

# **Manual de introducción a R**

Creado por: Dr. Leonel Lerebours y Dr. Yunion Rosario Editado por: Virginia Vallejo

2025-07-29

# Contenido

# 1 Introducción

## 1.0.1 Consideraciones y recomendaciones de como mejorar el autoaprendizaje en R (¡y en otra herramienta similar!)

Este manual es muy específico para el entrenamiento de análisis de datos en salud, tanto para el nivel básico como intermedio, y tiene como propósito introducir el uso de R para ayudarte a facilitar el proceso de desarrollo de tus productos de investigación. Con esto dicho, hay temas que bien pudieran estar contenidos en un documento como este, pero por lo específicos que son, no aparecen detallados en este manual; por ejemplo, no abordaremos conceptos estadísticos básicos como las medidas de tendencia central, por tanto, partimos del supuesto de que ya tienes cierta familiaridad con estos temas.

El enfoque principal es que puedas comenzar a usar R, primero adaptándote al formato de comandos en vez del formato orientado a objetos, (que es la manera como aprendiste Excel o SPSS) y luego haciendo las tareas más comunes que se realizan rutinariamente tanto en el entrenamiento como en la práctica, es decir, hacer tablas, gráficos, exportar reportes, etc.; todo esto a un nivel introductorio.

El mundo de R es bien grande, (¡y qué bueno que es así!) y al principio pudiera ser tedioso para ti aprender a usarlo, sin embargo, desde que se comiences a producir resultados, notarás lo rápido que avanzarás. Una de las razones por las que R es una solución muy robusta para el análisis de grandes volúmenes de datos y para la elaboración de reportes, es que hay una comunidad muy vasta de personas que van aportando sus conocimientos y experiencia para hacer más fácil el aprendizaje y el uso de R.

En nuestros inicios, antes de ser instructores, fueron muchos intentos de ensayo y error, es decir, era agotador buscar en la web, “googlear” operaciones bien básicas del tipo “Cómo importar una base de datos de Excel a R”, o “Como hacer una tabla 2x2 en R”, o “Cómo calcular la diferencia en días de una misma columna de una variable fecha en un dataframe” (esta última en Excel es bien fácil, pero cuando hay que hacerlo con una base de datos bien grande, Excel colapsa mientras que con R es más rápido); otra tarea de apoyo en nuestros inicios era leer tutoriales, aplicarlos, obtener resultados erróneos, corregir, preguntar... en fin, dar muchas vueltas hasta dar con lo que andábamos buscando.

Este tipo de esfuerzos en la actualidad es muy diferente actualmente debido al uso de la inteligencia artificial como una plataforma que provee facilidades para avanzar rápido con el aprendizaje de cualquier lenguaje informático. Lo que ha sido un común denominador es que siempre hay respuestas, y muchas, de cómo hacerlo de diferentes formas, cualquier cosa

(¡Relacionada al análisis de datos!). En pocas palabras, cualquier problema o tarea que quieras hacer en R es muy probable que ya alguien la haya hecho.

Este manual es un ejemplo de eso, pues aprendimos a usar este lenguaje que literalmente es difícil, sin embargo, lo hemos simplificado para ti, ayudándote a reducir considerablemente el tiempo de búsqueda en tutoriales y Chat GPT, combinando el lenguaje de R y el lenguaje clínico. Lo más interesante de aprender a usar R con este estilo de ensayo y error es que aprendimos cosas que no teníamos la menor idea de que existían, por ejemplo, la teoría de los colores, la gramática de los gráficos, ejercicios estadísticos que pensábamos eran imposibles hacer en R, sin embargo, por la cantidad de tutoriales, ayudas, videos, cursos (la gran mayoría gratuitos o de muy bajo costo), hemos tenido la oportunidad de aprender más allá de lo esperado y te lo traemos resumido y bajo una metodología práctica, para que puedas aplicarlo de inmediato en tus tareas cotidianas.

De una forma u otra, el mundo de R representa un gran avance donde converge la tecnología y la buena voluntad de muchas personas, sin fines de lucro, que realmente creen en el crecimiento social a través de la información; por herramientas como R se ha popularizado el uso de la ciencia de datos, que es muy usada en muchas profesiones incluyendo la epidemiología, y ha facilitado mucho el proceso de difundir información, como las publicaciones de artículos. Entonces, como parte de la experiencia de comenzar a usar R es el autoaprendizaje, te invitamos como primera tarea o ejercicio a que “busques” qué es R, quiénes lo crearon, para qué lo crearon, y haciendo esta actividad te vas a dar cuenta lo mencionado en las líneas anteriores sobre la vasta cantidad de información disponible en cuanto al uso de R para la gestión de datos.

Por último, el ingrediente principal para aprender este lenguaje es tener en qué usarlo, es decir, una necesidad, en este caso, sobre procesamiento, análisis y reporte de datos. Esperamos que te motives a implementar esta herramienta y que te sea de provecho este manual, para que puedas seguir aprendiendo más de lo que esperas aprender durante este curso.

### 1.0.2 Definiciones claves

- **Objetos:** Son elementos que almacenan información y pueden ser de diferentes tipos como un valor simple (numérico o carácter), un dataframe, un vector, una función, una lista, gráficos entre otros. De forma abstracta un objeto es un contenedor que puede almacenar uno o varios elementos. Estos se “almacenan” en el ambiente de trabajo que reside en la memoria de la PC. Los objetos se crean usando el operador de asignación “<-” o signo “=”.
- **Dataframe:** Es el equivalente a una base de datos en formato de tabla o listado, donde cada columna es una variable y cada fila es una observación.
- **Vector:** Un vector es un objeto que consiste una lista de elementos que son del mismo tipo.

- **Rutina:** Es un conjunto de comandos o códigos que cumplen con una tarea en específico, ejemplo la importación de una base de datos y su limpieza (en inglés se le dice *script*)
- **Paquete:** Son varios archivos que contienen funciones, documentación, bases de datos. Son fundamentales para añadir funcionalidad a R. También es la vía como se comparte códigos re-producibles (funciones). A modo de analogía un paquete vendría siendo una herramienta (destornillador, martillo) y R sería la mesa de trabajo, donde cada vez que necesitas una herramienta (paquete) la traes a la mesa de trabajo.
- **Proyecto:** “Un proyecto es una carpeta de trabajo designado con un archivo .RProj. Al abrir un proyecto (usando Archivo -> Abrir proyecto en RStudio o haciendo doble clic en el archivo .Rproj fuera de R), la carpeta de trabajo se establecerá automáticamente en el directorio donde se encuentra el archivo .RProj.
- **Bases de datos** en diferentes formatos, preferiblemente en .csv, pero en Excel u otros formatos comunes se pueden trabajar)
- **Operadores :** Según en el libro de R para principiantes de Juan Bosco Mendoza (Vega, n.d.): *“Son los símbolos que le indican a R que debe realizar una tarea. Combinando datos y operadores es que logramos que R haga su trabajo. Existen operadores específicos para cada tipo de tarea. Los tipos de operadores principales son los siguientes: Aritméticos, Relacionales, Lógicos y De asignación.”*
- **Expresión:** En programación, una expresión es una combinación de constantes, variables o funciones, que es interpretada de acuerdo a las normas particulares de precedencia y asociación para un lenguaje de programación en particular.
- **Código:** son los signos y símbolos que tienen un significado o en sus diferentes combinaciones se pueden interpretar como un lenguaje
- **Función:** es un bloque de código reutilizable que realiza una tarea específica, tomando argumentos como entrada y devolviendo un resultado, lo que facilita la organización y reutilización del código.
- **Argumento:** es un valor o variable que se pasa a una función para que la función pueda realizar su tarea específica
- **Variable:** es un nombre que se le asigna a un valor o objeto, que puede ser de diferentes tipos (números, texto, etc.) y que se almacena en la memoria de la computadora.

## 2 Preparación del ambiente de trabajo

### 2.1 Como se instala R y Rstudio®

El primer paso para esta tarea es descargar el programa R, que se encuentra en la página web <https://cran.r-project.org/> . en esta hay varias versiones dependiendo que sistema operativo estás usando, ya sea Windows, macOS o Linux.

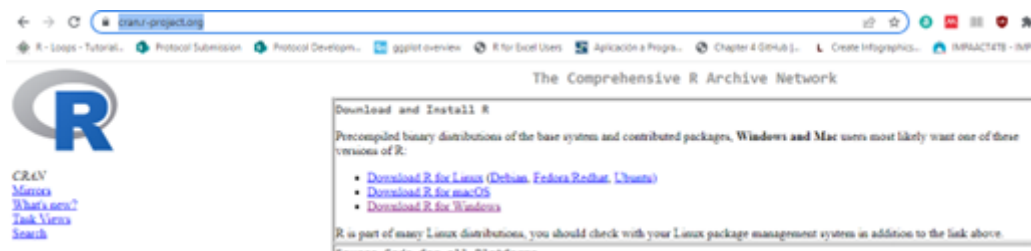


Figura 2.1: En esta página trata de descargar la versión más actualizada

Luego de descargarlo puedes instalarlo inmediatamente usando el ejecutable (para Windows y macOS) o la forma como se instalan software in Linux. Para este manual estamos en un ambiente de Windows, luego de instalar podemos acceder a la **consola** de R (una pantalla con un cursor parpadeando donde podemos escribir)



Figura 2.2: Este es la consola de R, aquí podemos comenzar a usar R, pero es un poco complejo usar solamente la consola

Aunque ya hayamos instalado R, al menos que seamos muy expertos en su uso, que conozcamos bien las sintaxis de las funciones, que *objetos* (más tarde veremos lo que son los objetos) están cargados en la memoria, entre otras cosas (¡también se puede usar como una simple calculadora!), es muy complicado usar solamente la consola, por lo que tenemos disponible aditamentos o accesorios que nos permiten trabajar más fácil con R, así como su aprendizaje. Para este manual el aditamento que usaremos es **Rstudio** de [Posit®](#).

### 2.1.1 Descargar e instalar Rstudio

Ahora vamos a instalar Rstudio, para esto vamos a descargarlo de la siguiente página web, [Descarga de Rstudio](#) donde vamos a buscar la versión gratuita de escritorio para para el sistema operativo que estamos usando.

En caso de que estés usando Linux, están las instrucciones en la página de como hacerlo, la versión de Windows y macOS es un ejecutable.

Luego de descargar la ultima versión disponible procedemos a instalar Rstudio, haciendo clic en el ejecutable, la instalación (en la versión de Windows por ejemplo) es muy similar a cualquier otro software que donde te pregunta el lugar donde será instalado y varias ventanas donde se ve el progreso de instalación. Si todo salió bien, es decir que se instaló sin errores, pues tendremos disponible en la barra de acceso directo y en el escritorio, (si elegimos esta opción durante la instalación) también tendremos un acceso directo.

Si tienes Windows, te compartimos el paso a paso en [este video](#) para descargar e instalar R y también R Studio Desktop.

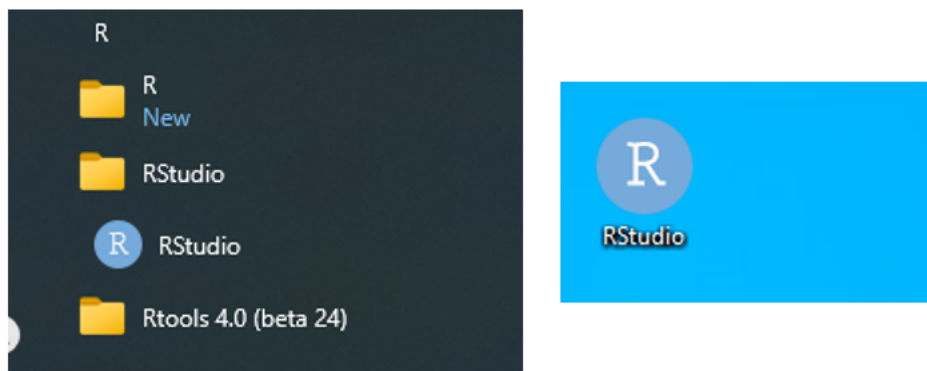


Figura 2.3: En Windows, la barra de inicio o las aplicaciones y si se creó el acceso directo (derecha) en el escritorio

Otro software que se debe instalar si el sistema operativo que usas es Windows es **rtools**, dado que algunos paquetes necesitan que esté instalado para funcionar de forma correcta.

Este se adquiere desde la página de [Descarga rtools](#) y elige la versión más reciente y descarga el formato ejecutable (termina en “installer”) en cualquier unidad de almacenamiento. Su instalación sencilla, solo ejecutar el programa y hacer clic en siguiente las veces que sea necesario.

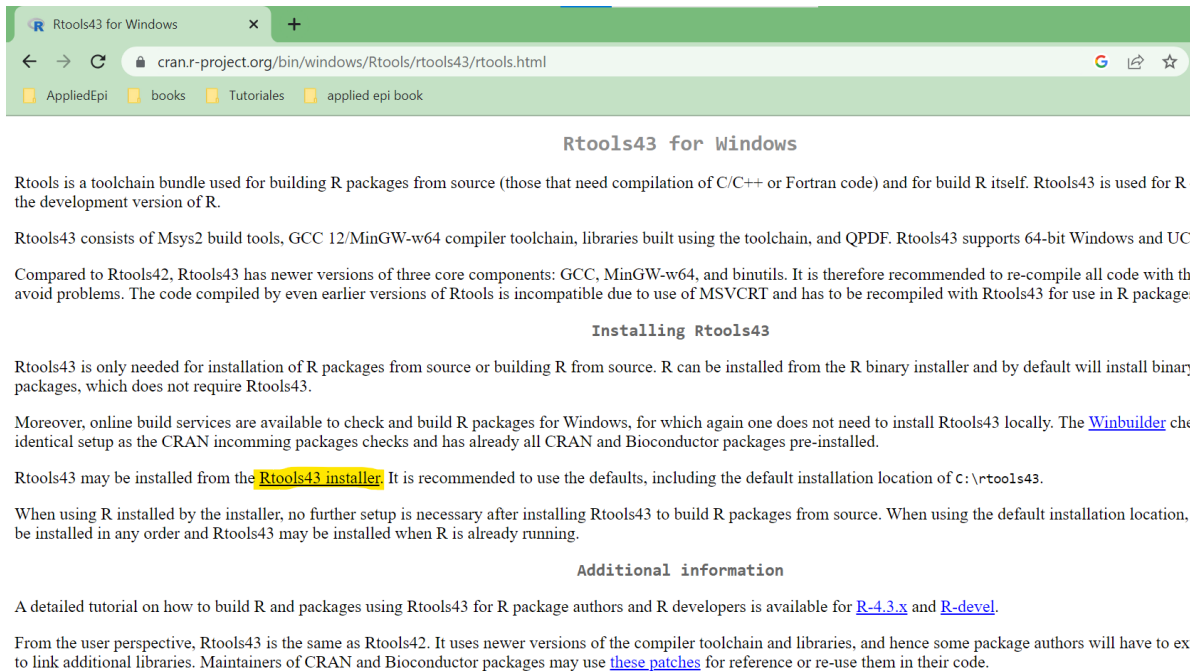


Figura 2.4: Busca esta version señalada en la imagen

## 2.2 Interfaz de Rstudio®

Para facilitar el aprendizaje y uso de R, vamos a usar RStudio que es un entorno de desarrollo integrado o IDE (Integrated Development Environment) y tiene la gran ventaja de que hay mucha documentación sobre su uso, es muy cómoda de trabajar porque nos ayuda con la escritura de los códigos, la organización de los archivos, tiene varios visores o paneles para ver los códigos, las salidas, los objetos cargados y otras ayudas más (Figura 5). RStudio ha permitido la difusión del uso de R, lo que ha permitido la propagación de su uso en todas las áreas donde se hace análisis de datos.



