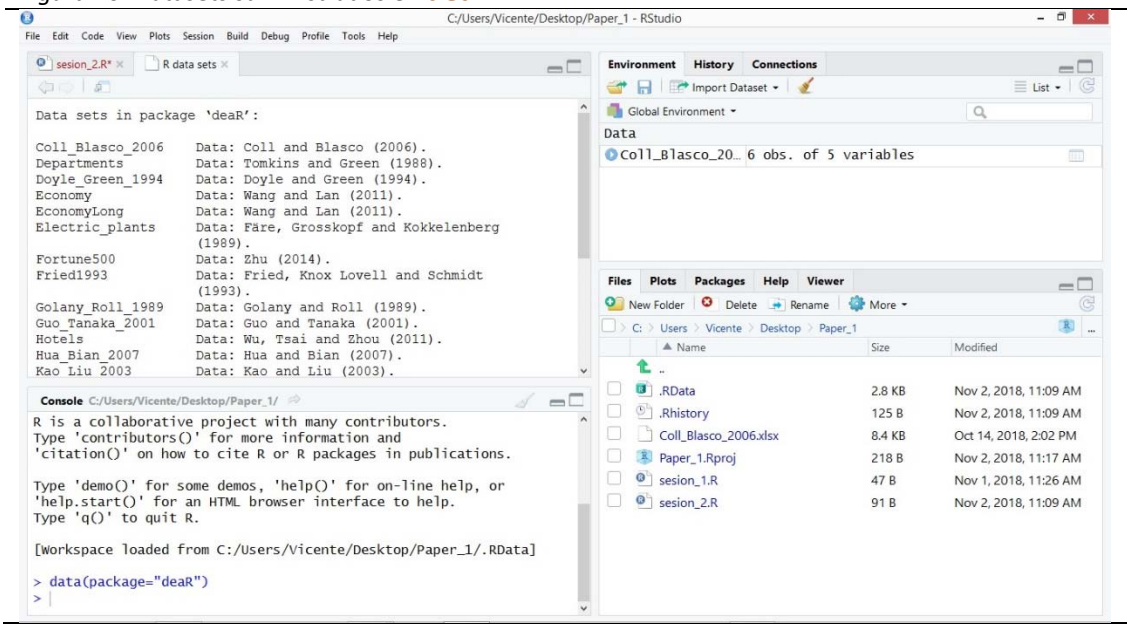
Para ver los datos disponibles en **deaR**, escribimos (ver Figura 20):

data(package="deaR")

y ejecutamos la instrucción.

[5] Obtener ayuda en R: https://www.r-project.org/help.html

Figura 20. Datasets suministrados en **deaR**.



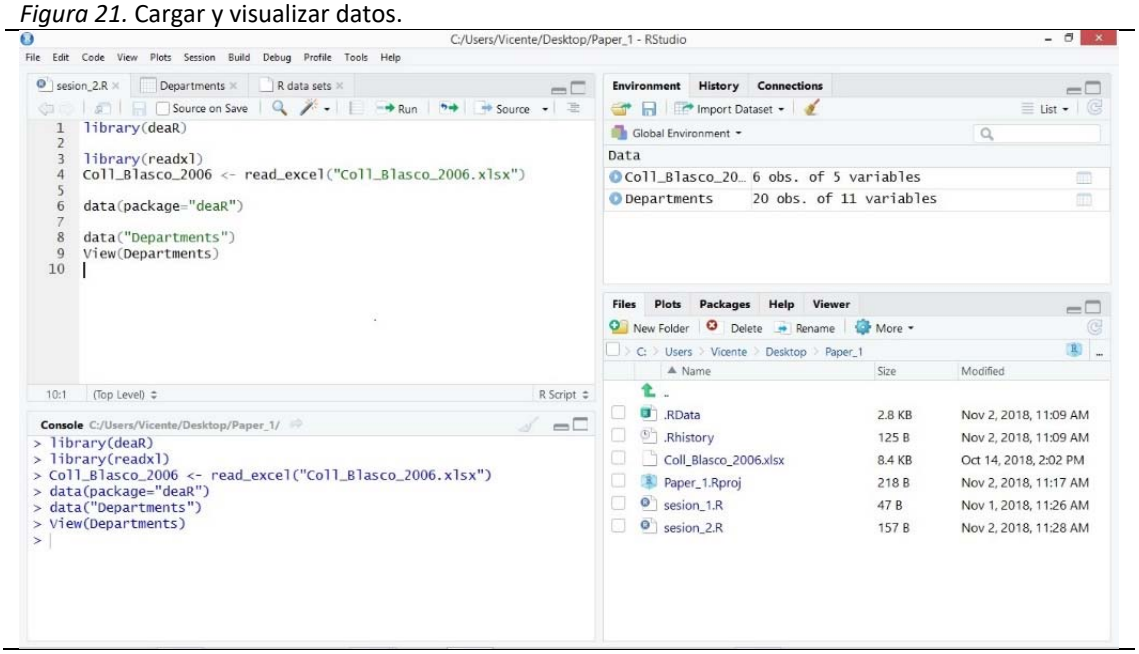
Para cargar un dataset se utiliza la función data(). Por ejemplo, vamos a cargar los datos de Tomkins y Green (1988)[6]. El nombre de este dataset es "*Departments*". Por tanto, escribimos (ver Figura 21):

data("Departments")

en el script "sesion2.R". Si queremos visualizar los datos escribimos (ver Figura 21):

View(Departments)

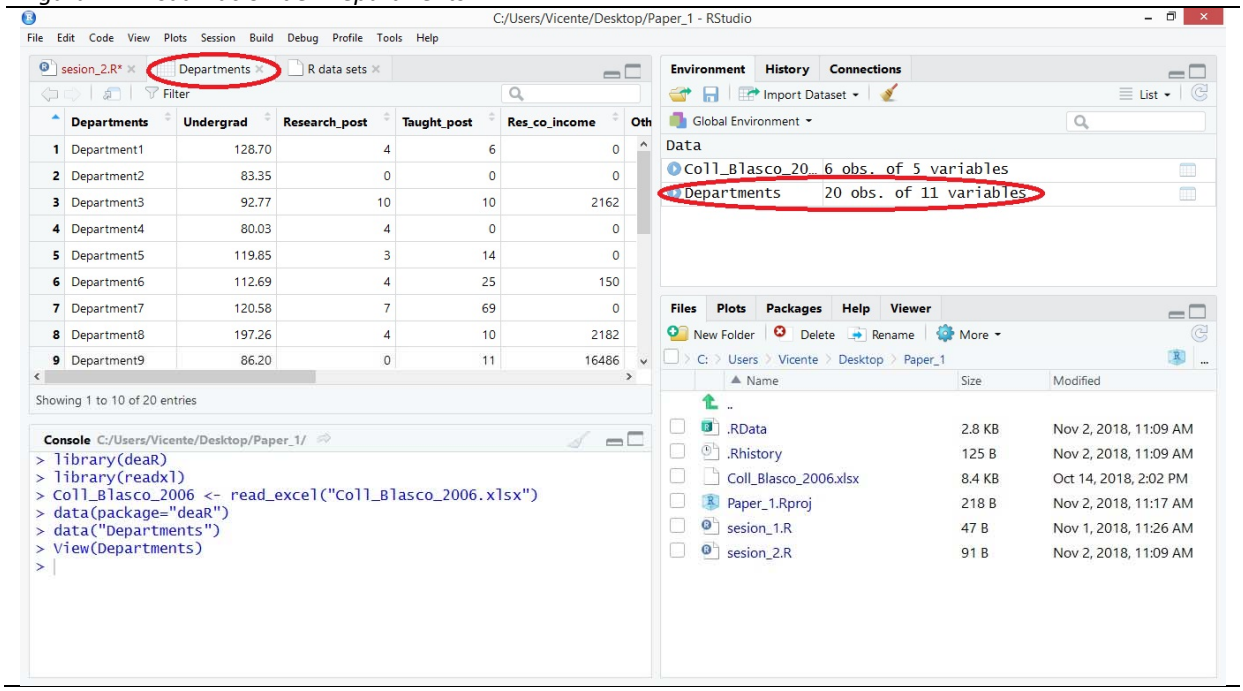
Figura 21. Cargar y visualizar datos.



Ejecutamos las instrucciones haciendo clic en Run (ver Figura 22).

[6] Tomkins, C.; Green, R. (1988). "An Experiment in the Use of Data Envelopment Analysis for Evaluating the Efficiency of UK University Departments of Accounting", Financial Accountability and Management, 4(2), 147-164. https://doi.org/10.1111/j.1468-0408.1988.tb00296.x

Figura 22. Visualización de "*Departments*".



Observad que ahora tenemos dos objetos ("*Coll_Blasco_2006*" y "*Departments*"). "*Departments*" es un dataframe que consta que 20 observaciones (DMUs) y 11 variables.

Ahora, **guardamos "*sesion_2.R*" y cerramos el proyecto "*Paper_1*". Salimos de RStudio.**

7.2. Adecuar los datos al formato de deaR.

Una vez cargados los datos, el siguiente paso es adecuarlos al formato que utiliza **deaR** para leerlos.

deaR dispone de tres funciones de lectura de datos. Cada formato de lectura responde a una determinad tipología de modelo DEA:

> **read_data()**: si vamos a ejecutar un modelo DEA convencional (o clásico).
>
> **read_malmquist()**: si vamos a aplicar el Índice de productividad de Malmquist.
>
> **read_data_fuzzy()**: si vamos a ejecutar un modelo DEA con datos inciertos (DEA fuzzy).

7.2.1. Ayuda de deaR[7].

Usando la ayuda de **deaR** podemos aprender cómo usar una función específica. Accediendo a la ayuda de **deaR** podemos leer sobre los argumentos de la función o sobre ejemplos sobre cómo usar la función. Vamos a ver cómo usar la función de ayuda con el Ejemplo 3.

[7] Getting help with R: https://www.r-project.org/help.html

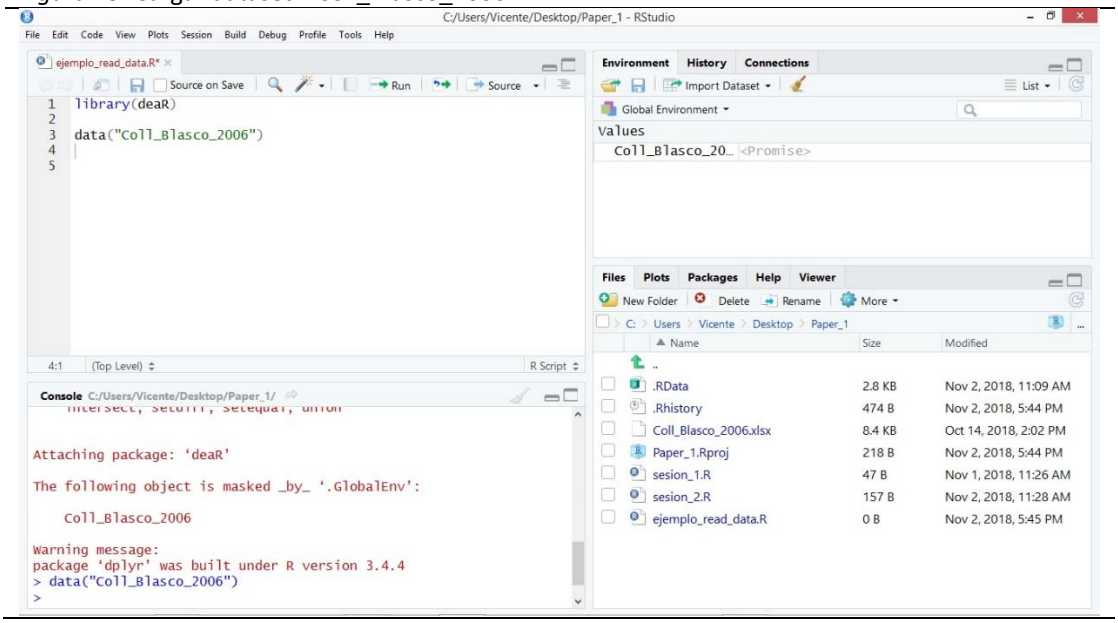
Ejemplo 3. Usando la ayuda de **deaR**.

1. Abrimos el proyecto "*Paper_1*" y creamos un nuevo script. Llamamos a este script: "*ejemplo_read_data*"

2. Cargamos el paquete **deaR**

3. Cargamos los datos de **deaR**: "*Coll_Blasco_2006*" (ver Figura 23).

Figura 23. Cargar dataset: "Coll_Blasco_2006"

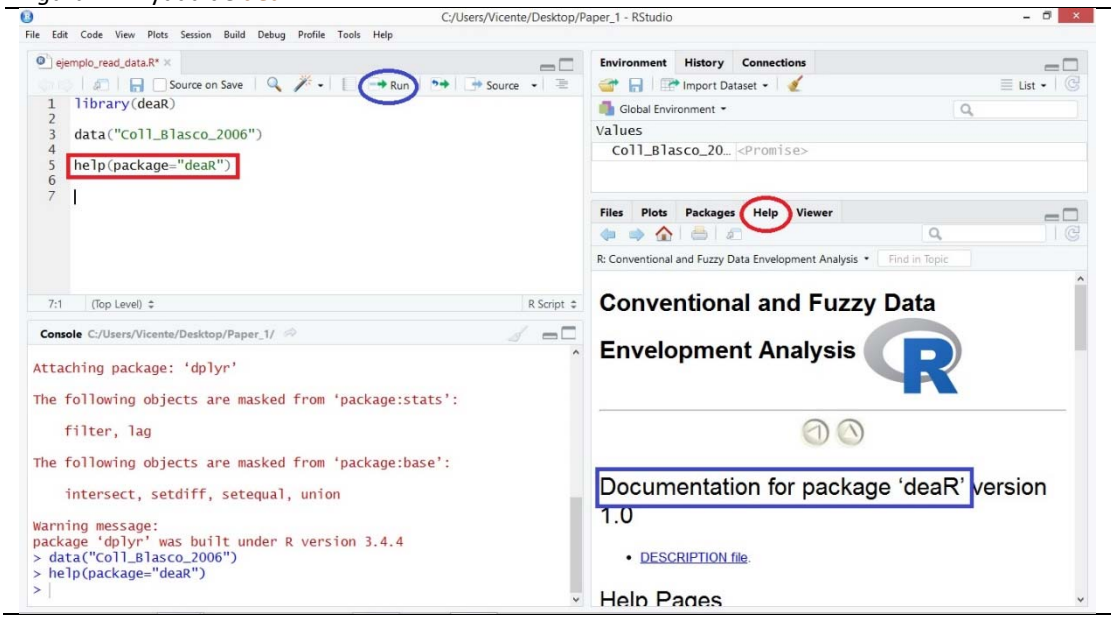


Para acceder a la documentación de las funciones de **deaR** utilizamos la función help(). Escribimos:

help(package="deaR").

En el menú *Help* (ver ventana inferior izquierda) aparecerá un listado con todas las funciones y datos de **deaR**. Es la documentación de **deaR** (ver Figura 24).

Figura 24. Ayuda de **deaR**.



7.2.2. La función read_data().

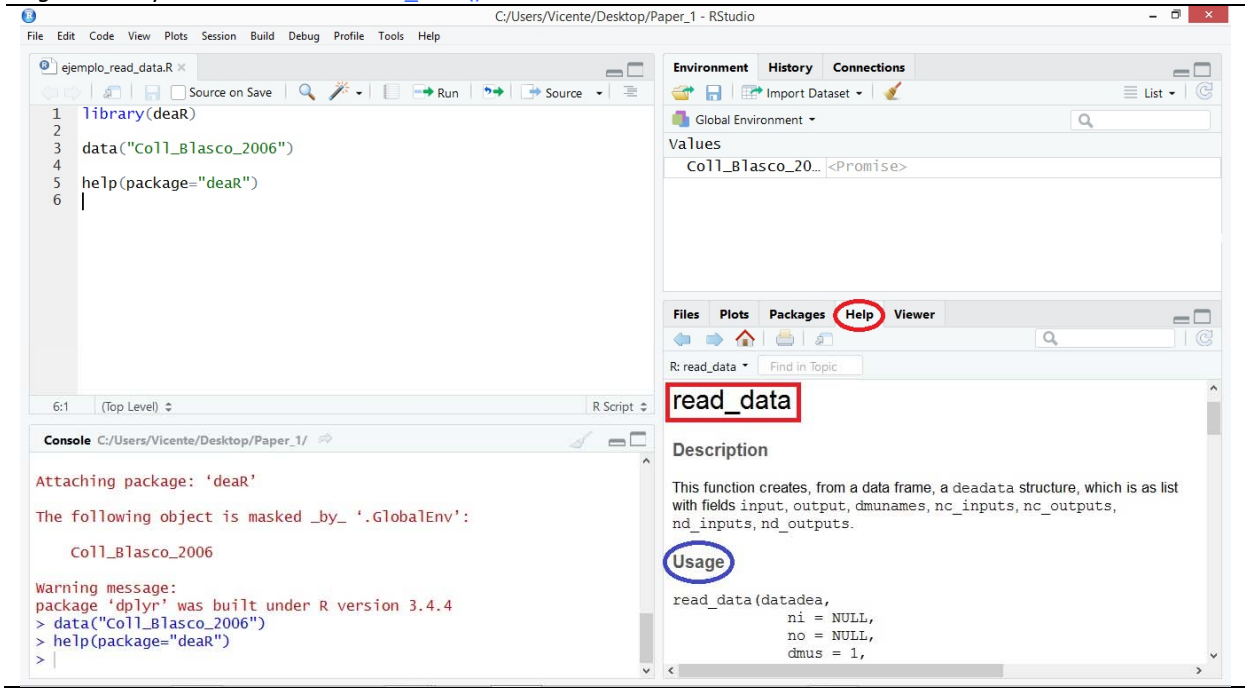
Una vez hemos accedido a la documentación (ayuda) de **deaR**, haciendo clic sobre "*read_data*" obtendremos la ayuda específica de esta función. También podemos obtener el mismo resultado si en el script escribimos

help(read_data) (o **?read_data**)

y ejecutamos la instrucción.

En la sección *Usage* se muestran todos los argumentos de la función read_data(), y son explicados brevemente en la sección *Arguments* (ver Figura 25).

Figura 25. Ayuda de la función read_data().