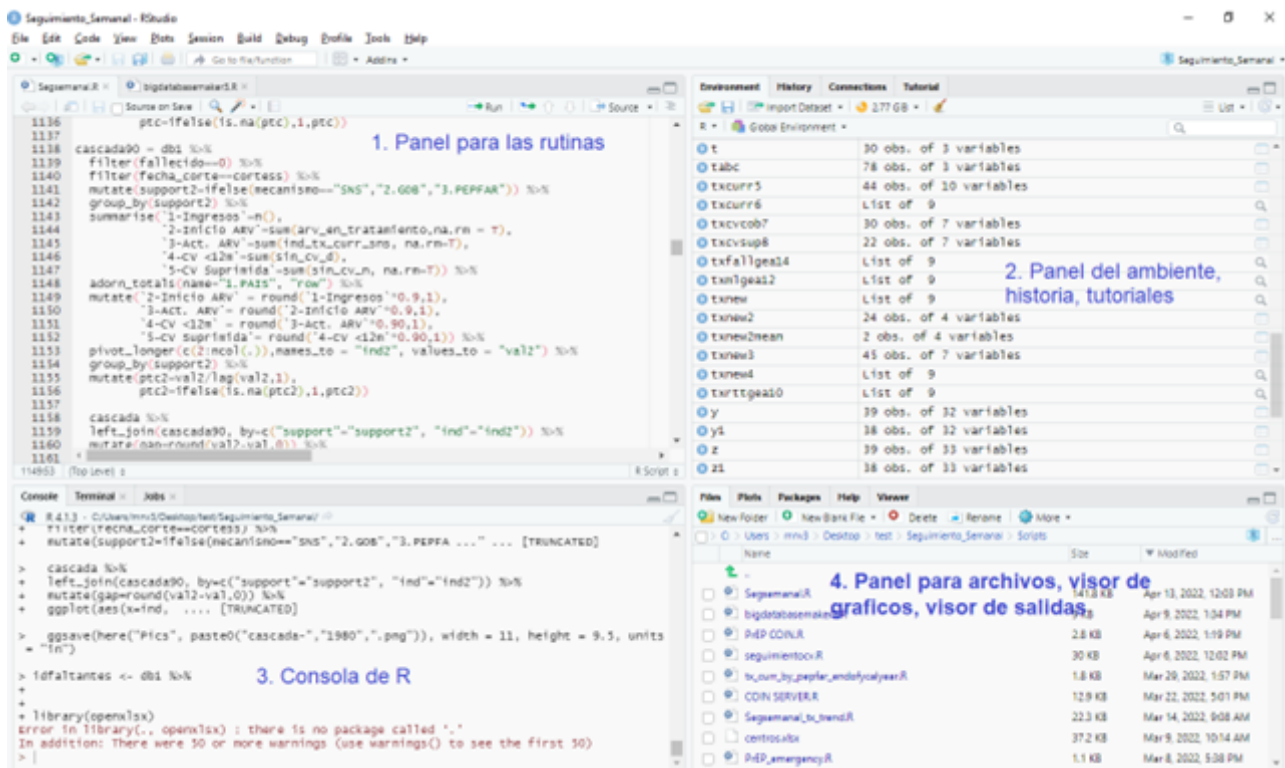


Figura 2.5: En esta imagen podemos ver los 4 paneles principales y el menú de la interfaz de Rstudio

- 1. Panel de las rutinas:** este panel es el editor de texto donde vamos a escribir los códigos (rutinas) que vamos a usar para nuestras tareas, como las tablas, los gráficos, etc., que son automáticamente ejecutadas en la consola de R. Cada rutina se puede guardar como un archivo (muy similar a un archivo de texto normal), la característica más importante de este panel es que nos permite ver si el código tiene errores, pues a veces la falta de una simple coma o un caracter nos arroja error cuando ejecutamos comandos. También nos ofrece la herramienta de autocompletar y nos facilita mucho la organización del código. Para ejecutar una línea de código simplemente ponemos el cursor al inicio o al final de la línea y presionamos Ctrl+Enter o en el botón “run”, mientras que para ejecutar la rutina completa, hacemos clic en el botón “source”. Más adelante veremos varios ejemplos.
- 2. Panel del ambiente de trabajo o panel de objetos cargados:** en este panel podemos ver cuales objetos tenemos cargados en la memoria del sistema, y nos permite ver qué tipo de objeto es (si es un *dataframe*, un *vector*, una *matriz*, una *función*, una *lista*). Esto nos ayuda a seguir el trabajo que vamos realizando, por ejemplo, cuando cargamos una base de datos desde un archivo de Excel en un objeto *dataframe* podemos ver cuantas filas y variables tiene el archivo. También a través de este panel podemos salvar la sesión de trabajo, esto es útil cuando hacemos una pausa y queremos retomar más adelante lo

que estábamos haciendo.

3. **Consola de R:** este es el lugar donde ocurre todo, puedes directamente escribir los comandos, las funciones, etc., pero para esto tenemos el panel de las rutinas, aquí también vas a ver los mensajes de errores cuando ejecutas una rutina o un comando, y por cada línea que se escribe se presiona “enter” para ejecutar los comandos. La ventaja más grande de utilizar RStudio es que hasta en la consola te identifica si hay errores o comandos incompletos y autocompletar.
4. **Panel para archivos, visor de gráficos y de salidas:** en este panel tenemos varias ventanas donde nos permite ver los archivos disponibles (muy parecido al explorador de Windows o a la carpeta de Documentos de tu computadora). A través del menú de este panel podemos crear carpetas, borrar, mover o copiar archivos, también podemos definir el directorio de trabajo (vamos a ver más adelante en detalle qué es el directorio o lugar de trabajo), a la derecha de Archivos o Files está el visor de gráficos (Plots), que podemos ampliar y poder copiar el gráfico que se está presentando y más a la derecha está el visor de salidas (Viewer), como tablas y datos en formato HTML. También en este panel está la ventana para instalar paquetes (término muy importante que lo veremos más adelante) y la ventana de ayuda (Figura 6).

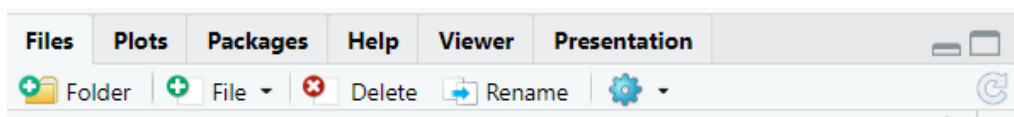


Figura 2.6: Figura 6.

Antes de comenzar a trabajar puedes ir familiarizándote con esta interface. En RStudio hay muchas opciones que iremos explicando en la medida que vayamos avanzando, mientras tanto, te mostrados dos herramientas interesantes del menú principal de RStudio: **Tools** y **Help**.

Si vas al menú principal de RStudio, puedes entrar en la sección que dice “**Tools**” y dentro del menú desplegable seleccionas la opción de “**Global Options**” (al final del menú desplegable), allí encontrarás un menú, y dentro de “**Appearance**” está la opción “Editor theme” donde podrás cambiar los colores y el tipo de letra de las ventanas, así podrás para adaptar el ambiente a tu gusto. En Ayuda (Help) puedes encontrar los accesos directos para usar el teclado, por ejemplo, salvar la rutina en la que estás trabajando, reiniciar R, ejecutar toda la rutina, entre otros atajos. En la Figura 7 te mostramos cómo llegar a cada una de estas herramientas.

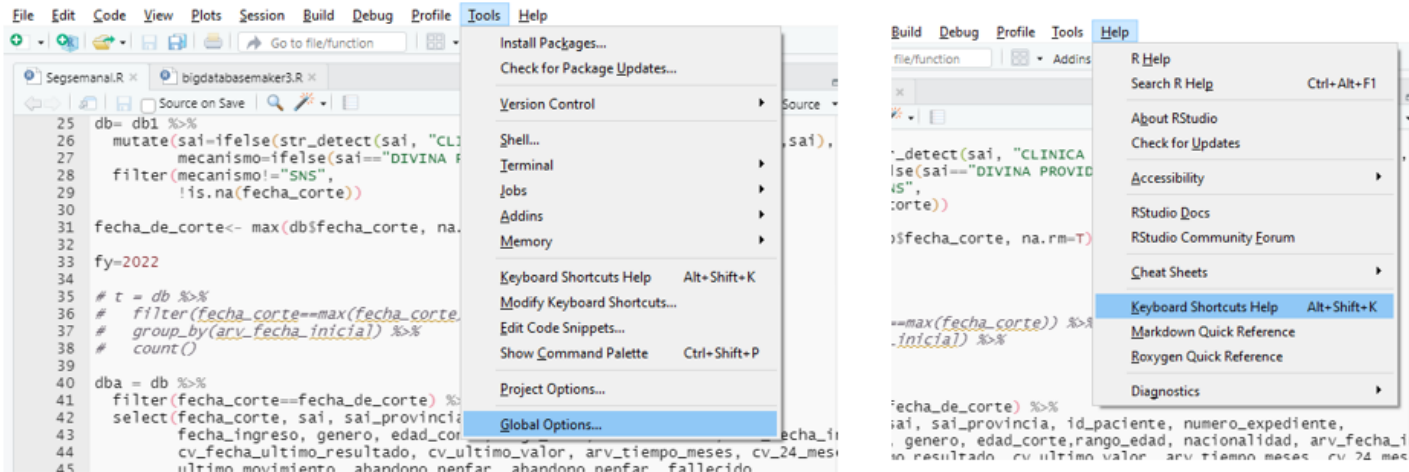
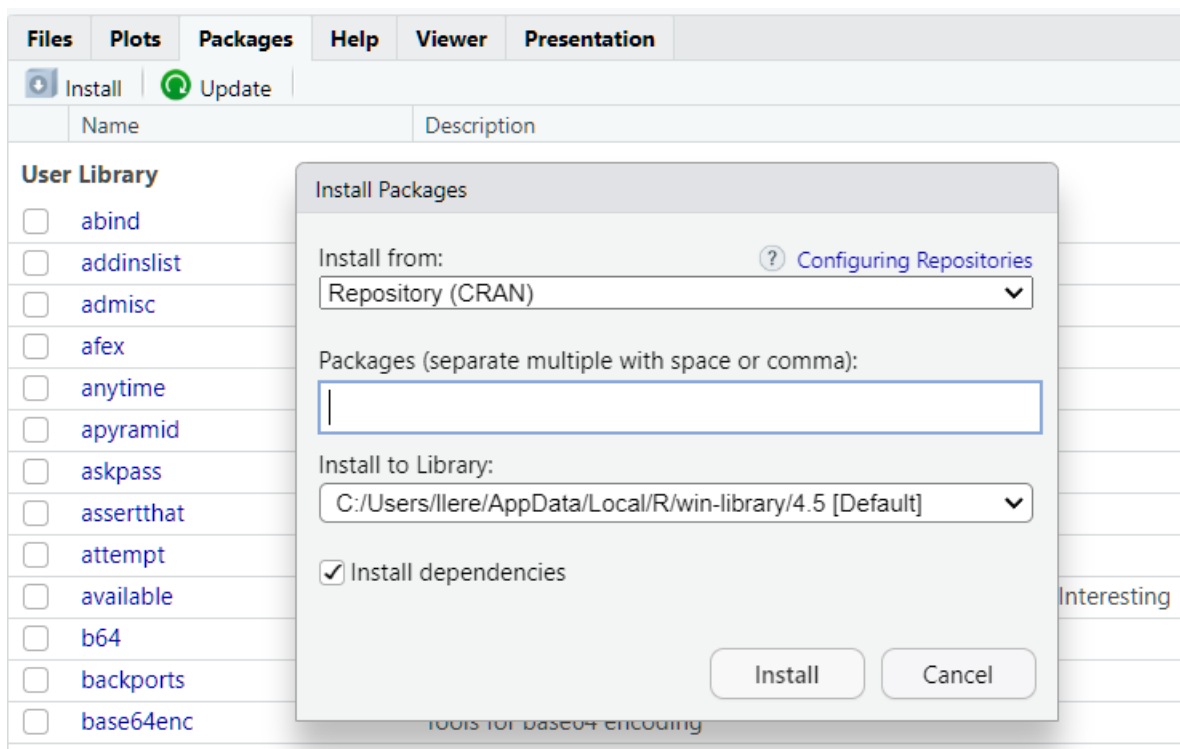


Figura 2.7: Figura 7. Seleccionar “Tool” luego “Global Options” y en ayuda, buscar los atajos del teclado

## 2.3 Instalación de Paquetes (librerías de funciones)

Los paquetes son extensiones para añadir más funcionalidades a R. Es quizás la razón por la cual R es ahora mismo una de las mejores herramientas para trabajar con datos dada la cantidad de paquetes específicos para realizar tareas como hacer tablas por ejemplo. En la literatura, los paquetes se identifican con { }, ejemplo: (Wickham and RStudio 2023){**tidyverse**} y los logos son una figura dentro de un hexágono.

Con respecto a la instalación de paquetes, usualmente estos deben estar publicados en CRAN ([Aquí el enlace](#)) y se instalan desde el panel de paquetes (“Packages”).



## 2.4 Como instalar y cargar un paquete en R

Después de instalar R, Rstudio solo tenemos las funcionalidades que trae **R base**, que nos permiten hacer muchas cosas sin tener que instalar nada más, pero podemos hacer que R sea más completo, ahora vamos a ver que es un paquete y su importancia.

Como R es un lenguaje de programación por lo tanto podemos crear programas o conjunto de funciones y estos programas podemos exportarlos para luego usarlos más adelante.

Estos programas o complementos aumentan la capacidad de R a través de funciones o comandos, por ejemplo, **R base**, no nos permite exportar directamente los resultados o salidas en formato Excel o crear tablas con formato de forma directa. Por lo tanto, los paquetes o subprogramas hacen que R sea más que solo un lenguaje de programación.

Hay paquetes creados por epidemiólogos, para epidemiólogos como el **{epitools}** (Aragon et al. 2020) o **{epikit}** (Spina et al. 2023) que nos permiten con pocos comando o funciones hacer tareas como calcular medidas de asociación (OR, Riesgo relativo) y facilitar los análisis en epidemiología y otros paquetes que hacen las tareas de análisis de datos como son el **{openxlsx}** (Schauberger et al. 2023) para manipular y crear archivos de Excel o el muy famoso y aclamado paquete **tidyverse** (Wickham and RStudio 2023) del cual hay este paquete hay libros escritos para el procesamiento y manejo de datos por mencionar algunos.

Los paquetes nos permiten mejorar sobremanera nuestra productividad en general. Cada vez que iniciamos una sesión con Rstudio, en nuestras rutinas debemos cargar los paquetes que vamos a usar.

Dado que R es una plataforma colaborativa, cuando un paquete es creado pasa por un proceso de validación antes de ser publicados en el repositorio de R. Es por esto que, para su instalación debemos saber si están disponibles. Los paquetes, usualmente, después de ya ser validados, son publicados en el repositorio de R (The Comprehensive R Archive Network o [CRAN](#)) que es la página donde descargamos R. También pueden estar disponibles en otras páginas donde podemos descargarlos e instalarlos manualmente. Después de instalar un paquete solo es necesario verificar si ha sido actualizado, tal como haces con cualquier otro programa.

Usualmente los autores de los paquetes tienen tutoriales o páginas llamadas “Vignettes” o viñetas donde podemos ver las funcionalidades y tutoriales. Por ejemplo, ejemplo ingresa en la barra de búsqueda en tu explorador estas palabras: “R package epitools”, y aparecerán muchos resultados para diferentes tareas. Sigue estos pasos para verificar si un paquete existe buscando directamente desde RStudio:

1. Ir al panel de archivos
2. Hacer clic en la ventana de paquetes o “Packages”
3. Hacer clic en “install”. Aparecerá una ventana donde escribirás el nombre del paquete. Si el paquete existe en CRAN, aparecerá en un listado y lo seleccionarás
4. Hacer clic en “install”.

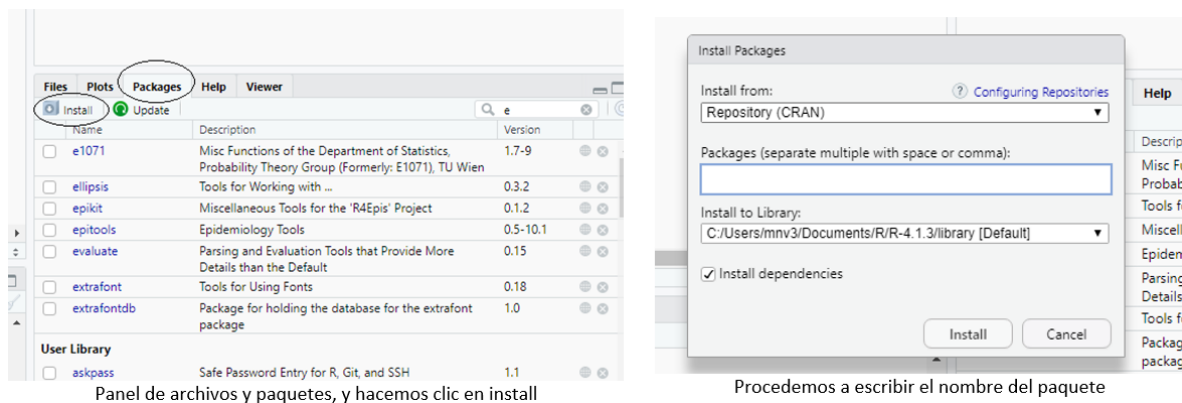


Figura 2.8: instalando un paquete en Rstudio

Si el paquete no está disponible en CRAN pero si en la web y podemos descargarlo y cambiar donde dice “**install from**” a “**package archive file**” y buscar en el disco duro y proceder a instalar (es muy raro que se de este caso y los paquetes que vamos a instalar todos están disponible en CRAN).

La otra forma de instalar los paquetes es directamente desde la consola o en un archivo de rutina usando la función `install.package()` donde escribimos entre comillas el nombre del paquete que queremos instalar.

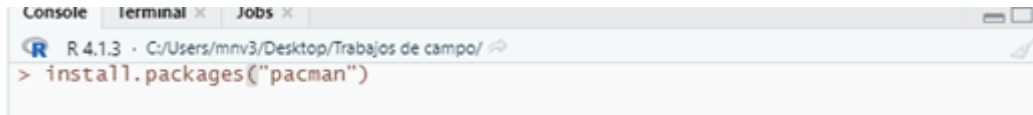


Figura 2.9: Figura 14. En esta imagen vemos en la consola la función para instalar el paquete pacman

Luego de esta introducción sobre qué es un paquete y cómo se instala, vamos a hacer el siguiente ejercicio de instalar varios paquetes que usaremos de forma constante en los ejercicios y tareas contenidos en este manual. Ya sea directamente en la consola (ver imagen anterior), o a través del panel de archivos, vamos a instalar este paquete “**pacman**” (Tyler Rinker) que es un paquete para manejar paquetes y nos ahorrará muchos pasos, por ejemplo, detecta si un paquete necesario está instalado o no, y procede a instalarlo, o ayuda a instalar paquetes desde otras fuentes alternativas a CRAN. Después de instalarlo podemos ver en la consola el mensaje de que se instaló correctamente (*package ‘pacman’ successfully unpacked and MD5 sums checked*).

En la rutina que comenzamos hace un momento atrás vamos a hacer la siguiente tarea:

- Escribe o copia y pega el siguiente comando (recuerda, para ejecutar un comando o varios selecciona estos y presiona Ctrl+Enter o clic en “Run”):

```
pacman::p_load("tidyverse", "janitor", "gtsummary", "epikit", "here",
```

```
"epitool"
```

Espera un momento si es la primera vez que se ejecuta para que así se instalen los paquetes que están en el comando.. Antes de continuar, vamos a explicar el comando anterior:

En Rstudio, en el editor de rutinas o en la misma consola, cuando queremos ver las funciones o comandos disponibles en un paquete podemos escribir el nombre del paquete seguido de dos puntos nos aparecerá una ventana con un listado de dichas funciones, por ejemplo, este último comando usamos el paquete **pacman** y dentro de este paquete usamos la función de `p_load` (para cargar paquetes, que también los instala si no están ya instalados)

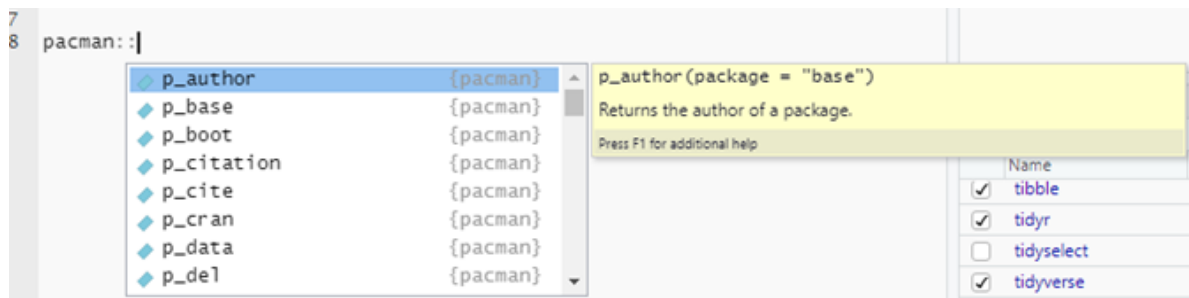


Figura 2.10: Esta es una de las funcionalidades de Rstudio que nos facilitan el trabajo , luego de escribir el nombre del paquete y dos veces dos puntos nos aparece el listado de funciones y también una ventana amarilla para ayuda de la función presionando F1 para ver en el panel de archivos, la ventana de ayuda y así ver los ejemplos de cómo usar dicha función.

Luego de escribir el nombre del paquete y dos veces dos puntos, nos aparece el listado de funciones y también una ventana amarilla para ayuda de la función, que también aparece presionando F1 o haciendo una búsqueda en el panel de archivos en la ventana de ayuda, donde también se pueden ver los ejemplos de cómo usar las funciones.

Los paquetes que hemos instalado hasta este momento no los vamos a detallar por ahora, sino que, en la medida que vayamos haciendo los ejercicios, vamos a ir explicando para qué sirven a través de las funciones que traen cada uno. Es oportuno señalar que, dentro de los paquetes disponibles en R, existen paquetes útiles para facilitar el proceso de importar y exportar funciones específicas para las tareas del epidemiólogo, para hacer reportes y para facilitar el proceso de la gestión de los datos.

Para ver que paquetes tenemos cargados podemos escribir en la consola `search()`

```
search()
```

```
[1] ".GlobalEnv"      "package:rio"      "package:readxl"
[4] "package:openxlsx" "package:epitools" "package:here"
[7] "package:epikit"   "package:gtsummary" "package:janitor"
[10] "package:lubridate" "package:forcats"   "package:stringr"
[13] "package:dplyr"     "package:purrr"     "package:readr"
[16] "package:tidyr"     "package:tibble"    "package:ggplot2"
[19] "package:tidyverse" "package:stats"     "package:graphics"
[22] "package:grDevices" "package:utils"     "package:datasets"
[25] "package:methods"   "Autoloads"         "package:base"
```

Los dos capítulos siguientes son muy importantes. Debes familiarizarte con ellos porque son los fundamentos para poder trabajar en R, veremos ejemplos de las sintaxis de las expresiones,

los diferentes tipos de objetos de datos, y a medida que vayamos practicando se irá haciendo más fácil entender el código, las funciones y las operaciones que hacemos. Te recomendamos que, aparte de este manual, abundes más sobre los temas que verás a continuación para que agilices tu proceso de aprendizaje con R.



## 3 Comenzar a trabajar con R y la interfaz de Rstudio

Ya luego de tener todo lo necesario instalado, ahora vamos a realizar los siguientes pasos con la finalidad de ir aprendiendo a organizar los trabajos que estaremos haciendo, como son los productos o asignaciones que se deben hacer durante la capacitación, como son los análisis de vigilancia, análisis de brotes, evaluación de sistema de vigilancia entre otros.

Una gran recomendación es tener lo más organizado posible cada elemento hecho en R o cualquier documento asociado a los análisis o trabajos que vas a realizar; esto porque en nuestro caso, cuando comenzamos a trabajar con RStudio guardábamos las rutinas y las salidas en diferentes lugares, como principiantes al fin, no teníamos una estructura, y luego cuando necesitábamos re-usar una rutina, buscar un documento de salida era muy difícil de encontrar y perdíamos tiempo en esa búsqueda, a veces nos tomaba más que lo que se toma ejecutar una rutina. Luego aprendimos sobre lo que vamos a ver a continuación en cuanto a trabajar bajo proyectos en RStudio.

### 3.1 Creación de un proyecto

El primer paso para comenzar es decidir dónde estarán los archivos relacionados al o a los trabajos que se van a hacer, es decir, debes definir si almacenarás tu trabajo en un disco dentro de la PC o en la “Nube” (ejemplo, OneDrive, Google drive) para que así siempre facilites tu trabajo, especialmente cuando necesites hacer cambios o actualizar las bases de datos.

Esto es sumamente importante dado que nos ayudará a mantenernos organizados. Todos los archivos como las bases de datos, las referencias, las rutinas, deben estar almacenados en un lugar donde te sea fácil encontrarlos y también para la ejecución o desarrollo del proyecto o proyectos en que estés trabajando.

Para el segundo paso vamos a entrar en RStudio y vamos a crear un **archivo de proyecto** de la siguiente forma:

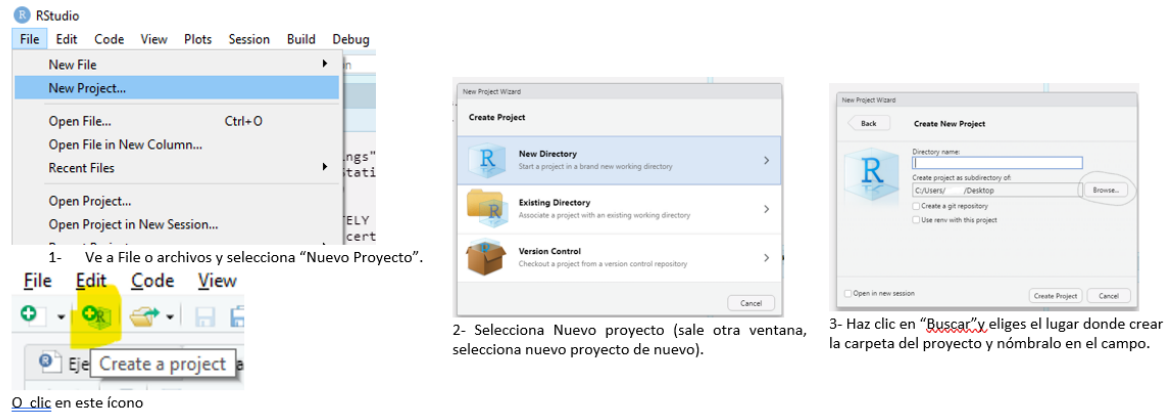


Figura 3.1: Figura 10. Creando un proyecto

1. Crea el proyecto llamándolo "mi\_primer\_proyecto" siguiendo los pasos de la Figura 10.
2. Guarda el proyecto nuevo en el "escritorio" o en "Documentos" si usas Windows.
3. Para crear las carpetas, en la ventana inferior derecha o panel para archivos, ve a "Archivos" o "File", luego haz clic en "new folder" para crear las siguientes carpetas:
  - a. "Base de datos" para almacenar tus datos en cualquier formato, como datos en Excel, por ejemplo.
  - b. "Rutinas" para salvar las rutinas que irás creando.
  - c. "Referencias" vas a poner aquí los documentos de soporte como referencias, artículos, etc.
  - d. "Salidas", donde vamos a salvar los documentos generados a partir de las rutinas, en esta última podemos tener una subcarpeta que se llame "imágenes" para las salidas que son imágenes, como gráficos o tablas salvadas en formato de imagen.