La función read_data() tiene los siguientes argumentos:

- *datadea*: Se refiere al conjunto de datos a analizar (tiene que ser una dataframe).
- *dmus*: Indicar el número de la columna donde se encuentran las DMUs. Por defecto **deaR** considera que las DMUs se encuentran en la primera columna.
- *ni*: Es el número de inputs.
- *no*: Es el número de outputs.
- *inputs*: En lugar de indicar el número de inputs se puede indicar el número de las columnas donde se encuentran los inputs.
- *outputs*: En lugar de indicar el número de outputs se puede indicar el número de las columnas donde se encuentran los outputs.
- *nc_inputs*: Si entre los inputs hay inputs no controlables se puede indicar qué input es no controlable.
- *nc_outputs*: Si entre los outputs hay outputs no controlables se puede indicar qué output es no controlable.

- *nd_inputs*: Si entre los inputs hay inputs no discrecionales se puede indicar qué input es no discrecional.
- *nd_outputs*: Si entre los outputs hay outputs no discrecionales se puede indicar qué output es no discrecional.
- *ud_inputs*: Si entre los inputs hay inputs no deseables (bad inputs) se puede indicar qué input es no deseable.
- *ud_outputs*: Si entre los outputs hay outputs no deseables (bad outputs) se puede indicar qué output es no deseable.

En el Ejemplo 4 se explica cómo usar la función read_data(). La documentación de **deaR** proporciona ejemplos de todas las funciones del paquete.

Ejemplo 4. Usando la función read_data().

En este momento, tenemos cargado el dataset "*Coll_Blasco_2006*". Este dataset es un dataframe que tiene 6 DMUs (columna 1) con 2 inputs (columnas 2 y 3) y 2 outputs (columnas 4 y 5).

Supongamos que queremos analizar la eficiencia de estas DMUs y que para ello vamos a utilizar el modelo DEA BCC.

Como el modelo DEA BCC es un modelo DEA convencional (no es fuzzy), lo primero que tenemos que hacer es adecuar los datos Coll_Blasco_2006 al formato que utiliza **deaR**. Para ello es para lo que utilizamos la función read_data().

Escribimos en el script "*ejemplo_read_data*" lo siguiente (ver Figura 26):

```
data_example <- read_data(Coll_Blasco_2006, ni=2, no=2)
```

Muy importante: Lectura y comprensión de la instrucción.

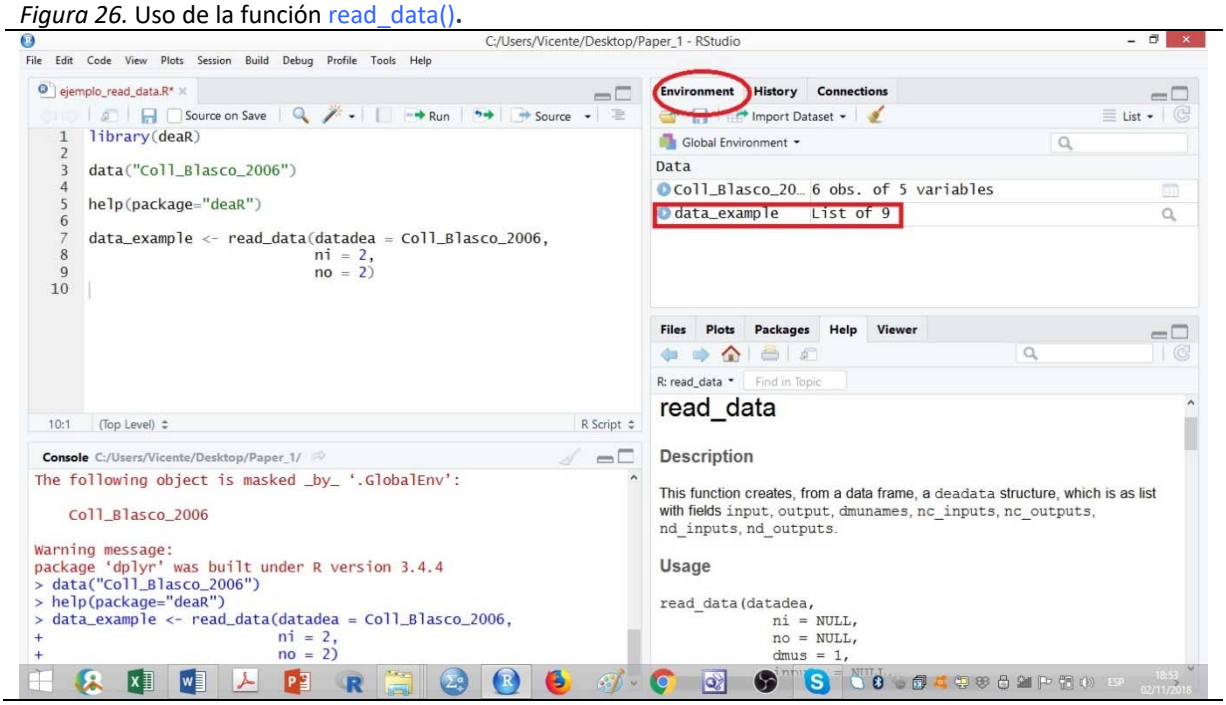
La parte derecha del símbolo de asignación (<-) está diciendo: lee los datos (read_data) en Coll_Blasco_2006 que tiene 2 inputs (ni=2) y 2 outputs(no=2). Las DMUS se encuentran en la primera columna (dmus=1). Este argumento no aparece en la función porque es el valor por defecto.

El resultado de la función lo asignas (<-) al objeto "*data_example*".

Ejecutar la instrucción (Run).

Observad como ahora en el *Environment* aparece el objeto "*Coll_Blasco_2006*" y el objeto "*data_example*". Observad también que "*data_example*" es una lista de 9 elementos (ver Figura 26).

Figura 26. Uso de la función read_data().



Ahora los datos están preparados para ejecutar cualquier modelo DEA convencional (ver sección 7.3.). En este caso deberíamos utilizar el objeto "*data_example*".

Recomendación: practicar la función read_data() con los ejemplos que aparecen en la ayuda del paquete.

Guardar "*ejemplo_read_data.R*".

7.2.3. La función read_malmquist().

Si tenemos datos temporarles y queremos analizar la eficiencia y la productividad de un conjunto de DMUs con el Índice de Productividad de Malmquist, tenemos que utilizar la función read_malmquist() para adecuar los datos al formato de lectura de **deaR**.

Con **deaR** los datos temporales pueden estar en dos formatos:

- **Formato ancho**: las DMUs y los inputs y outputs de los diferentes años por columna. Por ejemplo, ver el dataset de deaR "*Economy*"[8] (Figura 27).
- **Formato largo**: Periodo de tiempo por columna. DMUs, inputs y outputs por columna, pero agrupados por el periodo de tiempo. Por ejemplo, ver el dataset de deaR "*EconomyLong*" (Figura 28).

En el siguiente ejemplo intentamos mostrar la explicación anterior.

Ejemplo 5. Datos en formatos ancho y largo.

Ahora, vamos a mostrar estos diferentes formatos de datos. Para eso:

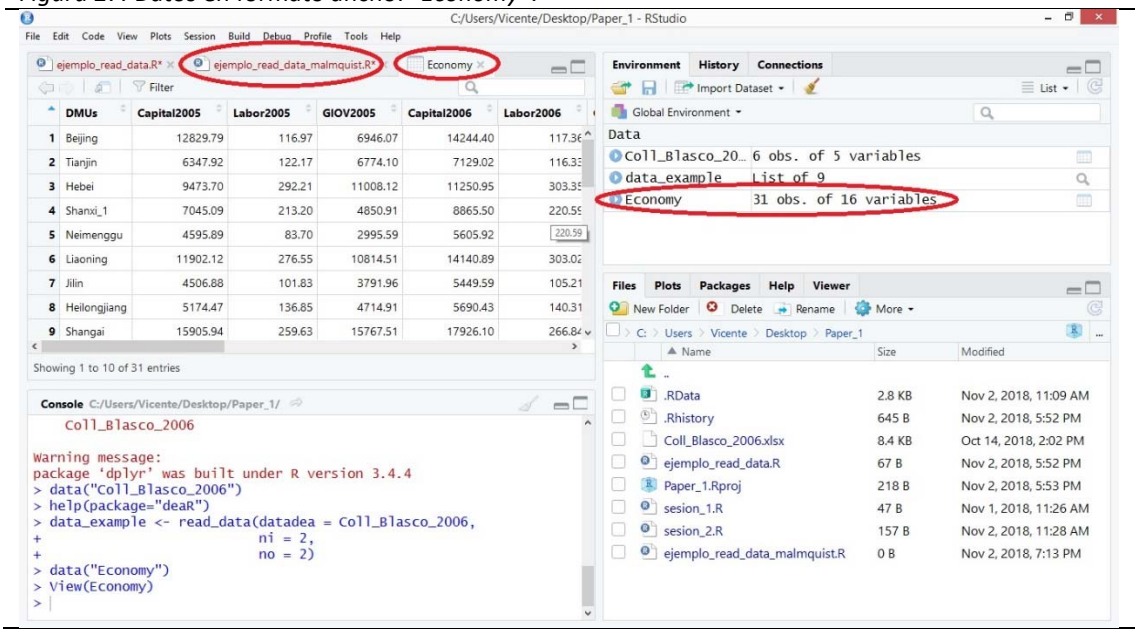
1. Creamos un nuevo script y lo llamamos "*ejemplo_read_malmquist*".

[8] Wang, Y.; Lan, Y. (2011). "Measuring Malmquist Productiviy Index: A New Approach Based on Double Frontiers Data Envelopment Analysis". Mathematical and Computer Modelling, 54, 2760-2771. https://doi.org/10.1016/j.mcm.2011.06.064

Nota: Si cerramos la sesión de trabajo, tenemos que abrir el proyecto "*Paper_1*" y luego crear el script. En este caso, no tenemos que olvidar cargar nuevamente **deaR**.

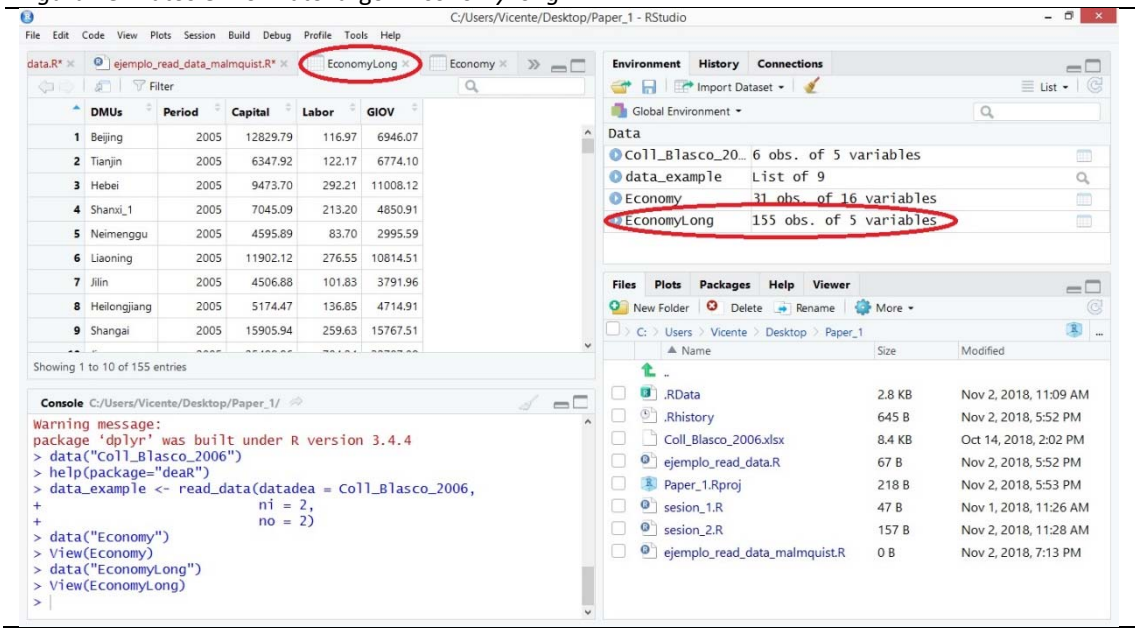
2. Cargamos el dataset de **deaR**: "Economy".
3. Visualizamos "Economy" (ver Figura 27). Este objeto (dataset) está formado por 31 DMUs con 2 inputs (*Capital* y *Labor*) y 1 output (*GIOV*) para 5 años (desde 2005 hasta 2009). "Economy" es un dataset en formato ancho.

Figura 27. Datos en formato ancho: "*Economy*".



4. Ahora cargamos el dataset "EconomyLong".
5. Visualizar "EconomyLong" (ver Figura 28). Este nuevo objeto de R (que es un dataset) tiene 155 observaciones (31 DMUs x 5 years), con 2 inputs (*Capital* y *Labor*) y 1 output (*GIOV*). La columna *Period* se refiere al periodo de tiempo 2005 a 2009.

Figura 28. Datos en formato largo: "*EconomyLong*".



Nota: observad como en *Environment* (ventana superior derecha) se van listando todos los objetos que vamos creando durante la sesión de trabajo.

La función read_malmquist() tiene los siguientes argumentos[9]:

- *datadea*: Se refiere al conjunto de datos a analizar (tiene que ser una dataframe).
- *nper*: Es el número de periodos de tiempo (con datos en formato wide)
- *percol*: Es el número de la columna que contiene el periodo de tiempo (con datos en formato long).
- *arrangement*: Indicar "horizontal" con datos en formato wide y "vertical" con datos en formato long.
- *dmus*: Indicar el número de la columna donde se encuentran las DMUs. Por defecto, **deaR** considera que las DMUs se encuentran en la primera columna.
- *ni*: Es el número de inputs.
- *no*: Es el número de outputs.
- *inputs*: En lugar de indicar el número de inputs se puede indicar el número de las columnas donde se encuentran los inputs.
- *outputs*: En lugar de indicar el número de outputs se puede indicar el número de las columnas donde se encuentran los outputs.
- *nc_inputs*: Si entre los inputs hay inputs no controlables se puede indicar qué input es no controlable.
- *nc_outputs*: Si entre los outputs hay outputs no controlables se puede indicar qué output es no controlable.
- *nd_inputs*: Si entre los inputs hay inputs no discrecionales se puede indicar qué input es no discrecional.
- *nd_outputs*: Si entre los outputs hay outputs no discrecionales se puede indicar qué output es no discrecional.
- *ud_inputs*: Si entre los inputs hay inputs no deseables (bad inputs) se puede indicar qué input es no deseable.
- *ud_outputs*: Si entre los outputs hay outputs no deseables (bad outputs) se puede indicar qué output es no deseable.

La versión actual de **deaR** no permite la presencia de inputs/outputs no deseables para calcular el índice de productividad de Malmquist. Esta característica será incorporada en una versión posterior.

En los Ejemplos 6 y 7 podemos ver cómo usar la función read_malmquist() con datos en formato ancho y largo, respectivamente.

[9] Podemos utilizar la ayuda de deaR: **help(read_malmquist)**.

Ejemplo 6. Función read_malmquist() con datos en formato ancho.

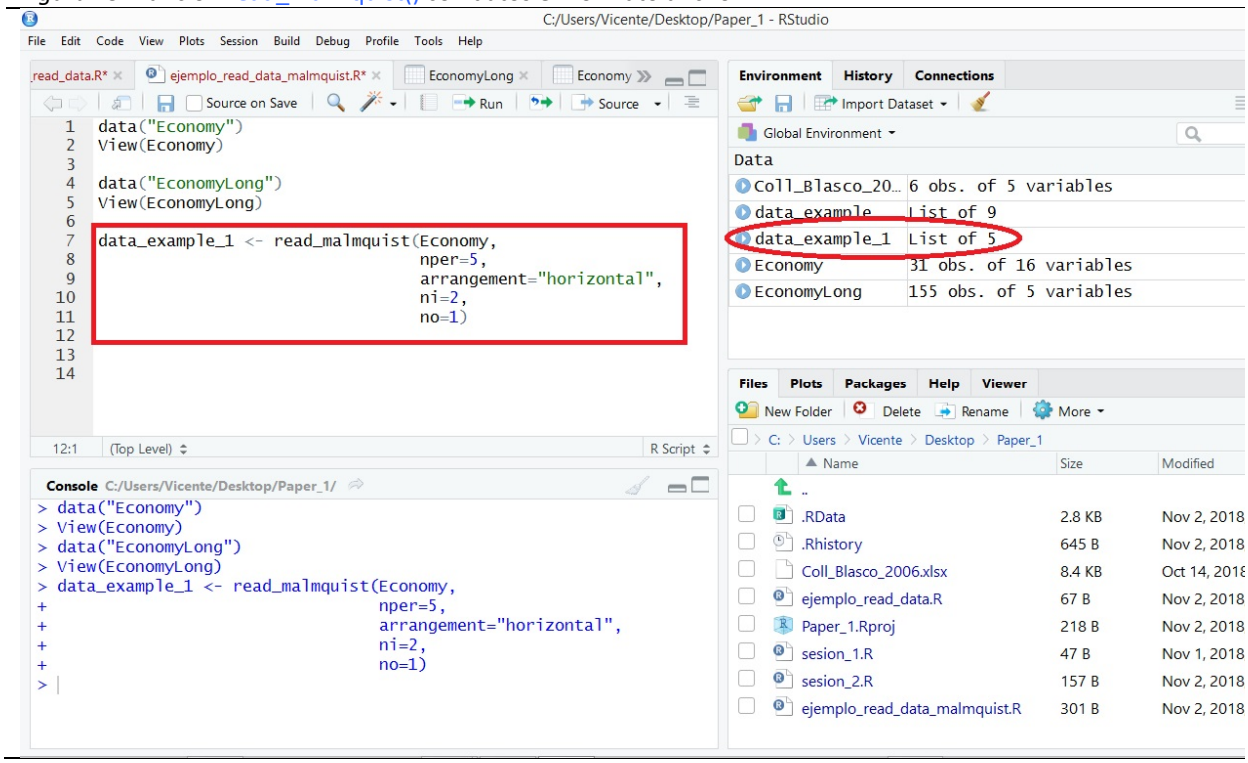
Para este ejemplo vamos a usar el dataset "*Economy*". Este dataset ya lo tenemos cargado en nuestra sesión de trabajo.