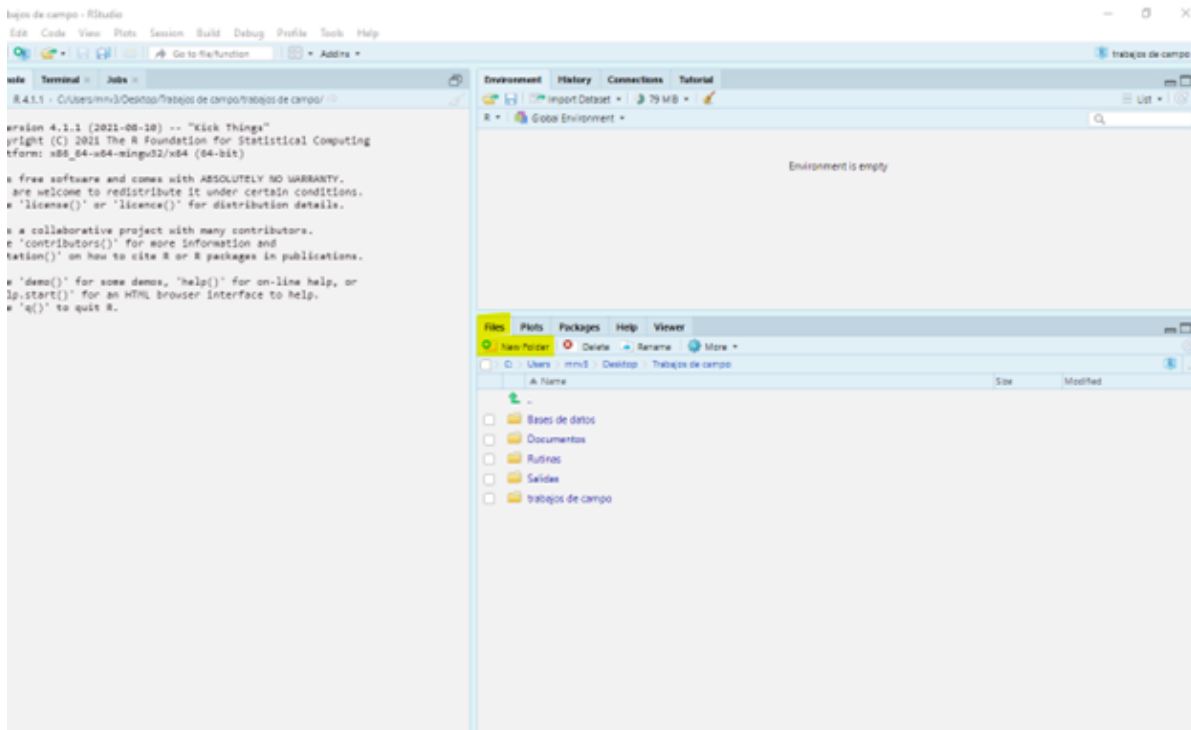


Figura 3.2: Al finalizar, tu panel de archivos se tendría que ver como se muestra en la Figura 11

Si has hecho todos estos pasos y tu pantalla se parece a la imagen anterior es porque has creado un proyecto. Puede verificar el nombre del proyecto en la esquina superior derecha de la interfaz de RStudio.

Aparte de mantener una organización para mantener todo en orden, la mayor ventaja de usar proyectos es que a RStudio se le facilita encontrar dónde están los archivos que serán usados, por ejemplo, si ubicas las bases de datos correctamente, será más fácil para ti importar los datos a R y también exportar archivos a otras herramientas de análisis.

### *¿Por qué en la imagen anterior falta un panel*

En la imagen anterior solo vemos 3 ventanas porque no hemos abierto o creado una rutina, solo tenemos la consola, el panel del ambiente de trabajo y el panel de archivos.

## 3.2 Archivos de rutina (script) en R

Siguiendo la secuencia de pasos para comenzar a trabajar con R en Rstudio, vamos a ver uno de los elementos más importantes que son las rutinas y que la usaremos constantemente.

Hasta este punto, hemos mencionado varias veces la palabra “rutina”. Una rutina no es más que un “documento” donde escribimos comandos (funciones, cálculos, etc.), que puede ser ejecutado las veces que sea necesario (como por ejemplo procesar y generar un reporte semanal). Realmente crear una rutina es un procedimiento muy sencillo porque se trata simplemente de escribir en un editor de texto las funciones que generan los resultados que esperamos.

Dado que podemos salvar las rutinas como archivos, podemos compartirlas y guardarlas para luego usarlas como referencia.

Para crear una nueva rutina en RStudio vamos a “*file*” -> “*new file*” -> “*R Script*” (en inglés rutina es igual a *Script* por lo que estaremos utilizando ambos conceptos de manera indistinta a lo largo del curso).

Después de aceptar ahora tenemos el panel de rutinas habilitado (igual que un editor de texto) tal como se ilustra en la Figura 12.

Cuando salves la rutina nueva por primera vez, te pedirá dónde guardar el archivo, entonces procede a guardarlo en la carpeta de “Rutinas” que previamente creaste en el Panel de archivos y ya podrás comenzar a trabajar. Es importante anotar que RStudio no salva el avance de tu rutina automáticamente, sino que siempre debes guardar cada cierto tiempo, ya sea a través del menú o utilizando el atajo **Ctrl+S**.

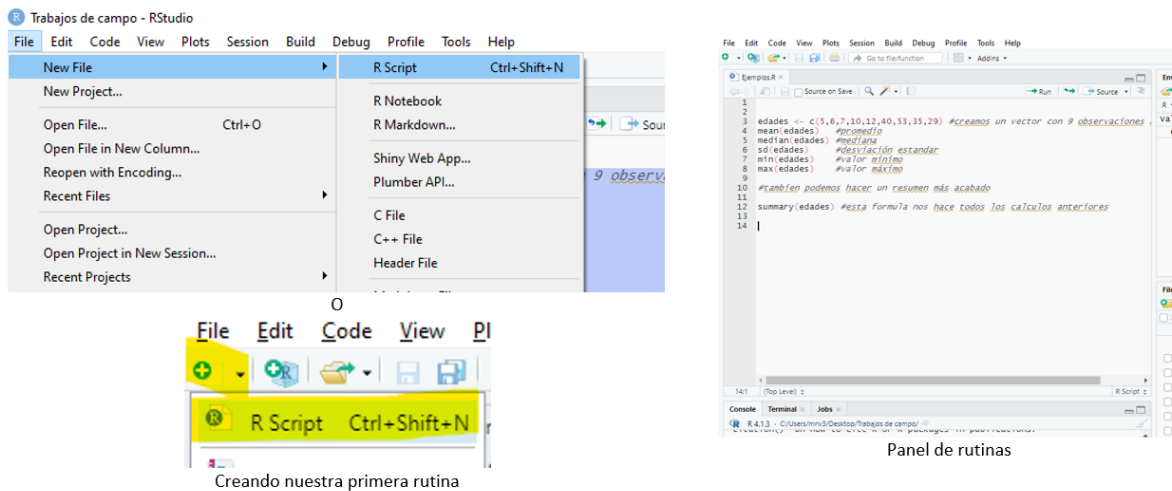


Figura 3.3: Figura 12. Pasos para comenzar un archivo de rutinas o (Ctrl+Shift+N)

Si has llegado hasta aquí, entonces ya tienes una gran parte del camino recorrido, es decir, ya tienes el programa instalado y disponible, un proyecto creado y tu primera rutina guardada. Si desde ya comenzáramos a trabajar, podemos escribir comandos directamente en la consola o en el documento de la rutina para hacer operaciones como aplicar a nuestros datos los

**estadísticos de tendencia central**, para que vayas viendo y familiarizándote con R. Vamos a hacer la siguiente prueba paso a paso:

1. Escribe el texto debajo en el documento de la rutina, o si estas desde tu PC, copia y pega todo este texto, selecciónalo (como en cualquier editor de texto o Ctrl+A) y presiona **Ctrl+Enter** o haz clic en “**Run**”.

```
edades <- c(5,6,7,10,12,40,53,35,29) #creamos un vector con 9 observaciones numericas
mean(edades)    #promedio
median(edades)  #mediana
sd(edades)      #desviación estandar
min(edades)     #valor minimo
max(edades)     #valor máximo
#tambien podemos hacer un resumen más acabado, con menos comandos
summary(edades) #esta función nos hace todos los calculos anteriores}
```

2. Luego de ejecutar los comandos, podemos ver el resultado en la consola. Como puedes observar, está escrito lo mismo que la rutina pero debajo de cada comando hay un resultado, es decir, vas a ver el resultado de calcular el promedio a través de la función mean() cuyo resultado es 21.8, y así sucesivamente. La consola debería verse algo similar a esto:

```
edades <- c(5,6,7,10,12,40,53,35,29) #creamos un vector con 9 observaciones numericas
mean(edades)    #promedio
```

```
[1] 21.88889
```

```
median(edades) #mediana
```

```
[1] 12
```

```
sd(edades)      #desviación estandar
```

```
[1] 17.73728
```

```
min(edades)     #valor minimo
```

```
[1] 5
```

```
max(edades)      #valor máximo
```

```
[1] 53
```

```
#tambien podemos hacer un resumen más acabado, con menos comandos  
summary(edades) #esta función nos hace todos los calculos anteriores
```

| Min. | 1st Qu. | Median | Mean  | 3rd Qu. | Max.  |
|------|---------|--------|-------|---------|-------|
| 5.00 | 7.00    | 12.00  | 21.89 | 35.00   | 53.00 |

Una de las ventajas que tiene RStudio de trabajar con los documentos de rutina es que puedes tener varias rutinas abiertas a la misma vez y navegar entre ellas. Esto te puede resultar útil para utilizar una rutina dentro de otra, es decir, como las rutinas se pueden guardar en la PC, vas a darte cuenta de que hay muchos procedimientos que se repiten y que solo necesitas modificarlos levemente, en estos casos, mientras estás haciendo una rutina puedes usar otra como referencia para tomar códigos que ya has usado antes o que te han compartido. *¿Cuál es la ventaja de esto? ¿que no es obligatorio aprenderse todas las funciones o comandos de R!*

Una muy buena práctica es comentar todo lo que haces para luego saber qué hiciste en una secuencia de códigos. Para esto, solo tienes que comenzar una línea con el carácter “#” o signo de número, y todo el texto después de este carácter cambia de color y se pone en formato itálico, tornando el texto de color verde de forma predeterminada, aunque el color verde pudiera variar dependiendo de la configuración de colores que hayas elegido, tal como te explicamos en la sección 2.2. Cuando estemos en los pasos de hacer tablas y gráficos, vamos a usar mucho los comentarios.

El editor de texto de rutina en RStudio también nos permite ver si hay errores en el código, otra de las tantas funcionalidades que nos ofrece el programa para hacer más fácil la escritura de códigos.

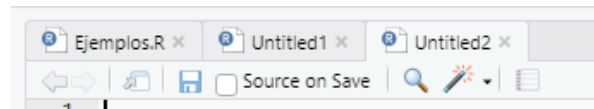


Figura 3.4: Varias rutinas abiertas (con **Ctrl+shift+tab** podemos movernos entre las rutinas sin usar el ratón)

## 4 Operadores en R

Antes de continuar con los paquetes y funciones, debemos conocer algunos conceptos básicos para poder trabajar con R. Se trata de los operadores, los de uso común en matemáticas (como suma, resta, multiplicación y división), los de asignación, los booleanos, entre otros. Es bueno que sepas, si acaso no lo sabes, que todos los lenguajes de programación usan operadores.

Los operadores son los que nos permiten escribir las expresiones que usaremos en nuestras rutinas y las funciones. En R, hay operadores que se usan constantemente para crear nuestro código.

Desde que abres R, puedes utilizarlo como una calculadora (muy avanzada, por cierto), es decir, si escribes en la consola  $2+2$  y presionas **Enter** tendrás un resultado, pero si queremos correr varias veces esta misma operación podemos guardarla en un **objeto** y en vez de escribir de nuevo la operación (el  $2+2$ ) podemos simplemente llamar este **objeto** con solo escribirlo y vamos a obtener el mismo resultado. manejar los objetos nos agiliza mucho el trabajo.

Dentro de las funciones podemos ver si un valor o un objeto está presente, así como también reasignar un valor o hacer operaciones matemáticas. A continuación, te mostramos los operadores que utilizaremos con más frecuencia.

### 4.0.1 Resumen de los operadores más usados para la escritura de las expresiones en R

| Tipo       | Operador            | Descripción   | Ejemplo   |
|------------|---------------------|---|---|
| Asignación | <code>&lt;-</code>  | Para asignar un valor a un objeto (el signo de <code>=</code> es similar) se compone de “menor que” seguido de una raya o “dash”  | <code>x &lt;- 2+2</code> con esto creamos (asignamos) la operación $2+2$ al objeto <code>x</code><br>si volvemos a usar <code>x</code> como objeto y le asignamos otro valor, este se sobrescribirá |
| Asignación | <code>&lt;-"</code> | Las comillas son para declarar texto, si escribes cualquier caracter entre comillas será reconocida como una cadena de caracteres | <code>z &lt;- "Me gusta R"</code> asignamos al objeto <code>z</code> el texto   |
| Evaluación | <code>==</code>     | igual, igual significa Igual, para evaluar si un objeto es igual a otro o una variable es igual a un valor                        | <code>x == 4</code> (la respuesta es <code>TRUE</code> )  |

| Tipo Operador | Descripción   | Ejemplo   |
|---------------|---|---|
| Evaluación    | Significa no es igual, para evaluar si un objeto es diferente a otro o una variable es diferente a un valor.  | <code>x != 4</code> (la respuesta es FALSE)   |
| Evaluación    | Signos para evaluar mayor que, menor que, mayor o igual que y menor o igual que, aparte de los números, estos operadores funcionan con texto, (b es menor que c por ejemplo)  | <code>x &lt; 5</code> (la respuesta es TRUE), <code>x &gt; 5</code> (la respuesta es FALSE)<br><code>x &gt;= 3</code> (la respuesta es TRUE)<br><code>x &lt;= 5</code> (la respuesta es FALSE)  |
| Evaluación    | Identifica si un elemento (como un número, por ejemplo) está dentro de un objeto (si queremos decir “no esta incluido en” solo agregamos signo de exclamación !)  | <code>2 %in% x</code> (la respuesta es TRUE)<br><code>2 !%in% x</code> (la respuesta es FALSE)  |
| Acceso        | Signo de dinero sirve para acceder a las variables de un dataframe o una lista  | <code>mtcars</code> (un dataframe interno de R) tiene 11 columnas, para seleccionar o aplicar una función a una de estas (como <b>mean</b> para promedio) escribes el nombre del dataframe seguido del signo de dinero y el nombre de la variable así : <b>mean(mtcars\$cyl)</b> para retornar el promedio o 6.1875 |
| Acceso        | Los corchetes son usados con los vectores, matrices y dataframe para acceder a un valor en base de una posición (para las matrices y dataframe) donde el primer valor es el numero de fila y el segundo valor es el numero de columna. Para los vectores solo se usa un valor para especificar la posición. | usando <b>mtcars</b> de nuevo, si quiero saber que valor está en la fila 20 de la columna 3 puedes escribir este código <b>mtcars[20,3]</b> y retorna 71.1 que es el valor de la variable <b>disp</b> de la fila del carro toyota corolla.  |
| Booleano      | Y o AND   | <code>y = c(1,3,4,9,11,10)</code> #un vector con 6 elementos<br><code>3 &amp; 4 %in% y</code> (la respuesta es TRUE, porque están ambos, si no es así, es FALSE )   |
| Booleano      | O o OR  | <code>3   30 %in% y</code> (la respuesta es TRUE, porque está al menos 1 de los elementos dentro del objeto o vector <b>y</b> , si ambos elementos faltaran, la respuesta seria FALSE)  |



| TipoOperador | Descripción   | Ejemplo   |
|--------------|---|---|
| Aritméticos  | Signos matemáticos para suma, resta, multiplicación, división y exponente. El igual es usado más para asignación  | 2*2 retorna 4<br>2+2 retorna 4<br>4/2 retorna 2<br>x = 2+2 (igual como asignación, otro ejemplo x=log(4), donde log() es una función que la usamos para asignar al objeto x el logaritmo de 4)                                  |
| Otros)       | Los paréntesis son símbolos que se usan mucho en operaciones y funciones para determinar que ocurra algo y en el orden, siempre desde los más internos hacia los externos   | mean(c(3,4,5,6,2,5)) el resultado es 4.16, si te fijas primero tenemos un set de paréntesis que crea un objeto, luego el otro set de paréntesis que indica calcular el promedio de este grupo usando la función <i>mean()</i> , |
| Otros%>%     | Es un operador para indicar una secuencia de comandos o expresiones que se realizan a partir de un objeto en cascada, significa “entonces o luego” (vamos a ver más de su uso cuando veamos transformación de datos) se puede escribir con Ctrl+M y hay que tener instalado el paquete <b>tidyverse</b> | c(3,4,5,6,2,5)%>%<br>mean() nos da el mismo resultado que mean(c(3,4,5,6,2,5)), básicamente una expresión como esta nos dice “crea un objeto de 5 numeros, <b>Entonces</b> (el pipe) obtén el promedio”                         |
| otros,       | La coma en R se usa para asignar un espacio de un elemento o un parámetro, por ejemplo, el primer elemento luego una coma, segundo elemento y luego una coma  | y <- c(“a”, “b”, “c”)<br>ggplot(data=df, aes(x=var1, y=var2, fill=cat1))  |

En la medida que vayas practicando los operadores se irán haciendo familiares y más fácil de entender. Siempre hay que tomar en cuenta que un operador mal usado dará error cuando ejecutamos un comando o código, verás que te ocurrirá a menudo dejar de escribir el último paréntesis, o cerrar comillas.

## 5 Objetos en R

Otro elemento de suma importancia a conocer cuando se trabaja con R, así como vimos la importancia de las rutinas, son los objetos y sus diferentes tipos, dado que es con esto que trabajamos. Son básicamente los contenedores de los datos, que R los almacena en la memoria del computador, es decir, a partir de estos es que haremos nuestras salidas como tablas, gráficos, reportes a través de la transformación (declarar, nombrar, filtrar reemplazar), entre otras. Básicamente los objetos son el fundamento de todo.

En R, hay 5 tipos de objetos de datos y se dividen en dos grupos de objetos: los que son de un solo tipo de dato (atómicos) o los que tienen varios tipos de datos (recursivos). Empecemos con los atómicos.

### 5.1 Vectores

Dentro de los objetos atómicos tenemos **los vectores**, los cuales no tienen dimensión, es decir, no están compuestos por filas o columnas como un dataframe, y pueden ser de clase numérica (entero y decimal), integer (números enteros solamente), caracter o texto, lógico (TRUE, FALSE) y factor (jerarquía en los valores). Un ejemplo de creación de un vector es el siguiente:

| Código para crear un vector numerico con 5 elementos |                        |                               |
|--|------------------------|-------------------------------|
| <b>x</b>   | <b>&lt;-</b>           | <b>c( 1, 2, 3, 4, 5)</b>      |
| Nombre del objeto                                    | Operador de asignación | función de combinar de R base |

| Código para crear un vector de caracteres de 5 elementos |                        |                                    |
|--|------------------------|------------------------------------|
| <b>y</b>   | <b>&lt;-</b>           | <b>c( "a", "b", "c", "d", "e")</b> |
| Nombre del objeto  | Operador de asignación | función de combinar de R base      |

Para los vectores que son de *caracteres o texto*, si al crearlos les ponemos números sin comillas, cuando los llamemos serán reconocidos como texto, por lo tanto, no se pueden hacer operaciones matemáticas con estos vectores aunque sean números.

Si escribes “x” (o el nombre que hayas usado para crear el vector) y presionas **Enter**, en la consola observarás los elementos contenidos en este vector.

Una de las funcionalidades más interesantes de R es que, en el caso de vectores numéricos, si haces una operación matemática como  $5 + x$  (nombre del vector), la operación se hará en todos los elementos del vector, es decir, el 5 se sumará con cada uno de los números. Si haces esto con un vector de texto o carácter, en la consola observarás un error que es una operación entre un argumento no numérico contra otro numérico.

A medida que avancemos con el desarrollo de los productos vamos a ver que los vectores los usamos de forma constante; un ejemplo común es crear un vector con los nombres de las provincias o municipios de interés para luego poder hacer un filtro invocando dicho vector. También se puede hacer combinando varios tipos de vectores con las mismas dimensiones, es decir, el mismo número de elementos podemos “unirlos” y formar un nuevo objeto como un **dataframe** o una lista.

## 5.2 Matrices

Otro tipo de objeto atómico o que solo admite un tipo de valor es **la matriz** que es básicamente un vector pero con dos dimensiones (columnas y filas) se crean usando la función **matrix()** y se le debe agregar el parámetro del total de filas así como el total de columnas por ejemplo de la siguiente forma:

| Código para crear un objeto tipo matriz de 5 filas y 2 columnas |                         |                               |   |                              |                                |
|---|-------------------------|-------------------------------|---|------------------------------|--------------------------------|
| <b>m</b>  | <b>&lt;-</b>            | <b>matrix(</b>                | <b>c(1,2,3,4,5,6,7,8,9,10),</b>                 | <b>nrow=5,</b>               | <b>ncol=2)</b>                 |
| Nombre del objeto   | Operador. de asignación | Función de creación de matrix | Vector u objeto que será transformado en matrix | Parametro de ttotal de filas | Parametro de total de columnas |

No entraremos detalles ahora con los objetos de tipo matriz porque estos objetos tienen sus usos bien específicos y no vamos a usarlos de forma constante como lo haremos con los vectores.

## 5.3 Dataframes (conjunto de datos)

Ahora vamos a ver uno de los objetos más importantes que vamos a usar, los **dataframes**, que es un objeto de tipo recursivo o que admite varios tipos de datos (texto, numérico, lógico) y es idéntico a una tabla, es decir, tiene dos dimensiones, columnas y filas. Las columnas se pueden ver como si fueran variables y las filas son los valores contenidos en esta variable.

Para los trabajos que vamos a realizar, este será el tipo de objeto con el que más nos sentiremos familiarizados, dado que lo más parecido a un dataframe es una base de datos, ya sea en Excel u otro tipo de programa.

Una de las ventajas que tiene Rstudio es que nos ayuda a ver fácilmente la estructura del objeto dataframe, cuantas variables o columnas y cuantas filas u observaciones contiene para visualizar el contenido.

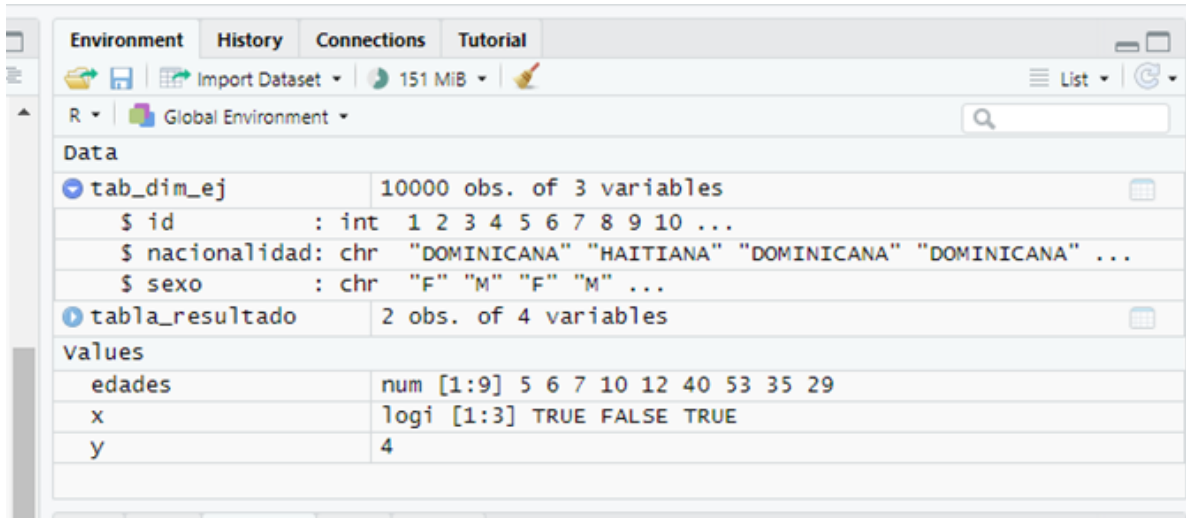


Figura 5.1: Panel de ambiente de trabajo en Rstudio donde podemos ver los objetos de datos disponibles, en esta imagen podemos ver dos dataframes (tab\_dim\_ej y tabla\_resultado) y debajo, en la categoría de values, 3 vectores, (edades, x y “y”) . Rstudio nos permite diferenciar rápidamente también ver los atributos de cada elemento. Para saber más sobre los tipos de objetos podemos usar dos funciones str() y class(), la primera para ver la estructura y la segunda para ver que clase de objeto es.

Para comenzar a usar los **dataframes** tenemos dos formas de hacerlo. Primero, creándolo directamente escribiendo los datos que usaremos, combinando varios vectores para crear una tabla como en el siguiente ejemplo:

```
var1 <- c("maria", "pedro", "juan", "jose") #creamos un vector con nombres por ejemplo
var2 <- c(25, 40, 30, 35) #creamos otro vector con las edades
var3 <- c("F", "M", "M", "M") #creamos otro vector con el sexo
var4 <- c("LR", "SD", "SC", "PP") # creamos otro vector con la procedencia
tabla <- data.frame(nombre=var1, edad=var2, sexo=var3, prov=var4)
```

tabla

|   | nombre | edad | sexo | prov |
|---|--------|------|------|------|
| 1 | maria  | 25   | F    | LR   |
| 2 | pedro  | 40   | M    | SD   |

|   |      |    |   |    |
|---|------|----|---|----|
| 3 | juan | 30 | M | SC |
| 4 | jose | 35 | M | PP |

| Código para crear un objeto tipo dataframe |                        |   |   |  |  |
|--|------------------------|---|---|--|--|
| Tabla                                      | <-                     | data.frame( Nombre = vector,,,,, stringAsFactor=FALSE check.names=TRUE) |   |  |  |
| Nombre del objeto                          | Operador de asignación | Función para crear un dataframe   | Parametro de nombre de la columna y a que vector corresponde, varios dependiendo del total de columnas, | Parametro para convertir los vectores tipo texto en factores (categóricos) | Parametro para evitar que se dupliquen los nombres de las columnas |

Figura 5.2: Anatomía de la función `data.frame()` para crear un dataframe. en R las funciones vienen con valores en los parámetros de forma predeterminada, incluso no hay que escribirlos. Un ejemplo es en esta función de `data.frame()` el parámetro `check.names=TRUE` viene predeterminado y por ende no hay que escribirlo.

Entonces si ejecutamos las líneas anteriores vamos a tener como resultado un dataframe nuevo con 4 variables y 4 observaciones. ¡Ojo! los vectores deben de tener la misma cantidad de elementos, de lo contrario, veremos el siguiente error: *“Error in data.frame(nombre = var1, edad = var2, sexo = var3, prov = var4) : arguments imply differing number of rows: 4, 3”*, esto significa que uno de los vectores no tiene la misma cantidad de elementos.

Para corregir esto, si el valor es desconocido en el vector que nos falta un elemento, solo agregamos un elemento nuevo llamado **NA**, sin comillas, que es una forma de decirle a R que el valor es desconocido, posteriormente, ejecutamos de nuevo las líneas donde hicimos el cambio para actualizar ¡y listo!

Este abordaje puede ser útil para tablas pequeñas, pero cuando tenemos que trabajar con bases de datos que tienen cientos o miles de observaciones, entonces creamos un dataframe importando los datos desde diferentes fuentes, como archivos de Excel, desde archivos de texto, incluso archivos nativos de otros programas estadísticos como SPSS o STATA.

En Rstudio podemos, sin necesidad de escribir una expresión o comando, usar el botón de **“import Dataset”** en el panel de ambiente de trabajo, y al hacer clic, se despliega un menú para diferentes tipos de formatos de bases de datos como Excel, csv, txt, SPSS, SAS o Stata.