Para adaptar "*Economy*" al formato de lectura usado por **deaR**, escribimos en el script ("*ejemplo_read_malmquist*") el siguiente texto:

```
data_example_1 <- read_malmquist(Economy,
                                 nper=5,
                                 arrangement="horizontal",
                                 ni=2,
                                 no=1)
```

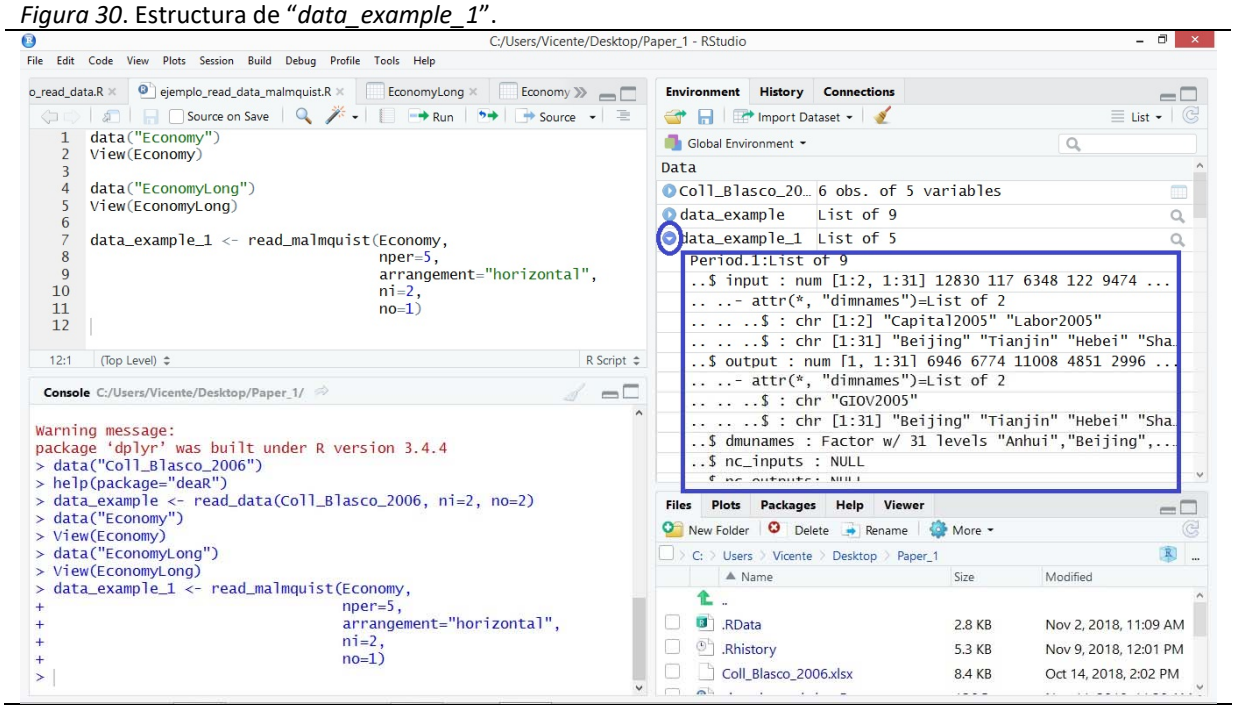
Al ejecutar la instrucción se crea un nuevo objeto ("*data_example_1*"), que es listado en el *Environment* (ver Figura 29).

Figura 29. Función read_malmquist() con datos en formato ancho.



Como podemos ver, "*data_example_1*" es una lista de 5 componentes. Si hacemos clic sobre el icono ▶ situado junto al nombre del objeto podemos ver su estructura (ve Figura 30).

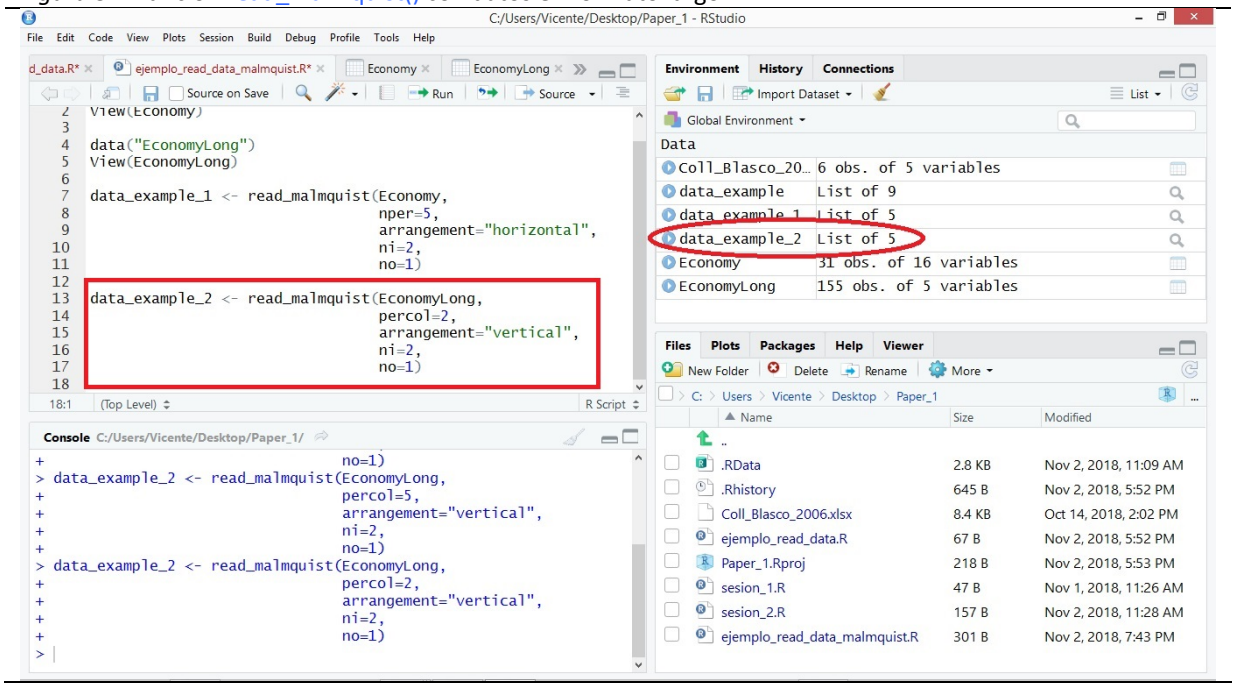
Figura 30. Estructura de "*data_example_1*".



Ejemplo 7. Función read_malmquist() con datos en formato largo.

Ahora, vamos a utilizar el dataset "*EconomyLong*". En la Figura 31 podemos ver la instrucción utilizada para adaptar los datos al formato de lectura de **deaR**.

Figura 31. Función read_malmquist() con datos en formato largo.



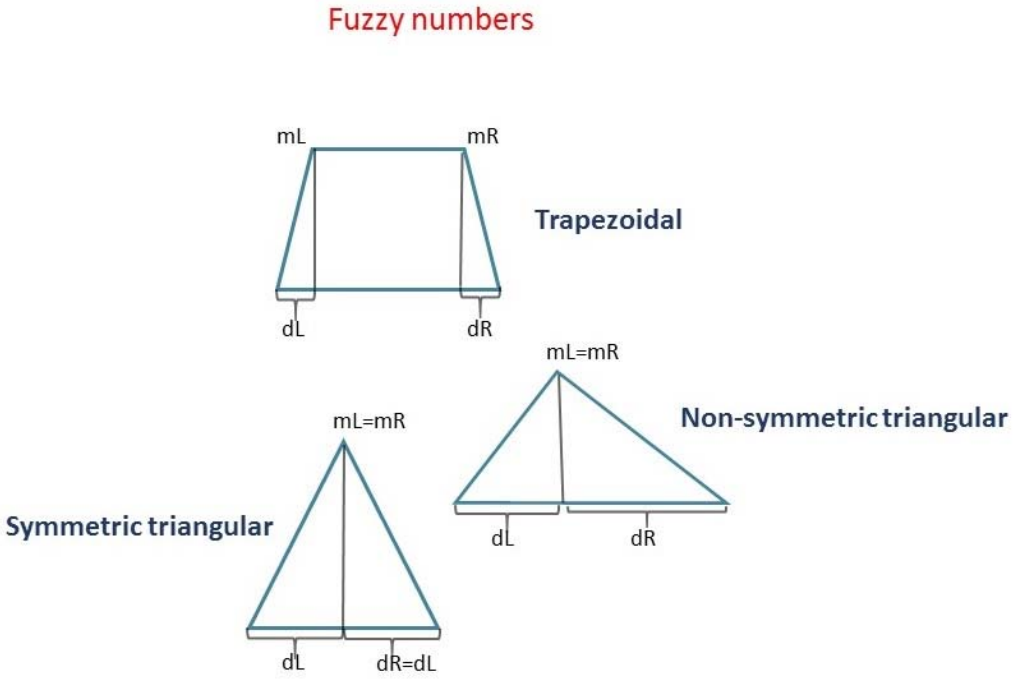
Guardamos el script "*ejemplo_read_data_malmquist.R*".

7.2.4. Función read_data_fuzzy().

Si tenemos datos inciertos y queremos analizar la eficiencia de un conjunto de DMUs con un modelo DEA fuzzy, tenemos que utilizar la función read_data_fuzzy() para adecuar los datos al formato de lectura de **deaR**.

deaR puede trabajar con números fuzzy trapezoidales, triangulares simétricos y triangulares no simétricos (ver Figura 32).

Figura 32. Números fuzzy.



La función read_data_fuzzy() tiene los siguientes argumentos[10]:

- *datadea*: Se refiere al conjunto de datos a analizar (tiene que ser una dataframe).
- *dmus*: Indicar el número de la columna donde se encuentran las DMUs. Por defecto **deaR** considera que las DMUs se encuentran en la primera columna.
- *Inputs.mL*: columnas donde se encuentran los valores mL de los inputs.
- *Inputs.mR*: columnas donde se encuentran los valores mR de los inputs.
- *Inputs.dL*: columnas donde se encuentran los valores dL de los inputs.
- *Inputs.dR*: columnas donde se encuentran los valores dR de los inputs.
- *Outputs.mL*: columnas donde se encuentran los valores mL de los outputs.
- *Outputs.mR*: columnas donde se encuentran los valores mR de los outputs.
- *Outputs.dL*: columnas donde se encuentran los valores dL de los outputs.
- *Outputs.dR*: columnas donde se encuentran los valores dR de los outputs.
- *nc_inputs*: Si entre los inputs hay inputs no controlables se puede indicar qué input es no controlable.

[10] Podemos utilizar la ayuda de deaR: **help(read_data_fuzzy).**

- *nc_outputs*: Si entre los outputs hay outputs no controlables se puede indicar qué output es no controlable.
- *nd_inputs*: Si entre los inputs hay inputs no discrecionales se puede indicar qué input es no discrecional.
- *nd_outputs*: Si entre los outputs hay outputs no discrecionales se puede indicar qué output es no discrecional.
- *ud_inputs*: Si entre los inputs hay inputs no deseables (bad inputs) se puede indicar qué input es no deseable.
- *ud_outputs*: Si entre los outputs hay outputs no deseables (bad outputs) se puede indicar qué output es no deseable.

En la versión actual de **deaR** solo se tiene en cuenta los inputs/outputs no deseables en los modelos fuzzy basados en el modelo DEA BCC.

Vemos un ejemplo.

Ejemplo 8. Función read_data_fuzzy().

1. Creamos un nuevo script y lo nombramos como: "*ejemplo_read_data_fuzzy*"

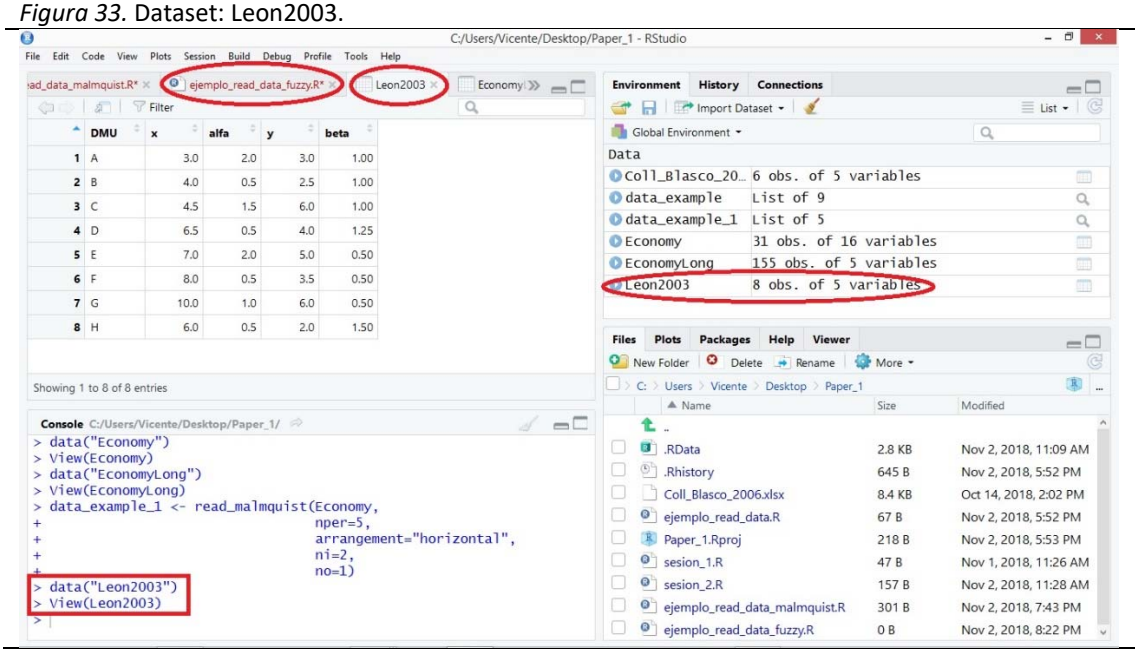
   Nota: Si cerramos la sesión de trabajo tenemos que abrir primero el proyecto "*Paper_1*" y luego crear el script. En este caso, no olvidar cargar **deaR**.

2. Cargamos el dataset "*Leon2003*"[11]:

   data("Leon2003").

   Este dataset (ver Figura 33) está formado por 8 DMUs con 1 input fuzzy triangular simétrico (alpha es la apertura del input) y 1 output fuzzy triangular simétrico (beta es la apertura del output).

Figura 33. Dataset: Leon2003.



[11] León, T.; Liern, V. Ruiz, J.; Sirvent, I. (2003). "A Possibilistic Programming Approach to the Assessment of Efficiency with DEA Models", Fuzzy Sets and Systems, 139, 407–419.https://doi.org/10.1016/S0165-0114(02)00608-5

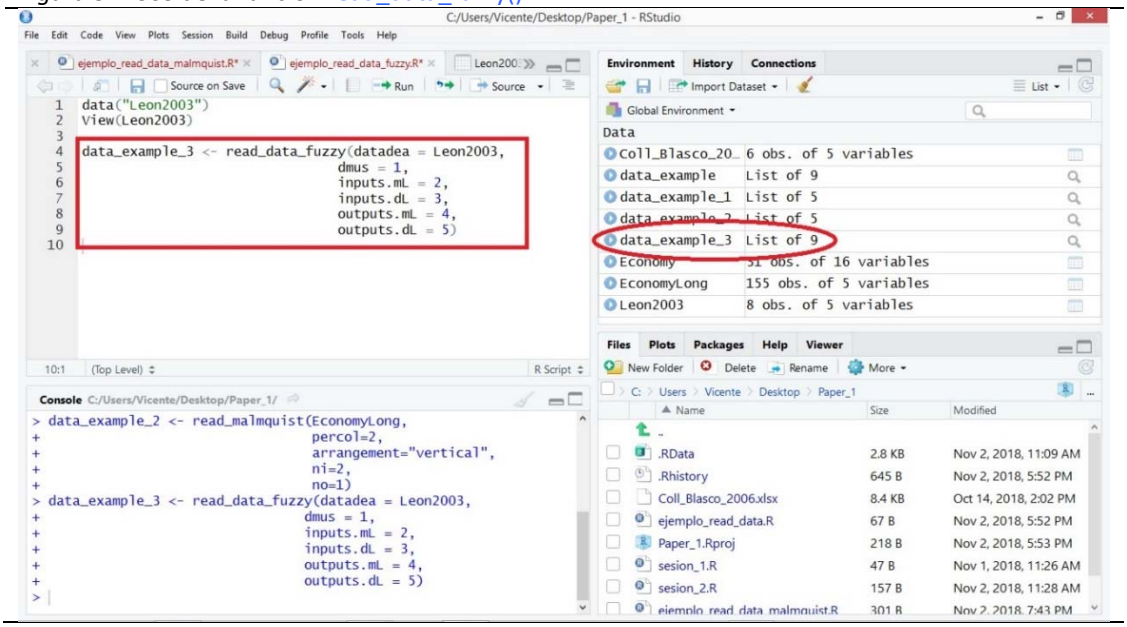
3. Ahora utilizamos la función read_data_fuzzy() para adecuar los datos al formato de lectura de **deaR**. Para ello, escribimos en el script:

```
data_example_3 <- read_data_fuzzy(Leon2003,
                                  inputs.mL=2,
                                  inputs.dL=3,
                                  outputs.mL=4,
                                  outputs.dL=5)
```

y ejecutamos la instrucción.

Nuestra sesión de trabajo debería parecerse a la siguiente captura de pantalla (ver Figura 34):

Figura 34. Uso de la función read_data_fuzzy().



En este ejemplo, inputs.dL=alpha y ouputs.dL=beta. Como estamos trabajando con números fuzzy triangulares simétricos: inputs.dL=inputs.dR y outputs.dL=outputs.dR

Hemos creado un nuevo objeto: "*data_example_3*". Este objeto es el que utilizaremos para ejecutar un determinado modelo DEA fuzzy.

Guardamos "*ejemplo_read_data_fuzzy.R*", cerramos el proyecto y salimos de RStudio.

7.3. Seleccionar y ejecutar un modelo DEA.

Una vez los datos están preparados para que puedan ser leídos por **deaR**, el siguiente paso consiste en seleccionar el modelo DEA y ejecutarlo.

En la versión actual de **deaR** (versión 1.0) se encuentran disponibles los siguientes modelos:

Conventional DEA models
Basic (radial) models (envelopment and multiplier forms)
Directional distance function model
(Weighted) Additive model

Conventional DEA models
Super-efficiency additive model
Radial Super-efficiency model
(Weighted) Non-radial model
Preference Structure model
(Weighted) Slack-based model
(Weighted) Super-efficiency slack-based model
Cross-efficiency (crs[12] and vrs[13])
Bootstrapping (Simar and Wilson algorithm)
FDH model
Productivity
Malmquist index
Fuzzy DEA models
Kao and Liu model[14]
Possibilistic model
Guo and Tanaka model
Fuzzy cross-efficiency[15] (only crs)

En la ayuda de **deaR** podemos encontrar información sobre cómo utilizar las diferentes funciones y ejemplos poniéndolas en práctica.

Aunque los modelos DEA disponibles en **deaR** son los listados arriba, dada la flexibilidad con la que se ha programado el paquete (y aquí reside una importante fortaleza de **deaR**), el propio usuario puede experimentar, probar e implementar variantes de estos modelos. Por ejemplo, si el usuario define apropiadamente los pesos en el "*Modelo Aditivo*", puede obtener los modelos MPI[16] o RAM[17]. Otro ejemplo. A partir del modelo "*Preference Structure*" (modelo no radial ponderado) pueden calcularse los modelos "*Cost Efficiency*", "*Revenue Efficiency*" y "*Profit Efficiency*".