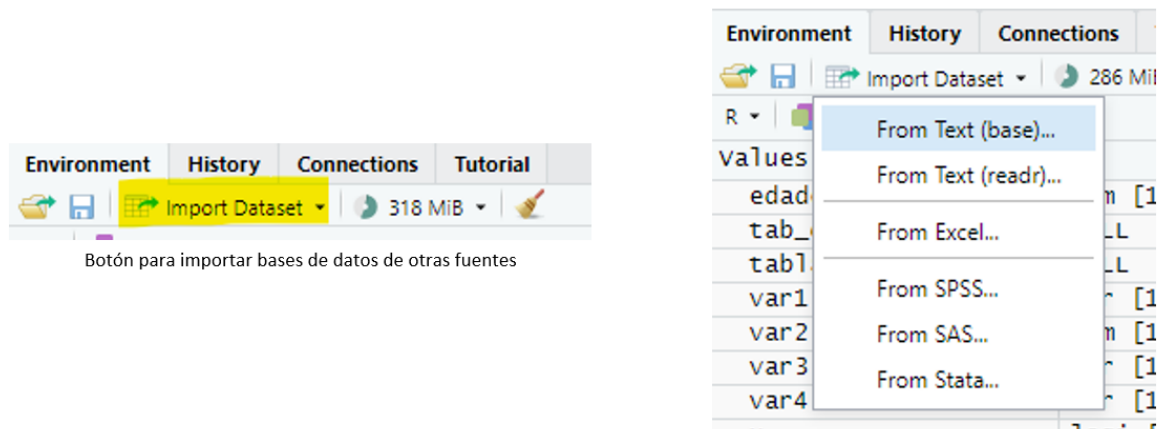


Figura 5.3: Botón para importar bases de datos de otras fuentes

Después de hacer clic en uno de los formatos, (ejemplo Excel) se abre una ventana para seleccionar el archivo de la base de datos, que variables o columnas vamos a incluir además de otros parámetros.

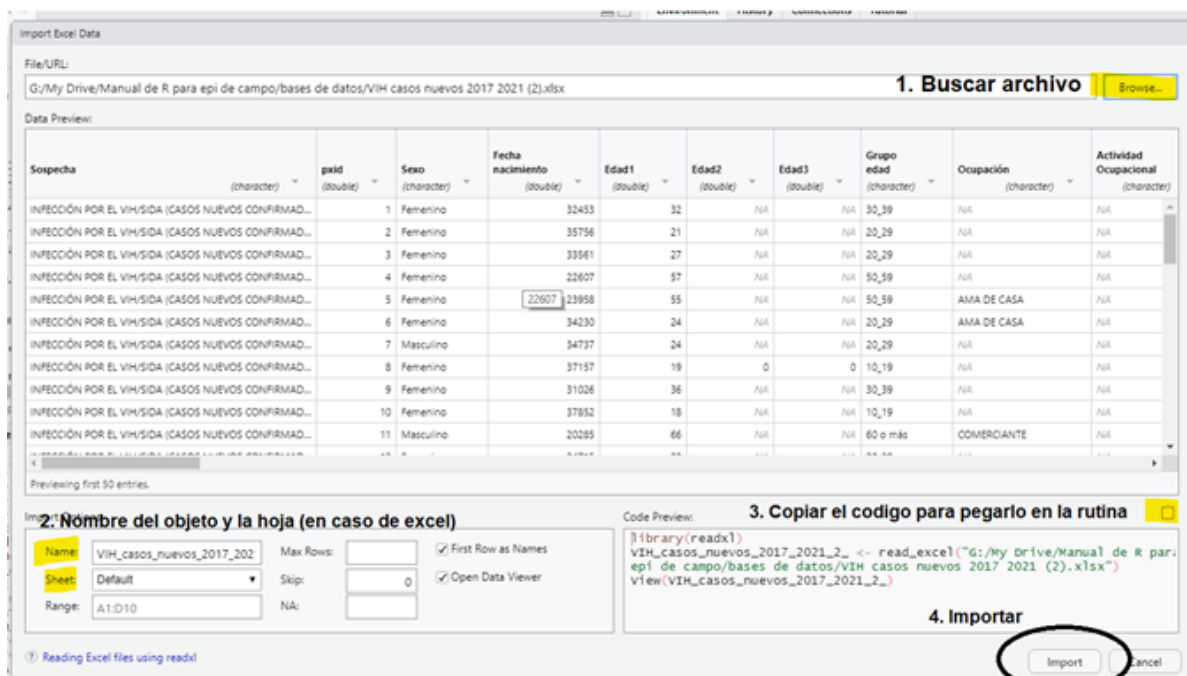


Figura 5.4: Pasos para importar un dataframe usando Rstudio

De forma predeterminada, en el nombre del objeto o dataframe estará el nombre del archivo

de la base de datos, y puedes cambiarlo a un nombre más amigable (ejemplo bd o base). Si estás usando un archivo de Excel que contiene varias hojas, puedes especificar cual hoja usarás en “**sheet**”, incluso, puedes elegir el rango de filas que usarás en caso de que no quieras seleccionar la hoja completa. De igual manera, puedes poner desde cuál fila se leerá el archivo, esto es importante si la base de datos en Excel tiene títulos y los datos se comienzan a partir de la 3ra o 4ta fila, en este caso, puedes en “**skip**” omitir la fila o las filas que no son necesarias.

Por último, si este proceso lo vas a usar repetidamente, debes copiar el código que está en el campo inferior derecho de la ventana y pegarlo en el panel de editor de rutinas, para que lo tengas en tu rutina, y desde ahí, puedes modificarlo de ser necesario, pues realmente este es un aditamento visual dado que Rstudio está generando un código para que no tengas que escribirlo; básicamente, está cargando el paquete **readxl** y usando la función de importar archivos de Excel llamada **read\_excel()**.

Otra forma para cargar bases de datos muy parecida a la que vimos anteriormente es usando el paquete **rio** (**input/output**) y el paquete **here** en combinación con la función **import()** y la función **here()**, que nos ahorra escribir toda la ruta del archivo de la base de datos cuando estamos en un proyecto en Rstudio. A continuación, un ejemplo:

La base de datos que vamos a importar está ubicada en la carpeta de Base de datos, solo necesitamos saber el nombre del archivo que vamos a usar, en este caso, tiene el nombre de “base de datos ejemplo.xlsx”:

```
pacman::p_load(rio) #Para cargar el paquete {rio}

mi_base_ejemplo <- import(here("Bases de datos", "base de datos ejemplo.xlsx"), setclass="tbl")
```

Para ver la base de datos utilizamos la función **View()**, con ella invocamos el visor de datos, o simplemente utilizamos la función **head()** para ver las primeras observaciones del dataframe en la consola. Como buena práctica, después de importar la base de datos en un dataframe, es ver su estructura para determinar si las columnas se importaron correctamente, esto podemos hacerlo viendo en el panel de ambiente de trabajo desplegando la flecha a la izquierda del nombre del dataframe o escribiendo en la consola la función **str()** donde vamos a ver el nombre de la variable, de qué clase es (*chr* si es texto, *num* si es numérico, *logi* si es lógico (true/false), *int* si es interger, etc).

Cuando sabemos de antemano que una variable es de tipo fecha, pero aparece como número luego de importar la base de datos, podemos transformarla a su formato original de fecha. Más adelante veremos cómo se hace.

En la siguiente tabla se resumen las funciones que nos servirán para revisar un dataframe. Al usarlas en la rutina, solo debemos poner el nombre del dataframe dentro de los paréntesis.

Si vemos por ejemplo que una variable que sabemos de antemano que es una fecha está como tipo numérica, luego podemos transformarla a su tipo original. Más adelante veremos cómo se hace.

Tabla 5.1: **Funciones que nos sirven para revisar un dataframe (solo debemos poner el nombre del dataframe dentro de los paréntesis)**

| Función              | Efecto  |
|----------------------|---|
| head()               | Muestra las 6 primeras filas  |
| tail()               | Muestra las ultimas 6 filas   |
| dim()                | Nos muestra las dimensiones del dataframe, total de observaciones y total de columnas |
| nrow()               | Nos muestra el total de filas u observaciones   |
| ncol()               | Nos muestra el total de columnas  |
| str()                | Nos hace un resumen de la estructura del dataframe, columna por columna               |
| names() o colnames() | Nos muestra los nombres de las columnas   |

Los resultados que arrojan estas funciones podemos guardarlas en un objeto, para fines del ejemplo, lo denominaremos “vector”. Si queremos comparar el total de columnas de un dataframe vs otro (por ejemplo, cuando queremos comparar un corte de la base de datos de una fecha vs otro corte) podemos crear el vector `col_dataframe_1 <- ncol(dataframe1)` y el vector `col_dataframe_2 <- ncol(dataframe2)` y luego comparar: `col_dataframe1 == col_dataframe2`, si tenemos el resultado TRUE, pues ambos dataframes tienen la misma cantidad de columnas, de lo contrario, uno de los dataframes tiene columnas de menos o de más.

Por igual, si hacemos el mismo ejercicio para ver cuántas filas ha crecido una base de datos podemos usar la función `nrow()` y restar el vector del `dataframe2 - dataframe1` para obtener la diferencia de filas.

```
filas_bd_1 <- nrow(dataframe1)
filas_bd_2 <- nrow(dataframe2)

diferencias_filas <- filas_bd_2-filas_bd_1
```

Esto es un ejemplo muy sencillo pero muchos procedimientos de transformación de datos usamos estas funciones.

## 5.4 Listas

Otro tipo de objeto que usa R son las listas, que es un tipo de objeto recursivo o que admite diferentes formatos de elementos. Las listas básicamente son objetos que combinan varios objetos que pueden ser matrices, vectores, dataframes, sin restricciones. En resumen, las listas



son mini contenedores, y se usan mucho para las funciones cuando necesitamos “pegar” o usar objetos de diferentes formatos para obtener un solo resultado. También son útiles cuando hacemos “**for loops**” o bucles, para hacer tareas repetitivas. El manejo de listas es un poco avanzado para comenzar a trabajar en R, pero son muy importantes y no pueden ser pasadas por alto en este nivel básico.

Cuando tratamos de visualizar una lista, lo primero que vemos es el *orden* del objeto dentro de doble corchete y luego el *contenido* del objeto. Copia el siguiente texto y pégalo en la consola para que veas un ejemplo:

```
#vamos a crear varios objetos

vect1 <- c("a", "b", "c")
vect2 <- c(seq(1:10))
vect3 <- letters
vect4 <- mtcars[1:5,2:3]
#vamos a crear una lista de objetos

lista <- list(vect1, vect2, vect3, vect4)
```

Si desde la misma consola llamamos al objeto *lista* que creamos, con solo escribir nombre del objeto, “lista” en este caso, tendremos el siguiente resultado:

```
lista

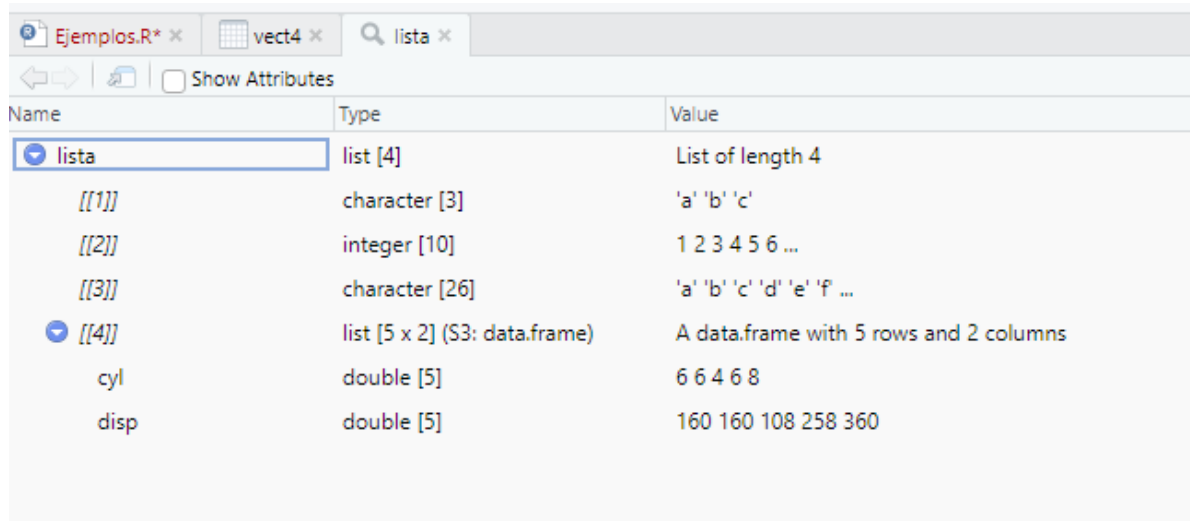
[[1]]
[1] "a" "b" "c"

[[2]]
[1] 1 2 3 4 5 6 7 8 9 10

[[3]]
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
[20] "t" "u" "v" "w" "x" "y" "z"

[[4]]
      cyl disp
Mazda RX4      6  160
Mazda RX4 Wag    6  160
Datsun 710      4  108
Hornet 4 Drive   6  258
Hornet Sportabout 8  360
```

También desde el panel del ambiente de trabajo en Rstudio podemos ver los objetos tipo listas, cuantos elementos tiene y si desplegamos podemos ver sus elementos, así como también el tipo de elemento y características.



| Name  | Type                          | Value                                  |
|-------|-------------------------------|--|
| lista | list [4]                      | List of length 4                       |
| [[1]] | character [3]                 | 'a' 'b' 'c'                            |
| [[2]] | integer [10]                  | 1 2 3 4 5 6 ...                        |
| [[3]] | character [26]                | 'a' 'b' 'c' 'd' 'e' 'f' ...            |
| [[4]] | list [5 x 2] (S3: data.frame) | A data.frame with 5 rows and 2 columns |
| cyl   | double [5]                    | 6 6 4 6 8                              |
| disp  | double [5]                    | 160 160 108 258 360                    |

Figura 5.5: Si queremos llamar un objeto dentro de una lista simplemente escribimos el nombre de la lista abrimos doble corchete y ponemos dentro el numero de orden que corresponde el elemento que queremos,

```
lista[[1]] #nos nos arroja el resultado de todo el contenido del elemento 1, que en este caso
```

```
[1] "a" "b" "c"
```

Dentro de los ejercicios que vamos a hacer usaremos las listas, por ejemplo, con estas podemos cargar múltiples archivos como objetos desde una carpeta para facilitar el proceso de importación.

Aquí un escenario de ejemplo:

En la carpeta de “**datos**” o “**Fuente**” dentro del proyecto en que se esté trabajando digamos que hay 3 bases de datos en formato excel con similar estructura (ejemplo, provincia\_a.xlsx, provincia\_b.xlsx, provincia\_c.xlsx) y la tarea es unir estas bases de datos en una sola base de datos:

```
#Cargando el paquete {rio} para importar bases de datos y {tidyverse} para manipulación de datos
pacman::p_load(rio,
               tidyverse)
```

```
#para crear un vector con las rutas completa de varios archivos
ruta_archivos <- dir("datos", #nombre de la carpeta donde están los archivos fuente
                    pattern = "xlsx", #patrón presente en todos los archivos a importar
                    full.names = TRUE #argumento para crear la ruta completa de cada archivo
                    )

lista_bases <- map(ruta_archivos, import)# la funcion map (que viene de tidyverse) va a crear
head(lista_bases[[1]]) #este comando nos permite ver las primeras 5 filas en la base de datos
```

## 5.5 Funciones

Otro tipo de objeto que no necesariamente almacena datos, pero sí podemos crear objetos de datos, son las funciones. Si te has dado cuenta, hasta ahora se han mencionado mucho dado que todo el código que vamos a ir escribiendo se hace mayormente con funciones, de hecho, como se mencionó arriba, la mayoría de los paquetes son una compilación de funciones. Nosotros podemos crear funciones que usan funciones ya sean de R base o incluso de otros paquetes.