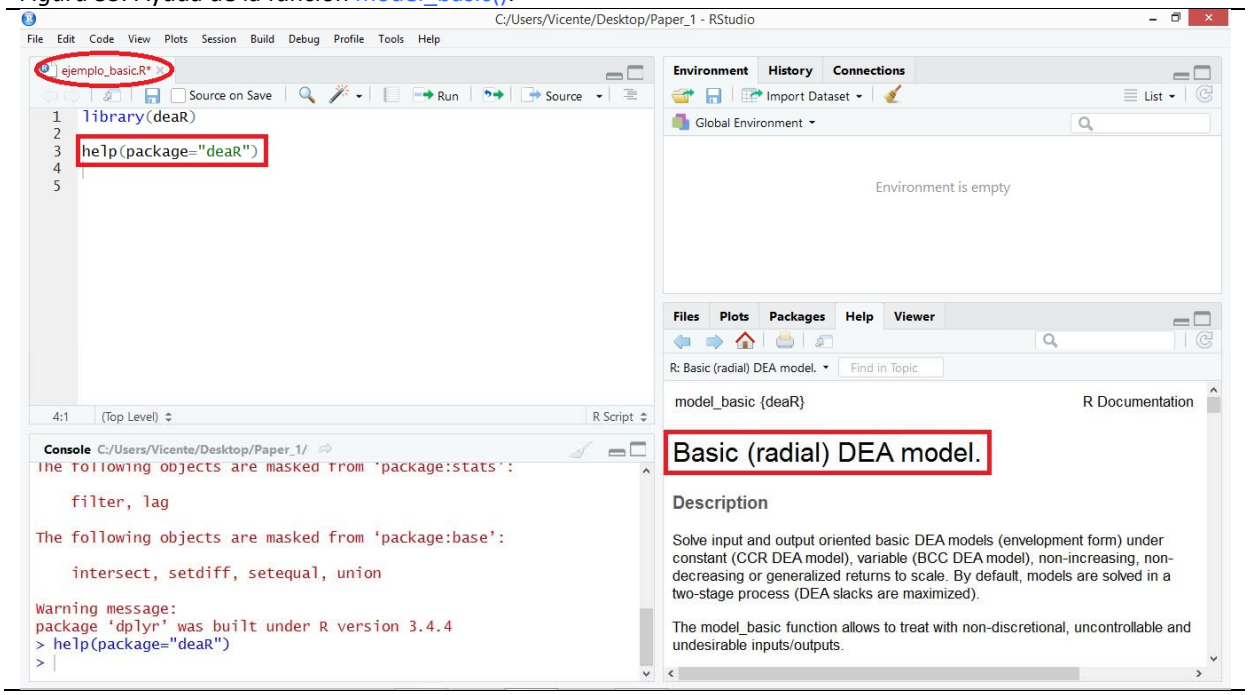
A continuación vamos a ver cómo utilizar la función model_basic() para calcular modelos DEA básicos. Para ello:

- Abrir el proyecto "*Paper_1*" y crear un nuevo script. Llamarlo "*ejemplo_basic*".
- Cargar **deaR**.
- Abrir la ayuda de **deaR**.
- Ir a la ayuda de la función model_basic() (ver Figura 35). También podemos escribir:

[12] crs = rendimientos constantes a escala.
[13] vrs = rendimientos variables a escala.
[14] Este modelo DEA fuzzy se ha extendido para un amplio número de modelos DEA. Ver la ayuda del paquete: **help("modelfuzzy_kaoliu")**.
[15] Basado en el modelo de Guo y Tanaka.
[16] Measure of Inefficiency Proportions (MPI).
[17] Range Adjusted Measure (RAM).

help("model_basic")

Figura 35. Ayuda de la función model_basic().



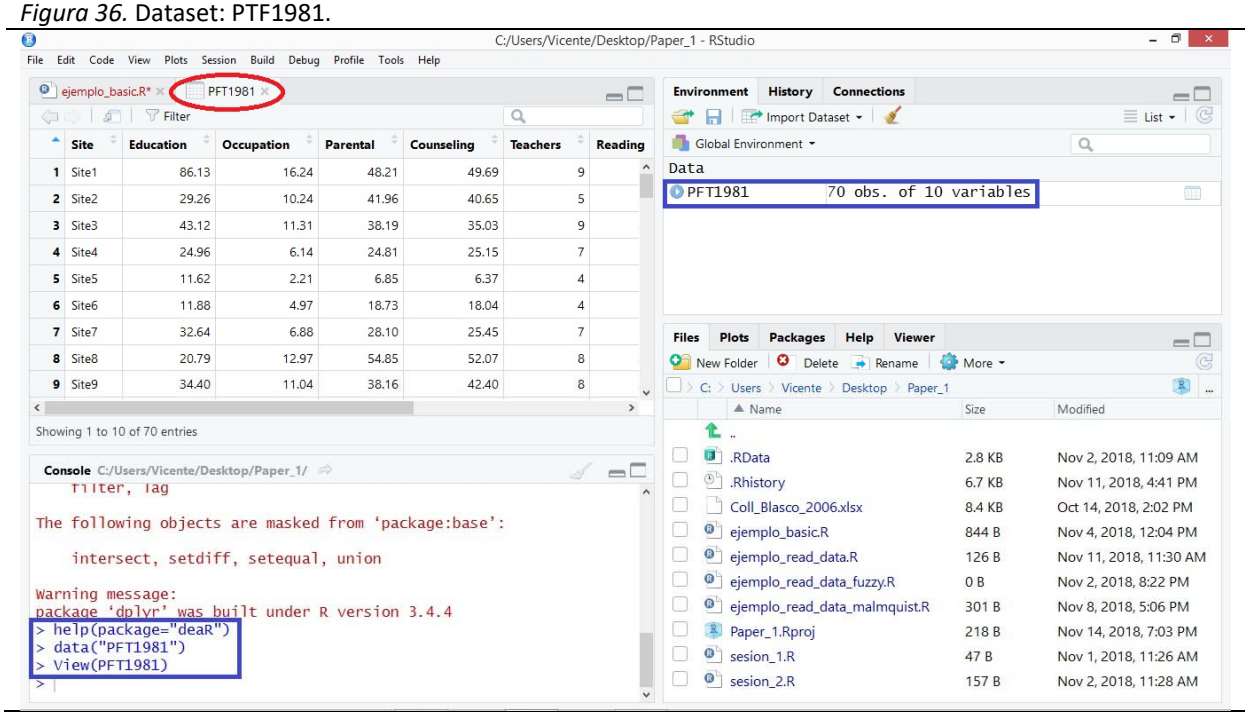
La función model_basic() tiene los siguientes argumentos:

- *datadea*: Conjunto de datos en el formato de lectura de **deaR**.
- *dmu_ref*: Selección de un subconjunto de DMUs
- *dmu_eval*: DMUs a evaluar del subconjunto seleccionado.
- *orientation*: Orientación del modelo: input, output o direccional.
- *dir_input*: Vector de dirección input en modelos direccionales.
- *dir_output*: Vector de dirección output en modelos direccionales.
- *rts*: Rendimientos a escala del modelo (constantes, variables, no-crecientes, no-decrecientes, generalizados).
- *L*: Si seleccionamos rendimientos a escala generalizados, límite inferior.
- *U*: Si seleccionamos rendimientos a escala generalizados, límite superior.
- *Maxslack*: Por defecto, las holguras son maximizadas en una segunda etapa.
- *weight_slack_i*: El usuario puede asignar pesos a las holguras inputs en la segunda etapa.
- *weight_slack_o*: El usuario puede asignar pesos a las holguras outputs en la segunda etapa.
- *vtrans_i*: Con inputs no deseables y rendimientos constantes a escala el usuario puede definir un vector de traslación. Por defecto es el máximo más 1.
- *vtrans_o*: Con outputs no deseables y rendimientos constantes a escala el usuario puede definir un vector de traslación. Por defecto es el máximo más 1.
- *compute_target*: Calcula los targets en la solución max slack.

- *compute_multiplier*: Si el usuario quiere obtener los multiplicadores inputs y outputs (forma multiplier).
- *returnlp*: Para cada DMU, devuelve el problema lineal de la primera etapa.

A continuación, vamos a aprender cómo usar la función model_basic() haciendo los Ejemplos 9 y 10. Para seguir estos ejemplos tenemos que cargar el dataset "*PFT1981*"[18]. Este dataset está formado por 70 DMUs con 5 inputs (*Education*, *Occupation*, *Parental*, *Counseling*, *Teachers*) y 3 outputs (*Reading*, *Math*, *Coopersmith*). La Figura de abajo (Figura 36) muestra las instrucciones que deberíamos escribir en el script "*ejemplo_basic*" para cargar los datos.

Figura 36. Dataset: PTF1981.



Ejemplo 9. Función model_basic().

De las 70 DMUs (school sites) de "*PTF1981*", hay 49 DMUs en Project Follow Through (PFT) y 21 DMUs en Non-Follow Through (NFT).

En este ejemplo, vamos a calcular la eficiencia de las 49 DMUS en PFT con el modelo DEA CCR input-orientado.

En primer lugar, tenemos que usar la función read_data() para adaptar los datos al formato de lectura de **deaR** (ver sección 7.2.2.). Ejecutar la instrucción c Run

Ahora, tenemos que usar la función model_basic() porque queremos calcular la eficiencia utilizando el modelo básico CCR (que es un modelo DEA convencional). Pero como sólo queremos la eficiencia de las primeras 49 DMUs, que son las que participan en PFT, utilizamos el argumento *dmu_ref=1:49*. Además, como queremos evaluar las 49 DMUs utilizamos el argumento: *dmu_eval=1:49*. Por tanto, tenemos que escribir en el script "*ejemplo_basic*" el siguiente texto:

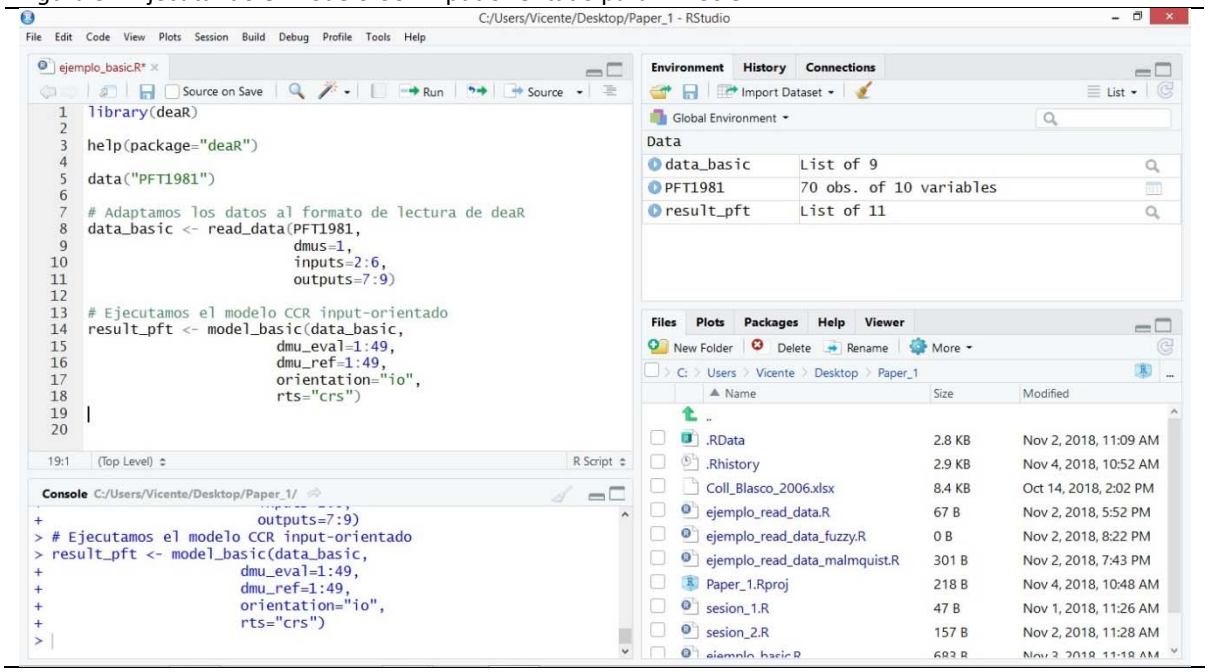
[18] Charnes, A.; Cooper, W.W.; Rhodes, E. (1981). "Evaluating Program and Managerial Efficiency: An Application of Data Envelopment Analysis to Program Follow Through", Management Science, 27(6), 668-697. https://pubsonline.informs.org/doi/abs/10.1287/mnsc.27.6.668

```
result_pft <- model_basci(PFT1981,
                          dmu_ref=1:49,
                          dmu_eval=1:49,
                          orientation="io",
                          rts="crs")
```

y ejecutamos la instrucción (ver Figura 37).

> Nota: También se puede seleccionar todas las instrucciones y ejecutarlas conjuntamente con Run en lugar de hacerlo por separado.

Figura 37. Ejecutando el modelo CCR input orientado para DMUs en PFT.

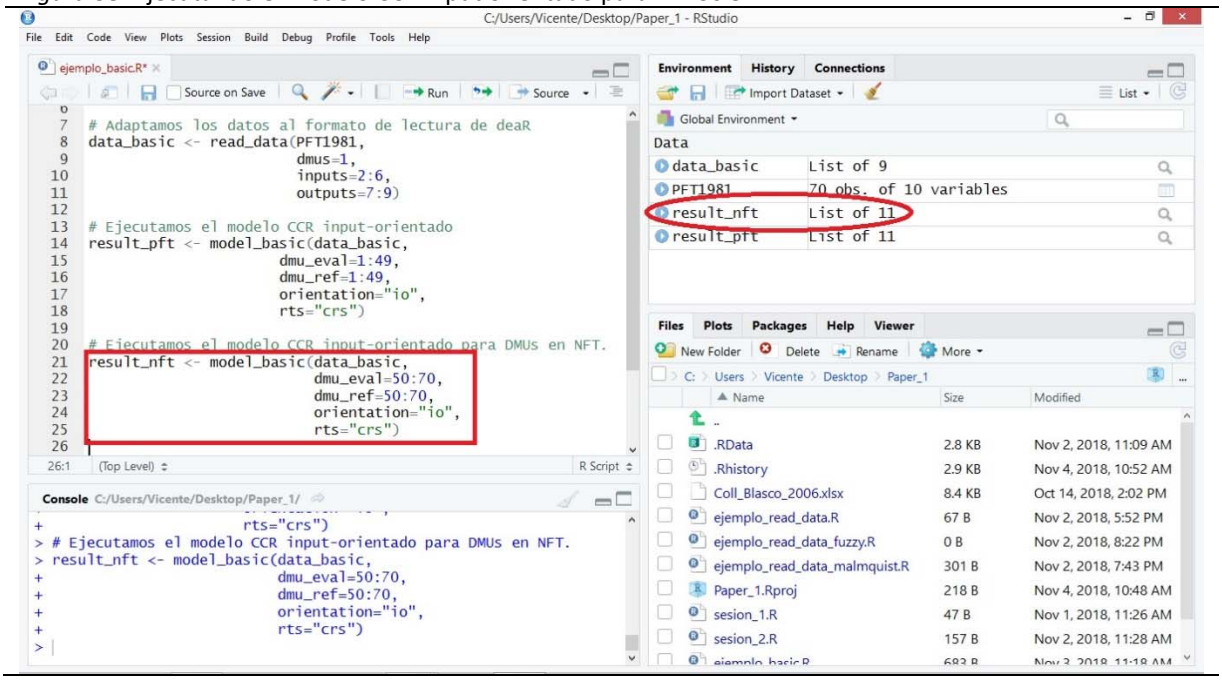


Ejemplo 10. Función model_basic().

Llegados a este punto , **¿podrías ejecutar el modelo DEA CCR input-orientado para las DMUs en NFT?** (DMUs en NFT son las DMUs desde la 50 a la 70).

La respuesta a esta pregunta se encuentra en la siguiente página (Figura 38).

Figura 38. Ejecutando el modelo CCR input orientado para DMUs en NFT.

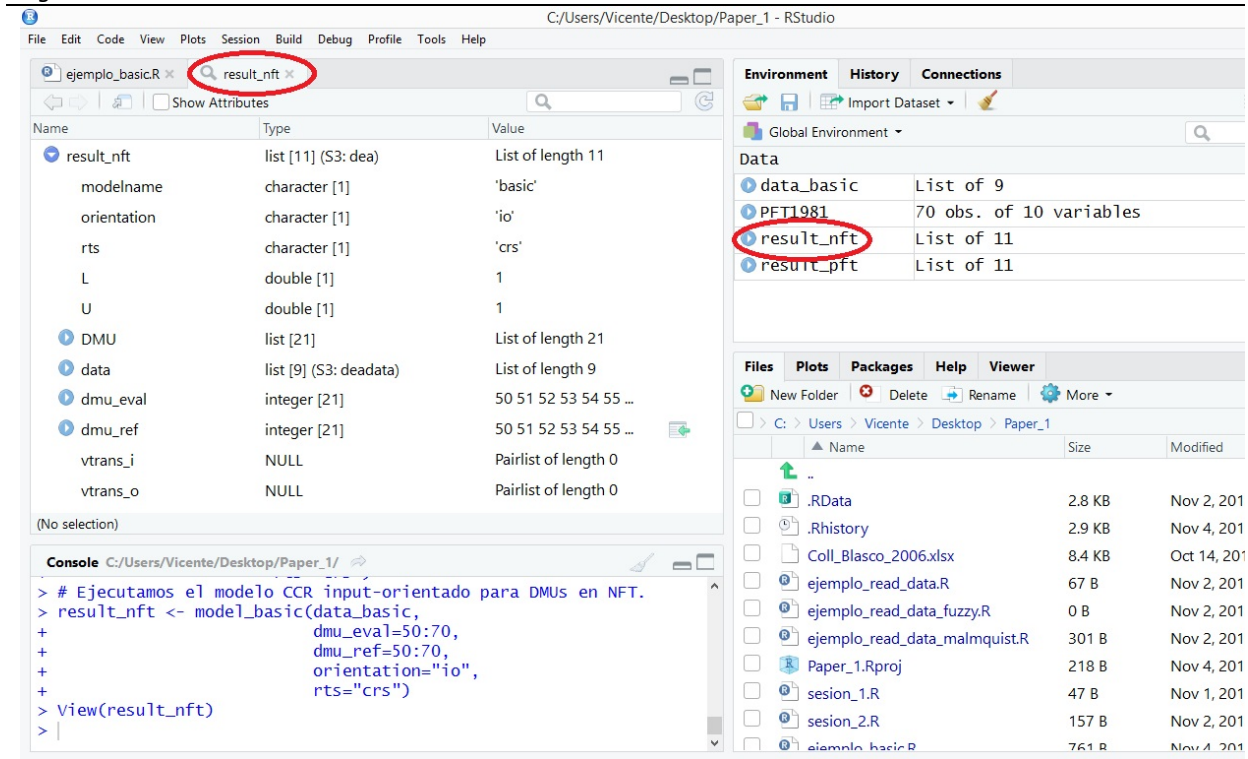


Guardamos el script "*ejemplo_basic*".

7.4. Extracción de los principales resultados.

En los Ejemplos 9 y 10, después de ejecutar la función model_basic(), todos los resultados se encuentran almacenados en los objetos "*result_pft*" y "*result_nft*" Estos objetos son una lista de 11 componentes. Podemos ver el contenido de la lista haciendo clic sobre el nombre del objeto (ver Figura 39) o ver la estructura de la lista haciendo clic sobre la flecha ()

Figura 39. Estructura de los resultados de un modelo DEA básico.



Para extraer los resultados del análisis DEA, **deaR** cuenta con una serie de funciones específicas. Estas funciones son:

- ✓ **efficiencies()**: Extrae las puntuaciones de eficiencia.
- ✓ **slacks()**: Extrae las holguras.
- ✓ **targets()**: Extrae los valores objetivo (targets).
- ✓ **lambdas()**: Extrae las intensidades o lambdas.
- ✓ **references()**: Extrae el conjunto de referencia de las DMUs ineficientes.
- ✓ **rts()**: Extrae los rendimientos a escala que caracterizan a una DMU.
- ✓ **multipliers()**: Extrae los multiplicadores (o pesos) del modelo DEA en forma multiplicativa.

En el Ejemplo 11 vamos a practicar con estas funciones.

Ejemplo 11. Extracción de resultados.

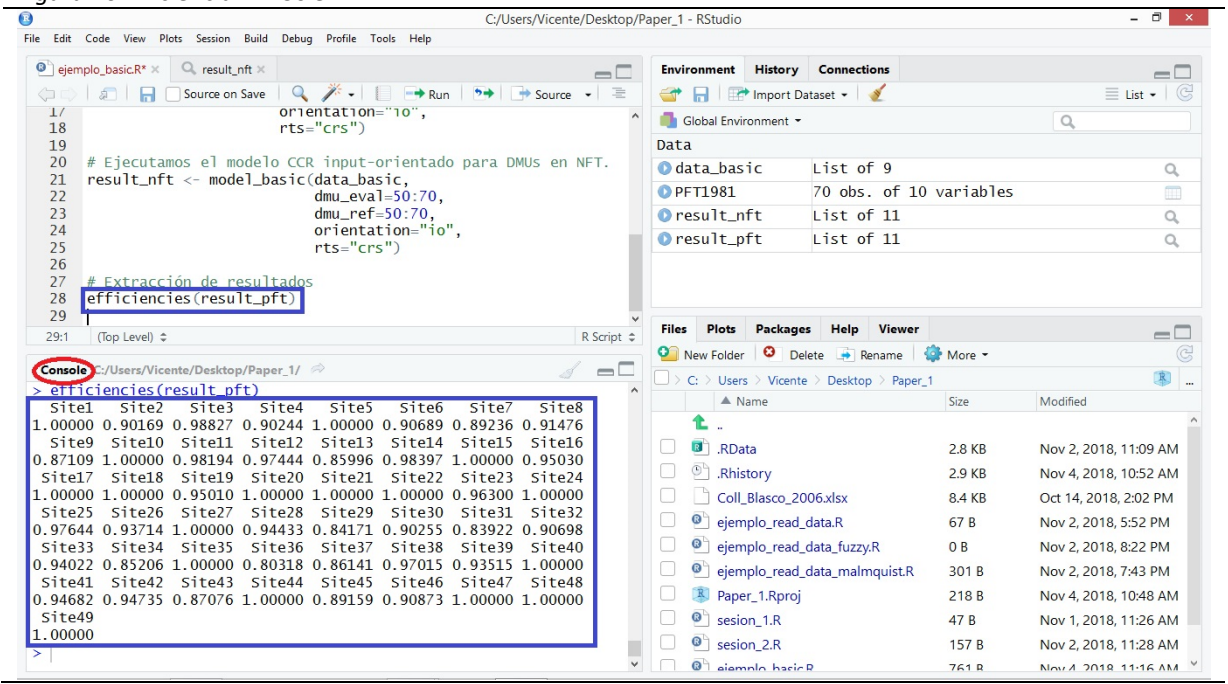
Para extraer las puntuaciones de eficiencia de "*result_pft*", que contiene los resultados del modelo DEA CCR input-orientado para las DMUs en PFT, solo tenemos que escribir la instrucción:

efficiencies(result_pft)

y ejecutarla.

Las puntuaciones de eficiencia se mostrarán en la *Consola*, como puede verse en la siguiente Figura.

Figura 40. Eficiencia DMUs en PFT.



De forma similar, para extraer los conjuntos de referencia escribimos y ejecutamos la instrucción:

references(result_pft)

El resto de funciones (slacks(), targets(), lambdas(), rts() y multipliers()) se utilizan de forma similar. Observad los resultados que se extraen.

Guardar el script "*ejemplo_basic.R*", cerrar el proyecto y salir de RStudio.

7.5. Resumen de resultados. La función summary().

Además de las funciones anteriores (efficiencies(), lambdas(), etc.), **deaR** cuenta con la función summary(), que resume los resultados del análisis DEA. La función summary() sirve para resumir los resultados tanto de los modelos DEA convencionales como de los modelos DEA fuzzy. Para utilizar esta función tenemos que saber que tiene la siguiente estructura[19]:

summary(objeto, exportExcel = TRUE, filename = NULL)

Los argumentos de esta función se refieren a:

- *object*: Es el objeto en al que hemos asignado los resultados de un modelo DEA convencional o DEA fuzzy.
- *exportExcel*: El valor por defecto para este argumento es TRUE. Por tanto, la función resumen automáticamente creará un fichero Excel con los principales resultados en el directorio de trabajo. El usuario puede elegir no crear el fichero Excel de resumen de resultados (exportExcel= FALSE).
- *filename*: Por defecto, el valor del argumento es NULL. Por tanto, el nombre por defecto del fichero Excel será: "*ResutsDEAaño-mes-dia_hora:minuto:segundo.xlsx*". Por supuesto, el usuario puede dar nombre al fichero Excel.

Como hemos comentado, la forma de utilizar la función summary() para resumir los resultados de una análisis DEA es idéntica tanto en los modelos DEA convencionales como los modelos DEA fuzzy. La diferencia se encuentra en el fichero Excel que se genera en el resumen. Es decir, las hojas Excel que se crean son distintas según el modelo. Vamos a ver esto en los Ejemplos 12 y 13.

Ejemplo 12. Resumen de resultados: modelo DEA convencional.

Seguir los siguientes pasos:

1. Abrir el proyecto "*Paper_1*".
2. Crear un nuevo script: "*Resumen_DEA*"
3. Cargar **deaR**.

Supuesto: Cargar el dataset de **deaR**: "*Hua_Bian_2007*"[20]. Este dataset tiene 30 DMUs con, por este orden, 2 inputs (D-Input1, D-Input2), 2 outputs (D-Output1, D-Output2) y 1 output no deseable (UD-Output1).

[19] Podemos utilizar la ayuda de deaR: **help(summary.dea)** o **help(summary.dea_fuzzy)**.

[20] Hua Z.; Bian Y. (2007). DEA with Undesirable Factors. In: Zhu J., Cook W.D. (eds) Modeling Data Irregularities and Structural Complexities in Data Envelopment Analysis. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-71607-7_6

Queremos obtener el resumen de resultados del modelo BCC output-orientado. Como hay un output no deseable, para tenerlo en cuenta en el análisis utilizaremos como vector de traslación el utilizado por Hua y Bian (2007): vtrans_o=1500.

Importante: Recuerda los pasos para utilizar **deaR**.

Paso 1. Cargar los datos.

Paso 2. Adaptar los datos al formato de lectura de **deaR**.

Paso 3. Ejecutar el modelo DEA.

Paso 4. Extraer los resultados.

Intentadlo!!!

(la solución en la siguiente página)

Solución: Como podemos ver en la Figura 41, para resolver el Ejemplo 12 tenemos que escribir en el script "*Resumen_DEA*" las siguientes instrucciones:

Paso 1. Cargar los datos:

```
data("Hua_Bian_2007")
```

Pas 2. Adaptar los datos:

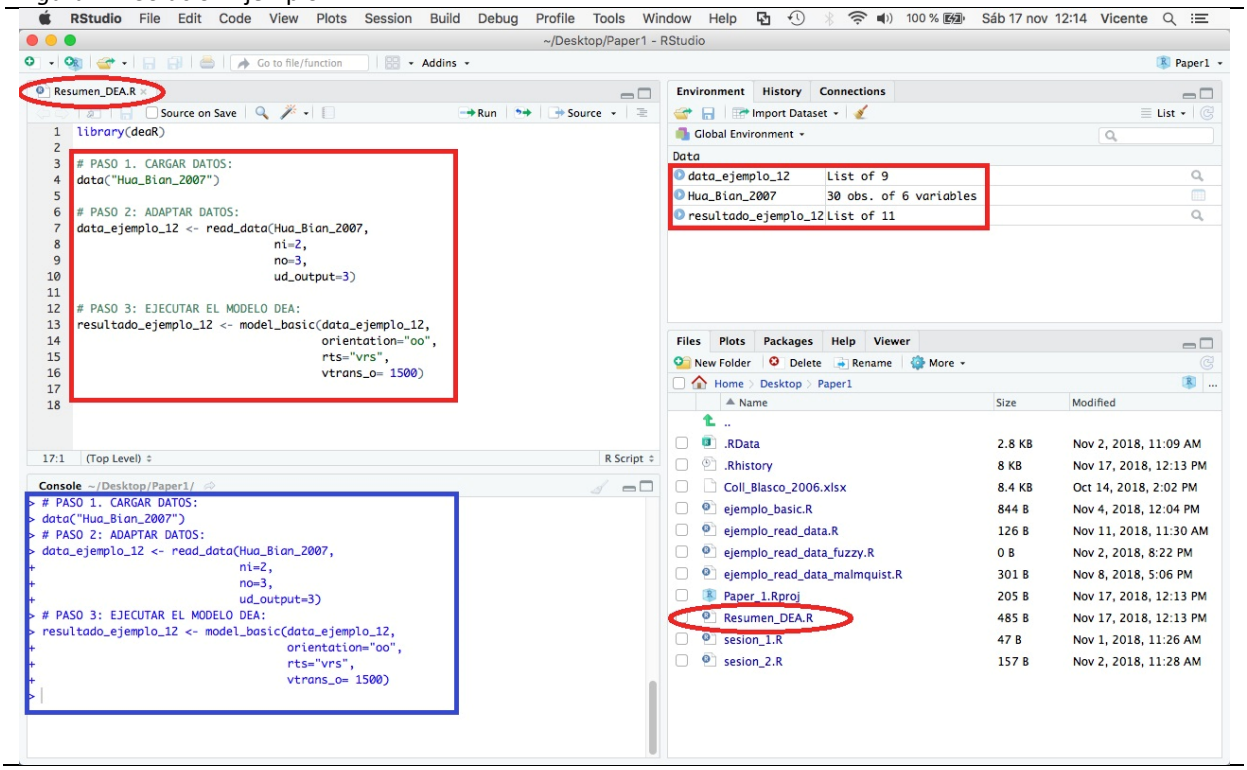
```
data_ejemplo_12 <- read_data(Hua_Bian_2007,
                             ni=2,
                             no=3,
                             ud_output=3)
```

Observación: Tenemos 3 outputs (no=3) y el tercer output es el output no deseable (ud_output=3)

Paso 3. Ejecutamos el modelo DEA

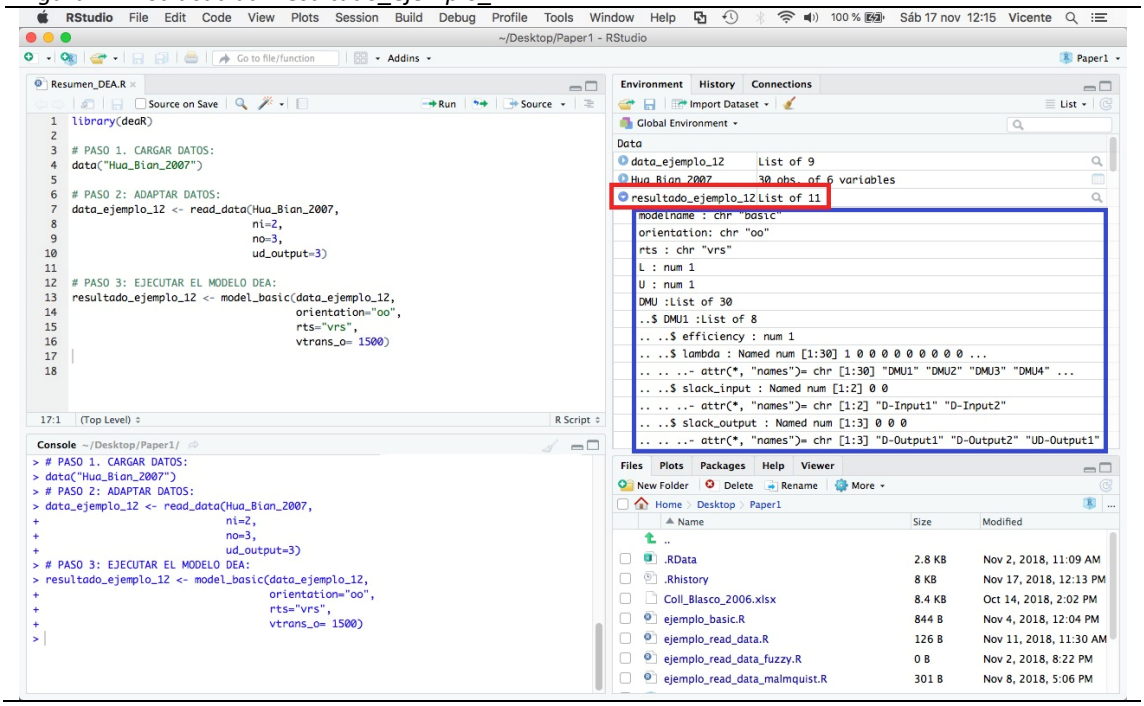
```
resultado_ejemplo_12 <- model_basic(data_ejemplo_12,
                                    orientation="oo",
                                    rts="vrs",
                                    vtrans_o= 1500)
```

Figura 41. Solución Ejemplo 12.



Todos los resultados del modelo BCC output orientado que hemos ejecutado se encuentran en el objeto "*resultado_ejemplo_12*", que es una lista de 11 componentes (ver Figura 42).

Figura 42. Estrucutrua "*resultado_ejemplo_12*".



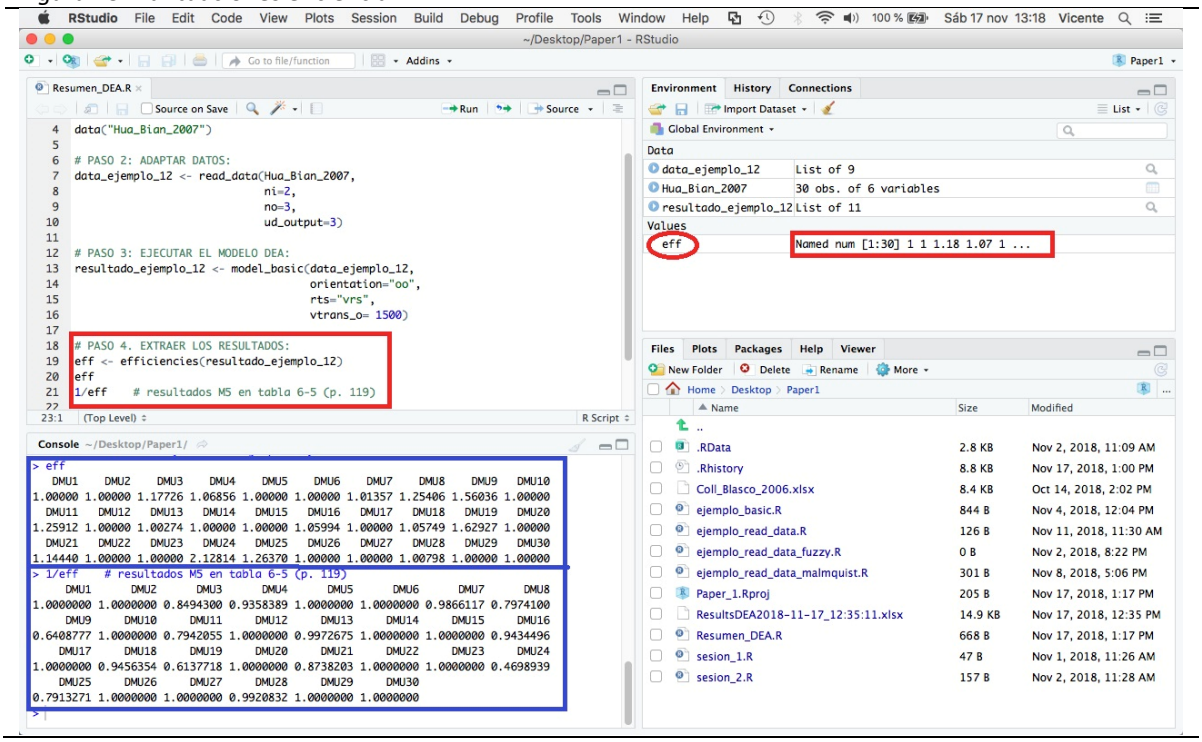
En este punto, podemos extraer los resultados parciales con las funciones: efficiencies(), lambdas(), multipliers(), rts(), references(), slacks(), targets(); y el resumen de resultados con la función summary().

Como se muestra en la Figura 43, extraemos las puntuaciones de eficiencia de las DMUs con la función efficiencies(). Las eficiencias son mostradas en la *Consola* y asignadas al objeto "*eff*". Para obtener los mismos resultados que los mostrados por Hua y Bian (2007), escribimos en el script:

1/eff

y ejecutamos la instrucción.

Figura 43. Puntuaciones eficiencia.

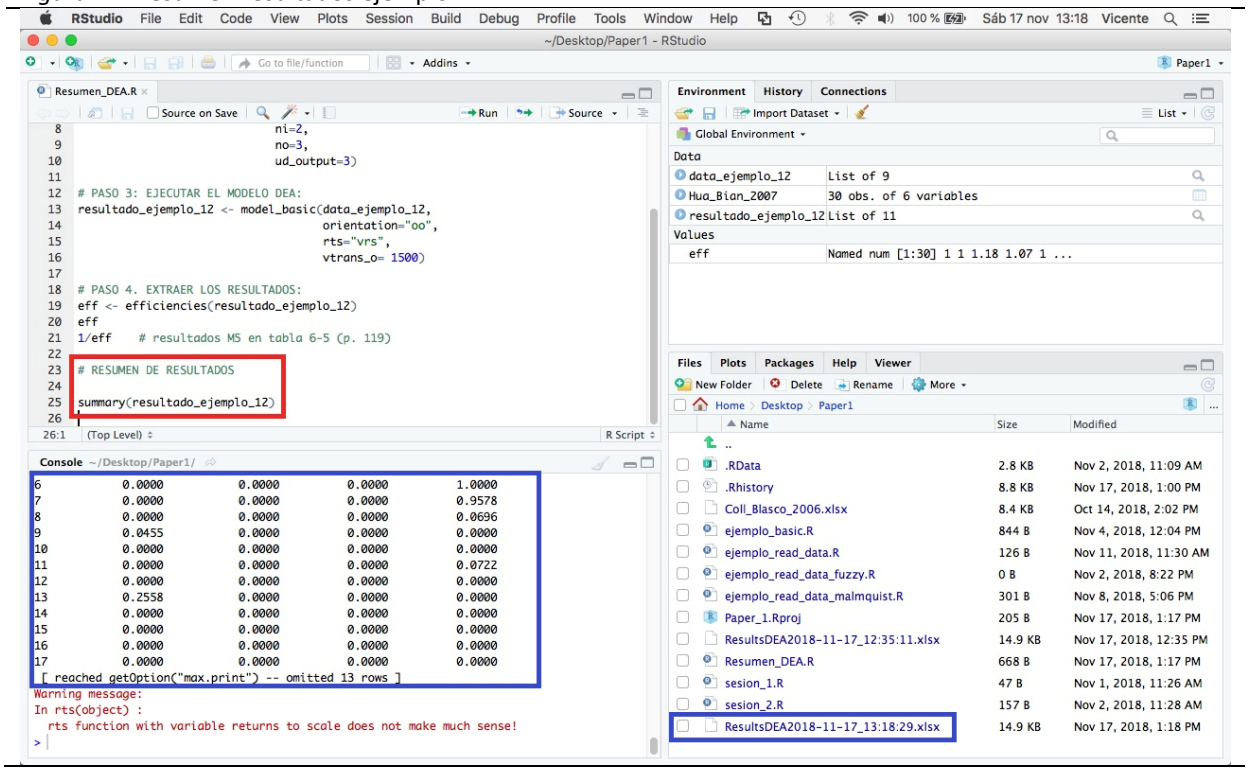


Para obtener un resumen de todos los resultados obtenidos al ejecutar el modelo DEA BCC output-orientado utilizamos la función summary(). Escribimos en el script "*Resumen_DEA*":

summary(resultado_ejemplo_12)

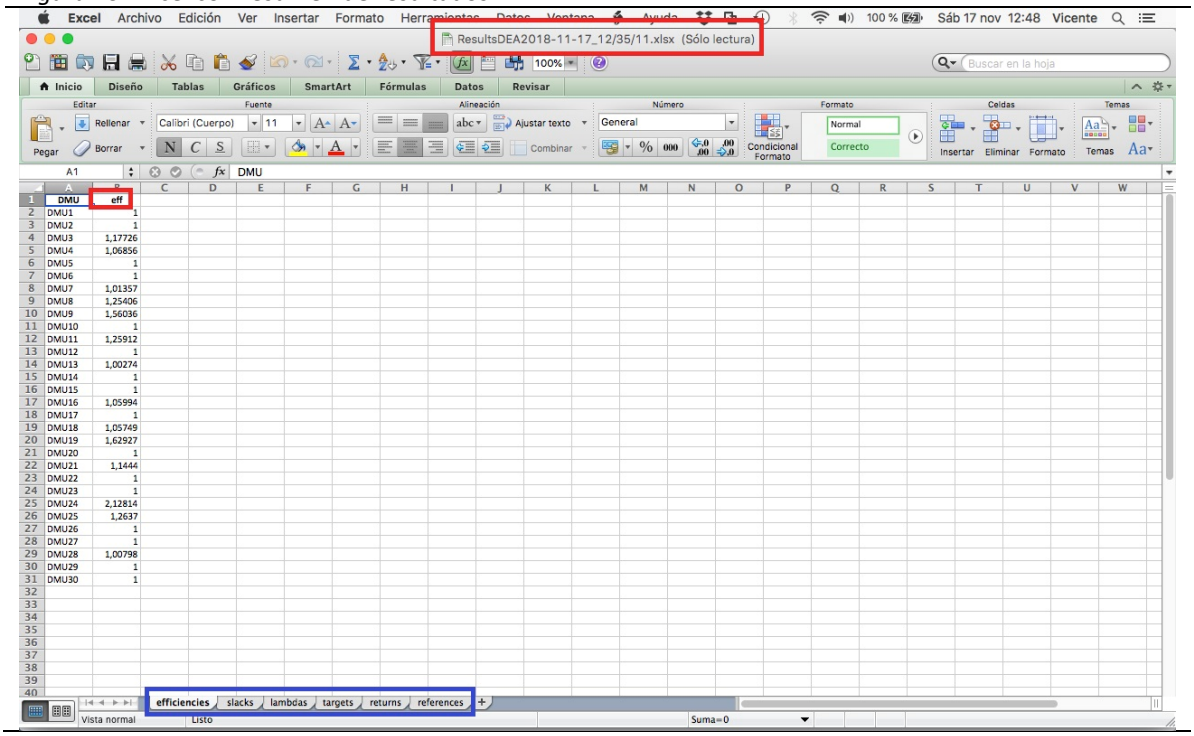
Todos los resultados son mostrados en la *Consola*. Aunque no hay muchas DMUs (solo 30), hay muchos resultados. No es práctico utilizar la función summary() para ver los resultados en la pantalla. Sin embargo, como podemos ver en la Figura 44, **deaR** ha creado un fichero Excel que tiene todos estos resultados. Observad el nombre que por defecto se ha dado al fichero.

Figura 44. Resumen resultados ejemplo 12.



Abrir el fichero Excel para ver cómo se ofrecen los resultados (ver Figura 45). Para ello, hacemos clic sobre el fichero "*ResutsDEAaño-mes-dia_hora:minuto:segundo.xlsx*" y seleccionamos la opción "*View file*".

Figura 45. Excel con resumen de resultados.



Guardamos el script "*Resumen_DEA*".

Ahora, en el Ejemplo 13 vamos a replicar los resultados de León, et al. (2003)[21].

Ejemplo 13. Resumen de resultados: modelo DEA fuzzy.

León et al. (2013) utilizan técnicas de programación posibilística para medir la eficiencia basada en la forma envolvente del modelo BCC input-orientado. Los autores utilizados por los autores en su artículo se encuentran en el dataset "Leon2003", que sumistrado con **deaR**.

Creamos un nuevo script: "*Resumen_DEA_fuzzy*". Para replicar los resultados que los autores muestran en la Tabla 2 (página 419), escribimos en el script:

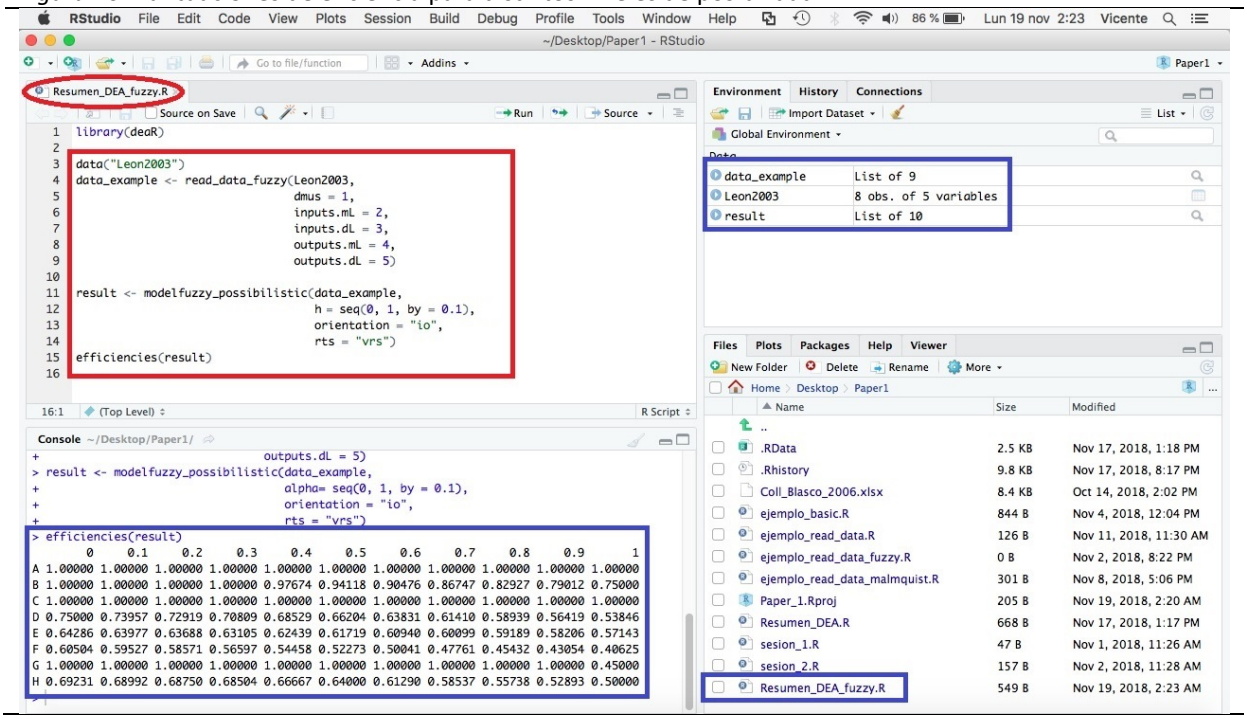
```
data("Leon2003")
data_example <- read_data_fuzzy(Leon2003,
                                dmus = 1,
                                inputs.mL = 2,
                                inputs.dL = 3,
                                outputs.mL = 4,
                                outputs.dL = 5)

result <- modelfuzzy_possibilistic(data_example,
                                   h= seq(0, 1, by = 0.1),
                                   orientation = "io",
                                   rts = "vrs")
efficiencies(result)
```

[21] Léon, T.; Liern, V. Ruiz, J.; Sirvent, I. (2003). "A Possibilistic Programming Approach to the Assessment of Efficiency with DEA Models", Fuzzy Sets and Systems, 139, 407–419. https://doi.org/10.1016/S0165-0114(02)00608-5

Nota: h=seq(0, 1, by=0.1) genera una secuencia de valores para los diferentes niveles de posibilidad: h=(0, 0.1, 0.2,…,1).

ejecutamos las **instrucciones**. En la *Consola* (ver Figura 46) se mostrarán las puntuaciones de eficiencia del modelo BCC input-orientado para los distintos niveles de posibilidad *h*.

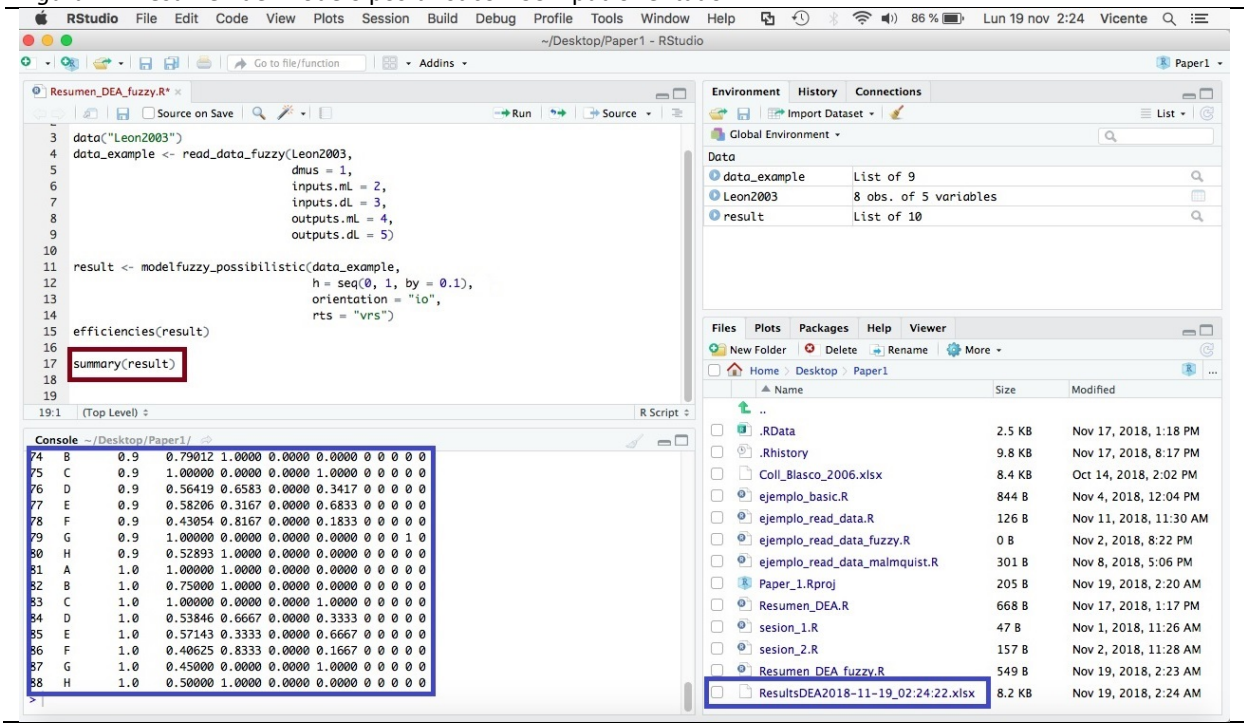
Figura 46. Puntuaciones de eficiencia para distintos niveles de posibilidad.



Para obtener un resumen de resultados en pantalla (ver Figura 47) y exportarlos a un fichero Excel escribimos la siguiente instrucción en el script y la ejecutamos:

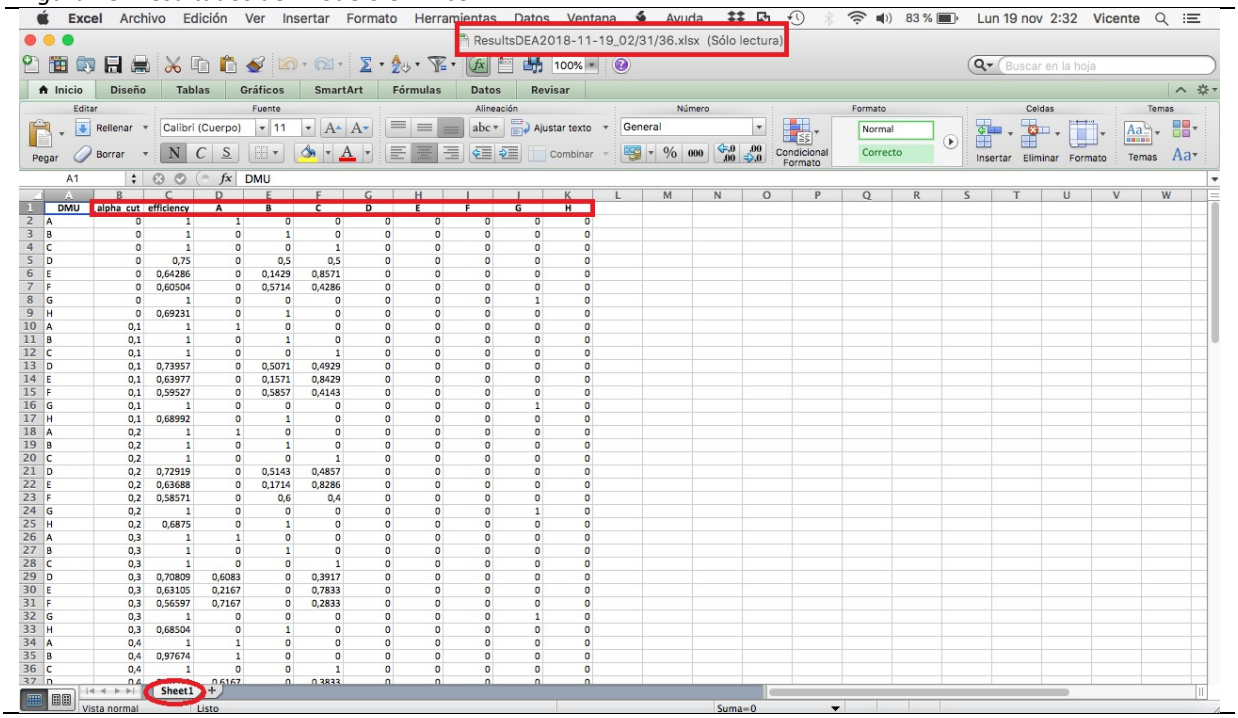
summary(result)

Figura 47. Resumen del modelo posibilistico BCC input-orientado.



En el modelo posibilístico el resumen de resultados consta de: (1) las puntuaciones de eficiencia y (2) los conjuntos de referencia. Estos son los resultados que se exportan al libro de Excel (ver Figura 48).

Figura 48. Resultados del modelo en Excel.



Guardamos el script "*Resumen_DEA_fuzzy*".

7.6. Representaciones gráficas. La función plot().

Con la función plot() podemos obtener algunas representaciones gráficas de los resultados obtenidos al ejecutar un modelo DEA convencional, la eficiencia cruzada o el índice de productividad de Malmquist. En los ejemplos 14, 15 y 16 se muestra cómo obtener los gráficos de estos modelos.

La función plot() se utiliza de la siguiente forma:

plot(*x*)

donde **x** es el objeto donde se almacenan los resultados de un modelo DEA.

Después de usar la función plot() el gráfico aparecerá en el *Viewer* (ventana inferior derecha). Podemos guardar un gráfico haciendo clic en la pestaña *Export* del *Viewer*. Es posible guardar un gráfico como una imagen en formato: "*png*", "*jpeg*" o "*tiff*". También podemos copiar el gráfico en el portapapeles y pegarlo, por ejemplo, en un document Word.

En esta versión 1.0 no están disponibles gráficos para modelos DEA fuzzy.

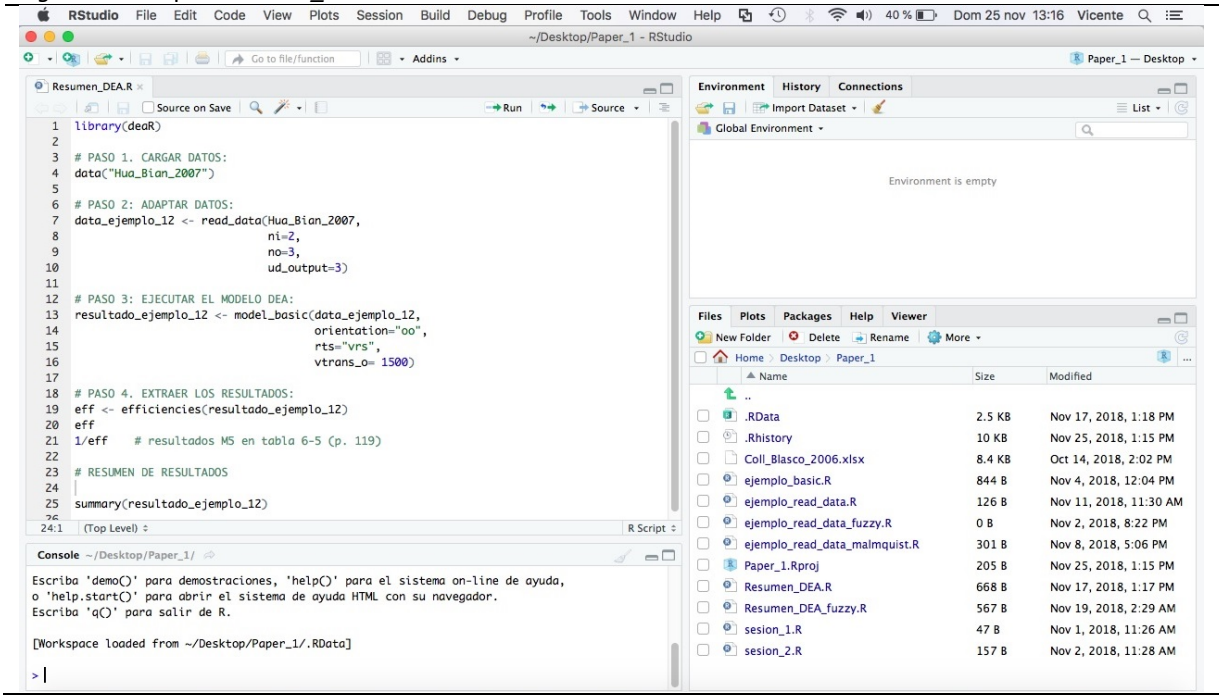
Actualmente estamos trabajando para mejorar las salidas gráficas de **deaR**, que serán incorporadas en la próxima versión.

Ejemplo 14. Plot: modelo DEA básico.

Abrimos el script "*Resumen_DEA*". Si cerramos la sesión de trabajo: abrimos el proyecto "*Paper_1*", cargamos **deaR** y abrimos el script.

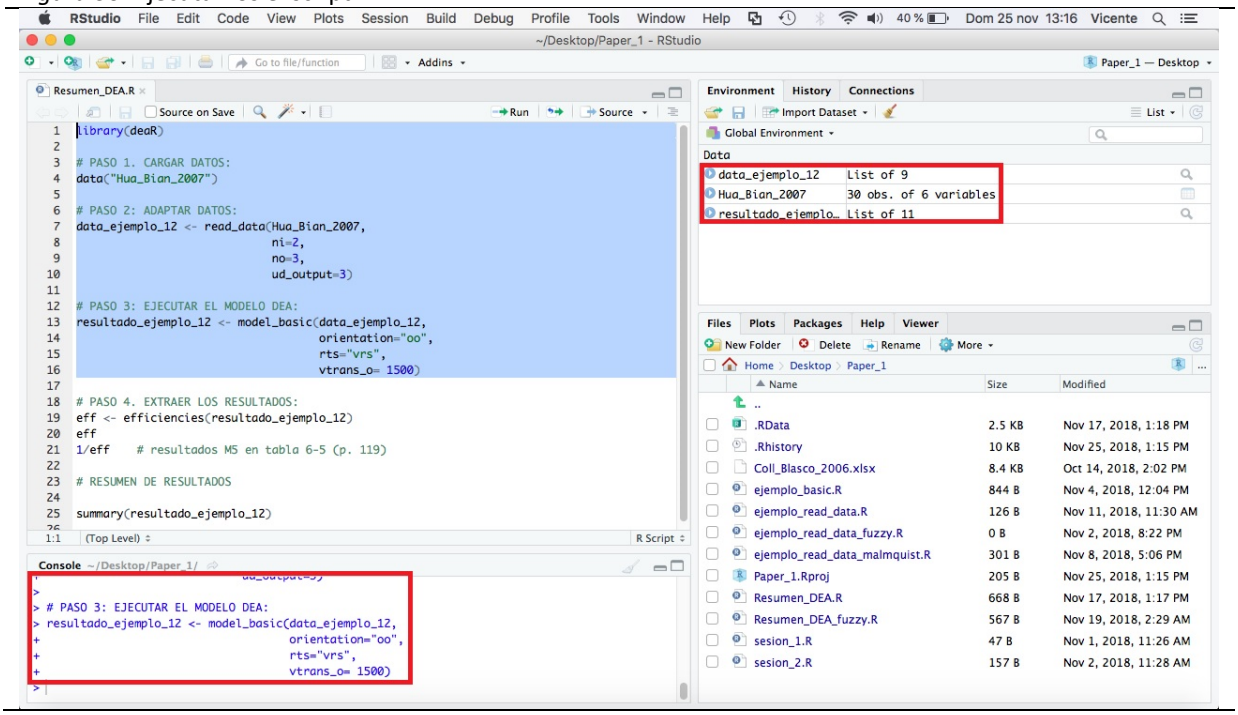
Nuestra sesión de trabajo debería ser similar a la que se muestra en la Figura 49.

Figura 49. Script "*Resumen_DEA*".



Como se muestra en la Figura 50, seleccionamos desde la línea 1, **library(deaR)**, hasta la línea 16 y ejecutamos la selección. Inmediatamente, en el *Environment* se listarán tres objetos: "*data_ejemplo_12*", "*Hua_Bian_2007*" y "*resultado_ejemplo_12*".

Figura 50. Ejecutamos el script.



Los resultados del modelo DEA están almacenados en el objeto "*resultado_ejemplo_12*". Para representar gráficamente algunos de estos resultados vamos a escribir en la línea la 27 del script la siguiente instrucción:

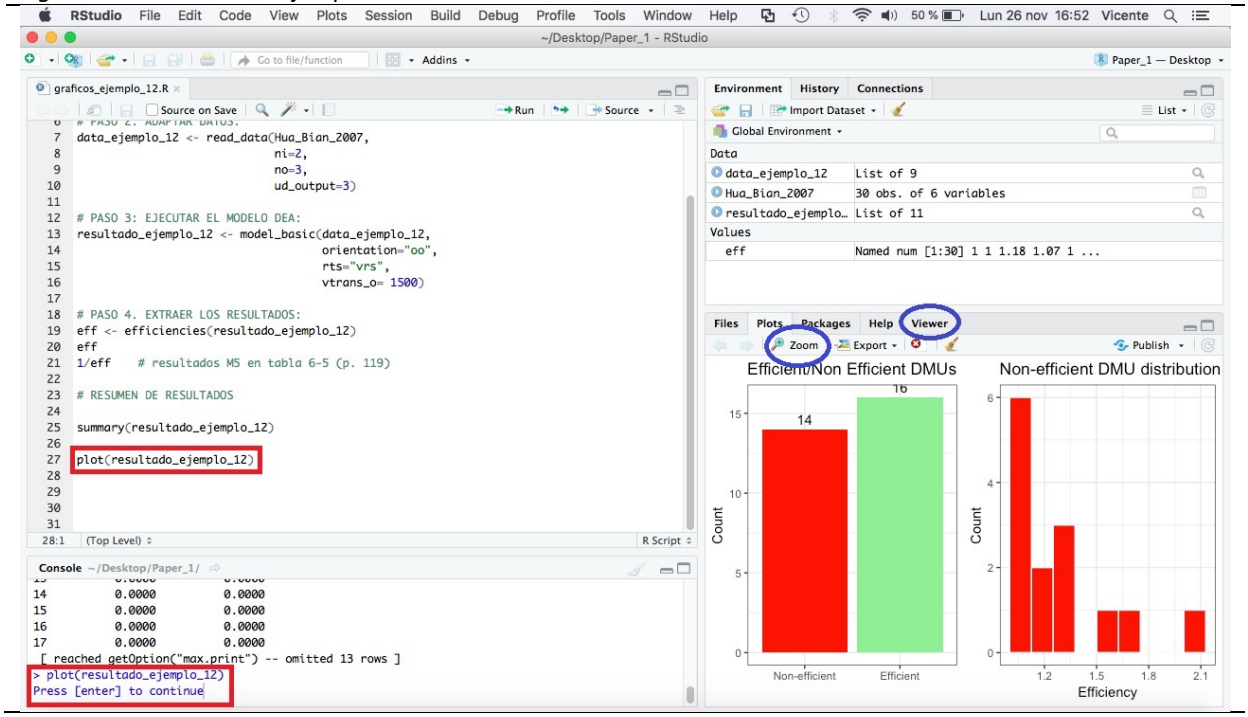
plot(resultado_ejemplo_12)

y la ejecutamos. En la *Consola* aparece el mensaje:

Press [enter] to continue

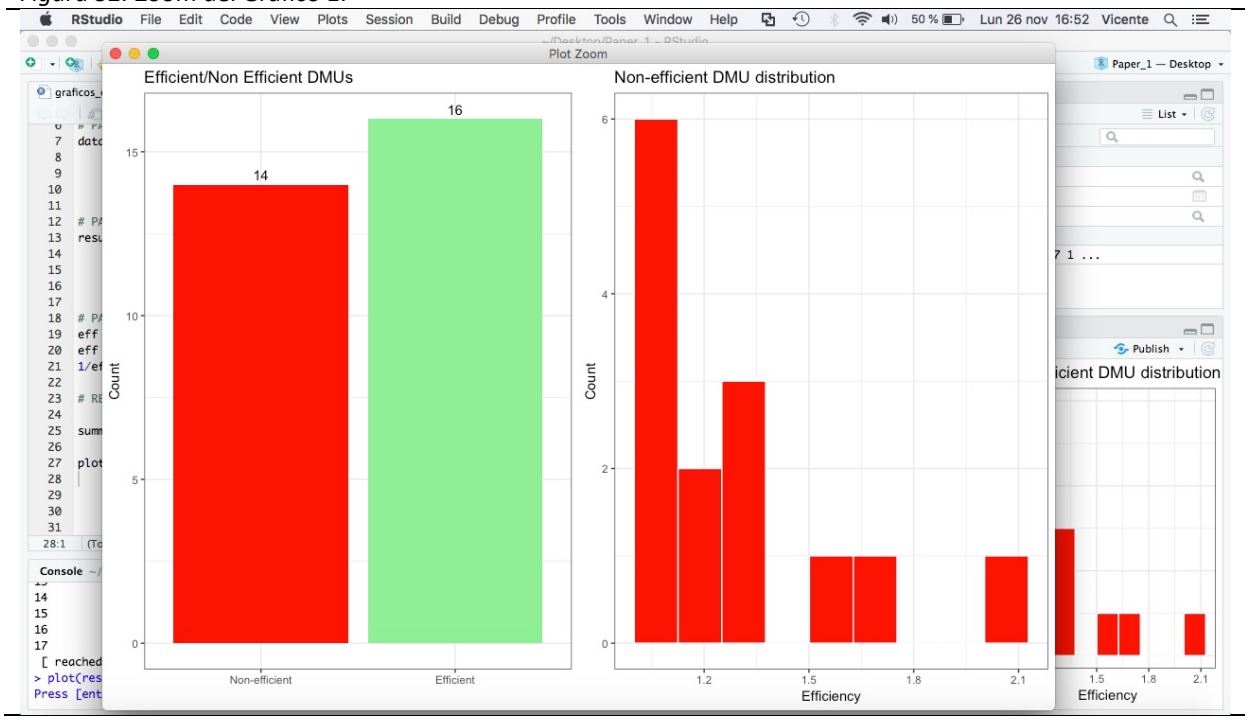
y en la pestaña Viewer (ventana inferior derecha) se mostrará un primer gráfico de resultados, como podemos ver en la Figura 51.

Figura 51. Gráfico 1 del ejemplo 12.



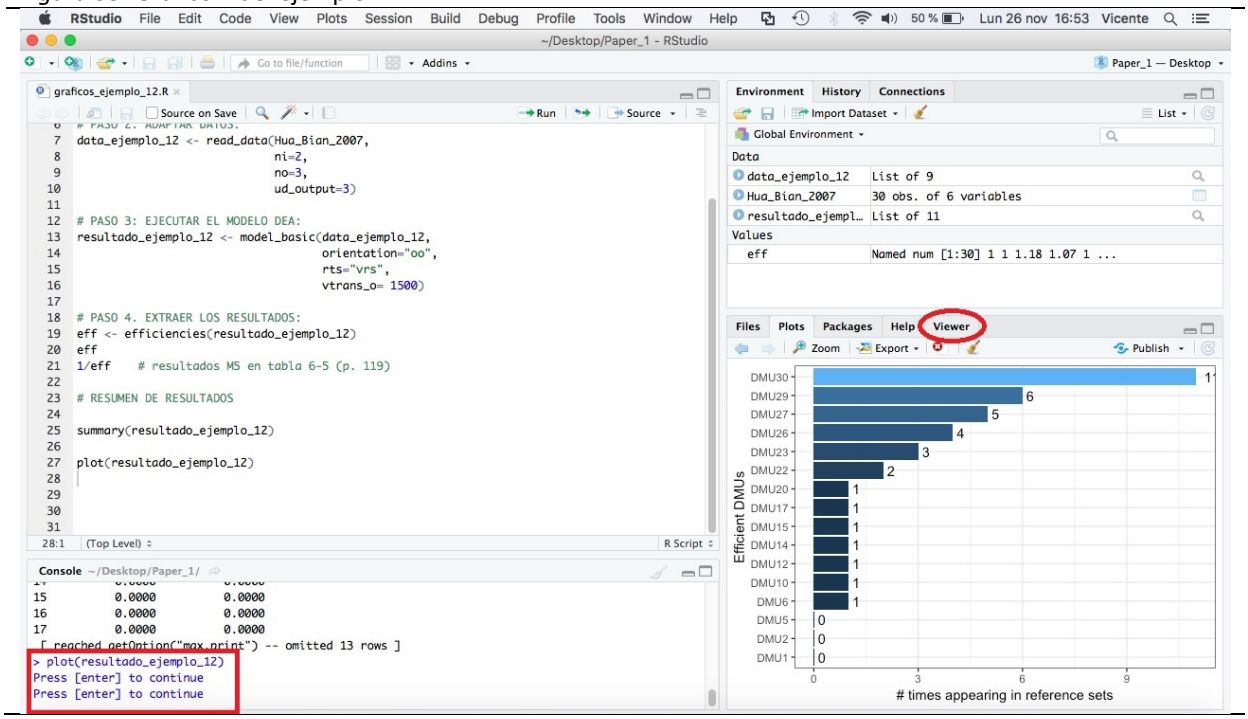
En el gráfico anterior se han representado el número de DMUs eficientes y no eficientes, así como la distribución de la puntuación de eficiencia de las DMUs ineficientes. Si hacemos clic en el botón de Zoom se ampliará el gráfico (ver Figura 52).

Figura 52. Zoom del Gráfico 1.



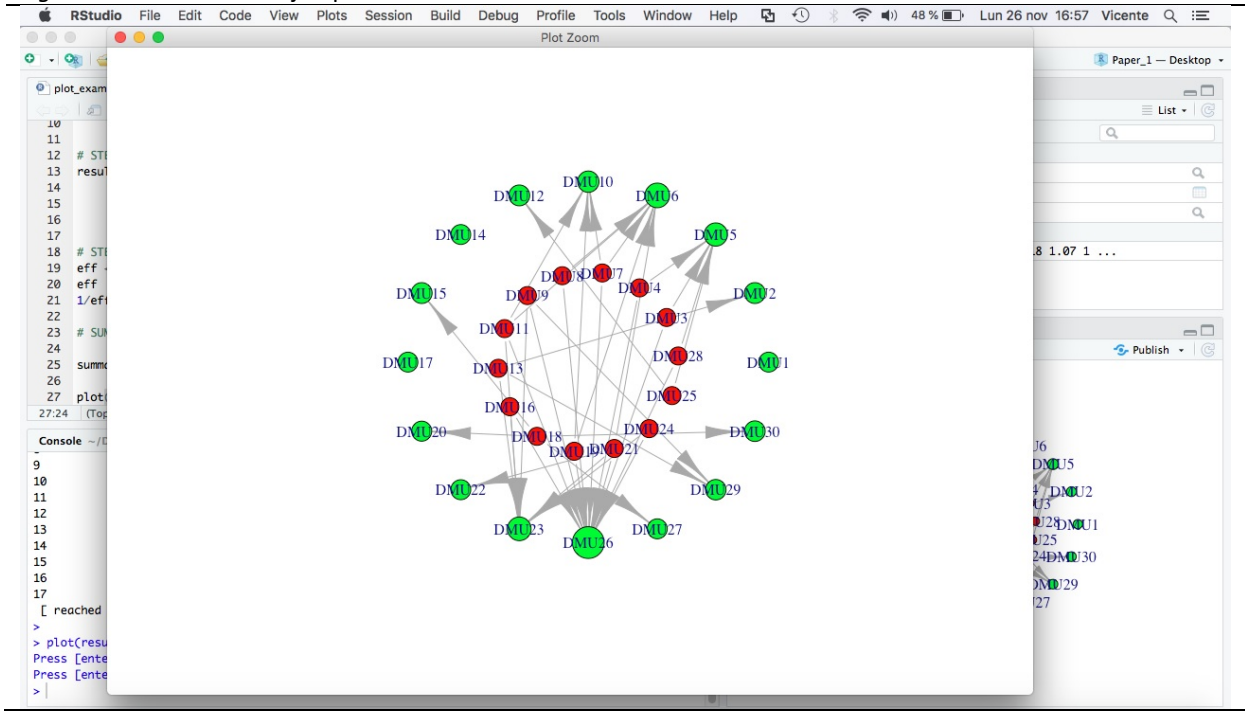
Al pulsar la tecla Enter aparecerá el segundo gráfico (ver Figura 53). En esta ocasión se representa el número de veces que una DMU eficiente forma parte del conjunto de referencia de DMUs ineficientes.

Figura 53. Gráfico 2 del ejemplo 12.



Por último, al pulsar nuevamente la tecla *Enter* se visualiza el último gráfico (ver Figura 54).

Figura 54. Gráfico 3 del ejemplo 12.



En la Figura 54 vemos un gráfico de redes en el que los círculos verdes representan las DMUs eficientes y los círculos rojos las ineficientes. En este gráfico puede verse cómo se relacionan las DMUs ineficientes con las eficientes, que se sitúan en el exterior tratando de transmitir la idea de que forman la frontera eficiente. Además, podemos observar que no todos los

círculos verdes tienen el mismo diámetro. En este caso, el tamaño del círculo pretende transmitir la idea de la de la DMU eficiente para el conjunto de DMUs ineficientes.

Guardamos el script con el nombre: "*graficos_ejemplo_12*".

Ejemplo 15. Plot: índice de Malquist.

Creamos una nuevo script y lo llamamos "*graficos_malmquist*". Si cerramos la sesión de trabajo: abrimos el proyecto "*Paper_1*", cargamos **deaR** y creamos el script.

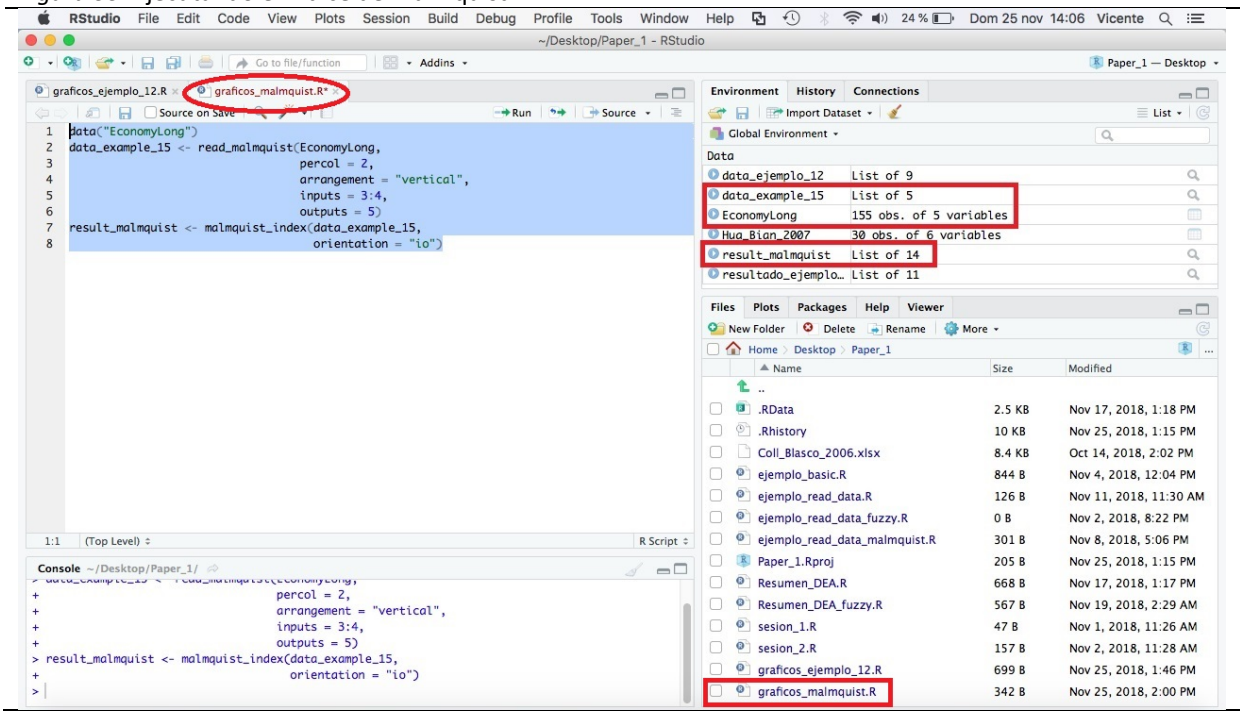
Vamos a ejecutar un modelo Malmquist. Para ello, escribimos en el script:

```
data("EconomyLong")
data_example_15 <- read_malmquist(EconomyLong,
                                  percol = 2,
                                  arrangement = "vertical",
                                  inputs = 3:4,
                                  outputs = 5)
result_malmquist <- malmquist_index(data_example_15,
                                    orientation = "io")
```

Nota: Los datos están en formato largo.

Ejecutamos las instrucciones del script (ver Figura 55).

Figura 55. Ejecutando el índice de Malmquist.



Los resultados del índice de Malmquist están almacenados en el objeto "*result_malmquist*". Por tanto, para obtener las gráficas escribimos en el script:

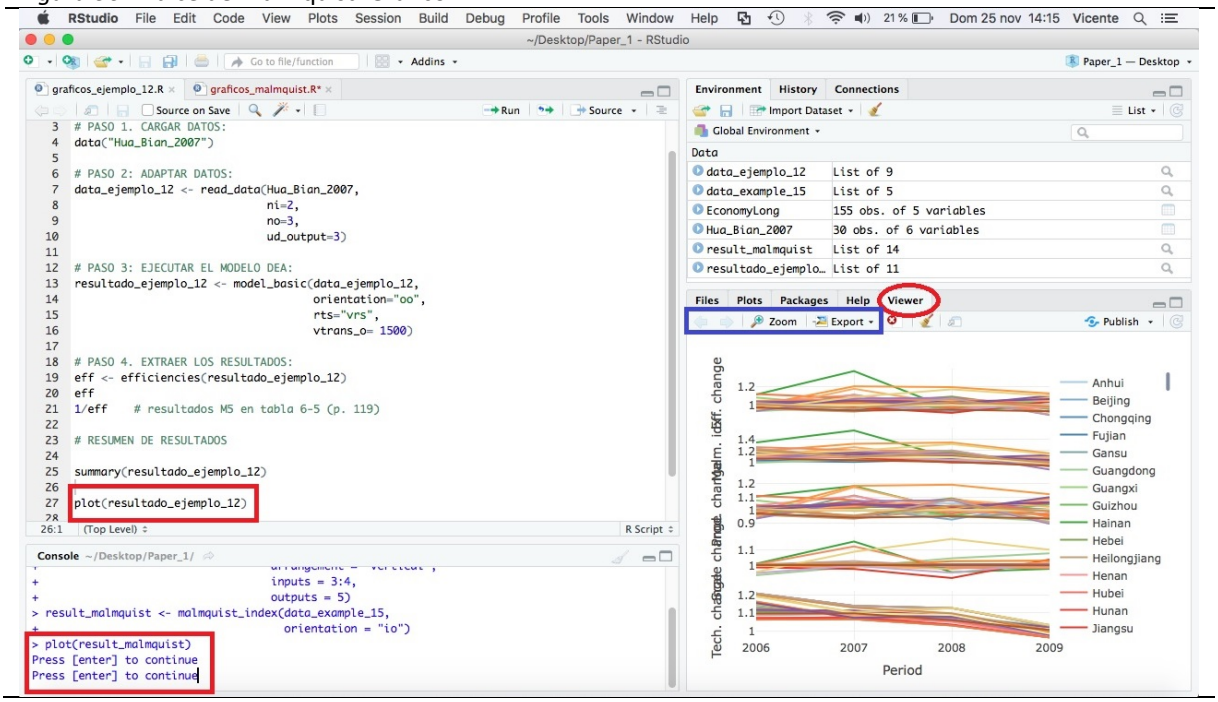
```
plot(result_malmquist)
```

y ejecutamos la instrucción. En la *Consola* aparecerá el mensaje:

Press [enter] to continue

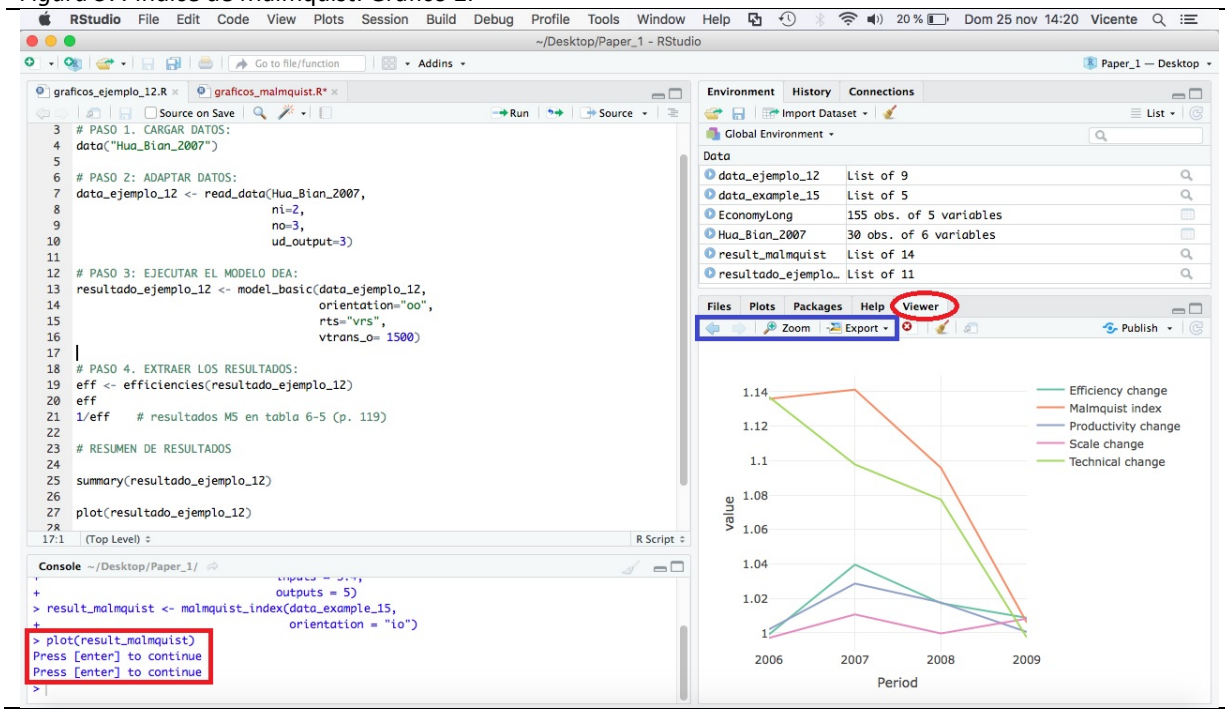
Al pulsar la tecla *Enter* aparecerá un gráfico en la pestaña *Viewer* (parte inferior izquierda) (ver Figura 56). Podemos hacer clic en *Zoom* para ver mejor el gráfico. En este primer gráfico se representan los distintos componentes del índice de Malmquist por DMU. Si hay muchas DMUs no se verá gran cosa, pero es posible seleccionar DMUs de interés.

Figura 56. Índice de Malmquist: Gráfico 1.



Si volvemos a pulsar la tecla *Enter* (en la *Consola*), se crea un segundo gráfico (ver Figura 57). En esta ocasión, se representan los componentes del índice de Malmquist por periodo. Podemos ampliar el gráfico haciendo clic sobre *Zoom* y avanzar (o retroceder) gráficos haciendo clic sobre las flechas. En este gráfico también es posible seleccionar los componentes del índice de Malmquist que queremos mostrar.

Figura 57. Índice de Malmquist: Gráfico 2.



Guardamos el script: "*graficos_malmquist*".

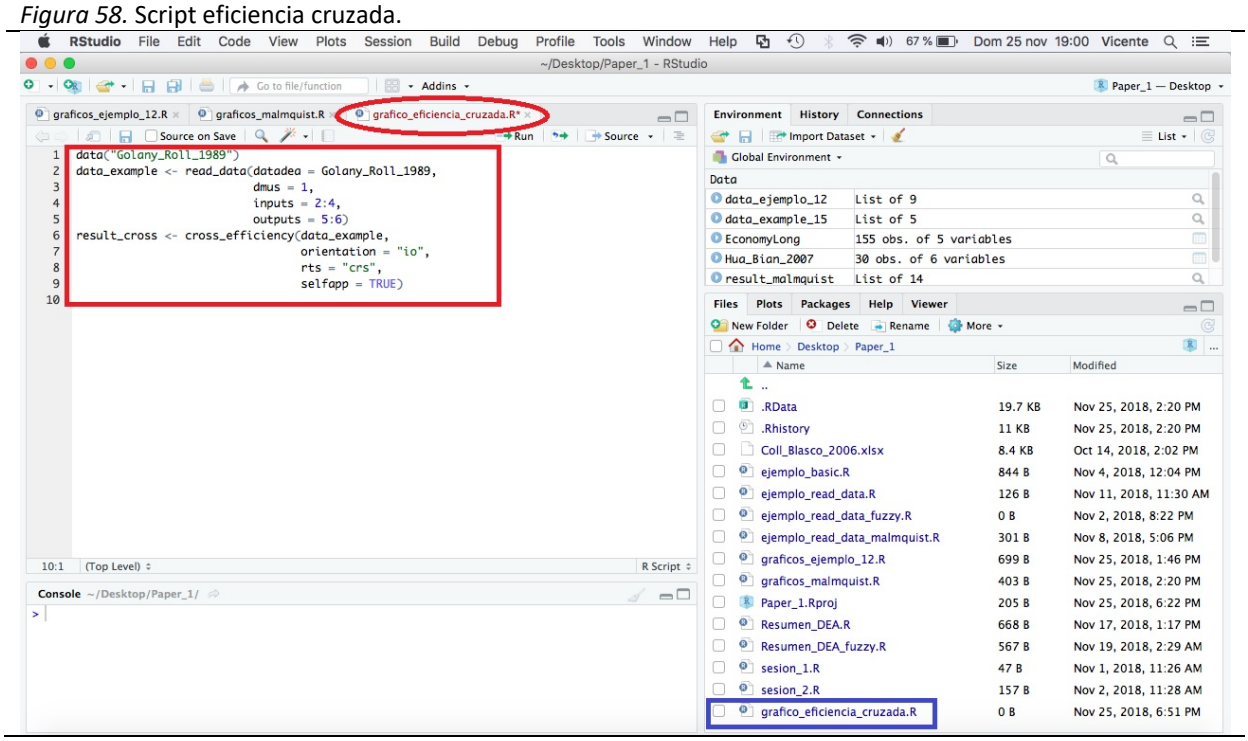
Ejemplo 16. Plot: Eficiencia cruzada.

Creamos una nuevo script y lo llamamos "*grafico_eficiencia_cruzada*". Si cerramos la sesión de trabajo: abrimos el proyecto "*Paper_1*", cargamos **deaR** y creamos el script.

Escribimos y ejecutamos las siguientes instrucciones (ver Figura 58):

```
data("Golany_Roll_1989")

data_example <- read_data(datadea = Golany_Roll_1989,
                          dmus = 1,
                          inputs = 2:4,
                          outputs = 5:6)

result_cross <- cross_efficiency(data_example,
                                 orientation = "io",
                                 rts = "crs",
                                 selfapp = TRUE)
```

Figura 58. Script eficiencia cruzada.

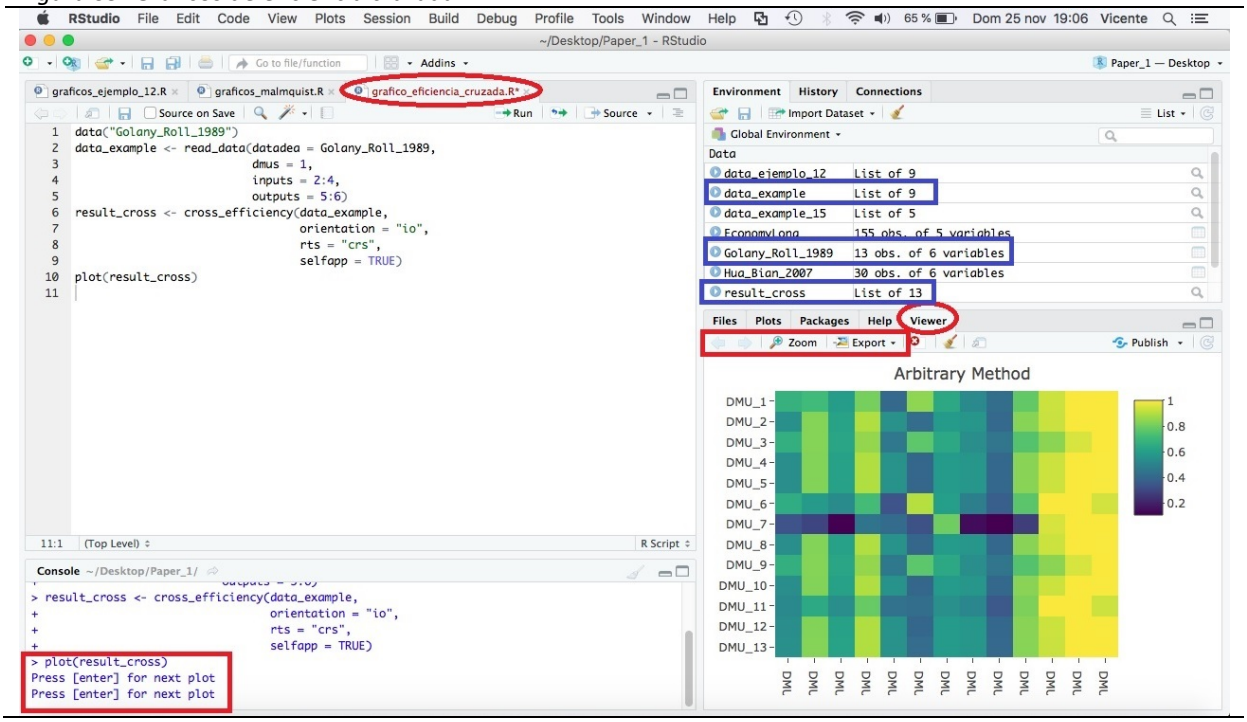


En el objeto "*result_cross*" se han almacenado los resultados de la eficiencia cruzada correspondientes a los modelos: arbitrario, benevolente y agresivo. Todos estos resultados son mostrados en forma de mapa de calor con la función plot(). Para ello, escribimos en el script:

```
plot(result_cross)
```

y pulsamos la tecla *Enter* en la *Consola* para mostrar los distintos gráficos. El resultado debería ser similar al que se muestra en la Figura 59.

Figura 59. Gráficos de eficiencia cruzada.



Guardamos el script "*grafico_eficiencia_cruzada*", cerramos el proyecto y salimos de RStudio.

Esperamos que **deaR** te sea útil y lo uses tanto en investigación como en docencia.

Agradeceremos cualquier comentario y sugerencia para mejorar **deaR.**