Las funciones nos ayudan a agilizar el trabajo porque son un conjunto de instrucciones dentro de nuestra rutina. Un ejemplo de esto es que podemos crear una función que nos produzca un gráfico de curva epidémica de un período y lugar determinados y obtendríamos el resultado, el gráfico, y los dos parámetros: el periodo y el lugar.

Para crear una función usamos la siguiente sintaxis:

|                   |                        |  |  |
|-------------------|------------------------|--|--|
| <b>mi_funcion</b> | <b>&lt;-</b>           | <b>function( par_a, par_b, ....)</b>                                   | <b>{ cuerpo de la función }</b>            |
| Nombre del objeto | Operador de asignación | Función para crear funciones y los parámetros que contendrá la función | Expresiones o códigos que usará la función |

Vamos hacer un ejercicio simple para entender mejor una función:

```
#una funcion para calcular el indice de masa corporal
mi_funcion <- function(peso, altura){

  resultado <- round(peso/(altura/100)^2,2) #asigna al elemento resultado el la operación entera

  print(paste0("Formula: ",peso,"kg./",altura/100,"(m)2")) # retorna un texto con el peso y la altura

  print(paste0("El indice de masa corporal es ", resultado)) # retorna el resultado en un solo texto en verde

}
```

```
mi_funcion(peso=115,altura=180) #llamamos la funcion y escribimos los parametros
```

```
[1] "Formula: 115kg./1.8(m)2"  
[1] "El indice de masa corporal es 35.49"
```

Para entender mejor cómo trabaja una función usando el ejemplo anterior, escribimos la función **function(){ }** luego del nombre de la función y el operador de asignación, y dentro de los paréntesis escribimos el nombre que queremos usar para los parámetros separados por coma, como hicimos en el ejemplo anterior, usamos peso y altura. Luego, dentro de las llaves vamos a proceder a escribir las expresiones para obtener un resultado; para esta función que calcula el índice de masa corporal (IMC), creamos un vector que es el resultado de la operación de la fórmula del IMC, donde el parámetro 1 es el peso que es dividido entre el parámetro 2, luego el parámetro 2 que es la altura, que a su vez es dividida entre 100 y luego elevada al cuadrado; la siguiente expresión presenta un texto con los parámetros y luego presenta otro texto con el resultado del IMC.

De las funciones se pueden escribir libros, de hecho, hay una vasta documentación de cómo trabajar con ellas, pues son el pilar de la programación en R.

Cuando te sientas más familiarizado y tengas más experiencia en R, puedes dedicarle tiempo a aprender sobre cómo trabajar más profundamente con las funciones. Por ahora, vamos a enfocarnos en los paquetes, que tienen funciones que nos facilitan el trabajo para el análisis de datos.

## 6 Transición desde Excel a R

En el aprendizaje de hacer análisis de datos usualmente se comienza usando hojas de cálculo como Microsoft Excel® para hacer tareas comunes como tablas, gráficos y cálculos estadísticos de forma general.

Si ya estás trabajando con datos de forma regular, es muy probable que también te hayas expuesto a Excel, o lo usas constantemente en tu día a día como parte de las herramientas para procesar, analizar y presentar datos.

En nuestro caso, incluso sabiendo de la existencia de R, creíamos que con Excel podíamos hacerlo todo, como resolver cualquier situación relacionada con los datos, dado que MS Excel tiene muchas funcionalidades y es una herramienta muy completa como hoja de cálculo.

A pesar de lo robusto que puede ser Excel u otra hoja de cálculo, hay límites donde la funcionalidad de R va más allá, empezando por la cantidad de datos que se pueden cargar en un archivo de Excel, el cual tiene un límite de 1,048,576 filas por hoja y 16,384 columnas, y aunque se puede usar Power Query para tener más filas y columnas, no menos cierto es que el archivo se hace muy “pesado” o “lento” para trabajar cuando se usa cierta cantidad de filas y/o columnas. Con R se pueden manejar bases de datos más grandes de manera más rápida, dependiendo de la memoria RAM instalada de la computadora. Te recomendamos que busques en la web cómo aumentar la memoria RAM de tu computadora.

Volviendo a Excel, una hoja de cálculo en general es muy intuitiva y fácil de aprender, solo con abrir una hoja podemos comenzar a escribir agregando datos, escribir funciones para cálculos, hacer gráficos, etc., todo en un entorno guiado a través de los clics del ratón.

En cambio, R, como es un lenguaje de programación, es basado en comandos, sintaxis, donde mayormente todo debe de ser escrito, y realmente, es poco intuitivo al momento de comenzar a usarlo debido a lo complejo que resulta aprender un lenguaje nuevo. Sin embargo, esta forma de proceder a través de texto, también tiene sus ventajas cuando vamos a realizar una tarea, dado que debemos tener definido, de una forma u otra, lo que vamos a hacer y cuáles comandos deben de ser ejecutados para lograr la tarea. De hecho, de forma general, así es que debemos de trabajar, es decir, primero tener un plan en mente, cómo será ejecutado y definir cada paso. La idea es que el cambio de ambiente (pasar de usar más el ratón que escribir) puede ser un poco difícil al principio y sentirnos frustrados, pero con la constante exposición a un nuevo ambiente nos vamos adaptando, y se le hace fácil al cerebro en la medida que vamos practicando cada vez más y más.

En pocas palabras, es estar un poco “abierto” a cosas nuevas y darse la oportunidad de aprender y esforzarse más.

## 6.1 Tareas que se realizan en Excel (u otras hojas de cálculos) y su equivalente en R

Entrando en el tema de que cosas podemos hacer muy parecidas en R que se hacen con Excel, que son muy propias de lo que hacemos en epidemiología cuando analizamos datos es la exploración, en R podemos al igual que Excel podemos ver las bases de datos en un formato muy parecido a Excel, por ejemplo, podemos usar el comando **View()** (con V mayúscula) puedes escribir en la consola **View(mtcars)[1]** y se nos abrirá una ventana una tabla (no editable por cierto) pero nos sirve para ver los datos y filtrar muy parecido a Excel.

Otra tarea muy común que realizamos en Excel son las “pivot tables” o tablas dinámicas, que nos permiten hacer tablas resúmenes de una base de datos con unos cuantos clics de ratón, en R, podemos hacer lo mismo con varios comandos.

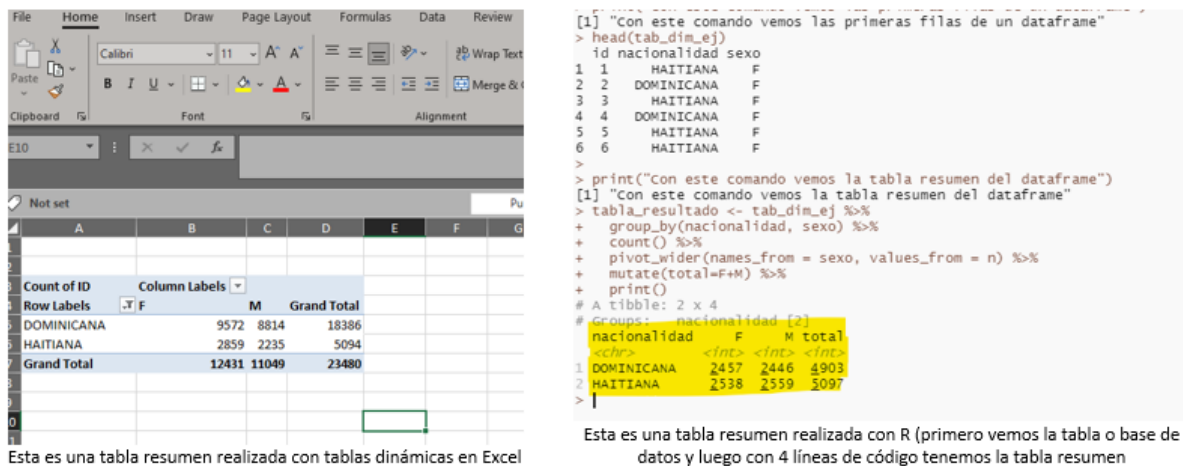


Figura 6.1: Comparación entre Excel y R, tablas dinámicas

A simple vista se pudiera ver que en R es más complicado porque hay que escribir lo que se va a realizar, pero tomando en cuenta que en nuestro trabajo del día a día como investigadores hacemos muchas tablas resúmenes en Excel o tenemos que actualizarlas, y esto realmente toma mucho tiempo, sin embargo, en R solo nos tome tiempo hacer la primera tabla resumen, y posteriormente, tendríamos la ventaja de reusar el código solo cambiando algunos parámetros (como las variables), ahorrándonos mucho tiempo al no tener que repetir la misma tarea una y otra vez.

En Excel también usamos mucho las funciones y fórmulas para hacer cálculos, un ejemplo común es crear grupos de edades a partir de una variable numérica que representa la edad. De una forma u otra, usar funciones o fórmulas en Excel es similar a escribir códigos, y si puedes hacer fórmulas en Excel, entonces significa que ¡ya tienes experiencia codificando! En R, gracias a la disponibilidad de tantos paquetes (como el **epikit**), tenemos funciones que son específicamente para esto que se requiere menos esfuerzo.

En resumen, hay una vasta documentación disponible sobre cómo adaptarse al uso de R para usuarios de Excel, cuáles son las diferencias entre R y Excel y cómo se complementan. Cuando veamos sobre exportación de datos, veremos un buen ejemplo de esto.

En la web hay mucha documentación acerca de cómo acostumbrarse a realizar análisis de datos con R para usuarios de otros programas; recomendamos que hagas una búsqueda sobre el tema para que vayas adaptándote.

Nuestra mayor recomendación es que comiences a practicar con R haciendo lo que ya sabes hacer con Excel. También puedes fácilmente ver la documentación de una función escribiendo el signo de interrogación (?) delante de la función que quieres: **?sum()** y luego en el panel de ayuda vas a ver cuáles son los argumentos y también un ejemplo. En caso de no entender, busca en internet. ¿Te has dado cuenta de que hacemos mucho énfasis en “*Buscar en la web*”? Es porque esa es la mayor ventaja que tiene R como lenguaje de programación para análisis de datos, la comunidad de apoyo.

[1] mtcars es una base de datos que trae R internamente como ejemplo sobre carros, para saber más de esta escribe en la consola `help("mtcars")`<sup>1</sup>

---

<sup>1</sup>**mtcars** es una base de datos que trae R internamente como ejemplo sobre carros, para saber más de esta escribe en la consola `help("mtcars")`

## 7 Análisis de datos usando R

### 7.1 Tareas que se deben de hacer para llevar a cabo un proyecto de análisis de datos

Es posible que ya tengas conocimiento sobre la estructura del documento que debes desarrollar para analizar tus datos, ya sea para toma de decisiones o para elaborar un trabajo final. En esta sección nos enfocaremos en la generación de resultados o salidas de R, donde tienes que realizar tablas, gráficos y análisis estadísticos. A continuación, te mostramos un listado de los elementos requeridos en la sección de resultados partiendo de un contexto clínico o epidemiológico:

- Proporciona número de casos, incidencia o prevalencia de un evento.
- Características clínicas, por ejemplo, síntomas comunes, porcentaje de hospitalizados o fallecidos, resultados de laboratorio como porcentaje de confirmados o distribución por especie o subtipo (generalmente presentados en una tabla).
- Tiempo: casos por año, mes, semana u otro intervalo apropiado, para mostrar patrones o cambios a lo largo de un período determinado. Puede estratificarse por grupos de edad, sexo, región o características de persona; lugar (generalmente se presenta en un gráfico). Señale cambios importantes, tendencias estacionales, aberraciones (Ej., Brotes) u otros patrones inusuales.
- Lugar: como el área geográfica donde ocurre el evento. Esto, generalmente, se presenta con un mapa o una tabla.
- Persona, por ejemplo, por grupo de edad, sexo y otras características relevantes (generalmente presentadas en una tabla).

Existen otros resultados destacados que no se incluyen en las categorías enumeradas anteriormente. Por ejemplo, muchos informes resumidos de vigilancia epidemiológica incluyen datos sobre la integridad y puntualidad de los informes de cada fuente de informes.

Con una base de datos de ejemplo vamos a realizar los pasos en R para producir los requerimientos sugeridos anteriormente. Vamos a usar como fuente de datos para este ejercicio una base de datos de casos de VIH positivos notificados. (Esta base está disponible en el siguiente [link](#)).



Esta base de datos (o la que vayas a usar) debes de colocarla en la subcarpeta de “**Bases de datos**” dentro de la carpeta del **proyecto** (ejemplo la mi carpeta donde está el proyecto de Rstudio se llama trabajos de campo). A modo de refrescamiento, ver la siguiente imagen de la estructura de sub-carpetas de un proyecto, su jerarquía.

| Estructura                         | Ejemplos                                   |
|------------------------------------|--|
| -Escritorio u otra parte del disco | "C:\user\destktop\"                        |
| -Proyecto de R                     | "C:\user\destktop\mi_proyecto"             |
| -Datos                             | "C:\user\destktop\mi_proyecto\datos"       |
| -Rutinas                           | "C:\user\destktop\mi_proyecto\mis_rutinas" |
| -Documentos                        | "C:\user\destktop\mi_proyecto\documentos"  |

Figura 7.1: Estructura básica recomendada para un proyecto en rstudio.

### 7.1.1 Organización general antes de comenzar

Como refrescamiento, antes de comenzar recuerda crear el proyecto (ver la sección de Comenzar a trabajar con R y la interfaz de Rstudio)

#### Procesamiento de datos (cargar, editar, transformar *dataframes*)

- Crear un documento de rutina de R, (Ctrl+shift+N o en File, R script),
- Comentar al menos el título del trabajo que estas haciendo (escribir # , que es el carácter para hacer comentarios, después de este puedes escribir cualquier cosa y no será interpretado como código).
- Instalar el paquete **pacman**(luego nos permitirá con mayor facilidad instalar el resto)
- Cargar los siguientes paquetes:
  1. **rio** (para cargar archivos excel y otros formatos)
  2. **tidyverse** (para transformar, revisar la base de datos)
  3. **janitor** (para tablas y limpieza de datos)
  4. **flextable** (para formato de presentación de las tablas)
  5. **lubridate** (para trabajar con funciones con variables de formatos de fecha)
  6. **skimr** (para revisar la base de datos)
  7. **here** (para ayudarnos a encontrar los archivos que vamos a usar, también a guardarlos)

8. **gtsummary** (para hacer tablas presentables y cálculos que usamos con frecuencia en epidemiología)

Para cargar estos paquetes procedemos a usar la función **p\_load()** del paquete *{pacman}*, en el panel de editor de rutinas, escribes el siguiente comando:

```
install.packages("pacman") #para instalar el paquete pacman (solo una vez)
```

```
pacman::p_load(rio,
               tidyverse,
               janitor,
               lubridate,
               skimr,
               here,
               flextable,
               gtsummary) #para instalar y cargar los paquetes necesarios
```

Luego de copiar o de introducir el código, preciona **Ctrl+ENTER** al final de la línea de código o en “run” para ejecutarlo.

Otra alternativa para cargar los paquetes es con la función de R base **library()**. Al igual que con **p\_load()** de *pacman*, escribes el nombre o los nombres de los paquetes separados por coma. **Ojo**, esta función solo carga los paquetes, si el paquete no está instalado, te dará error.

NOTA: Debes tener conexión a internet para poder descargar estos paquetes.

Después de instalar los paquetes no necesitarás instalarlos de nuevo, a menos que re-instales o actualices Rstudio.

Antes de continuar, guarda la rutina en la carpeta de tu proyecto “rutinas” utilizando cualquiera de estas opciones:

- Presionando la combinación de teclas **Ctrl+S**
- Haciendo clic en el icono de guardar
- Desde el menú: File -> Save

El próximo paso es cargar la base de datos, y esto lo podemos hacer de dos formas: escribiendo directamente en la rutina o en la consola, tal como vimos cuando explicamos los **dataframes** en el capítulo 5 de objetos de R, o bien, usando la interfaz de Rstudio.

Recuerda tener tus bases de datos en la carpeta de “datos” dentro de tu carpeta del proyecto, esto es muy importante para facilitar el trabajo.

Vamos a ver cómo sería el código para cargar un archivo de Excel:

```
#para crear un objeto dataframe (nuestra base)
base <- import(here("datos", "sinave_vih.xlsx")) %>%
  clean_names()
```

Explicando un poco el código anterior: estamos creando un nuevo objeto llamado “**base**”, a través de la función de **import** del paquete **rio**, que sirve para cargar archivos tipo .xls, .xlsx., y con la finalidad de localizar la ruta del archivo de la forma más sencilla posible, usamos la función **here()** del paquete **here** escribiendo los parámetros de la sub-carpeta, es decir, agregando entre comillas el nombre de la carpeta donde está la base de datos (primer parámetro: “datos”), y después de una coma se agrega el segundo parámetro que es el nombre del archivo, en este ejemplo, el archivo se denomina *sinave\_vih.xlsx*. Luego sigue un operador pipe (%>%) que significa “luego” y a través de la función **clean\_names()** del paquete **janitor** podremos “normalizar” los nombres de las columnas o variables de la base de datos importada. Esta normalización consiste, por ejemplo, en estandarizar los nombres de las variables poniéndolas todas en minúsculas, quitar caracteres poco comunes o espacios, entre otros ajustes que consideremos importantes.

Si estás utilizando la misma base de datos del ejemplo, y si hiciste los pasos correctamente, debes de tener una imagen similar a la de la Figura 23, donde puedes ver a la derecha que ya tienes un objeto *dataframe* cargado (es decir, la base de datos) que contiene 25,227 filas y 71 variables o columnas. (ver figura Figura ??)

Donde puedes ver a la derecha que ya tienes un objeto dataframe cargado (la base de datos) que tiene 25,227 filas y 71 variables o columnas.

#### 7.1.1.1 Exploración de la base de datos

Entonces, la primera pregunta que te hacemos es; ¿Cuál es el próximo paso por seguir? Realmente debería ser el análisis, pero primero, sería bueno revisar la base de datos para ver los datos “malos” es decir, hacer una exploración para identificar valores anormales, campos vacíos o datos que se cargaron mal, como pasa a veces con las fechas.

El paquete de **{rio}** con la fórmula de **import()** hace un intento de determinar los tipos de variables que se cargan desde el archivo de Excel, pero a veces falla y es, usualmente, con las fechas, porque sin querer había una fecha escrita en un formato no reconocido y el resto como número.

Para explorar la base de datos, podemos hacerlo de forma directa haciendo clic en el panel de ambiente de trabajo en el objeto base (o el nombre que le hayas dado), o puedes escribir en la consola de comandos **View(base)** para cargar el visor de datos.

Otra forma más completa de explorar la base de datos es a través de la función **skim()** o **skim\_tee()** del paquete **{skimr}** (Waring et al. 2022), puedes utilizar cualquiera de las dos porque ambas producen el mismo reporte. Con estas funciones obtendremos un resumen de

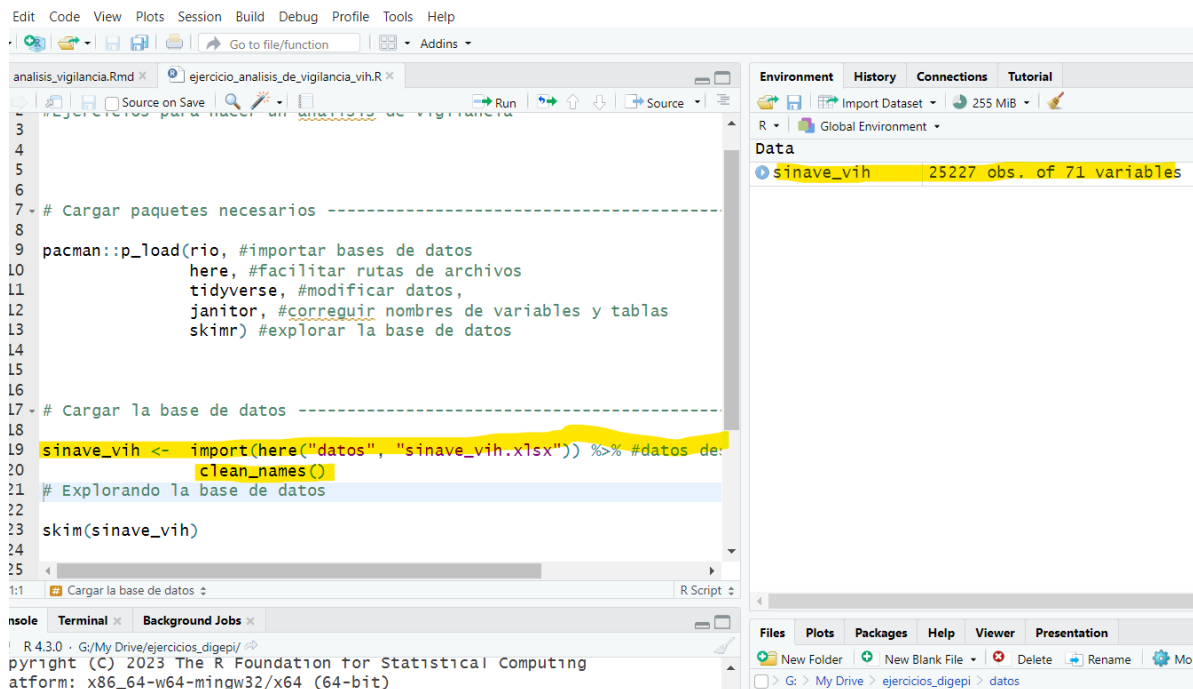


Figura 7.2

cada variable, qué tipo de variable es cada una y muestra el total de campos vacíos, valores únicos, entre otros detalles importantes.

Con estos simples pasos, ya estamos entrando de lleno en el análisis. Recuerda, siempre el primer paso es verificar los datos, si hay valores extremos, datos faltantes, etc., esta es una buena práctica (diríamos que obligatoria) cuando estamos haciendo análisis, luego viene la limpieza de los datos.

Luego de escribir en tu rutina el último comando, debes tener escrito el siguiente código y obtener este resultado:

```

base <- import(here("datos", "sinave_vih.xlsx")) %>%
  clean_names()

skimr::skim_tee(base) #para genera un mini reporte de la base

```

-- Data Summary -----

|                   | Values |
|-------------------|--------|
| Name              | data   |
| Number of rows    | 25615  |
| Number of columns | 41     |

-----  
Column type frequency:

|           |    |
|-----------|----|
| character | 20 |
| logical   | 1  |
| numeric   | 17 |
| POSIXct   | 3  |

-----  
Group variables               None

-- Variable type: character -----

|    | skim_variable           | n_missing | complete_rate | min | max | empty | n_unique |
|----|-------------------------|-----------|---------------|-----|-----|-------|----------|
| 1  | sexo                    | 0         | 1             | 8   | 9   | 0     | 2        |
| 2  | grupo_edad              | 0         | 1             | 2   | 8   | 0     | 9        |
| 3  | actividad_ocupacional   | 24767     | 0.0331        | 5   | 120 | 0     | 167      |
| 4  | grupo_ocupacional       | 24769     | 0.0330        | 21  | 76  | 0     | 10       |
| 5  | categoría_de_afiliación | 0         | 1             | 10  | 23  | 0     | 5        |
| 6  | nivel_educativo         | 6763      | 0.736         | 8   | 25  | 0     | 6        |
| 7  | provincia               | 0         | 1             | 2   | 2   | 0     | 33       |
| 8  | colectivo               | 8976      | 0.650         | 6   | 36  | 0     | 11       |
| 9  | region                  | 0         | 1             | 1   | 4   | 0     | 9        |
| 10 | tipo_atencion           | 82        | 0.997         | 8   | 13  | 0     | 4        |
| 11 | complicaciones          | 24005     | 0.0629        | 7   | 69  | 0     | 19       |
| 12 | muestra                 | 268       | 0.990         | 2   | 2   | 0     | 2        |
| 13 | resultado_final         | 20575     | 0.197         | 9   | 10  | 0     | 3        |
| 14 | condicion               | 0         | 1             | 4   | 6   | 0     | 2        |
| 15 | gravedad                | 16183     | 0.368         | 5   | 20  | 0     | 3        |
| 16 | edad_fecha_defuncion    | 25510     | 0.00410       | 5   | 8   | 0     | 74       |
| 17 | diag_final              | 24394     | 0.0477        | 10  | 56  | 0     | 6        |
| 18 | clasf_final             | 24394     | 0.0477        | 10  | 10  | 0     | 3        |
| 19 | fuentes_deteccion       | 0         | 1             | 14  | 24  | 0     | 5        |
| 20 | confirmado_por          | 24396     | 0.0476        | 11  | 19  | 0     | 2        |
|    | whitespace              |           |               |     |     |       |          |
| 1  | 0                       |           |               |     |     |       |          |
| 2  | 0                       |           |               |     |     |       |          |
| 3  | 0                       |           |               |     |     |       |          |
| 4  | 0                       |           |               |     |     |       |          |
| 5  | 0                       |           |               |     |     |       |          |
| 6  | 0                       |           |               |     |     |       |          |
| 7  | 0                       |           |               |     |     |       |          |
| 8  | 0                       |           |               |     |     |       |          |
| 9  | 0                       |           |               |     |     |       |          |
| 10 | 0                       |           |               |     |     |       |          |
| 11 | 0                       |           |               |     |     |       |          |

```

12      0
13      0
14      0
15      0
16      0
17      0
18      0
19      0
20      0

```

```
-- Variable type: logical -----
```

```

  skim_variable      n_missing complete_rate mean count
1 fecha_inicio_erupcion      25615           0 NaN ": "

```

```
-- Variable type: numeric -----
```

```

  skim_variable      n_missing complete_rate      mean      sd      p0
1 pxid              0           1      12808      7395.      1
2 fecha_nacimiento    0           1      30282.      4960.     8416
3 edad1             119          0.995      36.8       13.4      0
4 edad2            20494          0.200      0.0842      0.710      0
5 edad3            20537          0.198      0.101       1.40      0
6 pais_procedencia    0           1        1.18       0.398      1
7 semana_inicio_sintomas  0           1       25.6       15.3      1
8 mes_inicio_sintomas  0           1        6.32       3.50      1
9 ano_inicio_sintomas  0           1      2019.       1.41     2016
10 semana_atencion    0           1       25.7       15.2      1
11 mes_atencion       0           1        6.31       3.48      1
12 ano_atencion       0           1      2019.       1.40     2017
13 semana_toma_muestra 13577          0.470      25.5       14.9      1
14 fecha_toma_muestra 13577          0.470  43672.       531.     42737
15 institucion        0           1        4.53       0.757      1
16 semana_notificacion 0           1       26.1       15.0      1
17 fecha_notificacion  0           1     43738.       513.     42738

```

```

  p25  p50  p75  p100 hist
1 6404. 12808 19212. 25615
2 27144. 30902 33921 44501
3   27    35    45    98
4    0     0     0    11
5    0     0     0    30
6    1     1     1     3
7   11    26    39    53
8    3     6     9    12
9 2018   2019 2020 2021

```

```

10    11    26    39    53
11     3     6     9    12
12  2018   2019  2020   2021
13    12    25    38    53
14 43214  43638 44174  44567
15     4     5     5     5
16    12    26    39    53
17 43315  43718 44232  44615

```

```

-- Variable type: POSIXct -----
  skim_variable      n_missing complete_rate min
1 fecha_inicio_sintomas      0         1 2016-12-03 00:00:00
2 fecha_atencion            0         1 2017-01-02 00:00:00
3 fecha_defuncion        25510    0.00410 2017-02-03 00:00:00
  max                median          n_unique
1 2022-01-01 00:00:00 2019-08-15 00:00:00      1811
2 2022-01-01 00:00:00 2019-08-23 00:00:00      1653
3 2022-01-01 00:00:00 2020-10-06 00:00:00       100

```

En el ejemplo anterior:

- ¿Cuántas variables de texto, numéricas, lógicas (si/no, 1/0), de fechas se cargaron?
- ¿Cuántas de las variables están en blanco o tienen muchos valores vacíos, cómo es la distribución de las variables numéricas y fechas, (valores extremos)?
- ¿Hay variables que se importaron incorrectamente? Variables que son de un formato y se importaron de otro tipo (fechas que se importan como texto o número por ejemplo)

Estas son las preguntas que debemos hacernos a partir de este resumen, para ir viendo la data y hacer la limpieza de datos, excluir columnas o variables, filtrar valores extremos o editarlos, cambiar o corregir el formato, etc.

En el ejemplo anterior vemos que vemos que el resultado arrojó 4 tablas con un tipo de variable cada una. La reproducimos aquí para visualizarla mejor: