Sumario

[Manual de R para Epidemiología 2](#__RefHeading___Toc31371_2034561403)

[Sobre este libro 5](#__RefHeading___Toc31381_2034561403)

[#Notas editoriales y técnicas 5](#__RefHeading___Toc31383_2034561403)

[#Descargando el manual y los datos 10](#__RefHeading___Toc31393_2034561403)

[Conocimientos básicos 16](#__RefHeading___Toc31399_2034561403)

[#Aspectos básicos de R 16](#__RefHeading___Toc31401_2034561403)

[#Transición a R 48](#__RefHeading___Toc31429_2034561403)

[#Paquetes recomendados 57](#__RefHeading___Toc31439_2034561403)

[#Proyectos en R 58](#__RefHeading___Toc31445_2034561403)

[#Importación y exportación 60](#__RefHeading___Toc31455_2034561403)

[Gestión de datos 76](#__RefHeading___Toc31487_2034561403)

[#Limpieza de datos y funciones básicas 76](#__RefHeading___Toc31489_2034561403)

[#Trabajar con fechas 103](#__RefHeading___Toc31517_2034561403)

[#Caracteres y cadenas 114](#__RefHeading___Toc31543_2034561403)

[#Factores 128](#__RefHeading___Toc31563_2034561403)

[#Pivotar datos 137](#__RefHeading___Toc31581_2034561403)

[#Agrupar datos 143](#__RefHeading___Toc31593_2034561403)

[#Unir datos 151](#__RefHeading___Toc31617_2034561403)

[#Desduplicación 164](#__RefHeading___Toc31629_2034561403)

[#Iteración, bucles y listas 172](#__RefHeading___Toc31643_2034561403)

[Análisis 187](#__RefHeading___Toc31655_2034561403)

[#Tablas descriptivas 187](#__RefHeading___Toc31657_2034561403)

[#Tests estadísticos sencillos 202](#__RefHeading___Toc31673_2034561403)

[#Regresión univariable y multivariable 206](#__RefHeading___Toc31687_2034561403)

[#Valores faltantes 215](#__RefHeading___Toc31701_2034561403)

[#Tasas estandarizadas 225](#__RefHeading___Toc31717_2034561403)

[#Medias móviles 231](#__RefHeading___Toc31729_2034561403)

[#Series temporales y detección de brotes 236](#__RefHeading___Toc31739_2034561403)

[#Modelización de epidemias 248](#__RefHeading___Toc31759_2034561403)

[#Rastreo de contactos 255](#__RefHeading___Toc31771_2034561403)

[#Análisis de encuestas 262](#__RefHeading___Toc31785_2034561403)

[#Análisis de supervivencia 270](#__RefHeading___Toc31807_2034561403)

[#Conceptos básicos de los SIG 280](#__RefHeading___Toc31823_2034561403)

[Visualización de datos 298](#__RefHeading___Toc31849_2034561403)

[#Tablas para presentaciones 298](#__RefHeading___Toc31851_2034561403)

[#Conceptos básicos de ggplot 303](#__RefHeading___Toc31865_2034561403)

[#Consejos de ggplot 321](#__RefHeading___Toc31895_2034561403)

[#Curvas epidémicas 333](#__RefHeading___Toc31925_2034561403)

[#Pirámides demográficas y escalas de Likert 353](#__RefHeading___Toc31943_2034561403)

[#Gráficos de calor 359](#__RefHeading___Toc31955_2034561403)

[#Diagramas y gráficos 365](#__RefHeading___Toc31965_2034561403)

[#Análisis de combinaciones 373](#__RefHeading___Toc31979_2034561403)

[#Cadenas de transmisión 375](#__RefHeading___Toc31989_2034561403)

[#Árboles filogenéticos 382](#__RefHeading___Toc32003_2034561403)

[#Gráficos interactivos 388](#__RefHeading___Toc32017_2034561403)

[Informes y Dashboards 391](#__RefHeading___Toc32029_2034561403)

[#Informes con R Markdown 391](#__RefHeading___Toc32031_2034561403)

[#Organización de informes rutinarios 409](#__RefHeading___Toc32053_2034561403)

[#Dashboards con R Markdown 414](#__RefHeading___Toc32069_2034561403)

[#Dashboards con Shiny 425](#__RefHeading___Toc32091_2034561403)

[Miscelánea 442](#__RefHeading___Toc32113_2034561403)

[#Funciones de escritura 443](#__RefHeading___Toc32115_2034561403)

[#Interacciones con directorios 449](#__RefHeading___Toc32135_2034561403)

[#Control de versiones y colaboración con Git y Github 453](#__RefHeading___Toc32155_2034561403)

[#Errores comunes 472](#__RefHeading___Toc32183_2034561403)

[#Cómo obtener ayuda 475](#__RefHeading___Toc32191_2034561403)

[#R en redes locales 478](#__RefHeading___Toc32201_2034561403)

[#data.table 482](#__RefHeading___Toc32211_2034561403)

Código

* Mostrar todos los códigos
* Ocultar todo el código

# Manual de R para Epidemiología

#### el equipo del manual

#### 2021-06-04

## R para epidemiología aplicada y salud pública

**Este manual pretende:**

* Servir como breve guía de referencia para escribir código en R
* Proporcionar ejemplos detallados que aborden problemas epidemiológicos.
* Ayudar a profesionales de la epidemiología en su transición a R
* Ser accesible en entornos con baja conectividad a Internet a través de una [**versión sin conexión**](#download-handbook-and-data)

**Escrito por profesionales de la epidemiología, para profesionales de la epidemiología**

Somos epis de campo de todo el mundo, escribiendo en nuestro tiempo libre para ofrecer este recurso a la comunidad. Tu apoyo y comentarios son muy bienvenidos:

* [**Formulario de respuesta**](https://forms.gle/A5SnRVws7tPD15Js9) estructurado
* [**Envía un correo electrónico a epiRhandbook@gmail.com**](mailto:epiRhandbook@gmail.com) o un tweet [**a @epiRhandbook**](https://twitter.com/epirhandbook)
* Envía problemas a nuestro [**repositorio de Github**](https://github.com/epirhandbook/Epi_R_handbook)

## Cómo utilizar este manual

* Navega por las páginas del índice o utiliza el cuadro de búsqueda
* Clica en los iconos "Copy" para copiar el código
* Puedes seguir paso a paso las lecciones utilizando nuestras bases de [datos de ejemplo](#download-handbook-and-data)
* Consulta la sección "Recursos" de cada página para obtener más material

**Versión sin conexión**

Consulta las instrucciones en el [manual de descarga](#download-handbook-and-data) y la página de [datos](#download-handbook-and-data).

**Idiomas**

**Queremos traducir este manual a otros idiomas. Si deseas ayudar, ponte en contacto con nosotros.**

## Agradecimientos

Este manual ha sido elaborado mediante la colaboración de profesionales de la epidemiología de todo el mundo, basándonos en nuestra experiencia en organismos sanitarios locales, estatales, provinciales y nacionales, la Organización Mundial de la Salud (OMS), Médicos Sin Fronteras (MSF), sistemas hospitalarios e instituciones académicas.

Este manual **no** es un producto aprobado por ninguna organización específica. Aunque nos esforzamos por ser precisos, no ofrecemos ninguna garantía sobre el contenido de este libro.

### Colaboradores

**Redactor jefe:** [Neale Batra](https://www.linkedin.com/in/neale-batra/)

**Equipo central del proyecto:** [Neale Batra](https://www.linkedin.com/in/neale-batra/), [Alex Spina](https://github.com/aspina7), [Amrish Baidjoe](https://twitter.com/Ammer_B), Pat Keating, [Henry Laurenson-Schafer](https://github.com/henryls1), [Finlay Campbell](https://github.com/finlaycampbell)

**Autores**: [Neale Batra](https://www.linkedin.com/in/neale-batra/), [Alex Spina](https://github.com/aspina7), [Paula Blomquist](https://www.linkedin.com/in/paula-bianca-blomquist-53188186/), [Finlay Campbell](https://github.com/finlaycampbell), [Henry Laurenson-Schafer](https://github.com/henryls1), [Isaac Florence](../../../../../C:/Users/Neale/OneDrive%20-%20Neale%20Batra/Documents/Analytic%20Software/R/Projects/R%20handbook/Epi_R_handbook/www.Twitter.com/isaacatflorence), [Natalie Fischer](https://www.linkedin.com/in/nataliefischer211/), [Aminata Ndiaye](https://twitter.com/aminata_fadl), [Liza Coyer](https://www.linkedin.com/in/liza-coyer-86022040/), [Jonathan Polonsky](https://twitter.com/jonny_polonsky), [Yurie Izawa](https://ch.linkedin.com/in/yurie-izawa-a1590319), [Chris Bailey](https://twitter.com/cbailey_58?lang=en), [Daniel Molling](https://www.linkedin.com/in/daniel-molling-4005716a/), [Isha Berry](https://twitter.com/ishaberry2), [Emma Buajitti](https://twitter.com/buajitti), [Mathilde Mousset](https://mathildemousset.wordpress.com/research/), [Sara Hollis](https://www.linkedin.com/in/saramhollis/), Wen Lin

**Revisores**: Pat Keating, Annick Lenglet, Margot Charette, Danielly Xavier, Esther Kukielka, Michelle Sloan, Aybüke Koyuncu, Rachel Burke, Kate Kelsey, [Berhe Etsay](https://www.linkedin.com/in/berhe-etsay-5752b1154/), John Rossow, Mackenzie Zendt, James Wright, Laura Haskins, [Flavio Finger](../../../../../C:/Users/Neale/OneDrive%20-%20Neale%20Batra/Documents/Analytic%20Software/R/Projects/R%20handbook/Epi_R_handbook/ffinger.github.io), Tim Taylor, [Jae Hyoung Tim Lee](https://www.linkedin.com/in/jaehyoungtlee/), [Brianna Bradley](https://www.linkedin.com/in/brianna-bradley-bb8658155), [Wayne Enanoria,](https://www.linkedin.com/in/wenanoria) Manual Albela Miranda, [Molly Mantus](https://www.linkedin.com/in/molly-mantus-174550150/), Pattama Ulrich, Joseph Timothy, Adam Vaughan, Olivia Varsaneux, Lionel Monteiro, Joao Muianga

**Ilustraciones**: Calder Fong

### Financiación y apoyo

El manual recibió financiación de apoyo de [TEPHINET](https://www.tephinet.org/), la red mundial de Programas de Formación en Epidemiología de Campo (FETP) a través de una subvención para el desarrollo de capacidades de emergencia COVID-19.

La Red de Antiguos Alumnos de EPIET ([EAN)](https://epietalumni.net/) proporcionó apoyo administrativo, con un agradecimiento especial a Annika Wendland. EPIET es el Programa Europeo de Formación en Epidemiología de Intervención.

Un agradecimiento especial a Médicos Sin Fronteras (MSF) Centro Operativo de Ámsterdam (OCA) por su apoyo durante la elaboración de este manual.

Esta publicación fue apoyada por el Acuerdo de Cooperación número NU2GGH001873, financiado por los Centros para el Control y la Prevención de Enfermedades a través de TEPHINET, un programa de The Task Force for Global Health. Su contenido es responsabilidad exclusiva de los autores y no representa necesariamente las opiniones oficiales de los Centros para el Control y la Prevención de Enfermedades, el Departamento de Salud y Servicios Humanos, The Task Force for Global Health, Inc. o TEPHINET.

### Inspiración

La multitud de tutoriales y viñetas que aportaron conocimientos para el desarrollo del contenido del manual se acreditan en sus respectivas páginas.

De manera más general, las siguientes fuentes han servido de inspiración para este manual:  
[El proyecto "R4Epis"](https://r4epis.netlify.app/) (una colaboración entre MSF y RECON)  
[R Epidemics Consortium (RECON)](https://www.repidemicsconsortium.org/)  
[El libro R for Data Science (R4DS)](https://r4ds.had.co.nz/)  
[bookdown: Creación de libros y documentos técnicos con R Markdown](https://bookdown.org/yihui/bookdown/)  
[Netlify](https://www.netlify.com/) alberga este sitio web

## Condiciones de uso y contribución

### Licencia

  
Esta obra está bajo una [licencia Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](http://creativecommons.org/licenses/by-nc-sa/4.0/).

Los cursos académicos y los programas de formación en epidemiología pueden utilizar este manual con sus estudiantes. Si tiene preguntas sobre el uso que se le va a dar, envía un correo electrónico [**a**](mailto:epiRhandbook@gmail.com) epiRhandbook@gmail.com.

### Cita

Batra, Neale, et al. The Epidemiologist R Handbook. 2021.

### Contribución

Si quieres hacer una contribución de contenido, por favor, ponte en contacto con nosotros primero a través de Github o por correo electrónico. Estamos implementando un calendario de actualizaciones y estamos creando una guía para colaboradores.

Ten en cuenta que el proyecto epiRhandbook se publica con un [Código de Conducta del Colaborador](https://contributor-covenant.org/version/2/0/CODE_OF_CONDUCT.html). Al contribuir a este proyecto, te comprometes a respetar sus términos.

&&&&&

# Sobre este libro

# #Notas editoriales y técnicas

{#editorial-and-technical-notes}

En esta página describimos la filosofía, el estilo y las decisiones editoriales elegidas para la elaboración de este manual.

## Enfoque y estilo

El público potencial de este libro es amplio. Seguramente será utilizado tanto por personas muy noveles con R, como por usuarios experimentados buscando los mejores consejos y prácticas. Por lo tanto, este debe ser accesible y conciso a la vez. Por ello, nuestro enfoque fue proporcionar la información suficiente para que alguien muy nuevo en R pueda aplicar y seguir el código.

Otros puntos:

* Se trata de un libro de referencia de códigos acompañado de ejemplos relativamente breves, no de un libro de texto completo sobre R o ciencia de datos
* Este es un manual de R para su uso dentro de la epidemiología aplicada - no un manual sobre los métodos o ciencia de la epidemiología aplicada
* Se trata de un documento vivo: los paquetes de R óptimos para una tarea determinada cambian a menudo, por lo que agradecemos que exista debate sobre cuáles son los más empleados en el manual

### Paquetes de R

**Muchas opciones**

Uno de los aspectos más difíciles de aprender en R es saber qué paquete utilizar para una tarea determinada. Es muy común pelearse con una tarea para luego darse cuenta de que hay un paquete de R que hace todo eso en una línea de código.

En este manual, tratamos de ofrecerte al menos dos maneras de completar cada tarea: un método probado y comprobado (probablemente en R **base** o **tidyverse**) y un paquete especial de R que está hecho a medida para ese propósito. Queremos que tengas un par de opciones en caso de que no puedas descargar un paquete determinado o de que éste no te funcione.

A la hora de elegir los paquetes a utilizar, hemos dado prioridad a los paquetes y enfoques de R que han sido probados y aprobados por la comunidad, que minimizan el número de paquetes utilizados en una sesión de trabajo típica, que son estables (no cambian con frecuencia) y que realizan la tarea de forma sencilla y limpia.

En general, este manual da prioridad a los paquetes y funciones de R de **tidyverse**. Tidyverse es una colección de paquetes de R diseñados para ciencia de datos que comparten la gramática y estructuras de datos subyacentes. Todos los paquetes tidyverse pueden instalarse o cargarse a través del paquete **tidyverse**. Más información en el [sitio web de tidyverse](https://www.tidyverse.org/).

Cuando es aplicable, también ofrecemos opciones de código usando R **base** - los paquetes y funciones que vienen con R en la instalación. Esto se debe a que somos conscientes de que parte de la audiencia de este libro podría no tener una buena conexión a internet para descargar paquetes adicionales.

**Vinculación explícita de las funciones a los paquetes**

Es frustrante cuando en algunos tutoriales de R, se muestra una función (en código), pero no se sabe bien de qué paquete es. En este libro intentamos evitar esta situación.

En el texto explicativo, los nombres de los paquetes se escriben en negrita (por ejemplo, **dplyr**) y las funciones se escriben así: mutate(). Nos esforzaremos en dejar claro el paquete del que proviene una función, ya sea haciendo referencia al paquete en el texto o especificando el paquete en el código mediante esta sintaxis: dplyr::mutate(). Puede parecer redundante, pero lo hacemos a propósito.

Consulta la página sobre [los fundamentos de R](#r-basics) para saber más sobre los paquetes y las funciones.

### Código de estilo

En el manual, utilizamos con frecuencia "líneas nuevas", haciendo que nuestro código parezca "largo". Lo hacemos por varias razones:

* De esta forma se pueden escribir comentarios explicativos con #, los cuales están situados adyacentes a cada línea de código
* En general, el código más largo (en vertical) es más fácil de leer
* Es más fácil de leer en una pantalla estrecha (no es necesario desplazarse lateralmente)
* Con las sangrías, puede ser más fácil saber qué argumentos pertenecen a cada función

Como resultado, el código que podría estar escrito:

...se escribe así:

El código de R generalmente no se ve afectado por nuevas líneas o sangrías. Al escribir el código, si se inicia una nueva línea después de una coma, se aplicarán patrones de sangría automáticos.

También utilizamos muchos espacios (por ejemplo, n = 1 en lugar de n=1) porque es más fácil de leer. ¡Sé amable con la gente que lee tu código!

### Nomenclatura

En este manual, generalmente hacemos referencia a "columnas" y "filas" en lugar de "variables" y "observaciones". Como se explica en este manual sobre ["datos ordenados"](https://tidyr.tidyverse.org/articles/tidy-data.html), la mayoría de los conjuntos de datos estadísticos epidemiológicos se componen estructuralmente de filas, columnas y valores.

Las variables contienen los valores que miden el mismo atributo subyacente (como el grupo de edad, el resultado o la fecha de inicio). Las observaciones contienen todos los valores medidos en la misma unidad (por ejemplo, una persona, un lugar o una muestra de laboratorio). Por lo tanto, estos aspectos pueden ser más difíciles de definir de forma tangible.

En los conjuntos de datos "ordenados", cada columna es una variable, cada fila es una observación y cada celda es un único valor. Sin embargo, algunos conjuntos de datos que se encuentran no se ajustan a este molde: unos datos de formato "amplio" puede tener una variable dividida en varias columnas (véase un ejemplo en la página [Pivotar datos](#pivoting-data)). Del mismo modo, las observaciones pueden estar divididas en varias filas.

La mayor parte de este manual trata sobre la gestión y la transformación de datos, por lo que las referencias a las estructuras de datos concretas de filas y columnas son más relevantes que las observaciones y las variables más abstractas. Las excepciones se dan sobre todo en las páginas sobre análisis de datos, en las que verás más referencias a las variables y las observaciones.

### Notas

Estos son los tipos de notas que puede encontrar en el manual:

**NOTA: Esto es una** nota  
**CONSEJO: Esto es un** consejo.  
**PRECAUCIÓN: Esto es una nota de** precaución.  
**PELIGRO: Esto es una advertencia**.

## Decisiones editoriales

A continuación, hacemos un seguimiento de las decisiones editoriales importantes en torno a la elección de paquetes y funciones. Si no estás de acuerdo o quieres ofrecer una nueva herramienta para que la consideremos, únete o inicia una conversación en nuestra [página de Github](https://github.com/epirhandbook/Epi_R_handbook).

**Tabla de paquetes, funciones y otras decisiones editoriales**

| **Asunto** | **Considerado** | **Resultado** | **Breve explicación** |
| --- | --- | --- | --- |
| Enfoque general de codificación | **tidyverse**, **data.table**, **base** | **tidyverse**, con una página sobre **data.table**, y menciones de alternativas de R **base** para los lectores sin internet | Legibilidad de **tidyverse**, universalidad, más enseñado |
| Carga de paquetes | library(),install.packages(), require(), **pacman** | **pacman** | Acorta y simplifica el código para la mayoría de los casos de instalación/carga de paquetes múltiples |
| Importación y exportación | **rio**, muchos otros paquetes | **rio** | Facilidad para muchos tipos de archivos |
| Agrupación para las estadísticas de síntesis | **dplyr** group\_by(), **stats** aggregate() | **dplyr** group\_by() | Consecuente con el énfasis en **tidyverse** |
| Pivotar tablas | **tidyr** (funciones de pivote), **reshape2** (melt/cast), **tidyr** (spread/gather) | **tidyr** (funciones pivote) | **reshape2** se ha retirado, **tidyr** utiliza funciones pivot a partir de la v1.0.0 |
| Limpiar los nombres de las columnas | **linelist**, **janitor** | **janitor** | Se hace hincapié en la consolidación de los paquetes |
| Semanas epidemiológicas. Epiweeks | **lubridate**, **aweek**, **tsibble**, **zoo** | **Normalmente lubridate**, los otros para casos específicos | La flexibilidad, la coherencia y las perspectivas de mantenimiento de los paquetes de **lubridate** |
| Etiquetas ggplot | labs(), ggtitle()/ylab()/xlab() | labs() | Todas las etiquetas en un solo lugar, la simplicidad |
| Convertir en factor | factor(), **forcats** | **forcats** | Sus diversas funciones también se convierten en factor en el mismo comando |
| Curvas epidémicas | **incidence**, **ggplot2**, **EpiCurve** | **incidence2** por rapidez, **ggplot2** para tareas detalladas | fiabilidad |
| Concatenación | paste(), paste0(), str\_glue(), glue() | str\_glue() | Sintaxis más sencilla que las funciones de pegado; dentro de **stringr** |

## Revisiones

| **Fecha** | **Cambios importantes** |
| --- | --- |
| 10 de mayo de 2021 | Lanzamiento de la versión 1.0.0 |

## Información de la sesión (R, RStudio, paquetes)

A continuación se presenta la información sobre las versiones de R, RStudio y los paquetes de R utilizados en esta versión del Manual.

&&&&&

# #Descargando el manual y los datos

{#download-handbook-and-data}

## Descargar el manual sin conexión

Puedes descargar la versión sin conexión de este manual. Éste es un archivo HTML que puedes ver en tu navegador web sin acceder a Internet. Si estás pensando en utilizar este manual sin conexión, debes tener en cuenta algunas cosas:

* Al abrir el archivo, las imágenes y el índice pueden tardar uno o dos minutos en cargarse.
* Este manual tiene un diseño ligeramente diferente: una página muy larga con el índice a la izquierda. Para buscar términos específicos utiliza Ctrl+f (Cmd-f)
* Consulta la página de [Paquetes recomendados](#suggested-packages-1) para ayudarte a instalar los paquetes de R adecuados antes de que pierdas la conectividad a Internet
* Instala nuestro paquete R **epirhandbook** que contiene todos los datos del ejemplo (el proceso de instalación se describe a continuación)

**Hay dos maneras de descargar el manual:**

### Utilizando el enlace de descarga

Para acceder rápidamente, **clica con el botón derecho** [en este enlace](https://github.com/epirhandbook/Epi_R_handbook/raw/master/offline_long/Epi_R_Handbook_offline.html) **y selecciona "Guardar enlace como"**.

Si es un Mac, utiliza Cmd+clic. Si es un móvil, mantén clicado el enlace y selecciona "Guardar enlace". El manual se descargará en el dispositivo. Si aparece una pantalla con código HTML sin procesar, asegúrate de haber seguido las instrucciones anteriores o prueba la opción 2.

### Utilizando nuestro paquete R

Ofrecemos un paquete llamado **epirhandbook**. Este incluye la función download\_book() que descarga el archivo del manual desde nuestro repositorio de Github a tu ordenador.

Este paquete también contiene una función get\_data() que descarga todos los datos del ejemplo en tu ordenador.

Ejecuta el siguiente código para instalar nuestro paquete R **epirhandbook** desde el [repositorio de Github *appliedepi*](https://github.com/appliedepi/epirhandbook). Este paquete no está en CRAN, así que utiliza la función especial p\_install\_gh() para instalarlo desde Github.

Ahora, puedes cargar el paquete para utilizarlo en la sesión actual de R:

A continuación, ejecuta la función del paquete download\_book() (con los paréntesis vacíos) para descargar el manual en tu ordenador. Suponiendo que estés en RStudio, aparecerá una ventana que te permitirá seleccionar una ubicación para guardarlo.

## Descarga los datos para seguir el manual

Para "seguir" las páginas del manual, puedes descargar los datos y los resultados de los ejemplos.

### Utiliza nuestro paquete para R

El método más sencillo para descargar todos los datos es instalar nuestro paquete **epirhandbook**. Contiene una función get\_data() que guarda todos los datos del ejemplo en una carpeta de tu elección en tu ordenador.

Para instalar nuestro paquete **epirhandbook**, ejecuta el siguiente código. Este paquete no está en CRAN, así que utiliza la función p\_install\_gh() para instalarlo. La entrada hace referencia a nuestra organización de Github ("appliedepi") y al paquete **epirhandbook**.

Ahora, carga el paquete para utilizarlo en tu sesión actual de R:

A continuación, utiliza la función get\_data() del paquete para descargar los datos de ejemplo en tu ordenador. Ejecuta get\_data("all") para obtener todos los datos de ejemplo, o escribe un nombre de archivo específico y una extensión entre comillas para recuperar sólo un archivo.

Los datos ya se han descargado con el paquete, y sólo hay que transferirlos a una carpeta del ordenador. Aparecerá una ventana emergente para seleccionar la ubicación de la carpeta de almacenamiento. Te sugerimos que crees una nueva carpeta de "datos", ya que hay unos 30 archivos (incluidos los datos de ejemplo y los resultados de ejemplo).

Una vez que hayas utilizado get\_data() para guardar un archivo en tu ordenador, tendrás que importarlo a R. Consulta la página de [importación y exportación](#import-and-export) para más detalles.

Si lo deseas, puedes revisar todos los datos utilizados en este manual en la [**carpeta "data"**](https://github.com/epirhandbook/Epi_R_handbook/tree/master/data) de nuestro repositorio de Github.

### Descargar uno por uno

Esta opción implica la descarga de los datos archivo por archivo desde nuestro repositorio de Github a través de un enlace o un comando de R específico para el archivo. Algunos tipos de archivos permiten un botón de descarga, mientras que otros pueden descargarse mediante un comando de R.

#### Listado de casos

Se trata de un brote de ébola ficticio, ampliado por el equipo del manual a partir de los datos ebola\_sim de las prácticas del paquete **Outbreaks**.

* [Clica para descargar el listado de casos “en bruto” -raw- (.xlsx)](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/case_linelists/linelist_raw.xlsx). Este listado de casos "en bruto" es una hoja de cálculo de Excel con datos desordenados. Utilízala para seguir la página de [limpieza de datos y funciones básicas](#cleaning-data-and-core-functions).
* [Clica para descargar](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/case_linelists/linelist_cleaned.rds) [el](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/case_linelists/linelist_raw.xlsx) listado de casos “en [limpi](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/case_linelists/linelist_cleaned.rds)o" -clean- (.rds). Utiliza este archivo para todas las demás páginas de este manual que utilizan el listado de casos. Un archivo .rds es un tipo de archivo específico de R que conserva los tipos de columnas. Esto asegura que sólo tendrás que hacer una limpieza mínima después de importar los datos a R.

Otros archivos relacionados:

* [Clica para descargar](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/case_linelists/linelist_cleaned.xlsx) [el](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/case_linelists/linelist_raw.xlsx) listado de casos “en [limpi](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/case_linelists/linelist_cleaned.rds)o" -clean- [como archivo Excel](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/case_linelists/linelist_cleaned.xlsx)
* Parte de la página de limpieza utiliza un "diccionario de limpieza" (archivo .csv). Puedes cargarlo directamente en R ejecutando los siguientes comandos:

#### Recuento de datos de malaria

Estos datos son recuentos ficticios de casos de malaria por grupos de edad, centro y día. Un archivo .rds es un tipo de archivo específico de R que conserva los tipos de columnas. Esto asegura que sólo tendrás que hacer una limpieza mínima después de importar los datos a R.

[Clica para descargar los datos del recuento de casos de malaria (archivo .rds)](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/malaria_facility_count_data.rds)

#### Datos en escala Likert

Se trata de datos ficticios de una encuesta tipo Likert, utilizados en la página sobre [Pirámides demográficas y escalas de Likert](#demographic-pyramids-and-likert-scales). Puedes cargar estos datos directamente en R ejecutando los siguientes comandos:

#### Flexdashboard

A continuación se encuentran los enlaces al archivo asociado a la página sobre [Dashboards con R Markdown](#dashboards-with-r-markdown):

* Para descargar el código de R Markdown para el panel de control del brote, clica con el botón derecho en este [enlace](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/flexdashboard/outbreak_dashboard.Rmd) (Cmd+clic para Mac) y luego "Guardar enlace como".
* Para descargar el código HTML del panel de control, clica con el botón derecho en este [enlace](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/flexdashboard/outbreak_dashboard_test.html) (Cmd+clic para Mac) y luego "Guardar enlace como".

#### Rastreo de contactos

La página de [rastreo de contactos](#contact-tracing-1) muestra el análisis de los datos de rastreo de contactos, utilizando como ejemplo datos de [Go.Data](https://github.com/WorldHealthOrganization/godata/tree/master/analytics/r-reporting). Los datos utilizados en la página pueden descargarse como archivos .rds clicando en los siguientes enlaces:

[Clica para descargar los datos de la investigación de casos (archivo .rds)](https://github.com/WorldHealthOrganization/godata/blob/master/analytics/r-reporting/data/cases_clean.rds?raw=true)

[Clica para descargar los datos de registro de los contactos (archivo .rds)](https://github.com/WorldHealthOrganization/godata/blob/master/analytics/r-reporting/data/contacts_clean.rds?raw=true)

[Clica para descargar los datos de seguimiento de los contactos (archivo .rds)](https://github.com/WorldHealthOrganization/godata/blob/master/analytics/r-reporting/data/followups_clean.rds?raw=true)

**NOTA: Los** datos de rastreo de contactos estructurados de otro software (por ejemplo, KoBo, DHIS2 Tracker, CommCare) pueden tener un aspecto diferente. Si quieres contribuir con una muestra de datos alternativos o con contenido para esta página, [por favor, ponte en contacto, con nosotros](#contact_us).

**CONSEJO:** Si estás utilizando Go.Data y quiere conectarte a su API, consulta la página de importación y exportación [(sección API)](#import_api) y la [Comunidad de Prácticas de Go.Data](https://community-godata.who.int/).

#### SIG (GIS)

Los archivos geográficos (Shapefiles) tienen varios archivos subcomponentes, cada uno con una extensión de archivo diferente. Un archivo tendrá la extensión ".shp", pero otros tienen la extensión ".dbf", ".prj", etc.

La página de [fundamentos del SIG](#gis-basics) contiene enlaces al sitio web de Humanitarian Data Exchange, donde se pueden descargar los shapefiles directamente como archivos comprimidos.

Por ejemplo, los datos de los puntos de las instalaciones sanitarias se pueden descargar [aquí](https://data.humdata.org/dataset/hotosm_sierra_leone_health_facilities). Descarga "hotosm\_sierra\_leone\_health\_facilities\_points\_shp.zip". Una vez guardado en tu ordenador, "descomprime" la carpeta. Hay varios archivos con diferentes extensiones (por ejemplo, ".shp", ".prj", ".shx") todos ellos deben guardarse en la misma carpeta. A continuación, para importar en R, proporciona la ruta del archivo y el nombre del archivo ".shp" a st\_read() del paquete **sf** (como se describe en la página de [fundamentos del SIG](#gis-basics)).

Si sigues la opción 1 para descargar todos los datos de ejemplo (a través de nuestro paquete **epirhandbook**), todos los shapefiles están incluidos en el paquete.

También puedes descargar los shapefiles de la carpeta "data" de R Handbook Github (véase la subcarpeta "gis"). Sin embargo, ten en cuenta que tendrás que descargar cada subfichero individualmente en tu ordenador. En Github, clica en cada archivo individualmente y descárgalo clicando en el botón "Download". A continuación, puedes ver cómo el shapefile "sle\_adm3" consta de muchos archivos, cada uno de los cuales tendría que ser descargado de Github.

#### Árboles filogenéticos

Mira la página sobre [árboles filogenéticos](#phylogenetic-trees-1). El archivo Newick con el árbol filogenético construido a partir de la secuenciación del genoma completo de 299 muestras de Shigella sonnei y los datos de las muestras correspondientes (convertidos en un archivo de texto). Las muestras belgas y los datos resultantes han sido proporcionados amablemente por el NRC belga para Salmonella y Shigella en el marco de un proyecto dirigido por un fellow del programa ECDC EUPHEM, y también se publicarán en un manuscrito. Los datos internacionales están disponibles en bases de datos públicas (ncbi) y han sido publicados previamente.

* Para descargar el archivo del árbol filogenético "Shigella\_tree.txt", clica con el botón derecho en este [enlace](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/phylo/Shigella_tree.txt) (Cmd+clic para Mac) y selecciona "Guardar enlace como".
* Para descargar el archivo "sample\_data\_Shigella\_tree.csv" con información adicional sobre cada muestra, clica con el botón derecho en este [enlace](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/phylo/sample_data_Shigella_tree.csv) (Cmd+clic para Mac) y selecciona "Guardar enlace como".
* Para ver el nuevo árbol de subconjuntos creado, clica con el botón derecho en este [enlace](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/phylo/Shigella_subtree_2.txt) (Cmd+clic para Mac) y selecciona "Guardar enlace como". El archivo .txt se descargará en tu ordenador.

Tras la descarga, se pueden importar los archivos .txt con la función read.tree() del paquete **ape**, como se explica en la página.

#### **Estandari**zación

Consulta la página de [tasas estandarizadas](#standardised-rates). Puedes cargar los datos directamente desde nuestro repositorio de Github en Internet en tu sesión de R con los siguientes comandos:

#### Series temporales y detección de brotes

Véase la página sobre [series temporales y detección de brotes](#time-series-and-outbreak-detection). Utilizamos los casos de campylobacter notificados en Alemania entre 2002 y 2011, disponibles en el paquete R **surveillance**. (nb. este conjunto de datos ha sido adaptado del original, en el sentido de que se han eliminado 3 meses de datos de finales de 2011 para fines de demostración).

[Clica para descargar Campylobacter en Alemania (.xlsx)](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/time_series/campylobacter_germany.xlsx)

También utilizamos datos climáticos de Alemania de 2002 a 2011 (temperatura en grados centígrados y lluvia caida en milímetros) . Estos datos se descargaron de los datos del reanálisis por satélite Copernicus de la UE utilizando el paquete **ecmwfr**. Tendrá que descargarlos todos e importarlos con stars::read\_stars() como se explica en la página de series temporales.

[Clica para descargar el tiempo de Alemania 2002 (archivo .nc)](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/time_series/weather/germany_weather2002.nc)

[Clica para descargar el tiempo de Alemania 2003 (archivo .nc)](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/time_series/weather/germany_weather2003.nc)

[Clica para descargar el tiempo en Alemania 2004 (archivo .nc)](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/time_series/weather/germany_weather2004.nc)

[Clica para descargar el tiempo en Alemania 2005 (archivo .nc)](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/time_series/weather/germany_weather2005.nc)

[Clica para descargar el tiempo en Alemania 2006 (archivo .nc)](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/time_series/weather/germany_weather2006.nc)

[Clica para descargar el tiempo de Alemania 2007 (archivo .nc)](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/time_series/weather/germany_weather2007.nc)

[Clica para descargar el tiempo de Alemania 2008 (archivo .nc)](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/time_series/weather/germany_weather2008.nc)

[Clica para descargar el tiempo en Alemania 2009 (archivo .nc)](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/time_series/weather/germany_weather2009.nc)

[Clica para descargar el tiempo en Alemania 2010 (archivo .nc)](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/time_series/weather/germany_weather2010.nc)

[Clica para descargar el tiempo en Alemania 2011 (archivo .nc)](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/time_series/weather/germany_weather2011.nc)

#### Análisis de encuestas

Para el capítulo sobre [análisis de encuestas](https://epirhandbook.com/survey-analysis.html) utilizamos datos ficticios de encuestas de mortalidad basados en las plantillas de encuestas de MSF OCA. Estos datos ficticios se generaron como parte del [proyecto "R4Epis"](https://r4epis.netlify.app/).

[Clica para descargar los datos de la encuesta ficticia (.xlsx)](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/surveys/survey_data.xlsx)

[Clica para descargar el diccionario de datos de la encuesta ficticia (.xlsx)](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/surveys/survey_dict.xlsx)

[Clica para descargar los datos de la población de la encuesta ficticia (.xlsx)](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/surveys/population.xlsx)

#### Shiny

El capítulo sobre [Dashboards con Shiny](#dashboards-with-shiny) demuestra la construcción de una sencilla aplicación para mostrar datos sobre la malaria.

Para descargar los archivos R que producen la aplicación Shiny:

Puedes [clicar aquí para descargar el archivo app.R que contiene tanto la interfaz de usuario como el código del servidor para la aplicación Shiny.](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/malaria_app/app.R)

Puedes [clicar aquí para descargar el archivo facility\_count\_data.rds](https://github.com/epirhandbook/Epi_R_handbook/blob/master/data/malaria_app/data/facility_count_data.rds) que contiene datos sobre la malaria para la aplicación Shiny. Ten en cuenta que puede ser necesario almacenarlo dentro de una carpeta "data" para que las rutas de los archivos here() funcionen correctamente.

Puedes [clicar aquí para descargar el archivo global.R](https://github.com/epirhandbook/Epi_R_handbook/blob/master/data/malaria_app/global.R) que debe ejecutarse antes de que se abra la aplicación, como se explica en dicho capítulo.

Puedes [clicar aquí para descargar el archivo plot\_epicurve.R](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/malaria_app/funcs/plot_epicurve.R) que es originado por global.R. Ten en cuenta que puede necesitar almacenarlo dentro de una carpeta "funcs" para que las rutas de los archivos here() funcionen correctamente.

&&&&&

# Conocimientos básicos

# #Aspectos básicos de R

{#r-basics}

Bienvenido.

Esta página repasa los aspectos esenciales de R. No pretende ser un tutorial exhaustivo, pero proporciona los fundamentos y puede ser útil para refrescar la memoria. La sección de [Recursos para el aprendizaje](#learning) enlaza con tutoriales más completos.

Partes de esta página han sido adaptadas con permiso del [proyecto R4Epis](https://r4epis.netlify.app/).

Consulta la página sobre [Transición a R](#transition-to-r) para obtener consejos sobre cómo cambiar a R desde STATA, SAS o Excel.

## ¿Por qué utilizar R?

Como se indica en el [sitio web del proyecto](https://www.r-project.org/about.html) R, éste es un lenguaje de programación y un entorno para la computación estadística y gráficos. Es muy versátil, ampliable y dirigido por la comunidad.

**Coste**

El uso de R es gratuito. Hay una fuerte ética en la comunidad de material libre y de código abierto.

**Reproducibilidad**

La gestión y el análisis de los datos a través de un lenguaje de programación (en comparación con Excel u otra herramienta principalmente manual) mejora **la reproducibilidad**, facilita la **detección de errores** y alivia la carga de trabajo.

**Comunidad**

La comunidad de usuarios de R es enorme y colaborativa. Cada día se desarrollan nuevos paquetes y herramientas para abordar problemas cotidianos, que son examinados por la comunidad de usuarios. Por ejemplo, [R-Ladies](https://rladies.org/) es una asociación mundial cuya misión es promover la diversidad de género en la comunidad de R, siendo una de las mayores asociaciones de usuarios de R. Es probable que tengas un grupo cerca.

## Términos clave

**RStudio** - RStudio es una interfaz gráfica de usuario (GUI) para facilitar el uso de **R.** Lea más [en la sección RStudio](#rstudio).

**Objetos** - Todo lo que se almacena en R - conjuntos de datos, variables, una lista de nombres de pueblos, un número total de población, incluso resultados como gráficos - son objetos a los que se les asigna un nombre y pueden ser referenciados en comandos posteriores. Lea más [en la sección Objetos](#objects).

**Funciones** - Una función es una operación de código que acepta entradas y devuelve una salida transformada. Lea más [en la sección Funciones](#functions).

**Paquetes** - Un paquete de R es un conjunto de funciones que se pueden compartir. Lea más [en la sección Paquetes](#packages).

**Scripts** - Un script es el archivo de documento que contiene tus comandos. Lee más [en la sección de Scripts](#scripts)

## Recursos para el aprendizaje

### Recursos en RStudio

**Documentación de ayuda**

Busque en la pestaña "Ayuda" de RStudio la documentación sobre los paquetes de R y funciones específicas. Esto está dentro del panel que también contiene Archivos, Gráficos y Paquetes (normalmente en el panel inferior derecho). También puede escribir el nombre de un paquete o función en la consola de R después de un signo de interrogación para abrir la página de ayuda correspondiente. No incluya paréntesis.

Por ejemplo: ?filter o ?diagrammeR.

**Tutoriales interactivos**

Hay varias formas de aprender R de forma interactiva dentro de RStudio.

El propio RStudio ofrece un Tutorial que se encuentra en el paquete de R [**learnr**](https://blog.rstudio.com/2020/02/25/rstudio-1-3-integrated-tutorials/). Simplemente instale este paquete y abra un tutorial a través de la nueva pestaña "Tutorial" en el panel superior derecho de RStudio (que también contiene las pestañas Entorno e Historia).

El paquete de R [**swirl**](https://swirlstats.com/) ofrece cursos interactivos en la consola de R. Instale y cargue este paquete, luego ejecuta el comando swirl() (paréntesis vacío) en la consola de R. Verá que aparecen indicaciones en la consola. Responda escribiendo en la consola. Le guiará a través de un curso de tu elección.

### Hojas de referencia (Cheatsheets)

Hay muchas "hojas de referencias o trucos" (Cheatsheets) en PDF disponibles en el [sitio web de RStudio](https://rstudio.com/resources/cheatsheets/), por ejemplo:

* Factores con el paquete **forcats**
* Fechas y horarios con el paquete **lubridate**
* Cadenas con el paquete **stringr**
* Operaciones iterativas con el paquete **purrr**
* Importación de datos
* Transformación de datos con el paquete **dplyr**
* **R Markdown** (para crear documentos como PDF, Word, Powerpoint...)
* **Shiny** (para crear aplicaciones web interactivas)
* Visualización de datos con el paquete **ggplot2**
* Cartografía (GIS)
* Mapas interactivos con el paquete leaflet
* Python con R (paquete **reticulate**)

En este enlace puedes encontrar un recurso en línea, específicamente para [los usuarios de Excel](https://jules32.github.io/r-for-excel-users/)

### Twitter

R tiene una vibrante comunidad en Twitter en la que puedes aprender trucos, atajos y noticias: sigue estas cuentas:

* Síguenos! [@epiRhandbook](https://twitter.com/epirhandbook)
* R Function A Day [@rfuntionaday](https://twitter.com/rfunctionaday) (Es un recurso increíble)
* R para ciencia de datos [@rstats4ds](https://twitter.com/rstats4ds?lang=en)
* RStudio [@RStudio](https://twitter.com/rstudio?lang=en)
* Trucos de RStudio [@rstudiotips](https://twitter.com/rstudiotips)
* R-Bloggers [@Rbloggers](https://twitter.com/Rbloggers)
* R-ladies [@RLadiesGlobal](https://twitter.com/RLadiesGlobal)
* Hadley Wickham [@hadleywickham](https://twitter.com/hadleywickham?ref_src=twsrc^google|twcamp^serp|twgr^author)

También:

**#epitwitter** y **#rstats**

### Recursos gratuitos en línea

Un texto definitivo es el libro [R for Data Science](https://r4ds.had.co.nz/) de Garrett Grolemund y Hadley Wickham

El sitio web del proyecto [R4Epis](https://r4epis.netlify.app/) tiene como objetivo "desarrollar herramientas estandarizadas de limpieza de datos, análisis y elaboración de informes para cubrir los tipos comunes de brotes y estudios realizados en la población en un entorno de respuesta de emergencia de MSF". Se pueden encontrar materiales de formación sobre los fundamentos de R, plantillas para informes de RMarkdown sobre brotes y encuestas, y tutoriales para ayudar a configurarlos.

### **Idiomas** distintos del inglés

[Material de RStudio en Español](https://www.rstudio.com/collections/espanol/)

[Introduction à R et au tidyverse (francés)](https://juba.github.io/tidyverse/index.html)

## Instalación

### R y RStudio

**Cómo instalar R**

Visita este sitio web <https://www.r-project.org/>y descarga la última versión de R adecuada a tu ordenador.

**Cómo instalar RStudio**

Visita este sitio web <https://rstudio.com/products/rstudio/download/>y descarga la última versión gratuita de RStudio para escritorio adecuada para tu ordenador.

**Permisos**

**Ten en cuenta** que debes instalar R y RStudio en una unidad donde tengas permisos de lectura y escritura. De lo contrario, la capacidad para instalar paquetes de R (algo frecuente) se verá afectada. Si tienes problemas, intenta abrir RStudio con el botón derecho en el icono y seleccionando "Ejecutar como administrador". Puedes encontrar otros consejos en la página [R en unidades de red](#r-on-network-drives).

**Cómo actualizar R y RStudio**

Tu versión de R se muestra al inicio de la consola de R. También puede ejecutar sessionInfo().

Para actualizar R, puedes ir al sitio web mencionado anteriormente y vuelva a instalar R. También puede utilizar el paquete **installr** (en Windows) ejecutando installr::updateR(). Esto abrirá cuadros de diálogo para ayudarle a descargar la última versión de R y actualizar sus paquetes a la nueva versión de R. Puedes encontrar más detalles en la [documentación de](https://www.r-project.org/nosvn/pandoc/installr.html) **installr**.

Ten en cuenta que la versión antigua de R seguirá existiendo en tu ordenador. Puedes ejecutar temporalmente una versión anterior (una "instalación" más antigua) de R clicando en "Herramientas" -> "Opciones globales" en RStudio y eligiendo una versión de R. Esto puede ser útil si quieres utilizar un paquete que no ha sido actualizado para funcionar en la versión más reciente de R.

Para actualizar RStudio, puede ir a la página web anterior y volver a descargar RStudio. Otra opción es clicar en "Ayuda" -> "Buscar actualizaciones" dentro de RStudio, pero esto puede no mostrar las últimas actualizaciones.

Para ver qué versiones de R, RStudio o paquetes se utilizaron cuando se hizo este Manual, consulte la página de [Notas técnicas y editoriales](#editorial-and-technical-notes).

### Otros programas que puedes necesitar instalar

* TinyTeX (para compilar un documento RMarkdown en PDF)
* Pandoc (para compilar documentos RMarkdown)
* RTools (para construir paquetes para R)
* phantomjs (para guardar imágenes fijas de redes animadas, como cadenas de transmisión)

#### TinyTex

TinyTex es una distribución LaTeX personalizada, útil cuando se trata de producir PDFs desde R.   
Ver <https://yihui.org/tinytex/>para más información.

Para instalar TinyTex desde R:

#### Pandoc

Pandoc es un conversor de documentos, un software separado de R. **Viene incluido con RStudio y no debería ser necesario descargarlo.** Ayuda en el proceso de conversión de documentos Rmarkdown a formatos como el .pdf y añade funcionalidades complejas.

#### RTools

RTools es una colección de software para construir paquetes para R

Se instala desde este sitio web: <https://cran.r-project.org/bin/windows/Rtools/>

#### phantomjs

Esto se utiliza a menudo para hacer "capturas de pantalla" de las páginas web. Por ejemplo, cuando se hace una cadena de transmisión con el paquete **epicontacts**, se produce un archivo HTML que es interactivo y dinámico. Si desea una imagen estática, puede ser útil utilizar el paquete [**webshot**](https://wch.github.io/webshot/articles/intro.html) para automatizar este proceso. Para ello se necesita el programa externo "phantomjs". Puedes instalar phantomjs a través del paquete **webshot** con el comando webshot::install\_phantomjs().

## RStudio

### Orientación de RStudio

**Primero, abre RStudio.** Como sus iconos pueden ser muy similares, asegúrate de que estás abriendo RStudio y no R.

Para que RStudio funcione, también debes tener instalado R en el ordenador (consulte las instrucciones de instalación más arriba).

**RStudio** es una interfaz (GUI) para facilitar el uso de **R**. Puedes pensar que R es el motor de un vehículo, que hace el trabajo crucial, y RStudio es la carrocería del vehículo (con asientos, accesorios, etc.) que te ayuda a usar el motor para avanzar. Puedes ver la hoja de trucos completa de la interfaz de usuario de RStudio (PDF) [aquí](https://www.rstudio.com/wp-content/uploads/2016/01/rstudio-IDE-cheatsheet.pdf)

Por defecto, RStudio muestra cuatro paneles rectangulares.

**CONSEJO:** Si tu RStudio sólo muestra un panel izquierdo es porque aún no tiene ningún script abierto.

**El panel **Source****

**Este** panel de código Fuente u origen, por defecto en la parte superior izquierda, es un espacio para editar, ejecutar y guardar tus [scripts](#scripts). Los scripts contienen los comandos que desea ejecutar. Este panel también puede mostrar conjuntos de datos (data frames) para su visualización.

Para los usuarios de Stata, este panel es similar a las ventanas de Do-file y del Editor de Datos.

**El panel **Console****

La consola de R, por defecto el panel izquierdo o inferior izquierdo en R Studio, es el hogar del "motor" de R. Aquí es donde se ejecutan realmente los comandos y aparecen las salidas no gráficas y los mensajes de error/advertencia. Puedes introducir y ejecutar directamente comandos en la Consola de R, pero ten en cuenta que estos comandos no se guardan como cuando se ejecutan comandos desde un script.

Si está familiarizado con Stata, la consola de R es como la ventana de comandos y también la ventana de resultados.

**El panel Environment**

**Este** panel de Entorno, por defecto en la parte superior derecha, se utiliza más a menudo para ver breves resúmenes de los [objetos](#objects) en el Entorno R en la sesión actual. Estos objetos pueden incluir conjuntos de datos importados, modificados o creados, parámetros que haya definido (por ejemplo, una semana epi específica para el análisis), o vectores o listas que haya definido durante el análisis (por ejemplo, nombres de regiones). Puedes clicar en la flecha situada junto al nombre de un dataframe para ver sus variables.

En Stata, esto es muy similar a la ventana del Gestor de Variables.

Este panel también contiene el Historial donde puede ver los comandos ejecutados anteriormente. También tiene una pestaña "Tutorial" donde puedes completar tutoriales interactivos de R si tienes el paquete **learnr** instalado. También tiene una pestaña de "Conexiones" para las conexiones externas, y puede tener un panel "Git" si decide interactuar con Github.

**Panel Files, Plots, Packages, Help, Viewer**  
Este panel inferior derecho incluye varias pestañas importantes. La pestaña Files (Archivos) permite navegar por las carpetas y puede utilizarse para abrir o eliminar archivos. En la pestaña Plots (Gráficos), se mostrarán todos los gráficos, incluyendo los mapas. Las salidas interactivas o HTML se mostrarán en la pestaña Viewer (Visor). El panel Packages (Paquetes) permite ver, instalar, actualizar, eliminar, cargar/descargar paquetes de R y ver qué versión del paquete tiene. En la [sección de paquetes](#packages) más abajo se puede aprender más sobre los paquetes. Por último, en el panel de Ayuda (Help) se mostrará la documentación y los archivos de ayuda.

Este panel contiene los equivalentes en Stata de las ventanas Plots Manager y Project Manager.

### Configuración de RStudio

Cambie la configuración y la apariencia de RStudio en el menú desplegable Tools (Herramientas), seleccionando Global Options (Opciones globales). Allí puede cambiar la configuración por defecto, incluyendo la apariencia/color de fondo.

**Reiniciar**

Si se cuelga R, se puede reiniciar yendo al menú Sesión y clicando en "Restart R” (Reiniciar R)". Esto evita la molestia de cerrar y abrir RStudio. Al hacer esto, se eliminará todo el entorno de esa sesión de R.

### Atajos de teclado

Algunos atajos de teclado muy útiles están abajo. Se pueden ver todos los atajos de teclado para Windows, Max y Linux en la segunda página de esta [hoja de trucos de la interfaz de usuario](https://www.rstudio.com/wp-content/uploads/2016/01/rstudio-IDE-cheatsheet.pdf) de RStudio.

| **Windows/Linux** | **Mac** | **Acción** |
| --- | --- | --- |
| Esc | Esc | Interrumpir el comando actual (útil si accidentalmente ejecutó un comando incompleto y no puede evitar ver "+" en la consola de R) |
| Ctrl+s | Cmd+s | Guardar (script) |
| Tab | Tab | Autocompletar |
| Ctrl + Enter | Cmd + Enter | Ejecutar la(s) línea(s) de código actual(es) |
| Ctrl + Shift + C | Cmd + Shift + c | Comentar/descomentar las líneas resaltadas |
| Alt + - | Opción + - | Insertar <- |
| Ctrl + Shift + m | Cmd + Shift + m | Insertar %>% |
| Ctrl + l | Cmd + l | Limpiar la consola de R |
| Ctrl + Alt + b | Cmd + Opción + b | Ejecutar desde el inicio hasta la línea actual |
| Ctrl + Alt + t | Cmd + Opción + t | Ejecutar la sección de código actual (R Markdown) |
| Ctrl + Alt + i | Cmd + Shift + r | Insertar un trozo (chunk) de código (en R Markdown) |
| Ctrl + Alt + c | Cmd + Opción + c | Ejecutar el código chunk actual (R Markdown) |
| flechas arriba/abajo en la consola R | En el mismo | Recorrer los comandos ejecutados recientemente |
| Shift + flechas arriba/abajo en el script | En el mismo | Seleccionar varias líneas de código |
| Ctrl + f | Cmd + f | Buscar y reemplazar en el script actual |
| Ctrl + Shift + f | Cmd + Shift + f | Buscar en archivos (buscar/reemplazar en muchos scripts) |
| Alt + l | Cmd + Opción + l | Plegar el código seleccionado |
| Shift + Alt + l | Cmd + Shift + Opción+l | Desplegar el código seleccionado |

**CONSEJO:** Utiliza la tecla Tab cuando escribas para activar la función de autocompletar de RStudio. Esto puede evitar errores de ortografía. Pulsa el tabulador mientras escribes para que aparezca un menú desplegable de posibles funciones y objetos, basándose en lo que escrito hasta ese momento.

## Funciones

Las funciones son la pieza principal en el uso de R. Las funciones son la forma de realizar tareas y operaciones. Muchas vienen instaladas con R, mientras muchas otras están disponibles para su descarga en paquetes (explicados en la sección de [paquetes](#packages)), ¡E incluso, puedes escribir tus propias funciones personalizadas.

Esta sección básica sobre las funciones explica:

* Qué es una función y cómo funciona
* Qué son los argumentos de una función
* Cómo obtener ayuda para entender una función

Una nota rápida sobre la sintaxis: En este manual, las funciones se escriben en código-texto con paréntesis abiertos, así: filter(). Como se explica en la sección de [paquetes](#packages), las funciones se descargan dentro de los paquetes. En este manual, los nombres de los paquetes se escriben en **negrita**, como **dplyr**. A veces en el código de ejemplo puede ver el nombre de la función vinculado explícitamente al nombre de su paquete con dos dos puntos (::) como: dplyr::filter(). El propósito de esta vinculación se explica en la sección de paquetes.

### Funciones simples

**Una función es como una máquina que recibe entradas, realiza alguna acción con esas entradas y produce una salida.** El resultado depende de la función.

**Las funciones suelen operar sobre algún objeto colocado dentro de los paréntesis de la función**. Por ejemplo, la función sqrt() calcula la raíz cuadrada de un número:

El objeto proporcionado a una función también puede ser una columna de unos datos (véase la sección [Objetos](#objects) para conocer todos los tipos de objetos). Dado que R puede almacenar múltiples conjuntos de datos, tendrá que especificar tanto el set de datos como la columna. Una forma de hacerlo es utilizar la notación $ para vincular el nombre de los datos y el nombre de la columna (dataset$column). En el siguiente ejemplo, la función summary() se aplica a la columna numérica age en los datos linelist, y la salida es un resumen de los valores numéricos y faltantes de la columna.

**NOTA:** Entre bastidores, una función representa un complejo código adicional que ha sido envuelto para el usuario en un comando sencillo.

### Funciones con múltiples argumentos

Las funciones suelen pedir varias entradas, llamadas **argumentos**, situadas dentro del paréntesis de la función, normalmente separadas por comas.

* Algunos argumentos son necesarios para que la función funcione correctamente, mientras otros son opcionales
* Los argumentos opcionales tienen una configuración por defecto
* Los argumentos pueden tomar caracteres, números, lógica (TRUE/FALSE) y otras entradas

He aquí una divertida función ficticia, llamada oven\_bake(), como ejemplo de una función típica (hacer en el horno). Toma un objeto de entrada (“input”) (por ejemplo, una base de datos, en este ejemplo "masa") y realiza operaciones en él según lo especificado por los argumentos adicionales (minutos = y temperatura =). La salida (“output”) puede imprimirse en la consola, o guardarse como un objeto utilizando el operador de asignación <-.

**En un ejemplo más realista**, el comando age\_pyramid() que aparece a continuación produce un gráfico de pirámide de edad basado en grupos de edad definidos y una columna de división binaria, como el género. La función recibe tres argumentos dentro de los paréntesis, separados por comas. Los valores suministrados a los argumentos establecen linelist como los datos (dataframe) a utilizar, age\_cat5 como la columna a contar, y gender como la columna binaria a utilizar para dividir la pirámide por color según género.

El comando anterior puede escribirse de forma equivalente a la de más abajo, con un estilo más largo con una nueva línea para cada argumento. Este estilo puede ser más fácil de leer, y más fácil de escribir "comentarios" con # para explicar cada segmento de código (¡comentar en el código es considerado una buena práctica!). Para ejecutar este comando más largo puedes seleccionar todo el texto y clicar en "Ejecutar", o simplemente colocar el cursor en la primera línea y luego clicar las teclas Ctrl y Enter simultáneamente.

No es necesario especificar la primera mitad de una asignación de argumentos (por ejemplo, data =) si los argumentos se escriben en su orden específico (especificado en la documentación de la función). El código siguiente produce exactamente la misma pirámide que la anterior, porque la función espera ese orden de los argumentos: conjunto de datos, la variable de age\_group, y la variable de split\_by .

€€€hasta aqui

**Un comando age\_pyramid() más complejo podría incluir los argumentos opcionales para:**

* Mostrar proporciones en lugar de recuentos (estableciendo proporcional = TRUE cuando el valor por defecto es FALSE)
* Especificar los dos colores a utilizar (pal = es la abreviatura de "paleta" y se suministra con un vector de dos nombres de color. Para saber cómo se hace un vector con la función c() puedes consultar la página de [objetos](#objectstructure) .

**NOTA:** En los argumentos en los que se especifican ambas partes del argumento (por ejemplo, proporcional = TRUE), no importa el orden de estos argumentos.

### Escribir funciones

R es un lenguaje orientado a las funciones, por lo que debería sentirse capacitado para escribir tus propias funciones. La creación de funciones aporta varias ventajas:

* Facilitar la programación modular, es decir, la separación del código en partes independientes y manejables.
* Sustituye el repetitivo copiar y pegar, que puede dar lugar a errores
* Dar a las piezas de código nombres fáciles de recordar

En la página [Escribir funciones](#writing-functions-1) se trata en profundidad cómo escribir funciones.

## Paquetes

**Los paquetes contienen funciones.**

Un paquete de R es un conjunto de código y documentación que se puede compartir y que contiene funciones predefinidas. Los usuarios de la comunidad R desarrollan paquetes todo el tiempo atendiendo a problemas específicos, ¡es probable que alguno pueda ayudarte en tu trabajo! En tu uso de R instalarás y utilizarás cientos de paquetes.

En la instalación, R contiene paquetes y funciones **"base"** que realizan tareas elementales comunes. Pero muchos usuarios de R crean funciones especializadas, que son verificadas por la comunidad de R y que puedes descargar como **paquete** para tu propio uso. En este manual, los nombres de los paquetes se escriben en **negrita**. Uno de los aspectos más desafiantes de R es que a menudo hay muchas funciones o paquetes donde elegir para una tarea determinada.

### Instalar y cargar

Las funciones están contenidas en **paquetes** que pueden descargarse ("instalarse") en tu ordenador desde Internet. Una vez descargado un paquete, se almacena en tu "biblioteca". Puedes acceder a las funciones que contiene durante una sesión de R "cargando" el paquete.

Piensa en R como tu biblioteca personal: Cuando se descarga un paquete, tu biblioteca adquiere un nuevo libro de funciones, pero cada vez que quieras utilizar una función de ese libro, debes tomar prestado ("cargar") ese libro de tu biblioteca.

En resumen: para utilizar las funciones disponibles en un paquete de R, hay que realizar dos pasos:

1. El paquete debe ser **instalado** (una vez), y
2. El paquete debe ser **cargado** (cada sesión de R)

#### Tu biblioteca

Tu "biblioteca" es en realidad una carpeta en tu ordenador, que contiene una carpeta para cada paquete que se ha instalado. Averigua dónde está instalado R en tu ordenador, y busca una carpeta llamada "win-library". Por ejemplo: R\win-library\4.0 (la 4.0 es la versión de R - tendrá una biblioteca diferente para cada versión de R que haya descargado).

Puedes imprimir la ruta del archivo de tu biblioteca introduciendo .libPaths() (paréntesis vacíos). Esto resulta especialmente importante si se trabaja con [R en unidades de red](#r-on-network-drives).

#### Instalar desde CRAN

Lo más habitual es que los usuarios de R descarguen paquetes de CRAN. CRAN (Comprehensive R Archive Network) es un almacén público online de paquetes de R que han sido publicados por los miembros de la comunidad R.

¿Te preocupan los virus y la seguridad al descargar un paquete de CRAN? Lee [este artículo](https://support.rstudio.com/hc/en-us/articles/360042593974-R-and-R-Package-Security) sobre el tema.

#### Cómo instalar y cargar

En este manual, sugerimos utilizar el paquete **pacman** (abreviatura de "packages manager"). Ofrece una interesante función p\_load() que instalará un paquete si es necesario y lo cargará para su uso en la sesión actual de R.

La sintaxis es bastante sencilla. Sólo hay que listar los nombres de los paquetes dentro de los paréntesis de p\_load(), separados por comas. Este comando instalará los paquetes **rio**, **tidyverse** y **here** si aún no están instalados, y los cargará para su uso. Esto hace que el enfoque de p\_load() sea conveniente y conciso si se comparten scripts con otros. Ten en cuenta que los nombres de los paquetes distinguen entre mayúsculas y minúsculas.

Fíjate que hemos utilizado la sintaxis pacman::p\_load() que escribe explícitamente el nombre del paquete (**pacman**) antes del nombre de la función (p\_load()), conectado por dos dos puntos ::. Esta sintaxis es útil porque también carga el paquete **pacman** (suponiendo que ya esté instalado).

Hay funciones **base de R** alternativas que verás a menudo. La función base **de R** para instalar un paquete es install.packages(). El nombre del paquete a instalar debe proporcionarse entre paréntesis y entre comillas. Si desea instalar varios paquetes en un solo comando, deben ser listados dentro de un vector de caracteres c().

Nota: este comando instala un paquete, pero no lo carga para utilizarlo en la sesión actual.

La instalación también se puede realizar clicando en el panel "Paquetes" de RStudio, luego en "Instalar" y buscando el nombre del paquete deseado.

La función alternativa en R **base** para **cargar** un paquete (después de haberlo instalado) es library(). Sólo puedes cargar un paquete a la vez (otra razón para usar p\_load()). Se puede escribir el nombre del paquete con o sin comillas.

Para comprobar si un paquete está instalado y/o cargado, puedes mirar en la pestaña de paquetes de RStudio. Si el paquete está instalado, se muestra allí con el número de versión. Si su casilla está marcada, está cargado para la sesión actual.

**Instalar desde Github**

A veces, necesitas instalar un paquete que aún no está disponible en CRAN. O tal vez el paquete está disponible en CRAN pero quieres la versión de desarrollo con nuevas características que aún no se ofrecen en la versión más estable publicada en CRAN. Éstas suelen estar alojadas en el sitio web [github.com](https://github.com/) en un "repositorio" de código gratuito y de acceso público. Lee más sobre Github en la página del manual sobre [Control de versiones y colaboración con Git y Github](#version-control-and-collaboration-with-).

Para descargar los paquetes de R desde Github, puedes utilizar la función p\_load\_gh() de **pacman**, que instalará el paquete si es necesario, y lo cargará para utilizarlo en tu sesión actual de R. Las alternativas de instalación incluyen el uso de los paquetes **remotes** o **devtools**. Puedes leer más sobre todas las funciones de **pacman** en la [documentación del paquete](https://cran.r-project.org/web/packages/pacman/pacman.pdf).

Para instalar desde Github, tienes que proporcionar más información. Debe proporcionar:

1. El ID de Github del propietario del repositorio
2. El nombre del repositorio que contiene el paquete
3. (opcional) El nombre de la "rama" (versión de desarrollo específica) que quieras descargar

En los ejemplos siguientes, la primera palabra entre comillas es el ID de Github del propietario del repositorio, después de la barra es el nombre del repositorio (el nombre del paquete).

Si quieres instalar desde una "rama" (versión) distinta de la rama principal, añade el nombre de la rama tras una "@", después del nombre del repositorio.

Si no hay diferencia entre la versión de Github y la versión en tu ordenador, no se realizará ninguna acción. Puedes "forzar" una reinstalación usando p\_load\_current\_gh() con el argumento update = TRUE. Puedes leer más sobre **pacman** en esta [viñeta online](http://trinker.github.io/pacman/vignettes/Introduction_to_pacman.html)

**Instalar desde ZIP o TAR**

Puedes instalar el paquete desde una URL:

O bien, descargarlo en tu ordenador en un archivo comprimido:

Opción 1: utilizar install\_local() del paquete **remotes**

Opción 2: utilizando install.packages() desde R **base**, proporcionando la ruta del archivo ZIP y estableciendo type = "source” y repos = NULL.

### Sintaxis del código

Para mayor claridad en este manual, las funciones van a veces precedidas por el nombre de su paquete utilizando el símbolo :: de la siguiente manera: nombre\_del\_paquete::nombre\_de\_la\_función()

Una vez cargado un paquete para una sesión, este estilo explícito no es necesario. Se puede utilizar simplemente nombre\_de\_la\_funcion(). Sin embargo, escribir el nombre del paquete es útil cuando el nombre de una función es común y puede existir en varios paquetes (por ejemplo, plot()). Escribir el nombre del paquete también cargará el paquete si no está todavía cargado.

### Ayuda a la función

Para leer más sobre una función, se puede buscar en la pestaña Ayuda de la parte inferior derecha de RStudio. También se puede ejecutar un comando como ?nombre\_de\_la\_funcion (ponga el nombre de la función después de un signo de interrogación) y aparecerá la página de ayuda en la pestaña de ayuda. Por último, intenta buscar otros recursos en Internet.

### Actualizar paquetes

Puedes actualizar los paquetes reinstalándolos. También puedes clicar en el botón verde "Update" en la pestaña de paquetes de RStudio para ver qué paquetes tienen nuevas versiones para instalar. Ten en cuenta que tu código antiguo puede necesitar ser actualizado si hay una revisión importante en el funcionamiento de una función.

### Eliminar paquetes

Utiliza p\_delete() de **pacman**, o remove.packages() de R **base.** Alternativamente, puedes buscar la carpeta que lo contiene en tu biblioteca y borrarla manualmente.

### Dependencias

Los paquetes a menudo dependen de otros paquetes para funcionar. Estos se llaman dependencias. Si una dependencia no está instalada, entonces el paquete que depende de ella también puede no instalarse.

Se pueden ver las dependencias de un paquete con p\_depends(), y ver qué paquetes dependen de él con p\_depends\_reverse()

### Funciones enmascaradas

No es raro que dos o más paquetes contengan el mismo nombre de función. Por ejemplo, el paquete **dplyr** tiene una función filter(), pero también la tiene el paquete **stats**. La función filter() por defecto depende del orden en que estos paquetes se cargan por primera vez en la sesión de R - el último será el predeterminado para el comando filter().

Puedes comprobar el orden en la Pestaña de Entorno de R Studio - clica en el desplegable de "Global Evironment" (Entorno Global) y ver el orden de los paquetes. Las funciones de los paquetes más bajos en esa lista desplegable enmascararán las funciones del mismo nombre en los paquetes que aparecen más arriba en la lista desplegable. Cuando se carga por primera vez un paquete, R le advertirá en la consola si se está produciendo el enmascaramiento, pero esto puede ser fácil de pasar por alto.

Aquí hay formas de arreglar el enmascaramiento:

1. Especifique el nombre del paquete en el comando. Por ejemplo, utiliza dplyr::filter()
2. Reordene el orden de carga de los paquetes (por ejemplo, dentro de p\_load()), e **iniciar una nueva sesión de R**

### Detach / descargar

Para descargar (detach) un paquete, utiliza este comando, con el nombre correcto del paquete y sólo dos puntos. Ten en cuenta que esto puede no resolver el enmascaramiento.

### Instalar la versión anterior

Consulta esta [guía](https://support.rstudio.com/hc/en-us/articles/219949047-Installing-older-versions-of-packages) para instalar una versión anterior de un paquete concreto.

### Paquetes recomendados

Consulte la página de [Paquetes recomendados](#suggested-packages-1) para obtener una lista de paquetes que recomendamos para la epidemiología del día a día.

## Scripts

Los scripts son una parte fundamental de la programación. Son documentos que contienen comandos (por ejemplo, funciones para crear y modificar conjuntos de datos, imprimir visualizaciones, etc.). Puedes guardar un script y volver a ejecutarlo más tarde. Almacenar y ejecutar tus comandos desde un script tiene muchas ventajas (frente a teclear los comandos uno a uno en la "línea de comandos" de la consola de R):

* Portabilidad: puedes compartir tu trabajo con otros enviándoles tus scripts
* Reproducibilidad: para que tú y los demás sepan exactamente lo que ha hecho
* Control de versiones: para que puedas hacer un seguimiento de los cambios realizados por ti mismo o por tus colegas
* Comentarios/anotaciones: para explicar a tus compañeros lo que has hecho

### Comentarios

En un script también puede anotar ("comentar") alrededor de su código R. Los comentarios son útiles para explicarte a sí mismo y a otros lectores lo que está haciendo. Puedes añadir un comentario escribiendo el símbolo de almohadilla (#) y escribiendo el comentario después de él. El texto comentado aparecerá en un color diferente al del código R.

Cualquier código escrito después de la # no se ejecutará. Por lo tanto, colocar un # antes del código es también una forma útil de bloquear temporalmente una línea de código ("comentar") si no quieres borrarla). Puedes comentar varias líneas a la vez resaltándolas y clicando Ctrl+Mayús+c (Cmd+Mayús+c en Mac).

* Comenta lo que haces y ***por qué*** lo haces.
* Divide tu código en secciones lógicas
* Acompaña tu código con un texto describiendo paso a paso lo que está haciendo (por ejemplo, pasos numerados)

### Estilo

Es importante ser consciente de tu estilo de codificación, especialmente si trabajas en equipo. Nosotros abogamos por la [guía de estilo](https://style.tidyverse.org/) **de tidyverse**. También hay paquetes como **styler** y **lintr** que te ayudan a ajustarte a este estilo.

Algunos puntos muy básicos para que tu código sea legible para los demás:  
\* Al nombrar los objetos, utiliza sólo letras minúsculas, números y barras bajas \_, por ejemplo, mis\_datos\*   
Utiliza espacios frecuentemente, incluso alrededor de los operadores, por ejemplo, n = 1 y edad\_nueva <- edad\_vieja + 3

### **Script** de ejemplo

A continuación se muestra un ejemplo de un breve script de R. ¡Recuerda!, cuanto mejor expliques brevemente el código con los comentarios, ¡más gustará a tus colegas!

### R markdown

Un script de R markdown es un tipo de script de R en el que el propio script se convierte en un documento de salida (PDF, Word, HTML, Powerpoint, etc.). Se trata de herramientas increíblemente útiles y versátiles que suelen utilizarse para crear informes dinámicos y automatizados. ¡Hasta este sitio web y este manual se han hecho con scripts de R markdown!

Vale la pena señalar que los usuarios principiantes de R también pueden utilizar R Markdown - ¡no te dejes intimidar! Para saber más, consulta el capítulo del manual sobre [Informes con R Markdown](#reports-with-r-markdown).

### Cuadernos de R

No hay ninguna diferencia entre escribir en Rmarkdown o en un cuaderno de R. Sin embargo, la ejecución del documento difiere ligeramente. Consulta este [sitio](http://uc-r.github.io/r_notebook) para obtener más detalles.

### Shiny

Las aplicaciones/sitios web de Shiny están contenidos en un script, que debe llamarse app.R. Este archivo tiene tres componentes:

1. Una interfaz de usuario (ui)
2. Una función de servidor
3. Una llamada a la función shinyApp

Consulta la página del manual sobre [Dashboards con Shiny](#dashboards-with-shiny), o este tutorial en línea: [Tutorial de Shiny](https://shiny.rstudio.com/tutorial/written-tutorial/lesson1/)

Hace un tiempo, el archivo anterior se dividía en dos archivos (*ui.R* y *server.R*)

### Código plegado

Puedes contraer porciones de código para facilitar la lectura del script.

Para ello, cree una cabecera de texto con #, escriba tu cabecera y sigue con al menos 4 guiones (-), almohadillas (#) o iguales (=). Cuando hayas hecho esto, aparecerá una pequeña flecha en el "margen" de la izquierda (junto al número de fila). Puedes clicar en esta flecha y en el código de abajo hasta que la siguiente cabecera se pliegue y aparezca un icono de flecha doble en su lugar.

Para expandir el código, clica de nuevo en la flecha del margen o en el icono de la flecha doble. También hay atajos de teclado como se explica en la [sección de RStudio](#rstudio) de esta página.

Al crear cabeceras con #, también activará el índice de contenidos en la parte inferior del script (véase más abajo) que puede utilizar para navegar por el script. Se pueden crear subcabeceras añadiendo más símbolos #, por ejemplo # para las primarias, ## para las secundarias y ### para las terciarias.

A continuación se muestran dos versiones de un script de ejemplo. A la izquierda está el original con cabeceras comentadas. A la derecha, se han escrito cuatro guiones después de cada cabecera, haciéndolas plegables. Dos de ellas están plegadas, y se puede ver que la Tabla de Contenidos en la parte inferior muestra cada sección.

Otras áreas de código que son automáticamente elegibles para plegarlas son las zonas entre corchetes { } como las definiciones de funciones o los bloques condicionales (sentencias if else). Puedes leer más sobre el plegado de código en el [sitio de](https://support.rstudio.com/hc/en-us/articles/200484568-Code-Folding-and-Sections) RStudio.

## Directorio de trabajo

El directorio de trabajo (o Working Directory “WD”) es la ubicación de la carpeta raíz utilizada por R para su trabajo - donde R busca y guarda los archivos por defecto. Por defecto, guardará los nuevos archivos y resultados en esta ubicación, y buscará los archivos de datos para importar aquí también.

El directorio de trabajo aparece en texto gris en la parte superior de la consola de RStudio. También puede imprimir el directorio de trabajo actual ejecutando getwd() (deja los paréntesis vacíos).

### Enfoque recomendado

**Consulte la página sobre** [**proyectos de R**](#r-projects) **para obtener detalles sobre nuestro enfoque recomendado para gestionar tu directorio de trabajo.**   
Una forma común, eficiente y sin problemas de gestionar tu directorio de trabajo y las rutas de los archivos es combinar estos 3 elementos en un flujo de trabajo [orientado a los proyectos de R](#r-projects):

1. Un proyecto R para almacenar todos tus archivos (ver página sobre [proyectos R](#r-projects))
2. El paquete **here** para localizar los archivos (véase la página sobre [importación y exportación](#import-and-export))
3. El paquete **rio** para importar/exportar archivos (véase la página sobre [importación y exportación](#import-and-export))

### Establecerlo mediante comandos

Hasta hace poco, a muchas personas que aprendían R se les enseñaba a comenzar sus scripts con un comando setwd(). En vez de esto, piensa mejor en lugar el uso de un flujo de trabajo [orientado al proyecto R](#r-projects) y lee las [razones para no usar setwd()](https://www.tidyverse.org/blog/2017/12/workflow-vs-script/). En resumen, tu trabajo se convierte en algo específico de tu ordenador, las rutas de archivo utilizadas para importar y exportar archivos se vuelven "frágiles", y esto dificulta gravemente la colaboración y el uso de tu código en cualquier otro ordenador. Hay alternativas fáciles!

Como se ha indicado anteriormente, aunque no recomendamos este enfoque en la mayoría de las circunstancias, puede utilizar el comando setwd() con la ruta del archivo de la carpeta deseada entre comillas, por ejemplo:

**PELIGRO:** Establecer un directorio de trabajo con setwd() puede ser "frágil" si la ruta del archivo es específica de un ordenador. En su lugar, utiliza rutas de archivos relativas a un directorio raíz del proyecto R (con el paquete **here**).

### Establecerlo manualmente

Para establecer el directorio de trabajo manualmente (el equivalente de apuntar y clicar en setwd()), clica en el menú desplegable Session y luego “Set Working Directory” (Establecer el directorio de trabajo) y entonces “Choose Directory” (Elegir el directorio). Esto establecerá el directorio de trabajo para esa sesión específica de R. Nota: si utiliza este enfoque, tendrás que hacerlo manualmente cada vez que abras RStudio.

### Dentro de un proyecto R

Si se utiliza un proyecto R, el directorio de trabajo será por defecto la carpeta raíz del proyecto R que contiene el archivo ".rproj". Esto se aplicará si clicas en abrir el proyecto en RStudio (el archivo con extensión ".rproj").

### Directorio de trabajo en R markdown

En un script de R markdown, el directorio de trabajo por defecto es la carpeta en la que se guarda el archivo Rmarkdown (.Rmd). Si se utiliza un proyecto R y el paquete **here**, esto no se aplica y el directorio de trabajo será here() como se explica en la página de [proyectos R](#r-projects).

Si quieres cambiar el directorio de trabajo de un Rmarkdown independiente (no en un proyecto R), si utiliza setwd() sólo se aplicará a ese trozo de código específico. Para hacer el cambio para todos párrafos, edite el trozo de configuración para añadir el parámetro root.dir =, como se indica a continuación:

Es mucho más fácil usar Rmarkdown dentro de un proyecto R y usar el paquete **here**.

### **Escribir** las rutas de los archivos

Quizás la fuente más común de frustración para un principiante de R (al menos en una máquina Windows) es escribir una ruta de archivo para importar o exportar datos. Hay una explicación exhaustiva sobre la mejor manera de introducir las rutas de los archivos en la página de [importación y exportación](#import-and-export), pero aquí hay algunos puntos clave:

****Ruta**s rotas**

A continuación se muestra un ejemplo de una ruta de archivo "absoluta" o de "dirección completa". Es probable que se rompa si se utiliza en otro ordenador. Una excepción es si estás usando una unidad compartida/de red.

C:/Nombre de usuario/Documento/Software analítico/R/Proyectos/Análisis2019/datos/Marzo2019.csv

**Dirección de la barra**

Si escribe una ruta de archivo, Ten en cuenta la dirección de las barras inclinadas (/). Utiliza barras inclinadas (/) para separar los componentes ("data/provincial.csv"). Para los usuarios de Windows, la forma predeterminada en que se muestran las rutas de los archivos es con barras invertidas (\) - por lo que tendrá que cambiar la dirección de cada barra. Si utiliza el paquete **here**, tal y como se describe en la página de [proyectos de R,](#r-projects) la dirección de la barra no es un problema.

**Rutas relativas de archivos**

Generalmente recomendamos proporcionar rutas de archivo "relativas" en su lugar - es decir, la ruta relativa a la raíz de tu proyecto R. Puedes hacer esto usando el paquete **here** como se explica en la página de [proyectos R](#r-projects). Una ruta de archivo relativa podría ser así:

Incluso si se utilizan rutas de archivo relativas dentro de un proyecto R, se pueden utilizar rutas absolutas para importar/exportar datos fuera del proyecto R.

## Objetos

Todo en R es un objeto, y R es un lenguaje "orientado a objetos". Estas secciones lo explicarán:

* Cómo crear objetos (<-)
* Tipos de objetos (por ejemplo, dataframe -conjunto de datos-, vectores...)
* Cómo acceder a las subpartes de los objetos (por ejemplo, las variables de unos datos)
* Tipos de objetos (por ejemplo, numéricos, lógicos, enteros, dobles, caracteres, factores)

### Todo es un objeto

Esta sección está adaptada del [*proyecto R4Epis*](https://r4epis.netlify.app/training/r_basics/objects/).   
Todo lo que se almacena en R -conjuntos de datos (dataframe), variables, una lista de nombres de pueblos, un número total de población, incluso resultados como gráficos- son **objetos a** los que se **asigna un nombre** y a los que **se puede hacer referencia** en comandos posteriores.

Un objeto existe cuando se le ha asignado un valor (véase la sección de asignación más adelante). Cuando se le asigna un valor, el objeto aparece en el Entorno -Environment- (en el panel superior derecho de RStudio). A partir de ese momento se puede operar con él, manipularlo, cambiarlo y redefinirlo.

### Definición de objetos (<-)

**Crea objetos asignándoles un valor con el operador <-.**   
Puedes pensar que el operador de asignación <- significa: "se define como". Los comandos de asignación suelen seguir un orden estándar:

**nombre\_objeto** <- **valor (**o proceso/cálculo que produce un valor)

Por ejemplo, es posible que desee registrar la semana de notificación epidemiológica actual como un objeto para su referencia en un código posterior. En este ejemplo, el objeto current\_week se crea cuando se le asigna el valor "2018-W10" (las comillas hacen que sea un valor de tipo carácter). El objeto current\_week aparecerá entonces en el panel de Entorno RStudio (arriba a la derecha) y podrá ser referenciado en comandos posteriores.

Ver los comandos de R y su salida en los cuadros de abajo.

**NOTA:** Observa que el [1] en la consola de resultados simplemente indica que está viendo el primer elemento de resultados

**ATENCIÓN: El valor de un objeto puede ser sobrescrito en** cualquier momento ejecutando un comando de asignación para redefinir su valor. Por lo tanto, el **orden de ejecución de los comandos es muy importante**.

El siguiente comando redefinirá el valor de current\_week:

**Signos de igualdad =**

También verás signos de igualdad en el código R:

* Un doble signo de igualdad == entre dos objetos o valores formula una pregunta lógica: "¿es esto igual a aquello?".
* También verás signos de igualdad dentro de las funciones que se utilizan para especificar los valores de los argumentos de la función (lee sobre ellos en las secciones siguientes), por ejemplo max(edad, na.rm = TRUE).
* Puedes utilizar un único signo de igualdad = en lugar de <- para crear y definir objetos, pero se desaconseja hacerlo. Puedes leer por qué se desaconseja [aquí](https://renkun.me/2014/01/28/difference-between-assignment-operators-in-r/).

**Conjuntos de datos (dataframes)**

Los conjuntos de datos también son objetos (normalmente "dataframes") y se les debe asignar un nombre cuando se importan. En el código siguiente, se crea el objeto linelist y se le asigna el valor de un archivo CSV importado con el paquete **rio** y su función import().

Puedes obtener más información sobre la importación y la exportación de datos en la sección sobre [importación y exportación](#import-and-export).

**ATENCIÓN:** Una nota rápida sobre la denominación de los objetos:

* Los nombres de los objetos no deben contener espacios, debes utilizar el guión bajo (\_) o un punto (.) en lugar de un espacio.
* Los nombres de los objetos distinguen entre mayúsculas y minúsculas (lo que significa que Dataset\_A es diferente de dataset\_A).
* Los nombres de los objetos deben empezar por una letra (no pueden empezar por un número como 1, 2 o 3).

**Salidas**

Las salidas, como las tablas y los gráficos proporcionan un ejemplo de cómo las salidas pueden ser guardadas como objetos, o simplemente mostrarse en la consola sin ser guardadas. Una tabla cruzada de género y resultado utilizando la función **base** de R table() puede mostrarse directamente en la consola de R (sin guardarse).

Pero la misma tabla puede guardarse como un objeto con nombre. Y luego, opcionalmente, se puede mostrar o imprimir.

**Columnas**

Las columnas de unos datos también son objetos y pueden definirse, sobrescribirse y crearse como se describe a continuación en la sección sobre Columnas.

Puedes utilizar el operador de asignación de **base** R para crear una nueva columna. A continuación, se crea la nueva columna bmi (Índice de masa corporal), y para cada fila el nuevo valor es resultado de una operación matemática sobre el valor de la fila en las columnas wt\_kg y ht\_cm.

Sin embargo, en este manual, hacemos hincapié en un enfoque diferente para definir las columnas, que utiliza la función mutate() del paquete **dplyr** y la canalización con el operador de tubería (%>%). La sintaxis es más fácil de leer y hay otras ventajas que se explican en la página sobre [Limpieza de datos y funciones básicas](#cleaning-data-and-core-functions). Puedes leer más sobre la canalización en la sección de canalización más abajo.

### Estructura de los objetos

**Los objetos pueden ser un solo dato (por ejemplo, mi\_número <- 24), o pueden consistir en datos estructurados.**

El gráfico siguiente está tomado de [este tutorial de R en línea](http://venus.ifca.unican.es/Rintro/dataStruct.html). Muestra algunas estructuras de datos comunes y sus nombres. No se incluyen en esta imagen los datos espaciales, de los que se habla en la página de [fundamentos del SIG](#gis-basics).

En epidemiología (y en particular en epidemiología de campo), lo más habitual es encontrar dataframes (conjuntos de datos) y vectores:

| **Estructura común** | **Explicación** | **Ejemplo** |
| --- | --- | --- |
| Vectores | Un contenedor para una secuencia de objetos singulares, todos del mismo tipo (por ejemplo, numérico, carácter). | **Las "variables" (columnas) en los **dataframes** son vectores** (por ejemplo, la columna de edad en años age\_years). |
| Dataframes | Vectores (por ejemplo, columnas) unidos que tienen todos el mismo número de filas. | linelist es un dataframe. |

Ten en cuenta que para crear un vector que "está solo" (no forma parte de un dataframe) se utiliza la función c() para combinar los diferentes elementos. Por ejemplo, si se crea un vector de colores de la escala de colores del gráfico: vector\_de\_colores <- c("azul", "rojo2", "naranja", "gris")

### Tipos de objeto

Todos los objetos almacenados en R tienen un tipo (class) que indica a R cómo manejar el objeto. Hay muchos tipos de datos, pero los más comunes son:

| **el tipo** | **Explicación** | **Ejemplos** |
| --- | --- | --- |
| Character (Carácter) | Son textos/palabras/frases **"entre comillas"**. No se pueden realizar operaciones matemáticas con estos objetos. | "Los objetos de carácter están entre comillas" |
| Integer (Entero) | Números **sólo enteros** (sin decimales) | -5, 14, o 2000 |
| Numeric (Numérico) | Son números y **pueden incluir decimales**. Si están entre comillas se considerarán de tipo de caracteres. | 23,1 o 14 |
| Factor | Se trata de vectores que tienen un **orden determinado** o jerarquía de valores | Una variable de situación económica con valores ordenados |
| Date (Fecha) | **Una vez que se le dice a R que ciertos datos son Fechas**, estos datos pueden ser manipulados y mostrados de maneras especiales. Para más información, consulte la página sobre el [trabajo con fechas](#working-with-dates-1). | 2018-04-12 o 15/3/1954 o miércoles 4 de enero de 1980 |
| Logical (Lógica) | Los valores deben ser uno de los dos valores especiales TRUE o FALSE (nótese que **no son** "TRUE" y "FALSE" entre comillas) | TRUE o FALSE |
| data.frame | Un dataframe es la forma en que R almacena unos datos **típico**. Consiste en vectores (columnas) de datos unidos, que tienen todos el mismo número de observaciones (filas). | El set de datos AJS de ejemplo denominado linelist\_raw contiene 68 variables con 300 observaciones (filas) cada una. |
| tibble | Los tibbles son una variación de los dataframe, la principal diferencia operativa es que muestran de forma más agradable en la consola (muestran las 10 primeras filas y sólo las columnas que caben en la pantalla) | Cualquier conjunto de datos, lista o matriz puede convertirse en un tibble con as\_tibble() |
| List (lista) | Una lista es como un vector, pero contiene otros objetos que pueden ser de otras tipos diferentes | Una lista puede contener un solo número, un dataframe de datos, un vector e incluso otra lista. |

**Puedes comprobar el tipo de un objeto escribiendo su nombre en la función class()**. Nota: puede hacer referencia a una columna específica dentro de unos datos utilizando la notación $ para separar el nombre de los datos y el nombre de la columna.

class(linelist) # el tipo debe ser un dataframe o un tibble

class(linelist$age) # lel tipo debe ser numérico

class(linelist$gender) # el tipo debe ser el carácter

A veces, una columna puede ser convertida automáticamente por R en un tipo diferente. ¡Cuidado con esto! Por ejemplo, si tiene un vector o columna de números, pero se inserta un valor de carácter... toda la columna cambiará al tipo carácter.

Un ejemplo común de esto es cuando se manipula unos datos para imprimir una tabla - si se hace una fila de totales y se intenta pegar porcentajes en la misma celda como números (por ejemplo, 23 (40%)), toda la columna numérica de arriba se convertirá en carácter y ya no se podrá utilizar para cálculos matemáticos. **A veces, tendrá que convertir objetos o columnas a otro tipo.**

| **Función** | **Acción** |
| --- | --- |
| as.character() | Convierte al tipo de carácter |
| as.numeric() | Convierte al tipo numérico |
| as.integer() | Convierte al tipo entero |
| as.date() | Convierte al tipo de fecha - Nota: ver la sección de [fechas](#dates) para más detalles |
| factor() | Convierte en factor - Nota: la redefinición del orden de los niveles de valor requiere argumentos adicionales |

Asimismo, existen funciones **base de** R para comprobar si un objeto ES de un tipo específico, como is.numeric(), is.character(), is.double(), is.factor(), is.integer()

Aquí hay [más material en línea sobre tipos y estructuras de datos en R](https://swcarpentry.github.io/r-novice-inflammation/13-supp-data-structures/).

### Columnas/Variables ($)

**Una columna en un dataframe es técnicamente un "vector" (véase la tabla anterior)** - una serie de valores que deben ser todos del mismo tipo (ya sea carácter, numérico, lógico, etc).

Un vector puede existir independientemente de un dataframe, por ejemplo, un vector de nombres de columnas que se desea incluir como variables explicativas en un modelo. Para crear un vector "independiente", utiliza la función c() como se indica a continuación:

**Las columnas de un dataframe también son vectores y pueden ser llamadas, referenciadas, extraídas o creadas utilizando el símbolo $.** El símbolo $ conecta el nombre de la columna con el nombre de su dataframe. En este manual, tratamos de utilizar la palabra "columna" en lugar de "variable".

Al escribir el nombre del dataframe seguido de $ también verá un menú desplegable de todas las columnas del dataframe. Puedes desplazarte por ellas con la tecla de flecha, seleccionar una con la tecla Intro y evitar errores ortográficos.

**CONSEJO AVANZADO:** Algunos objetos más complejos (por ejemplo, una lista, o un objeto epicontacts) pueden tener múltiples niveles a los que se puede acceder a través de múltiples signos de dólar. Por ejemplo epicontacts$linelist$date\_onset

### Acceso/índice con corchetes ([ ])

Es posible que tengas que mirar sólo partes de los objetos, lo que también se llama "indexación", que a menudo se hace utilizando los corchetes [ ]. El uso de $ en un dataframe para acceder a una columna también es un tipo de indexación.

Los corchetes también sirven para devolver partes específicas de un resultado, como la salida de una función summary():

Los corchetes también funcionan en los dataframes para ver filas y columnas específicas. Puedes hacerlo utilizando la sintaxis dataframe[filas, columnas]:

Ten en cuenta que también puedes lograr la indexación de filas/columnas anterior en dataframes y tibbles utilizando la sintaxis **de dplyr** (funciones filter() para filas, y select() para columnas). Puedes leer más sobre estas funciones básicas en la página de [Limpieza de datos y funciones básicas](#cleaning-data-and-core-functions).

Para filtrar en base al "número de fila", puedes utilizar la función **dplyr** row\_number() (con paréntesis) como parte de una sentencia lógica de filtrado. A menudo se utiliza el operador %in% y un rango de números como parte de esa sentencia lógica, como se muestra a continuación. Para ver las primeras N filas, también puede utilizar la función especial **de dplyr** head().

Cuando se indexa un objeto del tipo **list**, los corchetes simples siempre devuelven con el tipo list, incluso si sólo se devuelve un único objeto. Los corchetes dobles, sin embargo, pueden utilizarse para acceder a un solo elemento y devolver un tipo diferente al de la lista.   
Los corchetes también pueden escribirse uno tras otro, como se demuestra a continuación.

Esta [explicación visual de la indexación de listas, con pimenteros](https://r4ds.had.co.nz/vectors.html" \l "lists-of-condiments), es divertida y útil.

Este es el aspecto de la lista cuando se muestra en la consola. Mira cómo hay dos elementos con nombre:

* hospital, un vector de caracteres
* addresses, unos datos de direcciones

Ahora extraeremos información, utilizando varios métodos:

### Eliminar objetos

Puedes eliminar objetos individuales de tu entorno R poniendo el nombre en la función rm() (sin comillas):

Puedes eliminar todos los objetos (limpiar tu espacio de trabajo) ejecutando:

## Tuberías (Pipes) (%>%)

**Hay dos enfoques generales para trabajar con objetos:**

1. **Pipes/tidyverse** - las tuberías envían un objeto de función a función - el énfasis está en la acción, no en el objeto
2. **Definir objetos intermedios** - un objeto se redefine una y otra vez - el énfasis está en el objeto

### ****Tuberías****

**Explicado de forma sencilla, el operador de tubería (%>%) pasa un resultado intermedio de una función a la siguiente.**   
Puedes pensar en él como si dijera "entonces". Muchas funciones pueden enlazarse con %>%.

* **Las tuberías hacen hincapié en una secuencia de acciones, no en el objeto sobre el que se realizan las acciones**
* Las tuberías son mejores cuando hay que realizar una secuencia de acciones en un objeto
* Las tuberías provienen del paquete **magrittr**, que se incluye automáticamente en los paquetes **dplyr** y **tidyverse**
* Las tuberías pueden hacer que el código sea más limpio y fácil de leer, más intuitivo

Más información sobre este enfoque en la [guía de estilo](https://style.tidyverse.org/pipes.html) de tidyverse

He aquí un ejemplo para comparar, utilizando funciones ficticias para "hornear un pastel". Primero, el método de la tubería:

Aquí hay otro [enlace](https://cfss.uchicago.edu/notes/pipes/" \l ":~:text=Pipes are an extremely useful,code and combine multiple operations) que describe la utilidad de las tuberías.

El uso de tuberías no es una función **base**. Para usar piping, debe estar instalado y cargado el paquete magrittr (esto se hace normalmente cargando el paquete **tidyverse** o **dplyr** que lo incluyen). Puedes [leer más sobre tuberías (pipes) en la documentación de magrittr](https://magrittr.tidyverse.org/).

Ten en cuenta que, al igual que otros comandos de R, las tuberías se pueden utilizar para mostrar sólo el resultado, o para guardar/re-guardar un objeto, dependiendo de si el operador de asignación <- está involucrado. Mira ambas cosas a continuación:

%<>% Es una "tubería de asignación" del paquete **magrittr**, que canaliza un objeto hacia adelante y también redefine el objeto. Debe ser el primer operador de tubería en la cadena. Es una forma abreviada. Los dos comandos siguientes son equivalentes:

### Definir objetos intermedios

Este enfoque para cambiar objetos/dataframes puede ser mejor si:

* Necesitas manipular múltiples objetos
* Hay pasos intermedios que son significativos y merecen nombres de objetos separados

**Riesgos:**

* Crear nuevos objetos para cada paso significa crear muchos objetos. Si usas el equivocado, ¡puedes no darte cuenta!
* Nombrar todos los objetos puede ser confuso
* Los errores pueden no ser fácilmente detectables

O bien nombrar cada objeto intermedio, o sobrescribir el original, o combinar todas las funciones juntas. Todo ello conlleva sus propios riesgos.

A continuación se muestra el mismo ejemplo de "pastel" falso que el anterior, pero utilizando este estilo:

Combinar todas las funciones juntas - esto es difícil de leer:

## Operadores y funciones clave

Esta sección detalla los operadores en R, como por ejemplo

* Operadores de definición
* Operadores relacionales (menor que, igual a ..)
* Operadores lógicos (and, or...)
* Tratamiento de los valores faltantes
* Operadores y funciones matemáticas (+/-, >, sum(), median(), ...)
* El operador %in%

### Operadores de asignación

**<-**

El operador de asignación básico en R es <-. como nombre\_objeto <- valor.   
Este operador de asignación también se puede escribir como =. Aconsejamos el uso de <- para el uso general de R.   
También aconsejamos rodear tales operadores con espacios, para mejorar la legibilidad.

**<<-**

Si se [escriben funciones](#writing-functions-1), o se utiliza R de forma interactiva con scripts de origen, entonces puede ser necesario utilizar este operador de asignación <<- (de R **base).** Este operador se utiliza para definir un objeto en un entorno R superior "padre". Vea esta [referencia online](https://stat.ethz.ch/R-manual/R-devel/library/base/html/assignOps.html).

**%<>%**

Se trata de un "tubería de asignación" del paquete **magrittr**, que canaliza un objeto hacia adelante y también redefine el objeto. Debe ser el primer operador de tubería en la cadena. Es una forma abreviada, como se muestra a continuación en dos ejemplos equivalentes:

Lo anterior equivale a lo siguiente:

**%<+%**

Se utiliza para añadir datos a los árboles filogenéticos con el paquete **ggtree**. Consulte la página sobre [árboles filogenéticos](#phylogenetic-trees-1) o este [libro de recursos](https://yulab-smu.top/treedata-book/) en línea.

### Operadores relacionales y lógicos

Los operadores relacionales comparan valores y se utilizan a menudo al definir nuevas variables y subconjuntos de datos. Estos son los operadores relacionales más comunes en R:

| **Significado** | **Operador** | **Ejemplo** | **Ejemplo de resultado** |
| --- | --- | --- | --- |
| Igual a | == | "A" == "a" | FALSE (porque R distingue entre mayúsculas y minúsculas) Ten en cuenta que == (doble igual) es diferente de = (simple igual), que actúa como el operador de asignación *<-* |
| No es igual a | != | 2 != 0 | TRUE |
| Mayor que | > | 4 > 2 | TRUE |
| Menor que | < | 4 < 2 | FALSE |
| Mayor o igual que | >= | 6 >= 4 | TRUE |
| Menor o igual que | <= | 6 <= 4 | FALSE |
| Falta el valor | is.na() | is.na(7) | FALSE (véase la página sobre [Valores faltantes](#missing-data)) |
| No falta el valor | es.na() | es.na(7) | TRUE |

Los operadores lógicos o booleanos, como AND y OR, suelen utilizarse para conectar operadores relacionales y crear criterios más complicados. Las sentencias complejas pueden requerir paréntesis () para la agrupación y el orden de aplicación.

| **Significado** | **Operador** |
| --- | --- |
| Y | & |
| O | | (barra vertical) |
| Paréntesis | ( ) Se utiliza para agrupar criterios y aclarar el orden de las operaciones |

Por ejemplo, a continuación, tenemos un listado con dos variables que queremos utilizar para crear nuestra definición de caso, hep\_e\_rdt, un resultado de la prueba y other\_cases\_in\_hh, que nos dirá si hay otros casos en el hogar. El comando siguiente utiliza la función case\_when() para crear la nueva variable case\_def tal que:

| **Criterios del ejemplo anterior** | **Valor resultante en la nueva variable "case\_def"** |
| --- | --- |
| Si falta el valor de las variables rdt\_result y other\_cases\_in\_home | NA (valor faltante) |
| Si el valor de rdt\_result es "Positivo" | "Confirmado" |
| Si el valor de rdt\_result NO es "Positivo" Y el valor de other\_cases\_in\_home es "Si" | "Probable" |
| Si no se cumple alguno de los criterios anteriores | "Sospechoso" |

Ten en cuenta que R distingue entre mayúsculas y minúsculas, por lo que "Positivo" es diferente de "positivo"...

### Valores faltantes

En R, los valores faltantes (missing value) se representan con el valor especial NA (una palabra "reservada" para ello) (letras mayúsculas N y A - sin comillas). Si importa datos que registran valores missing de otra manera (por ejemplo, 99, "Nulo", o .), es posible que quieras re-codificar esos valores a NA. En la página de [importación y exportación](#import-and-export) se explica cómo hacerlo.

**Para comprobar si un valor es NA, utiliza la función especial is.na()**, que devuelve TRUE o FALSE.

Puedes leer más sobre los valores faltantes, infinitos, NULL e imposibles en la página sobre V[alores faltantes](#missing-data). Aprende a convertir los valores missing al importar datos en la página sobre [Importación y exportación](#import-and-export).

### Matemáticos y estadísticos

Todos los operadores y funciones de esta página están disponibles automáticamente en R **base**.

#### Operadores matemáticos

Se utilizan a menudo para realizar sumas, divisiones, crear nuevas columnas, etc. A continuación se muestran los operadores matemáticos más comunes en R. No es importante poner espacios alrededor de los operadores.

| **Propósito** | **Ejemplo en R** |
| --- | --- |
| adición | 2 + 3 |
| sustracción | 2 - 3 |
| multiplicación | 2 \* 3 |
| división | 30 / 5 |
| exponente | 2^3 |
| orden de las operaciones | ( ) |

#### Funciones matemáticas

| **Propósito** | **Función** |
| --- | --- |
| redondeo | round(x, digits = n) |
| redondeo | janitor::round\_half\_up(x, dígits = n) |
| Redondear hacia arriba | ceiling(x) |
| redondear hacia abajo | floor(x) |
| valor absoluto | abs(x) |
| raíz cuadrada | sqrt(x) |
| exponente | exponent(x) |
| logaritmo natural | log(x) |
| log base 10 | log10(x) |
| log base 2 | log2(x) |

Nota: para round() los dígits = especifican el número de decimales. Utiliza signif() para redondear a un número de cifras significativas.

#### Notación científica

La probabilidad de que se utilice la notación científica depende del valor de la opción scipen.

De la documentación de ?options: scipen es una penalización que se aplica cuando se decide obtener valores numéricos en notación fija o exponencial. Los valores positivos se inclinan hacia la notación fija y los negativos hacia la notación científica: se preferirá la notación fija a menos que tenga más dígitos que 'scipen'.

Si se establece en un número bajo (por ejemplo, 0) estará "activada" siempre. Para "desactivar" la notación científica en tu sesión de R, configúrela con un número muy alto, por ejemplo:

#### Redondeo

**PELIGRO:** round() utiliza el "redondeo del banquero" que redondea hacia arriba desde un 0,5 sólo si el número superior es par. Utiliza round\_half\_up() de **janitor** para redondear consistentemente las mitades hacia arriba al número entero más cercano. [Aquí tienes es](https://cran.r-project.org/web/packages/janitor/vignettes/janitor.html" \l "explore-records-with-duplicated-values-for-specific-combinations-of-variables-with-get_dupes)ta explicación

#### Funciones estadísticas

**PRECAUCIÓN:** Las funciones que aparecen a continuación incluyen por defecto los valores faltantes en los cálculos. Los valores faltantes darán como resultado una salida de NA, a menos que se especifique el argumento na.rm = TRUE. Esto se puede escribir de forma abreviada como na.rm = T.

| **Objetivo** | **Función** |
| --- | --- |
| media (promedio) | mean(x, na.rm=T) |
| mediana | median(x, na.rm=T) |
| desviación estándar | sd(x, na.rm=T) |
| cuantiles\* | quantile(x, probs) |
| suma | sum(x, na.rm=T) |
| valor mínimo | min(x, na.rm=T) |
| valor máximo | max(x, na.rm=T) |
| rango de valores numéricos | range(x, na.rm=T) |
| resumen\*\* | summary(x) |

Notas:

* \*quantile(): x es el vector numérico a examinar, y probs = es un vector numérico con probabilidades entre 0 y 1.0, por ejemplo c(0.5, 0.8, 0.85)
* \*\*summary(): da un resumen estadístico sobre un vector numérico incluyendo la media, mediana y percentiles más comunes

**PELIGRO:** Si proporciona un vector de números a una de las funciones anteriores, asegúrese de envolver los números dentro de c() .

#### Otras funciones útiles

| **Objetivo** | **Función** | **Ejemplo** |
| --- | --- | --- |
| crear una secuencia | seq(de, a, por) | seq(1, 10, 2) |
| repetir x, n veces | rep(x, nveces) | rep(1:3, 2) o bien rep(c("a", "b", "c"), 3) |
| subdividir un vector numérico | cut(x, n) | cut(linelist$age, 5) |
| tomar una muestra aleatoria | sample(x, tamaño) | sample(linelist$id, size = 5, replace = TRUE) |

### %in%.

Un operador muy útil para comparar valores, y para evaluar rápidamente si un valor está dentro de un vector o dataframe.

Para preguntar si un valor **no está** %in% en un vector, ponga un signo de exclamación (!) **delante** de la declaración lógica:

%in% es muy útil cuando se utiliza la función **dplyr** case\_when(). Se puede definir un vector previamente y referenciarlo después. Por ejemplo:

Nota: Si quieres detectar una cadena parcial, quizás usando str\_detect() de **stringr**, no aceptará un vector de caracteres como c("1", "Yes", "yes", "y"). En su lugar, se le debe dar una expresión regular - una cadena condensada con barras OR, como "1|Yes|yes|y". Por ejemplo, str\_detect(hospitalized, "1|Yes|yes|y"). Consulte la página sobre [Caracteres y cadenas](#characters-and-strings) para obtener más información.

Puedes convertir un vector de caracteres en una expresión regular con este comando:

## Errores & avisos

En esta sección se explica:

* La diferencia entre errores y avisos (warnings)
* Consejos generales de sintaxis para escribir código R
* Código de asistencia

En la página de [Errores y ayuda] se pueden encontrar los errores y avisos más comunes, así como consejos para la resolución de problemas.

### Error vs aviso

Cuando se ejecuta un comando, la Consola de R puede mostrarte mensajes de aviso o error en texto rojo.

* Una **aviso** significa que R ha completado su comando, pero ha tenido que dar pasos adicionales o ha producido una salida inusual de la que deberías estar al tanto.
* Un **error** significa que R no pudo completar su comando.

Busca pistas:

* El mensaje de error/advertencia suele incluir un número de línea para el problema.
* Si un objeto "es desconocido" o "no se encuentra", quizás lo hayas escrito mal, hayas olvidado llamar a un paquete con library(), o hayas olvidado volver a ejecutar tu script después de hacer cambios.

Si todo lo demás falla, copia el mensaje de error en Google junto con algunos términos clave; lo más probable es que a alguien le haya pasado lo mismo y ya haya resuelto el problema.

### Consejos generales de sintaxis

Algunas cosas que hay que recordar al escribir comandos en R, para evitar errores y advertencias:

* Cierra siempre los paréntesis - consejo: cuenta el número de paréntesis de apertura "(" y de cierre ")" de cada trozo de código
* Evita los espacios en los nombres de columnas y objetos. Utiliza barras bajas ( \_ ) o puntos ( . ) en su lugar
* No olvides separar los argumentos de una función con comas
* R distingue entre mayúsculas y minúsculas, lo que significa que Variable\_A es diferente de variable\_A

### Código de asistencia

Cualquier script (RMarkdown o de otro tipo) te dará pistas cuando haya cometido un error. Por ejemplo, si te olvidaste de escribir una coma donde se necesita, o de cerrar un paréntesis, RStudio levantará una bandera en esa línea, a la derecha del script, para avisarte.

&&&&&

# #Transición a R

{#transition-to-r}

A continuación, te ofrecemos algunos consejos y recursos que resultan útiles si te estás pasando a R.

R se introdujo a finales de la década de 1990 y desde entonces su alcance ha crecido de forma espectacular. Sus capacidades son tan amplias que las alternativas comerciales han reaccionado a los desarrollos de R para seguir siendo competitivas. ([lea este artículo que compara R, SPSS, SAS, STATA y Python](https://www.inwt-statistics.com/read-blog/comparison-of-r-python-sas-spss-and-stata.html)).

Además, R es mucho más fácil de aprender que hace 10 años. Antes, R tenía fama de ser difícil para los principiantes. Ahora es mucho más fácil, con interfaces de usuario amigables como RStudio, código intuitivo como **tidyverse**, y muchos recursos tutoriales.

**¡No te dejes intimidar, ven a descubrir el mundo de R!**

## Desde Excel

La transición de Excel directamente a R es un objetivo muy alcanzable. Puede parecer desalentador, ¡pero puedes hacerlo!

Es cierto que alguien con grandes conocimientos de Excel puede realizar actividades muy avanzadas sólo con Excel, incluso utilizando herramientas de scripting como VBA. Excel se utiliza en todo el mundo y es una herramienta esencial para la epidemiología. Sin embargo, complementarlo con R puede mejorar y ampliar drásticamente tus flujos de trabajo.

### Beneficios

Descubrirás que el uso de R ofrece inmensos beneficios en cuanto a ahorro de tiempo, análisis más consistentes y precisos, reproducibilidad, posibilidad de compartir y una corrección de errores más rápida. Como cualquier software nuevo, hay una "curva" de aprendizaje en la que hay que invertir tiempo para familiarizarse. Los dividendos serán significativos y se te abrirá un inmenso abanico de nuevas posibilidades con R.

Excel es un software muy conocido que permite que un principiante pueda realizar análisis y visualizaciones simples con "apuntar y clicar" de manera sencilla. En comparación, puede llevar un par de semanas sentirse cómodo con las funciones y la interfaz de R. Sin embargo, R ha evolucionado en los últimos años para ser mucho más amigable con los principiantes.

Muchos flujos de trabajo de Excel se basan en la memoria y en la repetición, por lo que hay muchas posibilidades de error. Además, generalmente la limpieza de datos, la metodología de análisis y las ecuaciones utilizadas están ocultas a la vista. A un nuevo colega le puede llevar mucho tiempo aprender lo que hace un libro de Excel y cómo resolver problemas que surjan. Con R, todos los pasos se escriben explícitamente en el script y pueden verse, editarse, corregirse y aplicarse fácilmente a otros conjuntos de datos.

**Para comenzar tu transición de Excel a R debes ajustar tu mentalidad en algunos aspectos importantes:**

### Datos ordenados (tidy data)

Debes utilizar datos "ordenados" (tidy), legibles por la máquina en lugar de datos desordenados "legibles por el ser humano". Estos son los tres requisitos principales que los datos "ordenados" deben cumplir, como se explica en este tutorial sobre [datos "ordenados" en R](https://r4ds.had.co.nz/tidy-data.html):

* Cada variable debe tener su propia columna
* Cada observación debe tener su propia fila
* Cada valor debe tener su propia celda

Para los usuarios de Excel: piensa en el papel que desempeñan las ["tablas" de Excel](https://exceljet.net/excel-tables) para estandarizar los datos y hacer que el formato sea más predecible.

Un ejemplo de datos "ordenados" sería el listado de casos utilizado en este manual: cada variable está contenida en una columna, cada observación (un caso) tiene su propia fila y cada valor está en una sola celda. A continuación, puede ver las primeras 50 filas del listado:

La principal razón por la que uno se encuentra con datos no ordenados es porque muchas hojas de cálculo en Excel están diseñadas para dar prioridad a la lectura fácil por parte de los humanos, no a la lectura fácil por parte de las máquinas/el software.

Para ayudarte a ver la diferencia, se presentan algunos ejemplos ficticios de **datos no ordenados** a continuación, los cuales dan prioridad a la legibilidad humana sobre la legibilidad mecánica:

Problemas: En la hoja de cálculo de arriba, hay celdas combinadas que no son fácilmente digeridas por R. No está claro qué fila debe utilizarse para la "cabecera". A la derecha hay un diccionario basado en colores y los valores de las celdas están representados por colores, lo que tampoco es fácilmente interpretado por R (¡ni por los humanos que padecen daltonismo!). Además, se combinan diferentes informaciones en una celda (varias organizaciones asociadas que trabajan en un área, o el estado "TBC" en la misma celda que "Partner D").

Problemas: En la hoja de cálculo anterior, hay numerosas filas y columnas vacías adicionales dentro de los datos, lo que provocará dolores de cabeza a la hora de limpiarlos con R. Además, las coordenadas GPS están repartidas en dos filas para un centro de tratamiento determinado. Y, una nota adicional, ¡las coordenadas GPS están en dos formatos diferentes!

Los conjuntos de datos "ordenados" pueden no ser tan legibles para el ojo humano, pero facilitan mucho la limpieza y el análisis de los datos. Los datos ordenados pueden almacenarse en varios formatos, por ejemplo, "largos" o "anchos" (véase la página sobre [Pivotar datos](#pivoting-data)), pero se siguen observando los principios anteriores.

### Funciones

Puede que la palabra "función" en R sea nueva, pero el concepto existe también en Excel y se le conoce como fórmulas. Las fórmulas en Excel también requieren una sintaxis precisa (por ejemplo, la colocación de puntos y comas y paréntesis). Lo único que hay que hacer es aprender algunas funciones nuevas y cómo funcionan en R.

### Scripts

En lugar de clicar en los botones y arrastrar las celdas, escribirás cada paso y procedimiento en un "script" (secuencia de órdenes). Los usuarios de Excel pueden estar familiarizados con las "macros VBA", que también emplean un enfoque de scripting (secuencia de comando VBA).

El script de R consiste en instrucciones paso a paso. Esto permite que cualquier colega pueda leer el script y ver fácilmente los pasos que has dado. Esto también ayuda a depurar errores o cálculos inexactos. Consulta la sección de [Aspectos básicos de R](#r-basics) sobre scripts para ver algunos ejemplos.

Este es un ejemplo de un script en R:

### Recursos útiles en la migración Excel a R

Aquí hay algunos enlaces a tutoriales que te ayudarán en la transición a R desde Excel:

* [R vs. Excel](https://www.northeastern.edu/graduate/blog/r-vs-excel/)
* [Curso de RStudio en R para usuarios de Excel](https://rstudio-conf-2020.github.io/r-for-excel/)

### Interacción R-Excel

R tiene formas robustas de importar libros de Excel, trabajar con los datos, exportar/guardar archivos de Excel y trabajar con los detalles de las hojas de Excel.

Es cierto que algunos de los formatos más estéticos de Excel pueden perderse en la traducción (por ejemplo, la cursiva, el texto lateral, etc.). Si tu flujo de trabajo requiere que pases documentos de un lado a otro entre R y Excel conservando el formato original de Excel, prueba paquetes como **openxlsx**.

## Desde Stata

**Llegando a R desde Stata**

A muchas personas en el campo de la epidemiología se les enseña primero a usar Stata, y puede parecer desalentador pasar a R. Sin embargo, si eres un usuario habitual de Stata, el salto a R es ciertamente más manejable de lo que podrías pensar. Si bien hay algunas diferencias clave entre Stata y R en la forma en que se pueden crear y modificar los datos, así como en la forma en que se implementan las funciones de análisis - después de aprender estas diferencias clave serás capaz de adaptar tus habilidades.

A continuación se presentan algunas traducciones clave entre Stata y R, que pueden ser útiles mientras revisas esta guía.

**Notas generales**

| **STATA** | **R** |
| --- | --- |
| Sólo se puede ver y manipular unos datos a la vez | Puedes ver y manipular varios conjuntos de datos al mismo tiempo, por lo que con frecuencia tendrás que especificar qué conjunto de datos dentro del código |
| Comunidad en línea disponible a través de <https://www.statalist.org/> | Comunidad online disponible a través de [RStudio](https://community.rstudio.com/), [StackOverFlow](https://stackoverflow.com/questions/tagged/r) y [R-bloggers](https://www.r-bloggers.com/) |
| Funcionalidad de apuntar y clicar como opción | Funcionalidad mínima de apuntar y clicar |
| Ayuda para los comandos disponibles mediante help [comando] | Ayuda disponible utilizando el formato [función]? o búsqueda en el panel de ayuda |
| Comentar el código usando \* o /// o /\* TEXTO \*/ | Comentar el código con # |
| Casi todos los comandos son propios de Stata. Las funciones nuevas/escritas por el usuario pueden instalarse como archivos **ado** utilizando el paquete **ssc install** | R se instala con las funciones **base**, pero el uso típico implica la instalación de otros paquetes desde CRAN (véase la página sobre [Aspectos básicos de R](#r-basics)) |
| El análisis se suele escribir en un archivo **do** | El análisis se escribe en un script de R en el panel de fuentes de RStudio. Los scripts de R markdown son una alternativa. |

**Directorio de trabajo**

| **STATA** | **R** |
| --- | --- |
| Los directorios de trabajo implican rutas de archivo absolutas (por ejemplo, "C:/nombredeusuario/documentos/proyectos/datos/") | Los directorios de trabajo pueden ser absolutos, o relativos a la carpeta raíz del proyecto utilizando el paquete **here** (ver [Importar y exportar](#import-and-export)) |
| Ver el directorio de trabajo actual con **pwd** | Utiliza getwd() o here() (si utilizas el paquete **here**), con paréntesis vacíos |
| Establecer el directorio de trabajo con **cd** "ubicación de la carpeta" | Usar setwd("ubicación de la carpeta"), o set\_here("ubicación de la carpeta”, si utilizas el paquete **here**) |

**Importación y visualización de datos**

| **STATA** | **R** |
| --- | --- |
| Comandos específicos por tipo de archivo | Usar import() del paquete **rio** para casi todos los tipos de archivos. Existen funciones específicas como alternativas (véase [Importar y exportar](#import-and-export)) |
| La lectura de los archivos csv se realiza mediante la **importación delimitada** "nombrearchivo.csv" | Usar import("nombredearchivo.csv") |
| La lectura de los archivos xslx se realiza mediante la **importación de excel** "nombre de archivo.xlsx" | Usar import("nombredearchivo.xlsx") |
| Examinar sus datos en una nueva ventana utilizando el comando **browse** | Ver unos datos en el panel de origen de RStudio utilizando View(dataset). Es necesario especificar el nombre de los datos a la función en R porque se pueden mantener varios conjuntos de datos al mismo tiempo. Atención a la "V" mayúscula en esta función |
| Obtener una visión general de alto nivel de su set de datos utilizando **summarize**, que proporciona los nombres de las variables y la información básica | Obtener una visión general de los datos mediante summary(dataset) |

**Manipulación básica de datos**

| **STATA** | **R** |
| --- | --- |
| Las columnas de los datos suelen denominarse "variables" | Más a menudo se denominan "columnas" o a veces "vectores" o "variables" |
| No es necesario especificar los datos | En cada uno de los siguientes comandos, es necesario especificar los datos - véase la página sobre [Limpieza de datos y funciones básicas](#cleaning-data-and-core-functions) para ver ejemplos |
| Las nuevas variables se crean con el comando **generate** varname = | Generar nuevas variables utilizando la función mutate(varname = ). Consultar la página sobre [Limpieza de datos y funciones básicas](#cleaning-data-and-core-functions) para obtener detalles sobre todas las funciones de **dplyr** que aparecen a continuación. |
| Las variables se renombran mediante **rename** nombre\_antiguo nombre\_nuevo | Las columnas pueden renombrarse mediante la función rename(nombre\_antiguo = nombre\_nuevo) |
| Las variables se eliminan con **drop** variable | Las columnas pueden eliminarse mediante la función select() con el nombre de la columna detrás de un signo menos, entre paréntesis |
| Las variables factoriales se pueden etiquetar utilizando una serie de comandos como **label define** | El etiquetado de los valores puede hacerse convirtiendo la columna en tipo Factor y especificando los niveles. Mira en la página sobre [Factores](#factors). Los nombres de las columnas no suelen estar etiquetados como en Stata. |

**Análisis descriptivo**

| **STATA** | **R** |
| --- | --- |
| Tabular los recuentos de una variable mediante el **tab** variable | Proporcionar los datos y el nombre de la columna al comando table() como table(conjunto\_de\_datos$nombre\_columna). Alternativamente, utilizar count(varname) del paquete **dplyr**, como se explica en [Agrupar datos](#grouping-data) |
| La tabulación cruzada de dos variables en una tabla de 2x2 se realiza con **tab** variable1 variable2 | Utilizar table(conjunto\_de\_datos$nombre\_variable1, conjunto\_de\_datos$nombre\_variable2 o count(nombre\_variable1, nombre\_variable2) |

Aunque esta lista ofrece una visión general de los fundamentos de la traducción de los comandos de Stata a R, no es exhaustiva. Hay muchos otros grandes recursos para los usuarios de Stata que podrían ser de interés en tu transición a R:

* <https://dss.princeton.edu/training/RStata.pdf>
* <https://clanfear.github.io/Stata_R_Equivalency/docs/r_stata_commands.html>
* <http://r4stats.com/books/r4stata/>

## Desde SAS

**Pasar de SAS a R**

SAS se utiliza habitualmente en las agencias de salud pública y en los campos de investigación académica. Aunque la transición a un nuevo lenguaje no suele ser un proceso sencillo, entender las diferencias clave entre SAS y R puede ayudarte a empezar a navegar por el nuevo lenguaje utilizando el lenguaje de partida. A continuación se describen las principales traducciones en materia de gestión de datos y análisis descriptivo entre SAS y R.

**Notas generales**

| **SAS** | **R** |
| --- | --- |
| Comunidad en línea disponible a través del [Servicio de Atención al Cliente de SAS](https://support.sas.com/en/support-home.html) | Comunidad online disponible a través de RStudio, StackOverFlow y R-bloggers |
| Ayuda para los comandos disponibles mediante help [comando] | Ayuda disponible usando el formato [función]? o buscando en el panel de ayuda |
| Comentar el código usando \* TEXTO ; o /\* TEXTO \*/ | Comentar el código con # |
| Casi todos los comandos están incorporados. Los usuarios pueden escribir nuevas funciones utilizando macros SAS, SAS/IML, SAS Component Language (SCL) y, más recientemente, los procedimientos Proc Fcmp y Proc Proto | R se instala con las funciones **base**, pero el uso típico implica la instalación de otros paquetes desde CRAN (véase la página sobre [Aspectos básicos de R](#r-basics)) |
| El análisis suele realizarse escribiendo un programa SAS en la ventana del Editor. | Análisis escrito en un script de R en el panel de fuentes de RStudio. Los scripts de R markdown son una alternativa. |

**Directorio de trabajo**

| **SAS** | **R** |
| --- | --- |
| Los directorios de trabajo pueden ser absolutos o relativos a la carpeta raíz del proyecto, definiendo la carpeta raíz con %let rootdir=/ruta raíz; %include "&rootdir/subfoldername/archivo" | Los directorios de trabajo pueden ser absolutos, o relativos a la carpeta raíz del proyecto utilizando el paquete **here** (ver [Importar y exportar](#import-and-export)) |
| Ver el directorio de trabajo actual con %put %sysfunc(getoption(work)); | Utilizar getwd() o here() (si utilizas el paquete **here**), con paréntesis vacíos |
| Establecer el directorio de trabajo con el nombre de la carpeta "ubicación de la carpeta" | Utiliza setwd("ubicación de la carpeta"), o set\_here("ubicación de la carpeta si utilizas el paquete **here**) |

**Importación y visualización de datos**

| **SAS** | **R** |
| --- | --- |
| Utiliza el procedimiento Proc Import o la sentencia Data Step Infile. | Utiliza import() del paquete **rio** para casi todos los tipos de archivos. Existen funciones específicas como alternativas (véase [Importar y exportar](#import-and-export)) |
| La lectura de los archivos csv se realiza mediante Proc Import datafile="nombre de archivo.csv" out=nombre de archivo dbms=CSV; run; O mediante la [sentencia Data Step Infile](http://support.sas.com/techsup/technote/ts673.pdf) | Usar import("nombredearchivo.csv") |
| La lectura de los archivos xslx se realiza utilizando Proc Import datafile="filename.xlsx" out=work.filename dbms=xlsx; run; O utilizando [la sentencia Data Step Infile|Use](http://support.sas.com/techsup/technote/ts673.pdf) import("filename.xlsx") Examinar los datos en una nueva ventana abriendo la ventana del Explorador y seleccionar la biblioteca deseada y los datos | Ver unos datos en el panel de RStudio utilizando View(dataset). Se necesita especificar el nombre del set de datos a la función en R porque se pueden mantener múltiples conjuntos de datos al mismo tiempo. Atención a la "V" mayúscula en esta función |

**Manipulación básica de datos**

| **SAS** | **R** |
| --- | --- |
| Las columnas de los datos suelen denominarse "variables" | Más a menudo se denominan "columnas" o a veces "vectores" o "variables" |
| No es necesario ningún procedimiento especial para crear una variable. Las nuevas variables se crean simplemente escribiendo el nombre de la nueva variable, seguido de un signo igual, y luego una expresión para el valor | Generar nuevas variables utilizando la función mutate(). Consulta la página sobre [Limpieza de datos y funciones básicas](#cleaning-data-and-core-functions) para obtener detalles sobre todas las funciones de **dplyr** que aparecen a continuación. |
| Las variables se renombran utilizando rename \*nombre\_antiguo=nuevo\_nombre\*. | Las columnas pueden renombrarse mediante la función rename(nuevo\_nombre = nombre\_antiguo) |
| Las variables se guardan con \*\*keep\*\*=nombre de la variable | Las columnas pueden seleccionarse mediante la función select() con el nombre de la columna entre paréntesis |
| Las variables se eliminan con \*\*drop\*\*=nombre de la variable | Las columnas pueden eliminarse mediante la función select() con el nombre de la columna detrás de un signo menos, entre paréntesis |
| Las variables factoriales pueden etiquetarse en el mediante la sentencia Label | El etiquetado de los valores puede hacerse convirtiendo la columna en una de tipo Factor y especificando los niveles. Véase la página sobre [Factores](#factors). Los nombres de las columnas no se suelen etiquetar. |
| Los registros se seleccionan utilizando la sentencia Where o If. Las condiciones de selección múltiple se separan con el comando "and". | Los registros se seleccionan mediante la función filter() con múltiples condiciones de selección separadas por un operador AND (&) o una coma |
| Los conjuntos de datos se combinan utilizando la sentencia Merge. Los conjuntos de datos que se van a combinar deben ordenarse primero mediante el procedimiento Proc Sort. | El paquete **dplyr** ofrece algunas funciones para fusionar conjuntos de datos. Para más detalles, consulta la página de [unión de datos](#joining-data). |

**Análisis descriptivo**

| **SAS** | **R** |
| --- | --- |
| Obtener una visión general de los datos mediante el procedimiento Proc Summary, que proporciona los nombres de las variables y las estadísticas descriptivas | Obtener una visión general de tus datos mediante summary(dataset) o skim(dataset) del paquete **skimr** |
| Tabular los recuentos de una variable utilizando proc freq data=Dataset; Tables varname; Run; | Véase la página sobre [tablas descriptivas](#descriptive-tables). Las opciones incluyen table() de R **base**, y tabyl() del paquete **janitor**, entre otras. Ten en cuenta que tendrás que especificar los datos y el nombre de la columna, ya que R mantiene múltiples conjuntos de datos. |
| La tabulación cruzada de dos variables en una tabla 2x2 se realiza con proc freq data=Dataset; Tables rowvar\*colvar; Run; | De nuevo, puedes utilizar table(), tabyl() u otras opciones como se describe en la página de [tablas descriptivas](#descriptive-tables). |

**Algunos recursos útiles:**

[R para usuarios de SAS y SPSS (2011)](https://www.amazon.com/SAS-SPSS-Users-Statistics-Computing/dp/1461406846/ref=sr_1_1?dchild=1&gclid=EAIaIQobChMIoqLOvf6u7wIVAhLnCh1c9w_DEAMYASAAEgJLIfD_BwE&hvadid=241675955927&hvdev=c&hvlocphy=9032185&hvnetw=g&hvqmt=e&hvrand=16854847287059617468&hvtargid=kwd-44746119007&hydadcr=16374_10302157&keywords=r+for+sas+users&qid=1615698213&sr=8-1)

[SAS y R, segunda edición (2014)](https://www.amazon.com/SAS-Management-Statistical-Analysis-Graphics-dp-1466584491/dp/1466584491/ref=dp_ob_title_bk)

## Interoperabilidad de los datos

Consulta la página de [importación y exportación](#import-and-export) para obtener detalles sobre cómo el paquete **rio** puede importar y exportar los archivos .dta de STATA, los archivos .xpt y .sas7bdat de SAS, los archivos .por y .sav de SPSS, y muchos otros.

&&&&&

# #Paquetes recomendados

{#suggested-packages-1}

A continuación hay una larga lista de paquetes de R de utilidad para el trabajo epidemiológico común. Puedes copiar este código, ejecutarlo, y todos estos paquetes se instalarán desde CRAN y se cargarán para su uso en la sesión actual de R. Si un paquete ya está instalado, se cargará.

Puedes modificar el código con símbolos # para excluir los paquetes que no quieras.

Hay que tener en cuenta:

* Instala primero el paquete pacman antes de ejecutar el código siguiente. Puedes hacerlo con install.packages("pacman"). En este manual hacemos hincapié en p\_load() de **pacman**, que instala el paquete si es necesario y lo carga para utilizarlo en la sesión actual de R. También puedes cargar paquetes ya instalados con library() de R **base**.
* En el código siguiente, los paquetes que se incluyen al instalar/cargar otro paquete se indican con una sangría y un hash (#). Por ejemplo, cómo **ggplot2** aparece bajo **tidyverse**.
* Si varios paquetes tienen funciones con el mismo nombre, puede ocurrir enmascaramiento cuando la función del paquete cargado más recientemente tiene prioridad. Lee más en la página de [Aspectos generales de R](#r-basics). Puedes gestionar estos conflictos con el paquete **conflicted** .
* Consulta la sección de A[spectos generales de R](#r-basics) sobre paquetes para obtener más información sobre **pacman** y el enmascaramiento.

Para ver las versiones de R, RStudio y los paquetes de R utilizados durante la producción de este manual, consulte la página de [notas editoriales y técnicas](#editorial-and-technical-notes).

## Paquetes desde CRAN

## Paquetes desde Github

A continuación se muestran los comandos para instalar dos paquetes directamente desde los repositorios de Github.

* La versión de desarrollo de **epicontacts** tiene la capacidad de hacer árboles de transmisión con un eje temporal-x
* El paquete **epirhandbook** contiene todos los datos de ejemplo de este manual y puede utilizarse para descargar la versión sin conexión del manual.

&&&&&

# #Proyectos en R

{#r-projects}

Un proyecto R permite agrupar tu trabajo en una carpeta portable y autocontenida. Dentro del proyecto, todos los scripts relevantes, los archivos de datos, las figuras/resultados y el historial se almacenan en subcarpetas y, lo que es más importante, el directorio de trabajo es la carpeta raíz del proyecto.

## Uso sugerido

Una forma común, eficiente y sin problemas de utilizar R es combinar estos 3 elementos. Un proyecto de trabajo concreto se aloja dentro de un proyecto R. Cada elemento se describe en las secciones siguientes.

1. Un **proyecto en R** 
   * Un entorno de trabajo autónomo con carpetas para datos, scripts, salidas, etc.
2. Se utiliza el paquete **here** para resolver las rutas relativas de los archivos
   * Las rutas de los archivos se escriben en relación con la carpeta raíz del proyecto R - véase [Importar y exportar](#import-and-export) para más información
3. El paquete **rio** para importar/exportar
   * import() y export() manejan cualquier tipo de archivo por su extensión (por ejemplo, .csv, .xlsx, .png)

## Creación de un proyecto R

Para crear un proyecto R, selecciona "New proyect" en el menú File (Archivo).

* Si quieres crear una nueva carpeta para el proyecto, selecciona "New directory" e indica dónde quieres que se cree.
* Si deseas crear el proyecto dentro de una carpeta existente, clica en "Existing Directory" e indica la carpeta.
* Si quieres clonar un repositorio de Github, selecciona la tercera opción "Version Control" y luego "Git". Consulta la página [Control de versiones y colaboración con Git y Github](#version-control-and-collaboration-with-) para más detalles.

El proyecto R que has creado estará en una carpeta que contiene un archivo .Rproj. Este archivo es un acceso directo y probablemente la forma principal de abrir tu proyecto. También puedes abrir un proyecto seleccionando "Open Project" en el menú File. Alternativamente, en el extremo superior derecho de RStudio verás un icono de R projects y un menú desplegable de proyectos R disponibles.

Para salir de un proyecto R, abre un nuevo proyecto o cierra el proyecto (Archivo - Cerrar proyecto).

### Cambiar de proyecto

Para cambiar entre proyectos, clica en el icono de R projects en el menú desplegable en la parte superior derecha de RStudio. Verás las opciones de Cerrar proyecto, Abrir proyecto y una lista de proyectos recientes.

### Ajustes

Generalmente se aconseja que inicies RStudio cada vez con una "pizarra limpia" - es decir, con tu espacio de trabajo **no** arrastrado de la sesión anterior. Esto significará que los objetos y resultados de una sesión no persistirán en la siguiente sesión (deberás volver a crearlos al ejecutar tus scripts). Esto es bueno, porque te obligará a escribir mejores scripts y evitar errores a largo plazo.

Para configurar RStudio para que haga "borrón y cuenta nueva" cada vez que se inicie:

* Selecciona "Project Options" en el menú Tools (Herramientas).
* En la pestaña "General", configura RStudio para que **no** restaure .RData en el espacio de trabajo al iniciar, y para que **no** guarde el espacio de trabajo en .RData al salir.

### Organización

Es habitual tener subcarpetas en tu proyecto. Piensa en tener carpetas como "datos", "scripts", "figuras", "presentaciones". Puedes añadir carpetas de la forma típica en que añadirías una nueva carpeta en tu ordenador. Alternativamente, puedes mirar en la página sobre [interacciones de directorios](#directory-interactions) para aprender a crear nuevas carpetas con los comandos de R.

### Control de versiones

Piensa en un sistema de control de versiones. Podría ser algo tan simple como tener fechas en los nombres de los scripts (por ejemplo, "transmission\_analysis\_2020-10-03.R") y una carpeta de "archivado". También es buena idea tener un texto de cabecera comentado en la parte superior de cada script con una descripción, etiquetas, autores y registro de cambios.

Un método más complicado implicaría utilizar Github o una plataforma similar para el control de versiones. Consulta la página sobre [Control de versiones y colaboración con Git y Github](#version-control-and-collaboration-with-).

Un consejo es que puedes buscar en todo un proyecto o carpeta utilizando la herramienta "Buscar en archivos" (menú Edición). Puedes buscar e incluso reemplazar cadenas en varios archivos.

## Ejemplos

A continuación se muestran algunos ejemplos de importación/exportación/guardado utilizando here() desde un proyecto R. Lea más sobre el uso del paquete here en la página de [importación y exportación](#import-and-export).

Importación de *linelist\_raw.xlsx* desde la carpeta "data" de tu proyecto R

Exportar linelist *de* objetos de R como "my\_linelist.rds" a la carpeta "clean" dentro de la carpeta "data" de tu proyecto R.

Guardando el último gráfico impreso como "epicurve\_2021-02-15.png" dentro de la carpeta "epicurves" en la carpeta "outputs" de tu proyecto R.

## Recursos

Página web de RStudio sobre [uso de proyectos R](https://support.rstudio.com/hc/en-us/articles/200526207-Using-Projects)

&&&&&

# #Importación y exportación

{#import-and-export}

En esta página describimos las formas de localizar, importar y exportar archivos:

* Uso del paquete **rio** para import() y export() de forma flexible muchos tipos de archivos
* Uso del paquete **here** para localizar archivos relativos a la raíz de un proyecto R - para evitar complicaciones de las rutas de los archivos que son específicas de un ordenador
* Escenarios específicos de importación, como:
  + Hojas de Excel específicas
  + Encabezados desordenados y filas que se saltan
  + Desde las hojas de Google
  + A partir de los datos publicados en los sitios web
  + Con las API
  + Importar el archivo más reciente
* Introducción manual de datos
* Tipos de archivos específicos de R, como RDS y RData
* Exportar/guardar archivos y gráficos

## Resumen

Cuando importas un "conjunto de datos" en R, generalmente estás creando un nuevo objeto dataframe en tu entorno de R y lo definirá como un archivo importado (por ejemplo, Excel, CSV, TSV, RDS) que se encuentra en tus directorios de carpetas en una determinada ruta/dirección.

Puedes importar/exportar muchos tipos de archivos, incluidos los creados por otros programas estadísticos (SAS, STATA, SPSS). También puedes conectarte a bases de datos relacionales.

R tiene incluso sus propios formatos de datos:

* Un archivo RDS (.rds) almacena un único objeto R, como un dataframe. Son útiles para almacenar datos limpios, ya que mantienen los tipos de columnas de R. Lee más en [esta sección](#import_rds).
* Un archivo RData (.Rdata) puede utilizarse para almacenar múltiples objetos, o incluso un espacio de trabajo R completo. Lee más en [esta sección](#import_rdata).

## El paquete ****rio****

El paquete de R que recomendamos es: **rio**. El nombre "rio" es una abreviatura de "R I/O" (input/output).

Sus funciones import() y export() pueden manejar muchos tipos de archivos diferentes (por ejemplo, .xlsx, .csv, .rds, .tsv). Cuando se proporciona una ruta de archivo a cualquiera de estas funciones (incluyendo la extensión del archivo como ".csv"), **rio** leerá la extensión y utilizará la herramienta correcta para importar o exportar el archivo.

La alternativa al uso de **rio** es utilizar funciones de muchos otros paquetes, cada uno de los cuales es específico para un tipo de archivo. Por ejemplo, read.csv() (R **base**), read.xlsx() (paquete **openxlsx**), y write\_csv() (**readr** pacakge), etc. Estas alternativas pueden ser difíciles de recordar, mientras que usar import() y export() de **rio** es fácil.

Las funciones de **rio** import() y export() utilizan el paquete y la función adecuados para un archivo determinado, basándose en su extensión. Al final de esta página puedes ver una tabla completa de los paquetes/funciones que utiliza **rio** en segundo plano. También puede utilizarse para importar archivos de STATA, SAS y SPSS, entre otras docenas de tipos de archivos.

La importación/exportación de shapefiles requiere otros paquetes, como se detalla en la página sobre [los fundamentos de SIG](#gis-basics).

## El paquete ****here****

El paquete **here** y su función here() facilitan la tarea de decirle a R dónde encontrar y guardar tus archivos - en esencia, construye rutas de archivos.

Utilizado junto con un proyecto R, **here te permite** describir la ubicación de los archivos en tu proyecto R en relación con el directorio raíz de los proyectos de R (la carpeta de nivel superior). Esto es útil cuando el proyecto R puede ser compartido o accedido por múltiples personas/ordenadores. Evita las complicaciones debidas a las rutas de archivos únicas en diferentes ordenadores (por ejemplo, "C:/Users/Laura/Documents..." al "comenzar" la ruta de archivos en un lugar común a todos los usuarios (la raíz del proyecto R).

Así es como funciona here() dentro de un proyecto R:

* Cuando el paquete **here se** carga por primera vez dentro del proyecto R, coloca un pequeño archivo llamado ".here" en la carpeta raíz de tu proyecto R como un "punto de referencia" o "ancla"
* En tus scripts, para referenciar un archivo en las subcarpetas del proyecto R, se utiliza la función here() para construir la ruta del archivo en relación con ese ancla
* Para construir la ruta de los archivos, escribe los nombres de las carpetas más allá de la raíz, entre comillas, separados por comas, y finalmente termina con el nombre y la extensión del archivo, como se muestra a continuación
* Las rutas de los archivos here() pueden utilizarse tanto para la importación como para la exportación

Por ejemplo, a continuación, la función import() recibe una ruta de archivo construida con here().

El comando aquí ("data", "linelists", "ebola\_linelist.xlsx") en realidad está proporcionando la ruta completa del archivo que es única para el ordenador del usuario:

"C:/Userss/Laura/Documents/my\_R\_proyect/data/llineists/ebola\_linelist.xlsx"

Lo bueno es que el comando que utiliza here() puede ejecutarse en cualquier ordenador que acceda al proyecto R.

**CONSEJO:** Si no estás seguro de dónde está la raíz ".here", ejecuta la función here() con los paréntesis vacíos.

Lee más sobre **este** paquete [en este enlace](https://here.r-lib.org/).

## Rutas de los archivos

Al importar o exportar datos, debes proporcionar una ruta de archivo. Puedes hacerlo de tres maneras:

1. Recomendado: proporcionar una ruta de archivo "relativa" con el paquete **here**
2. Proporcionar la ruta "completa" / "absoluta" del archivo
3. Seleccionar manualmente los archivos

### Rutas de archivos "relativas"

En R, las rutas de archivo "relativas" consisten en la ruta de archivo relativa a la raíz de un proyecto R. Permiten rutas de archivo más simples que pueden funcionar en diferentes ordenadores (por ejemplo, si el proyecto R está en una unidad compartida o se envía por correo electrónico). Como se ha descrito [anteriormente](#here) las rutas de archivo relativas se facilitan mediante el uso del paquete **here**.

A continuación se muestra un ejemplo de una ruta de archivo relativa construida con here(). Suponemos que el trabajo está en un proyecto R que contiene una subcarpeta "data" y dentro de ella una subcarpeta "linelists", en la que está el archivo .xlsx de interés.

### Rutas de archivos "absolutas"

Las rutas absolutas o "completas" de los archivos pueden proporcionarse a funciones como import(), pero son "frágiles", ya que son únicas para el ordenador específico del usuario y, por tanto, no se recomiendan.

A continuación se muestra un ejemplo de ruta absoluta de archivos, donde en el ordenador de Laura hay una carpeta "analysis", una subcarpeta "data" y dentro de ésta una subcarpeta "linelists", en la que se encuentra el archivo .xlsx de interés.

Hay que tener en cuenta algunas cosas sobre las rutas absolutas de los archivos:

* **Evita utilizar rutas absolutas de archivos, ya que** el script se parará si se ejecuta en un ordenador diferente
* Utilica barras inclinadas (/), como en el ejemplo anterior (nota: esto NO es el valor predeterminado para las rutas de archivos de Windows)
* Las rutas de archivos que comienzan con barras dobles (por ejemplo, "//...") probablemente **no serán reconocidas por R** y producirán un error. Considera la posibilidad de trasladar tu trabajo a una unidad "con nombre" o "con letra" que comience con una letra (por ejemplo, "J:" o "C:"). Consulta la página sobre [interacciones de directorios](#directory-interactions) para obtener más detalles sobre esta cuestión.

Un escenario en el que las rutas absolutas de los archivos pueden ser apropiadas es cuando se quiere importar un archivo desde una unidad compartida que tiene la misma ruta de archivo completa para todos los usuarios.

**SUGERENCIA:** Para convertir rápidamente todos los \ a /, resalte el código de interés, usa Ctrl+f (en Windows), marque la casilla de opción para "En selección", y luego usa la funcionalidad de reemplazo para convertirlos.

### Seleccionar manualmente el archivo

Puedes importar datos manualmente mediante uno de estos métodos:

1. En la pestaña de entorno de RStudio, clica en "Import Dataset", y selecciona el tipo de datos
2. Clica en File / Import dataset / (selecciona el tipo de datos)
3. Para codificar la selección manual, utiliza el comando de R base file.choose() (dejando los paréntesis vacíos) para provocar la aparición de una **ventana emergente** que permita al usuario seleccionar manualmente el archivo de su ordenador. Por ejemplo:

**CONSEJO:** La **ventana emergente** puede aparecer DETRÁS de la ventana de RStudio.

## Importar datos

Utilizar import() para importar unos datos es bastante sencillo. Simplemente escribe la ruta del archivo (incluyendo el nombre y la extensión del archivo) entre comillas. Si utilizas here() para construir la ruta del archivo, sigue las instrucciones anteriores. A continuación se muestran algunos ejemplos:

Importar un archivo csv que se encuentra en tu "directorio de trabajo" o en la carpeta raíz del proyecto R:

Importación de la primera hoja de un libro de Excel que se encuentra en las subcarpetas "data" y "linelists" del proyecto R (la ruta del archivo construida con here()):

Importar un dataframe (un archivo .rds) utilizando una ruta de archivo absoluta:

### Hojas de Excel específicas

Por defecto, si proporcionas un libro de Excel (.xlsx) a import(), se importará la primera hoja del libro. Si deseas importar una **hoja** específica, incluye el nombre de la hoja en el argumento which =. Por ejemplo:

Si utiliza el método here() para proporcionar una vía relativa a import(), puede seguir indicando una hoja específica añadiendo el argumento which = después del paréntesis de cierre de la función here().

Para exportar un dataframe de R a una hoja de Excel específica y que el resto del libro de Excel permanezca sin cambios, tendrás que importar, editar y exportar con un paquete alternativo destinado a este fin, como **openxlsx**. Vea más información en la página sobrelas [interacciones de directorios](#directory-interactions) o [en esta página de github](https://ycphs.github.io/openxlsx/).

Si tu libro de Excel es .xlsb (libro de Excel en formato binario) es posible que no puedas importarlo con **rio**. Considere la posibilidad de volver a guardarlo como .xlsx, o de utilizar un paquete como **readxlsb**, creado para [este fin](https://cran.r-project.org/web/packages/readxlsb/vignettes/read-xlsb-workbook.html).

### Valores faltantes

Es posible que desees designar qué valor(es) de tu set datos debe(n) considerarse como missing. Como se explica en la página sobre [Valores faltantes](#missing-data), el valor en R para los valores faltantes es NA, pero tal vez los datos que vas a importar utiliza 99, "Missing", o simplemente el espacio de caracteres vacíos "" en su lugar.

Utiliza el argumento na = para import() y proporcione el(los) valor(es) entre comillas (incluso si son números). Puedes especificar varios valores incluyéndolos dentro de un vector, utilizando c() como se muestra a continuación.

Aquí, el valor "99" en los datos importados se considera ausente y se convierte en NA en R.

Aquí, cualquiera de los valores "Missing", "" (celda vacía), o " " (un solo espacio) en los datos importados se convierten en NA en R.

### Saltar filas

A veces, puedes querer evitar la importación de una fila de datos. Puedes hacerlo con el argumento skip = si utiliza import() de **rio** en un archivo .xlsx o .csv. Proporciona el número de filas que deseas omitir.

Desafortunadamente, skip = sólo acepta un valor entero, no un rango (por ejemplo, "2:10" no funciona). Para omitir la importación de filas específicas que no son consecutivas desde el principio, considera la posibilidad de importar varias veces y utilizar bind\_rows() de **dplyr**. Mira en el ejemplo siguiente cómo se omite sólo la fila 2.

### Gestionar una segunda fila de cabecera

A veces, tus datos pueden tener una segunda fila, por ejemplo, si se trata de una fila de "diccionario de datos", como se muestra a continuación. Esta situación puede ser problemática porque puede hacer que todas las columnas se importen como de tipo "carácter".

A continuación se muestra un ejemplo de este tipo de conjunto de datos (en el que la primera fila es el diccionario de datos).

#### Eliminar la segunda fila de la cabecera

Para eliminar la segunda fila de la cabecera, tendrás que importar los datos dos veces.

1. Importar los datos para almacenar los nombres correctos de las columnas
2. Importe los datos de nuevo, saltándose las dos primeras filas (cabecera y segunda fila)
3. Vincular los nombres correctos en el dataframe reducido

El argumento exacto utilizado para vincular los nombres de las columnas correctas depende del tipo de archivo de datos (.csv, .tsv, .xlsx, etc.). Esto se debe a que **rio** utiliza una función diferente para los distintos tipos de archivos (véase la tabla anterior).

**Para los archivos de Excel:** (col\_names =)

**Para archivos CSV:** (col.names =)

**Opción de copia de seguridad** - cambiar los nombres de las columnas como un comando separado

#### Hacer un diccionario de datos

Bonificación! Si tienes una segunda fila que es un diccionario de datos, puedes crear fácilmente un diccionario de datos propio a partir de ella. Este consejo está adaptado de este [post](https://alison.rbind.io/post/2018-02-23-read-multiple-header-rows/).

#### Combinar las dos filas de la cabecera

En algunos casos, cuando los datos sin procesar tienen dos filas de cabecera (o, más concretamente, la segunda fila de datos es una cabecera secundaria), es posible que desee "combinarlas" o añadir los valores de la segunda fila de cabecera a la primera fila de cabecera.

El comando siguiente definirá los nombres de las columnas del dataframe como la combinación (pegada) de los primeros encabezados (verdaderos) con el valor inmediatamente inferior (en la primera fila).

### Hojas de cálculo de Google

Puedes importar datos de una hoja de cálculo de Google en línea con el paquete **googlesheet4** y autenticando tu acceso a la hoja de cálculo.

A continuación, se importa y guarda una hoja de Google de demostración. Este comando puede solicitar la confirmación de la autentificación de tu cuenta de Google. Sigue las indicaciones y las ventanas emergentes de tu navegador web para conceder a los paquetes de la API de Tidyverse permisos para editar, crear y eliminar sus hojas de cálculo en Google Drive.

La hoja que aparece a continuación es "visible para cualquiera con el enlace" y puedes intentar importarla.

La hoja también puede importarse utilizando sólo el ID de la hoja, una parte más corta de la URL:

Otro paquete, **googledrive** ofrece funciones útiles para escribir, editar y eliminar hojas de Google. Por ejemplo, utilizando las funciones gs4\_create() y sheet\_write() que se encuentran en este paquete.

Aquí hay otros tutoriales útiles en línea:  
[tutorial básico de importación de hojas de Google](https://arbor-analytics.com/post/getting-your-data-into-r-from-google-sheets/)  
[tutorial más detallado](https://googlesheets4.tidyverse.org/articles/googlesheets4.html)  
[interacción entre googlesheets4 y tidyverse](https://googlesheets4.tidyverse.org/articles/articles/drive-and-sheets.html)

## Múltiples archivos: importar, exportar, dividir, combinar

Consulta la página sobre [Iteración, bucles y listas](#iteration-loops-and-lists) para ver ejemplos de cómo importar y combinar múltiples archivos, o múltiples archivos de libros de Excel. Esa página también tiene ejemplos sobre cómo dividir un dataframe en partes y exportar cada una por separado, o como hojas específicas en un libro de Excel.

## Importar desde Github

Importar datos directamente de Github a R puede ser muy fácil o puede requerir algunos pasos - dependiendo del tipo de archivo. A continuación se presentan algunos enfoques:

### Archivos CSV

Es fácil importar un archivo .csv directamente desde Github a R con un comando de R.

1. Ve al repositorio de Github, localiza el archivo de interés y clica sobre él
2. Clica en el botón "Raw" (entonces verá los datos csv "crudos", como se muestra a continuación)
3. Copia la URL (dirección web)
4. Pega la URL entre comillas dentro del comando R import()

### Archivos XLSX

Es posible que no puedas ver los datos "en crudo" de algunos archivos (por ejemplo, .xlsx, .rds, .nwk, .shp)

1. Ve al repositorio de Github, localica el archivo de interés y clica sobre él
2. Clica en el botón "Download", como se muestra a continuación
3. Guarda el archivo en tu ordenador e impórtalo en R

### Shapefiles

Los Shapefiles tienen muchos archivos subcomponentes, cada uno con una extensión de archivo diferente. Un archivo tendrá la extensión ".shp", pero otros tienen ".dbf", ".prj", etc. Para descargar un shapefile de Github, tendrás que descargar cada uno de los archivos subcomponentes individualmente, y guardarlos en la misma carpeta de tu ordenador. En Github, clica en cada archivo individualmente y descárgalos clicando en el botón "Download".

Una vez guardado en tu ordenador, puedes importar el archivo shape como se muestra en la página de [fundamentos de SIG](#gis-basics) utilizando st\_read() del paquete **sf**. Sólo tienes que proporcionar la ruta del archivo y el nombre del archivo ".shp", siempre que los demás archivos relacionados estén en la misma carpeta de tu ordenador.

A continuación, se puede ver cómo el shapefile "sle\_adm3" consta de muchos archivos, cada uno de los cuales debe descargarse de Github.

## Introducción manual de datos

### Entrada por filas

Utiliza la función tribble del paquete **tibble** del tidyverse ([referencia online de tibble](https://tibble.tidyverse.org/reference/tribble.html)).

Observa que las cabeceras de las columnas comienzan con una tilde (~). Observa también que cada columna debe contener sólo un tipo de datos (carácter, numérico, etc.). Puedes utilizar tabulaciones, espacios y nuevas filas para que la entrada de datos sea más intuitiva y legible. Los espacios no importan entre los valores, pero cada fila está representada por una nueva línea de código. Por ejemplo:

Y ahora mostramos el nuevo conjunto de datos:

### Entrada por columnas

Dado que un dataframe consiste en vectores (columnas verticales), el enfoque básico para la creación manual de dataframes en R espera que definas cada columna y luego las unas. Esto puede ser contrario a la intuición en epidemiología, ya que normalmente pensamos en nuestros datos en filas (como arriba).

**ATENCIÓN:** Todos los vectores deben tener la misma longitud (el mismo número de valores).

A continuación, los vectores pueden unirse mediante la función data.frame():

Y ahora mostramos el nuevo conjunto de datos:

### Pegar desde el portapapeles

Si copias los datos de otro lugar y los tienes en el portapapeles, puedes probar una de las dos formas siguientes:

Con el paquete **clipr**, puedes utilizar read\_clip\_tbl() para importar como un dataframe, o simplemente read\_clip() para importar como un vector de caracteres. En ambos casos, deja los paréntesis vacíos.

También puedes exportar fácilmente al portapapeles de tu sistema con **clipr.** Consulta la sección siguiente sobre Exportación.

Alternativamente, pueder utilizar la función read.table() de R **base** con file = "clipboard") para importar como un dataframe:

## Importar el archivo más reciente

A menudo puedes recibir actualizaciones diarias de tus datos. En este caso, querrás escribir un código que importe el archivo más reciente. A continuación presentamos dos maneras de abordar esto:

* Seleccionar el archivo en función de la fecha del nombre del archivo
* Seleccionar el archivo en función de los metadatos del archivo (última modificación)

### Fechas en el nombre del archivo

Este enfoque se basa en tres premisas:

1. Confía en las fechas de los nombres de los archivos
2. Las fechas son numéricas y aparecen generalmente en el mismo formato (por ejemplo, año, mes y día)
3. No hay otros números en el nombre del archivo

Te explicaremos cada paso y te los mostraremos combinados al final.

En primer lugar, utiliza dir() de R **base** para extraer sólo los nombres de los archivos de la carpeta de interés. Consulta la página sobre [interacciones con directorios](#directory-interactions) para obtener más detalles sobre dir(). En este ejemplo, la carpeta de interés es la carpeta "linelists" dentro de la carpeta "example" dentro de "data" dentro del proyecto R.

Una vez que tengas este vector de nombres, puede extraer las fechas de ellos aplicando str\_extract() de **stringr** utilizando esta expresión regular. Extrae cualquier número en el nombre del archivo (incluyendo cualquier otro carácter en el medio como guiones o barras). Puedes leer más sobre **stringr** en la página [Cadenas y caracteres].

Suponiendo que las fechas estén escritas en general con el mismo formato de fecha (por ejemplo, Año, Mes y Día) y que los años tengan 4 dígitos, puedes utilizar las funciones de conversión de **lubridate** (ymd(), dmy() o mdy()) para convertirlas en fechas. Para estas funciones, no importan los guiones, espacios o barras, sino el orden de los números. Lee más en la página [Trabajar con fechas](#working-with-dates-1).

La función **base** de R que.max() puede utilizarse para devolver la posición del índice (por ejemplo, 1ª, 2ª, 3ª, ...) del valor máximo de la fecha. El último archivo se identifica correctamente como el 6º archivo - "case\_linelist\_2020-10-08.xlsx".

Si condensamos todos estos comandos, el código completo podría ser como el siguiente. Observa que el . en la última línea es un marcador de posición para el objeto canalizado en ese punto de la secuencia de tuberías. En ese punto el valor es simplemente el número 6. Esto se coloca entre corchetes dobles para extraer el 6º elemento del vector de nombres de archivo producido por dir().

Ahora puede utilizar este nombre para terminar la ruta relativa del archivo, con here():

Y ahora puede importar el último archivo:

### Utiliza la información del archivo

Si tus archivos no tienen fechas en sus nombres (o no te fías de esas fechas), puedes intentar extraer la última fecha de modificación de los metadatos del archivo. Utiliza las funciones del paquete **fs** para examinar la información de los metadatos de cada archivo, que incluye la hora de la última modificación y la ruta del archivo.

A continuación, proporcionamos la carpeta de interés a dir\_info() de **fs**. En este caso, la carpeta de interés está en el proyecto R en la carpeta "data", la subcarpeta "example", y su subcarpeta "linelists". El resultado es un dataframe con una línea por archivo y columnas para tiempo\_de\_modificación, ruta, etc. Puedes ver un ejemplo visual de esto en la página sobre [interacciones de directorios](#directory-interactions).

Podemos ordenar este dataframe de archivos por la columna modification\_time, y luego mantener sólo la fila superior/última (archivo) con la función head() de R **base**. A continuación, podemos extraer la ruta de este último archivo sólo con la función **dplyr** pull() en la ruta de la columna. Finalmente podemos pasar esta ruta de archivo a import(). El archivo importado se guarda como latest\_file.

## APIs

Una "Interfaz de Programación Automatizada" (API) puede utilizarse para solicitar directamente datos de un sitio web. Las API son un conjunto de reglas que permiten que una aplicación de software interactúe con otra. El cliente (tu) envía una "solicitud" y recibe una "respuesta" con contenido. Los paquetes de R **httr** y **jsonlite** pueden facilitar este proceso.

Cada sitio web habilitado para la API tendrá su propia documentación y detalles con los que hay que familiarizarse. Algunos sitios están disponibles públicamente y cualquiera puede acceder a ellos. Otros, como las plataformas con ID de usuario y credenciales, requieren autenticación para acceder a sus datos.

Ni que decir tiene que es necesario disponer de una conexión a Internet para importar datos a través de la API. Daremos brevemente ejemplos de uso de las API para importar datos, y enlazaremos con otros recursos.

Nota: recuerda que los datos pueden estar publicados\* en un sitio web sin una API, que puede ser más fácil de recuperar. Por ejemplo, un archivo CSV publicado puede ser accesible simplemente proporcionando la URL del sitio a import() como se describe en la sección sobre la [importación desde Github](#import_github).\*

### Petición HTTP

El intercambio de la API se realiza normalmente a través de una solicitud HTTP. HTTP es el Protocolo de Transferencia de Hipertexto, y es el formato subyacente de una solicitud/respuesta entre un cliente y un servidor. La entrada y la salida exactas pueden variar en función del tipo de API, pero el proceso es el mismo: una "Solicitud" (a menudo Solicitud HTTP) del usuario, que suele contener una consulta, seguida de una "Respuesta", que contiene información de estado sobre la solicitud y posiblemente el contenido solicitado.

Estos son algunos de los componentes de una petición HTTP:

* La URL completa de la API
* El "Método" (o "Verbo")
* Headers (Encabezados)
* Body (Cuerpo)

El "método" de la petición HTTP es la acción que se quiere realizar. Los dos métodos HTTP más comunes son GET y POST, pero otros pueden ser PUT, DELETE, PATCH, etc. Cuando se importan datos a R lo más probable es que se utilice GET.

Después de la solicitud, tu ordenador recibirá una "respuesta" en un formato similar al que se envió, incluyendo la URL, el estado HTTP (¡status 200 es lo que quieres!), el tipo de archivo, el tamaño y el contenido deseado. A continuación, tendrá que analizar esta respuesta y convertirla en un dataframe viable dentro de su entorno R.

### Paquetes

El paquete **httr** funciona bien para manejar peticiones HTTP en R. Requiere poco conocimiento previo de las APIs de la web y puede ser utilizado por personas menos familiarizadas con la terminología de desarrollo de software. Además, si la respuesta HTTP es .json, puede utilizar **jsonlite** para analizar la respuesta.

### Datos de acceso público

A continuación se muestra un ejemplo de solicitud HTTP, tomado de un tutorial [de Trafford Data Lab](https://www.trafforddatalab.io/open_data_companion/" \l "A_quick_introduction_to_APIs). Este sitio tiene varios otros recursos para aprender y ejercicios de API.

Escenario: Queremos importar una lista de establecimientos de comida rápida en la ciudad de Trafford, Reino Unido. Se puede acceder a los datos desde la API de la Food Standards Agency (Agencia de Normas Alimentarias), que proporciona datos de calificación de higiene alimentaria para el Reino Unido.

Estos son los parámetros de nuestra solicitud:

* Verbo HTTP: GET
* URL del punto de la API: <http://api.ratings.food.gov.uk/Establishments>
* Parámetros seleccionados: nombre, dirección, longitud, latitud, businessTypeId, ratingKey, localAuthorityId
* Cabeceras: "x-api-version", 2
* Formato(s) de datos: JSON, XML
* Documentación: <http://api.ratings.food.gov.uk/help>

El código R sería el siguiente:

Ahora puede limpiar y utilizar el dataframe de respuesta, que contiene una fila por establecimiento de comida rápida.

### Se requiere autenticación

Algunas APIs requieren autenticación - para que se demuestre quién eres, para acceder a datos restringidos. Para importar estos datos, es posible que tengas que utilizar primero un método POST para proporcionar un nombre de usuario, una contraseña o un código. Esto devolverá un token de acceso, que puede ser utilizado para posteriores solicitudes del método GET para recuperar los datos deseados.

A continuación se muestra un ejemplo de consulta de datos de Go.Data, que es una herramienta de investigación de brotes. Go.Data utiliza una API para todas las interacciones entre el front-end de la web y las aplicaciones de los smartphones utilizadas para la recogida de datos. Go.Data se utiliza en todo el mundo. Dado que los datos de los brotes son sensibles y sólo debes poder acceder a los datos de tu brote, se requiere autenticación.

A continuación se muestra un ejemplo de código R que utiliza **httr** y **jsonlite** para conectarse a la API de Go.Data para importar datos sobre el seguimiento de los contactos de tu brote.

**PRECAUCIÓN:** Si estásimportando grandes cantidades de datos desde una API que requiere autenticación, es posible que se agote el tiempo de espera. Para evitarlo, recupera el access\_token antes de cada solicitud GET de la API y prueba a utilizar filtros o límites en la consulta.

**CONSEJO:** La función fromJSON() del paquete **jsonlite** no se desmonta completamente la primera vez que se ejecuta, por lo que es probable que todavía tengas elementos de la lista en tu tibble resultante. Tendrás que desmontar aún más ciertas variables, dependiendo de lo anidado que esté tu .json. Para ver más información sobre esto, consulte la documentación del paquete **jsonlite**, como la [función flatten()](https://rdrr.io/cran/jsonlite/man/flatten.html).

Para más detalles, mira la documentación en el [Explorador de LoopBack](https://loopback.io/doc/en/lb4/index.html), la página de [Rastreo de Contactos](#contact-tracing-1) o los consejos de la API en [el repositorio Github de Go.Data](https://worldhealthorganization.github.io/godata/api-docs)

Puedes leer más sobre el paquete httr [aquí](https://httr.r-lib.org/articles/quickstart.html)

Esta sección también se inspiró en [este tutorial](https://www.dataquest.io/blog/r-api-tutorial/) y [este tutorial](https://medium.com/@traffordDataLab/querying-apis-in-r-39029b73d5f1).

## Exportar

### Con el paquete ****rio****

Con **rio**, puedes utilizar la función export() de forma muy similar a import(). Primero indica el nombre del objeto de R que desea guardar (por ejemplo, linelist) y luego pon entre comillas la ruta de acceso al archivo donde deseas guardarlo, incluyendo el nombre y la extensión de archivo deseados. Por ejemplo:

Esto guarda linelist del dataframe como un libro de Excel en el directorio de trabajo/carpeta raíz del proyecto:

Se puede guardar el mismo dataframe como un archivo csv cambiando la extensión. Por ejemplo, también lo guardamos en una ruta de archivo construida con here():

### Al portapapeles

Para exportar un dataframe al "portapapeles" de tu ordenador (para luego pegarlo en otro software como Excel, Google Spreadsheets, etc.) puedes utilizar write\_clip() del paquete **clipr**.

## Archivos RDS

Además de .csv, .xlsx, etc., también puedes exportar/guardar dataframes de R como archivos .rds. Este es un formato de archivo específico de R, y es muy útil si sabes que vas a trabajar con los datos exportados de nuevo en R.

Se almacenan los tipos de columnas, por lo que no hay que volver a hacer la limpieza cuando se importan (con un archivo Excel o incluso CSV esto puede ser un dolor de cabeza). También es un archivo más pequeño, lo que es útil para la exportación e importación si tu conjunto de datos es grande.

Por ejemplo, si trabajas en un equipo de Epidemiología y necesitas enviar archivos a un equipo de SIG para la elaboración de mapas, y ellos también utilizan R, ¡sólo tienes que enviarles el archivo .rds! Así se conservan todas los tipos de columnas y ellos tienen menos trabajo que hacer.

## Archivos y listas de datos

Los archivos .Rdata pueden almacenar múltiples objetos de R - por ejemplo, múltiples dataframes, resultados de modelos, listas, etc. Esto puede ser muy útil para consolidar o compartir muchos de tus datos para un proyecto determinado.

En el siguiente ejemplo, se almacenan múltiples objetos R dentro del archivo exportado "mis\_objetos.Rdata":

Nota: si estás intentando importar una lista, utiliza import\_list() de **rio** para importarla con la estructura y el contenido originales completos.

## Guardar los gráficos

Las instrucciones sobre cómo guardar los gráficos, como los creados por ggplot(), se discuten en profundidad en la página de [fundamentos de ggplot](#ggplot-basics).

En resumen, ejecuta ggsave("mi\_ruta\_de\_archivo\_y\_nombre.png") después de obtener tu gráfico. Puedes proporcionar un objeto de trazado guardado plot = argumento, o sólo especificar la ruta de archivo de destino (con extensión de archivo) para guardar el gráfico mostrado más recientemente. También puedes controlar la anchura widt =, la altura height =, las unidades unitss = y los puntos por pulgada dpi =.

La forma de guardar un gráfico de red, como un árbol de transmisión, se aborda en la página [Cadenas de transmisión](#transmission-chains).

## Recursos

El [manual de importación y exportación de datos de R](https://cran.r-project.org/doc/manuals/r-release/R-data.html)  
[Capítulo de R 4 Data Science sobre la importación de datos](https://r4ds.had.co.nz/data-import.html" \l "data-import)  
[documentación de ggsave()](https://ggplot2.tidyverse.org/reference/ggsave.html)

A continuación se muestra una tabla, extraída de la [viñeta](https://cran.r-project.org/web/packages/rio/vignettes/rio.html) online de **rio**. Para cada tipo de datos muestra: la extensión de archivo esperada, el paquete que **rio** utiliza para importar o exportar los datos, y si esta funcionalidad está incluida en la versión instalada por defecto de **rio**.

| **Formato** | **Extensión típica** | **Paquete de importación** | **Paquete de exportación** | **Instalado por defecto** |
| --- | --- | --- | --- | --- |
| Datos separados por comas | .csv | data.table fread() | data.table | Sí |
| Datos separados por pipe (|) | .psv | data.table fread() | data.table | Sí |
| Datos separados por tab | .tsv | data.table fread() | data.table | Sí |
| SAS | .sas7bdat | haven | haven | Sí |
| SPSS | .sav | haven | haven | Sí |
| Stata | .dta | haven | haven | Sí |
| SAS | XPORT | .xpt | haven | haven |
| SPSS portátil | .por | haven |  | Sí |
| Excel | .xls | readxl |  | Sí |
| Excel | .xlsx | readxl | openxlsx | Sí |
| Sintaxis R | .R | base | base | Sí |
| Objetos R guardados | .RData, .rda | base | base | Sí |
| Objetos R serializados | .rds | base | base | Sí |
| Epiinfo | .rec | foreign |  | Sí |
| Minitab | .mtp | foreign |  | Sí |
| Systat | .syd | foreign |  | Sí |
| "XBASE" | archivos de la base de datos | .dbf | foreign | foreign |
| Formato de archivo de relación de atributos de Weka | .arff | foreign | foreign | Sí |
| Formato de intercambio de datos | .dif | utils |  | Sí |
| Datos de Fortran | ninguna extensión reconocida | utils |  | Sí |
| Datos con formato de ancho fijo | .fwf | utils | utils | Sí |
| gzip datos separados por comas | .csv.gz | utils | utils | Sí |
| CSVY (CSV + cabecera de metadatos YAML) | .csvy | csvy | csvy | No |
| EViews | .wf1 | hexView |  | No |
| Formato de intercambio Feather R/Python | .feather | feather | feather | No |
| Almacenamiento rápido | .fst | fst | fst | No |
| JSON | .json | jsonlite | jsonlite | No |
| Matlab | .mat | rmatio | rmatio | No |
| Hoja de cálculo OpenDocument | .ods | readODS | readODS | No |
| Tablas HTML | .html | xml2 | xml2 | No |
| Documentos XML superficiales | .xml | xml2 | xml2 | No |
| YAML | .yml | yaml | yaml | No |
| El portapapeles por defecto es tsv |  | clipr | clipr | No |

&&&&&

# Gestión de datos

# #Limpieza de datos y funciones básicas

{#cleaning-data-and-core-functions}

Esta página muestra los pasos más comunmente utilizados en el proceso de "limpieza" de unos datos, y también explica el uso de muchas funciones esenciales de gestión de datos de R.

Para mostrarlo, esta página comienza importando unos datos de un listado de casos crudo, y se avanza paso a paso a través del proceso de limpieza. En el código R, esto se manifiesta como una cadena de "tuberías", que hacen referencia al operador "tuberías" %>% que pasa unos datos de una operación a la siguiente.

### Funciones principales

Este manual hace hincapié en el uso de las funciones de la familia de paquetes de R [**tidyverse**](https://www.tidyverse.org/). Las funciones esenciales que se muestran en esta página se enumeran a continuación.

Muchas de estas funciones pertenecen al paquete [**dplyr**](https://dplyr.tidyverse.org/), que proporciona funciones "verbales" para resolver los retos de la manipulación de datos (el nombre hace una referencia a unos alicates - [plier](https://www.thefreedictionary.com/plier" \l ":~:text=also ply·er (plī′,holding%2C bending%2C or cutting.)") - de dataframes). **dplyr** forma parte de la familia de paquetes de R **tidyverse** (que también incluye **ggplot2**, **tidyr**, **stringr**, **tibble**, **purrr**, **magrittr** y **forcats**, entre otros).

| **Función** | **Utilidad** | **Paquete** |
| --- | --- | --- |
| [%>%](https://magrittr.tidyverse.org/reference/pipe.html) | "canalizar" (pasar) datos de una función a la siguiente | **magrittr** |
| [mutate()](https://dplyr.tidyverse.org/reference/mutate.html) | crear, transformar y redefinir columnas | **dplyr** |
| [selecct()](https://dplyr.tidyverse.org/reference/select.html) | mantener, eliminar, seleccionar o renombrar columnas | **dplyr** |
| [rename()](https://dplyr.tidyverse.org/reference/rename.html) | cambiar el nombre de las columnas | **dplyr** |
| clean\_names() | estandarizar la sintaxis de los nombres de las columnas | **janitor** |
| [as.character()](https://rdrr.io/r/base/character.html), [as.numeric()](https://rdrr.io/r/base/numeric.html), [as.Date()](https://rdrr.io/r/base/as.Date.html), etc. | convertir el tipo de una columna | **base** R |
| across() | transformar varias columnas a la vez | **dplyr** |
| funciones **tidyselect** | utilizar la lógica para seleccionar las columnas | **tidyselect** |
| [filter()](https://dplyr.tidyverse.org/reference/filter.html) | mantener ciertas filas | **dplyr** |
| [distinct()](https://dplyr.tidyverse.org/reference/distinct.html) | des-duplicar filas | **dplyr** |
| rowwise() | operaciones por/en cada fila | **dplyr** |
| add\_row() | añadir filas manualmente | **tiblle** |
| [arrange()](https://dplyr.tidyverse.org/reference/arrange.html) | ordenar las filas | **dplyr** |
| recode() | recodificar los valores de una columna | **dplyr** |
| case\_when() | recodificar los valores de una columna con criterios lógicos más complejos | **dplyr** |
| replace\_na(), na\_if(), coalesce() | funciones especiales de recodificación | **tidyr** |
| age\_categories() y [cut()](https://rdrr.io/r/base/cut.html) | crear grupos categóricos a partir de una columna numérica | **epikit** y **base** R |
| clean\_variable\_spelling() | recodificación/limpieza de valores mediante un diccionario de datos | **lista de líneas** |
| [which()](https://rdrr.io/r/base/which.html) | aplicar los criterios lógicos; devolver los índices | **base** R |

Si quieres ver cómo se comparan estas funciones con los comandos de Stata o SAS, consulta la página sobre la [transición a R](#transition-to-r).

Puedes encontrar una gestión de datos alternativa en el paquete R **data.table** con operadores como := y el uso frecuente de corchetes [ ]. Este enfoque y la sintaxis se explican brevemente en la página [data.table](https://epirhandbook.com/data-table.html" \l "data-table).

### Nomenclatura

En este manual, generalmente hacemos referencia a "columnas" y "filas" en lugar de "variables" y "observaciones". Como se explica en este manual sobre ["datos ordenados"](https://tidyr.tidyverse.org/articles/tidy-data.html), la mayoría de los conjuntos de datos estadísticos epidemiológicos se componen estructuralmente de filas, columnas y valores.

Las variables contienen los valores que miden el mismo atributo subyacente (como el grupo de edad, el resultado o la fecha de inicio). Las observaciones contienen todos los valores medidos en la misma unidad (por ejemplo, una persona, un lugar o una muestra de laboratorio). Por lo tanto, estos aspectos pueden ser más difíciles de definir de forma tangible.

En los conjuntos de datos "ordenados", cada columna es una variable, cada fila es una observación y cada celda es un único valor. Sin embargo, algunos conjuntos de datos que se encuentran no se ajustan a este molde: unos datos de formato "amplio" puede tener una variable dividida en varias columnas (véase un ejemplo en la página [Pivotar datos](#pivoting-data)). Del mismo modo, las observaciones pueden estar divididas en varias filas.

La mayor parte de este manual trata sobre la gestión y la transformación de datos, por lo que las referencias a las estructuras de datos concretas de filas y columnas son más relevantes que las observaciones y las variables más abstractas. Las excepciones se dan sobre todo en las páginas sobre análisis de datos, en las que verás más referencias a las variables y las observaciones.

## Limpieza de tuberías

**Esta página recorre los pasos típicos de limpieza, añadiéndolos secuencialmente a una cadena de tuberías de limpieza.**

En el análisis epidemiológico y el procesamiento de datos, los pasos de limpieza se realizan a menudo de forma secuencial, enlazados entre sí. En R, esto se manifiesta a menudo como una "tubería" de limpieza, en la que los datos en bruto se pasan o se "canalizan" de un paso de limpieza a otro.

Estas cadenas utilizan las funciones de **dplyr** y el operador %>% **de magrittr**. Esta tubería comienza con los datos "en bruto" ("linelist\_raw.xlsx") y termina con un dataframe de R "limpio" (linelist) que se puede utilizar, guardar, exportar, etc.

En un proceso de limpieza, el orden de los pasos es importante. Los pasos de limpieza pueden incluir:

* Importación de datos
* Limpieza o cambio de los nombres de las columnas
* Des-duplicación
* Creación y transformación de columnas (por ejemplo, recodificación o normalización de valores)
* Filtrado o añadido de filas

## Carga de paquetes

Este trozo de código muestra la carga de los paquetes necesarios para el análisis. En este manual destacamos p\_load() de **pacman**, que instala el paquete si es necesario y lo carga para su uso. También puede cargar los paquetes instalados con library() de R **base.** Consulta la página sobre [Aspectos generales de R](#r-basics) para obtener más información sobre los paquetes de R.

## Importar datos

### Importar

Aquí importamos el archivo Excel de la lista de casos "en bruto" utilizando la función import() del paquete **rio**. El paquete **rio** maneja con flexibilidad muchos tipos de archivos (por ejemplo, .xlsx, .csv, .tsv, .rds. Consulta la página sobre [importación y exportación](#import-and-export) para obtener más información y consejos sobre situaciones inusuales (por ejemplo, omitir filas, establecer valores que faltan, importar hojas de Google, etc.).

Si quieres seguirlo, [clica para descargar linelist "en crudo"](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/case_linelists/linelist_raw.xlsx) (como archivo .xlsx).

Si tu conjunto de datos es grande y tarda mucho en importarse, puede ser útil que el comando de importación esté separado de la cadena de tuberías y que el "crudo" se guarde como un archivo distinto. Esto también permite comparar fácilmente las versiones original y limpia.

A continuación importamos el archivo de Excel sin procesar y lo guardamos como el dataframe linelist\_raw. Asumimos que el archivo se encuentra en tu directorio de trabajo o en la raíz del proyecto R, por lo que no se especifican subcarpetas en la ruta del archivo.

Puedes ver las primeras 50 filas del dataframe a continuación. Nota: la función **base** de R head(n) te permite ver sólo las primeras n filas en la consola de R.

### Revisar

Puedes utilizar la función skim() del paquete **skimr** para obtener una visión general de todo el dataframe (véase la página sobre [tablas descriptivas](#descriptive-tables) para más información). Las columnas se resumen por tipo, como carácter, numérico. Nota: "POSIXct" es un tipo de fecha cruda (ver [Trabajar con fechas](#working-with-dates-1).

## Nombres de columnas

En R, los nombres de las columnas son la "cabecera" o el valor "superior" de una columna. Se utilizan para referirse a las columnas en el código, y sirven como etiqueta por defecto en las figuras.

Otros programas estadísticos, como SAS y STATA, utilizan "etiquetas" que coexisten como versiones impresas más largas de los nombres de columna más cortos. Aunque R ofrece la posibilidad de añadir etiquetas de columna a los datos, en la práctica no se hace hincapié en ello. Para hacer que los nombres de las columnas sean "fáciles de imprimir" para las figuras, normalmente se ajusta su visualización dentro de los comandos de gráficas que crean las salidas (por ejemplo, los títulos de los ejes o de las leyendas de una gráfica, o las cabeceras de las columnas en una tabla impresa - véase la [sección de escalas de la página de consejos de ggplot](#ggplot_tips_scales) y las páginas de [Tablas para la presentación](#tables-for-presentation)). Si deseas asignar etiquetas de columna en los datos, lee más online [aquí](https://cran.r-project.org/web/packages/expss/vignettes/labels-support.html) y [aquí](https://cran.r-project.org/web/packages/labelled/vignettes/intro_labelled.html).

Como los nombres de las columnas de R se utilizan con mucha frecuencia, deben tener una sintaxis "limpia". Sugerimos lo siguiente:

* Nombres cortos
* Sin espacios (sustituir por barras bajas \_ )
* No hay caracteres inusuales (&, #, <, >, ...)
* Nomenclatura de estilo similar (por ejemplo, todas las columnas de fecha nombradas como **date\_onset**, **date\_report**, **date\_death...**)

Los nombres de las columnas de linelist\_raw se muestran a continuación utilizando names() de R **base**. Podemos ver que inicialmente

* Algunos nombres contienen espacios (por ejemplo, infection date)
* Se utilizan diferentes patrones de nomenclatura para las fechas (date onset vs. infection date)
* Debe haber habido una cabecera fusionada en las dos últimas columnas del .xlsx. Lo sabemos porque el nombre de dos columnas fusionadas ("merged\_header") fue asignado por R a la primera columna, y a la segunda columna se le asignó un nombre de marcador de posición "...28" (ya que entonces estaba vacía y es la columna 28).

**NOTA:** Para hacer referencia a un nombre de columna que incluya espacios, rodee el nombre con tildes, por ejemplo: linelist$` '\x60infection date\x60'`. Ten en cuenta que en tu teclado, la tilde (`) es diferente de la comilla simple (').

### Etiquetas

Algunos otros programas estadísticos, como SAS, tienen etiquetas de variables

### Limpieza automática

La función clean\_names() del paquete **janitor** estandariza los nombres de las columnas y los hace únicos haciendo lo siguiente:

* Convierte todos los nombres para que estén compuestos sólo por barras bajas, números y letras
* Los caracteres acentuados se transliteran a ASCII (por ejemplo, la o alemana con diéresis se convierte en "o", la "ñ" española se convierte en "n")
* Se puede especificar la preferencia de mayúsculas para los nuevos nombres de columna utilizando case = argumento ("snake" es el valor por defecto, las alternativas incluyen “sentence”, “title”, “small\_camel”...)
* Puedes especificar sustituciones de nombres concretos proporcionando un vector a replace = argumento (por ejemplo, replace = c(onset = "date\_of\_onset")
* Esta es una [viñeta](https://cran.r-project.org/web/packages/janitor/vignettes/janitor.html" \l "cleaning) en línea

A continuación, el proceso de limpieza comienza utilizando clean\_names() en sobre linelist\_raw.

**NOTA:** El nombre de la última columna "...28" se ha cambiado por "x28".

### Limpieza manual de nombres

A menudo es necesario renombrar las columnas manualmente, incluso después del paso de estandarización anterior. A continuación, el renombramiento se realiza utilizando la función rename() del paquete **dplyr**, como parte de una cadena de tuberías. rename() utiliza el estilo NUEVO = ANTIGUO - el nombre nuevo de la columna se escribe antes que el antiguo.

A continuación, se añade un comando de renombramiento a la tubería de limpieza. Se han añadido espacios estratégicamente para alinear el código y facilitar la lectura.

Ahora puedes ver que los nombres de las columnas han cambiado:

#### Renombrar por posición de columna

También puedes renombrar por la posición de la columna, en lugar del nombre de la columna, por ejemplo:

#### Renombrar mediante select() y summarise()

Como método abreviado, también puedes cambiar el nombre de las columnas dentro de las funciones de **dplyr** select() y summarise(). select() se utiliza para mantener sólo ciertas columnas (y se muestra más adelante en esta página). summarise() se muestra en las páginas [Agrupar datos](#grouping-data) y [Tablas descriptivas](#descriptive-tables). Estas funciones también utilizan el formato nombre\_nuevo = nombre\_antiguo. He aquí un ejemplo:

### Otros retos

#### Nombres de columnas de Excel vacíos

R no puede tener columnas de conjuntos de datos que no tengan nombres de columnas (cabeceras). Así, si importa unos datos de Excel con datos pero sin cabeceras de columna, R rellenará las cabeceras con nombres como "...1" o "...2". El número representa el número de la columna (por ejemplo, si la cuarta columna de los datos no tiene cabecera, R la nombrará "...4").

Puedes limpiar estos nombres manualmente haciendo referencia a su número de posición (véase el ejemplo anterior), o a su nombre asignado (linelist\_raw$...1).

#### Nombres de columnas y celdas fusionadas de Excel

Las celdas combinadas en un archivo de Excel son una ocurrencia común cuando se reciben datos. Como se explica en [Transición a R](#transition-to-r), las celdas combinadas pueden ser agradables para la lectura humana de los datos, pero no son "datos ordenados" y causan muchos problemas para la lectura de los datos por parte de las máquinas. R no puede ajustar las celdas combinadas.

Recuerda a las personas que introducen los datos que **los datos legibles para el ser humano no son lo mismo que los datos legibles para la máquina**. Esfuérzate en formar a los usuarios sobre los principios de los [**datos ordenados**](https://r4ds.had.co.nz/tidy-data.html). Si es posible, intenta cambiar los procedimientos para que los datos lleguen en un formato ordenado y sin celdas fusionadas.

* Cada variable debe tener su propia columna.
* Cada observación debe tener su propia fila.
* Cada valor debe tener su propia celda.

Al utilizar la función import() de **rio**, el valor de una celda combinada se asignará a la primera celda y las siguientes estarán vacías.

Una solución para tratar las celdas combinadas es importar los datos con la función readWorkbook() del paquete **openxlsx**. Establece el argumento fillMergedCells = TRUE. Esto da el valor en una celda fusionada a todas las celdas dentro del rango de fusión.

**PELIGRO:** Si los nombres de las columnas se fusionan con readWorkbook(), terminarás con nombres de columnas duplicados, que tendrás que arreglar manualmente - ¡R no funciona bien con nombres de columnas duplicados! Puedes renombrarlas haciendo referencia a su posición (por ejemplo, la columna 5), como se explica en la sección de limpieza manual de nombres de columnas.

## Seleccionar o reordenar columnas

Utiliza select() de **dplyr** para seleccionar las columnas que deseas conservar y para especificar su orden en el dataframe.

**ATENCIÓN:** En los ejemplos siguientes, el dataframe linelist se modifica con select() y se muestra, pero no se guarda. Esto es a efectos de demostración. Los nombres de las columnas modificadas se imprimen pasando el dataframe a names().

**Aquí están TODOS los nombres de las columnas en linelist en este punto de la cadena de limpieza:**

### Mantener las columnas

**Selecciona sólo las columnas que desees conservar**

Escribe sus nombres en el comando select(), sin comillas. Aparecerán en el dataframe en el orden que indiques. Ten en cuenta que si incluyes una columna que no existe, R devolverá un error (véase el uso de any\_of() más adelante si no deseas ningún error en esta situación).

### Funciones de ayuda "tidyselect".

Estas funciones de ayuda existen para facilitar la especificación de las columnas a conservar, descartar o transformar. Provienen del paquete **tidyselect**, que se incluye en **tidyverse** y se basa en la forma en que se seleccionan las columnas en las funciones de **dplyr**.

Por ejemplo, si deseas reordenar las columnas, everything() es una función útil para indicar "todas las demás columnas no mencionadas". El comando siguiente mueve las columnas date\_onset y date\_hospitalisation al principio (izquierda) de los datos, pero mantiene todas las demás columnas después. Fíjate en que verything() se escribe con paréntesis vacíos:

Aquí hay otras funciones de ayuda "tidyselect" que también funcionan dentro de las funciones de **dplyr** como select(), across() y summarise():

* everything() - todas las demás columnas no mencionadas
* last\_col() - la última columna
* where() - aplica una función a todas las columnas y selecciona las que son TRUE
* contains() - columnas que contienen una cadena de caracteres
  + ejemplo: select(contains("time"))
* starts\_with() - coincide con un prefijo especificado
  + ejemplo: select(starts\_with("date\_"))
* ends\_with() - coincide con un sufijo especificado
  + ejemplo: select(ends\_with("\_post"))
* matches() - para aplicar una expresión regular (regex)
  + ejemplo: select(matches("[pt]al"))
* num\_range() - un rango numérico como x01, x02, x03
* any\_of() - coincide con la columna SI existe pero no devuelve ningún error si no se encuentra
  + ejemplo: select(any\_of(date\_onset, date\_death, cardiac\_arrest))

Además, utiliza operadores normales como c() para listar varias columnas, : para columnas consecutivas, ! para opuestas, & para “Y” y | para “O”.

Utiliza where() para especificar criterios lógicos para las columnas. Si escribes una función dentro de where(), no incluyas los paréntesis vacíos de la función. El comando siguiente selecciona las columnas de tipo Numeric.

Utiliza contains() para seleccionar sólo las columnas en las que el nombre de la columna contiene una cadena de caracteres especificada. ends\_with() y starts\_with() proporcionan más matices.

La función matches() funciona de forma similar a contains(), pero puede escribirse en una expresión regular (mira la página sobre [Caracteres y cadenas](#characters-and-strings)), como varias cadenas separadas por barras “O” dentro de los paréntesis:

**ATENCIÓN:** Si has escrito un nombre de columna y no existen datos para ella, puede devolver un error y detener tu código. Considera el uso de any\_of() para citar columnas que pueden o no existir, especialmente útil en selecciones negativas (eliminar).

Sólo existe una de estas columnas, pero no se produce ningún error y el código continúa sin detener su cadena de limpieza.

### Eliminar columnas

**Indica qué columnas se van a eliminar** colocando el símbolo "-" delante del nombre de la columna (por ejemplo, select(-outcome)), o un vector de nombres de columnas (como se indica a continuación). Todas las demás columnas se mantendrán.

También puede eliminar una columna utilizando la sintaxis de R **base**, definiéndola como NULL. Por ejemplo:

### Independiente

select() también puede utilizarse como un comando independiente (no en una cadena de tuberías). En este caso, el primer argumento es el dataframe original sobre el que se va a operar.

#### Añadir a la cadena de tuberías

En linelist\_raw, hay algunas columnas que no necesitamos: row\_num, merged\_header y x28. Las eliminamos con un comando select() en la cadena de tuberías de limpieza:

## Des-duplicación

Consulta la página sobre [Des-duplicación](#de-duplication) para ver la cantidad de opciones sobre cómo eliminar las duplicidades (Des-duplicar). Aquí sólo se presenta un ejemplo muy sencillo de des-duplicación de filas.

El paquete **dplyr** ofrece la función distinct(). Esta función examina cada fila y reduce el dataframe con sólo filas únicas. Es decir, elimina las filas que están 100% duplicadas.

Al evaluar las filas duplicadas, tiene en cuenta un rango de columnas - por defecto considera todas las columnas. Como se muestra en la página de des-duplicación, puedes ajustar este rango de columnas para que la singularidad de las filas sólo se evalúe con respecto a determinadas columnas.

En este sencillo ejemplo, simplemente añadimos el comando vacío distinct() a la cadena de tuberías. Esto garantiza que no haya filas que estén 100% duplicadas de otras filas (evaluadas en todas las columnas).

Comenzamos con nrow(linelist) filas en linelist.

Después de la desduplicación hay nrow(linelist) filas. Las filas eliminadas habrían sido 100% duplicados de otras filas.

A continuación, se añade el comando distinct() a la cadena de tuberías de limpieza:

## Creación y transformación de columnas

**Recomendamos utilizar la función dplyr mutate() para añadir una nueva columna, o para modificar una existente.**

A continuación se muestra un ejemplo de creación de una nueva columna con mutate(). La sintaxis es: mutate(nombre\_nueva\_columna = valor o transformación)

En Stata, esto es similar al comando generate, pero también se puede utilizar mutate() de R para modificar una columna existente.

### Nuevas columnas

El comando más básico de mutate() para crear una nueva columna podría tener este aspecto. Crea una nueva columna new\_col donde el valor en cada fila es 10.

También puedes referenciar valores en otras columnas, para realizar cálculos. A continuación, se crea una nueva columna bmi para mantener el Índice de Masa Corporal (BMI) de cada caso - calculado mediante la fórmula BMI = kg/m^2, utilizando la columnas ht\_cm y wt\_kg.

Si creas varias columnas nuevas, separa cada una con una coma y una nueva línea. A continuación se muestran ejemplos de nuevas columnas, incluidas las que consisten en valores de otras columnas combinadas mediante str\_glue() del paquete **stringr** (véase la página sobre [Caracteres y cadenas](#characters-and-strings).

Revisa las nuevas columnas. A efectos de demostración, sólo se muestran las nuevas columnas y las columnas utilizadas para crearlas:

**CONSEJO:** Una variación de mutate() es la función transmute(). Esta función añade una nueva columna al igual que mutate(), pero también elimina todas las demás columnas que no se mencionan dentro de sus paréntesis.

### Convertir el tipo de columna

Las columnas que contienen valores que son fechas, números o valores lógicos (TRUE/FALSE) sólo se comportarán como se espera si están correctamente clasificadas. Hay una diferencia entre "2" de tipo carácter y 2 de tipo numérico!

Hay formas de establecer el tipo de la columna durante los comandos de importación, pero esto suele ser engorroso. Consulta la sección [Aspectos básicos de R](#r-basics) sobre los tipos de objeto para saber más sobre la conversión de los tipos de objetos y columnas.

En primer lugar, vamos a realizar algunas comprobaciones en las columnas importantes para ver si son del tipo correcto. También vimos esto al principio cuando ejecutamos skim().

Actualmente, el tipo de la columna de edad es un carácter. Para realizar análisis cuantitativos, necesitamos que estos números sean reconocidos como numéricos.

El tipo de la columna date\_onset ¡también es un carácter! Para realizar los análisis, ¡estas fechas deben ser reconocidas como fechas!

Para resolver esto, utiliza la capacidad de mutate() para redefinir una columna mediante una transformación. Definimos la columna como ella misma, pero convertida a un tipo diferente. He aquí un ejemplo básico, convirtiendo o asegurando que la columna edad es de tipo Numeric:

De forma similar, puedes utilizar as.character() y as.logical(). Para convertir al tipo Factor, puedes utilizar factor() de R **base** o as\_factor() de **forcats**. Lee más sobre esto en la página de [Factores](#factors).

Hay que tener cuidado al convertir al tipo Fecha. En la página [Trabajar con fechas](#working-with-dates-1) se explican varios métodos. Normalmente, los valores de fecha en el fichero crudo deben estar todos en el mismo formato para que la conversión funcione correctamente (por ejemplo, "MM/DD/AAAA", o "DD MM AAAA"). Después de convertir al tipo Fecha, comprueba tus datos para confirmar que cada valor se ha convertido correctamente.

### Datos agrupados

Si tu dataframe ya está agrupado (véase la página sobre [Agrupar datos](#grouping-data)), mutate() puede comportarse de forma diferente que si el dataframe no está agrupado. Cualquier función de resumen, como mean(), median(), max(), etc. calculará con datos agrupados, no con filas de registros individualizados.

Lee más sobre el uso de mutate () sobre dataframes agrupados en esta [documentación de tidyverse mutate](https://dplyr.tidyverse.org/reference/mutate.html).

### Transformar múltiples columnas

A menudo, para escribir un código conciso, se desea aplicar la misma transformación a varias columnas a la vez. Se puede aplicar una transformación a varias columnas a la vez utilizando la función across() del paquete **dplyr** (también contenido en el paquete **tidyverse**). across() se puede utilizar con cualquier función **de dplyr**, pero se suele utilizar dentro de select(), mutate(), filter() o summarise(). Mira cómo se aplica a summarise() en la página sobre [Tablas descriptivas](#descriptive-tables).

Especificar los argumentos de las columnas .cols = y la(s) función(es) a aplicar a .fns =. Cualquier argumento adicional a la función .fns puede incluirse después de una coma, todavía dentro de across().

#### selección de columnas across()

Especificar las columnas de .cols =. Puedes nombrarlas individualmente, o utilizar funciones de ayuda "tidyselect". Especifica la función en .fns =. Ten en cuenta que utilizando el modo de función mostrado a continuación, la función se escribe sin sus paréntesis ().

Aquí la transformación as.character() se aplica a columnas específicas nombradas dentro de across().

Las funciones de ayuda "tidyselect" están disponibles para ayudarle a especificar las columnas. Se detallan más arriba en la sección sobre Selección y reordenación de columnas, e incluyen: everything(), last\_col(), where(), starts\_with(), ends\_with(), contains(), matches(), num\_range() y any\_of().

Este es un ejemplo de cómo se pueden cambiar **todas las columnas** al tipo carácter:

Convertir en caracteres todas las columnas cuyo nombre contenga la cadena "fecha" (fíjate en la colocación de comas y paréntesis):

A continuación, un ejemplo de mutación de las columnas que actualmente son de tipo POSIXct (un tipo datetime cruda que muestra etiquetas) - en otras palabras, donde la función is.POSIXct() evalúa a TRUE. Entonces queremos convertirlas con la función as.Date() en columnas de tipo Date normal.

* Ten en cuenta que dentro de across() también utilizamos la función where() como is.POSIXct está evaluando a TRUE o FALSE.
* Ten en cuenta que is.POSIXct() es del paquete **lubridate**. Otras funciones "is" similares como is.character(), is.numeric(), e is.logical() son de R **base**

#### funciones across()

Puedes leer la documentación de ayuda con detalles sobre cómo proporcionar funciones a across() escribiendo ?across: hay varias formas de especificar la(s) función(es) a realizar en una columna e incluso puedes definir tus propias funciones:

* Puedes escribir el nombre de la función sola (por ejemplo, media o as.carácter)
* Puedes escribir la función en **estilo purrr (por** ejemplo, ~ mean(.x, na.rm = TRUE)) (mira [esta página](#iteration-loops-and-lists))
* Puedes especificar varias funciones escribiendo una lista (por ejemplo, list(media = media, n\_miss = ~ suma(is.na(.x))).
  + Si proporcionas varias funciones, se devolverán varias columnas transformadas por cada columna de entrada, con nombres únicos con formato col\_fn. Puedes ajustar cómo se nombran las columnas nuevas con el argumento .names = utilizando la sintaxis **glue** (mira la página sobre [Caracteres y cadenas](#characters-and-strings)) donde {.col} y {.fn} son la abreviatura de la columna de entrada y la función.

Aquí hay algunos recursos en línea sobre el uso de across(): [pensamientos/razones del creador Hadley Wickham](https://www.tidyverse.org/blog/2020/04/dplyr-1-0-0-colwise/)

### coalesce()

Esta función de **dplyr** encuentra el primer valor no missing en cada posición. Rellena los valores que faltan con el primer valor disponible en el orden que especifiques.

Aquí hay un ejemplo fuera del contexto de un dataframe: Supongamos que tienes dos vectores, uno que contiene el pueblo de detección del paciente y otro que contiene el pueblo de residencia del paciente. Puedes utilizar coalesce para elegir el primer valor no ausente de cada índice:

Esto funciona de la misma manera si se proporcionan columnas del dataframe: para cada fila, la función asignará el nuevo valor de la columna con el primer valor que no falte en las columnas proporcionadas (en el orden proporcionado).

Este es un ejemplo de operación "por filas". Para cálculos más complicados por filas, consulta la sección siguiente sobre cálculos por filas.

### Matemáticas acumulativas

Si desea que una columna refleje la suma/media/min/máxima, etc. acumulada, tal y como se ha evaluado en las filas de un dataframe hasta ese punto, utiliza las siguientes funciones:

cumsum() devuelve la suma acumulada, como se muestra a continuación:

Esto se puede utilizar en un dataframe al crear una nueva columna. Por ejemplo, para calcular el número acumulado de casos por día en un brote, considere un código como este:

A continuación se muestran las 10 primeras filas:

Consulta la página sobre [curvas epidémicas](#epidemic-curves) para saber cómo representar la incidencia acumulada con epicurve.

Véase también:  
cumsum(), cummean(), cummin(), cummax(), cumany(), cumall()

### Utilizando R ****base****

Para definir una nueva columna (o redefinir una columna) utilizando R **base**, escribe el nombre del dataframe, conectado con $, a la nueva columna (o la columna a modificar). Utiliza el operador de asignación <- para definir el nuevo valor o valores. Recuerda que al usar R **base** debes especificar siempre el nombre del dataframe antes del nombre de la columna (por ejemplo, dataframe$columna). Este es un ejemplo de creación de la columna bmi usando R **base**:

### Añadir tuberías a la cadena

**A continuación, se añade una nueva columna a la cadena de tuberías y se convierten algunos tipos.**

## Recodificar valores

A continuación se presentan algunos escenarios en los que es necesario recodificar (cambiar) los valores:

* para editar un valor específico (por ejemplo, una fecha con un año o formato incorrecto)
* para conciliar valores que no se escriben igual
* para crear una nueva columna de valores categóricos
* para crear una nueva columna de categorías numéricas (por ejemplo, categorías de edad)

### Valores específicos

Para cambiar los valores manualmente puedes utilizar la función recode() dentro de la función mutate().

Imagínatee que hay una fecha sin sentido en los datos (por ejemplo, "2014-14-15"): podrías corregir la fecha manualmente en los datos originales, o bien, podrías escribir el cambio en la serie de comandos de limpieza a través de mutate() y recode(). Esto último es más transparente y reproducible para cualquier otra persona que quiera entender o repetir su análisis.

La línea mutate() anterior puede leerse como: "mutar la columna date\_onset para que sea igual a la columna date\_onset recodificada de forma que el VALOR ANTIGUO se cambie por el NUEVO VALOR". Ten en cuenta que este patrón (VIEJO = NUEVO) para recodificar() es el opuesto a la mayoría de los patrones de R (nuevo = viejo). La comunidad de desarrollo de R está trabajando en la revisión de esto.

**Aquí hay otro ejemplo de recodificación de múltiples valores dentro de una columna.**

En linelist hay que limpiar los valores de la columna "hospital". Hay varias grafías diferentes y muchos valores que faltan.

El comando recode() de abajo redefine la columna "hospital" como la columna actual "hospital", pero con los cambios especificados en la recodificación. ¡No olvides las comas después de cada uno!

Ahora vemos que se han corregido y consolidado las grafías de la columna hospital:

**CONSEJO:** El número de espacios antes y después de un signo de igualdad no importa. Haz que tu código sea más fácil de leer alineando el signo = para todas o la mayoría de las filas. Además, considera la posibilidad de añadir una fila de comentarios con hash (#) para aclarar a los futuros lectores qué lado es VIEJO y qué lado es NUEVO.

**CONSEJO:** A veces existe un valor con caracteres en blanco en unos datos (no reconocido como valor Missing - Na de R. Puedes hacer referencia a este valor con dos comillas sin espacio intermedio ("").

### Por lógica

A continuación demostramos cómo recodificar los valores de una columna utilizando lógica y condiciones:

* Uso de replace(), ifelse() e if\_else() para una lógica simple
* Uso de case\_when() para una lógica más compleja

### Lógica simple

#### replace()

Para recodificar con criterios lógicos simples, puedes utilizar replace() dentro de mutate(). replace() es una función de R **base**. Utiliza una condición lógica para especificar las filas a cambiar . La sintaxis general es:

mutate(col\_to\_change = replace(col\_to\_change, criterio para filas, nuevo valor)).

Una situación común para utilizar replace() es **cambiar sólo un valor en una fila, utilizando un identificador de fila único**. A continuación, el género se cambia a "Mujer" en la fila donde la columna case\_id es "2195".

El comando equivalente utilizando la sintaxis de R **base** y los paréntesis de indexación [ ] está abajo. Se lee como "Cambia el valor de la columna género del dataframe linelist a 'Mujer'" (para las filas en las que la columna case\_id del listado tiene el valor '2195').

#### ifelse() y if\_else()

Otra herramienta para la lógica simple es ifelse() y su compañero if\_else(). Sin embargo, en la mayoría de los casos para la recodificación es más claro utilizar case\_when() (detallado a continuación). Estos comandos "if else" son versiones simplificadas de una sentencia de programación if y else. La sintaxis general es:

ifelse(condición, valor a devolver si la condición evalúa como TRUE, valor a devolver si la condición evalúa como FALSE)

A continuación se define la columna fuente\_conocida. Su valor en una fila determinada se establece como "conocido" si no falta el valor de la fila en la columna fuente. Si falta el valor de la fuente, el valor de source\_known se establece como "desconocido".

if\_else() es una versión especial de **dplyr** que maneja fechas. Ten en cuenta que si el valor "verdadero" es una fecha, el valor "falso" también debe calificar una fecha, de ahí que se utilice el valor especial NA\_real\_ en lugar de simplemente NA.

**Evita encadenar muchos comandos ifelse... ¡utiliza** case\_when**() en su lugar!** case\_when() es mucho más fácil de leer y cometerá menos errores.

Fuera del contexto de un dataframe, si deseas que un objeto utilizado en su código cambie su valor, considera el uso de switch() de R **base**.

### Lógica compleja

Utiliza case\_when() de **dplyr** si estás recodificando en muchos grupos nuevos, o si necesita utilizar sentencias lógicas complejas para recodificar valores. Esta función evalúa si cada fila del dataframe cumple los criterios especificados y asigna el nuevo valor correcto.

Los comandos case\_when() consisten en sentencias que tienen un lado derecho (RHS) y un lado izquierdo (LHS) separados por una "tilde" ~. Los criterios lógicos están en el lado izquierdo y los valores de conformidad están en el lado derecho de cada sentencia. Las declaraciones están separadas por comas.

Por ejemplo, aquí utilizamos las columnas edad y edad\_unidad para crear una columna edad\_años:

A medida que se evalúa cada fila de los datos, los criterios se aplican/evalúan en el orden en que se escriben las sentencias case\_when(), de arriba a abajo. Si el criterio superior se evalúa como TRUE para una fila determinada, se asigna el valor RHS, y los criterios restantes ni siquiera se prueban para esa fila. Por lo tanto, es mejor escribir los criterios más específicos primero y los más generales al final.

En esta línea, en su declaración final, coloque TRUE en el lado izquierdo, lo que capturará cualquier fila que no cumpla ninguno de los criterios anteriores. Al lado derecho de esta declaración se le podría asignar un valor como "¡comprobado!" o faltante.

**PELIGRO: Los vvalores del lado derecho deben ser todos del mismo tipo**: numéricos, de caracteres, de fecha, lógicos, etc. Para asignar faltantes (NA), puede ser necesario utilizar variaciones especiales de NA como NA\_character\_, NA\_real\_ (para numérico o POSIX), y as.Date(NA). Lea más en [Trabajar con fechas](#working-with-dates-1).

### Valores faltantes

A continuación se presentan funciones especiales para el tratamiento de los valores faltantes en el contexto de la limpieza de datos.

Consulte la página sobre [Valores faltantes](#missing-data) para obtener consejos más detallados sobre la identificación y el tratamiento de los valores faltantes. Por ejemplo, la función is.na() que comprueba lógicamente la ausencia de datos.

**replace\_na()**

Para cambiar los valores faltantes (NA) por un valor específico, como "Missing", utiliza la función **dplyr** replace\_na() dentro de mutate(). Ten en cuenta que se utiliza de la misma manera que recodificar anteriormente - el nombre de la variable debe repetirse dentro de replace\_na().

**fct\_explicit\_na()**

Esta es una función del paquete **forcats**. El paquete **forcats** maneja columnas del tipo Factor. Los factores son la forma en que R maneja valores ordenados como c("Primero", "Segundo", "Tercero") o para establecer el orden en que los valores (por ejemplo, hospitales) aparecen en las tablas y gráficos. Vea la página sobre [Factores](#factors).

Si tus datos son del tipo Factor y trata sde convertir NA en "Missing" utilizando replace\_na(), obtendrás este error: nivel de factor no válido, NA generado. Ha intentado añadir "Missing" como valor, cuando no estaba definido como un posible nivel del factor, y ha sido rechazado.

La forma más fácil de resolver esto es utilizar la función **forcats** fct\_explicit\_na() que convierte una columna en factor de tipo, y convierte los valores NA en el carácter "(Missing)".

Una alternativa más lenta sería añadir el nivel del factor utilizando fct\_expand() y luego convertir los valores que faltan.

**na\_if()**

Para convertir un valor específico en NA, utiliza na\_if() de **dplyr**. El comando siguiente realiza la operación opuesta a replace\_na(). En el siguiente ejemplo, cualquier valor de "Missing" en la columna hospital se convierte en NA.

Nota: na\_if() **no puede utilizarse para criterios lógicos** (por ejemplo, "todos los valores > 99") - utiliza replace() o case\_when() para ello:

### Diccionario de limpieza

Utiliza el paquete R **linelist** y su función clean\_variable\_spelling() para limpiar un dataframe con un diccionario de limpieza. **linelist** es un paquete desarrollado por [RECON](https://github.com/reconhub/linelist) - el Consorcio R Epidemics.

1. Crear un diccionario de limpieza con 3 columnas:
   * Una columna "desde" (el valor incorrecto)
   * Una columna "para" (el valor correcto)
   * Una columna que especifica la columna a la que se aplicarán los cambios (o ".global" para aplicarlo a todas las columnas)

Nota: Las entradas del diccionario global serán anuladas por las entradas del diccionario específico de la columna.

1. Importe el archivo del diccionario a R. Este ejemplo puede descargarse a través de las instrucciones de la página [Descargar manual y datos](#download-handbook-and-data).
2. Pase linelist crudas a clean\_variable\_spelling(), especificando a wordlists = el dataframe del diccionario de limpieza. El argumento spelling\_vars = puede utilizarse para especificar a qué columna del diccionario se refieren las columnas (la tercera por defecto), o puede establecerse como NULL para que el diccionario se aplique a todas las columnas de caracteres y factores. Ten en cuenta que esta función puede tardar mucho en ejecutarse.

Ahora desplácese a la derecha para ver cómo han cambiado los valores - en particular el género (de minúsculas a mayúsculas), y todas las columnas de síntomas se han transformado de sí/no a 1/0.

Ten en cuenta que los nombres de las columnas en el diccionario de limpieza deben corresponder a los nombres en este punto de su script de limpieza. Consulte esta [referencia en línea para el paquete linelist](https://www.repidemicsconsortium.org/linelist/reference/clean_data.html) para obtener más detalles.

#### Añadir a la cadena de tuberías

**A continuación, se añaden algunas columnas y transformaciones de columna nuevas a la cadena de tuberías.**

## Categorías numéricas

Aquí describimos algunos enfoques especiales para crear categorías a partir de columnas numéricas. Algunos ejemplos comunes son las categorías de edad, los grupos de valores de laboratorio, etc. Aquí discutiremos:

* age\_categories(), del paquete **epikit**
* cut(), de la **base** R
* case\_when()
* ruptura de cuantiles con quantile() y ntile()

### Distribución de la revisión

Para este ejemplo crearemos una columna age\_cat utilizando la columna age\_years.

En primer lugar, examina la distribución de tus datos, para hacer los puntos de corte apropiados. Consulte la página sobre [los fundamentos de ggplot](#ggplot-basics).

**ATENCIÓN:** A veces, las variables numéricas se importarán como tipo "carácter". Esto ocurre si hay caracteres no numéricos en algunos de los valores, por ejemplo una entrada de "2 meses" para la edad, o (dependiendo de la configuración de su configuración local de R) si se utiliza una coma en el lugar de los decimales (por ejemplo, "4,5" para significar cuatro años y medio)..

### categorías\_de\_edad()

Con el paquete **epikit**, puede utilizar la función age\_categories() para categorizar y etiquetar fácilmente las columnas numéricas (nota: esta función puede aplicarse también a las variables numéricas no relacionadas con la edad). Como bonum, la columna de salida es automáticamente un factor ordenado.

Aquí están las entradas requeridas:

* Un vector numérico (columna)
* El argumento breakers = - proporciona un vector numérico de puntos de ruptura para los nuevos grupos

Primero, el ejemplo más sencillo:

Los valores de ruptura que especificas son por defecto los límites inferiores - es decir, están incluidos en el grupo "superior" / los grupos están "abiertos" en la parte inferior/izquierda. Como se muestra a continuación, puede añadir 1 a cada valor de ruptura para conseguir grupos que estén abiertos por la parte superior/derecha.

Puedes ajustar cómo se muestran las etiquetas con el separador =. El valor predeterminado es "-"

Puedes ajustar cómo se manejan los números superiores, con el argumento ceiling =. Para establecer un corte superior establezca ceiling = TRUE. En este uso, el valor de ruptura más alto proporcionado es un "techo" y no se crea una categoría "XX+". Cualquier valor por encima del valor de corte más alto (o hasta el límite superior =, si está definido) se categoriza como NA. A continuación se muestra un ejemplo con techo = TRUE, de modo que no hay categoría de XX+ y los valores por encima de 70 (el valor de ruptura más alto) se asignan como NA.

Alternativamente, en lugar de los interruptores =, puede proporcionar todos los inferiores =, superiores =, y por =:

* lower = El número más bajo que se quiere considerar - por defecto es 0
* upper = El número más alto que quiere que se considere
* por = El número de años entre los grupos

Consulte la página de ayuda de la función para obtener más detalles (introduzca ?age\_categories en la consola de R).

### cortar()

cut() es una alternativa **básica** de R a age\_categories(), pero creo que verá por qué age\_categories() se desarrolló para simplificar este proceso. Algunas diferencias notables de age\_categories() son:

* No es necesario instalar/cargar otro paquete
* Puedes especificar si los grupos están abiertos/cerrados a la derecha/izquierda
* Debes proporcionar etiquetas precisas
* Si quieres que el 0 se incluya en el grupo más bajo debes especificar esto

La sintaxis básica dentro de cut() es proporcionar primero la columna numérica que se va a cortar (edad\_años), y luego el argumento breaks, que es un vector numérico c() de puntos de ruptura. Utilizando cut(), la columna resultante es un factor ordenado.

Por defecto, la categorización se produce de manera que el lado derecho/superior es "abierto" e inclusivo (y el lado izquierdo/inferior es "cerrado" o exclusivo). Este es el comportamiento opuesto al de la función age\_categories(). Las etiquetas por defecto utilizan la notación "(A, B]", lo que significa que A no está incluido pero B sí. **Invierta este comportamiento proporcionando el argumento right = TRUE**.

Así, por defecto, los valores "0" se excluyen del grupo más bajo, y se categorizan como NA! Los valores "0" podrían ser codificados por los bebés como edad 0, así que ¡tenga cuidado! Para cambiar esto, añada el argumento include.lowest = TRUE para que cualquier valor "0" se incluya en el grupo más bajo. La etiqueta generada automáticamente para la categoría más baja será entonces "[A],B]". Ten en cuenta que si incluye el argumento include.lowest = TRUE **y** right = TRUE, la inclusión extrema se aplicará ahora al valor del punto de ruptura y a la categoría más altos, no a los más bajos.

Puedes proporcionar un vector de etiquetas personalizadas utilizando el argumento labels =. Como se escriben manualmente, ¡tenga mucho cuidado de que sean precisas! Compruebe su trabajo utilizando una tabulación cruzada, como se describe a continuación.

A continuación se muestra un ejemplo de cut() aplicado a age\_years para crear la nueva variable age\_cat:

**Compruebe su trabajo!** Verifique que cada valor de edad fue asignado a la categoría correcta cruzando las columnas numéricas y de categoría. Examine la asignación de los valores límite (por ejemplo, 15, si las categorías vecinas son 10-15 y 16-20).

**Reetiquetado de los valores NA**

Puedes asignar a los valores NA una etiqueta como "Missing". Como la nueva columna es del tipo Factor (valores restringidos), no puede simplemente mutarla con replace\_na(), ya que este valor será rechazado. En su lugar, utiliza fct\_explicit\_na() de **forcats** como se explica en la página de [Factores](#factors).

**Realiza rápidamente pausas y etiquetas**

Para una forma rápida de hacer rupturas y etiquetar vectores, utiliza algo como lo siguiente. Consulte la página de [fundamentos de R](#r-basics) para obtener referencias sobre seq() y rep().

Lea más sobre cut() en su página de ayuda introduciendo ?cut en la consola de R.

### Roturas cuantílicas

En el entendimiento común, los "cuantiles" o "percentiles" suelen referirse a un valor por debajo del cual cae una proporción de valores. Por ejemplo, el percentil 95 de las edades en linelist sería la edad por debajo de la cual cae el 95% de las edades.

Sin embargo, en el lenguaje común, "cuartiles" y "deciles" también pueden referirse a los grupos de datos divididos por igual en 4 o 10 grupos (Ten en cuenta que habrá un punto de ruptura más que un grupo).

Para obtener los puntos de ruptura de los cuantiles, se puede utilizar quantile() del paquete **stats** de R **base.** Se proporciona un vector numérico (por ejemplo, una columna en unos datos) y un vector de valores de probabilidad numérica que van de 0 a 1,0. Los puntos de ruptura se devuelven como un vector numérico. Explore los detalles de las metodologías estadísticas introduciendo ?quantile.

* Si su vector numérico de entrada tiene valores faltantes, es mejor establecer na.rm = TRUE
* Establecer nombres = FALSE para obtener un vector numérico sin nombre

Puedes utilizar los resultados de quantile() como puntos de ruptura en age\_categories() o cut(). A continuación creamos una nueva columna deciles utilizando cut() donde los puntos de ruptura se definen utilizando quantiles() en age\_years. A continuación, mostramos los resultados utilizando tabyl() de **janitor** para que pueda ver los porcentajes (véase la página de [tablas descriptivas](#descriptive-tables)). Observe cómo no son exactamente el 10% en cada grupo.

### Grupos de tamaño uniforme

Otra herramienta para hacer grupos numéricos es la función **dplyr** ntile(), que intenta dividir los datos en n grupos de tamaño uniforme - pero Ten en cuenta que, a diferencia de *quantile(),* el mismo valor podría aparecer en más de un grupo. Proporcione el vector numérico y luego el número de grupos. Los valores de la nueva columna creada son sólo "números" de grupo (por ejemplo, del 1 al 10), no el rango de valores en sí mismo como cuando se utiliza cut().

### case\_when()

Es posible utilizar la función **dplyr** case\_when() para crear categorías a partir de una columna numérica, pero es más fácil utilizar age\_categories() de **epikit** o cut() porque éstas crearán un factor ordenado automáticamente.

Si utiliza case\_when(), por favor, revise el uso adecuado como se ha descrito anteriormente en la sección Re-codificar valores de esta página. También Ten en cuenta que todos los valores del lado derecho deben ser del mismo tipo. Por lo tanto, si quiere NA en el lado derecho debe escribir "Missing" o utilizar el valor especial NA\_character\_.

### Añadir a la cadena de tuberías

A continuación, se añade el código para crear dos columnas categóricas de edad a la cadena de tuberías de limpieza:

## Añadir filas

### Uno a uno

Añadir filas una a una manualmente es tedioso pero puede hacerse con add\_row() de **dplyr**. Recuerde que cada columna debe contener valores de un solo tipo (ya sea carácter, numérico, lógico, etc.). Así que añadir una fila requiere matizar para mantener esto.

Utiliza .before y .after. para especificar la ubicación de la fila que desea añadir. .before = 3 pondrá la nueva fila antes de la tercera fila actual. El comportamiento por defecto es añadir la fila al final. Las columnas no especificadas se dejarán vacías (NA).

El nuevo número de fila puede parecer extraño ("...23") pero los números de fila de las filas preexistentes han cambiado. Por lo tanto, si utiliza el comando dos veces, examine/pruebe la inserción cuidadosamente.

Si uno de los tipos que proporcionas está desactivado, verás un error como este:

Error: No se puede combinar ..1$fecha de infección <fecha> y ..2$fecha de infección <carácter>.

(al insertar una fila con un valor de fecha, recuerde envolver la fecha en la función as.Date() como as.Date("2020-10-10")).

### Encuadernar filas

Para combinar conjuntos de datos uniendo las filas de un dataframe al fondo de otro dataframe, puede utilizar bind\_rows() de **dplyr**. Esto se explica con más detalle en la página [Unir datos](#joining-data).

## Filtrar filas

Un paso típico de limpieza después de haber limpiado las columnas y recodificado los valores es filtrar el dataframe para filas específicas usando el verbo **dplyr** filter().

Dentro de filter(), especifique la lógica que debe ser TRUE para que se mantenga una fila en los datos. A continuación mostramos cómo filtrar filas basándose en condiciones lógicas simples y complejas.

### Filtro simple

Este sencillo ejemplo redefine linelist del dataframe como ella misma, habiendo filtrado las filas para que cumplan una condición lógica. **Sólo se conservan las filas en las que la declaración lógica dentro de los paréntesis se evalúa como TRUE.**

En este ejemplo, la sentencia lógica es gender == "f", que pregunta si el valor de la columna gender es igual a "f" (distingue entre mayúsculas y minúsculas).

Antes de aplicar el filtro, el número de filas del listado es nrow(linelist).

Después de aplicar el filtro, el número de filas del listado es linelist %>% filter(gender == "f") %>% nrow().

### Filtrar los valores que faltan

Es bastante común querer filtrar las filas que tienen valores faltantes. Resista la tentación de escribir filter(!is.na(columna) & !is.na(columna)) y utiliza en su lugar la función de tidyr que está hecha a medida para este propósito: drop\_na(). Si se ejecuta con paréntesis vacíos, elimina las filas con cualquier valor que falte. Como alternativa, puede proporcionar los nombres de las columnas específicas que deben evaluarse para comprobar si faltan, o utilizar las funciones de ayuda "tidyselect" descritas [anteriormente](#clean_tidyselect).

Consulte la página sobre [datos ausentes](#missing-data) para conocer muchas técnicas para analizar y gestionar los datos ausentes.

### Filtrar por número de fila

En un dataframe o tibble, cada fila suele tener un "número de fila" que (cuando se ve en R Viewer) aparece a la izquierda de la primera columna. No es en sí misma una columna real en los datos, pero puede utilizarse en una sentencia filter().

Para filtrar en base al "número de fila", puede utilizar la función **dplyr** row\_number() con paréntesis abiertos como parte de una sentencia lógica de filtrado. A menudo se utiliza el operador %in% y un rango de números como parte de esa sentencia lógica, como se muestra a continuación. Para ver las primeras N filas, también puede utilizar la función especial **de dplyr** head().

También puede convertir los números de fila en una verdadera columna pasando su dataframe a la función **tibble** rownames\_to\_column() (no ponga nada en los paréntesis).

### Filtro complejo

Se pueden construir sentencias lógicas más complejas utilizando paréntesis ( ), OR |, negación ! , %en%, y operadores AND &. Un ejemplo es el siguiente:

Nota: Puedes utilizar el operador ! delante de un criterio lógico para negarlo. Por ejemplo, !is.na(columna) se evalúa como verdadero si el valor de la columna no falta. Del mismo modo, !column %in% c("a", "b", "c") es verdadero si el valor de la columna no está en el vector.

#### Examinar los datos

A continuación se muestra un sencillo comando de una línea para crear un histograma de las fechas de inicio. Vea que un segundo brote más pequeño de 2012-2013 también está incluido en este conjunto de datos sin procesar. **Para nuestros análisis, queremos eliminar las entradas de este brote anterior.**

#### Cómo manejan los filtros los valores numéricos y de fecha que faltan

Podemos simplemente filtrar por date\_onset a las filas posteriores a junio de 2013? **Precaución. La aplicación del código filter(date\_onset > as.Date("2013-06-01")) eliminaría todas las filas de la epidemia posterior con una fecha de inicio ausente!**

**PELIGRO:** Filtrar a mayor que (>) o menor que (<) una fecha o número puede eliminar cualquier fila con valores faltantes (NA). Esto se debe a que NA es tratado como infinitamente grande y pequeño.

(Consulte la página sobre el [*trabajo con fechas*](#working-with-dates-1) para obtener más información sobre el trabajo con fechas y el paquete ***lubridate***)

#### Diseñar el filtro

Examinar una tabulación cruzada para asegurarse de que excluimos sólo las filas correctas:

¿Qué otros criterios podemos filtrar para eliminar el primer brote (en 2012 y 2013) de los datos? Vemos que:

* La primera epidemia en 2012 y 2013 ocurrió en el Hospital A, el Hospital B, y que también hubo 10 casos en el Hospital del Puerto.
* Los hospitales A y B no tuvieron casos en la segunda epidemia, pero el Hospital del Puerto sí.

Queremos excluir:

* Las filas nrow(linelist %>% filter(hospital %in% c("Hospital A", "Hospital B") | date\_onset < as.Date("2013-06-01"))) con inicio en 2012 y 2013 en cualquiera de los hospitales A, B o Puerto:
  + Excluir nrow(linelist %>% filter(date\_onset < as.Date("2013-06-01"))) filas con inicio en 2012 y 2013
  + Excluir nrow(linelist %>% filter(hospital %in% c('Hospital A', 'Hospital B') & is.na(date\_onset))) filas de los hospitales A y B con fechas de inicio ausentes
  + **No** excluir nrow(linelist %>% filter(!hospital %in% c('Hospital A', 'Hospital B') & is.na(date\_onset))) otras filas con fechas de inicio ausentes.

Comenzamos con un listado de nrow(linelist)`. Aquí está nuestra declaración de filtro:

Cuando volvemos a hacer la tabulación cruzada, vemos que los hospitales A y B se eliminan por completo, y los 10 casos del Hospital del Puerto de 2012 y 2013 se eliminan, y todos los demás valores son los mismos, tal y como queríamos.

Se pueden incluir varias sentencias dentro de un comando de filtrado (separadas por comas), o siempre se puede canalizar a un comando filter() separado para mayor claridad.

Nota: algunos lectores pueden notar que sería más fácil filtrar sólo por *date\_hospitalisation* porque es 100% completo sin valores faltantes. Esto es cierto. Pero *date\_onset* se utiliza para demostrar un filtro complejo.

### Independiente

El filtrado también puede realizarse como un comando independiente (no como parte de una cadena de tuberías). Como otros verbos **de dplyr**, en este caso el primer argumento debe ser el propio conjunto de datos.

También puede utilizar la **base** R para hacer un subconjunto utilizando corchetes que reflejen las [filas, columnas] que desea conservar.

### Revisar rápidamente los registros

A menudo se quiere revisar rápidamente unos pocos registros, para sólo unas pocas columnas. La función **base de** R View() imprimirá un dataframe para su visualización en su RStudio.

Vea linelist en RStudio:

Aquí hay dos ejemplos de visualización de celdas específicas (filas específicas y columnas específicas):

**Con las funciones dplyr filter() y select():**

Dentro de View(), canalice los datos a filter() para mantener ciertas filas, y luego a select() para mantener ciertas columnas. Por ejemplo, para revisar las fechas de inicio y hospitalización de 3 casos específicos:

Puedes lograr lo mismo con la sintaxis **básica** de R, utilizando los corchetes [ ] para el subconjunto que desea ver.

#### Añadir a la cadena de tuberías

## Cálculos por filas

Si desea realizar un cálculo dentro de una fila, puede utilizar rowwise() de **dplyr**. Consulte esta viñeta en línea sobre los cálculos [por filas](https://cran.r-project.org/web/packages/dplyr/vignettes/rowwise.html).   
Por ejemplo, este código aplica rowwise() y luego crea una nueva columna que suma el número de las columnas de síntomas especificadas que tienen valor "sí", para cada fila del listado. Las columnas se especifican dentro de sum() por su nombre dentro de un vector c(). rowwise() es esencialmente un tipo especial de group\_by(), por lo que es mejor utilizar ungroup() cuando haya terminado (página sobre [Agrupar datos](#grouping-data)).

Al especificar la columna a evaluar, puede utilizar las funciones de ayuda "tidyselect" descritas en la sección select() de esta página. Sólo tiene que hacer un ajuste (porque no las está utilizando dentro de una función **de dplyr** como select() o summarise()).

Ponga los criterios de especificación de la columna dentro de la función **dplyr** c\_across(). Esto se debe a que c\_across ([documentación](https://dplyr.tidyverse.org/reference/c_across.html)) está diseñada para trabajar con rowwise() específicamente. Por ejemplo, el siguiente código:

* Aplica rowwise() para que la siguiente operación (sum()) se aplique dentro de cada fila (no sumando columnas enteras)
* Crea una nueva columna num\_NA\_dates, definida para cada fila como el número de columnas (con nombre que contiene "fecha") para las que is.na() se evaluó como TRUE (son valores faltantes).
* ungroup() para eliminar los efectos de rowwise() en los pasos siguientes

También podría proporcionar otras funciones, como max() para obtener la fecha más reciente o más reciente de cada fila:

## Ordenar y clasificar

Utiliza la función **dplyr** arrange() para ordenar las filas por los valores de las columnas.

Lista simple de las columnas en el orden en que deben ser ordenadas. Especifique .by\_group = TRUE si desea que la ordenación se realice primero por cualquier agrupación aplicada a los datos (véase la página sobre [Agrupar datos](#grouping-data)).

Por defecto, la columna se ordenará en orden "ascendente" (que se aplica a las columnas numéricas y también a las de caracteres). Puedes ordenar una variable en orden "descendente" envolviéndola con desc().

La ordenación de los datos con arrange() es particularmente útil cuando se hacen [Tablas para su presentación](#tables-for-presentation), utilizando slice() para tomar las filas "superiores" por grupo, o estableciendo el orden de los niveles de los factores por orden de aparición.

Por ejemplo, para ordenar las filas de nuestra lista de líneas por hospital y luego por fecha de inicio en orden descendente, utilizaríamos

# #Trabajar con fechas

{#working-with-dates}

Trabajar con fechas en R requiere más atención que trabajar con otros tipos de objetos. A continuación, ofrecemos algunas herramientas y ejemplos para hacer este proceso menos doloroso. Por suerte, las fechas pueden manejarse fácilmente con la práctica y con un conjunto de paquetes útiles como **lubridate**.

Al importar los datos en bruto, R suele interpretar las fechas como objetos de carácter, lo que significa que no pueden utilizarse para operaciones generales con fechas, como la creación de series temporales y el cálculo de intervalos de tiempo. Para hacer las cosas más difíciles, hay muchas maneras de formatear una fecha y debe ayudar a R a saber qué parte de una fecha representa qué (mes, día, hora, etc.).

Las fechas en R son su propio tipo de objeto - el tipo Date. Hay que tener en cuenta que también hay un tipo que almacena objetos con fecha y hora. Los objetos fecha-hora se denominan formalmente tipos POSIXt, POSIXct, y/o POSIXlt (la diferencia no es importante). Estos objetos se denominan informalmente tipos datetime.

* Es importante hacer que R reconozca cuando una columna contiene fechas.
* Las fechas son un tipo de objeto y pueden ser difíciles de trabajar.
* Aquí presentamos varias formas de convertir columnas de fecha al tipo Date.

## Preparación

### Cargar paquetes

Este trozo de código muestra la carga de paquetes necesaria para esta página. En este manual destacamos p\_load() de **pacman**, que instala el paquete si es necesario y lo carga para su uso. También puede cargar los paquetes instalados con library() de R **base.** Consulte la página sobre [los fundamentos de R](#r-basics) para obtener más información sobre los paquetes de R.

### Importar datos

Importamos los datos de casos de una epidemia de ébola simulada. Si desea descargar los datos para seguirlos paso a paso, consulte las instrucciones en la página de [descarga de manuales y datos](#download-handbook-and-data). Asumimos que el archivo está en el directorio de trabajo, por lo que no se especifican subcarpetas en esta ruta de archivo.

## Fecha actual

Puedes obtener la fecha actual del "sistema" o la fecha-hora del sistema de su ordenador haciendo lo siguiente con la **base** R.

Con el paquete **lubridate** también se pueden devolver con today() y now(), respectivamente. date() devuelve la fecha y la hora actuales con los nombres del día de la semana y del mes.

## Convertir en fecha

Después de importar unos datos a R, los valores de las columnas de fecha pueden tener el aspecto de "1989/12/30", "05/06/2014" o "13 Ene 2020". En estos casos, es probable que R siga tratando estos valores como valores de carácter. Hay que decirle a R que estos valores son fechas... y cuál es el formato de la fecha (qué parte es Día, cuál es Mes, cuál es Año, etc).

Una vez dicho esto, R convierte estos valores al tipo Date. En segundo plano, R almacenará las fechas como números (el número de días desde su fecha "origen" 1 Ene 1970). No interactuarás con el número de la fecha a menudo, pero esto permite a R tratar las fechas como variables continuas y permitir operaciones especiales como el cálculo de la distancia entre las fechas.

Por defecto, los valores del tipo Date en R se muestran como AAAA-MM-DD. Más adelante en esta sección discutiremos cómo cambiar la visualización de los valores de fecha.

A continuación presentamos dos enfoques para convertir una columna de valores de carácter al tipo Date.

**CONSEJO:** Puedes comprobar el tipo actual de una columna con la función **base** de R class(), como class(linelist$date\_onset).

### ****base**** R

as.Date() es la función estándar y **básica** de R para convertir un objeto o una columna en el tipo Date (nótese la "D" en mayúscula).

El uso de as.Date() requiere que:

* Se especifica el formato ***existente*** de la fecha de carácter bruto o la fecha de origen si se suministran las fechas como números (véase la sección sobre las fechas de Excel)
* Si se utiliza en una columna de caracteres, todos los valores de fecha deben tener el mismo formato exacto (si no es el caso, pruebe con guess\_dates() del paquete **linelist**)

En **primer lugar**, comprueba el tipo de su columna con class() de la **Rbase**. Si no estás seguro o estás confundido sobre el tipo de datos (por ejemplo, ve "POSIXct", etc.) puede ser más fácil convertir primero la columna al tipo Character con as.character(), y luego convertirla al tipo Date.

**En segundo lugar**, dentro de la función as.Date(), utiliza el argumento format = para indicar a R el formato actual de los componentes de la fecha con caracteres - qué caracteres se refieren al mes, al día y al año, y cómo están separados. Si sus valores ya están en uno de los formatos de fecha estándar de R ("AAAA-MM-DD" o "AAAA/MM/DD") el argumento format = no es necesario.

Para formatear =, proporcione una cadena de caracteres (entre comillas) que represente el formato actual de la fecha utilizando las abreviaturas especiales "strptime" que aparecen a continuación. Por ejemplo, si sus fechas de caracteres están actualmente en el formato "DD/MM/AAAA", como "24/04/1968", entonces usaría format = "%d/%m/%Y" para convertir los valores en fechas. **Es necesario poner el formato entre comillas. Y no olvides las barras o guiones.**

La mayoría de las abreviaturas de strptime se enumeran a continuación. Puedes ver la lista completa ejecutando ?strptime.

%d = Número del día del mes (5, 17, 28, etc.)  
%j = Número del día del año (día juliano 001-366)  
%a = Día de la semana abreviado (lunes, martes, miércoles, etc.)  
%A = Día de la semana completo (lunes, martes, etc.) %w = Número del día de la semana (0-6, el domingo es 0)  
%u = Número del día de la semana (1-7, el lunes es 1)  
%W = Número de la semana (00-53, el lunes es el comienzo de la semana)  
%U = Número de la semana (01-53, el domingo es el comienzo de la semana)  
%m = Número del mes (p. ej. 01, 02, 03, 04)%b = Mes abreviado (enero, febrero, etc.)%B = Mes completo (enero, febrero, etc.)%y = Año de dos dígitos (p. ej.p. ej. 01, 02, 03, 04)%b =   
Mes abreviado (enero, febrero, etc.)  
%B = Mes completo (enero, febrero, etc.)  
%y = Año de 2 dígitos (p. ej. 89)  
%Y = Año de 4 dígitos (p. ej. 1989)  
%h = Horas (reloj de 24 horas)  
%m = Minutos%s   
= Segundos %z = Desplazamiento respecto a GMT%Z   
= Huso horario (carácter)

**CONSEJO:** El argumento format = de as.Date() no le dice a R el formato que quiere que tengan las fechas, sino cómo identificar las partes de la fecha tal y como son antes de ejecutar el comando.

**CONSEJO:** Asegúrate que en el argumento format = se utiiliza el mismo separador de partes de fechas (por ejemplo, /, -, o espacio) que está en tus fechas.

Una vez que los valores están en el tipo Fecha, R los mostrará por defecto en el formato estándar, que es AAAA-MM-DD.

### ****lubridate****

La conversión de objetos de carácter a fechas puede facilitarse utilizando el paquete **lubridate**. Se trata de un paquete **tidyverse** diseñado para hacer que el trabajo con fechas y horas sea más sencillo y consistente que en el R **básico.** Por estas razones, **lubridate** se considera a menudo el paquete estándar de oro para las fechas y la hora, y se recomienda siempre que se trabaje con ellas.

El paquete **lubridate** proporciona varias funciones de ayuda diferentes diseñadas para convertir objetos de caracteres en fechas de una manera intuitiva y más indulgente que especificando el formato en as.Date(). Estas funciones son específicas para el formato de fecha aproximado, pero permiten una variedad de separadores, y sinónimos para las fechas (por ejemplo, 01 vs Jan vs Enero) - se denominan según las abreviaturas de los formatos de fecha.

La función ymd() convierte de forma flexible los valores de fecha suministrados como **año, luego mes y luego día**.

La función mdy() convierte de forma flexible los valores de fecha suministrados como **mes, luego día y luego año**.

La función dmy() convierte de forma flexible los valores de fecha suministrados como **día, luego mes y luego año**.

Si se utiliza la canalización, la conversión de una columna de caracteres a fechas con **lubridate** podría tener este aspecto:

Una vez completado, puede ejecutar class() para verificar el tipo de la columna

Una vez que los valores están en el tipo Fecha, R los mostrará por defecto en el formato estándar, que es AAAA-MM-DD.

Ten en cuenta que las funciones anteriores funcionan mejor con años de 4 dígitos. Los años de 2 dígitos pueden producir resultados inesperados, ya que lubridate intenta adivinar el siglo.

Para convertir un año de 2 dígitos en un año de 4 dígitos (todos en el mismo siglo) puede convertirlo a tipo carácter y luego combinar los dígitos existentes con un prefijo usando str\_glue() del paquete **stringr** (ver [Caracteres y cadenas](#characters-and-strings)). A continuación, convierta a fecha.

### Combinar columnas

Puedes utilizar las funciones **de lubridate** make\_date() y make\_datetime() para combinar varias columnas numéricas en una columna de fecha. Por ejemplo, si tiene columnas numéricas onset\_day, onset\_month y onset\_year en el dataframe linelist:

## Fechas en Excel

En el fondo, la mayoría de los programas informáticos almacenan las fechas como números. R almacena las fechas desde un origen del 1 de enero de 1970. Así, si ejecuta as.numeric(as.Date("1970-01-01)) obtendrá 0.

Microsoft Excel almacena las fechas con un origen del 30 de diciembre de 1899 (Windows) o del 1 de enero de 1904 (Mac), dependiendo de su sistema operativo. Consulte esta [guía de Microsoft](https://docs.microsoft.com/en-us/office/troubleshoot/excel/1900-and-1904-date-system) para obtener más información.

Las fechas de Excel suelen importarse a R como estos valores numéricos en lugar de como caracteres. Si los datos que ha importado de Excel muestran las fechas como números o caracteres como "41369"... utiliza as.Date() (o la función as\_date() de **lubridate**) para convertir, pero **en lugar de suministrar un "formato" como el anterior, suministre la fecha de origen de Excel al** argumento origin =.

Esto no funcionará si la fecha de Excel se almacena en R como un tipo de carácter, así que asegúrese de que el número es de tipo numérico.

**NOTA:** Debe proporcionar la fecha de origen en el formato de fecha por defecto de R ("AAAA-MM-DD").

## Citas desordenadas

La función guess\_dates() del paquete **linelist** intenta leer una columna de fecha "desordenada" que contiene fechas en muchos formatos diferentes y convertir las fechas a un formato estándar. Puedes [leer más en línea sobre)](https://www.repidemicsconsortium.org/linelist/reference/guess_dates.html). Si guess\_dates() aún no está disponible en CRAN para R 4.0.2, intente instalarlo mediante pacman::p\_load\_gh("reconhub/linelist").

Por ejemplo guess\_dates vería un vector de las siguientes fechas de caracteres "03 Ene 2018", "07/03/1982", y "08/20/85" y las convertiría al tipo Date como 2018-01-03, 1982-03-07, y 1985-08-20.

Algunos argumentos opcionales para guess\_dates() que puede incluir son

* error\_tolerance - Proporción de entradas que no pueden ser identificadas como fechas a tolerar (por defecto 0,1 o 10%)
* last\_date - la última fecha válida (por defecto, la fecha actual)
* first\_date - la primera fecha válida. Por defecto, cincuenta años antes de last\_date.

## Trabajar con el tipo fecha-hora

Como se mencionó anteriormente, R también soporta un tipo datetime - una columna que contiene información de fecha **y** hora. Al igual que con el tipo Date, a menudo es necesario convertirlas de objetos de caracteres a objetos datetime.

### Convertir fechas con horas

Un objeto datetime estándar se formatea con la fecha en primer lugar, seguida de un componente de tiempo - por ejemplo, 01 Ene 2020, 16:30. Al igual que con las fechas, hay muchas maneras de formatearlas, y hay numerosos niveles de precisión (horas, minutos, segundos) que se pueden suministrar.

Por suerte, también existen funciones de ayuda **de lubridate** para ayudar a convertir estas cadenas en objetos datetime. Estas funciones son extensiones de las funciones de ayuda a la fecha, con \_h (sólo se suministran las horas), \_hm (se suministran las horas y los minutos), o \_hms (se suministran las horas, los minutos y los segundos) añadidas al final (por ejemplo, dmy\_hms()). Se pueden utilizar como se indica:

Convertir datetime con sólo horas a objeto datetime

Convertir datetime con horas y minutos a objeto datetime

Convertir datetime con horas, minutos y segundos a objeto datetime

Puedes indicar la zona horaria, pero se ignora. Consulte la sección más adelante en esta página sobre las zonas horarias.

Cuando se trabaja con un dataframe, las columnas de fecha y hora pueden combinarse para crear una columna de fecha y hora utilizando str\_glue() del paquete **stringr** y una función **lubridate** apropiada. Consulte la página sobre [Caracteres y cadenas](#characters-and-strings) para obtener detalles sobre **stringr**.

En este ejemplo, el dataframe del listado tiene una columna con formato "horas:minutos". Para convertirla en una fecha, hay que seguir algunos pasos:

1. Cree una columna de tiempo de admisión "limpia" con los valores faltantes rellenados con la mediana de la columna. Hacemos esto porque **lubridate** no opera con valores faltantes. Combínela con la columna fecha\_hospitalización y utiliza la función ymd\_hm() para convertirla.

### Convertir los tiempos en solitario

Si tus datos contienen sólo un carácter de tiempo (horas y minutos), puedes convertirlos y manipularlos como tiempos utilizando strptime() desde **la base** R. Por ejemplo, para obtener la diferencia entre dos de estos tiempos:

Sin embargo, Ten en cuenta que si no se proporciona un valor de fecha, se asume que la fecha es hoy. Para combinar una cadena de fecha y una cadena de hora, vea cómo usar **stringr** en la sección anterior. Lea más sobre strptime() [aquí](https://rdrr.io/r/base/strptime.html).

Para convertir números de un solo dígito a dos dígitos (por ejemplo, para "rellenar" las horas o los minutos con ceros a la izquierda para conseguir 2 dígitos), consulte esta [sección "Longitud de relleno" de la página Caracteres y cadenas](#str_pad).

### Tiempo de extracción

Puedes extraer elementos de una hora con hour(), minute(), o second() de **lubridate**.

He aquí un ejemplo de extracción de la hora y posterior clasificación por parte del día. Comenzamos con la columna hora\_admisión, que es de tipo Carácter en formato "HH:MM". En primer lugar, se utiliza strptime() como se ha descrito anteriormente para convertir los caracteres en tipo datetime. A continuación, se extrae la hora con hour(), devolviendo un número del 0 al 24. Por último, se crea una columna time\_period utilizando la lógica con case\_when() para clasificar las filas en Mañana/Tarde/Tarde/Noche en función de su hora de entrada.

Para saber más sobre case\_when(), consulte la página sobre [Limpieza de datos y funciones básicas](#cleaning-data-and-core-functions).

## Trabajar con fechas

lubridate también puede utilizarse para otras funciones, como la **extracción de aspectos de una fecha/hora**, la **realización de cálculos aritméticos de fechas** o el **cálculo de intervalos de fechas**

Aquí definimos una fecha que se utilizará para los ejemplos:

### Extraer los componentes de la fecha

Puedes extraer aspectos comunes como el mes, el día, el día de la semana:

También puede extraer componentes de tiempo de un objeto o columna de fecha-hora. Esto puede ser útil si quiere ver la distribución de los tiempos de admisión.

Hay varias opciones para recuperar las semanas. Consulte la sección sobre semanas epidemiológicas más abajo.

Ten en cuenta que si desea mostrar una fecha de una forma determinada (por ejemplo, "enero de 2020" o "jueves 20 de marzo" o "semana 20 de 1977") puede hacerlo de forma más flexible, tal y como se describe en la sección sobre Visualización de fechas.

### Fecha matemática

Puedes añadir ciertos números de días o semanas utilizando su respectiva función de **lubridación**.

### Intervalos de fechas

La diferencia entre las fechas se puede calcular mediante:

1. Asegúrese de que ambas fechas son del mismo tipo
2. Utiliza la resta para devolver la diferencia "difftime" entre las dos fechas
3. Si es necesario, convierta el resultado en tipo numéricoa para realizar los cálculos matemáticos posteriores

A continuación se calcula y muestra el intervalo entre dos fechas. Se pueden encontrar intervalos utilizando el símbolo de resta "menos" en los valores que son de tipo Fecha. Ten en cuenta, sin embargo, que el tipo del valor devuelto es "difftime", como se muestra a continuación, y debe ser convertido a numérico.

Para realizar operaciones posteriores sobre un "difftime", conviértalo en numérico con as.numeric().

Todo esto puede unirse para trabajar con datos, por ejemplo:

En un contexto de dataframe, si falta alguna de las fechas anteriores, la operación fallará para esa fila. El resultado será un NA en lugar de un valor numérico. Cuando utilices esta columna para los cálculos, asegúrate de establecer el argumento na.rm = en TRUE. Por ejemplo:

## Visualización de la fecha

Una vez que las fechas son del tipo correcto, a menudo se desea que se muestren de forma diferente, por ejemplo para que se muestren como "lunes 05 de enero" en lugar de "2018-01-05". También puede querer ajustar la visualización para agrupar las filas por los elementos de fecha mostrados, por ejemplo, para agrupar por mes-año.

### formato()

Ajuste la visualización de la fecha con la función **base** de R format(). Esta función acepta una cadena de caracteres (entre comillas) que especifica el formato de salida deseado en las abreviaturas strptime "%" (la misma sintaxis que se utiliza en as.Date()). A continuación se muestran las abreviaturas más comunes.

Nota: el uso de format() convertirá los valores al tipo Character, por lo que generalmente se utiliza hacia el final de un análisis o sólo para fines de visualización. Puedes ver la lista completa ejecutando ?strptime.

%d = Número del día del mes (5, 17, 28, etc.)  
%j = Número del día del año (día juliano 001-366)  
%a = Día de la semana abreviado (lunes, martes, miércoles, etc.)  
%A = Día de la semana completo (lunes, martes, etc.)  
%w = Número del día de la semana (0-6, el domingo es 0)  
%u = Número del día de la semana (1-7, el lunes es 1)  
%W = Número de la semana (00-53, el lunes es el comienzo de la semana)  
%U = Número de la semana (01-53, el domingo es el comienzo de la semana)  
%m = Número del mes (p. ej. 01, 02, 03, 04)%b = Mes abreviado (enero, febrero, etc.)%B = Mes completo (enero, febrero, etc.)%y = Año de dos dígitos (p. ej.p. ej. 01, 02, 03, 04)%b =   
Mes abreviado (enero, febrero, etc.)  
%B = Mes completo (enero, febrero, etc.)  
%y = Año de 2 dígitos (p. ej. 89)  
%Y = Año de 4 dígitos (p. ej. 1989)  
%h = Horas (reloj de 24 horas)  
%m = Minutos%s   
= Segundos%z   
= Desplazamiento respecto a GMT%Z   
= Huso horario (carácter)

Un ejemplo de formato de la fecha de hoy:

Ten en cuenta que si utiliza str\_glue(), Ten en cuenta que dentro de las esperadas comillas dobles " sólo debe utilizar comillas simples (como arriba).

### Mes-Año

Para convertir una columna de fecha al formato mes-año, le sugerimos que utilice la función as.yearmon() del paquete **zoo**. Esto convierte la fecha al tipo "yearmon" y mantiene el orden correcto. Por el contrario, usar format(columna, "%Y %B") convertirá al tipo Carácter y ordenará los valores alfabéticamente (incorrectamente).

A continuación, se crea una nueva columna yearmonth a partir de la columna date\_onset, utilizando la función as.yearmon(). La ordenación por defecto (correcta) de los valores resultantes se muestra en la tabla.

Por el contrario, se puede ver cómo sólo utilizando format() se consigue el formato de visualización deseado, pero no el orden correcto.

Nota: si está trabajando dentro de un ggplot() y quiere ajustar sólo cómo se muestran las fechas, puede ser suficiente proporcionar un formato strptime al argumento date\_labels = en scale\_x\_date() - puede utilizar "%b %Y" o "%Y %b". Consulte la página de [consejos de ggplot](#ggplot-tips).

**zoo** también ofrece la función as.yearqtr(), y puedes usar scale\_x\_yearmon() cuando uses ggplot().

## Semanas epidemiológicas

### ****lubridate****

Consulte la página sobre [Agrupar datos](#grouping-data) para ver ejemplos más extensos de agrupación de datos por fecha. A continuación describimos brevemente la agrupación de datos por semanas.

Generalmente recomendamos utilizar la función floor\_date() de **lubridate**, con el argumento unidad = "semana". Esto redondea la fecha hacia abajo al "inicio" de la semana, como se define por el argumento week\_start =. El inicio de la semana por defecto es el 1 (para los lunes), pero se puede especificar cualquier día de la semana como inicio (por ejemplo, el 7 para los domingos). floor\_date() es versátil y se puede utilizar para redondear hacia abajo a otras unidades de tiempo estableciendo la unidad = a "segundo", "minuto", "hora", "día", "mes" o "año".

El valor devuelto es la fecha de inicio de la semana, en tipo Date. El tipo Date es útil a la hora de trazar los datos, ya que serán fácilmente reconocidos y ordenados correctamente por ggplot().

Si sólo está interesado en ajustar las fechas para que se muestren por semanas en un gráfico, consulte la sección de esta página sobre Visualización de fechas. Por ejemplo, al trazar una epicurva puede formatear la visualización de la fecha proporcionando la nomenclatura strptime "%" deseada. Por ejemplo, utiliza "%Y-%W" o "%Y-%U" para devolver el año y el número de semana (dado el comienzo de la semana del lunes o del domingo, respectivamente).

### Recuentos semanales

Consulte la página sobre [Agrupar datos](#grouping-data) para obtener una explicación detallada de la agrupación de datos con count(), group\_by() y summarise(). A continuación se muestra un breve ejemplo.

1. Crear una nueva columna "semana" con mutate(), utilizando floor\_date() con unidad = "semana"
2. Obtenga el recuento de filas (casos) por semana con count(); filtre los casos a los que les falte la fecha
3. Termine con complete() de **tidyr** para asegurarse de que todas las semanas aparecen en los datos - incluso las que no tienen filas/casos. Por defecto, los valores de recuento para cualquier fila "nueva" son NA, pero puede hacerlos 0 con el argumento fill =, que espera una lista con nombre (abajo, n es el nombre de la columna de recuentos).

Aquí están las primeras filas del dataframe resultante:

### Alternativas a la Epiweek

Ten en cuenta que **lubridate** también tiene las funciones week(), epiweek() e isoweek(), cada una de las cuales tiene fechas de inicio ligeramente diferentes y otros matices. Sin embargo, en términos generales, floor\_date() debería ser todo lo que necesita. Lea los detalles de estas funciones introduciendo ?week en la consola o leyendo la documentación [aquí](https://www.rdocumentation.org/packages/lubridate/versions/1.7.4/topics/week).

Puedes considerar el uso del paquete **aweek** para establecer semanas epidemiológicas. Puedes leer más sobre él [en el sitio web de RECON](https://www.repidemicsconsortium.org/aweek/). Tiene las funciones date2week() y week2date() en las que puede establecer el día de inicio de la semana con week\_start = "lunes". Este paquete es el más fácil si se desea obtener resultados del tipo "semana" (por ejemplo, "2020-W12"). Otra ventaja de **aweek** es que cuando date2week() se aplica a una columna de fecha, la columna devuelta (formato de semana) es automáticamente del tipo Factor e incluye niveles para todas las semanas en el lapso de tiempo (esto evita el paso extra de complete() descrito anteriormente). Sin embargo, **aweek** no tiene la funcionalidad de redondear fechas a otras unidades de tiempo como meses, años, etc.

Otra alternativa para las series temporales que también funciona bien para mostrar un formato de "semana" ("2020 W12") es yearweek() del paquete **tsibble**, como se demuestra en la página sobre [series temporales y detección de brotes](#time-series-and-outbreak-detection).

## Conversión de fechas/zonas horarias

Cuando los datos están presentes en diferentes husos horarios, a menudo puede ser importante normalizar estos datos en un huso horario unificado. Esto puede suponer un reto adicional, ya que el componente de zona horaria de los datos debe codificarse manualmente en la mayoría de los casos.

En R, cada objeto datetime tiene un componente de zona horaria. Por defecto, todos los objetos datetime llevarán la zona horaria local para el ordenador que se está utilizando - esto es generalmente específico para una ubicación en lugar de una zona horaria nombrada, ya que las zonas horarias a menudo cambian en los lugares debido al horario de verano. No es posible compensar con precisión las zonas horarias sin un componente de tiempo de una fecha, ya que el evento que representa una columna de fecha no puede ser atribuido a un tiempo específico, y por lo tanto los cambios de tiempo medidos en horas no pueden ser razonablemente contabilizados.

Para tratar las zonas horarias, hay una serie de funciones de ayuda en lubridate que pueden utilizarse para cambiar la zona horaria de un objeto datetime de la zona horaria local a una zona horaria diferente. Las zonas horarias se establecen atribuyendo una zona horaria válida de la base de datos tz al objeto datetime. Una lista de estos se puede encontrar aquí - si la ubicación que está utilizando los datos no está en esta lista, las grandes ciudades cercanas en la zona horaria están disponibles y sirven el mismo propósito.

<https://en.wikipedia.org/wiki/List_of_tz_database_time_zones>

Esto puede parecer muy abstracto, y a menudo no es necesario si el usuario no está trabajando a través de zonas horarias.

## Cálculos de retardo y de avance

lead() y lag() son funciones del paquete **dplyr** que ayudan a encontrar los valores anteriores (retardados) o posteriores (principales) en un vector, normalmente un vector numérico o de fechas. Esto es útil cuando se hacen cálculos de cambio/diferencia entre unidades de tiempo.

Supongamos que se quiere calcular la diferencia de casos entre una semana actual y la anterior. Los datos se proporcionan inicialmente en recuentos semanales, como se muestra a continuación.

**Al utilizar lag() o lead(), el orden de las filas en el dataframe es muy importante. - presta atención a si tus fechas/números son ascendentes o descendentes**

En primer lugar, cree una nueva columna que contenga el valor de la semana anterior (retardada).

* Controla el número de unidades hacia atrás/adelante con n = (debe ser un entero no negativo)
* Utiliza default = para definir el valor colocado en las filas no existentes (por ejemplo, la primera fila para la que no hay un valor retardado). Por defecto es NA.
* Utiliza order\_by = TRUE si tus filas no están ordenadas por su columna de referencia

A continuación, cree una nueva columna que sea la diferencia entre las dos columnas de los casos:

Puedes leer más sobre lead() y lag() en la documentación [aquí](https://dplyr.tidyverse.org/reference/lead-lag.html) o introduciendo ?lag en tu consola.

## Recursos

**Página de lubridate** [tidyverse](https://lubridate.tidyverse.org/)  
  
Página de **lubridate** RStudio [cheatsheetR](https://rawgit.com/rstudio/cheatsheets/master/lubridate.pdf)   
for Data Science sobre [fechas y horas](https://r4ds.had.co.nz/dates-and-times.html)  
[Tutorial en línea](https://www.statmethods.net/input/dates.html) [Formatos de fecha](https://www.r-bloggers.com/2013/08/date-formats-in-r/)

# su #Caracteres y cadenas

{#characters-and-strings}

Esta página demuestra el uso del paquete **stringr** para evaluar y manejar valores de caracteres ("cadenas").

1. Combinar, ordenar, dividir, organizar - str\_c(), str\_glue(), str\_order(), str\_split()
2. Limpiar y normalizar
   * Ajustar la longitud - str\_pad(), str\_trunc(), str\_wrap()
   * Cambio de mayúsculas y minúsculas - str\_to\_upper(), str\_to\_title(), str\_to\_lower(), str\_to\_sentence()
3. Evaluar y extraer por posición - str\_length(), str\_sub(), word()
4. Patrones
   * Detectar y localizar - str\_detect(), str\_subset(), str\_match(), str\_extract()
   * Modificar y reemplazar - str\_sub(), str\_replace\_all()
5. Expresiones regulares ("regex")

Para facilitar la visualización, la mayoría de los ejemplos se muestran actuando sobre un vector de caracteres definido brevemente, aunque pueden adaptarse fácilmente a una columna dentro de un dataframe.

Esta [viñeta de stringr](https://cran.r-project.org/web/packages/stringr/vignettes/stringr.html) proporcionó gran parte de la inspiración para esta página.

## Preparación

### Cargar paquetes

Instala o carga el paquete **stringr** y otros paquetes **tidyverse**.

### Importar datos

En esta página haremos referencia de vez en cuando a la lista limpia de casos de una epidemia de ébola simulada. Si quiere seguir el proceso, [clica para descargar la lista de casos "limpia"](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/case_linelists/linelist_cleaned.rds) (como archivo .rds). Importe los datos con la función import() del paquete **rio** (maneja muchos tipos de archivos como .xlsx, .csv, .rds - vea la página de [importación y exportación](#import-and-export) para más detalles).

A continuación se muestran las primeras 50 filas del listado.

## Unir, dividir y ordenar

Esta sección abarca:

* Uso de str\_c(), str\_glue() y unite() para combinar cadenas
* Uso de str\_order() para ordenar las cadenas
* Uso de str\_split() y separate() para dividir cadenas

### Combinar cadenas

Para combinar o concatenar varias cadenas en una sola, sugerimos utilizar str\_c de **stringr**. Si tiene valores de caracteres distintos para combinar, simplemente proporciónelos como argumentos únicos, separados por comas.

El argumento sep = inserta un valor de carácter entre cada uno de los argumentos proporcionados (por ejemplo, insertando una coma, un espacio o una nueva línea "\n")

El argumento colapso = es relevante si está introduciendo múltiples vectores como argumentos a str\_c(). Se utiliza para separar los elementos de lo que sería un vector de salida, de forma que el vector de salida sólo tenga un elemento de carácter largo.

El ejemplo siguiente muestra la combinación de dos vectores en uno (nombres y apellidos). Otro ejemplo similar podría ser el de las jurisdicciones y su número de casos. En este ejemplo:

* El valor sep = aparece entre cada nombre y apellido
* El valor de colapso = aparece entre cada persona

Nota: Dependiendo del contexto de visualización deseado, al imprimir una cadena combinada de este tipo con nuevas líneas, puede ser necesario envolver toda la frase en cat() para que las nuevas líneas se impriman correctamente:

### Cadenas dinámicas

Utiliza str\_glue() para insertar código R dinámico en una cadena. Se trata de una función muy útil para crear pies de foto dinámicos, como se demuestra a continuación.

* Todo el contenido va entre comillas dobles str\_glue("")
* Cualquier código dinámico o referencias a valores predefinidos se colocan entre llaves {} dentro de las comillas dobles. Puedes haber muchas llaves en el mismo comando str\_glue().
* Para mostrar las comillas de caracteres '', utiliza comillas simples dentro de las comillas dobles que las rodean (por ejemplo, al proporcionar el formato de la fecha - véase el ejemplo siguiente)
* Consejo: Puedes utilizar \N para forzar una nueva línea
* Consejo: Utiliza format() para ajustar la visualización de la fecha, y utiliza Sys.Date() para mostrar la fecha actual

Un ejemplo sencillo, de un título de gráfico dinámico:

Un formato alternativo es utilizar marcadores de posición dentro de los paréntesis y definir el código en argumentos separados al final de la función str\_glue(), como se indica a continuación. Esto puede mejorar la legibilidad del código si el texto es largo.

**Sacar de un dataframe**

A veces, es útil extraer datos de un dataframe y pegarlos en secuencia. A continuación se muestra un ejemplo de dataframe. Lo utilizaremos para hacer una declaración resumida sobre las jurisdicciones y los recuentos de casos nuevos y totales.

Utiliza str\_glue\_data(), que está hecho especialmente para tomar datos de las filas del dataframe:

**Combinar cadenas a través de las filas**

Si está intentando "enrollar" valores en una columna del dataframe, por ejemplo, combinar valores de varias filas en una sola fila pegándolos con un separador, consulte la sección de la página de [Desduplicación](#de-duplication) sobre ["enrollar" valores](#str_rollup).

**dataframe a una línea**

Puedes hacer que la declaración aparezca en una línea utilizando str\_c() (especificando el dataframe y los nombres de las columnas), y proporcionando los argumentos sep = y collapse =.

Podría añadir el texto pre-fijo "Nuevos casos:" al principio de la sentencia envolviendo con un str\_c() separado (si "Nuevos casos:" estuviera dentro del str\_c() original aparecería varias veces).

### Unir columnas

Dentro de un dataframe, la unión de valores de caracteres de varias columnas puede lograrse con unite() de **tidyr**. Esto es lo contrario de separate().

Indique el nombre de la nueva columna unida. A continuación, indique los nombres de las columnas que desea unir.

* Por defecto, el separador utilizado en la columna unida es el guión bajo \_, pero puede cambiarse con el argumento sep =.
* remove = elimina las columnas de entrada del dataframe (TRUE por defecto)
* na.rm = elimina los valores faltantes al unir (FALSE por defecto)

A continuación, definimos un mini-dataframe con el que hacer una demostración:

Este es el dataframe de ejemplo:

A continuación, unimos las tres columnas de síntomas:

### Dividir

Para dividir una cadena basada en un patrón, utiliza str\_split(). Evalúa la(s) cadena(s) y devuelve una lista de vectores de caracteres formada por los valores recién divididos.

El sencillo ejemplo que sigue evalúa una cadena y la divide en tres. Por defecto, devuelve un objeto de tipo lista con un elemento (un vector de caracteres) por cada cadena proporcionada inicialmente. Si simplifica = TRUE devuelve una matriz de caracteres.

En este ejemplo, se proporciona una cadena y la función devuelve una lista con un elemento: un vector de caracteres con tres valores.

Si la salida se guarda, puede acceder al enésimo valor dividido con la sintaxis de corchetes. Para acceder a un valor específico puede utilizar una sintaxis como esta: el\_objeto\_devuelto[[1]][2], que accedería al segundo valor de la primera cadena evaluada ("fiebre"). Consulte la página de [fundamentos de R](#r-basics) para obtener más detalles sobre el acceso a los elementos.

Si se proporcionan varias cadenas mediante str\_split(), habrá más de un elemento en la lista devuelta.

Para devolver una "matriz de caracteres" en su lugar, que puede ser útil si se crean columnas de dataframes, establezca el argumento simplificar = TRUE como se muestra a continuación:

También puede ajustar el número de divisiones a crear con el argumento n =. Por ejemplo, esto restringe el número de divisiones a 2. Cualquier otra coma permanece dentro de los segundos valores.

Nota - los mismos resultados se pueden conseguir con *str\_split\_fixed()*, en la que no se da el argumento *simplificar, sino que se* debe designar el número de columnas (*n*).

### Columnas divididas

Si está intentando dividir una columna de un dataframe, es mejor utilizar la función separate() de **dplyr**. Se utiliza para dividir una columna de caracteres en otras columnas.

Digamos que tenemos un dataframe simple df (definido y unido en la [sección de unión](#str_unite)) que contiene una columna case\_ID, una columna de caracteres con muchos síntomas y una columna de resultados. Nuestro objetivo es separar la columna de síntomas en varias columnas, cada una de las cuales contiene un síntoma.

Asumiendo que los datos son canalizados en separate(), primero proporcione la columna a separar. A continuación, proporcione en = como un vector c() que contiene los nombres de las nuevas columnas, como se muestra a continuación.

* sep = el separador, puede ser un carácter, o un número (interpretado como la posición del carácter a dividir)
* remove = FALSE por defecto, elimina la columna de entrada
* convert = FALSE por defecto, hará que las cadenas "NA "s se conviertan en NA
* extra = controla lo que sucede si hay más valores creados por la separación que nuevas columnas nombradas.
  + extra = "warn" significa que verá una advertencia, pero dejará caer los valores en exceso (**el valor por defecto**)
  + extra = "drop" significa que los valores sobrantes se eliminarán sin previo aviso
  + **extra = "merge" sólo dividirá hasta el número de nuevas columnas listadas en into - esta configuración preservará todos tus datos**

A continuación se muestra un ejemplo con extra = "merge" - no se pierde ningún dato. Se definen dos nuevas columnas pero cualquier tercer síntoma se deja en la segunda columna nueva:

Cuando se utiliza el extra = "drop" por defecto a continuación, se da una advertencia pero se pierden los terceros síntomas:

**ATENCIÓN:** Si no proporcionas suficientes valores en las nuevas columnas, tus datos pueden quedar truncados.

### Ordenar alfabéticamente

Se pueden ordenar varias cadenas por orden alfabético. str\_order() devuelve el orden, mientras que str\_sort() devuelve las cadenas en ese orden.

Para utilizar un alfabeto diferente, añada el argumento locale =. Vea la lista completa de locales introduciendo stringi::stri\_locale\_list() en la consola de R.

### funciones base de R

Es común ver las funciones **base de** R paste() y paste0(), que concatenan vectores después de convertir todas las partes en caracteres. Actúan de forma similar a str\_c() pero la sintaxis es posiblemente más complicada - en los paréntesis cada parte está separada por una coma. Las partes son o bien texto de carácter (entre comillas) o bien objetos de código predefinidos (sin comillas). Por ejemplo:

Se pueden especificar los argumentos sep = y collapse =. paste() es simplemente paste0() con un sep = " " por defecto (un espacio).

## Limpiar y normalizar

### Cambiar de caso

A menudo hay que alterar las mayúsculas y minúsculas de un valor de cadena, por ejemplo los nombres de las jurisdicciones. Utiliza str\_to\_upper(), str\_to\_lower() y str\_to\_title(), de **stringr**, como se muestra a continuación:

Usando \*base\*\* R, lo anterior también se puede lograr con toupper(), tolower().

**Caso del título**

La transformación de la cadena para que cada palabra esté en mayúsculas puede lograrse con str\_to\_title():

Utiliza toTitleCase() del paquete de **herramientas** para lograr una capitalización más matizada (palabras como "a", "el" y "de" no se escriben en mayúsculas).

También puede utilizar str\_to\_sentence(), que sólo pone en mayúsculas la primera letra de la cadena.

### Longitud de la almohadilla

Utiliza str\_pad() para añadir caracteres a una cadena, con una longitud mínima. Por defecto se añaden espacios, pero también puede rellenar con otros caracteres utilizando el argumento pad =.

Por ejemplo, para rellenar números con ceros a la izquierda (como en el caso de las horas o los minutos), puede rellenar el número hasta una longitud mínima de 2 con pad = "0".

### Truncar

str\_trunc() establece una longitud máxima para cada cadena. Si una cadena supera esta longitud, se trunca (acorta) y se incluye una elipsis (...) para indicar que la cadena era antes más larga. Ten en cuenta que la elipsis se cuenta en la longitud. Los caracteres de la elipsis pueden cambiarse con el argumento ellipsis =. El argumento opcional side = especifica dónde aparecerá la elipsis dentro de la cadena truncada ("izquierda", "derecha" o "centro").

### Normalizar la longitud

Utiliza str\_trunc() para establecer una longitud máxima y, a continuación, utiliza str\_pad() para ampliar las cadenas muy cortas hasta esa longitud truncada. En el ejemplo siguiente, se establece 6 como longitud máxima (se trunca un valor), y luego se rellena un valor muy corto para alcanzar la longitud de 6.

### Eliminar los espacios en blanco iniciales y finales

Utiliza str\_trim() para eliminar los espacios, las nuevas líneas (\n) o los tabuladores (\t) de los lados de una cadena de entrada. Añada "derecha" "izquierda" o "ambos" al comando para especificar qué lado recortar (por ejemplo, str\_trim(x, "derecha").

### Eliminar los espacios en blanco repetidos dentro de

Utiliza str\_squish() para eliminar los espacios repetidos que aparecen dentro de una cadena. Por ejemplo, para convertir espacios dobles en espacios simples. También elimina espacios, nuevas líneas o tabulaciones en el exterior de la cadena como str\_trim().

Escribe ?str\_trim, ?str\_pad en tu consola de R para ver más detalles.

### Envolver en párrafos

Utiliza str\_wrap() para envolver un texto largo no estructurado en un párrafo estructurado con una longitud de línea fija. Proporciona la longitud de caracteres ideal para cada línea, y aplica un algoritmo para insertar nuevas líneas (\n) dentro del párrafo, como se ve en el ejemplo siguiente.

La función **base** cat() puede envolver el comando anterior para imprimir la salida, mostrando las nuevas líneas añadidas.

## Manejar por posición

### Extraer por posición de carácter

Utiliza str\_sub() para devolver sólo una parte de una cadena. La función toma tres argumentos principales:

1. el(los) vector(es) de caracteres
2. posición de inicio
3. posición final

Algunas notas sobre los números de posición:

* Si un número de posición es positivo, la posición se cuenta a partir del extremo izquierdo de la cadena.
* Si un número de posición es negativo, se cuenta a partir del extremo derecho de la cadena.
* Los números de posición son inclusivos.
* Las posiciones que se extienden más allá de la cadena serán truncadas (eliminadas).

A continuación se muestran algunos ejemplos aplicados a la cadena "neumonía":

### Extraer por posición de palabra

Para extraer la enésima 'palabra', utiliza word(), también de **stringr**. Proporcione la(s) cadena(s), luego la primera posición de la palabra a extraer, y la última posición de la palabra a extraer.

Por defecto, se asume que el separador entre 'palabras' es un espacio, a menos que se indique lo contrario con sep = (por ejemplo, sep = "\_" cuando las palabras están separadas por barra baja.

### Sustituir por posición de carácter

str\_sub() emparejado con el operador de asignación (<-) puede utilizarse para modificar una parte de una cadena:

Un ejemplo aplicado a varias cadenas (por ejemplo, una columna). Obsérvese la ampliación de la longitud de "VIH".

### Evaluar la longitud

Como alternativa, utiliza nchar() de la **base** R

## Patrones

Muchas funciones **de stringr** trabajan para detectar, localizar, extraer, hacer coincidir, reemplazar y dividir basándose en un patrón especificado.

### Detectar un patrón

Utiliza str\_detect() como se indica a continuación para detectar la presencia/ausencia de un patrón dentro de una cadena. Primero proporcione la cadena o vector a buscar (cadena =), y luego el patrón a buscar (patrón =). Ten en cuenta que, por defecto, la búsqueda distingue entre mayúsculas y minúsculas.

Se puede incluir el argumento negate = y ponerlo a TRUE si se quiere saber si el patrón NO está presente.

Para ignorar las mayúsculas y minúsculas, envuelva el patrón dentro de regex(), y dentro de regex() añada el argumento ignore\_case = TRUE (o T como abreviatura).

Cuando str\_detect() se aplica a un vector de caracteres o a una columna de un dataframe, devolverá TRUE o FALSE para cada uno de los valores.

Si necesita contar los TRUE, simplemente sume() la salida. Esto cuenta el número de TRUE.

Para buscar con varios términos, inclúyalos separados por barras OR (|) dentro del argumento pattern =, como se muestra a continuación:

Si necesita construir una larga lista de términos de búsqueda, puede combinarlos usando str\_c() y sep = |, luego definir esto es un objeto de caracteres, y luego referenciar el vector más adelante de manera más sucinta. El ejemplo siguiente incluye posibles términos de búsqueda de ocupación para proveedores médicos de primera línea.

Este comando devuelve el número de ocupaciones que contienen alguno de los términos de búsqueda para proveedores médicos de primera línea (occupation\_med\_frontline):

**Funciones de búsqueda de cadenas en la base R**

La función **base** grepl() funciona de forma similar a str\_detect(), en el sentido de que busca coincidencias con un patrón y devuelve un vector lógico. La sintaxis básica es grepl(patrón, cadenas\_de\_búsqueda, ignorar.caso = FALSE, ...). Una ventaja es que el argumento ignore.case es más fácil de escribir (no hay necesidad de involucrar la función regex()).

Asimismo, las funciones **base sub(**) y gsub() actúan de forma similar a str\_replace(). Su sintaxis básica es: gsub(patrón, reemplazo, cadenas\_de\_búsqueda, ignore.case = FALSE). sub() reemplazará la primera instancia del patrón, mientras que gsub() reemplazará todas las instancias del patrón.

#### Convertir comas en puntos

He aquí un ejemplo de uso de gsub() para convertir comas en puntos en un vector de números. Esto podría ser útil si tus datos proceden de otras partes del mundo que no sean Estados Unidos o Gran Bretaña.

El gsub() interno que actúa primero sobre las longitudes está convirtiendo cualquier punto en sin espacio "". El carácter de punto"." tiene que ser "escapado" con dos barras inclinadas para significar realmente un punto, porque "." en regex significa "cualquier carácter". A continuación, el resultado (con sólo comas) se pasa a la función externa gsub() en la que las comas se sustituyen por puntos.

### Sustituir todo

Utiliza str\_replace\_all() como herramienta de "búsqueda y sustitución". Primero, proporcione las cadenas a evaluar a string =, luego el patrón a reemplazar a pattern =, y luego el valor de reemplazo a replacement =. El ejemplo siguiente reemplaza todas las instancias de "dead" con "deceased". Ten en cuenta que esto distingue entre mayúsculas y minúsculas.

Notas:

* Para sustituir un patrón por NA, utiliza str\_replace\_na().
* La función str\_replace() reemplaza sólo la primera instancia del patrón dentro de cada cadena evaluada.

### Detectar dentro de la lógica

**Dentro de case\_when()**

str\_detect() se utiliza a menudo dentro de case\_when() (de **dplyr**). Digamos que ocupaciones es una columna en linelist. La función mutate() de abajo crea una nueva columna llamada is\_educator utilizando la lógica condicional a través de case\_when(). Vea la página sobre limpieza de datos para aprender más sobre case\_when().

Como recordatorio, puede ser importante añadir criterios de exclusión a la lógica condicional (negar = F):

### Localizar la posición del patrón

Para localizar la primera posición de un patrón, utiliza str\_locate(). Esta función da como resultado una posición inicial y una final.

Al igual que otras funciones str, existe una versión "\_all" (str\_locate\_all()) que devolverá las posiciones de todas las instancias del patrón dentro de cada cadena. La salida es una lista.

### Extraer una coincidencia

str\_extract\_all() devuelve los patrones coincidentes en sí mismos, lo que resulta muy útil cuando se han ofrecido varios patrones mediante condiciones "OR". Por ejemplo, buscando en el vector de cadenas de ocupaciones (véase la pestaña anterior) ya sea "enseñar", "profesor" o "tutor".

str\_extract\_all() devuelve una lista que contiene todas las coincidencias de cada cadena evaluada. Vea a continuación cómo la ocupación 3 tiene dos coincidencias de patrón dentro de ella.

str\_extract() extrae sólo la primera coincidencia en cada cadena evaluada, produciendo un vector de caracteres con un elemento por cada cadena evaluada. Devuelve NA cuando no hay coincidencias. Los NAs pueden ser eliminados envolviendo el vector devuelto con na.exclude(). Observe cómo la segunda de las coincidencias de la ocupación 3 no se muestra.

### Subconjunto y recuento

Las funciones alineadas incluyen str\_subset() y str\_count().

str\_subset() devuelve los valores reales que contienen el patrón:

str\_count() devuelve un vector de números: el **número de veces que** aparece un término de búsqueda en cada valor evaluado.

### Grupos Regex

EN CONSTRUCCIÓN

## Caracteres especiales

**Barra invertida como escape**

La barra invertida \N se utiliza para "escapar" del significado del siguiente carácter. De este modo, se puede utilizar una barra invertida para que una comilla aparezca dentro de otras comillas (\") - la comilla del medio no "romperá" las comillas circundantes.

Nota - por lo tanto, si quiere mostrar una barra invertida, debe escapar su significado con otra barra invertida. Así que debe escribir dos barras invertidas \ ~ para mostrar uno.

**Caracteres especiales**

| **Carácter especial** | **Representa a** |
| --- | --- |
| "\\" | barra invertida |
| "\n" | una nueva línea (newline) |
| "\"" | comillas dobles dentro de comillas dobles |
| '\'' | comillas simples dentro de comillas simples |
| "\N - acento grave - retorno de carro - tabulación - tabulación vertical". | retroceso |

Ejecuta ?"'" en la consola de R para mostrar una lista completa de estos caracteres especiales (aparecerá en el panel de ayuda de RStudio).

## Expresiones regulares (regex)

## Regex y caracteres especiales

Las expresiones regulares, o "regex", son un lenguaje conciso para describir patrones en las cadenas. Si no está familiarizado con él, una expresión regular puede parecer un lenguaje extraño. Aquí tratamos de desmitificar un poco este lenguaje.

Gran parte de esta sección está adaptada de [*este tutorial*](https://towardsdatascience.com/a-gentle-introduction-to-regular-expressions-with-r-df5e897ca432) y de [*esta hoja de trucos*](https://evoldyn.gitlab.io/evomics-2018/ref-sheets/R_strings.pdf). Aquí adaptamos selectivamente sabiendo que este manual podría ser visto por personas sin acceso a internet para ver los otros tutoriales.

Una expresión regular se aplica a menudo para extraer patrones específicos de texto "no estructurado", por ejemplo, notas médicas, quejas principales, historial del paciente u otras columnas de texto libre en un dataframe.

Hay cuatro herramientas básicas que se pueden utilizar para crear una expresión regular básica:

1. Juegos de caracteres
2. Metacaracteres
3. Cuantificadores
4. Grupos

**Juegos de caracteres**

Los conjuntos de caracteres, son una forma de expresar las opciones de la lista para una coincidencia de caracteres, entre paréntesis. Así, cualquier coincidencia se activará si cualquiera de los caracteres dentro de los paréntesis se encuentra en la cadena. Por ejemplo, para buscar vocales se podría utilizar este conjunto de caracteres "[aeiou]". Otros conjuntos de caracteres comunes son:

| **Juego de caracteres** | **Partidos para** |
| --- | --- |
| "[A-Z]" | cualquier letra mayúscula |
| "[a-z]" | cualquier letra minúscula |
| "[0-9]" | cualquier dígito |
| [:alnum:] | cualquier carácter alfanumérico |
| [:dígito:] | cualquier dígito numérico |
| [:alpha:] | cualquier letra (mayúscula o minúscula) |
| [:superior:] | cualquier letra mayúscula |
| [:lower:] | cualquier letra minúscula |

Los conjuntos de caracteres pueden combinarse dentro de un paréntesis (¡sin espacios!), como "[A-Za-z]" (cualquier letra mayúscula o minúscula), u otro ejemplo "[t-z0-5]" (de la t a la z en minúscula o del número 0 al 5).

**Metacaracteres**

Los metacaracteres son la abreviatura de los conjuntos de caracteres. A continuación se enumeran algunos de los más importantes:

| **Meta carácter** | **Representa a** |
| --- | --- |
| "\\s" | un solo espacio |
| "\\w" | cualquier carácter alfanumérico (A-Z, a-z, o 0-9) |
| "\\d" | cualquier dígito numérico (0-9) |

**Cuantificadores**

Normalmente no se desea buscar una coincidencia en un solo carácter. Los cuantificadores le permiten designar la longitud de las letras/números para permitir la coincidencia.

Los cuantificadores son números escritos entre corchetes { } después del carácter que cuantifican, por ejemplo,

* "A{2}" devolverá instancias de **dos** letras A mayúsculas.
* "A{2,4}" devolverá instancias de **entre dos y cuatro** letras A mayúsculas (¡no ponga espacios!).
* "A{2,}" devolverá instancias de **dos o más** letras A mayúsculas.
* "A+" devolverá instancias de **una o más** letras A mayúsculas (grupo extendido hasta que se encuentre un carácter diferente).
* Preceder con un asterisco \* para devolver **cero o más** coincidencias (útil si no está seguro de que el patrón está presente)

Utilizando el símbolo + como cuantificador, la coincidencia se producirá hasta que se encuentre un carácter diferente. Por ejemplo, esta expresión devolverá todas las palabras (caracteres alfa: "[A-Za-z]+"

Cuando se utiliza un cuantificador de {2}, sólo se devuelven los pares de A consecutivos. Se identifican dos pares dentro de AAAA.

Cuando se utiliza un cuantificador de {2,4}, se devuelven grupos de A consecutivos de dos a cuatro.

Con el cuantificador +, se devuelven grupos de **uno o más**:

**Posición relativa**

Expresan los requisitos de lo que precede o sigue a un patrón. Por ejemplo, para extraer frases, "dos números que van seguidos de un punto" (""). (?<=\a.)\a(?=[A-Z])

| **Declaración de posición** | **Coincide con** |
| --- | --- |
| "(?<=b)a" | **"**a" que **va precedida** de una "b" |
| "(?<!b)a" | **"**a" que **NO va precedida** de una "b" |
| "a(?=b)" | "a" que **va seguida** de una "b" |
| "a(?!b)" | "a" que **NO va** seguida de una "b" |

**Grupos**

La captura de grupos en su expresión regular es una forma de tener una salida más organizada al momento de la extracción.

**Ejemplos de Regex**

A continuación se presenta un texto libre para los ejemplos. Intentaremos extraer información útil del mismo utilizando un término de búsqueda de expresión regular.

Esta expresión coincide con todas las palabras (cualquier carácter hasta llegar a un no carácter como un espacio):

La expresión "[0-9]{1,2}" coincide con números consecutivos de 1 o 2 dígitos. También podría escribirse "\d{1,2}", o "[:digit:]{1,2}".

Puedes ver una lista útil de expresiones regex y consejos en la página 2 de [esta hoja de trucos](https://evoldyn.gitlab.io/evomics-2018/ref-sheets/R_strings.pdf)

Vea también este [tutorial](https://towardsdatascience.com/a-gentle-introduction-to-regular-expressions-with-r-df5e897ca432).

## Recursos

Puedes encontrar una hoja de referencia para las funciones **de stringr** [aquí](https://evoldyn.gitlab.io/evomics-2018/ref-sheets/R_strings.pdf)

Puedes encontrar una viñeta sobre **stringr** [aquí](https://cran.r-project.org/web/packages/stringr/vignettes/stringr.html)

# #Factores

{#factors}

En R, los factores son un tipo de datos que permiten categorías ordenadas con un conjunto fijo de valores aceptables.

Normalmente, se convierte una columna de tipo numérico o de caracteres en un factor si se desea establecer un orden intrínseco a los valores ("niveles") para que puedan mostrarse de forma no alfabética en gráficos y tablas. Otro uso común de los factores es normalizar las leyendas de los gráficos para que no fluctúen si ciertos valores están temporalmente ausentes de los datos.

En esta página se muestra el uso de las funciones del paquete **forcats** (nombre abreviado de "**Para** variables categóricas") y algunas funciones **básicas de** R. También se aborda el uso de **lubridate** y **aweek** para casos de factores especiales relacionados con semanas epidemiológicas.

Puedes encontrar una lista completa de las funciones de **los forcats** en línea [aquí](https://forcats.tidyverse.org/reference/index.html). A continuación mostramos algunas de las más comunes.

## Preparación

### Cargar paquetes

Este trozo de código muestra la carga de los paquetes necesarios para los análisis. En este manual destacamos p\_load() de **pacman**, que instala el paquete si es necesario y lo carga para su uso. También puede cargar los paquetes instalados con library() de R **base.** Consulte la página sobre [los fundamentos de R](#r-basics) para obtener más información sobre los paquetes de R.

### Importar datos

Importamos los datos de casos de una epidemia de ébola simulada. Si quieres seguir el proceso, [clica para descargar linelist "limpia"](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/case_linelists/linelist_cleaned.rds) (como archivo .rds). Importe sus datos con la función import() del paquete **rio** (acepta muchos tipos de archivos como .xlsx, .rds, .csv - vea la página de [importación y exportación](#import-and-export) para más detalles).

### Nueva variable categórica

Para la demostración en esta página utilizaremos un escenario común - la creación de una nueva variable categórica.

Ten en cuenta que si convierte una columna numérica en una de tipo factor, no podrás calcular estadísticas numéricas sobre ella.

#### Crear columna

Utilizamos la columna existente days\_onset\_hosp (días desde el inicio de los síntomas hasta el ingreso en el hospital) y creamos una nueva columna delay\_cat clasificando cada fila en una de varias categorías. Lo hacemos con la función **dplyr** case\_when(), que aplica secuencialmente criterios lógicos (lado derecho) a cada fila y devuelve el valor correspondiente del lado izquierdo para la nueva columna delay\_cat. Lea más sobre case\_when() en [Limpieza de datos y funciones básicas](#cleaning-data-and-core-functions).

#### Orden de valores por defecto

Tal y como se creó con case\_when(), la nueva columna delay\_cat es una columna categórica del tipo Character - aún no es un factor. Así, en una tabla de frecuencia, vemos que los valores únicos aparecen en un orden alfanumérico por defecto - un orden que no tiene mucho sentido intuitivo:

Del mismo modo, si hacemos un gráfico de barras, los valores también aparecen en este orden en el eje x (ver la página de [fundamentos de ggplot](#ggplot-basics) para más información sobre **ggplot2** - el paquete de visualización más común en R).

## Convertir en factor

Para convertir una columna numérica o de caracteres a de tipo factor, puedes utilizar cualquier función del paquete **forcats** (muchas se detallan [a continuación](#fct_adjust)). Las convertirán a de tipo factor y luego también realizarán o permitirán cierto ordenamiento de los niveles - por ejemplo usando fct\_relevel() le permite especificar manualmente el orden de los niveles. La función as\_factor() simplemente convierte el tipo sin ninguna otra capacidad.

La función **base de** R factor() convierte una columna en factor y permite especificar manualmente el orden de los niveles, como un vector de caracteres a su argumento niveles =.

A continuación utilizamos mutate() y fct\_relevel() para convertir la columna delay\_cat de tipo carácter a tipo factor. La columna delay\_cat se crea en la sección de [preparación](#fct_newcat) anterior.

Los "valores" únicos de esta columna se consideran ahora "niveles" del factor. Los niveles tienen un orden, que puede imprimirse con la función de R **base** levels(), o alternativamente verse en una tabla de recuento mediante table() de R **base** o tabyl() de **janitor**. Por defecto, el orden de los niveles será alfanumérico, como antes. Ten en cuenta que NA no es un nivel de factor.

La función fct\_relevel() tiene la utilidad adicional de permitirle especificar manualmente el orden de los niveles. Simplemente escriba los valores de nivel en orden, entre comillas, separados por comas, como se muestra a continuación. Ten en cuenta que la ortografía debe coincidir exactamente con los valores. Si desea crear niveles que no existen en los datos, utiliza [fct\_expanden su lugar](#fct_add)).

Ahora podemos ver que los niveles están ordenados, como se especificó en el comando anterior, en un orden sensato.

Ahora el orden de la trama también tiene un sentido más intuitivo.

## Añadir o quitar niveles

### Añadir

Si necesita añadir niveles a un factor, puede hacerlo con fct\_expand(). Basta con escribir el nombre de la columna seguido de los nuevos niveles (separados por comas). Al tabular los valores, podemos ver los nuevos niveles y los recuentos de cero. Puedes utilizar table() de R **base**, o tabyl() de **janitor**:

Nota: existe una función especial **de forcats** para añadir fácilmente valores faltantes (NA) como nivel. Véase la sección sobre valores [ausentes](#fct_missing) más adelante.

### Gota

Si utiliza fct\_drop(), los niveles "no utilizados" con recuento cero se eliminarán del conjunto de niveles. Los niveles que hemos añadido anteriormente ("No admitido en un hospital") existen como nivel, pero ninguna fila tiene realmente esos valores. Por tanto, se eliminarán aplicando fct\_drop() a nuestra columna de factores:

## Ajustar el orden de los niveles

El paquete **forcats** ofrece funciones útiles para ajustar fácilmente el orden de los niveles de un factor (después de haber definido una columna como de tipo factor):

Estas funciones pueden aplicarse a una columna de factores en dos contextos:

1. A la columna del dataframe, como es habitual, para que la transformación esté disponible para cualquier uso posterior de los datos
2. Dentro de un gráfico, para que el cambio se aplique sólo dentro del gráfico

### Manualmente

Esta función se utiliza para ordenar manualmente los niveles de los factores. Si se utiliza en una columna no factorial, la columna se convertirá primero en de tipo factor.

Dentro del paréntesis, indique primero el nombre de la columna del factor y, a continuación, indique

* Todos los niveles en el orden deseado (como un vector de caracteres c()), o
* Un nivel y se corrige la colocación utilizando el argumento after =

He aquí un ejemplo de redefinición de la columna delay\_cat (que ya es el tipo Factor) y especificando todo el orden de niveles deseado.

Si sólo quiere mover un nivel, puede especificarlo a fct\_relevel() solo y dar un número al argumento after = para indicar en qué lugar del orden debe estar. Por ejemplo, el comando siguiente desplaza "<2 días" a la segunda posición:

### Dentro de un gráfico

Los comandos **forcats** pueden utilizarse para establecer el orden de los niveles en el dataframe, o sólo dentro de un gráfico. Al utilizar el comando para "envolver" el nombre de la columna dentro del comando de trazado ggplot(), puede invertir/nivelar/etc. la transformación sólo se aplicará dentro de ese trazado.

A continuación, se crean dos gráficos con ggplot() (véase la página de [fundamentos de ggplot](#ggplot-basics)). En el primero, la columna delay\_cat se asigna al eje x del gráfico, con su orden de nivel por defecto como en linelist de datos. En el segundo ejemplo se envuelve dentro de fct\_relevel() y se cambia el orden en el gráfico.

Ten en cuenta que el título del eje x por defecto es ahora bastante complicado - puede sobrescribir este título con el argumento **ggplot2** labs().

### Invertir

Es bastante común que se quiera invertir el orden de los niveles. Basta con envolver el factor con fct\_rev().

Ten en cuenta que si desea revertir sólo una leyenda del gráfico pero no los niveles reales del factor, puede hacerlo con guides() (ver [consejos de ggplot](#ggplot-tips)).

### Por frecuencia

Para ordenar por la frecuencia con que el valor aparece en los datos, utiliza fct\_infreq(). Cualquier valor que falte (NA) se incluirá automáticamente al final, a menos que se convierta en un nivel explícito (véase [esta sección](#fct_missing)). Puedes invertir el orden envolviendo más con fct\_rev().

Esta función puede utilizarse dentro de un ggplot(), como se muestra a continuación.

### Por apariencia

Utiliza fct\_inorder() para establecer el orden de los niveles para que coincida con el orden de aparición en los datos, empezando por la primera fila. Esto puede ser útil si primero organiza cuidadosamente() los datos en el dataframe, y luego utiliza esto para establecer el orden de los factores.

### Por estadística resumida de otra columna

Puedes utilizar fct\_reorder() para ordenar los niveles de una columna por una estadística de resumen de otra columna. Visualmente, esto puede dar lugar a gráficos agradables en los que las barras/puntos ascienden o descienden de forma constante a través del gráfico.

En los ejemplos siguientes, el eje x es delay\_cat, y el eje y es la columna numérica ct\_blood (valor de umbral de ciclo). Los gráficos de caja muestran la distribución del valor CT por grupo delay\_cat. Queremos ordenar los gráficos de caja en orden ascendente por el valor de TC mediano del grupo.

En el primer ejemplo de abajo, se utiliza el orden por defecto de los niveles alfa-numéricos. Se puede ver que las alturas de los gráficos de caja están mezcladas y no en ningún orden particular. En el segundo ejemplo, la columna delay\_cat (asignada al eje x) se ha envuelto en fct\_reorder(), la columna ct\_blood se da como segundo argumento, y la "mediana" se da como tercer argumento (también podría usar "max", "mean", "min", etc). Por lo tanto, el orden de los niveles de delay\_cat reflejará ahora los valores ascendentes de la mediana del CT de cada grupo de delay\_cat. Esto se refleja en el segundo gráfico: los gráficos de caja se han reordenado de forma ascendente. Nótese cómo NA (missing) aparecerá al final, a menos que se convierta en un nivel explícito.

Observe que en este ejemplo no se requieren pasos previos a la llamada a ggplot() - la agrupación y los cálculos se realizan internamente en el comando ggplot.

### Por valor "final"

Utiliza fct\_reorder2() para los gráficos de líneas agrupadas. Ordena los niveles (y, por tanto, la leyenda) para que se alineen con la ordenación vertical de las líneas en el "final" del gráfico. Técnicamente hablando, "ordena por los valores y asociados a los valores x más grandes".

Por ejemplo, si tiene líneas que muestran los recuentos de casos por hospital a lo largo del tiempo, puede aplicar fct\_reorder2() al argumento color = dentro de aes(), de forma que el orden vertical de los hospitales que aparecen en la leyenda se alinee con el orden de las líneas en el extremo terminal del gráfico. Lea más en la [documentación en línea](https://forcats.tidyverse.org/reference/fct_reorder.html).

## Valores faltantes

Si tiene valores NA en su columna de factores, puede convertirlos fácilmente a un nivel con nombre como "Missing" con fct\_explicit\_na(). Los valores NA se convierten por defecto en "(Missing)" al final del orden de los niveles. Puedes ajustar el nombre del nivel con el argumento na\_level =.

A continuación, esta operación se realiza en la columna delay\_cat y se imprime una tabla con tabyl() con NA convertido en "Missing delay".

## Combinar niveles

### Manualmente

Puedes ajustar las visualizaciones de los niveles manualmente con fct\_recode(). Es como la función **dplyr recode**() (véase [Limpieza de datos y funciones básicas](#cleaning-data-and-core-functions)), pero permite la creación de nuevos niveles de factores. Si utiliza la función simple recode() en un factor, los nuevos valores recodificados serán rechazados a menos que ya hayan sido establecidos como niveles permitidos.

Esta herramienta también puede utilizarse para "combinar" niveles, asignando a varios niveles el mismo valor recodificado. Sólo hay que tener cuidado de no perder información. Considere la posibilidad de realizar estos pasos de combinación en una nueva columna (sin sobreescribir la columna existente).

fct\_recode() tiene una sintaxis diferente a la de recode(). recode() utiliza OLD = NEW, mientras que fct\_recode() utiliza NEW = OLD.

Los niveles actuales de delay\_cat son:

Los nuevos niveles se crean utilizando la sintaxis fct\_recode(columna, "nuevo" = "viejo", "nuevo" = "viejo", "nuevo" = "viejo") y se imprimen:

Aquí se combinan manualmente con fct\_recode(). Obsérvese que no se produce ningún error en la creación de un nuevo nivel "Menos de 5 días".

### Reducir a "Otros"

Puedes utilizar fct\_other() para asignar manualmente niveles de factor a un nivel "Otro". A continuación, todos los niveles de la columna hospital, aparte de "Hospital del Puerto" y "Hospital Central", se combinan en "Otros". Puedes proporcionar un vector para mantener =, o eliminar =. Puedes cambiar la visualización del nivel "Otro" con other\_level =.

### Reducir por frecuencia

Puedes combinar los niveles del factor menos frecuente automáticamente utilizando fct\_lump().

Para "agrupar" muchos niveles de baja frecuencia en un grupo "Otros", haga una de las siguientes cosas:

* Establezca n = como el número de grupos que desea conservar. Los n niveles más frecuentes se mantendrán, y todos los demás se combinarán en "Otros".
* Establezca prop = como la proporción de frecuencia del umbral para los niveles por encima de los cuales desea mantener. Todos los demás valores se combinarán en "Otros".

Puedes cambiar la visualización del nivel "Otros" con other\_level =. A continuación, todos los hospitales excepto los dos más frecuentes se combinan en "Otros hospitales".

, warn ## Mostrar todos los niveles

Una de las ventajas del uso de factores es la estandarización del aspecto de las leyendas de los gráficos y de las tablas, independientemente de los valores que estén realmente presentes en unos datos.

Si está preparando muchas figuras (por ejemplo, para varias jurisdicciones), querrá que las leyendas y las tablas aparezcan de forma idéntica incluso con distintos niveles de compleción o composición de los datos.

### En los gráficos

En una figura ggplot(), basta con añadir el argumento drop = FALSE en la función scale\_xxxx() correspondiente. Se mostrarán todos los niveles de los factores, independientemente de si están presentes en los datos. Si sus niveles de columna de factores se muestran con fill =, entonces en scale\_fill\_discrete() incluya drop = FALSE, como se muestra a continuación. Si sus niveles se muestran con x = (al eje x) color = o tamaño =, deberá proporcionar esto a scale\_color\_discrete() o scale\_size\_discrete() según corresponda.

Este ejemplo es un gráfico de barras apiladas de la categoría de edad, por hospital. Añadiendo scale\_fill\_discrete(drop = FALSE) se garantiza que todos los grupos de edad aparezcan en la leyenda, aunque no estén presentes en los datos.

### En tablas

Tanto la **base** R table() como tabyl() de **janitor** mostrarán todos los niveles de los factores (incluso los no utilizados).

Si utiliza count() o summarise() de **dplyr** para hacer una tabla, añada el argumento .drop = FALSE para incluir los recuentos de todos los niveles del factor, incluso los no utilizados.

Lea más en la página de [tablas descriptivas](#descriptive-tables), o en la [documentación de scale\_discrete](https://ggplot2.tidyverse.org/reference/scale_discrete.html), o en la [documentación de count()](https://dplyr.tidyverse.org/reference/count.html). Puedes ver otro ejemplo en la página de rastreo de [contactos](#contact-tracing-1).

## Epiweeks

Por favor, consulte la extensa discusión sobre cómo crear semanas epidemiológicas en la página de [Agrupar datos](#grouping-data).   
Consulte también la página [Trabajar con fechas](#working-with-dates-1) para obtener consejos sobre cómo crear y dar formato a las semanas epidemiológicas.

### Epiweeks en un gráfico

Si tu objetivo es crear epiweeks para mostrarlos en un gráfico, puede hacerlo simplemente con floor\_date() de **lubridate**, como se explica en la página de [Agrupar datos](#grouping-data). Los valores devueltos serán del tipo Date con el formato YYYY-MM-DD. Si utiliza esta columna en un gráfico, las fechas se ordenarán correctamente de forma natural, y no tendrá que preocuparse de los niveles o de la conversión al tipo Factor. Vea el histograma ggplot() de las fechas de inicio más abajo.

En este enfoque, puede ajustar la visualización de las fechas en un eje con scale\_x\_date(). Consulte la página sobre [curvas epidémicas](#epidemic-curves) para obtener más información. Puedes especificar un formato de visualización "strptime" al argumento date\_labels = de scale\_x\_date(). Estos formatos utilizan marcadores de posición "%" y se tratan en la página [Trabajar con fechas](#working-with-dates-1). Utiliza "%Y" para representar un año de 4 dígitos, y "%W" o "%U" para representar el número de la semana (semana del lunes o del domingo respectivamente).

### Epiweeks en los datos

Sin embargo, si tu propósito al factorizar no es trazar, puede enfocar esto de dos maneras:

1. Para un control preciso de la visualización, convierta la columna de la epi-semana **lubrificada** (AAAA-MM-DD) al formato de visualización deseado (AAAA-WWw) dentro del propio dataframe, y luego conviértala en el tipo Factor.

En primer lugar, utiliza format() de R **base** para convertir la visualización de la fecha de YYYY-MM-DD a YYYY-Www (consulte la página [Trabajar con fechas](#working-with-dates-1)). En este proceso el tipo será convertida a carácter. A continuación, convierta de carácter a tipo Factor con factor().

**PELIGRO:** Si coloca las semanas por delante de los años ("Www-YYY") ("%W-%Y"), la ordenación por defecto del nivel alfanumérico será incorrecta (por ejemplo, 01-2015 estará antes que 35-2014). Podría ser necesario ajustar manualmente el orden, lo que sería un proceso largo y doloroso.

1. Para una visualización rápida por defecto, utiliza el paquete **aweek** y su función date2week(). Puedes establecer el week\_start = día, y si establece factor = TRUE entonces la columna de salida es un factor ordenado. Como ventaja, el factor incluye niveles para todas las semanas posibles en el lapso - incluso si no hay casos esa semana.

Consulte la página [Trabajar con fechas](#working-with-dates-1) para obtener más información sobre **aweek**. También ofrece la función inversa week2date().

## Recursos

Página de R for Data Science sobre [factores](https://r4ds.had.co.nz/factors.html)  
[viñeta del paquete aweek](https://cran.r-project.org/web/packages/aweek/vignettes/introduction.html)

# #Pivotar datos

{#pivoting-data}

En la gestión de datos, se puede entender que el pivoteo se refiere a uno de los dos procesos:

1. La creación de tablas dinámicas, que son tablas de estadísticas que resumen los datos de una tabla más extensa
2. La conversión de una tabla de formato **largo** a formato **ancho**, o viceversa.

**En esta página, nos centraremos en la última definición.** La primera es un paso crucial en el análisis de datos, y se trata en las páginas [Agrupar datos](#grouping-data) y [Tablas descriptivas](#descriptive-tables).

En esta página se tratan los formatos de los datos. Es útil conocer la idea de "datos ordenados", en la que cada variable tiene su propia columna, cada observación tiene su propia fila y cada valor tiene su propia celda. Se puede encontrar más información sobre este tema [en este capítulo en línea de R for Data Science](https://r4ds.had.co.nz/tidy-data.html).

## Preparación

### Cargar paquetes

Este trozo de código muestra la carga de los paquetes necesarios para los análisis. En este manual destacamos p\_load() de **pacman**, que instala el paquete si es necesario y lo carga para su uso. También puede cargar los paquetes instalados con library() de R **base.** Consulte la página sobre [los fundamentos de R](#r-basics) para obtener más información sobre los paquetes de R.

### Importar datos

### Datos del recuento de la malaria

En esta página, utilizaremos unos datos ficticio de casos diarios de malaria, por centro y grupo de edad. Si quieres seguirlo, [clica aquí para descargarlo (como archivo .rds)](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/malaria_facility_count_data.rds). Importe los datos con la función import() del paquete **rio** (maneja muchos tipos de archivos como .xlsx, .csv, .rds - vea la página de [importación y exportación](#import-and-export) para más detalles).

A continuación se muestran las primeras 50 filas.

### Datos del caso Linelist

En la parte posterior de esta página, también utilizaremos los datos de casos de una epidemia de ébola simulada. Si quieres seguir el proceso, [clica aqui para descargar el listado "limpio"](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/case_linelists/linelist_cleaned.rds) (como archivo .rds). Importa tus datos con la función import() del paquete **rio** (acepta muchos tipos de archivos como .xlsx, .rds, .csv - vea la página de [importación y exportación](#import-and-export) para más detalles).

## De ancho a largo

### "Formato "ancho

Los datos suelen introducirse y almacenarse en un formato "amplio", en el que las características o respuestas de un sujeto se almacenan en una sola fila. Aunque esto puede ser útil para la presentación, no es ideal para algunos tipos de análisis.

Tomemos como ejemplo el set de datos count\_data importado en la sección Preparación. Puedes ver que cada fila representa un "centro-día". Los recuentos de casos reales (las columnas más a la derecha) se almacenan en un formato "amplio", de modo que la información de cada grupo de edad en un día determinado del centro se almacena en una sola fila.

Cada observación de este conjunto de datos se refiere a los recuentos de paludismo en una de las 65 instalaciones en una fecha determinada, que va desde count\_data$data\_date %>% min() hasta count\_data$data\_date %>% max(). Estas instalaciones están situadas en una provincia (Norte) y cuatro distritos (Spring, Bolo, Dingo y Barnard). El set de datos proporciona los recuentos globales de malaria, así como los recuentos específicos por edad en cada uno de los tres grupos de edad: <4 años, 5-14 años y 15 años o más.

Los datos "amplios" como éste no se ajustan a las normas de "datos ordenados", porque los encabezados de las columnas no representan realmente "variables", sino que representan valores de una hipotética variable "grupo de edad".

Este formato puede ser útil para presentar la información en una tabla, o para introducir datos (por ejemplo, en Excel) a partir de formularios de informes de casos. Sin embargo, en la etapa de análisis, estos datos normalmente deben ser transformados a un formato "más largo" más alineado con los estándares de "datos ordenados". El paquete R **ggplot2,** en particular, funciona mejor cuando los datos están en un formato "largo".

La visualización de los recuentos totales de malaria a lo largo del tiempo no plantea ninguna dificultad con los datos en su formato actual:

Sin embargo, ¿qué pasaría si quisiéramos mostrar las contribuciones relativas de cada grupo de edad a este recuento total? En este caso, necesitamos asegurarnos de que la variable de interés (grupo de edad), aparezca en set de datos en una sola columna que pueda pasarse al argumento aes() de {ggplot2} "mapping aesthetics".

### pivot\_longer()

La función **tidyr** pivot\_longer() hace que los datos sean "más largos". **tidyr** forma parte del **tidyverse** de los paquetes de R.

Acepta un rango de columnas para transformar (especificado a cols =). Por lo tanto, puede operar sólo en una parte de unos datos. Esto es útil para los datos de la malaria, ya que sólo queremos pivotar las columnas de recuento de casos.

En este proceso, terminará con dos "nuevas" columnas - una con las categorías (los antiguos nombres de las columnas), y otra con los valores correspondientes (por ejemplo, recuento de casos). Puedes aceptar los nombres por defecto para estas nuevas columnas, o puede especificar los suyos propios a names\_to = y values\_to = respectivamente.

Veamos pivot\_longer() en acción...

### Pivote estándar

Queremos utilizar la función pivot\_longer() de **tidyr** para convertir los datos "anchos" en un formato "largo". Concretamente, para convertir las cuatro columnas numéricas con datos sobre los recuentos de malaria en dos nuevas columnas: una que contenga los grupos de edad y otra que contenga los valores correspondientes.

Observe que el dataframe recién creado (df\_long) tiene más filas (12.152 frente a 3.038); se ha hecho más largo. De hecho, es precisamente cuatro veces más largo, porque cada fila de los datos originales representa ahora cuatro filas en df\_long, una para cada una de las observaciones de recuento de malaria (<4 años, 5-14 años, 15 años+ y total).

Además de ser más largo, el nuevo conjunto de datos tiene menos columnas (8 frente a 10), ya que los datos que antes se almacenaban en cuatro columnas (las que empiezan por el prefijo malaria\_) se almacenan ahora en dos.

Dado que los nombres de estas cuatro columnas comienzan con el prefijo malaria\_, podríamos haber hecho uso de la práctica función "tidyselect" starts\_with() para conseguir el mismo resultado (véase la página [Limpieza de datos y funciones básicas](#cleaning-data-and-core-functions) para conocer más sobre estas funciones de ayuda).

o por posición:

o por rango de nombres:

Estas dos nuevas columnas reciben los nombres por defecto de nombre y valor, pero podemos anular estos valores por defecto para proporcionar nombres más significativos, que pueden ayudar a recordar lo que se almacena dentro, utilizando los argumentos names\_to y values\_to. Utilicemos los nombres age\_group y counts:

Ahora podemos pasar este nuevo conjunto de datos a {ggplot2}, y asignar la nueva columna recuento al eje y y la nueva columna grupo\_de\_edad al argumento fill = (el color interno de la columna). Esto mostrará los recuentos de malaria en un gráfico de barras apilado, por grupo de edad:

Examina esta nueva trama y compárala con la que hemos creado antes: ¿qué ha fallado?

Nos hemos encontrado con un problema común al manejar los datos de vigilancia: hemos incluido también los recuentos totales de la columna malaria\_tot, por lo que la magnitud de cada barra en el gráfico es el doble de lo que debería ser.

Podemos manejar esto de varias maneras. Podríamos simplemente filtrar estos totales dset de datos antes de pasarlo a ggplot():

Como alternativa, podríamos haber excluido esta variable al ejecutar pivot\_longer(), manteniéndola así en set de datos como una variable independiente. Observa cómo se "expanden" sus valores para llenar las nuevas filas.

### Pivotar datos de múltiples tipos

El ejemplo anterior funciona bien en situaciones en las que todas las columnas que se quieren "pivotar más" son del mismo tipo (carácter, numérico, lógico...).

Sin embargo, habrá muchos casos en los que, en el trabajo de campo, se trabaje con datos preparados por personas no especializadas y que sigan su propia lógica no estándar - como señaló Hadley Wickham (haciendo referencia a Tolstoi) en su [artículo seminal](https://vita.had.co.nz/papers/tidy-data.pdf) sobre los principios de **Tidy Data**: "Como las familias, los conjuntos de datos ordenados son todos iguales, pero cada conjunto de datos desordenado es desordenado a su manera".

Un problema particularmente común que encontrará será la necesidad de pivotar columnas que contienen diferentes tipos de datos. Este pivote resultará en el almacenamiento de estos diferentes tipos de datos en una sola columna, lo cual no es una buena situación. Hay varios enfoques que uno puede tomar para separar el desorden que esto crea, pero hay un paso importante que puedes seguir usando pivot\_longer() para evitar crear tal situación tu mismo.

Tomemos una situación en la que ha habido una serie de observaciones en diferentes pasos de tiempo para cada uno de los tres elementos A, B y C. Ejemplos de estos elementos podrían ser individuos (por ejemplo, contactos de un caso de ébola que se rastrean cada día durante 21 días) o puestos de salud de aldeas remotas que se supervisan una vez al año para garantizar que siguen funcionando. Utilicemos el ejemplo del rastreo de contactos. Imaginemos que los datos se almacenan de la siguiente manera:

Como puede verse, los datos son un poco complicados. Cada fila almacena información sobre un elemento, pero con la serie temporal cada vez más alejada hacia la derecha a medida que avanza el tiempo. Además, los tipos de columnas alternan entre valores de fecha y caracteres.

Un ejemplo particularmente malo que encontró este autor fue el de los datos de vigilancia del cólera, en el que se añadieron 8 nuevas columnas de observaciones cada día en el transcurso de **4 años**. El simple hecho de abrir el archivo de Excel en el que se almacenaban estos datos me llevó más de 10 minutos en mi ordenador portátil.

Para trabajar con estos datos, necesitamos transformar el dataframe a formato largo, pero manteniendo la separación entre una columna de fecha y una columna de caracteres (estado), para cada observación de cada elemento. Si no lo hacemos, podríamos terminar con una mezcla de tipos de variables en una sola columna (un gran "no-no" cuando se trata de gestión de datos y de datos ordenados):

Arriba, nuestro pivote ha fusionado fechas y caracteres en una sola columna de valor. R reaccionará convirtiendo toda la columna en tipo carácter, y se pierde la utilidad de las fechas.

Para evitar esta situación, podemos aprovechar la estructura sintáctica de los nombres de las columnas originales. Hay una estructura de nombres común, con el número de observación, un guión bajo, y luego "estado" o "fecha". Podemos aprovechar esta sintaxis para mantener estos dos tipos de datos en columnas separadas después del pivote.

Para ello:

* Proporcionar un vector de caracteres al argumento names\_to =, siendo el segundo elemento (".value" ). Este término especial indica que las columnas pivotadas se dividirán basándose en un carácter de su nombre...
* También debe proporcionar el carácter de "división" al argumento names\_sep =. En este caso, es el guión bajo "\_".

Así, la denominación y división de las nuevas columnas se basa en el guión bajo de los nombres de las variables existentes.

**Toques finales**:

Ten en cuenta que la columna de fecha está actualmente en el tipo carácter - podemos convertirla fácilmente en tipo fecha utilizando las funciones mutate() y as\_date() descritas en la página [Trabajar con fechas](#working-with-dates-1).

También podemos querer convertir la columna de observación a un formato numérico eliminando el prefijo "obs" y convirtiendo a numérico. Podemos hacer esto con str\_remove\_all() del paquete **stringr** (véase la página [Caracteres y cadenas](#characters-and-strings)).

Y ahora, podemos empezar a trabajar con los datos en este formato, por ejemplo, trazando un mosaico de calor descriptivo:

## De largo a ancho

En algunos casos, es posible que queramos convertir unos datos a un formato más amplio. Para ello, podemos utilizar la función pivot\_wider().

Un caso de uso típico es cuando queremos transformar los resultados de un análisis en un formato que sea más digerible para el lector (como una [tabla para su presentación](#tables-for-presentation)). Por lo general, se trata de transformar unos datos en el que la información de un sujeto está repartida en varias filas en un formato en el que esa información se almacena en una sola fila.

### Datos

Para esta sección de la página, utilizaremos la lista de casos (véase la sección [Preparación](#pivot_prep)), que contiene una fila por caso.

Aquí están las primeras 50 filas:

Supongamos que queremos conocer los recuentos de individuos en los diferentes grupos de edad, por género:

Esto nos da un largo conjunto de datos que es genial para producir visualizaciones en **ggplot2**, pero no es ideal para la presentación en una tabla:

### Pivote más amplio

Por lo tanto, podemos utilizar pivot\_wider() para transformar los datos en un formato mejor para incluirlos como tablas en nuestros informes.

El argumento nombres\_desde especifica la columna de la que generar los nuevos nombres de columna, mientras que el argumento valores\_desde especifica la columna de la que tomar los valores para rellenar las celdas. El argumento id\_cols = es opcional, pero se puede proporcionar un vector de nombres de columnas que no deben ser pivotadas, y que por tanto identificarán cada fila.

Esta tabla es mucho más fácil de leer y, por tanto, mejor para incluirla en nuestros informes. Se puede convertir en una tabla bonita con varios paquetes, como **flextable** y **knitr**. Este proceso se elabora en la página [Tablas para la presentación](#tables-for-presentation).

## Llenar

En algunas situaciones después de un pivote, y más comúnmente después de un bind, nos quedan huecos en algunas celdas que nos gustaría rellenar.

### Datos

Por ejemplo, tome dos conjuntos de datos, cada uno con observaciones para el número de medición, el nombre del centro y el recuento de casos en ese momento. Sin embargo, el segundo conjunto de datos también tiene la variable Año.

Cuando realizamos un bind\_rows() para unir los dos conjuntos de datos, la variable Year se rellena con NA para aquellas filas en las que no había información previa (es decir, el primer conjunto de datos):

### llenar()

En este caso, Año es una variable útil para incluir, especialmente si queremos explorar las tendencias a lo largo del tiempo. Por lo tanto, utilizamos fill() para rellenar esas celdas vacías, especificando la columna a rellenar y la dirección (en este caso **hacia arriba**):

Alternativamente, podemos reordenar los datos para que tengamos que rellenar en sentido descendente:

Ahora tenemos unos datos útil para trazar:

Pero es menos útil para presentarlo en una tabla, así que practiquemos la conversión de este largo y desordenado dataframe en un dataframe más amplio y ordenado:

N.B. En este caso, tuvimos que especificar que sólo se incluyeran las tres variables Instalación, Año y Casos, ya que la variable adicional Medida interferiría en la creación de la tabla:

## Recursos

Aquí hay un [tutorial](https://datacarpentry.org/r-socialsci/03-dplyr-tidyr/index.html) útil

# #Agrupar datos

{#grouping-data}

Esta página cubre cómo agrupar y agregar datos para el análisis descriptivo. Hace uso de la familia de paquetes **tidyverse** para funciones comunes y fáciles de usar.

La agrupación de datos es un componente esencial de la gestión y el análisis de datos. Los datos agrupados se resumen estadísticamente por grupos y pueden representarse gráficamente por grupos. Las funciones del paquete **dplyr** (parte del **tidyverse**) facilitan la agrupación y las operaciones posteriores.

En esta página se tratarán los siguientes temas:

* Agrupar datos con la función group\_by()
* Datos no agrupados
* resumir() datos agrupados con estadísticas
* La diferencia entre count() y tally()
* arrange() aplicado a los datos agrupados
* filter() aplicado a los datos agrupados
* mutate() aplicada a los datos agrupados
* select() aplicado a datos agrupados
* El comando **básico** R aggregate() como alternativa

## Preparación

### Cargar paquetes

Este trozo de código muestra la carga de los paquetes necesarios para los análisis. En este manual destacamos p\_load() de **pacman**, que instala el paquete si es necesario y lo carga para su uso. También puede cargar los paquetes instalados con library() de R **base.** Consulte la página sobre [los fundamentos de R](#r-basics) para obtener más información sobre los paquetes de R.

### Importar datos

Importamos los datos de casos de una epidemia de ébola simulada. Si quieres seguirlo, [clica para descargar linelist "limpia"](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/case_linelists/linelist_cleaned.rds) (como archivo .rds). Los datos se importan mediante la función import() del paquete **rio**. Consulte la página sobre [importación y exportación](#import-and-export) para conocer las distintas formas de importar datos.

Las primeras 50 filas del listado:

## Agrupación

La función group\_by() de **dplyr** agrupa las filas por los valores únicos de la columna que se le especifica. Si se especifican varias columnas, las filas se agrupan por las combinaciones únicas de valores entre las columnas. Cada valor único (o combinación de valores) constituye un grupo. Los cambios posteriores en set de datos o los cálculos pueden realizarse en el contexto de cada grupo.

Por ejemplo, el siguiente comando toma linelist y agrupa las filas por valores únicos en la columna resultado, guardando la salida como un nuevo dataframe ll\_por\_resultado. La(s) columna(s) de agrupación se colocan dentro de los paréntesis de la función group\_by().

**Ten en cuenta que no hay ningún cambio perceptible en los datos** después de ejecutar group\_by(), hasta que se aplique otro verbo **de dplyr como** mutate(), summarise(), o arrange() en el dataframe "agrupado".

Sin embargo, puede "ver" las agrupaciones imprimiendo el dataframe. Al imprimir un dataframe agrupado, verá que se ha transformado en un [objeto de clasetibble](https://tibble.tidyverse.org/) que, al imprimirse, muestra qué agrupaciones se han aplicado y cuántos grupos hay, escrito justo encima de la fila de cabecera.

### Grupos únicos

**Los grupos creados reflejan cada combinación única de valores en las columnas de agrupación.**

Para ver los grupos y el número de filas en cada grupo, pase los datos agrupados a tally(). Para ver sólo los grupos únicos sin recuento puede pasar a group\_keys().

Vea a continuación que hay **tres** valores únicos en el resultado de la columna de agrupación: "Muerte", "Recuperación" y NA. Véase que hubo nrow(linelist %>% filter(outcome == "Death")) muertes, nrow(linelist %>% filter(outcome == "Recover")) recuperaciones, y nrow(linelist %>% filter(is.na(outcome)) sin resultado registrado.

Se puede agrupar por más de una columna. A continuación, el dataframe se agrupa por resultado y género, y luego se cuenta. Observe cómo cada combinación única de resultado y género se registra como su propio grupo, incluyendo los valores faltantes para cualquier columna.

### Nuevas columnas

También puede crear una nueva columna de agrupación dentro de la sentencia group\_by(). Esto equivale a llamar a mutate() antes de group\_by(). Para una tabulación rápida este estilo puede ser útil, pero para una mayor claridad en su código considere crear esta columna en su propio paso mutate() y luego canalizarla a group\_by().

### Añadir/soltar columnas de agrupación

Por defecto, si ejecuta group\_by() sobre datos que ya están agrupados, se eliminarán los grupos antiguos y se aplicarán los nuevos. Si desea añadir nuevos grupos a los existentes, incluya el argumento .add = TRUE.

\*\* Mantener todos los grupos\*\*

Si se agrupa en una columna de de tipo factor, puede haber niveles del factor que no estén presentes en los datos. Si agrupa en esta columna, por defecto esos niveles no presentes se descartan y no se incluyen como grupos. Para cambiar esto de manera que todos los niveles aparezcan como grupos (incluso si no están presentes en los datos), establezca .drop = FALSE en su comando group\_by().

## Desagregue

Los datos que han sido agrupados permanecerán agrupados hasta que sean específicamente desagrupados mediante ungroup(). Si se olvida de desagrupar, puede dar lugar a cálculos incorrectos. A continuación se muestra un ejemplo de eliminación de todas las agrupaciones:

También puede eliminar la agrupación sólo para columnas específicas, colocando el nombre de la columna dentro de ungroup().

**NOTA:** El verbo count() desagrupa automáticamente los datos después del recuento.

## Resumir

Véase la sección **dplyr** de la página [Tablas descriptivas](#descriptive-tables) para una descripción detallada de cómo producir tablas de resumen con summarise(). Aquí abordamos brevemente cómo cambia su comportamiento cuando se aplica a datos agrupados.

La función **dplyr** summarise() (o summarize()) toma un dataframe y lo convierte en un nuevo dataframe de resumen, con columnas que contienen los estadísticos de resumen que definas. En un dataframe sin agrupar, las estadísticas de resumen se calcularán a partir de todas las filas. La aplicación de summarise() a los datos agrupados produce esas estadísticas de resumen para cada grupo.

La sintaxis de summarise() es tal que se proporciona el nombre de la(s) **nueva(s)** columna(s) de resumen, un signo de igualdad y, a continuación, una función estadística para aplicar a los datos, como se muestra a continuación. Por ejemplo, min(), max(), median() o sd(). Dentro de la función estadística, indique la columna con la que se va a operar y cualquier argumento relevante (por ejemplo, na.rm = TRUE). Puedes utilizar sum() para contar el número de filas que cumplen un criterio lógico (con doble igual ==).

A continuación se muestra un ejemplo de summarise() aplicado sin datos agrupados. Las estadísticas devueltas se producen a partir del set de datos completo.

Por el contrario, a continuación se muestra la misma sentencia summarise() aplicada a los datos agrupados. Las estadísticas se calculan para cada grupo de resultados. Observe cómo las columnas de agrupación se trasladan al nuevo dataframe.

**SUGERENCIA:** La función summarise funciona tanto con la ortografía del Reino Unido como con la de EE.UU. - summarise() y summarize() llaman a la misma función.

## Recuentos y cuentas

count() y tally() proporcionan una funcionalidad similar pero son diferentes. Lea más sobre la distinción entre tally() y count() [aquí](https://dplyr.tidyverse.org/reference/tally.html)

### tally()

tally() es la abreviatura de summarise(n = n()), y no agrupa los datos. Por lo tanto, para lograr recuentos agrupados debe seguir un comando group\_by(). Puedes añadir sort = TRUE para ver primero los grupos más grandes.

### contar()

En cambio, count() hace lo siguiente:

1. aplica group\_by() a la(s) columna(s) especificada(s)
2. aplica summarise() y devuelve la columna n con el número de filas por grupo
3. aplica ungroup()

Al igual que con group\_by() puede crear una nueva columna dentro del comando count():

count() puede llamarse varias veces, con la funcionalidad "enrollada". Por ejemplo, para resumir el número de hospitales presentes para cada género, ejecuta lo siguiente. Ten en cuenta que el nombre de la columna final se ha cambiado de "n" por defecto para mayor claridad (con nombre =).

### Añadir recuentos

A diferencia de count() y summarise(), puede utilizar add\_count() para añadir una nueva columna n con los recuentos de filas por grupo, conservando todas las demás columnas del dataframe.

Esto significa que el número de recuento de un grupo, en la nueva columna n, se imprimirá en cada fila del grupo. Para fines de demostración, añadimos esta columna y luego reordenamos las columnas para facilitar la visualización. Consulte la sección siguiente sobre el [filtro del tamaño del grupo](#group_filter_grp_size) para ver otro ejemplo.

### Añadir los totales

Para añadir fácilmente filas o columnas de suma total después de utilizar tally() o count(), consulte la sección del **conserje de** la página [Tablas descriptivas](#tbl_janitor). Este paquete ofrece funciones como adorn\_totals() y adorn\_percentages() para añadir totales y convertirlos para mostrar porcentajes. A continuación se muestra un breve ejemplo:

Para añadir filas de totales más complejas que incluyan estadísticas de resumen distintas de las sumas, consulte [esta sección de la página Tablas descriptivas](#tbl_dplyr_totals).

## Agrupación por fecha

Al agrupar datos por fecha, debe tener (o crear) una columna para la unidad de fecha de interés - por ejemplo "día", "epiweek", "mes", etc. Puedes crear esta columna utilizando floor\_date() de **lubridate**, como se explica en la [sección Semanas epidémicas de](#dates_epi_wks) la página [Trabajar con fechas](#working-with-dates-1). Una vez que tenga esta columna, puede utilizar count() de **dplyr** para agrupar las filas por esos valores de fecha únicos y lograr recuentos agregados.

Un paso adicional común para las situaciones de fechas, es "rellenar" cualquier fecha en la secuencia que no esté presente en los datos. Utiliza complete() de **tidyr** para que la serie de fechas agregadas esté completa, incluyendo todas las unidades de fecha posibles dentro del rango. Sin este paso, una semana sin casos reportados podría no aparecer en tus datos.

Dentro de complete() redefine su columna de fecha como una secuencia de fechas seq.Date() desde el mínimo hasta el máximo - así las fechas se expanden. Por defecto, los valores del recuento de casos en cualquier nueva fila "expandida" serán NA. Puedes establecerlos a 0 utilizando el argumento fill = de complete(), que espera una lista con nombre (si su columna de recuentos se llama n, proporcione fill = list(n = 0). Consulte ?complete para obtener más detalles y la página [Trabajar con fechas](#dates_epi_wks) para ver un ejemplo.

### Lista de casos en días

Aquí hay un ejemplo de agrupación de casos en días sin usar complete(). Obsérvese que las primeras filas omiten las fechas sin casos.

A continuación añadimos el comando complete() para asegurarnos de que todos los días del rango están representados.

### Lista de casos en semanas

Se puede aplicar el mismo principio para las semanas. Primero cree una nueva columna que sea la semana del caso utilizando floor\_date() con unidad = "semana". A continuación, utiliza count() como en el caso anterior para obtener los recuentos de casos semanales. Termine con complete() para asegurarse de que todas las semanas están representadas, incluso si no contienen casos.

Aquí están las primeras 50 filas del dataframe resultante:

### Lista de casos en meses

Para agregar casos en meses, vuelva a utilizar floor\_date() del paquete **lubridate**, pero con el argumento unidad = "meses". Esto redondea cada fecha hacia abajo al día 1 de su mes. La salida será el tipo Date. Ten en cuenta que en el paso complete() también utilizamos by = "months".

### Recuentos diarios en semanas

Para agregar los recuentos diarios en recuentos semanales, utiliza floor\_date() como arriba. Sin embargo, utiliza group\_by() y summarize() en lugar de count() porque necesita sumar() los recuentos de casos diarios en lugar de limitarse a contar el número de filas por semana.

#### Recuentos diarios en meses

Para agregar los recuentos diarios en recuentos de meses, utiliza floor\_date() con unidad = "mes" como en el caso anterior. Sin embargo, utiliza group\_by() y summarize() en lugar de count() porque necesita sumar() los recuentos de casos diarios en lugar de limitarse a contar el número de filas por mes.

## Ordenar los datos agrupados

El uso del verbo **dplyr** arrange() para ordenar las filas de un dataframe se comporta igual cuando los datos están agrupados, a menos que se establezca el argumento .by\_group =TRUE. En este caso, las filas se ordenan primero por las columnas de agrupación y luego por cualquier otra columna que se especifique en arrange().

## Filtro de datos agrupados

### filtrar()

Cuando se aplica junto con funciones que evalúan el dataframe (como max(), min(), mean()), estas funciones se aplicarán ahora a los grupos. Por ejemplo, si desea filtrar y mantener las filas en las que los pacientes están por encima de la edad media, esto se aplicará ahora por grupo, filtrando para mantener las filas por encima de la edad media del grupo.

### Filas de corte por grupo

La función **dplyr** slice(), que [filtra las filas según su posición](https://dplyr.tidyverse.org/reference/slice.html) en los datos, también puede aplicarse por grupo. Recuerde que debe tener en cuenta la ordenación de los datos dentro de cada grupo para obtener la "rebanada" deseada.

Por ejemplo, para recuperar sólo los últimos 5 ingresos de cada hospital:

1. Agrupar linelist por columna hospitalaria
2. Ordenar los registros de la fecha de hospitalización más reciente a la más antigua dentro de cada grupo de hospitales
3. Rebanada para recuperar las 5 primeras filas de cada hospital

slice\_head() - selecciona n filas de la parte superior  
slice\_tail() - selecciona n filas del final  
slice\_sample() - selecciona aleatoriamente n filas  
slice\_min() - selecciona n filas con los valores más altos en order\_by = columna, usa with\_ties = TRUE para mantener los empates  
slice\_max() - selecciona n filas con los valores más bajos en order\_by = columna, utiliza with\_ties = TRUE para mantener los empates

Consulte la página de [desduplicación](#de-duplication) para ver más ejemplos y detalles sobre slice().

### Filtro por tamaño de grupo

La función add\_count() añade una columna n a los datos originales dando el número de filas en el grupo de esa fila.

A continuación, add\_count() se aplica a la columna hospital, por lo que los valores de la nueva columna n reflejan el número de filas del grupo de hospitales de esa fila. Observe cómo se repiten los valores de la columna n. En el ejemplo siguiente, el nombre de la columna n podría cambiarse utilizando name = dentro de add\_count(). Para fines de demostración reordenamos las columnas con select().

De este modo, resulta fácil filtrar los casos que fueron hospitalizados en un hospital "pequeño", por ejemplo, un hospital que admitió a menos de 500 pacientes:

## Mutar en datos agrupados

Para conservar todas las columnas y filas (no resumir) y añadir una nueva columna que contenga estadísticas de grupo, utiliza mutate() después de group\_by() en lugar de summarise().

Esto es útil si se desea obtener estadísticas de grupo en los datos originales con todas las demás columnas presentes, por ejemplo, para los cálculos que comparan una fila con su grupo.

Por ejemplo, este código calcula la diferencia entre la demora en el ingreso de una fila y la demora media de su hospital. Los pasos son:

1. Agrupar los datos por hospital
2. Utiliza la columna days\_onset\_hosp (retraso hasta la hospitalización) para crear una nueva columna que contenga el retraso medio en el hospital de esa fila
3. Calcular la diferencia entre las dos columnas

Seleccionamos() sólo ciertas columnas para mostrarlas, con fines de demostración.

## Seleccionar en datos agrupados

El verbo select() funciona con datos agrupados, pero las columnas de agrupación siempre se incluyen (aunque no se mencionen en select()). Si no desea estas columnas de agrupación, utiliza primero ungroup().

## Recursos

A continuación, algunos recursos útiles para obtener más información:

Puedes realizar cualquier función de resumen sobre datos agrupados; consulte la [hoja de trucos de transformación de datos de RStudio](https://github.com/rstudio/cheatsheets/blob/master/data-transformation.pdf)

La página de Data Carpentry sobre [**dplyrLas**](https://datacarpentry.org/R-genomics/04-dplyr.html) páginas de referencia de **tidyverse** sobre [group\_by()](https://dplyr.tidyverse.org/reference/group_by.html) y [agrupación](https://dplyr.tidyverse.org/articles/grouping.html)

Esta página sobre [Manipulación de datos](https://itsalocke.com/files/DataManipulationinR.pdf)

[Resumir con condiciones en dplyr](https://stackoverflow.com/questions/23528862/summarize-with-conditions-in-dplyr)

# #Unir datos

{#joining-data}

Arriba: un ejemplo animado de una unión por la izquierda ([*fuente de la imagen*](https://github.com/gadenbuie/tidyexplain/tree/master/images))

En esta página se describen las formas de "unir", "emparejar", "enlazar" y combinar de otro modo los dataframes.

Es raro que su análisis epidemiológico o flujo de trabajo no implique múltiples fuentes de datos y la vinculación de múltiples conjuntos de datos. Tal vez necesite conectar los datos de laboratorio con los resultados clínicos de los pacientes, o los datos de movilidad de Google con las tendencias de las enfermedades infecciosas, o incluso unos datos en una fase del análisis con una versión transformada de sí mismo.

En esta página demostramos el código para:

* Realiza uniones de dos dataframes de forma que las filas coincidan en función de los valores comunes de las columnas de los identificadores
* Unir dos dataframes basándose en coincidencias probabilísticas (probables) entre los valores
* Ampliar un dataframe vinculando directamente o ("añadiendo") filas o columnas de otro dataframe

## Preparación

### Cargar paquetes

Este trozo de código muestra la carga de los paquetes necesarios para los análisis. En este manual destacamos p\_load() de **pacman**, que instala el paquete si es necesario y lo carga para su uso. También puede cargar los paquetes instalados con library() de R **base.** Consulte la página sobre [los fundamentos de R](#r-basics) para obtener más información sobre los paquetes de R.

### Importar datos

Para empezar, importamos la lista de casos limpia de una epidemia de ébola simulada. Si quieres seguir el proceso, [clica aquí para descargar el listado “limpio"](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/case_linelists/linelist_cleaned.rds) (como archivo .rds). Import los datos con la función import() del paquete **rio** (maneja muchos tipos de archivos como .xlsx, .csv, .rds - mira la página de [importación y exportación](#import-and-export) para más detalles).

A continuación se muestran las primeras 50 filas del listado.

### Ejemplos de conjuntos de datos

En la sección de unión que sigue, utilizaremos los siguientes conjuntos de datos:

1. Una versión "en miniatura" de la lista de casos, que contiene sólo las columnas case\_id, date\_onset y hospital, y sólo las 10 primeras filas
2. Un dataframe separado llamado hosp\_info, que contiene más detalles sobre cada hospital

En la sección sobre el emparejamiento probabilístico, utilizaremos dos pequeños conjuntos de datos diferentes. El código para crear esos conjuntos de datos se da en esa sección.

#### Lista de casos "miniatura"

A continuación se muestra la lista de casos en miniatura, que contiene sólo 10 filas y sólo las columnas case\_id, date\_onset y hospital.

#### dataframe de información hospitalaria

A continuación se muestra el código para crear un dataframe separado con información adicional sobre siete hospitales (la población de captación y el nivel de atención disponible). Obsérvese que el nombre "Hospital Militar" pertenece a dos hospitales diferentes: uno de nivel primario que atiende a 10000 residentes y otro de nivel secundario que atiende a 50280 residentes.

Aquí está este dataframe:

### Pre-limpieza

Las uniones tradicionales (no probabilísticas) distinguen entre mayúsculas y minúsculas y requieren coincidencias de caracteres exactas entre los valores de los dos dataframes. Para demostrar algunos de los pasos de limpieza que puede necesitar antes de iniciar una unión, ahora limpiaremos y alinearemos los conjuntos de datos linelist\_mini y hosp\_info.

**Identificar las diferencias**

Necesitamos que los valores de la columna hosp\_name en el dataframe hosp\_info coincidan con los valores de la columna hospital en el dataframe linelist\_mini.

Aquí están los valores del dataframe linelist\_mini, impresos con la función **base de** R unique():

y aquí están los valores del dataframe hosp\_info:

Puedes ver que, aunque algunos de los hospitales existen en ambos dataframes, hay muchas diferencias en la ortografía.

**Alinear los valores**

Comenzamos limpiando los valores del dataframe hosp\_info. Como se explica en la página [Limpieza de datos y funciones básicas](#cleaning-data-and-core-functions), podemos recodificar los valores con criterios lógicos utilizando la función case\_when() de **dplyr**. Para los cuatro hospitales que existen en ambos dataframes, cambiamos los valores para alinearlos con los valores de linelist\_mini. Para los demás hospitales dejamos los valores como están (TRUE ~ nombre\_hospital).

**ATENCIÓN:** Normalmente, al limpiar se debe crear una nueva columna (por ejemplo, nombre\_del\_hospital\_limpio), pero para facilitar la demostración mostramos la modificación de la antigua columna

Los nombres de los hospitales que aparecen en ambos dataframes están alineados. Hay dos hospitales en hosp\_info que no están presentes en linelist\_mini - nos ocuparemos de ellos más adelante, en la unión.

Antes de una unión, a menudo es más fácil convertir una columna a todas las minúsculas o a todas las mayúsculas. Si necesita convertir todos los valores de una columna a MAYÚSCULAS o minúsculas, utiliza mutate() y envuelva la columna con una de estas funciones de **stringr**, como se muestra en la página sobre [Caracteres y cadenas](#characters-and-strings).

str\_to\_upper()  
str\_to\_upper()  
str\_to\_title()

## ****dplyr**** se une a

El paquete **dplyr** ofrece varias funciones de unión. **dplyr** está incluido en el paquete **tidyverse**. Estas funciones de unión se describen a continuación, con casos de uso sencillos.

Muchas gracias a [https://github.com/gadenbuie](https://github.com/gadenbuie/tidyexplain/tree/master/images) por los gifs informativos.

### Sintaxis general

Los comandos de unión pueden ejecutarse como comandos independientes para unir dos dataframes en un nuevo objeto, o pueden utilizarse dentro de una cadena de tuberías (%>%) para fusionar un dataframe en otro mientras se limpia o se modifica de alguna manera.

En el siguiente ejemplo, la función left\_join() se utiliza como un comando independiente para crear un nuevo dataframe joined\_data. Las entradas son los dataframes 1 y 2 (df1 y df2). El primer dataframe listado es el dataframe de referencia, y el segundo listado se une a él.

El tercer argumento por = es donde se especifican las columnas de cada dataframe que se utilizarán para alinear las filas de los dos dataframes. Si los nombres de estas columnas son diferentes, proporciónelos dentro de un vector c() como se muestra a continuación, donde las filas se emparejan sobre la base de valores comunes entre el identificador de la columna en df1 y el identificador de la columna en df2.

Si las columnas by de ambos dataframes tienen exactamente el mismo nombre, puede proporcionar sólo este nombre, entre comillas.

Si está uniendo los dataframes basándose en valores comunes en varios campos, enumere estos campos dentro del vector c(). Este ejemplo une filas si los valores de tres columnas de cada conjunto de datos se alinean exactamente.

Los comandos de unión también pueden ejecutarse dentro de una cadena de tuberías. Esto modificará el dataframe que se está canalizando.

En el ejemplo siguiente, df1 se pasa por las tuberías, df2 se une a él y, por tanto, df se modifica y se redefine.

**ATENCIÓN: ¡Las** uniones son específicas para cada caso! Por lo tanto, es útil convertir todos los valores a minúsculas o mayúsculas antes de la unión. Consulte la página sobre caracteres/cadenas.

### Uniones izquierda y derecha

**Una unión a la izquierda o a la derecha se utiliza habitualmente para añadir información a un dataframe**: la nueva información se añade sólo a las filas que ya existían en el dataframe de referencia. Estas uniones son comunes en el trabajo epidemiológico, ya que se utilizan para añadir información de unos datos a otro.

Al utilizar estas uniones, el orden de escritura de los dataframes en el comando es importante\*.

* En una unión a la izquierda, el primer dataframe escrito es la línea de base
* En una unión a la derecha, el segundo dataframe escrito es la línea de base

**Se conservan todas las filas del dataframe de referencia.** La información del otro dataframe (secundario) se une al dataframe de referencia sólo si hay una coincidencia a través de la(s) columna(s) del identificador. Además:

* Las filas del dataframe secundario que no coinciden se eliminan.
* Si hay muchas filas de la línea de base que coinciden con una fila del dataframe secundarios (muchos a uno), la información secundaria se añade a cada fila de la línea de base que coincide.
* Si una fila de la línea de base coincide con varias filas del dataframe secundario (uno a varios), se dan todas las combinaciones, lo que significa que se pueden añadir nuevas filas al dataframe devuelto.

Ejemplos animados de uniones a la izquierda y a la derecha ([fuente de la imagen](https://github.com/gadenbuie/tidyexplain/tree/master/images))

**Ejemplo**

A continuación se muestra el resultado de un left\_join() de hosp\_info (dataframe secundario, ver [aquí](#joins_hosp_info)) en linelist\_mini (dataframe de referencia, [ver aquí](#joins_llmini)). La linelist\_mini original tiene nrow(linelist\_mini) filas. Se muestra la linelist\_mini modificada. Observe lo siguiente:

* Se han añadido dos nuevas columnas, catchment\_pop y level en la parte izquierda de linelist\_mini
* Se mantienen todas las filas originales del dataframe de referencia linelist\_mini
* Cualquier fila original de linelist\_mini para "Hospital Militar" está duplicada porque coincide con dos filas en el dataframe secundario, por lo que se devuelven ambas combinaciones
* La columna del identificador de la unión del set de datos secundario (hosp\_name) ha desaparecido porque es redundante con la columna del identificador primario (hospital)
* Cuando una fila de referencia no coincide con ninguna fila secundaria (por ejemplo, cuando el hospital es "Otro" o "Falta"), NA (en blanco) rellena las columnas del dataframe secundario
* Se eliminaron las filas del dataframe secundario que no coincidían con el dataframe de referencia (hospitales "hermanas" e "ignace")

#### "¿Debo usar una unión a la derecha o a la izquierda?"

Para responder a la pregunta anterior, pregúntese "¿qué dataframe debe conservar todas sus filas?" - Utiliza éste como base. Una unión a la izquierda conserva todas las filas del primer dataframe escrito en el comando, mientras que una unión a la derecha conserva todas las filas del segundo dataframe.

Los dos comandos de abajo consiguen el mismo resultado - 10 filas de hosp\_info unidas en una línea de base linelist\_mini, pero utilizan diferentes uniones. El resultado es que el orden de las columnas variará en función de si hosp\_info llega por la derecha (en la unión izquierda) o llega por la izquierda (en la unión derecha). El orden de las filas también puede cambiar en consecuencia. Pero ambas consecuencias pueden ser tratadas posteriormente, utilizando select() para reordenar las columnas o arrange() para ordenar las filas.

Este es el resultado de hosp\_info en linelist\_mini a través de una unión a la izquierda (nuevas columnas entrando por la derecha)

Este es el resultado de hosp\_info en linelist\_mini a través de una unión a la derecha (nuevas columnas entrando desde la izquierda)

Considere también si su caso de uso está dentro de una cadena de tuberías (%>%). Si el set de datos de las tuberías es la línea de base, es probable que utiliza una unión izquierda para añadir datos a ella.

### Incorporación completa

**Una unión completa es la más inclusiva de las uniones**: devuelve todas las filas de ambos dataframes.

Si hay filas presentes en una y no en la otra (donde no se encontró ninguna coincidencia), el dataframe las incluirá y se hará más largo. Los valores faltantes NA se utilizan para rellenar los huecos creados. A medida que se une, observe el número de columnas y filas con cuidado para solucionar el problema de las coincidencias de mayúsculas y minúsculas y de los caracteres exactos.

El dataframe "base" es el que se escribe primero en el comando. El ajuste de esto no afectará a los registros devueltos por la unión, pero puede afectar al orden de las columnas resultantes, al orden de las filas y a las columnas de los identificadores que se conservan.

Ejemplo animado de una unión completa ([fuente de la imagen](https://github.com/gadenbuie/tidyexplain/tree/master/images))

**Ejemplo**

A continuación se muestra la salida de un full\_join() de hosp\_info (originalmente nrow(hosp\_info), [ver aquí](#joins_llmini)) en linelist\_mini (originalmente nrow(linelist\_mini), [ver aquí](#joins_llmini)). Ten en cuenta lo siguiente:

* Se mantienen todas las líneas de base (linelist\_mini)
* Las filas de la secundaria que no coinciden con la línea de base se mantienen ("ignace" y "sisters"), y los valores de las columnas correspondientes de la línea de base case\_id y onset se rellenan con los valores que faltan
* Asimismo, las filas del dataframe de referencia que no coinciden con el secundario ("Otros" y "Falta") se mantienen, y las columnas secundarias catchment\_pop y level se rellenan con los valores que faltan
* En el caso de coincidencias de uno a muchos o de muchos a uno (por ejemplo, filas para "Hospital Militar"), se devuelven todas las combinaciones posibles (alargando el dataframe final)
* Sólo se conserva la columna del identificador de la línea de base (hospital)

### Unión interna

**Una unión interna es la más restrictiva de las uniones**: sólo devuelve las filas que coinciden en ambos dataframes.   
Esto significa que el número de filas en el dataframe de referencia puede reducirse. El ajuste de qué dataframe es la "línea de base" (escrito en primer lugar en la función) no afectará a las filas que se devuelven, pero sí al orden de las columnas, al orden de las filas y a las columnas de los identificadores que se conservan.

Ejemplo animado de una unión interna ([fuente de la imagen](https://github.com/gadenbuie/tidyexplain/tree/master/images))

**Ejemplo**

A continuación se muestra la salida de un inner\_join() de linelist\_mini (línea de base) con hosp\_info (secundario). Observe lo siguiente:

* Se eliminan las filas de la línea de base que no coinciden con los datos secundarios (filas en las que el hospital es "Ausente" u "Otro")
* Asimismo, se eliminan las filas del dataframe secundario que no tenían ninguna coincidencia en la línea de base (filas en las que hosp\_name es "sisters" o "ignace")
* Sólo se conserva la columna del identificador de la línea de base (hospital)

### Semi-unión

Un semi join es un "filtering join" que utiliza otro conjunto de datos no para añadir filas o columnas, sino para realizar un filtrado.

Un **semi-join mantiene todas las observaciones en el dataframe de referencia que tienen una coincidencia en el dataframe secundario** (pero no añade nuevas columnas ni duplica ninguna fila para las coincidencias múltiples). Lea más sobre estas uniones de "filtrado" [aquí](https://towardsdatascience.com/level-up-with-semi-joins-in-r-a068426096e0).

Ejemplo animado de una semiunión ([fuente de la imagen](https://github.com/gadenbuie/tidyexplain/tree/master/images))

Como ejemplo, el siguiente código devuelve las filas del dataframe hosp\_info que tienen coincidencias en linelist\_mini basadas en el nombre del hospital.

### Anti unión

**El anti join es otro "filtering join" que devuelve las filas del dataframe de referencia que no tienen una coincidencia en el dataframe secundario.**

Lea más sobre el filtrado de las uniones [aquí](https://towardsdatascience.com/level-up-with-semi-joins-in-r-a068426096e0).

Los escenarios comunes para un anti-join incluyen la identificación de registros que no están presentes en otro dataframe, la solución de problemas de ortografía en un join (revisión de registros que deberían haber coincidido) y el examen de registros que fueron excluidos después de otro join.

**Al igual que con right\_join() y left\_join(), el dataframe de la línea de base (que aparece primero) es importante**. Las filas devueltas son sólo las del dataframe de referencia. Observe en el siguiente gif que la fila del dataframe secundario (fila púrpura 4) no se devuelve a pesar de que no coincide con la línea de base.

Ejemplo animado de una antijunta ([fuente de la imagen](https://github.com/gadenbuie/tidyexplain/tree/master/images))

#### Ejemplo simple de anti\_join()

Para un ejemplo sencillo, encontremos los hospitales de hosp\_info que no tienen ningún caso presente en linelist\_mini. Enumeramos primero hosp\_info, como dataframe de referencia. Se devuelven los hospitales que no están presentes en linelist\_mini.

#### Ejemplo de anti\_join() complejo

Para otro ejemplo, digamos que ejecutamos un inner\_join() entre linelist\_mini y hosp\_info. Esto devuelve sólo un subconjunto de los registros originales de linelist\_mini, ya que algunos no están presentes en hosp\_info.

Para revisar los registros de linelist\_mini que fueron excluidos durante el inner join, podemos ejecutar un anti-join con la misma configuración (linelist\_mini como línea de base).

Para ver los registros de hosp\_info que se excluyeron en la unión interna, también podríamos ejecutar una anti unión con hosp\_info como dataframe de referencia.

## Emparejamiento probabilístico

Si no dispone de un identificador único común a todos los conjuntos de datos para unirlos, considere la posibilidad de utilizar un algoritmo de coincidencia probabilística. Este algoritmo buscaría coincidencias entre los registros basándose en la similitud (por ejemplo, la distancia de cadena de Jaro-Winkler o la distancia numérica). A continuación se muestra un ejemplo sencillo utilizando el paquete **fastLink** .

**Cargar paquetes**

A continuación se presentan dos pequeños conjuntos de datos de ejemplo que utilizaremos para demostrar la correspondencia probabilística (casos y test\_resultados):

Aquí está el código utilizado para hacer los conjuntos de datos:

**El set de datos de casos tiene 9 registros** de pacientes que están a la espera de los resultados de las pruebas.

**El **set** de datos test\_results** tiene 14 registros y contiene la columna resultado, que queremos añadir a los registros en casos basados en la coincidencia probabilística de registros.

### Correspondencia probabilística

La función fastLink() del paquete **fastLink** puede utilizarse para aplicar un algoritmo de coincidencia. Esta es la información básica. Puedes leer más detalles introduciendo ?fastLink en su consola.

* Definir los dos dataframes para la comparación con los argumentos dfA = y dfB =
* En varnames = indique todos los nombres de columnas que se utilizarán para la comparación. Todos ellos deben existir tanto en dfA como en dfB.
* En stringdist.match = dar columnas de las que están en varnames para ser evaluadas en la cadena "distancia".
* En numeric.match = dar columnas de las que están en varnames para ser evaluadas en la distancia numérica.
* Los valores faltantes se ignoran
* Por defecto, cada fila de cualquiera de los dos dataframes coincide como máximo con una fila del otro dataframe. Si desea ver todas las coincidencias evaluadas, establezca dedupe.matches = FALSE. La deduplicación se realiza mediante la solución de asignación lineal de Winkler.

Sugerencia: dividir una columna de fecha en tres columnas numéricas separadas utilizando *day()*, *month()* y *year()* del paquete ***lubridate***

El umbral por defecto para las coincidencias es de 0,94 (umbral.coincidencia =), pero puede ajustarlo más alto o más bajo. Si define el umbral, Ten en cuenta que los umbrales más altos podrían producir más falsos negativos (filas que no coinciden y que en realidad deberían coincidir) y, del mismo modo, un umbral más bajo podría producir más falsos positivos.

A continuación, los datos se emparejan según la distancia de las cadenas en las columnas de nombre y distrito, y según la distancia numérica para el año, el mes y el día de nacimiento. Se establece un umbral de coincidencia del 95% de probabilidad.

**Revisar los partidos**

Definimos el objeto devuelto por fastLink() como fl\_output. Es de tipo lista, y en realidad contiene varios dataframes dentro de él, detallando los resultados de la coincidencia. Uno de estos dataframes es matches, que contiene las coincidencias más probables entre los casos y los resultados. Puedes acceder a este dataframe "coincidencias" con fl\_output$matches. A continuación, se guarda como my\_matches para facilitar el acceso posterior.

Cuando se imprime my\_matches, se ven dos vectores de columnas: los pares de números de fila/índices (también llamados "rownames") en los casos ("inds.a") y en los resultados ("inds.b") que representan las mejores coincidencias. Si falta un número de fila de un dataframe, entonces no se ha encontrado ninguna coincidencia en el otro dataframe con el umbral de coincidencia especificado.

Cosas a tener en cuenta:

* Las coincidencias se produjeron a pesar de las ligeras diferencias en la ortografía del nombre y las fechas de nacimiento:
  + "Tony B. Smith" coincide con "Anthony B Smith"
  + "María Rodríguez" coincide con "Marialisa Rodrigues"
  + "Betty Chase" coincide con "Elizabeth Chase"
  + "Olivier Laurent De Bordeaux" coincide con "Oliver Laurent De Bordow" (se ignora la fecha de nacimiento que falta)
* Una fila de casos (para "Blessing Adebayo", fila 9) no tuvo una buena coincidencia en los resultados, por lo que no está presente en my\_matches.

**Unir en base a las coincidencias probabilísticas**

Para utilizar estas coincidencias para unir los resultados a los casos, una estrategia es:

1. Utilizar left\_join() para unir mis\_parejas a los casos (haciendo coincidir los nombres de las ropas en los casos con "inds.a" en mis\_parejas)
2. A continuación, utiliza otro left\_join() para unir los resultados a los casos (haciendo coincidir los "inds.b" recién adquiridos en los casos con los rownames en los resultados)

Antes de las uniones, debemos limpiar los tres dataframes:

* Tanto dfA como dfB deben tener sus números de fila ("rowname") convertidos en una columna propia.
* Las dos columnas de my\_matches se convierten en tipo carácter, por lo que pueden unirse a los nombres de los personajes

Como se realiza utilizando el código anterior, el dataframe resultante completo contendrá todas las columnas tanto de los casos como de los resultados. A muchas de ellas se les añadirán los sufijos ".x" e ".y", ya que de lo contrario los nombres de las columnas estarían duplicados.

Alternativamente, para conseguir sólo los 9 registros "originales" en los casos con la(s) nueva(s) columna(s) de los resultados, utiliza select() en los resultados antes de las uniones, de forma que sólo contenga los nombres de las rutas y las columnas que desea añadir a los casos (por ejemplo, la columna resultado).

Si desea subconjuntar cualquiera de los dos conjuntos de datos sólo con las filas que coincidan, puede utilizar los siguientes códigos:

O, para ver sólo las filas que **no** coinciden:

### Deduplicación probabilística

La coincidencia probabilística también puede utilizarse para deduplicar unos datos. Consulte la página sobre deduplicación para conocer otros métodos de deduplicación.

Aquí comenzamos con el set de datos cases, pero ahora lo llamamos cases\_dup, ya que tiene 2 filas adicionales que podrían ser duplicados de filas anteriores: Ver "Tony" con "Anthony", y "Marialisa Rodrigues" con "Maria Rodriguez".

Ejecuta fastLink() como antes, pero compare el dataframe cases\_dup consigo mismo. Cuando los dos dataframes proporcionados son idénticos, la función asume que se quiere desduplicar. Observe que no especificamos stringdist.match = o numeric.match = como hicimos anteriormente.

Ahora, puede revisar los duplicados potenciales con getMatches(). Proporcione el dataframe como dfA = y dfB =, y proporcione la salida de la función fastLink() como fl.out =. fl.out debe ser del tipo fastLink.dedupe, o en otras palabras, el resultado de fastLink().

Véase la columna de la derecha, que indica los ID duplicados: las dos últimas filas se identifican como probables duplicados de las filas 2 y 3.

Para devolver los números de fila de las filas que probablemente sean duplicadas, puede contar el número de filas por valor único en la columna dedupe.ids, y luego filtrar para mantener sólo aquellas con más de una fila. En este caso, esto deja las filas 2 y 3.

Para inspeccionar las filas completas de los probables duplicados, ponga el número de fila en este comando:

## Encuadernación y alineación

Otro método para combinar dos dataframes es "unirlos". También se puede pensar en esto como "anexar" o "añadir" filas o columnas.

En esta sección también se discutirá cómo "alinear" el orden de las filas de un dataframe con el orden de otro dataframe. Este tema se discute más adelante en la sección sobre Vinculación de columnas.

### Encuadernar filas

Para unir las filas de un dataframe con el fondo de otro dataframe, utiliza bind\_rows() de **dplyr**. Es muy inclusivo, por lo que cualquier columna presente en cualquiera de los dataframes se incluirá en la salida. Algunas notas:

* A diferencia de la versión **básica** de R row.bind(), bind\_rows() de **dplyr** no requiere que el orden de las columnas sea el mismo en ambos dataframes. Siempre que los nombres de las columnas se escriban de forma idéntica, las alineará correctamente.
* Puedes especificar opcionalmente el argumento .id =. Proporcionar un nombre de columna de caracteres. Esto producirá una nueva columna que sirve para identificar de qué dataframe procede originalmente cada fila.
* Puedes utilizar bind\_rows() en una lista de dataframes de estructura similar para combinarlos en un dataframe. Vea un ejemplo en la página [Iteración, bucles y listas](#iteration-loops-and-lists) que implica la importación de múltiples listas de líneas con **purrr**.

Un ejemplo común de vinculación de filas es vincular una fila "total" a una tabla descriptiva hecha con la función summarise() de **dplyr**. A continuación, creamos una tabla de recuentos de casos y valores medianos de TC por hospital con una fila de totales.

La función summarise() se utiliza en los datos agrupados por hospital para devolver un dataframe resumido por hospital. Pero la función summarise() no produce automáticamente una fila de "totales", así que la creamos resumiendo los datos de nuevo, pero con los datos no agrupados por hospital. Esto produce un segundo dataframe de una sola fila. A continuación, podemos unir estos dataframes para obtener la tabla final.

Vea otros ejemplos trabajados como éste en las páginas de [Tablas descriptivas](#descriptive-tables) y [Tablas de presentación](#tables-for-presentation).

Este es el dataframe de hosp\_summary:

Cree un dataframe con las estadísticas "totales" (no agrupadas por hospital). Esto devolverá una sola fila.

Y a continuación está el dataframe de los totales. Observe que sólo hay dos columnas. Estas columnas también están en hosp\_summary, pero hay una columna en hosp\_summary que no está en los totales (hospital).

Ahora podemos unir las filas con bind\_rows().

Ahora podemos ver el resultado. Observe cómo en la última fila se rellena un valor NA vacío para la columna hospital que no estaba en hosp\_summary. Como se explica en la página de [Tablas para la presentación](#tables-for-presentation), podría "rellenar" esta celda con "Total" utilizando replace\_na().

### Vincular columnas

Existe una función similar **de dplyr** bind\_cols() que se puede utilizar para combinar dos dataframes de forma lateral. Ten en cuenta que las filas se emparejan entre sí por posición (no como una unión anterior) - por ejemplo, la fila 12 en cada dataframe se alineará.

Como ejemplo, unimos varias tablas de resumen. Para ello, también demostramos cómo reordenar el orden de las filas de un dataframe para que coincida con el orden de otro dataframe, con match().

Aquí definimos case\_info como un dataframe resumido de los casos del listado, por hospital, con el número de casos y el número de muertes.

Y digamos que aquí hay un dataframe diferente contact\_fu que contiene información sobre el porcentaje de contactos expuestos investigados y "seguidos", de nuevo por hospital.

Observe que los hospitales son los mismos, pero están en diferente orden en cada dataframe. La solución más sencilla sería utilizar un left\_join() en la columna de hospitales, pero también podría utilizar bind\_cols() con un paso adicional.

#### Utiliza match() para alinear la ordenación

Debido a que los órdenes de las filas son diferentes, un simple comando bind\_cols() daría lugar a un desajuste de los datos. Para solucionarlo podemos utilizar match() de R **base** para alinear las filas de un dataframe en el mismo orden que en otro. Asumimos para este enfoque que no hay valores duplicados en ninguno de los dos dataframes.

Cuando utilizamos match(), la sintaxis es match(VECTOR DE ORDENAMIENTO OBJETIVO, COLUMNA DEL DATAFRAME A CAMBIAR), donde el primer argumento es el orden deseado (ya sea un vector independiente, o en este caso una columna en un dataframe), y el segundo argumento es la columna del dataframe que se reordenará. La salida de match() es un vector de números que representa el ordenamiento correcto de las posiciones. Puedes obtener más información con ?match.

Puedes utilizar este vector numérico para reordenar el dataframe - colóquelo dentro de los subcorchetes [ ] antes de la coma. Lea más sobre **la** sintaxis del subconjunto de corchetes en la página de [fundamentos de R](#r-basics). El comando de abajo crea un nuevo dataframe, definido como el anterior en el que las filas están ordenadas en el vector numérico de arriba.

Ahora podemos unir las columnas del dataframe, con el orden correcto de las filas. Ten en cuenta que algunas columnas están duplicadas y será necesario limpiarlas con rename(). Lea más sobre bind\_rows() [aquí](https://dplyr.tidyverse.org/reference/bind.html).

Una alternativa en R básico a bind\_cols es cbind(), que realiza la misma operación.

## Recursos

La [página de tidyverse sobre las uniones](https://dplyr.tidyverse.org/reference/join.html)

La [página de R for Data Science sobre datos relacionales](https://r4ds.had.co.nz/relational-data.html)

La [página de tidyverse en dplyr](https://dplyr.tidyverse.org/reference/bind.html) en la encuadernación

Una viñeta sobre [fastLink](https://github.com/kosukeimai/fastLink) en la página de Github del paquete

Publicación que describe la metodología de [fastLink](https://imai.fas.harvard.edu/research/files/linkage.pdf)

Publicación que describe el [paquete RecordLinkage](https://journal.r-project.org/archive/2010/RJ-2010-017/RJ-2010-017.pdf)

# #Desduplicación

{#de-duplication}

Esta página cubre las siguientes técnicas de desduplicación:

1. Identificación y eliminación de filas duplicadas
2. "Rebanar" las filas para mantener sólo determinadas filas (por ejemplo, mínimas o máximas) de cada grupo de filas
3. "Rolling-up", o combinación de valores de varias filas en una sola fila

## Preparación

### Cargar paquetes

Este trozo de código muestra la carga de los paquetes necesarios para los análisis. En este manual destacamos p\_load() de **pacman**, que instala el paquete si es necesario y lo carga para su uso. También puede cargar los paquetes instalados con library() de R **base.** Consulte la página sobre [los fundamentos de R](#r-basics) para obtener más información sobre los paquetes de R.

### Importar datos

Para la demostración, utilizaremos unos datos de ejemplo que se crea con el código R que aparece a continuación.

Los datos son registros de encuentros telefónicos COVID-19, incluyendo encuentros con contactos y con casos. Las columnas incluyen recordID (generado por ordenador), personID, nombre, fecha del encuentro, hora del encuentro, el propósito del encuentro (ya sea para entrevistar como caso o como contacto), y symptoms\_ever (si la persona en ese encuentro reportó haber tenido síntomas alguna vez).

Este es el código para crear el set de datos obs:

#### Este es el dataframe

Utiliza los cuadros de filtro de la parte superior para revisar los encuentros de cada persona.

Hay que tener en cuenta algunas cosas al revisar los datos:

* Los dos primeros registros son 100% duplicados, incluido el ID de registro duplicado (¡debe ser un fallo informático!)
* Las dos segundas filas son duplicadas, en todas las columnas excepto en *recordID*
* Varias personas tuvieron múltiples encuentros telefónicos, en diversas fechas y horas, y como contactos y/o casos
* En cada encuentro se preguntaba a la persona si había tenido **alguna vez** síntomas, y parte de esta información falta.

Y aquí hay un resumen rápido de las personas y los propósitos de sus encuentros, usando tabyl() de **janitor**:

## Deduplicación

Esta sección describe cómo revisar y eliminar filas duplicadas en un dataframe. También muestra cómo manejar los elementos duplicados en un vector.

### Examinar las filas duplicadas

Para revisar rápidamente las filas que tienen duplicados, puede utilizar get\_dupes() del paquete **janitor**. Por defecto, se consideran todas las columnas cuando se evalúan los duplicados - las filas devueltas por la función son 100% duplicadas considerando los valores de todas las columnas.

En el dataframe obs, las dos primeras filas son 100% duplicadas - tienen el mismo valor en cada columna (incluyendo la columna recordID, que se supone que es única - debe ser algún fallo informático). El dataframe devuelto incluye automáticamente una nueva columna dupe\_count en el lado derecho, que muestra el número de filas con esa combinación de valores duplicados.

Ver los [datos originales](#dedup_data)

Sin embargo, si decidimos ignorar recordID, las filas 3 y 4 también son duplicados entre sí. Es decir, tienen los mismos valores en todas las columnas excepto en recordID. Puedes especificar las columnas específicas que se van a ignorar en la función mediante el símbolo - menos.

También puede especificar positivamente las columnas a considerar. A continuación, sólo se devuelven las filas que tienen los mismos valores en las columnas nombre y propósito. Observe cómo "amrish" tiene ahora dupe\_count igual a 3 para reflejar sus tres encuentros de "contacto".

Desplácese a la izquierda para ver más filas\*\*

Vea los [datos originales](#dedup_data).

Para más detalles, consulte la sección "get\_dupes" o esta [referencia en línea](https://cran.r-project.org/web/packages/janitor/vignettes/janitor.html" \l "explore-records-with-duplicated-values-for-specific-combinations-of-variables-with-get_dupes)

### Mantener sólo filas únicas

Para mantener sólo las filas únicas de un dataframe, utiliza distinct() de **dplyr** (como se demuestra en la página [Limpieza de datos y funciones básicas](#cleaning-data-and-core-functions)). Las filas duplicadas se eliminan de forma que sólo se conserva la primera de dichas filas. Por defecto, "primero" significa el número de fila más alto (orden de filas de arriba a abajo). Sólo se mantienen las filas únicas.

En el ejemplo siguiente, ejecutamos distinct() de forma que la columna recordID se excluye de la consideración - así **se eliminan dos filas duplicadas**. La primera fila (para "adam") estaba 100% duplicada y ha sido eliminada. También la fila 3 (para "amrish") estaba duplicada en todas las columnas excepto en recordID (que no se tiene en cuenta), por lo que también se ha eliminado. El set de datos de observaciones n es ahora nrow(obs)-2, no nrow(obs) rows).

Desplácese a la izquierda para ver el dataframe completo

**ATENCIÓN:** Si se utiliza distinct() en datos agrupados, la función se aplicará a cada grupo.

**Deduplicar en base a columnas específicas**

También puede especificar las columnas que serán la base de la desduplicación. De esta manera, la desduplicación sólo se aplica a las filas que son duplicadas dentro de las columnas especificadas. A menos que establezca .keep\_all = TRUE, todas las columnas no mencionadas se eliminarán.

En el ejemplo siguiente, la desduplicación sólo se aplica a las filas que tienen valores idénticos para las columnas de nombre y propósito. Por lo tanto, "brian" sólo tiene 2 filas en lugar de 3: su primer encuentro de "contacto" y su único encuentro de "caso". Para ajustar que se mantenga el último encuentro de brian de cada propósito, vea la pestaña de Rebanar dentro de los grupos.

Desplácese a la izquierda para ver el dataframe completo

Vea los [datos originales](#dedup_data).

### Deduplicar elementos en un vector

La función duplicado() de **base** R evaluará un vector (columna) y devolverá un vector lógico de la misma longitud (TRUE/FALSE). La primera vez que aparezca un valor, devolverá FALSE (no es un duplicado), y las siguientes veces que aparezca ese valor devolverá TRUE. Nótese que NA se trata igual que cualquier otro valor.

Para devolver sólo los elementos duplicados, se pueden utilizar paréntesis para subconjuntar el vector original:

Para devolver sólo los elementos únicos, utiliza unique() de la **base** R. Para eliminar los NA de la salida, anide na.omit() dentro de unique().

### Utilizando ****la base**** R

**Para devolver las filas duplicadas**

En R **base**, también se puede ver qué filas son 100% duplicadas en un dataframe df con el comando duplicado(df) (devuelve un vector lógico de las filas).

Así, también puede utilizar el subconjunto base [ ] en el dataframe para ver las filas duplicadas con df[duplicado(df),] (¡no olvide la coma, que significa que quiere ver todas las columnas!)

**Para devolver filas únicas**

Ver las notas anteriores. Para ver las filas únicas se añade el negador lógico ! delante de la función duplicada():  
df[!duplicado(df),]

**Para devolver las filas que son duplicados de sólo ciertas columnas**

Subconjunta el df que está dentro de los paréntesis de *duplicado()*, para que esta función opere sólo en ciertas columnas del df.

Para especificar las columnas, proporcione los números o nombres de las columnas después de una coma (recuerde que todo esto está dentro de la función duplicada()).

Asegúrese de mantener la coma , fuera después de la función duplicada() también!

Por ejemplo, para evaluar sólo las columnas 2 a 5 en busca de duplicados: df[!duplicated(df[, 2:5]),]  
Para evaluar sólo las columnas nombre y propósito en busca de duplicados: df[!duplicated(df[, c("nombre", "propósito)]),]

## Rebanar

Para "cortar" un dataframe para aplicar un filtro en las filas por número de fila/posición. Esto resulta especialmente útil si tiene varias filas por grupo funcional (por ejemplo, por "persona") y sólo quiere conservar una o algunas de ellas.

La función básica slice() acepta números y devuelve filas en esas posiciones. Si los números proporcionados son positivos, sólo se devuelven éstos. Si son negativos, no se devuelven esas filas. Los números deben ser todos positivos o todos negativos.

Vea los [datos originales](#dedup_data).

Existen diversas variantes: Se les debe proporcionar una columna y un número de filas a devolver (a n =).

* slice\_min() y slice\_max() mantienen sólo la(s) fila(s) con el valor(es) mínimo o máximo de la columna especificada. Esto también funciona para devolver el "min" y el "max" de los factores ordenados.
* slice\_head() y slice\_tail() - mantienen sólo la primera o la última fila.
* slice\_sample() - mantener sólo una muestra aleatoria de las filas.

Utiliza los argumentos n = o prop = para especificar el número o la proporción de filas que deben conservarse. Si no se utiliza la función en una cadena de tuberías, proporcione primero el argumento datos (por ejemplo, slice(datos, n = 2)). Para más información, consulte la sección "slice".

Otros argumentos:

.order\_by = utilizado en slice\_min() y slice\_max() esta es una columna para ordenar por antes de rebanar.  
with\_ties = TRUE por defecto, lo que significa que se mantienen los empates.  
.preserve = FALSE por defecto. Si es TRUE, la estructura de agrupación se recalcula después del corte.  
weight\_by = Opcional, columna numérica para ponderar por (un número mayor tiene más probabilidades de ser muestreado). También replace = para saber si el muestreo se realiza con/sin reemplazo.

**CONSEJO:** Al utilizar slice\_max() y slice\_min(), asegúrese de especificar/escribir el n = (por ejemplo, n = 2, no sólo 2). De lo contrario, puede obtener un error Error:...no está vacío.

**NOTA:** Es posible que encuentre la función [top\_n()](https://dplyr.tidyverse.org/reference/top_n.html), que ha sido sustituida por las funciones de corte.

### Cortar con grupos

Las funciones slice\_\*() pueden ser muy útiles si se aplican a un dataframe agrupado porque la operación de corte se realiza en cada grupo por separado. Utiliza la **función** group\_by() junto con slice() para agrupar los datos y tomar un corte de cada grupo.

Esto es útil para la desduplicación si tiene varias filas por persona pero sólo quiere mantener una de ellas. Primero se utiliza group\_by() con columnas clave que son las mismas por persona, y luego se utiliza una función slice en una columna que será diferente entre las filas agrupadas.

En el ejemplo siguiente, para mantener sólo el último encuentro por persona, agrupamos las filas por nombre y luego utilizamos slice\_max() con n = 1 en la columna de fecha. Ten en cuenta que Para aplicar una función como slice\_max() en las fechas, la columna de fecha debe ser de tipo Date.

Por defecto, los "empates" (por ejemplo, la misma fecha en este escenario) se mantienen, y todavía obtendríamos múltiples filas para algunas personas (por ejemplo, adam). Para evitar esto, establecemos with\_ties = FALSE. Sólo obtendremos una fila por persona.

**ATENCIÓN:** Si utiliza arrange(), especifique .by\_group = TRUE para que los datos se ordenen dentro de cada grupo.

**PELIGRO:** Si with\_ties = FALSE, se mantiene la primera fila de un empate. Esto puede ser engañoso. Vea cómo para Mariah, ella tiene dos encuentros en su última fecha (6 de enero) y el primero (el más temprano) se mantuvo. Es probable que queramos mantener su último encuentro en ese día. Vea cómo "romper" estos vínculos en el siguiente ejemplo.

Arriba, por ejemplo, podemos ver que sólo se conservó la fila de Amrish del 5 de enero, y sólo se conservó la fila de Brian del 7 de enero. Vea los [datos originales](#dedup_data).

**Romper los "lazos"**

Se pueden ejecutar múltiples sentencias de corte para "romper empates". En este caso, si una persona tiene varios encuentros en su última fecha, se mantiene el encuentro con la última hora (se utiliza lubridate::hm() para convertir los tiempos de los caracteres en un tipo de tiempo ordenable).   
Observe cómo ahora, la única fila que se mantiene para "Mariah" el 6 de enero es el encuentro 3 de las 08:32, no el encuentro 2 de las 07:25.

En el ejemplo anterior, también habría sido posible realizar un corte por número de *encuentro*, pero mostramos el corte por *fecha* y *hora a* modo de ejemplo.

**CONSEJO:** Para utilizar slice\_max() o slice\_min() en una columna "carácter", ¡mútela a un tipo de factor ordenado!

Vea los [datos originales](#dedup_data).

### Manténgalos todos pero márquelos

Si desea conservar todos los registros pero marcar sólo algunos para su análisis, considere un enfoque de dos pasos utilizando un número de registro/encuentro único:

1. Reduzca/corte el dataframe original a sólo las filas para el análisis. Guarde/conserve este dataframe reducido.
2. En el dataframe original, marque las filas según corresponda con case\_when(), basándose en si su identificador único de registro (recordID en este ejemplo) está presente en el dataframe reducido.

Vea los [datos originales](#dedup_data).

### Calcular la integridad de las filas

Cree una columna que contenga una métrica para la integridad de la fila (que no falte). Esto podría ser útil a la hora de decidir qué filas se priorizan sobre otras al desduplicar/repartir.

En este ejemplo, las columnas "clave" sobre las que se quiere medir la integridad se guardan en un vector de nombres de columnas.

A continuación se crea la nueva columna clave\_completa con mutate(). El nuevo valor de cada fila se define como una fracción calculada: el número de valores no ausentes en esa fila entre las columnas clave, dividido por el número de columnas clave.

Esto implica la función rowSums() de la **base** R. También se utiliza . , que dentro de la tubería se refiere al dataframe en ese punto de la tubería (en este caso, se está subconjuntando con corchetes []).

Desplácese a la derecha para ver más filas\*\*.

Vea los [datos originales](#dedup_data).

## Valores de roll-up

Esta sección describe:

1. Cómo "enrollar" valores de varias filas en una sola fila, con algunas variaciones
2. Una vez que haya "enrollado" los valores, cómo sobrescribir/priorizar los valores en cada celda

Esta pestaña utiliza los datos de ejemplo de la pestaña Preparación.

### Enrollar los valores en una fila

El ejemplo de código que se muestra a continuación utiliza group\_by() y summarise() para agrupar las filas por persona, y luego pega todos los valores únicos dentro de las filas agrupadas. Así, se obtiene una fila de resumen por persona. Algunas notas:

* Se añade un sufijo a todas las nuevas columnas ("\_roll" en este ejemplo)
* Si quieres mostrar sólo los valores únicos por celda, entonces envuelve el na.omit() con unique()
* na.omit() elimina los valores NA, pero si no se desea se puede eliminar paste0(.x)...

El resultado es una fila por grupo (ID), con entradas ordenadas por fecha y pegadas. Desplácese a la izquierda para ver más filas

Vea los [datos originales](#dedup_data).

**Esta variación sólo muestra valores únicos:**

**Esta variación añade un sufijo a cada columna.**   
En este caso, "\_roll" para indicar que se ha rodado:

### Sobrescribir valores/jerarquía

Si luego quiere evaluar todos los valores rodados, y mantener sólo un valor específico (por ejemplo, el "mejor" o el "máximo" valor), puede utilizar mutate() a través de las columnas deseadas, para implementar case\_when(), que utiliza str\_detect() del paquete **stringr** para buscar secuencialmente patrones de cadena y sobrescribir el contenido de la celda.

Ahora puede ver en la columna symptoms\_ever que si la persona alguna vez dijo "Sí" a los síntomas, entonces sólo se muestra "Sí".

Vea los [datos originales](#dedup_data).

## Desduplicación probabilística

A veces, puede querer identificar duplicados "probables" basándose en la similitud (por ejemplo, la "distancia" de la cadena) en varias columnas como el nombre, la edad, el sexo, la fecha de nacimiento, etc. Puedes aplicar un algoritmo de coincidencia probabilística para identificar duplicados probables.

Consulte la página sobre la [unión de datos](#joining-data) para obtener una explicación sobre este método. La sección sobre Coincidencia probabilística contiene un ejemplo de aplicación de estos algoritmos para comparar un dataframe consigo mismo, realizando así una desduplicación probabilística.

## Recursos

Gran parte de la información de esta página está adaptada de estos recursos y viñetas en línea:

[datanovia](https://www.datanovia.com/en/lessons/identify-and-remove-duplicate-data-in-r/)

[Referencia de dplyr tidyverse](https://dplyr.tidyverse.org/reference/slice.html)

[viñeta del conserje de la manivela](https://cran.r-project.org/web/packages/janitor/vignettes/janitor.html" \l "explore-records-with-duplicated-values-for-specific-combinations-of-variables-with-get_dupes)

# #Iteración, bucles y listas

{#iteration-loops-and-lists}

Con frecuencia nos enfrentamos a menudo a la repetición de análisis por subgrupos como países, distritos o grupos de edad. Estas son sólo algunas de las situaciones frecuentes que implican la iteración. La codificación de sus operaciones iterativas utilizando los enfoques que se indican a continuación le ayudará a realizar estas tareas repetitivas más rápidamente, a reducir la posibilidad de error y a reducir la longitud del código.

Esta página presentará dos enfoques de las operaciones iterativas: el uso de bucles for y el uso del paquete **purrr**.

1. Los bucles for iteran el código a través de una serie de entradas, pero son menos comunes en R que en otros lenguajes de programación. No obstante, los presentamos aquí como herramienta de aprendizaje y referencia
2. El paquete **purrr** es el enfoque **tidyverse** de las operaciones iterativas - funciona "mapeando" una función a través de muchas entradas (valores, columnas, conjuntos de datos, etc.)

En el camino, mostraremos ejemplos como:

* Importación y exportación de múltiples archivos
* Creación de epicurvas para múltiples jurisdicciones
* Ejecución de pruebas T para varias columnas en un dataframe

En la [sección de](#iter_purrr) **purrr** también proporcionaremos varios ejemplos de creación y manejo de listas.

## Preparación

### Cargar paquetes

Este trozo de código muestra la carga de los paquetes necesarios para los análisis. En este manual destacamos p\_load() de **pacman**, que instala el paquete si es necesario y lo carga para su uso. También puede cargar los paquetes instalados con library() de R **base.** Consulte la página sobre [los fundamentos de R](#r-basics) para obtener más información sobre los paquetes de R.

### Importar datos

Importamos los datos de casos de una epidemia de ébola simulada. Si quiere seguir el proceso, [clica para descargar linelist "limpia"](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/case_linelists/linelist_cleaned.rds) (como archivo .rds). Importe los datos con la función import() del paquete **rio** (maneja muchos tipos de archivos como .xlsx, .csv, .rds - vea la página de [importación y exportación](#import-and-export) para más detalles).

A continuación se muestran las primeras 50 filas del listado.

## para los bucles

### para los bucles en R

Los bucles for no se enfatizan en R, pero son comunes en otros lenguajes de programación. Como principiante, pueden ser útiles para aprender y practicar, ya que son más fáciles de "explorar", "depurar", y de otra manera comprender exactamente lo que está sucediendo para cada iteración, especialmente cuando todavía no te sientes cómodo escribiendo tus propias funciones.

Puedes pasar rápidamente de los bucles for a la iteración con funciones mapeadas con **purrr** (véase [la sección siguiente](#iter_purrr)).

### Componentes básicos

Un bucle for tiene tres partes fundamentales:

1. La **secuencia** de elementos a iterar
2. Las **operaciones** a realizar por elemento en la secuencia
3. El **contenedor** de los resultados (opcional)

La sintaxis básica es: for (elemento en la secuencia) {hacer operaciones con el elemento}. Observe los paréntesis y las llaves. Los resultados pueden ser impresos en la consola, o almacenados en un objeto R contenedor.

A continuación se muestra un sencillo ejemplo de bucle for.

### Secuencia

Esta es la parte "for" de un bucle for - las operaciones se ejecutarán "para" cada elemento de la secuencia. La secuencia puede ser una serie de valores (por ejemplo, nombres de jurisdicciones, enfermedades, nombres de columnas, elementos de listas, etc.), o puede ser una serie de números consecutivos (por ejemplo, 1,2,3,4,5). Cada enfoque tiene sus propias utilidades, que se describen a continuación.

La estructura básica de una declaración de secuencia es el elemento en el vector.

* Puedes escribir cualquier carácter o palabra en lugar de "item" (por ejemplo, "i", "num", "hosp", "district", etc.). El valor de este "elemento" cambia con cada iteración del bucle, pasando por cada valor del vector.
* El vector puede ser de valores de caracteres, nombres de columnas, o quizás una secuencia de números - estos son los valores que cambiarán con cada iteración. Puedes utilizarlos dentro de las operaciones del bucle for utilizando el término "item".

**Ejemplo: secuencia de valores de caracteres**

En este ejemplo, se realiza un bucle para cada valor de un vector de caracteres predefinido de nombres de hospitales.

Hemos elegido el término hosp para representar los valores del vector nombres\_de\_hospital. Para la primera iteración del bucle, el valor de hosp será hospital\_names[[1]]. Para la segunda iteración del bucle será nombres\_de\_hospital[[2]]. Y así sucesivamente...

**Ejemplo: secuencia de nombres de columnas**

Se trata de una variación de la secuencia de caracteres anterior, en la que se extraen los nombres de un objeto R existente y se convierten en el vector. Por ejemplo, los nombres de las columnas de un dataframe. Convenientemente, en el código de operaciones del bucle for, los nombres de las columnas se pueden utilizar para indexar (subconjuntar) su dataframe original

A continuación, la secuencia son los names() (nombres de columnas) del dataframe del listado. El nombre de nuestro "elemento" es col, que representará el nombre de cada columna a medida que avanzan los bucles.

A modo de ejemplo, incluimos código de operaciones dentro del bucle for, que se ejecuta para cada valor de la secuencia. En este código, los valores de la secuencia (nombres de las columnas) se utilizan para indexar (subconjuntar) linelist, una por una. Como se enseñó en la página de [fundamentos de R, se utilizan](#r-basics) dobles ramificaciones [[ ]] para el subconjunto. La columna resultante se pasa a is.na(), y luego a sum() para producir el número de valores de la columna que faltan. El resultado se imprime en la consola: un número por cada columna.

Una nota sobre la indexación con los nombres de las columnas - ¡cuando se refiera a la propia columna no escriba simplemente "col"! col representa sólo el nombre de la columna de caracteres! Para referirse a la columna completa debe utilizar el nombre de la columna como un índice en linelist a través de linelist[[col]].

**Secuencia de números**

En este enfoque, la secuencia es una serie de números consecutivos. Por lo tanto, el valor del "ítem" no es un valor de carácter (por ejemplo, "Hospital Central" o "fecha\_de\_inicio"), sino que es un número. Esto es útil para hacer un bucle a través de los dataframes, ya que puede utilizar el número del "ítem" dentro del bucle for para indexar el dataframe por número de fila.

Por ejemplo, digamos que quiere recorrer cada fila de su dataframe y extraer cierta información. Sus "elementos" serían números de fila numéricos. A menudo, los "elementos" en este caso se escriben como i.

El proceso del bucle for podría explicarse en palabras como "para cada elemento de una secuencia de números desde 1 hasta el número total de filas de mi dataframe, haz X". Para la primera iteración del bucle, el valor del "elemento" i sería 1. Para la segunda iteración, i sería 2, etc.

Aquí está el aspecto de la secuencia en código: for (i in 1:nrow(linelist)) {CODIGO DE OPERACIONES} donde i representa el "elemento" y 1:nrow(linelist) produce una secuencia de números consecutivos desde 1 hasta el número de filas en linelist.

Si desea que la secuencia sea numérica, pero parte de un vector (no de un dataframe), utiliza el atajo seq\_along() para devolver una secuencia de números para cada elemento del vector. Por ejemplo, para (i en seq\_along(nombres\_de\_hospital) {Código\_de\_operaciones}.

El código siguiente devuelve en realidad números, que se convertirían en el valor de i en su respectivo bucle.

Una ventaja de usar números en la secuencia es que es fácil usar también el número i para indexar un contenedor que almacene las salidas del bucle. Hay un ejemplo de esto en la sección de Operaciones más abajo.

### Operaciones

Este es el código dentro de las llaves { } del bucle for. Quieres que este código se ejecute para cada "elemento" de la secuencia. Por lo tanto, ¡ten cuidado de que cada parte de su código que cambia por el "ítem" esté correctamente codificado de manera que realmente cambie! Por ejemplo, recuerda usar [[ ]] para la indexación.

En el ejemplo siguiente, iteramos por cada fila del listado. Los valores de género y edad de cada fila se pegan juntos y se almacenan en el vector de caracteres contenedor cases\_demographics. Observe cómo también utilizamos la indexación [[i]] para guardar la salida del bucle en la posición correcta en el vector "contenedor".

### Contenedor

A veces los resultados de su bucle for se imprimirán en la consola o en el panel de gráficos de RStudio. Otras veces, querrá almacenar los resultados en un "contenedor" para su uso posterior. Dicho contenedor puede ser un vector, un dataframe o incluso una lista.

Lo más eficiente es crear el contenedor de los resultados antes de comenzar el bucle for. En la práctica, esto significa crear un vector vacío, un dataframe o una lista. Estos pueden ser creados con las funciones vector() para vectores o listas, o con matrix() y data.frame() para un dataframe.

**Vector vacío**

Utiliza vector() y especifique el modo = en función del tipo esperada de los objetos que va a insertar - ya sea "doble" (para contener números), "carácter" o "lógico". También debe establecer la longitud = por adelantado. Esta debe ser la longitud de su secuencia de bucle for.

Digamos que quiere almacenar la mediana de la demora hasta el ingreso para cada hospital. Utilizaría "double" y establecería que la longitud fuera el número de salidas esperadas (el número de hospitales únicos en el set de datos).

**Dataframe vacío**

Puedes hacer un dataframe vacío especificando el número de filas y columnas de esta manera:

**Lista vacía**

Es posible que desee almacenar algunos gráficos creados por un bucle for en una lista. Una lista es como un vector, pero contiene otros objetos R dentro de ella que pueden ser de diferentes tipos. Los elementos de una lista pueden ser un solo número, un dataframe, un vector e incluso otra lista.

En realidad, se inicializa una lista vacía utilizando el mismo comando vector() que el anterior, pero con modo = "lista". Especifica la longitud como quieras.

### Impresión

Ten en cuenta que para imprimir desde dentro de un bucle for probablemente tendrá que envolver explícitamente con la función print().

En este ejemplo, la secuencia es un vector de caracteres explícito, que se utiliza para subsumir linelist en un hospital. Los resultados no se almacenan en un contenedor, sino que se imprimen en la consola con la función print().

### Probando su bucle for

Para probar su bucle, puede ejecutar un comando para hacer una asignación temporal del "elemento", como i <- 10 o hosp <- "Hospital Central". Haga esto fuera del bucle y luego ejecuta su código de operaciones solamente (el código dentro de las llaves) para ver si se producen los resultados esperados.

### Gráficos en bucle

Para reunir los tres componentes (contenedor, secuencia y operaciones) vamos a intentar trazar una epicurva para cada hospital (véase la página sobre [curvas epidémicas](#epidemic-curves)).

Podemos hacer una bonita epicurva de todos los casos por género utilizando el paquete **incidence2** como se indica a continuación:

Para producir un gráfico separado para cada caso del hospital, podemos poner este código de epicurva dentro de un bucle for.

En primer lugar, guardamos un vector con nombre de los nombres únicos de los hospitales, hospital\_names. El bucle for se ejecutará una vez para cada uno de estos nombres: for (hosp en nombres\_de\_hospital). En cada iteración del bucle for, el nombre actual del hospital del vector se representará como hosp para su uso dentro del bucle.

Dentro de las operaciones del bucle, puede escribir el código R de forma normal, pero utilizando el "elemento" (hosp en este caso) sabiendo que su valor será cambiante. Dentro de este bucle:

* Se aplica un filtro() a linelist, de forma que la columna hospital debe ser igual al valor actual de hosp
* El objeto de incidencia se crea en linelist filtradas
* Se crea una gráfica del hospital actual, con un título autoajustable que utiliza hosp
* El gráfico del hospital actual se guarda temporalmente y luego se imprime
* El bucle sigue adelante para repetirse con el siguiente hospital en hospital\_names

### Seguimiento del progreso de un bucle

Un bucle con muchas iteraciones puede funcionar durante muchos minutos o incluso horas. Por lo tanto, puede ser útil imprimir el progreso en la consola de R. La sentencia if de abajo puede colocarse dentro de las operaciones del bucle para imprimir cada 100 números. Sólo tiene que ajustarla para que i sea el "elemento" de su bucle.

## ****purrr**** y listas

Otro enfoque de las operaciones iterativas es el paquete **purrr** - es el enfoque **tidyverse** de la iteración.

Si tiene que realizar la misma tarea varias veces, probablemente merezca la pena crear una solución generalizada que pueda utilizar en muchas entradas. Por ejemplo, producir gráficos para múltiples jurisdicciones, o importar y combinar muchos archivos.

También hay algunas otras ventajas de **purrr** - puedes usarlo con tuberías %>%, maneja los errores mejor que los bucles for normales, ¡y la sintaxis es bastante limpia y simple! Si está utilizando un bucle for, probablemente pueda hacerlo de forma más clara y sucinta con **purrr**!

Ten en cuenta que **purrr** es una herramienta de programación funcional. Es decir, las operaciones que se van a aplicar de forma iterativa están envueltas en funciones. Consulta la página [Escribir funciones](#writing-functions-1) para aprender a escribir tus propias funciones.

**purrr** también se basa casi por completo en listas y vectores, así que piensa en ello como si aplicara una función a cada elemento de esa lista/vector.

### Cargar paquetes

**purrr** forma parte del **tidyverse**, por lo que no es necesario instalar/cargar un paquete aparte.

### mapa()

Una de las funciones principales **de purrr** es map(), que "mapea" (aplica) una función a cada elemento de entrada de una lista/vector que has proporcionado.

La sintaxis básica es map(.x = SECUENCIA, .f = FUNCIÓN, OTROS ARGUMENTOS). Con un poco más de detalle:

* .x = son las entradas sobre las que se aplicará iterativamente la función .f - por ejemplo, un vector de nombres de jurisdicciones, columnas de un dataframe o una lista de dataframes
* .f = es la función a aplicar a cada elemento de la entrada .x - puede ser una función como print() que ya existe, o una función personalizada que tu definas. La función se suele escribir después de una tilde ~ (detalles más abajo).

Algunas notas más sobre la sintaxis:

* Si la función no necesita especificar más argumentos, puede escribirse sin paréntesis y sin tilde (por ejemplo, .f = media). Para proporcionar argumentos que tendrán el mismo valor en cada iteración, proporciónelos dentro de map() pero fuera del argumento .f =, como el na.rm = T en map(.x = mi\_lista, .f = media, na.rm=T).
* Puedes utilizar .x (o simplemente . ) dentro de la función .f = como marcador de posición para el valor .x de esa iteración
* Utiliza la sintaxis con tilde (~) para tener un mayor control sobre la función - escriba la función de forma normal con paréntesis, como por ejemplo: map(.x = mi\_lista, .f = ~media(., na.rm = T)). Utiliza esta sintaxis sobre todo si el valor de un argumento va a cambiar en cada iteración, o si es el propio valor .x (véanse los ejemplos siguientes)

La **salida de usar map() es una lista** - una lista es un tipo de objeto como un vector pero cuyos elementos pueden ser de diferentes tipos. Por lo tanto, una lista producida por map() podría contener muchos dataframes, o muchos vectores, muchos valores individuales, ¡o incluso muchas listas! Existen versiones alternativas de map() que se explican a continuación y que producen otros tipos de salidas (por ejemplo, map\_dfr() para producir un dataframe, map\_chr() para producir vectores de caracteres y map\_dbl() para producir vectores numéricos).

#### Ejemplo: importar y combinar hojas de Excel

**Hagamos una demostración con una tarea epidemiológica común:** - Quiere importar un libro de Excel con datos de casos, pero los datos están divididos en diferentes hojas con nombre en el libro. ¿Cómo puede importar y combinar eficazmente las hojas en un dataframe?

Supongamos que nos envían el siguiente libro de Excel. Cada hoja contiene casos de un determinado hospital.

Este es un enfoque que utiliza map():

1. mapear() la función import() para que se ejecute para cada hoja de Excel
2. Combinar los dataframes importados en uno solo utilizando bind\_rows()
3. A lo largo del proceso, conserva el nombre original de la hoja para cada fila, almacenando esta información en una nueva columna en el dataframe final

En primer lugar, tenemos que extraer los nombres de las hojas y guardarlos. Proporcionamos la ruta del archivo de Excel a la función excel\_sheets() del paquete **readxl**, que extrae los nombres de las hojas. Los almacenamos en un vector de caracteres llamado sheet\_names.

Aquí están los nombres:

Ahora que tenemos este vector de nombres, map() puede proporcionarlos uno a uno a la función import(). En este ejemplo, los nombres\_de\_hoja son .x e import() es la función .f.

Recuerde de la página de [importación y exportación](#import-and-export) que cuando se utiliza en libros de Excel, import() puede aceptar el argumento which = especificando la hoja a importar. Dentro de la función .f import(), proporcionamos which = .x, cuyo valor cambiará con cada iteración a través del vector nombres\_de\_hoja - primero "Hospital Central", luego "Hospital Militar", etc.

Hay que tener en cuenta que, como hemos utilizado map(), los datos de cada hoja de Excel se guardarán como un dataframe separado dentro de una lista. Queremos que cada uno de estos elementos de la lista (dataframes) tenga un nombre, así que antes de pasar sheet\_names a map() lo pasamos a través de set\_names() de **purrr**, lo que asegura que cada elemento de la lista obtenga el nombre apropiado.

Guardamos la lista de salida como combinada.

Cuando inspeccionamos la salida, vemos que los datos de cada hoja de Excel se guardan en la lista con un nombre. Esto es bueno, pero no hemos terminado.

Por último, utilizamos la función bind\_rows() (de **dplyr**) que acepta la lista de dataframes de estructura similar y los combina en un dataframe. Para crear una nueva columna a partir de los nombres de los elementos de la lista, utilizamos el argumento .id = y le proporcionamos el nombre deseado para la nueva columna.

A continuación se muestra toda la secuencia de comandos:

Y ahora tenemos un dataframe con una columna que contiene la hoja de origen!

Hay variaciones de map() que debe conocer. Por ejemplo, map\_dfr() devuelve un dataframe, no una lista. Por lo tanto, podríamos haberla utilizado para la tarea anterior y no haber tenido que enlazar filas. Pero entonces no habríamos podido capturar de qué hoja (hospital) procedía cada caso.

Otras variaciones son map\_chr(), map\_dbl(). Estas funciones son muy útiles por dos razones. En primer lugar, convierten automáticamente la salida de una función iterativa en un vector (no en una lista). En segundo lugar, pueden controlar explícitamente el tipo en la que vuelven los datos - te aseguras de que tus datos vuelven como un vector de caracteres con map\_chr(), o vector numérico con map\_dbl(). Volveremos a esto más adelante en la sección.

Las funciones map\_at() y map\_if() también son muy útiles para la iteración - ¡permiten especificar en qué elementos de una lista se debe iterar! Funcionan simplemente aplicando un vector de índices/nombres (en el caso de map\_at()) o una prueba lógica (en el caso de map\_if()).

Utilicemos un ejemplo en el que no queremos leer la primera hoja de datos del hospital. Usamos map\_at() en lugar de map(), y especificamos el argumento .at = a c(-1) que significa no usar el primer elemento de .x. Alternativamente, puede proporcionar un vector de números positivos, o nombres, a .at = para especificar qué elementos usar.

Ten en cuenta que el nombre de la primera hoja seguirá apareciendo como un elemento de la lista de salida, pero es sólo un nombre de un solo carácter (no un dataframe). Tendrá que eliminar este elemento antes de vincular las filas. Veremos cómo eliminar y modificar los elementos de la lista en una sección posterior.

### Dividir los datos y exportar

A continuación, damos un ejemplo de cómo dividir unos datos en partes y luego utilizar la iteración map() para exportar cada parte como una hoja de Excel separada, o como un archivo CSV separado.

#### Dividir los datos

Digamos que tenemos linelist de casos completa como un dataframe, y ahora queremos crear un listado separado para cada hospital y exportar cada una como un archivo CSV separado. A continuación, hacemos los siguientes pasos:

Utiliza group\_split() (de **dplyr**) para dividir el dataframe del listado por valores únicos en la columna hospital. La salida es una lista que contiene un dataframe por subconjunto de hospitales.

Podemos ejecutar View(linelist\_split) y ver que esta lista contiene 6 dataframes ("tibbles"), cada uno de los cuales representa los casos de un hospital.

Sin embargo, Ten en cuenta que los dataframes de la lista no tienen nombres por defecto. Queremos que cada uno de ellos tenga un nombre, y luego utilizar ese nombre al guardar el archivo CSV.

Un enfoque para extraer los nombres es utilizar pull() (de **dplyr**) para extraer la columna hospital de cada dataframe de la lista. Luego, para estar seguros, convertimos los valores a caracteres y luego usamos unique() para obtener el nombre de ese dataframe en particular. Todos estos pasos se aplican a cada dataframe mediante map().

Ahora podemos ver que cada uno de los elementos de la lista tiene un nombre. Se puede acceder a estos nombres mediante names(linelist\_split).

##### Más de una columna de group\_split()

Si desea dividir linelist por más de una columna de agrupación, por ejemplo, para producir un listado de subconjuntos por la intersección de hospital Y género, necesitará un enfoque diferente para nombrar los elementos de la lista. Esto implica recoger las "claves de grupo" únicas utilizando group\_keys() de **dplyr** - se devuelven como un dataframe. Luego puede combinar las claves de grupo en valores con unite() como se muestra a continuación, y asignar estos nombres conglomerados a linelist\_split.

Ahora combinamos las agrupaciones juntas, separadas por guiones, y las asignamos como los nombres de los elementos de la lista en linelist\_split. Esto requiere algunas líneas adicionales, ya que sustituimos NA por "Missing", utilizamos unite() de **dplyr** para combinar los valores de las columnas juntos (separados por guiones), y luego los convertimos en un vector sin nombre para poder utilizarlos como nombres de linelist\_split.

#### Exportar como hojas de Excel

Para exportar las listas de líneas del hospital como un libro de Excel con un listado por hoja, podemos simplemente proporcionar la lista con nombre linelist\_split a la función write\_xlsx() del paquete **writexl**. Esto tiene la capacidad de guardar un libro de Excel con múltiples hojas. Los nombres de los elementos de la lista se aplican automáticamente como los nombres de las hojas.

Ahora puede abrir el archivo de Excel y ver que cada hospital tiene su propia hoja.

#### Exportar como archivos CSV

Es un comando un poco más complejo, pero también puede exportar cada lista de líneas específica del hospital como un archivo CSV separado, con un nombre de archivo específico para el hospital.

De nuevo utilizamos map(): tomamos el vector de nombres de elementos de la lista (mostrado arriba) y utilizamos map() para iterar a través de ellos, aplicando export() (del paquete **rio**, véase la página [Importar y exportar](#import-and-export)) en el dataframe de la lista linelist\_split que tiene ese nombre. También utilizamos el nombre para crear un nombre de archivo único. Así es como funciona:

* Comenzamos con el vector de nombres de caracteres, pasado a map() como .x
* La función .f es export() , que requiere un dataframe y una ruta de archivo para escribir en
* La entrada .x (el nombre del hospital) se utiliza dentro de .f para extraer/indexar ese elemento específico de la lista linelist\_split. Esto hace que sólo se proporcione un dataframe a la vez a export().
* Por ejemplo, cuando map() itera por "Hospital Militar", entonces linelist\_split[[.x]] es en realidad linelist\_split[["Hospital Militar"]], devolviendo así el segundo elemento de linelist\_split - que son todos los casos del Hospital Militar.
* La ruta del archivo proporcionada a export() es dinámica mediante el uso de str\_glue() (ver página de [caracteres y cadenas](#characters-and-strings)):
  + here() se utiliza para obtener la base de la ruta del archivo y especificar la carpeta "data" (nótese las comillas simples para no interrumpir las comillas dobles de str\_glue())
* A continuación, una barra /, y luego de nuevo el .x que imprime el nombre actual del hospital para que el archivo sea identificable
* Por último, la extensión ".csv" que export() utiliza para crear un archivo CSV

Ahora puede ver que cada archivo se guarda en la carpeta "data" del proyecto R "Epi\_R\_handbook".

### Funciones personalizadas

Puedes crear su propia función para proporcionar a map().

Digamos que queremos crear curvas epidémicas para los casos de cada hospital. Para hacer esto usando **purrr**, nuestra función .f puede ser ggplot() y las extensiones con + como de costumbre. Como la salida de map() es siempre una lista, los gráficos se almacenan en una lista. Como son gráficos, pueden ser extraídas y trazadas con la función ggarrange() del paquete **ggpubr** ([documentación](https://rpkgs.datanovia.com/ggpubr/reference/ggarrange.html)).

Si este código de map() parece demasiado desordenado, se puede conseguir el mismo resultado guardando el comando específico de ggplot() como una función personalizada definida por el usuario, por ejemplo podemos llamarla make\_epicurve()). Esta función se utiliza entonces dentro de la función map(). .x se sustituirá iterativamente por el nombre del hospital, y se utilizará como hosp\_name en la función make\_epicurve(). Véase la página sobre [Funciones de escritura](#writing-functions-1).

### Asignación de una función a través de las columnas

Otro caso de uso común es asignar una función a varias columnas. A continuación, mapeamos() la función t.test() a través de columnas numéricas en linelist del dataframe, comparando los valores numéricos por género.

Recuerde de la página sobre [Pruebas estadísticas simples](#simple-statistical-tests) que t.test() puede tomar entradas en un formato de fórmula, como t.test(columna numérica ~ columna binaria). En este ejemplo, hacemos lo siguiente:

* Las columnas numéricas de interés se seleccionan del listado - éstas se convierten en las entradas .x de map()
* La función t.test() se suministra como la función .f, que se aplica a cada columna numérica
* Dentro del paréntesis de t.test():
  + el primer ~ precede al .f que map() iterará sobre el .x
  + el .x representa la columna actual que se suministra a la función t.test()
  + el segundo ~ es parte de la ecuación de la prueba t descrita anteriormente
  + la función t.test() espera una columna binaria en el lado derecho de la ecuación. Suministramos el vector linelist$gender de forma independiente y estática (Ten en cuenta que no se incluye en select()).

map() devuelve una lista, por lo que la salida es una lista de resultados de la prueba t, un elemento de la lista por cada columna numérica analizada.

Este es el aspecto de la lista t.test\_results cuando se abre (se ve) en RStudio. Hemos resaltado las partes que son importantes para los ejemplos de esta página.

* Puedes ver en la parte superior que toda la lista se llama t.test\_results y tiene cinco elementos. Esos cinco elementos se denominan age, wt\_km, ht\_cm, ct\_blood, temp después de cada variable que se utilizó en una prueba t con el género del listado.
* Cada uno de esos cinco elementos son a su vez listas, con elementos dentro de ellas como p.value y conf.int. Algunos de estos elementos, como p.value, son números individuales, mientras que otros, como conf.int, constan de dos o más elementos (media en el grupo f y media en el grupo m).

Nota: Recuerde que si desea aplicar una función sólo a determinadas columnas de un dataframe, también puede utilizar simplemente mutate() y across(), como se explica en la página [Limpieza de datos yfunciones básicas](#cleaning-data-and-core-functions). A continuación se muestra un ejemplo de aplicación de as.character() sólo a las columnas "edad". Observe la colocación de los paréntesis y las comas.

### Extracto de las listas

Como map() produce una salida del tipo List, dedicaremos algún tiempo a discutir cómo extraer datos de las listas utilizando las funciones **purrr** que las acompañan. Para demostrarlo, utilizaremos la lista t.test\_results de la sección anterior. Esta es una lista de 5 listas - cada una de las 5 listas contiene los resultados de una prueba t entre una columna del dataframe del listado y su columna binaria género. Consulte la imagen de la sección anterior para ver la estructura de la lista.

#### Nombres de elementos

Para extraer los nombres de los elementos en sí, basta con utilizar names() de la **base** R. En este caso, utilizamos names() en t.test\_results para devolver los nombres de cada sublista, que son los nombres de las 5 variables a las que se les realizaron pruebas t.

#### Elementos por nombre o cargo

Para extraer los elementos de la lista por su nombre o por su posición se pueden utilizar paréntesis [[ ]] como se describe en la página de [fundamentos de R](#r-basics). A continuación, utilizamos corchetes dobles para indexar la lista t.tests\_results y mostrar el primer elemento, que son los resultados de la prueba t sobre la edad.

Sin embargo, a continuación demostraremos el uso de las sencillas y flexibles funciones **purrr** map() y pluck() para lograr los mismos resultados.

#### desplumar()

pluck() extrae elementos por nombre o por posición. Por ejemplo, para extraer los resultados de la prueba t para la edad, puede utilizar pluck() así:

Indexe niveles más profundos especificando los niveles adicionales con comas. A continuación se extrae el elemento denominado "p.value" de la lista age dentro de la lista t.test\_results. También puede utilizar números en lugar de nombres de caracteres.

Puedes extraer estos elementos internos de todos los elementos de primer nivel utilizando map() para ejecutar la función pluck() en cada elemento de primer nivel. Por ejemplo, el siguiente código extrae los elementos "p.value" de todas las listas dentro de t.test\_results. La lista de resultados de la prueba t es el .x que se itera, pluck() es la función .f que se itera, y el valor "p-value" se proporciona a la función.

Como otra alternativa, map() ofrece una forma abreviada en la que puede escribir el nombre del elemento entre comillas, y lo extraerá. Si utiliza map() la salida será una lista, mientras que si utiliza map\_chr() será un vector de caracteres con nombre y si utiliza map\_dbl() será un vector numérico con nombre.

Puedes leer más sobre pluck() en su [documentación](https://purrr.tidyverse.org/reference/pluck.html) **purrr**. Tiene una función hermana chuck() que devolverá un error en lugar de NULL si un elemento no existe.

### Convertir una lista en un dataframe

Este es un tema complejo - vea la sección de Recursos para tutoriales más completos. Sin embargo, vamos a demostrar la conversión de la lista de resultados de la prueba t en un dataframe. Crearemos un dataframe con columnas para la variable, su valor p y las medias de los dos grupos (hombres y mujeres).

Estos son algunos de los nuevos enfoques y funciones que se utilizarán:

* La función tibble() se utilizará para crear un tibble (como un dataframe)
  + Rodeamos la función tibble() con corchetes { } para evitar que todo el t.test\_results se almacene como la primera columna tibble
* Dentro de tibble(), cada columna se crea explícitamente, de forma similar a la sintaxis de mutate():
  + El . representa t.test\_results
  + Para crear una columna con los nombres de las variables de la prueba t (los nombres de cada elemento de la lista) utilizamos names() como se ha descrito anteriormente
  + Para crear una columna con los valores p utilizamos map\_dbl() como se ha descrito anteriormente para extraer los elementos p.value y convertirlos en un vector numérico

Pero ahora vamos a añadir columnas que contengan las medias de cada grupo (hombres y mujeres).

Tendríamos que extraer la estimación del elemento, pero ésta contiene en realidad dos elementos en su interior (media en el grupo f y media en el grupo m). Por lo tanto, no se puede simplificar en un vector con map\_chr() o map\_dbl(). En su lugar, utilizamos map(), que usado dentro de tibble() creará una columna de tipo lista dentro del tibble! Sí, esto es posible!

Una vez que tengas esta columna de lista, hay varias funciones **de tidyr** (parte de **tidyverse**) que te ayudan a "rectangular" o "desanidar" estas columnas de "lista anidada". Lea más sobre ellas aquí, o ejecutando vignette("rectangle"). En resumen:

* unnest\_wider() - da a cada elemento de una lista-columna su propia columna
* unnest\_longer() - da a cada elemento de una lista-columna su propia fila
* hoist() - actúa como unnest\_wider() pero se especifica qué elementos se van a anular

A continuación, pasamos el tibble a unnest\_wider() especificando la columna de medias del tibble (que es una lista anidada). El resultado es que las medias se sustituyen por dos nuevas columnas, cada una de las cuales refleja los dos elementos que había antes en cada celda de medias.

### Listas de descarte, conservación y compactación

Dado que el trabajo con **purrr** implica a menudo listas, exploraremos brevemente algunas funciones de **purrr** para modificar listas. Consulte la sección de Recursos para ver tutoriales más completos sobre las funciones de **purrr**.

* list\_modify() tiene muchos usos, uno de los cuales puede ser eliminar un elemento de la lista
* keep() conserva los elementos especificados a .p =, o cuando una función suministrada a .p = evalúa a TRUE
* discard() elimina los elementos especificados a .p, o cuando una función suministrada a .p = evalúa a TRUE
* compact() elimina todos los elementos vacíos

Aquí hay algunos ejemplos que utilizan la lista combinada creada en la sección anterior sobre el [uso de map() para importar y combinar múltiples archivos](#iter_combined) (contiene 6 dataframes de lista de líneas de casos):

Los elementos pueden ser eliminados por su nombre con list\_modify() y estableciendo el nombre igual a NULL.

También puede eliminar elementos por criterio, proporcionando una ecuación "predicada" a .p = (una ecuación que evalúa a TRUE o FALSE). Coloque una tilde ~ antes de la función y utiliza .x para representar el elemento de la lista. Utilizando keep() se conservarán los elementos de la lista que se evalúen como TRUE. A la inversa, si se utiliza discard() se eliminarán los elementos de la lista que se evalúen como TRUE.

En el siguiente ejemplo, los elementos de la lista se descartan si su tipo no son dataframes.

Su función de predicado también puede hacer referencia a elementos/columnas dentro de cada elemento de la lista. Por ejemplo, a continuación, se descartan los elementos de la lista cuya media de la columna ct\_sangre sea superior a 25.

Este comando eliminaría todos los elementos vacíos de la lista:

### pmap()

ESTA SECCIÓN ESTÁ EN CONSTRUCCIÓN

## Aplicar funciones

La familia de funciones "apply" es una alternativa **básica de** R a **purrr** para operaciones iterativas. Puedes leer más sobre ellas [aquí](https://www.datacamp.com/community/tutorials/r-tutorial-apply-family).

## Recursos

[bucles for con Data Carpentry](https://datacarpentry.org/semester-biology/materials/for-loops-R/)

La [página de R for Data Science sobre la iteración](https://r4ds.had.co.nz/iteration.html" \l "iteration)

[Viñeta sobre escritura/lectura de archivos Excel](https://martinctc.github.io/blog/vignette-write-and-read-multiple-excel-files-with-purrr/)

Un [tutorial de](https://jennybc.github.io/purrr-tutorial/index.html) purrr por jennybc

Otro [tutorial de](http://www.rebeccabarter.com/blog/2019-08-19_purrr/) purrr por Rebecca Barter

Un [tutorial](http://zevross.com/blog/2019/06/11/the-power-of-three-purrr-poseful-iteration-in-r-with-map-pmap-and-imap/) de purrr sobre map, pmap e imap

[hoja de trucos purrr](https://raw.githubusercontent.com/rstudio/cheatsheets/master/pngs/thumbnails/purrr-cheatsheet-thumbs.png)

[consejos y trucos de purrr](https://www.hvitfeldt.me/blog/purrr-tips-and-tricks/)

[guardar y descartar](https://hookedondata.org/going-off-the-map/" \l "keep-and-discard)

# Análisis

# **#Tabla**s descriptivas

{#descriptive-tables}

Esta página demuestra el uso de **janitor**, **dplyr**, **gtsummary**, **rstatix** y **base** R para resumir datos y crear tablas con estadísticas descriptivas.

En esta página se explica cómo crear\* las tablas subyacentes, mientras que en la página [Tablas para la presentación](#tables-for-presentation) se explica cómo darles un buen formato e imprimirlas.\*

Cada uno de estos paquetes tiene ventajas y desventajas en cuanto a la simplicidad del código, la accesibilidad de los resultados y la calidad de los resultados impresos. Utiliza esta página para decidir qué enfoque se ajusta a su situación.

Tienes varias opciones para producir tablas de resumen de tabulación y tabulación cruzada. Algunos de los factores a tener en cuenta son la simplicidad del código, la posibilidad de personalización, la salida deseada (impresa en la consola de R, como dataframe, o como una imagen .png/.jpeg/.html "bonita"), y la facilidad de posprocesamiento. Ten en cuenta los siguientes puntos a la hora de elegir la herramienta para su situación.

* Utiliza tabyl() de **janitor** para producir y "adornar" tabulaciones y tabulaciones cruzadas
* Utiliza get\_summary\_stats() de **rstatix** para generar fácilmente dataframes de estadísticas de resumen numérico para múltiples columnas y/o grupos
* Utiliza summarise() y count() de **dplyr** para obtener estadísticas más complejas, ordenar las salidas de los dataframes o preparar los datos para ggplot()
* Utiliza tbl\_summary() de **gtsummary** para producir tablas detalladas listas para su publicación
* Utiliza table() de R **base** si no tiene acceso a los paquetes anteriores

## Preparación

### Cargar paquetes

Este trozo de código muestra la carga de los paquetes necesarios para los análisis. En este manual destacamos p\_load() de **pacman**, que instala el paquete si es necesario y lo carga para su uso. También puede cargar los paquetes instalados con library() de R **base.** Consulte la página sobre [los fundamentos de R](#r-basics) para obtener más información sobre los paquetes de R.

### Importar datos

Importamos los datos de casos de una epidemia de ébola simulada. Si quiere seguir el proceso, [clica para descargar linelist "limpia"](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/case_linelists/linelist_cleaned.rds) (como archivo .rds). Importa tus datos con la función import() del paquete **rio** (acepta muchos tipos de archivos como .xlsx, .rds, .csv - vea la página de [importación y exportación](#import-and-export) para más detalles).

A continuación se muestran las primeras 50 filas del listado.

## Buscar datos

### paquete ****skimr****

Utilizando el paquete skimr, puedes obtener una visión detallada y estéticamente agradable de cada una de las variables de tu conjunto de datos. Lee más sobre **skimr** en su [página de github](https://github.com/ropensci/skimr).

A continuación, se aplica la función skim() a todo el dataframe del listado. Se produce una visión general del dataframe y un resumen de cada columna (por tipo).

También puede utilizar la función summary(), de la **base** R, para obtener información sobre unos datos completo, pero esta salida puede ser más difícil de leer que utilizando **skimr**. Por lo tanto, la salida no se muestra a continuación, para conservar el espacio de la página.

### Estadísticas resumidas

Puedes utilizar las funciones **básicas de** R para devolver estadísticas de resumen sobre una columna numérica. Puedes devolver la mayoría de las estadísticas de resumen útiles para una columna numérica utilizando summary(), como se indica a continuación. Ten en cuenta que el nombre del dataframe también debe especificarse como se muestra a continuación.

Puedes acceder y guardar una parte específica de la misma con los corchetes de índice [ ]:

Puedes devolver estadísticas individuales con funciones básicas de R como max(), min(), median(), mean(), quantile(), sd() y range(). Consulte la página de [fundamentos de R](#r-basics) para obtener una lista completa.

**PRECAUCIÓN:** Si tus datos contienen valores faltantes, R quiere que lo sepas y por ello devolverá NA a menos que se especifique en las funciones matemáticas anteriores que quieres que R ignore los valores faltantes, mediante el argumento na.rm = TRUE.

Puedes utilizar la función get\_summary\_stats() de **rstatix** para devolver las estadísticas de resumen en un formato de dataframe. Esto puede ser útil para realizar operaciones posteriores o trazar los números. Consulte la página [Pruebas estadísticas simples](#simple-statistical-tests) para obtener más detalles sobre el paquete rstatix y sus funciones.

## paquete de ****conserjería****

Los paquetes **janitor** ofrecen la función tabyl() para producir tabulaciones y tabulaciones cruzadas, que pueden ser "adornadas" o modificadas con funciones de ayuda para mostrar porcentajes, proporciones, recuentos, etc.

A continuación, canalizamos el dataframe del listado a las funciones de **limpieza** e imprimimos el resultado. Si lo desea, también puede guardar las tablas resultantes con el operador de asignación <-.

### Tabyl simple

El uso por defecto de tabyl() en una columna específica produce los valores únicos, los recuentos y los "porcentajes" por columna (en realidad proporciones). Las proporciones pueden tener muchos dígitos. Puedes ajustar el número de decimales con adorn\_rounding() como se describe a continuación.

Como puede ver arriba, si hay valores que faltan se muestran en una fila etiquetada como <NA>. Puedes suprimirlos con show\_na = FALSE. Si no hay valores faltantes, esta fila no aparecerá. Si hay valores faltantes, todas las proporciones se dan como crudas (el denominador incluye los recuentos NA) y "válidas" (el denominador excluye los recuentos NA).

Si la columna es de tipo factor y sólo hay ciertos niveles en sus datos, todos los niveles seguirán apareciendo en la tabla. Puedes suprimir esta característica especificando show\_missing\_levels = FALSE. Lea más en la página de [Factores](#factors).

### Tabulación cruzada

Los recuentos de tabulación cruzada se consiguen añadiendo una o más columnas adicionales dentro de tabyl(). Ten en cuenta que ahora sólo se devuelven los recuentos - las proporciones y los porcentajes se pueden añadir con los pasos adicionales que se muestran a continuación.

### "Adornando" el tabyl

Utiliza las funciones de "adorno" **del conserje** para añadir totales o convertir a proporciones, porcentajes, o ajustar la visualización de otro modo. A menudo, canalizará el tabyl a través de varias de estas funciones.

| **Función** | **Resultado** |
| --- | --- |
| adornar\_totales() | Añade los totales (donde = "fila", "col", o "ambos"). Establecer nombre = para "Total". |
| adornar\_porcentajes() | Convertir los recuentos en proporciones, con denominador = "fila", "col" o "todos" |
| adorn\_pct\_formatting() | Convierte las proporciones en porcentajes. Especifique los dígitos =. Elimine el símbolo "%" con affix\_sign = FALSE. |
| adornar\_redondear() | Para redondear proporciones a dígitos = lugares. Para redondear porcentajes utiliza adorn\_pct\_formatting() con dígitos =. |
| adorn\_ns() | Añadir recuentos a una tabla de proporciones o porcentajes. Indique la posición = "atrás" para mostrar los recuentos entre paréntesis, o "adelante" para poner los porcentajes entre paréntesis. |
| adornar\_título() | Añade una cadena mediante los argumentos row\_name = y/o col\_name = |

Sea consciente del orden en que aplica las funciones anteriores. A continuación, algunos ejemplos.

Una simple tabla unidireccional con porcentajes en lugar de las proporciones por defecto.

Una tabulación cruzada con un total de filas y porcentajes de filas.

Una tabulación cruzada ajustada para que aparezcan tanto los recuentos como los porcentajes.

### Impresión del tabyl

Por defecto, el tabyl imprimirá en bruto en la consola de R.

Alternativamente, puede pasar el tabyl a **flextable** o un paquete similar para imprimirlo como una imagen "bonita" en el visor de RStudio, que podría exportarse como .png, .jpeg, .html, etc. Esto se discute en la página [Tablas de presentación](#tables-for-presentation). Ten en cuenta que si imprime de esta manera y utiliza adorn\_titles(), debe especificar placement = "combined".

### Uso en otras mesas

Puedes utilizar las funciones adorn\_\*() de **janitor** en otras tablas, como las creadas por summarise() y count() de **dplyr**, o table() de R **base**. Por ejemplo:

### Guardar el tabyl

Si convierte la tabla en una imagen "bonita" con un paquete como **flextable**, puede guardarla con funciones de ese paquete - como save\_as\_html(), save\_as\_word(), save\_as\_ppt(), y save\_as\_image() de **flextable** (como se discute más ampliamente en la página [Tablas para la presentación](#tables-for-presentation)). A continuación, la tabla se guarda como un documento de Word, en el que se puede seguir editando a mano.

### Estadísticas

Puedes aplicar pruebas estadísticas a las tabillas, como chisq.test() o fisher.test() del paquete **stats**, como se muestra a continuación. Ten en cuenta que los valores faltantes no están permitidos, por lo que se excluyen de la tabulación con show\_na = FALSE.

Consulte la página sobre [Tests estadísticos sencillos](#simple-statistical-tests) para obtener más código y consejos sobre estadística.

### Otros consejos

* Incluya el argumento na.rm = TRUE para excluir los valores faltantes de cualquiera de los cálculos anteriores.
* Si aplica cualquier función de ayuda adorn\_\*() a tablas no creadas por tabyl(), puede especificar una(s) columna(s) particular(es) para aplicarlas como adorn\_percentage(,,,c(cases,deaths)) (especifíquelos en el cuarto argumento sin nombre). La sintaxis no es sencilla. Considere la posibilidad de utilizar summarise() en su lugar.
* Puedes leer más detalles en la [página del conserje](https://cran.r-project.org/web/packages/janitor/vignettes/janitor.html) y en esta [viñeta del tabyl](https://cran.r-project.org/web/packages/janitor/vignettes/tabyls.html).

## paquete ****dplyr****

**dplyr** forma parte de los paquetes **tidyverse** y es una herramienta de gestión de datos muy común. La creación de tablas con las funciones de **dplyr** summarise() y count() es un enfoque útil para calcular estadísticas de resumen, resumir por grupos o pasar tablas a ggplot().

summarise() crea un nuevo dataframe de resumen. Si los datos no están agrupados, devolverá un dataframe de una fila con las estadísticas de resumen especificadas de todo el dataframe. Si los datos están agrupados, el nuevo dataframe tendrá una fila por grupo (véase la página [Agrupar datos](#grouping-data)).

Dentro del paréntesis de summarise(), se proporcionan los nombres de cada nueva columna de resumen, seguidos de un signo de igualdad y de una función estadística a aplicar.

**SUGERENCIA:** La función summarise funciona tanto con la ortografía británica como con la estadounidense (summarise() y summarize()).

### Obtener recuentos

La función más sencilla de aplicar dentro de summarise() es n(). Deje los paréntesis vacíos para contar el número de filas.

Esto se vuelve más interesante si hemos agrupado los datos de antemano.

El comando anterior se puede acortar utilizando la función count() en su lugar. count() hace lo siguiente:

1. Agrupa los datos por las columnas que se le proporcionan
2. Los resume con n() (creando la columna n)
3. Desagrupa los datos

Puedes cambiar el nombre de la columna de recuentos de la n por defecto a otra cosa especificando a nombre =.

Los recuentos tabulados de dos o más columnas de agrupación se siguen devolviendo en formato "largo", con los recuentos en la columna n. Consulte la página sobre [Pivoteo de datos](#pivoting-data) para conocer los formatos de datos "largos" y "anchos".

### Mostrar todos los niveles

Si está tabulando una columna de de tipo factor, puede asegurarse de que se muestren todos los niveles (no sólo los niveles con valores en los datos) añadiendo .drop = FALSE en el comando summarise() o count().

Esta técnica es útil para estandarizar sus tablas/trazados. Por ejemplo, si está creando cifras para varios subgrupos, o creando repetidamente la cifra para informes de rutina. En cada una de estas circunstancias, la presencia de valores en los datos puede fluctuar, pero puede definir niveles que permanezcan constantes.

Para más información, consulte la página sobre [los factores](#factors).

### Proporciones

Las proporciones pueden añadirse pasando la tabla por mutate() para crear una nueva columna. Defina la nueva columna como la columna de recuentos (n por defecto) dividida por la suma() de la columna de recuentos (esto devolverá una proporción).

Ten en cuenta que en este caso, sum() en el comando mutate() devolverá la suma de toda la columna n para utilizarla como denominador de la proporción. Como se explica [en la página Agrupar datos](#group_summarise), si sum() se utiliza en datos agrupados (por ejemplo, si el comando mutate() siguió inmediatamente a un comando group\_by()), devolverá sumas por grupo. Como se acaba de indicar, count() termina sus acciones desagrupando. Por lo tanto, en este escenario obtenemos proporciones de columnas completas.

Para mostrar fácilmente los porcentajes, puede envolver la proporción en la función percent() del paquete **scales** (Ten en cuenta que se convierte en tipo carácter).

A continuación se presenta un método para calcular las proporciones dentro de los grupos. Se basa en diferentes niveles de agrupación de datos que se aplican y eliminan selectivamente. En primer lugar, los datos se agrupan en función del resultado mediante group\_by(). A continuación, se aplica count(). Esta función agrupa además los datos por age\_cat y devuelve los recuentos para cada combinación de resultado-age-cat. Es importante destacar que, al finalizar su proceso, count() también desagrupa la agrupación age\_cat, por lo que la única agrupación de datos que queda es la agrupación original por resultado. Por lo tanto, el paso final del cálculo de las proporciones (denominador sum(n)) sigue estando agrupado por resultado.

### Trazando

Mostrar una tabla "larga" como la anterior con ggplot() es relativamente sencillo. Los datos están naturalmente en formato "largo", que es aceptado naturalmente por ggplot(). Vea más ejemplos en las páginas [ggplot basics](#ggplot-basics) y [ggplot tips](#ggplot-tips).

### Estadísticas resumidas

Una de las principales ventajas de **dplyr** y de summarise() es la capacidad de devolver resúmenes estadísticos más avanzados como la mediana(), la media(), el máximo(), el mínimo(), la sd() (desviación estándar) y los percentiles. También puede utilizar sum() para devolver el número de filas que cumplen ciertos criterios lógicos. Al igual que en el caso anterior, estas salidas pueden producirse para todo el conjunto de dataframes o por grupos.

La sintaxis es la misma: dentro de los paréntesis de summarise() se proporcionan los nombres de cada nueva columna de resumen, seguidos de un signo de igualdad y de una función estadística para aplicar. Dentro de la función estadística, indique la(s) columna(s) con la(s) que se va a operar y cualquier argumento relevante (por ejemplo, na.rm = TRUE para la mayoría de las funciones matemáticas).

También puede utilizar sum() para devolver el número de filas que cumplen un criterio lógico. La expresión que contiene se cuenta si se evalúa como TRUE. Por ejemplo:

* sum(edad\_años < 18, na.rm=T)
* sum(género == "masculino", na.rm=T)
* sum(respuesta %in% c("Probable", "Muy probable"))

A continuación, se resumen los datos del listado para describir los días de retraso desde el inicio de los síntomas hasta el ingreso en el hospital (columna days\_onset\_hosp), por hospital.

Algunos consejos:

* Utilizar sum() con una sentencia lógica para "contar" las filas que cumplen ciertos criterios (==)
* Ten en cuenta el uso de na.rm = TRUE dentro de funciones matemáticas como sum(), de lo contrario se devolverá NA si hay algún valor faltante
* Utiliza la función percent() del paquete **scales** para convertir fácilmente a porcentajes
  + Ajuste la precisión = a 0,1 o 0,01 para garantizar 1 o 2 decimales respectivamente
* Utilizar round() de **base** R para especificar los decimales
* Para calcular estas estadísticas en todo el set de datos, utiliza summarise() sin group\_by()
* Puedes crear columnas para los propósitos de cálculos posteriores (por ejemplo, denominadores) que eventualmente se eliminan de su dataframe con select().

### Estadísticas condicionales

Es posible que desee devolver estadísticas condicionales, por ejemplo, el máximo de filas que cumplen ciertos criterios. Esto se puede hacer sub-configurando la columna con corchetes [ ]. El ejemplo siguiente devuelve la temperatura máxima de los pacientes clasificados con o sin fiebre. Sin embargo, Ten en cuenta que puede ser más apropiado añadir otra columna al comando group\_by() y pivot\_wider() (como se demuestra [a continuación](#tbls_pivot_wider)).

### Pegado

La función str\_glue() de **stringr** es útil para combinar valores de varias columnas en una nueva columna. En este contexto, se suele utilizar después del comando summarise().

En la página [Caracteres y cadenas](#characters-and-strings), se discuten varias opciones para combinar columnas, incluyendo unite(), y paste0(). En este caso de uso, abogamos por str\_glue() porque es más flexible que unite() y tiene una sintaxis más sencilla que paste0().

A continuación, el dataframe de summary\_table (creado anteriormente) se muta de manera que las columnas delay\_mean y delay\_sd se combinan, se añade el formato de paréntesis a la nueva columna y se eliminan sus respectivas columnas antiguas.

Luego, para hacer la tabla más presentable, se añade una fila de totales con adorn\_totals() de **janitor** (que ignora las columnas no numéricas). Por último, utilizamos select() de **dplyr** para reordenar y renombrar los nombres de las columnas.

Ahora puede pasar a **flextable** e imprimir la tabla a Word, .png, .jpeg, .html, Powerpoint, RMarkdown, etc. (ver la página de [Tablas para la presentación](#tables-for-presentation)).

#### Percentiles

Los percentiles y cuantiles en **dplyr** merecen una mención especial. Para devolver los cuantiles, utiliza quantile() con los valores predeterminados o especifique el valor o los valores que desea con probs =.

Si desea devolver cuantiles por grupo, puede encontrar salidas largas y menos útiles si simplemente añade otra columna a group\_by(). Por lo tanto, pruebe este enfoque en su lugar: cree una columna para cada nivel de cuantil deseado.

Aunque **dplyr** summarise() ofrece ciertamente un control más fino, puede encontrar que todas las estadísticas de resumen que necesita pueden producirse con get\_summary\_stat() del paquete **rstatix**. Si se opera con datos agrupados, if devolverá 0%, 25%, 50%, 75% y 100%. Si se aplica a datos no agrupados, puede especificar los percentiles con probs = c(.05, .5, .75, .98).

### Resumir los datos agregados

Si comienza con datos agregados, al utilizar n() devuelve el número de filas, no la suma de los recuentos agregados. Para obtener sumas, utiliza sum() en la columna de recuentos de los datos.

Por ejemplo, digamos que se empieza con el dataframe de recuentos que se muestra a continuación, llamado linelist\_agg - muestra en formato "largo" los recuentos de casos por resultado y género.

A continuación creamos este dataframe de ejemplo de recuentos de casos del listado por resultado y sexo (se eliminan los valores faltantes para mayor claridad).

Para sumar los recuentos (en la columna n) por grupo, puede utilizar summarise() pero establecer la nueva columna igual a sum(n, na.rm=T). Para añadir un elemento condicional a la operación de suma, puede utilizar la sintaxis del subconjunto [ ] en la columna de recuentos.

### across() varias columnas

Puedes utilizar summarise() en varias columnas utilizando across(). Esto facilita la vida cuando se desea calcular las mismas estadísticas para muchas columnas. Coloque across() dentro de summarise() y especifique lo siguiente:

* .cols = como un vector de nombres de columnas c() o funciones de ayuda "tidyselect" (explicadas más adelante)
* .fns = la función a realizar (sin paréntesis) - puede proporcionar varias dentro de una lista()

A continuación, mean() se aplica a varias columnas numéricas. Se nombra explícitamente un vector de columnas a .cols = y se especifica una única función media (sin paréntesis) a .fns =. Cualquier argumento adicional para la función (por ejemplo, na.rm=TRUE) se proporciona después de .fns =, separado por una coma.

Puedes ser difícil conseguir el orden correcto de los paréntesis y las comas cuando se utiliza across(). Recuerde que dentro de across() debe incluir las columnas, las funciones y cualquier argumento extra necesario para las funciones.

Se pueden ejecutar varias funciones a la vez. A continuación se proporcionan las funciones mean y sd a .fns = dentro de una lista(). Tiene la oportunidad de proporcionar nombres de caracteres (por ejemplo, "mean" y "sd") que se añaden en los nuevos nombres de columna.

Aquí están esas funciones de ayuda "tidyselect" que puede proporcionar a .cols = para seleccionar columnas:

* everything() - todas las demás columnas no mencionadas
* last\_col() - la última columna
* where() - aplica una función a todas las columnas y selecciona las que son TRUE
* starts\_with() - coincide con un prefijo especificado. Ejemplo: starts\_with("date")
* ends\_with() - coincide con un sufijo especificado. Ejemplo: ends\_with("\_end")
* contains() - columnas que contienen una cadena de caracteres. Ejemplo: contains("tiempo")
* matches() - para aplicar una expresión regular (regex). Ejemplo: contains("[pt]al")
* num\_range() -
* any\_of() - coincide con el nombre de la columna. Es útil si el nombre puede no existir. Ejemplo: any\_of(date\_onset, date\_death, cardiac\_arrest)

Por ejemplo, para devolver la media de cada columna numérica utiliza where() y proporcione la función as.numeric() (sin paréntesis). Todo esto queda dentro del comando across().

### Pivote más amplio

Si prefiere su tabla en formato "ancho" puede transformarla utilizando la función **tidyr** pivot\_wider(). Es probable que tenga que renombrar las columnas con rename(). Para más información, consulte la página sobre [Pivoteo de datos](#pivoting-data).

El ejemplo siguiente comienza con la tabla "larga" age\_by\_outcome de la [sección de proporciones](#tbl_dplyr_prop). La creamos de nuevo y la imprimimos, para mayor claridad:

Para pivotar más ampliamente, creamos las nuevas columnas a partir de los valores de la columna existente age\_cat (estableciendo names\_from = age\_cat). También especificamos que los nuevos valores de la tabla provendrán de la columna existente n, con values\_from = n. Las columnas no mencionadas en nuestro comando de pivoteo (resultado) permanecerán sin cambios en el extremo izquierdo.

### Total de filas

Cuando summarise() opera con datos agrupados no produce automáticamente estadísticas "totales". A continuación, se presentan dos enfoques para añadir una fila de totales:

#### adornar\_totales() del ****conserje****

Si su tabla consiste sólo en recuentos o proporciones/porcentajes que pueden sumarse en un total, entonces puede añadir totales de suma usando adorn\_totals() de **janitor** como se describe en la sección anterior. Ten en cuenta que esta función sólo puede sumar las columnas numéricas - si desea calcular otras estadísticas de resumen total, vea el siguiente enfoque con **dplyr**.

A continuación, linelist se agrupa por género y se resume en una tabla que describe el número de casos con resultado conocido, los fallecidos y los recuperados. Al pasar la tabla por adorn\_totals() se añade una fila total en la parte inferior que refleja la suma de cada columna. Las funciones posteriores adorn\_\*() ajustan la visualización como se indica en el código.

#### resumir() en los datos "totales" y luego bind\_rows()

Si su tabla consta de estadísticas de resumen como la mediana(), la media(), etc., el enfoque adorn\_totals() mostrado anteriormente no será suficiente. En su lugar, para obtener los estadísticos de resumen de todo el set de datos debe calcularlos con un comando summarise() separado y luego vincular los resultados a la tabla de resumen agrupada original. Para hacer el enlace puede utilizar bind\_rows() de **dplyr** s descrito en la página de [unión de datos](#joining-data). A continuación se muestra un ejemplo:

Se puede hacer una tabla resumen de resultados por hospital con group\_by() y summarise() así:

Para obtener los totales, ejecuta el mismo comando summarise() pero agrupando los datos sólo por resultado (no por hospital), de la siguiente manera:

Podemos unir estos dos dataframes. Ten en cuenta que by\_hospital tiene 4 columnas, mientras que los totales tienen 3 columnas. Al utilizar bind\_rows(), las columnas se combinan por nombre, y cualquier espacio extra se rellena con NA (por ejemplo, los valores de la columna hospital para las dos nuevas filas de totales). Después de enlazar las filas, convertimos estos espacios vacíos en "Total" utilizando replace\_na() (véase la página de [limpieza de datos y funciones básicas](#cleaning-data-and-core-functions)).

Aquí está la nueva tabla con las filas "Total" en la parte inferior.

Esta tabla tiene un formato "largo", que puede ser lo que quieres. Opcionalmente, puedes pivotar esta tabla más ampliamente para hacerla más legible. Vea la sección sobre pivoteo más amplio arriba, y la página [Pivoteo de datos](#pivoting-data). También puede añadir más columnas, y organizarla de forma agradable. Este código está abajo.

Y luego puede imprimir esto muy bien como una imagen - abajo está la salida impresa con **flextable**. Puedes leer más en profundidad sobre este ejemplo y cómo lograr esta tabla "bonita" en la página [Tablas para la presentación](#tables-for-presentation).

## paquete ****gtsummary****

Si deseas imprimir tus estadísticas de resumen en un gráfico bonito y listo para su publicación, puede utilizar el paquete **gtsummary** y su función tbl\_summary(). El código puede parecer complejo al principio, pero los resultados se ven muy bien y se imprimen en su panel de RStudio Viewer como una imagen HTML. Lea una [viñeta aquí](http://www.danieldsjoberg.com/gtsummary/articles/tbl_summary.html).

También puede añadir los resultados de las pruebas estadísticas a las tablas de gtsummary. Este proceso se describe en la sección **gtsummary de la** página [Pruebas estadísticas simples](#stats_gt).

Para introducir tbl\_summary() mostraremos primero el comportamiento más básico, que realmente produce una tabla grande y bonita. Luego, examinaremos en detalle cómo hacer ajustes y tablas más a medida.

### Cuadro resumen

El comportamiento por defecto de tbl\_summary() es bastante increíble: toma las columnas que proporcionas y crea una tabla de resumen en un solo comando. La función imprime las estadísticas apropiadas para el tipo de columna: mediana y rango intercuartil (IQR) para las columnas numéricas, y recuentos (%) para las columnas categóricas. Los valores faltantes se convierten en "Desconocidos". Se añaden notas a pie de página para explicar las estadísticas, mientras que el N total se muestra en la parte superior.

### Ajustes

Ahora explicaremos cómo funciona la función y cómo hacer los ajustes. Los argumentos clave se detallan a continuación:

**by =**   
Puedes estratificar su tabla por una columna (por ejemplo, por resultado), creando una tabla de dos vías.

**statistic =Usa**   
una ecuación para especificar qué estadísticas mostrar y cómo mostrarlas. La ecuación tiene dos lados, separados por una tilde ~. En el lado derecho, entre comillas, está la visualización estadística deseada, y en el izquierdo están las columnas a las que se aplicará esa visualización.

* El lado derecho de la ecuación utiliza la sintaxis de str\_glue() de **stringr** (véase [Caracteres y cadenas](#characters-and-strings)), con la cadena de visualización deseada entre comillas y los propios estadísticos entre llaves. Puedes incluir estadísticas como "n" (para los recuentos), "N" (para el denominador), "media", "mediana", "sd", "max", "min", percentiles como "p##" como "p25", o porcentaje del total como "p". Consulte ?tbl\_summary para obtener más detalles.
* Para el lado izquierdo de la ecuación, puede especificar las columnas por su nombre (por ejemplo, edad o c(edad, sexo)) o utilizando ayudantes como all\_continuous(), all\_categorical(), contains(), starts\_with(), etc.

Un ejemplo sencillo de una ecuación estadística = podría ser como el siguiente, para imprimir sólo la media de la columna edad\_años:

Una ecuación un poco más compleja podría tener el aspecto de "({min}, {max})", incorporando los valores máximo y mínimo entre paréntesis y separados por una coma:

También puede diferenciar la sintaxis para columnas separadas o tipos de columnas. En el ejemplo más complejo de abajo, el valor proporcionado a statistc = es una **lista** que indica que para todas las columnas continuas la tabla debe imprimir la media con la desviación estándar entre paréntesis, mientras que para todas las columnas categóricas debe imprimir el n, el denominador y el porcentaje.

**dígitos =Ajustar**   
los dígitos y el redondeo. Opcionalmente, se puede especificar que sea sólo para columnas continuas (como a continuación).

**label =Ajustar**   
cómo debe mostrarse el nombre de la columna. Proporcione el nombre de la columna y su etiqueta deseada separados por una tilde. El valor por defecto es el nombre de la columna.

**missing\_text =Ajustar**   
cómo se muestran los valores faltantes. El valor por defecto es "Desconocido".

**type**   
=Se utiliza para ajustar cuántos niveles de la estadística se muestran. La sintaxis es similar a la de la estadística = en el sentido de que se proporciona una ecuación con columnas a la izquierda y un valor a la derecha. Dos escenarios comunes incluyen:

* type = all\_categorical() ~ "categorical" Fuerza a las columnas dicotómicas (por ejemplo, fiebre sí/no) a mostrar todos los niveles en lugar de sólo la fila "sí
* type = all\_continuous() ~ "continuous2" Permite estadísticas de varias líneas por variable, como se muestra en una sección posterior

En el siguiente ejemplo, cada uno de estos argumentos se utiliza para modificar la tabla resumen original:

### Estadísticas de varias líneas para variables continuas

Si desea imprimir varias líneas de estadísticas para variables continuas, puede indicarlo estableciendo el tipo = a "continuous2". Puedes combinar todos los elementos mostrados anteriormente en una tabla eligiendo qué estadísticas quiere mostrar. Para ello, debe indicar a la función que desea obtener una tabla introduciendo el tipo como "continuous2". El número de valores faltantes se muestra como "Desconocido".

Hay muchas otras formas de modificar estas tablas, incluyendo la adición de valores p, el ajuste del color y los títulos, etc. Muchas de ellas se describen en la documentación (introduzca ?tbl\_summary en Console), y algunas se dan en la sección de [pruebas estadísticas](https://epirhandbook.com/simple-statistical-tests.html).

## ****base**** R

Puedes utilizar la función table() para tabular y cruzar las columnas. A diferencia de las opciones anteriores, debe especificar el dataframe cada vez que haga referencia a un nombre de columna, como se muestra a continuación.

**ATENCIÓN:** Los valores NA (missing) **no** se tabularán a menos que se incluya el argumento useNA = "always" (que también podría establecerse como "no" o "ifany").

**CONSEJO:** Puedes utilizar el %$% de **magrittr** para eliminar la necesidad de repetir las llamadas al dataframe dentro de las funciones **base**. Por ejemplo, lo siguiente podría escribirse tabla %$%(resultado, useNA = "siempre")

Se pueden cruzar varias columnas enumerándolas una tras otra, separadas por comas. Opcionalmente, se puede asignar a cada columna un "nombre" como Outcome = linelist$outcome.

### Proporciones

Para devolver las proporciones, pase la tabla anterior a la función prop.table(). Utiliza el argumento márgenes = para especificar si desea que las proporciones sean de filas (1), de columnas (2) o de toda la tabla (3). Para mayor claridad, canalizamos la tabla a la función round() de **base** R, especificando 2 dígitos.

### Totales

Para añadir los totales de filas y columnas, pase la tabla a addmargins(). Esto funciona tanto para recuentos como para proporciones.

### Convertir en dataframe

Convertir un objeto table() directamente en un dataframe no es sencillo. A continuación se muestra un enfoque:

1. Cree la tabla, sin utilizar useNA = "always". En su lugar, convierta los valores NA en "(Missing)" con fct\_explicit\_na() de **forcats**.
2. Añade los totales (opcional) pasando por addmargins()
3. Pipe a la función **base de** R as.data.frame.matrix()
4. Enviar la tabla a la función **tibble** rownames\_to\_column(), especificando el nombre de la primera columna
5. Imprima, visualice o exporte según desee. En este ejemplo utilizamos flextable() del paquete **flextable** como se describe en la página [Tablas para la presentación](#tables-for-presentation). Esto imprimirá en el panel de visualización de RStudio como una bonita imagen HTML.

## Recursos

Gran parte de la información de esta página está adaptada de estos recursos y viñetas en línea:

[gtsummary](http://www.danieldsjoberg.com/gtsummary/articles/tbl_summary.html)

[dplyr](https://dplyr.tidyverse.org/articles/grouping.html)

# #Tests estadísticos sencillos

{#simple-statistical-tests}

Esta página muestra cómo realizar tests estadísticos sencillos utilizando R **básico**, **rstatix** y **gtsummary**.

* Prueba T
* Prueba de Shapiro-Wilk
* Prueba de suma de rangos de Wilcoxon
* Prueba de Kruskal-Wallis
* Prueba de Chi-cuadrado
* Correlaciones entre variables numéricas

...se pueden realizar muchas otras pruebas, pero sólo mostramos estas comunes y enlazamos con más documentación.

Cada uno de los paquetes mencionados aporta ciertas ventajas y desventajas:

* Utiliza las funciones **básicas de** R para imprimir una salida estadística en la consola de R
* Utiliza las funciones **rstatix** para devolver los resultados en un dataframe, o si desea que las pruebas se ejecuten por grupos
* Utiliza **gtsummary** si desea imprimir rápidamente tablas listas para su publicación

## Preparación

### Cargar paquetes

Este trozo de código muestra la carga de los paquetes necesarios para los análisis. En este manual destacamos p\_load() de **pacman**, que instala el paquete si es necesario y lo carga para su uso. También puede cargar los paquetes instalados con library() de R **base.** Consulte la página sobre [los fundamentos de R](#r-basics) para obtener más información sobre los paquetes de R.

### Importar datos

Importamos los datos de casos de una epidemia de ébola simulada. Si quiere seguir el proceso, [clica para descargar linelist "limpia"](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/case_linelists/linelist_cleaned.rds) (como archivo .rds). Importa tus datos con la función import() del paquete **rio** (acepta muchos tipos de archivos como .xlsx, .rds, .csv - vea la página de [importación y exportación](#import-and-export) para más detalles).

A continuación se muestran las primeras 50 filas del listado.

## ****base**** R

Puedes utilizar las funciones **básicas de** R para realizar pruebas estadísticas. Los comandos son relativamente sencillos y los resultados se imprimen en la consola de R para su visualización. Sin embargo, las salidas suelen ser listas y, por lo tanto, son más difíciles de manipular si desea utilizar los resultados en operaciones posteriores.

### Pruebas T

Una prueba [t](https://en.wikipedia.org/wiki/Student's_t-test), también llamada "Prueba t de Student", se utiliza normalmente para determinar si existe una diferencia significativa entre las medias de alguna variable numérica entre dos grupos. Aquí mostraremos la sintaxis para hacer esta prueba dependiendo de si las columnas están en el mismo dataframe.

**Sintaxis 1:** Esta es la sintaxis cuando tus columnas numéricas y categóricas están en el mismo dataframe. Proporcione la columna numérica en el lado izquierdo de la ecuación y la columna categórica en el lado derecho. Especifique los datos a data =. Opcionalmente, establezca paired = TRUE, y conf.level = (0.95 por defecto), y alternative = (ya sea "two.sided", "less", o "greater"). Introduzca ?t.test para obtener más detalles.

**Sintaxis 2:** Puedes comparar dos vectores numéricos separados utilizando esta sintaxis alternativa. Por ejemplo, si las dos columnas están en conjuntos de datos diferentes.

También puede utilizar una prueba t para determinar si la media de una muestra es significativamente diferente de algún valor específico. Aquí realizamos una prueba t de una muestra con la media poblacional conocida/hipotética como mu =:

### Prueba de Shapiro-Wilk

La [prueba de Shapiro-Wilk](https://en.wikipedia.org/wiki/Shapiro–Wilk_test) puede utilizarse para determinar si una muestra procede de una población distribuida normalmente (un supuesto de muchas otras pruebas y análisis, como la prueba t). Sin embargo, sólo puede utilizarse en una muestra de entre 3 y 5000 observaciones. Para muestras más grandes puede ser útil un [gráfico de cuantiles](https://ggplot2.tidyverse.org/reference/geom_qq.html).

### Prueba de suma de rangos de Wilcoxon

La prueba de suma de rangos de Wilcoxon, también llamada [prueba U de Mann-Whitney](https://en.wikipedia.org/wiki/Mann–Whitney_U_test), se utiliza a menudo para ayudar a determinar si dos muestras numéricas proceden de la misma distribución cuando tus poblaciones no se distribuyen normalmente o tienen una varianza desigual.

### Prueba de Kruskal-Wallis

La prueba de [Kruskal-Wallis](https://en.wikipedia.org/wiki/Kruskal–Wallis_one-way_analysis_of_variance) es una extensión de la prueba de suma de rangos de Wilcoxon que puede utilizarse para comprobar las diferencias en la distribución de más de dos muestras. Cuando sólo se utilizan dos muestras, los resultados son idénticos a los de la prueba de suma de rangos de Wilcoxon.

### Prueba de Chi-cuadrado

La [prueba de Chi-cuadrado de Pearson](https://en.wikipedia.org/wiki/Chi-squared_test) se utiliza para comprobar las diferencias significativas entre grupos categóricos.

## paquete ****rstatix****

El paquete **rstatix** ofrece la posibilidad de ejecutar pruebas estadísticas y recuperar los resultados en un marco "amigable". Los resultados se encuentran automáticamente en un dataframe para que pueda realizar operaciones posteriores con los resultados. También es fácil agrupar los datos que se pasan a las funciones, de modo que las estadísticas se ejecutan para cada grupo.

### Estadísticas resumidas

La función get\_summary\_stats() es una forma rápida de devolver estadísticas de resumen. Sólo tiene que enviar su conjunto de datos a esta función y proporcionar las columnas que desea analizar. Si no se especifica ninguna columna, las estadísticas se calculan para todas las columnas.

Por defecto, se devuelve una gama completa de estadísticas de resumen: n, max, min, mediana, 25%ile, 75%ile, IQR, desviación absoluta mediana (mad), media, desviación estándar, error estándar y un intervalo de confianza de la media.

Puedes especificar un subconjunto de estadísticas de resumen para devolver proporcionando uno de los siguientes valores a type =: "completo", "común", "robusto", "cinco\_números", "media\_sd", "media\_se", "media\_ci", "mediana\_iqr", "mediana\_mad", "cuantil", "media", "mediana", "min", "max".

También puede utilizarse con datos agrupados, de forma que se devuelva una fila por cada variable de agrupación:

También puede utilizar **rstatix** para realizar pruebas estadísticas:

### Prueba T

Utiliza una sintaxis de fórmula para especificar las columnas numéricas y categóricas:

O utiliza ~ 1 y especifique mu = para una prueba T de una muestra. Esto también puede hacerse por grupo.

Si procede, las pruebas estadísticas pueden realizarse por grupos, como se muestra a continuación:

### Prueba de Shapiro-Wilk

Como ya se ha dicho, el tamaño de la muestra debe estar entre 3 y 5000.

### Prueba de suma de rangos de Wilcoxon

### Prueba de Kruskal-Wallis

También conocida como la prueba U de Mann-Whitney.

### Prueba de Chi-cuadrado

La función de prueba chi-cuadrado acepta una tabla, así que primero creamos una tabulación cruzada. Hay muchas formas de crear una tabulación cruzada (véase [Tablas descriptivas](#descriptive-tables)), pero aquí utilizamos tabyl() de **janitor** y eliminamos la columna más a la izquierda de las etiquetas de valores antes de pasarla a chisq\_test().

Se pueden ejecutar muchas más funciones y pruebas estadísticas con las funciones de rstatix. Consulte la documentación de **rstatix** [en línea aquí](https://github.com/kassambara/rstatix) o introduciendo ?rstatix.

## paquete gtsummary

Utiliza **gtsummary** si desea añadir los resultados de una prueba estadística a una tabla bonita creada con este paquete (como se describe en la sección **gtsummary de la** página [Tablas descriptivas](#tbl_gt)).

La realización de pruebas estadísticas de comparación con tbl\_summary se lleva a cabo añadiendo la función add\_p a una tabla y especificando qué prueba utilizar. Es posible obtener valores p corregidos para pruebas múltiples utilizando la función add\_q. Ejecuta ?tbl\_summary para obtener más detalles.

### Prueba de Chi-cuadrado

Compara las proporciones de una variable categórica en dos grupos. La prueba estadística por defecto de add\_p() cuando se aplica a una variable categórica es realizar una prueba de independencia de chi-cuadrado con corrección de continuidad, pero si algún recuento de llamadas esperado es inferior a 5, se utiliza una prueba exacta de Fisher.

### Pruebas T

Compare la diferencia de medias de una variable continua en dos grupos. Por ejemplo, compare la media de edad por resultado del paciente.

### Prueba de suma de rangos de Wilcoxon

Compara la distribución de una variable continua en dos grupos. Por defecto se utiliza la prueba de suma de rangos de Wilcoxon y la mediana (IQR) cuando se comparan dos grupos. Sin embargo, para datos no distribuidos normalmente o para comparar varios grupos, la prueba de Kruskal-wallis es más apropiada.

### Prueba de Kruskal-Wallis

Comparar la distribución de una variable continua en dos o más grupos, independientemente de que los datos se distribuyan normalmente.

## Correlaciones

La correlación entre variables numéricas puede investigarse con el paquete **tidyverse**  
**corrr**. Permite calcular las correlaciones mediante Pearson, Kendall tau o Spearman rho. El paquete crea una tabla y también tiene una función para trazar automáticamente los valores.

## Recursos

Gran parte de la información de esta página está adaptada de estos recursos y viñetas en línea:

[gtsummary](http://www.danieldsjoberg.com/gtsummary/articles/tbl_summary.html) [dplyr](https://dplyr.tidyverse.org/articles/grouping.html) [corrr](https://corrr.tidymodels.org/articles/using-corrr.html) [sthda correlación](http://www.sthda.com/english/wiki/correlation-test-between-two-variables-in-r)

# #Regresión univariable y multivariable

{#univariate-and-multivariable-regression}

Esta página muestra el uso de las funciones de regresión **básicas** de R, como glm() y el paquete **gtsummary,** para observar las asociaciones entre variables (por ejemplo, odds ratios, risk ratios y hazard ratios). También utiliza funciones como tidy() del paquete **broom** para limpiar los resultados de la regresión.

1. Univariante: tablas de dos en dos
2. Estratificado: estimaciones mantel-haenszel
3. Multivariable: selección de variables, selección de modelos, tabla final
4. gráficas forestales

Para la regresión de riesgos proporcionales de Cox, véase la página de [análisis de supervivencia](#survival-analysis).

**NOTA:** Utilizamos el término multivariable para referirnos a una regresión con múltiples variables explicativas. En este sentido, un modelo multivariable sería una regresión con varios resultados - véase este [editorial](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3518362/) para más detalles

## Preparación

### Cargar paquetes

Este trozo de código muestra la carga de los paquetes necesarios para los análisis. En este manual destacamos p\_load() de **pacman**, que instala el paquete si es necesario y lo carga para su uso. También puede cargar los paquetes instalados con library() de R **base.** Consulte la página sobre [los fundamentos de R](#r-basics) para obtener más información sobre los paquetes de R.

### Importar datos

Importamos los datos de casos de una epidemia de ébola simulada. Si quiere seguir el proceso, [clica para descargar linelist "limpia"](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/case_linelists/linelist_cleaned.rds) (como archivo .rds). Importa tus datos con la función import() del paquete **rio** (acepta muchos tipos de archivos como .xlsx, .rds, .csv - vea la página de [importación y exportación](#import-and-export) para más detalles).

A continuación se muestran las primeras 50 filas del listado.

### Datos limpios

#### Almacenar las variables explicativas

Almacenamos los nombres de las columnas explicativas como un vector de caracteres. Esto se referenciará más adelante.

#### Convertir a 1's y 0's

A continuación convertimos las columnas explicativas de "sí"/"no", "m"/"f", y "muerto"/"vivo" a **1 / 0**, para cooperar con las expectativas de los modelos de regresión logística. Para hacer esto de manera eficiente, utilizamos across() de **dplyr** para transformar varias columnas a la vez. La función que aplicamos a cada columna es case\_when() (también de **dplyr**) que aplica la lógica para convertir los valores especificados en 1's y 0's. Vea las secciones sobre across() y case\_when() en la [página de Limpieza de datos y funciones básicas](#clean_across)).

Nota: el "." que aparece a continuación representa la columna que está siendo procesada por across() en ese momento.

#### Eliminar las filas con valores faltantes

Para eliminar las filas con valores faltantes, puede utilizar la función **tidyr** drop\_na(). Sin embargo, sólo queremos hacer esto para las filas a las que les faltan valores en las columnas de interés.

Lo primero que debemos hacer es asegurarnos de que nuestro vector explanatory\_vars incluye la columna edad (la edad habría producido un error en la operación anterior case\_when(), que sólo era para variables dicotómicas). A continuación, canalizamos linelist a drop\_na() para eliminar cualquier fila con valores faltantes en la columna de resultados o en cualquiera de las columnas explanatory\_vars.

Antes de ejecutar el código, el número de filas del listado es nrow(linelist).

El número de filas que quedan en linelist es nrow(linelist).

## Univariante

Al igual que en la página sobre [Tablas descriptivas](https://epirhandbook.com/descriptive-tables.html), su caso de uso determinará el paquete de R que utilice. Presentamos dos opciones para realizar análisis univariantes:

* Utiliza las funciones disponibles en R **base** para imprimir rápidamente los resultados en la consola. Utiliza el paquete **broom** para ordenar las salidas.
* Utiliza el paquete **gtsummary** para modelar y obtener resultados listos para su publicación

### ****base**** R

#### Regresión lineal

La función **base de** R lm() realiza una regresión lineal, evaluando la relación entre la respuesta numérica y las variables explicativas que se supone tienen una relación lineal.

Proporcione la ecuación como una fórmula, con los nombres de las columnas de respuesta y explicativa separados por una tilde ~. Además, especifique los datos a datos =. Defina los resultados del modelo como un objeto R, para utilizarlo más tarde.

A continuación, puede ejecutar summary() en los resultados del modelo para ver los coeficientes (estimaciones), el valor P, los residuos y otras medidas.

También se puede utilizar la función tidy() del paquete **broom** para obtener los resultados en una tabla. Lo que nos dicen los resultados es que por cada año de aumento de la edad la altura aumenta 3,5 cm y esto es estadísticamente significativo.

También puede utilizar esta regresión para añadirla a un **ggplot**, para hacer esto, primero juntamos los puntos de los datos observados y la línea ajustada en un dataframe utilizando la función augment() de **broom**.

También es posible añadir una simple recta de regresión lineal en **ggplot** utilizando la función geom\_smooth().

Consulte la sección de recursos al final de este capítulo para obtener tutoriales más detallados.

#### Regresión logística

La función glm() del paquete **stats** (parte de R **base**) se utiliza para ajustar los modelos lineales generalizados (GLM).

glm() puede utilizarse para la regresión logística univariable y multivariable (por ejemplo, para obtener Odds Ratios). Aquí están las partes principales:

* fórmula = El modelo se proporciona a glm() como una ecuación, con el resultado a la izquierda y las variables explicativas a la derecha de una tilde ~.
* familia = Determina el tipo de modelo a ejecutar. Para la regresión logística, utiliza la familia = "binomial", para poisson utiliza la familia = "poisson". Otros ejemplos se encuentran en la tabla siguiente.
* data = Especifica tu dataframe

Si es necesario, también puede especificar la función de enlace mediante la sintaxis family = familytype(link = "linkfunction")). Puedes leer más en la documentación sobre otras familias y argumentos opcionales como pesos = y subconjunto = (?glm).

| **Familia** | **Función de enlace por defecto** |
| --- | --- |
| "binomio" | (enlace = "logit") |
| "gaussiano" | (enlace = "identidad") |
| "Gamma" | (enlace = "inverso") |
| "inverso.gaussiano" | (enlace = "1/mu^2") |
| "poisson" | (enlace = "registro") |
| "cuasi" | (enlace = "identidad", varianza = "constante") |
| "cuasibinomio" | (enlace = "logit") |
| "quasipoisson" | (enlace = "registro") |

Cuando se ejecuta glm() lo más habitual es guardar los resultados como un objeto R con nombre. A continuación, puede imprimir los resultados en su consola utilizando summary() como se muestra a continuación, o realizar otras operaciones con los resultados (por ejemplo, exponer).

Si necesita ejecutar una regresión binomial negativa, puede utilizar el paquete **MASS**; glm.nb() utiliza la misma sintaxis que glm(). Para un recorrido por diferentes regresiones, consulte la [página de estadísticas de UCLA](https://stats.idre.ucla.edu/other/dae/).

#### Univariante glm()

En este ejemplo estamos evaluando la asociación entre diferentes categorías de edad y el resultado de muerte (codificado como 1 en la sección Preparación). A continuación se muestra un modelo univariante del resultado por age\_cat. Guardamos la salida del modelo como modelo y luego la imprimimos con summary() en la consola. Observe que las estimaciones proporcionadas son las probabilidades logarítmicas y que el nivel de referencia es el primer nivel del factor age\_cat ("0-4").

Para modificar el nivel de referencia de una variable determinada, asegúrese de que la columna es del tipo Factor y mueva el nivel deseado a la primera posición con fct\_relevel() (véase la página sobre [Factores](#factors)). Por ejemplo, a continuación tomamos la columna age\_cat y establecemos "20-29" como línea de base antes de canalizar el dataframe modificado en glm().

#### Resultados de la impresión

Para la mayoría de los usos, hay que hacer varias modificaciones a los resultados anteriores. La función tidy() del paquete **broom** es conveniente para hacer presentables los resultados del modelo.

Aquí demostramos cómo combinar los resultados del modelo con una tabla de recuentos.

1. Obtenga las estimaciones de log odds ratio exponenciadas y los intervalos de confianza pasando el modelo a tidy() y estableciendo exponentiate = TRUE y conf.int = TRUE.

A continuación, se muestra el modelo de tiburón resultante:

1. Combine estos resultados del modelo con una tabla de recuentos. A continuación, creamos la tabla cruzada de recuentos con la función tabyl() de **janitor**, como se explica en la página de [tablas descriptivas](#descriptive-tables).

Este es el aspecto de este dataframe counts\_table:

Ahora podemos unir la tabla\_de\_conteo y los resultados del modelo horizontalmente con bind\_cols() (**dplyr**). Recuerde que con bind\_cols() las filas de los dos dataframes deben estar perfectamente alineadas. En este código, como estamos enlazando dentro de una cadena de tuberías, utilizamos . para representar el objeto de tuberías counts\_table mientras lo enlazamos con el modelo. Para terminar el proceso, utilizamos select() para elegir las columnas deseadas y su orden, y finalmente aplicamos la función **base** de R round() en todas las columnas numéricas para especificar 2 decimales.

Este es el aspecto del dataframe combinado, impreso de forma agradable como una imagen con una función de **flextable**. En [Tablas para la presentación](#tables-for-presentation) se explica cómo personalizar dichas tablas con **flextable**, o bien puede utilizar otros numerosos paquetes como **knitr** o **GT**.

#### Bucle de múltiples modelos univariantes

A continuación presentamos un método que utiliza glm() y tidy() para un enfoque más sencillo, véase la sección sobre **gtsummary**.

Para ejecutar los modelos en varias variables de exposición para producir odds ratios univariantes (es decir, sin controlar entre sí), puede utilizar el enfoque siguiente. Utiliza str\_c() de **stringr** para crear fórmulas univariantes (véase [Caracteres y cadenas](#characters-and-strings)), ejecuta la regresión glm() en cada fórmula, pasa cada resultado de glm() a tidy() y finalmente colapsa todos los resultados del modelo juntos con bind\_rows() de **tidyr**. Este enfoque utiliza map() del paquete **purrr** para iterar - véase la página sobre [Iteración, bucles y listas](#iteration-loops-and-lists) para más información sobre esta herramienta.

1. Cree un vector de nombres de columnas de las variables explicativas. Ya lo tenemos como explanatory\_vars de la sección de preparación de esta página.
2. Utiliza str\_c() para crear múltiples fórmulas de cadena, con el resultado a la izquierda, y un nombre de columna de explanatory\_vars a la derecha. El punto . sustituye al nombre de la columna en explanatory\_vars.
3. Pase estas fórmulas de cadena a map() y establezca ~glm() como la función a aplicar a cada entrada. Dentro de glm(), establezca la fórmula de regresión como as.formula(.x), donde .x se sustituirá por la fórmula de cadena definida en el paso anterior. map() realizará un bucle sobre cada una de las fórmulas de cadena, ejecutando regresiones para cada una.
4. Los resultados de este primer map() se pasan a un segundo comando map(), que aplica tidy() a los resultados de la regresión.
5. Por último, la salida de la segunda función map() (una lista de dataframes ordenados) se condensa con bind\_rows(), dando lugar a un dataframe con todos los resultados univariantes.

Esta vez, el objeto final modelos es más largo porque ahora representa los resultados combinados de varias regresiones univariantes. Clica para ver todas las filas del modelo.

Como antes, podemos crear una tabla de recuentos a partir del listado para cada variable explicativa, vincularla a los modelos y hacer una bonita tabla. Comenzamos con las variables, e iteramos a través de ellas con map(). Iteramos a través de una función definida por el usuario que implica la creación de una tabla de recuentos con funciones **dplyr**. Luego se combinan los resultados y se vinculan con los resultados del modelo de los modelos.

A continuación se muestra el aspecto del dataframe. Consulte la página sobre [Tablas para la presentación](#tables-for-presentation) para obtener ideas sobre cómo convertir esta tabla en una bonita salida HTML (por ejemplo, con **flextable**).

### paquete ****gtsummary****

A continuación presentamos el uso de tbl\_uvregression() del paquete **gtsummary**. Al igual que en la página sobre [Tablas descriptivas](https://epirhandbook.com/descriptive-tables.html), las funciones de gtsummary hacen un buen trabajo al ejecutar estadísticas y producir salidas de aspecto profesional. Esta función produce una tabla de resultados de regresión univariante.

Seleccionamos sólo las columnas necesarias del listado (variables explicativas y la variable de resultado) y las introducimos en tbl\_uvregression(). Vamos a ejecutar una regresión univariante en cada una de las columnas que definimos como explanatory\_vars en la sección de preparación de datos (sexo, fiebre, escalofríos, tos, dolores, vómitos y age\_cat).

Dentro de la propia función, proporcionamos el método = como glm (sin comillas), la columna y = resultado (resultado), especificamos a method.args = que queremos ejecutar la regresión logística a través de la familia = binomial, y le decimos que exponencie los resultados.

La salida es HTML y contiene los recuentos

Hay muchas modificaciones que se pueden hacer a la salida de esta tabla, como ajustar las etiquetas de texto, poner en negrita las filas por su valor p, etc. Consulte los tutoriales [aquí](http://www.danieldsjoberg.com/gtsummary/articles/tbl_regression.html) y en otros lugares en línea.

## Estratificado

El análisis estratificado se está trabajando actualmente para **gtsummary**, esta página se actualizará a su debido tiempo.

## Multivariable

Para el análisis multivariable, volvemos a presentar dos enfoques:

* glm() y tidy()
* paquete **gtsummary**

El flujo de trabajo es similar para cada uno de ellos y sólo el último paso de elaborar una tabla final es diferente.

### Llevar a cabo la multivariable

Aquí utilizamos glm() pero añadimos más variables al lado derecho de la ecuación, separadas por símbolos de suma (+).

Para ejecutar el modelo con todas nuestras variables explicativas ejecutaríamos:

Si quiere incluir dos variables y una interacción entre ellas puede separarlas con un asterisco \* en lugar de un +. Si sólo especifica la interacción, sepárelas con dos puntos :. Por ejemplo:

Opcionalmente, puede utilizar este código para aprovechar el vector predefinido de nombres de columnas y volver a crear el comando anterior utilizando str\_c(). Esto puede ser útil si los nombres de sus variables explicativas cambian, o si no quiere escribirlos todos de nuevo.

#### Construir el modelo

Puedes construir su modelo paso a paso, guardando varios modelos que incluyan determinadas variables explicativas. Puedes comparar estos modelos con pruebas de razón de verosimilitud utilizando lrtest() del paquete **lmtest**, como se indica a continuación:

**NOTA: El** uso de la **base** anova(model1, model2, test = "Chisq) produce los mismos resultados

Otra opción es tomar el objeto modelo y aplicar la función step() del paquete **stats**. Especifique qué dirección de selección de variables desea utilizar al construir el modelo.

También puede desactivar la notación científica en su sesión de R, para mayor claridad:

Como se describe en la sección sobre el análisis univariante, pasamos la salida del modelo a tidy() para exponer las probabilidades logarítmicas y los IC. Finalmente, redondeamos todas las columnas numéricas a dos decimales. Desplácese para ver todas las filas.

Este es el aspecto del dataframe resultante:

### Combinar univariable y multivariable

#### Combinar con ****gtsummary****

El paquete **gtsummary** proporciona la función tbl\_regression(), que tomará los resultados de una regresión (glm() en este caso) y producirá una bonita tabla resumen.

Veamos la tabla:

También puede combinar varias tablas de salida diferentes producidas por **gtsummary** con la función tbl\_merge(). Ahora combinamos los resultados multivariables con los resultados univariables de **gtsummary** que creamos [anteriormente](#reg_gt_uni):

#### Combinar con ****dplyr****

Una forma alternativa de combinar los resultados univariables y multivariables de glm()/tidy() es con las funciones join **de dplyr**.

* Unir los resultados univariables de antes (univ\_tab\_base, que contiene los recuentos) con los resultados multivariables ordenados mv\_tab\_base
* Utilizar select() para mantener sólo las columnas que queremos, especificar su orden y renombrarlas
* Utiliza round() con dos decimales en toda la columna que sea de tipo Double

## forest plot

Esta sección muestra cómo producir un gráfico con los resultados de su regresión. Hay dos opciones, puedes construir un gráfico tú mismo usando **ggplot2** o usar un metapaquete llamado **easystats** (un paquete que incluye muchos paquetes).

Consulte la página sobre [los fundamentos de ggplot](#ggplot-basics) si no está familiarizado con el paquete de trazado **ggplot2**.

### Paquete ****ggplot2****

Puedes construir un gráfico de bosque con ggplot() trazando elementos de los resultados de la regresión multivariable. Añada las capas de los gráficos utilizando estos "geoms":

* estimaciones con geom\_point()
* intervalos de confianza con geom\_errorbar()
* una línea vertical en OR = 1 con geom\_vline()

Antes de trazar, es posible que desee utilizar fct\_relevel() del paquete **forcats** para establecer el orden de las variables/niveles en el eje y. ggplot() puede mostrarlos en orden alfanumérico, lo que no funcionaría bien para estos valores de categoría de edad ("30" aparecería antes de "5"). Vea la página sobre [Factores](#factors) para más detalles.

### paquetes ****easystats****

Una alternativa, si no desea el nivel de control fino que proporciona **ggplot2**, es utilizar una combinación de paquetes **easystats**.

La función model\_parameters() del paquete **parameters** hace el equivalente de la función tidy() del paquete **broom**. El paquete **see** acepta esos resultados y crea un gráfico forestal por defecto como un objeto ggplot().

## Recursos

El contenido de esta página se ha basado en estos recursos y viñetas en línea:

[Regresión lineal en R](https://www.datacamp.com/community/tutorials/linear-regression-R)

[gtsummary](http://www.danieldsjoberg.com/gtsummary/articles/tbl_regression.html)

[Página de estadísticas de la UCLA](https://stats.idre.ucla.edu/other/dae/)

[regresión escalonada sthda](http://www.sthda.com/english/articles/36-classification-methods-essentials/150-stepwise-logistic-regression-essentials-in-r/)

# #Valores faltantes

{#missing-data}

En esta página se explica cómo hacerlo:

1. Evaluar la falta de información
2. Filtrar las filas por falta de datos
3. Trazado de la falta de datos a lo largo del tiempo
4. Manejar cómo se muestra NA en los gráficos
5. Realiza la imputación de valores faltantes: MCAR, MAR, MNAR

## Preparación

### Cargar paquetes

Este trozo de código muestra la carga de los paquetes necesarios para los análisis. En este manual destacamos p\_load() de **pacman**, que instala el paquete si es necesario y lo carga para su uso. También puede cargar los paquetes instalados con library() de R **base.** Consulte la página sobre [los fundamentos de R](#r-basics) para obtener más información sobre los paquetes de R.

### Importar datos

Importamos los datos de casos de una epidemia de ébola simulada. Si quiere seguir el proceso, [clica para descargar linelist "limpia"](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/case_linelists/linelist_cleaned.rds) (como archivo .rds). Importa tus datos con la función import() del paquete **rio** (acepta muchos tipos de archivos como .xlsx, .rds, .csv - vea la página de [importación y exportación](#import-and-export) para más detalles).

A continuación se muestran las primeras 50 filas del listado.

### Falta la conversión en la importación

Al importar los datos, Ten en cuenta los valores que deben clasificarse como ausentes. Por ejemplo, 99, 999, "Falta", celdas en blanco ("") o celdas con un espacio vacío (" "). Puedes convertirlos en NA (la versión de R de los valores faltantes) durante el comando de importación de datos.   
Consulte la página sobre la importación de la sección de [datos faltantes](#import_missing) para obtener más detalles, ya que la sintaxis exacta varía según el tipo de archivo.

## Valores faltantes en R

A continuación, exploramos las formas en que se presenta y evalúa la falta en R, junto con algunos valores y funciones adyacentes.

### NA

En R, los valores que faltan se representan con un valor reservado (especial): NA. Ten en cuenta que se escribe sin comillas. "NA" es diferente y es sólo un valor de carácter normal (también una letra de los Beatles de la canción Hey Jude).

Tus datos pueden tener otras formas de representar la falta, como "99", o "Falta", o "Desconocido" - incluso puede tener el valor de carácter vacío "" que parece "en blanco", o un solo espacio " ". Sea consciente de ello y considere la posibilidad de [convertirlos enNAdurante la importación](#import_missing) o durante la limpieza de datos con na\_if().

En su limpieza de datos, también puede querer convertir en el otro sentido - cambiando todos los NA a "Missing" o similar con replace\_na() o con fct\_explicit\_na() para los factores.

### Versiones de NA

La mayoría de las veces, NA representa un valor que falta y todo funciona bien. Sin embargo, en algunas circunstancias puede encontrar la necesidad de variaciones de NA específicas para un tipo de objeto (carácter, numérico, etc.). Esto será poco frecuente, pero debe tenerlo en cuenta.   
El escenario típico para esto es cuando se crea una nueva columna con la función **dplyr** case\_when(). Como se describe en la página de [Limpieza de datos y funciones básicas](#clean_case_when), esta función evalúa cada fila del dataframe, valora si las filas cumplen con los criterios lógicos especificados (lado derecho del código), y asigna el nuevo valor correcto (lado izquierdo del código). Importante: todos los valores del lado derecho deben ser del mismo tipo.

Si desea NA en el lado derecho, es posible que tenga que especificar una de las opciones especiales de NA que se indican a continuación. Si los otros valores del lado derecho son caracteres, considere usar "Missing" en su lugar o, de lo contrario, use NA\_character\_. Si todos son numéricos, utiliza NA\_real\_. Si todos son fechas o lógicos, puede utilizar NA.

* NA - utilizar para fechas o TRUE/FALSE lógico
* NA\_character\_ - utilizar para caracteres
* NA\_real\_ - uso para numérico

De nuevo, no es probable que se encuentre con estas variaciones a menos que esté utilizando case\_when() para crear una nueva columna. Consulte la [documentación de R sobre NA](https://stat.ethz.ch/R-manual/R-devel/library/base/html/NA.html) para obtener más información.

### NULL

NULL es otro valor reservado en R. Es la representación lógica de una declaración que no es ni verdadera ni falsa. Es devuelto por expresiones o funciones cuyos valores son indefinidos. Generalmente no asignes NULL como valor, a menos que escribas funciones o quizás escribas una [aplicación**Shiny**](#dashboards-with-shiny) para devolver NULL en escenarios específicos.

La nulidad puede evaluarse con is.null() y la conversión puede hacerse con as.null().

Véase esta [entrada del blog](https://www.r-bloggers.com/2010/04/r-na-vs-null/) sobre la diferencia entre NULL y NA.

### NaN

Los valores imposibles se representan con el valor especial NaN. Un ejemplo de esto es cuando se fuerza a R a dividir 0 entre 0. Puedes evaluar esto con is.nan(). También puede encontrar funciones complementarias incluyendo is.infinite() y is.finite().

### Inf

Inf representa un valor infinito, como cuando se divide un número por 0.

Como ejemplo de cómo esto podría afectar a tu trabajo: digamos que tienes un vector/columna z que contiene estos valores: z <- c(1, 22, NA, Inf, NaN, 5)

Si desea utilizar max() en la columna para encontrar el valor más alto, puede utilizar el na.rm = TRUE para eliminar el NA del cálculo, pero el Inf y el NaN permanecen y se devolverá Inf. Para resolver esto, puede utilizar los corchetes [ ] y is.finite() para subconjuntar de manera que sólo se utilicen valores finitos para el cálculo: max(z[is.finite(z)]).

### Ejemplos

| **Comando R** | **Resultado** |
| --- | --- |
| 5 / 0 | Inf |
| 0 / 0 | NaN |
| 5 / NA | NA |
| 5 / Inf |0NA - 5|NAInf / 5|Class(NA)|Class "logical"(NaN)|Class "numeric(Inf)|Class "numeric(NULL)` | "NULL" |

"NAs introducidos por coerción" es un mensaje de advertencia común. Esto puede ocurrir si se intenta hacer una conversión ilegal como insertar un valor de carácter en un vector que de otra manera es numérico.

NULL se ignora en un vector.

La variación de un número da como resultado NA.

## Funciones útiles

Las siguientes son funciones **básicas de** R muy útiles a la hora de evaluar o manejar los valores faltantes:

### is.na() y !is.na()

Utiliza is.na()para identificar los valores que faltan, o utiliza su opuesto (con ! delante) para identificar los valores que no faltan. Ambos devuelven un valor lógico (TRUE o FALSE). Recuerde que puede sumar() el vector resultante para contar el número de TRUE, por ejemplo, sum(is.na(linelist$date\_outcome)).

### na.omit()

Esta función, si se aplica a un dataframe, eliminará las filas con valores faltantes. También es de **base** R. Si se aplica a   
un vector, eliminará los valores NA del vector al que se aplica. Por ejemplo:

### drop\_na()

Esta es una función **tidyr** que es útil en una [línea de limpieza de datos](#cleaning-data-and-core-functions). Si se ejecuta con los paréntesis vacíos, elimina las filas con valores faltantes. Si se especifican los nombres de las columnas en los paréntesis, se eliminarán las filas con valores faltantes en esas columnas. También puede utilizar la sintaxis "tidyselect" para especificar las columnas.

### na.rm = TRUE

Cuando se ejecuta una función matemática como max(), min(), sum() o mean(), si hay algún valor NA presente el valor devuelto será NA. Este comportamiento por defecto es intencionado, para que se le avise si falta algún dato.

Puedes evitarlo eliminando los valores faltantes del cálculo. Para ello, incluya el argumento na.rm = TRUE ("na.rm" significa "eliminar NA").

## Evaluar la ausencia de datos en un dataframe

Puedes utilizar el paquete **naniar** para evaluar y visualizar la falta de datos en linelist del dataframe.

### Cuantificación de la ausencia de datos

Para encontrar el porcentaje de todos los valores que faltan utiliza pct\_miss(). Utiliza n\_miss() para obtener el número de valores faltantes.

Las dos funciones siguientes devuelven el porcentaje de filas con algún valor ausente, o que están totalmente completas, respectivamente. Recuerde que NA significa que falta, y que `"" o " " no se contarán como faltantes.

### Visualización de la falta

La función gg\_miss\_var() le mostrará el número (o %) de valores faltantes en cada columna. Algunos matices:

* Puedes añadir un nombre de columna (no entre comillas) al argumento facet = para ver el gráfico por grupos
* Por defecto, se muestran los recuentos en lugar de los porcentajes, cámbielo con show\_pct = TRUE
* Puedes añadir etiquetas de eje y de título como para un ggplot() normal con + labs(...)

Aquí los datos son canalizados %>% en la función. El argumento facet = también se utiliza para dividir los datos.

Puedes utilizar vis\_miss() para visualizar el dataframe como un mapa de calor, mostrando si cada valor falta o no. También puede seleccionar() determinadas columnas del dataframe y proporcionar sólo esas columnas a la función.

### Explorar y visualizar las relaciones de ausencia

¿Cómo se visualiza algo que no existe? Por defecto, ggplot() elimina los puntos con valores faltantes de los gráficos.

**naniar** ofrece una solución mediante geom\_miss\_point(). Al crear un gráfico de dispersión de dos columnas, los registros con uno de los valores ausentes y el otro valor presente se muestran estableciendo los valores ausentes en un 10% más bajo que el valor más bajo de la columna, y coloreándolos de forma distinta.

En el gráfico de dispersión que aparece a continuación, los puntos rojos son registros en los que el valor de una columna está presente pero falta el valor de la otra columna. Esto permite ver la distribución de los valores que faltan en relación con los valores que no faltan.

Para evaluar la ausencia en el dataframe estratificado por otra columna, considere gg\_miss\_fct(), que devuelve un mapa de calor del porcentaje de ausencia en el dataframe por una columna de factor/categoría (o fecha):

Esta función también se puede utilizar con una columna de fechas para ver cómo ha cambiado la falta en el tiempo:

### Columnas de "sombra"

Otra forma de visualizar la ausencia de valores en una columna mediante los valores de una segunda columna es utilizar la "sombra" que puede crear **naniar.** bind\_shadow() crea una columna binaria NA/no NA para cada columna existente, y vincula todas estas nuevas columnas al conjunto de datos original con el apéndice "\_NA". Esto duplica el número de columnas - ver más abajo:

Estas columnas "sombra" pueden utilizarse para trazar la proporción de valores que faltan, por cualquier otra columna.

Por ejemplo, el siguiente gráfico muestra la proporción de registros que carecen de days\_onset\_hosp (número de días desde el inicio de los síntomas hasta la hospitalización), según el valor de ese registro en date\_hospitalisation. Esencialmente, se está trazando la densidad de la columna del eje x, pero estratificando los resultados (color =) por una columna de sombra de interés. Este análisis funciona mejor si el eje x es una columna numérica o de fecha.

También puede utilizar estas columnas "sombra" para estratificar un resumen estadístico, como se muestra a continuación:

A continuación se muestra una forma alternativa de trazar la proporción de los valores de una columna que faltan a lo largo del tiempo. No implica **naniar**. Este ejemplo muestra el porcentaje de observaciones semanales que faltan).

1. Agregue los datos en una unidad de tiempo útil (días, semanas, etc.), resumiendo la proporción de observaciones con NA (y cualquier otro valor de interés)
2. Trazar la proporción que falta como una línea usando ggplot()

A continuación, tomamos linelist, añadimos una nueva columna para la semana, agrupamos los datos por semana y luego calculamos el porcentaje de registros de esa semana en los que falta el valor. (Nota: si se desea el porcentaje de 7 días el cálculo sería ligeramente diferente).

A continuación, trazamos la proporción que falta como una línea, por semana. La página de [fundamentos de ggplot](#ggplot-basics) si no está familiarizado con el paquete de trazado **ggplot2**.

## Uso de datos con valores faltantes

### Filtrar las filas con valores faltantes

Para eliminar rápidamente las filas con valores faltantes, utiliza la función **dplyr** drop\_na().

La lista de líneas original tiene nrow(linelist) filas. El número ajustado de filas se muestra a continuación:

Puedes especificar que se eliminen las filas que faltan en determinadas columnas:

Puedes listar las columnas una tras otra, o utilizar [las funciones de ayuda "tidyselect"](#clean_tidyselect):

### Manejo de NA en ggplot()

A menudo es conveniente informar del número de valores excluidos de un gráfico en un pie de foto. A continuación se muestra un ejemplo:

En ggplot(), puedes añadir labs() y dentro de él un caption =. En el caption, puedes usar str\_glue() del paquete **stringr** para pegar los valores juntos en una frase de forma dinámica para que se ajusten a los datos. Un ejemplo es el siguiente:

* Observe el uso de \n para una nueva línea.
* Ten en cuenta que si varias columnas contribuyen a que los valores no se muestren (por ejemplo, la edad o el sexo si se reflejan en el gráfico), deberá filtrar también esas columnas para calcular correctamente el número no mostrado.

A veces, puede ser más fácil guardar la cadena como un objeto en comandos anteriores al comando ggplot(), y simplemente referenciar el objeto de cadena nombrado dentro de str\_glue().

### NA en los factores

Si su columna de interés es un factor, utiliza fct\_explicit\_na() del paquete **forcats** para convertir los valores NA en un valor de carácter. Vea más detalles en la página de [Factores](#factors). Por defecto, el nuevo valor es "(Missing)" pero puede ajustarse mediante el argumento na\_level =.

## Imputación

A veces, al analizar los datos, será importante "rellenar los huecos" e imputar los datos que faltan Aunque siempre se puede analizar simplemente unos datos después de eliminar todos los valores que faltan, esto puede causar problemas de muchas maneras. He aquí dos ejemplos:

1. Al eliminar todas las observaciones con valores faltantes o las variables con una gran cantidad de valores faltantes, podría reducir su potencia o su capacidad para realizar algunos tipos de análisis. Por ejemplo, como descubrimos antes, sólo una pequeña fracción de las observaciones de nuestro conjunto de datos de listas de líneas no tiene valores faltantes en todas nuestras variables. Si elimináramos la mayor parte de nuestro conjunto de datos, perderíamos mucha información. Además, la mayoría de nuestras variables tienen una cierta cantidad de valores faltantes; para la mayoría de los análisis, probablemente no sea razonable eliminar todas las variables que tienen muchos valores faltantes.
2. Dependiendo de la razón por la que faltan datos, el análisis de los datos que no faltan podría conducir a resultados sesgados o engañosos. Por ejemplo, como hemos sabido antes, nos faltan datos de algunos pacientes sobre si han tenido algunos síntomas importantes como fiebre o tos. Pero, como una posibilidad, tal vez esa información no se registró para las personas que obviamente no estaban muy enfermas. En ese caso, si elimináramos estas observaciones, estaríamos excluyendo a algunas de las personas más sanas de nuestro conjunto de datos, lo que podría sesgar los resultados.

Es importante pensar en por qué pueden faltar datos, además de ver cuántos faltan. Esto puede ayudarle a decidir la importancia de imputar los datos que faltan, así como el método de imputación de los datos que faltan que puede ser mejor en su situación.

### Tipos de datos que faltan

A continuación se presentan tres tipos generales de datos que faltan:

1. **Falta completamente al azar** (MCAR). Esto significa que no existe ninguna relación entre la probabilidad de que falten datos y cualquiera de las otras variables de los datos. La probabilidad de que falte es la misma para todos los casos. Pero, si tiene una fuerte razón para creer que sus datos son MCAR, analizar sólo los datos no ausentes sin imputar no sesgará sus resultados (aunque puede perder algo de potencia). TODO: considerar la discusión de las pruebas estadísticas para MCAR].
2. Falta al **azar (MAR**). Este nombre es, en realidad, un poco engañoso, ya que MAR significa que los datos faltan de forma sistemática y predecible en función del resto de la información que se tiene. Por ejemplo, puede que todas las observaciones de nuestro conjunto de datos con un valor ausente de fiebre no se hayan registrado porque se asumió que todos los pacientes con escalofríos y dolores tenían fiebre, por lo que nunca se les tomó la temperatura. Si es cierto, podríamos predecir fácilmente que cada observación que falta con escalofríos y dolores también tiene fiebre y utilizar esta información para imputar nuestros datos que faltan. En la práctica, esto es más bien un espectro. Quizá si un paciente tiene escalofríos y dolores, es más probable que también tenga fiebre si no se le toma la temperatura, pero no siempre. Esto sigue siendo predecible aunque no sea perfectamente predecible. Este es un tipo común de valores faltantes
3. **Desaparición no aleatoria** (MNAR). A veces, también se denomina Falta no aleatoria (**NMAR**). Esto supone que la probabilidad de que falte un valor NO es sistemática ni predecible utilizando el resto de la información que tenemos, pero tampoco falta al azar. En esta situación, los datos faltan por razones desconocidas o por razones de las que no se tiene ninguna información. Por ejemplo, en nuestro conjunto de datos puede faltar información sobre la edad porque algunos pacientes muy mayores no saben o se niegan a decir su edad. En esta situación, los datos que faltan sobre la edad están relacionados con el propio valor (y, por tanto, no son aleatorios) y no son predecibles en función del resto de la información que tenemos. El MNAR es complejo y, a menudo, la mejor manera de afrontarlo es intentar recopilar más datos o información sobre el motivo por el que faltan los datos en lugar de intentar imputarlos.

En general, la imputación de datos MCAR suele ser bastante sencilla, mientras que la MNAR es muy difícil, si no imposible. Muchos de los métodos comunes de imputación de datos asumen MAR.

### Paquetes útiles

Algunos paquetes útiles para la imputación de valores faltantes son Mmisc, missForest (que utiliza bosques aleatorios para imputar los valores faltantes) y mice (Imputación multivariada por ecuaciones encadenadas). Para esta sección sólo utilizaremos el paquete mice, que implementa una variedad de técnicas. El mantenedor del paquete mice ha publicado un libro en línea sobre la imputación de valores faltantes que entra en más detalle aquí [(](https://stefvanbuuren.name/fimd/)https://stefvanbuuren.name/fimd/).

Este es el código para cargar el paquete de ratones:

### Imputación de la media

A veces, si está haciendo un análisis simple o tiene una razón de peso para pensar que puede asumir MCAR, puede simplemente establecer los valores numéricos que faltan a la media de esa variable. Tal vez podamos asumir que las mediciones de temperatura que faltan en nuestro conjunto de datos eran MCAR o eran simplemente valores normales. Aquí está el código para crear una nueva variable que reemplaza los valores de temperatura faltantes con el valor medio de la temperatura en nuestro conjunto de datos. Sin embargo, en muchas situaciones reemplazar los datos con la media puede conducir a un sesgo, así que tenga cuidado.

También puede realizar un proceso similar para sustituir los datos categóricos por un valor específico. Para nuestro conjunto de datos, imagine que sabe que todas las observaciones con un valor faltante para su resultado (que puede ser "Muerte" o "Recuperación") son en realidad personas que han muerto (nota: esto no es realmente cierto para este conjunto de datos):

### Imputación de la regresión

Un método algo más avanzado consiste en utilizar algún tipo de modelo estadístico para predecir cuál es el valor que falta y sustituirlo por el valor predicho. Este es un ejemplo de creación de valores predichos para todas las observaciones en las que falta la temperatura, pero no la edad ni la fiebre, mediante una simple regresión lineal que utiliza el estado de la fiebre y la edad en años como predictores. En la práctica, es conveniente utilizar un modelo mejor que este tipo de enfoque simple.

O bien, utilizando el mismo enfoque de modelización a través del paquete de ratones para crear valores imputados para las observaciones de temperatura que faltan:

Este es el mismo tipo de enfoque de algunos métodos más avanzados, como el uso del paquete missForest para sustituir los datos que faltan por valores predichos. En ese caso, el modelo de predicción es un bosque aleatorio en lugar de una regresión lineal. También se pueden utilizar otros tipos de modelos para hacer esto. Sin embargo, aunque este enfoque funciona bien con MCAR, debe tener un poco de cuidado si cree que MAR o MNAR describen con más precisión su situación. La calidad de su imputación dependerá de lo bueno que sea su modelo de predicción e incluso con un modelo muy bueno la variabilidad de sus datos imputados puede estar subestimada.

### LOCF y BOCF

La última observación trasladada (LOCF) y la observación de referencia trasladada (BOCF) son métodos de imputación para datos de series temporales/longitudinales. La idea es tomar el valor observado anterior como reemplazo de los datos que faltan. Cuando faltan varios valores sucesivamente, el método busca el último valor observado.

La función fill() del paquete **tidyr** puede utilizarse para la imputación LOCF y BOCF (sin embargo, otros paquetes como **HMISC**, **zoo** y **data.table** también incluyen métodos para hacerlo). Para mostrar la sintaxis de fill(), crearemos un sencillo conjunto de datos de series temporales que contenga el número de casos de una enfermedad para cada trimestre de los años 2000 y 2001. Sin embargo, falta el valor del año para los trimestres posteriores al primero, por lo que tendremos que imputarlos. La unión fill() también se demuestra en la página [Pivoting data](#pivoting-data).

Nota: asegúrese de que sus datos están correctamente ordenados antes de utilizar la función fill(). fill() rellena por defecto "hacia abajo", pero también puede imputar valores en diferentes direcciones cambiando el parámetro .direction. Podemos hacer unos datos similar en el que el valor del año se registra sólo al final del año y falta para los trimestres anteriores:

En este ejemplo, LOCF y BOCF son claramente lo correcto, pero en situaciones más complicadas puede ser más difícil decidir si estos métodos son apropiados. Por ejemplo, es posible que falten valores de laboratorio para un paciente del hospital después del primer día. A veces, esto puede significar que los valores de laboratorio no cambiaron... ¡pero también podría significar que el paciente se recuperó y sus valores serían muy diferentes después del primer día! Utiliza estos métodos con precaución.

### Imputación múltiple

El libro en línea que mencionamos antes, escrito por el autor del paquete de ratones [(https://stefvanbuuren.name/fimd/),](https://stefvanbuuren.name/fimd/) contiene una explicación detallada de la imputación múltiple y de los motivos por los que conviene utilizarla. Pero, aquí hay una explicación básica del método:

Cuando se realiza una imputación múltiple, se crean múltiples conjuntos de datos con los valores faltantes imputados a valores de datos plausibles (dependiendo de los datos de su investigación, puede querer crear más o menos de estos conjuntos de datos imputados, pero el paquete de ratones establece el número por defecto en 5). La diferencia es que, en lugar de un valor único y específico, cada valor imputado se extrae de una distribución estimada (por lo que incluye cierta aleatoriedad). Como resultado, cada uno de estos conjuntos de datos tendrá valores imputados ligeramente diferentes (sin embargo, los datos no ausentes serán los mismos en cada uno de estos conjuntos de datos imputados). Todavía se utiliza algún tipo de modelo predictivo para hacer la imputación en cada uno de estos nuevos conjuntos de datos (mice tiene muchas opciones para los métodos de predicción, incluyendo Predictive Mean Matching, regresión logística y random forest), pero el paquete mice puede encargarse de muchos de los detalles del modelado.

Entonces, una vez que haya creado estos nuevos conjuntos de datos imputados, puede aplicar cualquier modelo estadístico o análisis que estuviera planeando hacer para cada uno de estos nuevos conjuntos de datos imputados y juntar los resultados de estos modelos. Esto funciona muy bien para reducir el sesgo tanto en MCAR como en muchas configuraciones de MAR y a menudo resulta en estimaciones de error estándar más precisas.

He aquí un ejemplo de aplicación del proceso de Imputación Múltiple para predecir la temperatura en nuestro conjunto de datos del listado utilizando una edad y un estado de fiebre (nuestro conjunto de datos modelo simplificado de arriba):

En este caso, utilizamos el método de imputación por defecto del ratón, que es el de Coincidencia de Medias Predictivas. A continuación, utilizamos estos conjuntos de datos imputados para estimar por separado y luego agrupar los resultados de las regresiones lineales simples en cada uno de estos conjuntos de datos. Hay muchos detalles que hemos pasado por alto y muchas configuraciones que puede ajustar durante el proceso de Imputación Múltiple mientras utiliza el paquete de ratones. Por ejemplo, no siempre tendrá datos numéricos y podría necesitar utilizar otros métodos de imputación (puede seguir utilizando el paquete mice para muchos otros tipos de datos y métodos). Pero, para un análisis más robusto cuando los datos faltantes son una preocupación significativa, la Imputación Múltiple es una buena solución que no siempre es mucho más trabajo que hacer un análisis de caso completo.

## Recursos

Viñeta sobre el [paquete naniar](https://cran.r-project.org/web/packages/naniar/vignettes/getting-started-w-naniar.html)

Galería de [visualizaciones de valores faltantes](https://cran.r-project.org/web/packages/naniar/vignettes/naniar-visualisation.html)

[Libro en línea](https://stefvanbuuren.name/fimd/) sobre imputación múltiple en R por el mantenedor del paquete **mice**

# #Tasas estandarizadas

{#standardised-rates}

Esta página te mostrará dos formas de estandarizarr un resultado, como las hospitalizaciones o la mortalidad, por características como la edad y el sexo.

* Uso del paquete **dsr**
* Uso del paquete **PHEindicatormethods**

Comenzamos demostrando ampliamente los procesos de preparación/limpieza/unión de datos, ya que esto es común cuando se combinan datos de población de múltiples países, datos de población estándar, muertes, etc.

## Resumen

Hay dos formas principales de estandarizar: la estandarización directa y la indirecta. Supongamos que queremos estandarizar la tasa de mortalidad por edad y sexo para el país A y el país B, y comparar las tasas estandarizadas entre estos países.

* Para la estandarización directa, tendrás que conocer el número de la población de riesgo y el número de muertes para cada estrato de edad y sexo, para el país A y el país B. Un estrato en nuestro ejemplo podría ser el de las mujeres entre 15-44 años.
* Para la estandarización indirecta, sólo es necesario conocer el número total de muertes y la estructura por edad y sexo de cada país. Por tanto, esta opción es factible si no se dispone de tasas de mortalidad específicas por edad y sexo o de cifras de población. La estandarización indirecta es, además, preferible en caso de números pequeños por estrato, ya que las estimaciones en la estandarización directa estarían influenciadas por una variación sustancial del muestreo.

## Preparación

Para mostrar cómo se realiza la estandarización, utilizaremos recuentos ficticios de la población y de las defunciones del país A y del país B, por edad (en categorías de 5 años) y por sexo (femenino, masculino). Para que los conjuntos de datos estén listos para su uso, realizaremos los siguientes pasos de preparación:

1. Cargar paquetes
2. Cargar conjuntos de datos
3. Unir los datos de población y mortalidad de los dos países
4. Pivote más largo para que haya una fila por estrato de edad y sexo
5. Limpiar la población de referencia (población estándar mundial) y unirla a los datos del país

En su caso, sus datos pueden tener un formato diferente. Tal vez sus datos sean por provincia, ciudad u otra zona de captación. Puedes que tenga una fila para cada muerte e información sobre la edad y el sexo de cada una (o de una proporción significativa) de estas muertes. En este caso, consulte las páginas sobre [Agrupación dedatos](#grouping-data), [Pivoteo de datos](#pivoting-data) y [Tablas descriptivas](#descriptive-tables) para crear unos datos con recuentos de eventos y población por estrato de edad y sexo.

También necesitamos una población de referencia, la población estándar. Para los fines de este ejercicio utilizaremos la world\_standard\_population\_by\_sex. La población estándar mundial se basa en las poblaciones de 46 países y se elaboró en 1960. Hay muchas poblaciones "estándar"; por ejemplo, el sitio web [del NHS de Escocia](https://www.opendata.nhs.scot/dataset/standard-populations) ofrece bastante información sobre la población estándar europea, la población estándar mundial y la población estándar de Escocia.

### Cargar paquetes

Este trozo de código muestra la carga de los paquetes necesarios para los análisis. En este manual destacamos p\_load() de **pacman**, que instala el paquete si es necesario y lo carga para su uso. También puede cargar los paquetes instalados con library() de R **base.** Consulte la página sobre [los fundamentos de R](#r-basics) para obtener más información sobre los paquetes de R.

**ATENCIÓN:** Si tiene una versión más reciente de R, el paquete **dsr** no puede descargarse directamente de CRAN. Sin embargo, todavía está disponible en el archivo CRAN. Puedes instalar y utilizar éste.

Para los que no son usuarios de Mac:

Para los usuarios de Mac:

### Cargar los datos de la población

Consulte la página de [descarga del manual y los datos](#download-handbook-and-data) para obtener instrucciones sobre cómo descargar todos los datos de ejemplo del manual. Puedes importar los datos de la página de normalización directamente a R desde nuestro repositorio de Github ejecutando los siguientes comandos import():

En primer lugar, cargamos los datos demográficos (recuentos de hombres y mujeres por categoría de edad de 5 años) de los dos países que vamos a comparar, "País A" y "País B".

### Recuento de muertes por carga

Convenientemente, también tenemos los recuentos de las muertes durante el período de interés, por edad y sexo. Los recuentos de cada país están en un archivo separado, que se muestra a continuación.

Muertes en el país A

Muertes en el país B

### Poblaciones limpias y muertes

Necesitamos unir y transformar estos datos de las siguientes maneras:

* Combinar las poblaciones de los países en un solo conjunto de datos y hacer un pivote "largo" para que cada estrato de edad y sexo sea una fila
* Combinar los recuentos de defunciones por país en un solo conjunto de datos y hacer un pivote "largo" para que cada estrato de edad y sexo sea una fila
* Unir las muertes a las poblaciones

En primer lugar, combinamos los conjuntos de datos de las poblaciones de los países, los pivotamos durante más tiempo y realizamos una pequeña limpieza. Para más detalles, consulte la página sobre [el pivoteo de datos](#pivoting-data).

Los datos de población combinados tienen ahora este aspecto (clica para ver los países A y B):

Y ahora realizamos operaciones similares en los dos conjuntos de datos de muertes.

Los datos de las muertes tienen ahora este aspecto, y contienen datos de ambos países:

Ahora unimos los datos de defunciones y población basándonos en las columnas comunes País, age\_cat5 y Sexo. Esto añade la columna Defunciones.

Ahora podemos clasificar Sexo, age\_cat5 y País como factores y establecer el orden de los niveles utilizando la función fct\_relevel() del paquete **forcats**, como se describe en la página sobre [Factores](#factors). Ten en cuenta que la clasificación de los niveles de los factores no cambia visiblemente los datos, pero el comando arrange() los ordena por País, categoría de edad y sexo.

**PRECAUCIÓN:** Si tiene pocas muertes por estrato, considere la posibilidad de utilizar categorías de 10 o 15 años, en lugar de categorías de 5 años para la edad.

### Carga de la población de referencia

Por último, para la estandarización directa, importamos la población de referencia (la "población estándar" mundial por sexo)

### Población de referencia limpia

Los valores de la categoría de edad en los dataframes country\_data y standard\_pop\_data tendrán que ser alineados.

Actualmente, los valores de la columna age\_cat5 del dataframe standard\_pop\_data contienen la palabra "years" y "plus", mientras que los del dataframe country\_data no. Tendremos que hacer coincidir los valores de la categoría de edad. Usamos str\_replace\_all() del paquete **stringr**, como se describe en la página sobre [Caracteres y cadenas](#characters-and-strings), para reemplazar estos patrones sin espacio "".

Además, el paquete **dsr** espera que en la población estándar, la columna que contiene los recuentos se llame "pop". Así que cambiamos el nombre de esa columna en consecuencia.

**ATENCIÓN:** Si intenta utilizar str\_replace\_all() para eliminar un símbolo de suma, no funcionará porque es un símbolo especial. "Escapa" de los especiales poniendo dos barras invertidas delante, como en str\_replace\_call(columna, "\\+", "").

### Crear unos datos con una población estándar

Por último, el paquete **PHEindicatormethods**, que se detalla [a continuación](#standard_phe), espera que las poblaciones estándar se unan a los recuentos de eventos y poblaciones del país. Por lo tanto, crearemos unos datos all\_data con ese fin.

Este conjunto de datos completo tiene el siguiente aspecto:

## paquete ****dsr****

A continuación mostramos el cálculo y la comparación de tasas estandarizadas directamente utilizando el paquete **dsr**. El paquete **dsr permite calcular y comparar tasas estandarizadas** directamente (¡no hay tasas estandarizadas indirectamente!).

En la sección de preparación de datos, hemos creado conjuntos de datos separados para los recuentos de países y la población estándar:

1. el objeto country\_data, que es una tabla de población con el número de habitantes y el número de muertes por estrato por país
2. el objeto standard\_pop\_clean, que contiene el número de población por estrato para nuestra población de referencia, la población estándar mundial

Utilizaremos estos conjuntos de datos separados para el enfoque **dsr**.

### Tasas estandarizadas

A continuación, calculamos las tasas por país directamente estandarizadas por edad y sexo. Utilizamos la función dsr().

Cabe destacar que dsr() espera un dataframe para las poblaciones de los países y los recuentos de eventos (muertes), y un dataframe ***separado*** con la población de referencia. También espera que en este conjunto de datos de la población de referencia el nombre de la columna de la unidad de tiempo sea "pop" (lo aseguramos en la sección de preparación de datos).

Hay muchos argumentos, como se anota en el código siguiente. En particular, el evento = se establece en la columna Muertes, y el fu = ("seguimiento") se establece en la columna Población. Establecemos los subgrupos de comparación como la columna País y estandarizamos en base a age\_cat5 y Sexo. A estas dos últimas columnas no se les asigna un argumento con nombre concreto. Consulte ?dsr para obtener más detalles.

Arriba, vemos que mientras el país A tenía una tasa de mortalidad bruta más baja que el país B, tiene una tasa estandarizada más alta después de la estandarización directa por edad y sexo.

### Ratios de tasas estandarizadas

La tasa de mortalidad estandarizada es 1,22 veces mayor en el país A en comparación con el país B (IC del 95%: 1,17-1,27).

### Diferencia de tasas estandarizadas

El país A tiene 4,24 muertes adicionales por cada 100.000 habitantes (IC del 95%: 3,24-5,24) en comparación con el país A.

## Paquete ****PHEindicatormethods****

Otra forma de calcular las tasas estandarizadas es con el paquete **PHEindicatormethods**. Este paquete permite calcular tanto directa como indirectamente las tasas estandarizadas. Mostraremos ambos métodos.

En esta sección se utilizará el dataframe all\_data creado al final de la sección Preparación. Este dataframe incluye las poblaciones de los países, los eventos de muerte y la población de referencia mundial estándar. Puedes verlo [aquí](#standard_all).

### Tasas estandarizadas directamente

A continuación, primero agrupamos los datos por país y luego los pasamos a la función phe\_dsr() para obtener directamente las tasas estandarizadas por país.

Cabe destacar que la población de referencia (estándar) puede proporcionarse como una **columna dentro del dataframe específico del país** o como un **vector separado**. Si se proporciona dentro del dataframe específico del país, hay que establecer stdpoptype = "field". Si se proporciona como un vector, hay que establecer stdpoptype = "vector". En este último caso, hay que asegurarse de que el orden de las filas por estratos es similar tanto en el dataframe específico del país como en la población de referencia, ya que los registros se emparejarán por posición. En nuestro ejemplo siguiente, proporcionamos la población de referencia como una columna dentro del dataframe específico del país.

Consulte la ayuda de ?phr\_dsr o los enlaces de la sección Referencias para obtener más información.

### Tasas estandarizadas indirectamente

Para la estandarización indirecta, se necesita una población de referencia con el número de muertes y el número de población por estrato. En este ejemplo, calcularemos las tasas del país A utilizando el país B como población de referencia, ya que la población de referencia de standard\_pop\_clean no incluye el número de muertes por estrato.

A continuación, creamos primero la población de referencia del país B. Luego, pasamos los datos de mortalidad y población del país A, los combinamos con la población de referencia y los pasamos a la función phe\_isr(), para obtener tasas estandarizadas indirectamente. Por supuesto, también se puede hacer a la inversa.

En nuestro ejemplo, la población de referencia se proporciona como un dataframe separado. En este caso, nos aseguramos de que los vectores x =, n =, x\_ref = y n\_ref = estén ordenados por los mismos valores de categoría de normalización (estrato) que los de nuestro dataframe específico del país, ya que los registros se emparejarán por posición.

Consulte la ayuda de ?phr\_isr o los enlaces de la sección Referencias para obtener más información.

## Recursos

Si desea ver otro ejemplo reproducible utilizando **dsr**, consulte [esta viñeta](https://mran.microsoft.com/snapshot/2020-02-12/web/packages/dsr/vignettes/dsr.html)

Si desea ver otro ejemplo en el que se utilizan **los métodos de PHEindicator**, visite [este sitio web](https://mran.microsoft.com/snapshot/2018-10-22/web/packages/PHEindicatormethods/vignettes/IntroductiontoPHEindicatormethods.html)

Ver el [archivo pdf de referencia de](https://cran.r-project.org/web/packages/PHEindicatormethods/PHEindicatormethods.pdf) **los métodos de indicación de PHE**

# #Medias móviles

{#moving-averages}

En esta página se tratan dos métodos para calcular y visualizar las medias móviles:

1. Calcular con el paquete **deslizante**
2. Calcular dentro de un comando ggplot() con el paquete **tidyquant**

## Preparación

### Cargar paquetes

Este trozo de código muestra la carga de los paquetes necesarios para los análisis. En este manual destacamos p\_load() de **pacman**, que instala el paquete si es necesario y lo carga para su uso. También puede cargar los paquetes instalados con library() de R **base.** Consulte la página sobre [los fundamentos de R](#r-basics) para obtener más información sobre los paquetes de R.

### Importar datos

Importamos los datos de casos de una epidemia de ébola simulada. Si quiere seguir el proceso, [clica para descargar linelist "limpia"](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/case_linelists/linelist_cleaned.rds) (como archivo .rds). Importe los datos con la función import() del paquete **rio** (maneja muchos tipos de archivos como .xlsx, .csv, .rds - vea la página de [importación y exportación](#import-and-export) para más detalles).

A continuación se muestran las primeras 50 filas del listado.

## Calcular con el ****deslizador****

**Utiliza este enfoque para calcular una media móvil en un dataframe antes de trazarla.**

El paquete **slider** proporciona varias funciones de "ventana deslizante" para calcular medias móviles, sumas acumulativas, regresiones móviles, etc. Trata un dataframe como un vector de filas, permitiendo la iteración por filas sobre un dataframe.

Estas son algunas de las funciones más comunes:

* slide\_dbl() - itera a través de una columna numérica (de ahí "\_dbl") realizando una operación mediante una ventana deslizante
  + slide\_sum() - función abreviada de suma rodante para slide\_dbl()
  + slide\_mean() - función abreviada de media móvil para slide\_dbl()
* slide\_index\_dbl() - aplica la ventana rodante en una columna numérica utilizando una columna separada para indexar la progresión de la ventana (útil si se rueda por fecha con algunas fechas ausentes)
  + slide\_index\_sum() - Función abreviada de suma rodante con indexación
  + slide\_index\_mean() - Función de acceso directo a la media móvil con indexación

El paquete **deslizante** tiene muchas otras funciones que se tratan en la sección de Recursos de esta página. Tocamos brevemente las más comunes.

**Argumentos básicos**

* .x, el primer argumento por defecto, es el vector sobre el que iterar y al que aplicar la función
* .i = para las versiones de "índice" de las funciones de **deslizamiento** - proporciona una columna para "indexar" el rollo (véase la sección [siguiente](#roll_index))
* .f =, el segundo argumento por defecto, bien:
  + Una función, escrita sin paréntesis, como media, o
  + Una fórmula, que se convertirá en una función. Por ejemplo ~ .x - mean(.x) devolverá el resultado del valor actual menos la media del valor de la ventana
* Para más detalles, consulte este [material de referencia](https://davisvaughan.github.io/slider/reference/slide.html)

**Tamaño de la ventana**

Especifique el tamaño de la ventana utilizando los argumentos .before, .after o ambos:

* .before = - Proporcionar un número entero
* .after = - Proporcionar un número entero
* .complete = - Ponga este valor a TRUE si sólo quiere que se realicen cálculos en ventanas completas

Por ejemplo, para conseguir una ventana de 7 días que incluya el valor actual y los seis anteriores, utiliza .before = 6. Para conseguir una ventana "centrada" proporcione el mismo número tanto a .before = como a .after =.

Por defecto, .complete = será FALSE por lo que si la ventana completa de filas no existe, las funciones utilizarán las filas disponibles para realizar el cálculo. Si se ajusta a TRUE, los cálculos sólo se realizan en ventanas completas.

**Ventana expansiva**

Para lograr operaciones acumulativas, establezca el argumento .before = en Inf. Esto realizará la operación sobre el valor actual y todos los que vengan antes.

### Enrollar por fecha

El caso más probable de uso de un cálculo rotativo en epidemiología aplicada es examinar una métrica a lo largo del tiempo. Por ejemplo, una medición continua de la incidencia de casos, basada en el recuento diario de casos.

Si tiene datos de series temporales limpios con valores para cada fecha, puede estar bien utilizar slide\_dbl(), como se demuestra aquí en la página de [series temporales y detección de brotes](#timeseries_moving).

Sin embargo, en muchas circunstancias de epidemiología aplicada puede haber fechas ausentes en sus datos, donde no hay eventos registrados. En estos casos, es mejor utilizar las versiones "índice" de las funciones **deslizantes**.

### Datos indexados

A continuación, mostramos un ejemplo utilizando slide\_index\_dbl() en la lista de casos. Digamos que nuestro objetivo es calcular una incidencia rodante de 7 días - la suma de casos utilizando una ventana rodante de 7 días. Si está buscando un ejemplo de media rodante, vea la sección de abajo sobre [rodamiento agrupado](#roll_slider_group).

Para empezar, se crean los datos daily\_counts para reflejar los recuentos diarios de casos del listado, calculados con count() de **dplyr**.

Aquí está el dataframe de daily\_counts - hay nrow(daily\_counts) filas, cada día está representado por una fila, pero especialmente al principio de la epidemia algunos días no están presentes (no hubo casos admitidos en esos días).

Es crucial reconocer que una función estándar de balanceo (como slide\_dbl() utilizaría una ventana de 7 filas, no de 7 días. Por lo tanto, si hay fechas ausentes, ¡algunas ventanas se extenderán realmente más de 7 días naturales!

Se puede conseguir una ventana móvil "inteligente" con slide\_index\_dbl(). El "índice" significa que la función utiliza una columna independiente como "índice" para la ventana móvil. La ventana no se basa simplemente en las filas del dataframe.

Si la columna índice es una fecha, tiene la posibilidad añadida de especificar la extensión de la ventana a .before = y/o .after = en unidades de **lubridate** days() o months(). Si hace estas cosas, la función incluirá los días ausentes en las ventanas como si estuvieran allí (como valores NA).

Mostremos una comparación. A continuación, calculamos la incidencia móvil de casos de 7 días con ventanas regulares e indexadas.

Obsérvese cómo en la columna normal de las 7 primeras filas el recuento aumenta constantemente a pesar de que las filas no tienen 7 días de diferencia. La columna adyacente "indexada" tiene en cuenta estos días naturales ausentes, por lo que sus sumas de 7 días son mucho menores, al menos en este periodo de la epidemia en el que los casos están más alejados.

Ahora puede trazar estos datos utilizando ggplot():

### Rodando por el grupo

Si agrupa sus datos antes de utilizar una función **deslizante**, las ventanas deslizantes se aplicarán por grupo. Tenga cuidado de organizar sus filas en el orden deseado por grupo.

Cada vez que se inicia un nuevo grupo, la ventana deslizante se reinicia. Por lo tanto, un matiz a tener en cuenta es que si sus datos están agrupados y ha establecido .complete = TRUE, tendrá valores vacíos en cada transición entre grupos. A medida que la función se desplaza hacia abajo a través de las filas, cada transición en la columna de agrupación reiniciará la acumulación del tamaño mínimo de la ventana para permitir un cálculo.

Consulte la página del manual sobre [Agrupar datos](#grouping-data) para obtener detalles sobre la agrupación de datos.

A continuación, contamos los casos del listado por fecha y por hospital. Luego ordenamos las filas en orden ascendente, primero ordenando por hospital y luego dentro de éste por fecha. A continuación establecemos group\_by(). Entonces podemos crear nuestra nueva media móvil.

Aquí está el nuevo conjunto de datos:

Ahora podemos trazar las medias móviles, mostrando los datos por grupo especificando ~ hospital a facet\_wrap() en ggplot(). Para divertirnos, trazamos dos geometrías: una geom\_col() que muestra los recuentos de casos diarios y una geom\_line() que muestra la media móvil de 7 días.

**PELIGRO:** Si obtiene un error que dice "slide() fue obsoleta en tsibble 0.9.0 y ahora está desaparecida. Por favor, utiliza slider::slide() en su lugar", significa que la función slide() del paquete **tsibble** está enmascarando la función slide() del paquete **slider**. Solucione esto especificando el paquete en el comando, como slider::slide\_dbl().

## Calcular con ****tidyquant**** dentro de ggplot()

El paquete **tidyquant** ofrece otro enfoque para calcular las medias móviles, esta vez desde el propio comando ggplot().

Debajo del listado, los datos se cuentan por fecha de inicio, y esto se traza como una línea descolorida (alfa < 1). Encima hay una línea creada con geom\_ma() del paquete **tidyquant**, con una ventana de 7 días (n = 7) con el color y el grosor especificados.

Por defecto geom\_ma() utiliza una media móvil simple (ma\_fun = "SMA"), pero se pueden especificar otros tipos, como:

* "EMA" - media móvil exponencial (más peso a las observaciones recientes)
* "WMA" - media móvil ponderada (los wts se utilizan para ponderar las observaciones en la media móvil)
* Otros se pueden encontrar en la documentación de la función

Consulte esta [viñeta](https://cran.r-project.org/web/packages/tidyquant/vignettes/TQ04-charting-with-tidyquant.html) para obtener más detalles sobre las opciones disponibles en **tidyquant**.

## Recursos

Vea la útil [viñeta](https://cran.r-project.org/web/packages/slider/vignettes/slider.html) en línea [para elpaquete de**deslizadores**](https://cran.r-project.org/web/packages/slider/vignettes/slider.html)

La [página github del](https://github.com/DavisVaughan/slider) **deslizador**

Una [viñeta](https://davisvaughan.github.io/slider/articles/slider.html) **deslizante**

[viñeta tidyquant](https://cran.r-project.org/web/packages/tidyquant/vignettes/TQ04-charting-with-tidyquant.html)

Si tu caso de uso requiere que te "saltes" los fines de semana e incluso los días festivos, puede que te guste el paquete **almanaque**.

# #Series temporales y detección de brotes

{#time-series-and-outbreak-detection}

## Resumen

Esta página demuestra el uso de varios paquetes para el análisis de series temporales. Principalmente se basa en paquetes de la familia [**tidyverts**](https://tidyverts.org/), pero también utilizará el paquete RECON [**trending**](https://github.com/reconhub/trending) para ajustar modelos más apropiados para la epidemiología de enfermedades infecciosas.

Ten en cuenta que en el siguiente ejemplo utilizamos unos datos del paquete de **vigilancia** sobre Campylobacter en Alemania (véase el [capítulo de datos](https://epirhandbook.com/download-handbook-and-data.html) del manual para más detalles). Sin embargo, si desea ejecutar el mismo código en unos datos con múltiples países u otros estratos, hay una plantilla de código de ejemplo para esto en el [repo de github de r4epis](https://github.com/R4EPI/epitsa).

Los temas que se tratan son:

1. Datos de series temporales
2. Análisis descriptivo
3. Ajuste de regresiones
4. Relación de dos series temporales
5. Detección de brotes
6. Series temporales interrumpidas

## Preparación

### Paquetes

Este trozo de código muestra la carga de los paquetes necesarios para los análisis. En este manual destacamos p\_load() de **pacman**, que instala el paquete si es necesario y lo carga para su uso. También se pueden cargar paquetes con library() desde **el** R **base.** Consulte la página sobre [los fundamentos de R](https://epirhandbook.com/r-basics.html) para obtener más información sobre los paquetes de R.

### Datos de carga

Puedes descargar todos los datos utilizados en este manual mediante las instrucciones de la página de [descarga del manual y los datos](#download-handbook-and-data).

Los datos de ejemplo utilizado en esta sección son los recuentos semanales de casos de campylobacter notificados en Alemania entre 2001 y 2011. [Puedes clicar aquí para descargar este archivo de datos (.xlsx).](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/time_series/campylobacter_germany.xlsx)

Este conjunto de datos es una versión reducida de los datos disponibles en el paquete de [**surveillance**](https://cran.r-project.org/web/packages/surveillance/). (para más detalles, cargue el paquete de vigilancia y consulte ?campyDE)

Importe estos datos con la función import() del paquete **rio** (maneja muchos tipos de archivos como .xlsx, .csv, .rds - vea la página de [importación y exportación](#import-and-export) para más detalles).

A continuación se muestran las 10 primeras filas de los recuentos.

### Datos limpios

El código siguiente se asegura de que la columna de la fecha tenga el formato adecuado. Para esta pestaña utilizaremos el paquete tsibble y, por tanto, la función yearweek se utilizará para crear una variable de semana de calendario. Hay otras maneras de hacer esto (ver la página de [Trabajo con fechas](https://epirhandbook.com/working-with-dates.html) para más detalles), sin embargo para las series de tiempo es mejor mantenerse dentro de un marco (**tsibble**).

### Descargar datos climáticos

En la sección de relación de dos series temporales de esta página, compararemos los recuentos de casos de campylobacter con los datos climáticos.

Los datos climáticos de cualquier parte del mundo pueden descargarse del satélite Copérnico de la UE. No se trata de mediciones exactas, sino que se basan en un modelo (similar a la interpolación), pero la ventaja es la cobertura horaria global, así como las previsiones.

Puedes descargar cada uno de estos archivos de datos climáticos en la página de [descarga de manuales y datos](#download-handbook-and-data).

Para propósitos de demostración aquí, mostraremos el código R para usar el paquete **ecmwfr** para extraer estos datos del almacén de datos climáticos de Copernicus. Es necesario crear una cuenta gratuita para que esto funcione. El sitio web del paquete tiene una [guía](https://github.com/bluegreen-labs/ecmwfr" \l "use-copernicus-climate-data-store-cds) útil de cómo hacerlo. A continuación se muestra un código de ejemplo de cómo hacer esto, una vez que tienes las claves de la API adecuada. Tienes que sustituir las X de abajo por los ID de tu cuenta. Tendrá que descargar un año de datos a la vez, de lo contrario el servidor se queda sin tiempo.

Si no estás seguro de las coordenadas de un lugar del que quieres descargar datos, puedes utilizar el paquete **tmaptools** para obtener las coordenadas de los mapas de calles abiertos. Una opción alternativa es el paquete [**photon**](https://github.com/rCarto/photon), aunque todavía no se ha publicado en CRAN; lo bueno de **photon** es que proporciona más datos contextuales para cuando hay varias coincidencias en la búsqueda.

### Cargar datos climáticos

Tanto si ha descargado los datos climáticos a través de nuestro manual, como si ha utilizado el código anterior, ahora debería tener 10 años de archivos de datos climáticos ".nc" almacenados en la misma carpeta de su ordenador.

Utiliza el siguiente código para importar estos archivos en R con el paquete **stars**.

Una vez importados estos archivos como datos del objeto, los convertiremos en un dataframe.

## Datos de series temporales

Existen varios paquetes para estructurar y manejar los datos de las series temporales. Como ya hemos dicho, nos centraremos en la familia de paquetes **tidyverts** y, por tanto, utilizaremos el paquete **tsibble** para definir nuestro objeto de serie temporal. Tener unos datos definido como objeto de serie temporal significa que es mucho más fácil estructurar nuestro análisis.

Para ello utilizamos la función tsibble() y especificamos el "índice", es decir, la variable que especifica la unidad de tiempo de interés. En nuestro caso se trata de la variable epiweek.

Si tuviéramos unos datos con recuentos semanales por provincia, por ejemplo, también podríamos especificar la variable de agrupación utilizando el argumento clave =. Esto nos permitiría hacer un análisis para cada grupo.

Si observamos el tipo(counts), veremos que, además de ser un dataframe ordenado ("tbl\_df", "tbl", "data.frame"), tiene las propiedades adicionales de un dataframe de series temporales ("tbl\_ts").

Se puede echar un vistazo rápido a los datos utilizando **ggplot2**. En el gráfico vemos que hay un claro patrón estacional y que no hay pérdidas. Sin embargo, parece haber un problema con la notificación al principio de cada año; los casos descienden en la última semana del año y luego aumentan en la primera semana del año siguiente.

**PELIGRO: La** mayoría de los conjuntos de datos no son tan limpios como este ejemplo. Tendrá que comprobar si hay duplicados y faltas como se indica a continuación.

### Duplicados

**tsibble** no permite observaciones duplicadas. Así que cada fila deberá ser única, o única dentro del grupo (variable clave). El paquete tiene algunas funciones que ayudan a identificar los duplicados. Entre ellas se encuentran are\_duplicated(), que proporciona un vector TRUE/FALSE para saber si la fila es un duplicado, y duplicates(), que proporciona un dataframe de las filas duplicadas.

Consulte la página sobre [Desduplicación](https://epirhandbook.com/de-duplication.html) para obtener más detalles sobre cómo seleccionar las filas que desee.

### Desaparecidos

En nuestra breve inspección anterior hemos visto que no hay faltas, pero también hemos visto que parece haber un problema de retraso en la notificación en torno al año nuevo. Una forma de abordar este problema podría ser establecer estos valores como faltantes y luego imputar los valores. La forma más sencilla de imputación de series temporales consiste en trazar una línea recta entre el último valor no faltante y el siguiente valor no faltante. Para ello, utilizaremos la función na\_interpolation() del paquete **imputeTS**.

Consulte la página de [datos faltantes](https://epirhandbook.com/missing-data.html) para conocer otras opciones de imputación.

Otra alternativa sería calcular una media móvil para intentar suavizar estos aparentes problemas de información (véase la siguiente sección y la página sobre [medias móviles](https://epirhandbook.com/moving-averages.html)).

## Análisis descriptivo

### Medias móviles

Si los datos tienen mucho ruido (los recuentos suben y bajan), puede ser útil calcular una media móvil. En el ejemplo siguiente, para cada semana se calcula la media de casos de las cuatro semanas anteriores. Esto suaviza los datos para hacerlos más interpretables. En nuestro caso esto no aporta mucho, así que nos quedaremos con los datos interpolados para el análisis posterior. Véase la página de [medias móviles](https://epirhandbook.com/moving-averages.html) para más detalles.

### Periodicidad

A continuación definimos una función personalizada para crear un periodograma. Consulte la página [Escribir funciones](#writing-functions-1) para obtener información sobre cómo escribir funciones en R.

En primer lugar, se define la función. Sus argumentos incluyen unos datos con un recuento de columnas, start\_week = que es la primera semana de los datos, un número para indicar cuántos períodos por año (por ejemplo, 52, 12) y, por último, el estilo de salida (véanse los detalles en el código siguiente).

**NOTA: Es posible** utilizar las semanas anteriores para añadirlas a los términos del seno y del coseno, sin embargo, utilizaremos una función para generar estos términos (véase la sección de regresión más adelante)

### Descomposición

La descomposición clásica se utiliza para desglosar una serie temporal en varias partes, que en conjunto conforman el patrón que se observa. Estas diferentes partes son:

* La tendencia-ciclo (la dirección a largo plazo de los datos)
* La estacionalidad (patrones repetitivos)
* El azar (lo que queda después de quitar la tendencia y la temporada)

### Autocorrelación

La autocorrelación le informa de la relación entre los recuentos de cada semana y las semanas anteriores (denominadas rezagos).

Utilizando la función ACF(), podemos producir un gráfico que nos muestre un número de líneas para la relación en diferentes rezagos. Cuando el retardo es 0 (x = 0), esta línea sería siempre 1, ya que muestra la relación entre una observación y ella misma (no se muestra aquí). La primera línea mostrada aquí (x = 1) muestra la relación entre cada observación y la observación anterior (retardo de 1), la segunda muestra la relación entre cada observación y la observación anterior (retardo de 2) y así sucesivamente hasta el retardo de 52 que muestra la relación entre cada observación y la observación de 1 año (52 semanas antes).

El uso de la función PACF() (para la autocorrelación parcial) muestra el mismo tipo de relación pero ajustada para todas las demás semanas intermedias. Esto es menos informativo para determinar la periodicidad.

Puedes probar formalmente la hipótesis nula de independencia en una serie temporal (es decir, que no está autocorrelacionada) utilizando la prueba de Ljung-Box (en el paquete **stats**). Un valor p significativo sugiere que hay autocorrelación en los datos.

## Ajuste de regresiones

Es posible ajustar un gran número de regresiones diferentes a una serie temporal, sin embargo, aquí demostraremos cómo ajustar una regresión binomial negativa, ya que suele ser la más apropiada para los datos de recuento en las enfermedades infecciosas.

### Términos de Fourier

Los términos de Fourier son el equivalente a las curvas sin y cosin. La diferencia es que éstos se ajustan basándose en la búsqueda de la combinación de curvas más adecuada para explicar sus datos.

Si sólo se ajusta un término de fourier, esto equivaldría a ajustar un sin y un cosin para el desfase más frecuente que se ve en el periodograma (en nuestro caso, 52 semanas). Utilizamos la función fourier() del paquete **forecast**.

En el código de abajo asignamos usando el $, ya que fourier() devuelve dos columnas (una para el pecado y otra para el coseno) y así se añaden al conjunto de datos como una lista, llamada "fourier" - pero esta lista se puede usar como una variable normal en la regresión.

### Binomio negativo

Es posible ajustar las regresiones utilizando las funciones **estadísticas** básicas o **MASS** (por ejemplo, lm(), glm() y glm.nb()). Sin embargo, utilizaremos las del paquete de **tendencias**, ya que esto permite calcular intervalos de confianza y predicción adecuados (que de otro modo no están disponibles). La sintaxis es la misma, y se especifica una variable de resultado, luego una tilde (~) y luego se añaden las diversas variables de exposición de interés separadas por un signo de más (+).

La otra diferencia es que primero definimos el modelo y luego lo ajustamos a los datos. Esto es útil porque permite comparar varios modelos diferentes con la misma sintaxis.

**CONSEJO:** Si desea utilizar tasas, en lugar de recuentos, puede incluir la variable de población como un término de compensación logarítmica, añadiendo offset(log(población). Entonces tendría que establecer que la población es 1, antes de usar predict() para producir una tasa.

**SUGERENCIA:** Para ajustar modelos más complejos, como ARIMA o prophet, consulte el paquete [**fable**](https://fable.tidyverts.org/index.html).

### Residuos

Para ver si nuestro modelo se ajusta a los datos observados, tenemos que observar los residuos. Los residuos son la diferencia entre los recuentos observados y los recuentos estimados a partir del modelo. Podríamos calcularlo simplemente utilizando case\_int - estimate, pero la función residuals() lo extrae directamente de la regresión por nosotros.

Lo que vemos a continuación es que no estamos explicando toda la variación que podríamos con el modelo. Es posible que debamos ajustar más términos de Fourier y abordar la amplitud. Sin embargo, para este ejemplo lo dejaremos como está. Los gráficos muestran que nuestro modelo es peor en los picos y en los valles (cuando los recuentos son los más altos y los más bajos) y que es más probable que subestime los recuentos observados.

## Relación de dos series temporales

En este caso, analizamos el uso de los datos meteorológicos (concretamente la temperatura) para explicar los recuentos de casos de campylobacter.

### Fusión de conjuntos de datos

Podemos unir nuestros conjuntos de datos utilizando la variable semana. Para obtener más información sobre la fusión, consulte la sección del manual sobre la [unión](https://epirhandbook.com/joining-data.html).

### Análisis descriptivo

En primer lugar, traza los datos para ver si hay alguna relación evidente. El siguiente gráfico muestra que hay una clara relación en la estacionalidad de las dos variables, y que la temperatura puede alcanzar su punto máximo unas semanas antes que el número de casos. Para más información sobre el pivoteo de datos, consulte la sección del manual sobre [pivoteo de datos](https://epirhandbook.com/pivoting-data.html).

### Retrasos y correlación cruzada

Para comprobar formalmente qué semanas están más relacionadas entre los casos y la temperatura. Podemos utilizar la función de correlación cruzada (CCF()) del paquete de **fiestas**. También se podría visualizar (en lugar de utilizar arrange) utilizando la función autoplot().

Vemos que un desfase de 4 semanas es el más correlacionado, por lo que creamos una variable de temperatura retardada para incluirla en nuestra regresión.

**PELIGRO:** Ten en cuenta que las primeras cuatro semanas de nuestros datos en la variable de temperatura rezagada faltan (NA) - ya que no hay cuatro semanas anteriores para obtener datos. Para utilizar este conjunto de datos con la función de **tendencia predict**(), necesitamos utilizar el argumento simulate\_pi = FALSE dentro de predict() más abajo. Si quisiéramos utilizar la opción de simulación, entonces tenemos que eliminar estas pérdidas y almacenarlas como un nuevo conjunto de datos añadiendo drop\_na(t2m\_lag4) al fragmento de código que aparece a continuación.

### Binomio negativo con dos variables

Ajustamos una regresión binomial negativa como se hizo anteriormente. Esta vez añadimos la variable de temperatura con un retraso de cuatro semanas.

**ATENCIÓN:** Observe el uso de simulate\_pi = FALSE dentro del argumento de predict(). Esto se debe a que el comportamiento por defecto de la **tendencia** es utilizar el paquete **ciTools** para estimar un intervalo de predicción. Esto no funciona si hay recuentos NA, y también produce intervalos más granulares. Véase ?trending::predict.trending\_model\_fit para más detalles.

Para investigar los términos individuales, podemos sacar la regresión binomial negativa original del formato de **tendencia** utilizando get\_model() y pasarla a la función tidy() del paquete **broom** para recuperar las estimaciones exponenciadas y los intervalos de confianza asociados.

Lo que esto nos muestra es que la temperatura retardada, tras controlar la tendencia y la estacionalidad, es similar a los recuentos de casos (estimación ~ 1) y está significativamente asociada. Esto sugiere que podría ser una buena variable para predecir el número de casos futuros (ya que las previsiones climáticas están disponibles).

Una rápida inspección visual del modelo muestra que podría hacer un mejor trabajo de estimación de los recuentos de casos observados.

#### Residuos

Volvemos a investigar los residuos para ver si nuestro modelo se ajusta a los datos observados. Los resultados y la interpretación aquí son similares a los de la regresión anterior, por lo que puede ser más factible quedarse con el modelo más simple sin temperatura.

## Detección de brotes

Aquí mostraremos dos métodos (similares) de detección de brotes. El primero se basa en las secciones anteriores. Utilizamos el paquete de **tendencias** para ajustar las regresiones a los años anteriores, y luego predecir lo que esperamos ver en el año siguiente. Si los recuentos observados están por encima de lo que esperamos, esto podría sugerir que hay un brote. El segundo método se basa en principios similares, pero utiliza el paquete de **vigilancia**, que tiene varios algoritmos diferentes para la detección de aberraciones.

**ATENCIÓN:** Normalmente, estás interesado en el año actual (donde sólo conoce los recuentos hasta la semana actual). Así que en este ejemplo pretendemos estar en la semana 39 de 2011.

### paquete de ****tendencias****

Para este método definimos una línea de base (que normalmente debería ser de unos 5 años de datos). Ajustamos una regresión a los datos de referencia y la utilizamos para predecir las estimaciones del año siguiente.

#### Fecha límite

Es más fácil definir las fechas en un lugar y luego utilizarlas en el resto del código.

Aquí definimos una fecha de inicio (cuando comenzaron nuestras observaciones) y una fecha de corte (el final de nuestro período de referencia - y cuando comienza el período que queremos predecir). También definimos cuántas semanas hay en nuestro año de interés (el que vamos a predecir)~. También definimos cuántas semanas hay entre nuestra fecha límite de referencia y la fecha final para la que nos interesa predecir.

**NOTA:** En este ejemplo pretendemos estar actualmente a finales de septiembre de 2011 ("2011 W39").

#### Añadir filas

Para poder pronosticar en un formato tidyverse, necesitamos tener el número correcto de filas en nuestro conjunto de datos, es decir, una fila por cada semana hasta la fecha final definida anteriormente. El código siguiente permite añadir estas filas por una variable de agrupación - por ejemplo, si tuviéramos varios países en unos datos, podríamos agrupar por país y luego añadir filas apropiadas para cada uno. La función group\_by\_key() de **tsibble** nos permite hacer esta agrupación y luego pasar los datos agrupados a las funciones de **dplyr**, group\_modify() y add\_row(). Luego especificamos la secuencia de semanas entre una después de la semana máxima disponible actualmente en los datos y la semana final.

#### Términos de Fourier

Tenemos que redefinir nuestros términos de fourier, ya que queremos ajustarlos sólo a la fecha de referencia y luego predecir (extrapolar) esos términos para el año siguiente. Para ello tenemos que combinar dos listas de salida de la función fourier() juntas; la primera es para los datos de referencia, y la segunda predice para el año de interés (definiendo el argumento h).

N.B. para enlazar filas tenemos que usar rbind() (en lugar de tidyverse bind\_rows) ya que las columnas de fourier son una lista (por lo que no se nombran individualmente).

#### Dividir los datos y ajustar la regresión

Ahora tenemos que dividir nuestro conjunto de datos en el período de referencia y el período de predicción. Esto se hace utilizando la función **dplyr** group\_split() después de group\_by(), y creará una lista con dos dataframes, uno para antes de su corte y otro para después.

A continuación, utilizamos la función pluck() del paquete **purrr** para extraer los conjuntos de datos de la lista (lo que equivale a utilizar corchetes, por ejemplo, dat[[1]]), y podemos ajustar nuestro modelo a los datos de referencia, y luego utilizar la función predict() para nuestros datos de interés después del corte.

Consulta la página sobre [Iteración, bucles y listas](#iteration-loops-and-lists) para saber más sobre **purrr**.

**ATENCIÓN:** Observe el uso de simulate\_pi = FALSE dentro del argumento de predict(). Esto se debe a que el comportamiento por defecto de la **tendencia** es utilizar el paquete **ciTools** para estimar un intervalo de predicción. Esto no funciona si hay recuentos NA, y también produce intervalos más granulares. Véase ?trending::predict.trending\_model\_fit para más detalles.

Como anteriormente, podemos visualizar nuestro modelo con **ggplot**. Resaltamos las alertas con puntos rojos para los recuentos observados por encima del intervalo de predicción del 95%. Esta vez también añadimos una línea vertical para etiquetar cuándo empieza la predicción.

#### Validación de la predicción

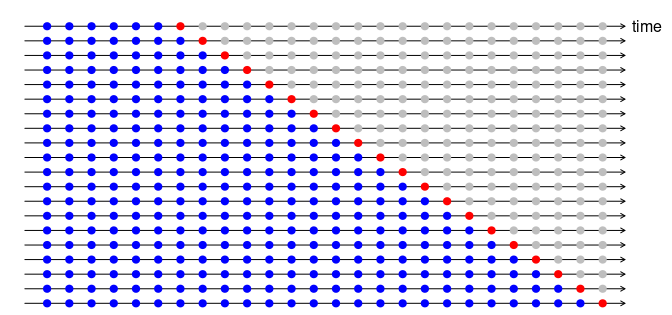
Más allá de la inspección de los residuos, es importante investigar lo bueno que es su modelo para predecir casos en el futuro. Esto le da una idea de la fiabilidad de sus alertas de umbral.

La forma tradicional de validar es ver lo bien que se puede predecir el último año anterior al actual (porque aún no se conocen los recuentos del "año actual"). Por ejemplo, en nuestro conjunto de datos, utilizaríamos los datos de 2002 a 2009 para predecir 2010, y luego veríamos la precisión de esas predicciones. A continuación, volveríamos a ajustar el modelo para incluir los datos de 2010 y los utilizaríamos para predecir los recuentos de 2011.

Como puede verse en la siguiente figura de Hyndman et al en ["Forecasting principles and practice"](https://otexts.com/fpp3/).

figura reproducida con permiso de los autores

La desventaja de esto es que no estás usando todos los datos disponibles, y no es el modelo final que estás usando para la predicción.

Una alternativa es utilizar un método llamado validación cruzada. En este caso, se pasan todos los datos disponibles para ajustar múltiples modelos de predicción a un año vista. Se utilizan cada vez más datos en cada modelo, como se ve en la siguiente figura del mismo [texto de Hyndman et al]([(](https://otexts.com/fpp3/)https://otexts.com/fpp3/). Por ejemplo, el primer modelo utiliza 2002 para predecir 2003, el segundo utiliza 2002 y 2003 para predecir 2004, y así sucesivamente. figura reproducida con permiso de los autores

A continuación, utilizamos la función map() del paquete **purrr** para recorrer cada conjunto de datos. A continuación, ponemos las estimaciones en unos datos y las fusionamos con los recuentos de casos originales, para utilizar el paquete **yardstick** para calcular las medidas de precisión. Calculamos cuatro medidas que incluyen: Error medio cuadrático (RMSE), Error medio absoluto (MAE), Error medio absoluto a escala (MASE), Error medio porcentual absoluto (MAPE).

**ATENCIÓN:** Observe el uso de simulate\_pi = FALSE dentro del argumento de predict(). Esto se debe a que el comportamiento por defecto de la **tendencia** es utilizar el paquete **ciTools** para estimar un intervalo de predicción. Esto no funciona si hay recuentos NA, y también produce intervalos más granulares. Véase ?trending::predict.trending\_model\_fit para más detalles.

### paquete de ****vigilancia****

En esta sección utilizamos el paquete de **vigilancia** para crear umbrales de alerta basados en algoritmos de detección de brotes. Hay varios métodos diferentes disponibles en el paquete, aunque aquí nos centraremos en dos opciones. Para más detalles, consulte estos documentos sobre la [aplicación](https://cran.r-project.org/web/packages/surveillance/vignettes/monitoringCounts.pdf) y la [teoría](https://cran.r-project.org/web/packages/surveillance/vignettes/glrnb.pdf) de los alogirmas utilizados.

La primera opción utiliza el método Farrington mejorado. Este método ajusta un glm binomial negativo (incluyendo la tendencia) y pondera a la baja los brotes pasados (valores atípicos) para crear un nivel de umbral.

La segunda opción utiliza el método glrnb. Esto también se ajusta a un glm binomial negativo, pero incluye la tendencia y los términos de fourier (por lo que se favorece aquí). La regresión se utiliza para calcular la "media de control" (~valores ajustados), y a continuación se utiliza un estadístico de relación de verosimilitud generalizada para evaluar si hay un cambio en la media de cada semana. Ten en cuenta que el umbral de cada semana tiene en cuenta las semanas anteriores, por lo que si hay un cambio sostenido se activará una alarma. (También hay que tener en cuenta que después de cada alarma el algoritmo se reinicia)

Para trabajar con el paquete de **vigilancia**, primero tenemos que definir un objeto de "series temporales de vigilancia" (utilizando la función sts()) para que encaje en el marco.

#### Método Farrington

A continuación, definimos cada uno de nuestros parámetros para el método Farrington en una lista. A continuación, ejecutamos el algoritmo utilizando farringtonFlexible() y luego podemos extraer el umbral de una alerta utilizando farringtonmethod@upperbound para incluirlo en nuestro conjunto de datos. También es posible extraer un TRUE/FALSE para cada semana si se activó una alerta (estaba por encima del umbral) utilizando farringtonmethod@alarm.

A continuación, podemos visualizar los resultados en ggplot como se hizo anteriormente.

#### Método GLRNB

Del mismo modo, para el método GLRNB definimos cada uno de nuestros parámetros para el en una lista, luego ajustamos el algoritmo y extraemos los límites superiores.

**ATENCIÓN:** Este método utiliza la "fuerza bruta" (similar al bootstrapping) para calcular los umbrales, por lo que puede llevar mucho tiempo.

Consulte la [viñeta GLRNB](https://cran.r-project.org/web/packages/surveillance/vignettes/glrnb.pdf) para más detalles.

Visualiza los resultados como antes.

## Series temporales interrumpidas

Las series temporales interrumpidas (también llamadas análisis de regresión segmentada o de intervención), se utilizan a menudo para evaluar el impacto de las vacunas en la incidencia de la enfermedad. Pero puede utilizarse para evaluar el impacto de una amplia gama de intervenciones o introducciones. Por ejemplo, cambios en los procedimientos hospitalarios o la introducción de una nueva cepa de enfermedad en una población. En este ejemplo, supondremos que se introdujo una nueva cepa de Campylobacter en Alemania a finales de 2008, y veremos si eso afecta al número de casos. Volveremos a utilizar la regresión binomial negativa. Esta vez, la regresión se dividirá en dos partes, una antes de la intervención (o introducción de la nueva cepa en este caso) y otra después (los períodos anterior y posterior). Esto nos permite calcular una tasa de incidencia comparando los dos periodos de tiempo. Explicar la ecuación podría aclararlo (si no es así, ignórela).

La regresión binomial negativa puede definirse como sigue:

\[\log(Y\_t)= β\_0 + β\_1 \times t+ β\_2 \times δ(t-t\_0) + β\_3\times(t-t\_0 )^+ + log(pop\_t) + e\_t\]

Donde: \(Y\_t\)es el número de casos observados en el momento \(t\)  
\(pop\_t\) es el tamaño de la población en 100.000s en el momento \(t\) (no se utiliza aquí)  
\(t\_0\) es el último año del preperíodo (incluyendo el tiempo de transición si lo hay)  
\(δ(x\) es la función indicadora (es 0 si x≤0 y 1 si x>0)  
\((x)^+\) es el operador de corte (es x si x>0 y 0 en caso contrario)  
\(e\_t\) denota el residuo Se pueden añadir los términos adicionales tendencia y estación según sea necesario.

\(β\_2 \times δ(t-t\_0) + β\_3\times(t-t\_0 )^+\) es la parte lineal generalizada del periodo posterior y es cero en el periodo anterior. Esto significa que las estimaciones \(β\_2\) y \(β\_3\) son los efectos de la intervención.

Aquí tenemos que volver a calcular los términos de Fourier sin previsión, ya que utilizaremos todos los datos de que disponemos (es decir, a posteriori). Además, tenemos que calcular los términos adicionales necesarios para la regresión.

A continuación, utilizamos estos términos para ajustar una regresión binomial negativa y elaboramos una tabla con el porcentaje de cambio. Lo que muestra este ejemplo es que no hubo ningún cambio significativo.

**ATENCIÓN:** Observe el uso de simulate\_pi = FALSE dentro del argumento de predict(). Esto se debe a que el comportamiento por defecto de la **tendencia** es utilizar el paquete **ciTools** para estimar un intervalo de predicción. Esto no funciona si hay recuentos NA, y también produce intervalos más granulares. Véase ?trending::predict.trending\_model\_fit para más detalles.

Como en el caso anterior, podemos visualizar los resultados de la regresión.

## Recursos

[previsión: principios y práctica libro de texto](https://otexts.com/fpp3/)  
[Estudios de casos de análisis de series temporales de EPIET](https://github.com/EPIET/TimeSeriesAnalysis)  
[Curso de Penn State](https://online.stat.psu.edu/stat510/lesson/1) [Manuscrito del paquete de vigilancia](https://www.jstatsoft.org/article/view/v070i10)

# #Modelización de epidemias

{#epidemic-modeling}

## Resumen

Existe un conjunto creciente de herramientas para la modelización de epidemias que nos permite realizar análisis bastante complejos con un esfuerzo mínimo. En esta sección se ofrece una visión general de cómo utilizar estas herramientas para:

* estimar el número de reproducción efectivo Rt y las estadísticas relacionadas, como el tiempo de duplicación
* elaborar proyecciones a corto plazo de la incidencia futura

No pretende ser una visión general de las metodologías y los métodos estadísticos en los que se basan estas herramientas, así que consulte la pestaña de Recursos para ver los enlaces a algunos documentos que cubren esto. Asegúrese de que conoce los métodos antes de utilizar estas herramientas, ya que así podrá interpretar con precisión sus resultados.

A continuación se muestra un ejemplo de uno de los resultados que produciremos en esta sección.

## Preparación

Utilizaremos dos métodos y paquetes diferentes para la estimación de Rt, a saber, **EpiNow** y **EpiEstim**, así como el paquete de **proyecciones** para la previsión de la incidencia de casos.

Este trozo de código muestra la carga de los paquetes necesarios para los análisis. En este manual destacamos p\_load() de **pacman**, que instala el paquete si es necesario y lo carga para su uso. También puede cargar los paquetes instalados con library() de R **base.** Consulte la página sobre [los fundamentos de R](#r-basics) para obtener más información sobre los paquetes de R.

Utilizaremos la lista de casos limpia para todos los análisis de esta sección. Si quiere seguir el proceso, [clica para descargar linelist "limpia"](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/case_linelists/linelist_cleaned.rds) (como archivo .rds). Consulte la página de [descarga del manual y los datos](#download-handbook-and-data) para descargar todos los datos de ejemplo utilizados en este manual.

## Estimación de Rt

### EpiNow2 vs. EpiEstim

El número de reproducción R es una medida de la transmisibilidad de una enfermedad y se define como el número esperado de casos secundarios por cada caso infectado. En una población totalmente susceptible, este valor representa el número básico de reproducción R0. Sin embargo, como el número de individuos susceptibles en una población cambia en el transcurso de un brote o pandemia, y como se aplican diversas medidas de respuesta, la medida de transmisibilidad más utilizada es el número de reproducción efectivo Rt; éste se define como el número esperado de casos secundarios por caso infectado en un tiempo t determinado.

El paquete **EpiNow2** proporciona el marco más sofisticado para estimar Rt. Tiene dos ventajas clave sobre el otro paquete comúnmente utilizado, **EpiEstim**:

* Tiene en cuenta los retrasos en la notificación y, por lo tanto, puede estimar la Rt incluso cuando los datos recientes son incompletos.
* Estima la Rt en función de las fechas de infección y no de las fechas de inicio de la notificación, lo que significa que el efecto de una intervención se reflejará inmediatamente en un cambio en la Rt, en lugar de con un retraso.

Sin embargo, también tiene dos desventajas fundamentales:

* Requiere conocer la distribución del tiempo de generación (es decir, la distribución de los retrasos entre la infección de un caso primario y uno secundario), la distribución del periodo de incubación (es decir, la distribución de los retrasos entre la infección y el inicio de los síntomas) y cualquier otra distribución de los retrasos que sea relevante para sus datos (por ejemplo, si tiene fechas de notificación, necesita la distribución de los retrasos desde el inicio de los síntomas hasta la notificación). Aunque esto permitirá una estimación más precisa de Rt, **EpiEstim** sólo requiere la distribución de intervalos en serie (es decir, la distribución de retrasos entre el inicio de los síntomas de un caso primario y uno secundario), que puede ser la única distribución disponible para usted.
* **EpiNow2** es significativamente más lento que **EpiEstim**, anecdóticamente por un factor de 100-1000. Por ejemplo, la estimación de Rt para el brote de la muestra considerada en esta sección tarda unas cuatro horas (esto se ejecutó para un gran número de iteraciones para asegurar una alta precisión y probablemente podría reducirse si fuera necesario, sin embargo los puntos son que el algoritmo es lento en general). Esto puede ser inviable si se actualizan regularmente las estimaciones de Rt.

Por tanto, el paquete que elija utilizar dependerá de los datos, el tiempo y los recursos informáticos de que disponga.

### EpiNow2

#### Estimación de las distribuciones de los retrasos

Las distribuciones de retraso necesarias para ejecutar **EpiNow2** dependen de los datos que tenga. Esencialmente, necesita poder describir el retraso desde la fecha de la infección hasta la fecha del evento que quiere usar para estimar Rt. Si está usando fechas de inicio, esto sería simplemente la distribución del periodo de incubación. Si se utilizan las fechas de notificación, se requiere el retraso desde la infección hasta la notificación. Como es poco probable que esta distribución se conozca directamente, **EpiNow2** le permite encadenar varias distribuciones de retraso; en este caso, el retraso desde la infección hasta el inicio de los síntomas (por ejemplo, el periodo de incubación, que probablemente se conoce) y desde el inicio de los síntomas hasta la notificación (que a menudo se puede estimar a partir de los datos).

Como tenemos las fechas de inicio de todos nuestros casos en linelist de ejemplo, sólo necesitaremos la distribución del periodo de incubación para relacionar nuestros datos (por ejemplo, las fechas de inicio de los síntomas) con la fecha de la infección. Podemos estimar esta distribución a partir de los datos o utilizar valores de la literatura.

Una estimación bibliográfica del periodo de incubación del ébola (tomada de [este documento](https://www.nejm.org/doi/full/10.1056/nejmoa1411100)) con una media de 9,1, una desviación estándar de 7,3 y un valor máximo de 30 se especificaría como sigue:

Ten en cuenta que **EpiNow2** requiere que estas distribuciones de retardo se proporcionen en una escala **logarítmica**, de ahí la llamada logarítmica alrededor de cada valor (excepto el parámetro max que, confusamente, tiene que proporcionarse en una escala natural). Los parámetros mean\_sd y sd\_sd definen la desviación estándar de las estimaciones de la media y la desviación estándar. Como no se conocen en este caso, elegimos el valor bastante arbitrario de 0,1.

En este análisis, en cambio, estimamos la distribución del periodo de incubación a partir de la propia lista de líneas utilizando la función bootstrapped\_dist\_fit, que ajustará una distribución lognormal a los retrasos observados entre la infección y el inicio en linelist.

La otra distribución que necesitamos es el tiempo de generación. Como tenemos datos sobre los tiempos de infección **y los enlaces de transmisión**, podemos estimar esta distribución a partir del listado calculando el retraso entre los tiempos de infección de los pares infector-infectado. Para ello, utilizamos la práctica función get\_pairwise del paquete **epicontacts**, que nos permite calcular las diferencias por pares de las propiedades del listado entre los pares de transmisión. Primero creamos un objeto epicontacts (ver la página de [cadenas de transmisión](#transmission-chains) para más detalles):

A continuación, ajustamos la diferencia de tiempos de infección entre pares de transmisión, calculada mediante get\_pairwise, a una distribución gamma:

#### Ejecución de ****EpiNow2****

Ahora sólo tenemos que calcular la incidencia diaria del listado, lo que podemos hacer fácilmente con las funciones de **dplyr** group\_by() y n(). Ten en cuenta que **EpiNow2** requiere que los nombres de las columnas sean fecha y confirmación.

A continuación, podemos estimar Rt mediante la función epinow. Algunas notas sobre las entradas:

* Podemos proporcionar cualquier número de distribuciones de retraso "encadenadas" al argumento delays; simplemente las insertaríamos junto al objeto incubation\_period dentro de la función delay\_opts.
* return\_output asegura que la salida sea devuelta dentro de R y no sólo guardada en un archivo.
* verbose especifica que queremos una lectura del progreso.
* El horizonte indica para cuántos días queremos proyectar la incidencia futura.
* Pasamos opciones adicionales al argumento stan para especificar durante cuánto tiempo queremos ejecutar la inferencia. Si aumentamos las muestras y las cadenas, obtendremos una estimación más precisa que caracterice mejor la incertidumbre, aunque tardará más en ejecutarse.

#### Análisis de los resultados

Una vez que el código ha terminado de ejecutarse, podemos trazar un resumen muy fácilmente, como se indica a continuación. Desplaza la imagen para ver la extensión completa.

También podemos consultar varias estadísticas resumidas:

Para otros análisis y trazados personalizados, puede acceder a las estimaciones diarias resumidas a través de $estimates$summarised. Convertiremos esto del data.table por defecto a un tibble para facilitar su uso con **dplyr**.

A modo de ejemplo, hagamos un gráfico del tiempo de duplicación y Rt. Sólo nos fijaremos en los primeros meses del brote, cuando Rt es muy superior a uno, para evitar trazar tiempos de duplicación extremadamente altos.

Utilizamos la fórmula log(2)/tasa de crecimiento para calcular el tiempo de duplicación a partir de la tasa de crecimiento estimada.

### EpiEstim

Para ejecutar **EpiEstim**, necesitamos proporcionar datos sobre la incidencia diaria y especificar el intervalo de serie (es decir, la distribución de los retrasos entre el inicio de los síntomas de los casos primarios y secundarios).

Los datos de incidencia pueden proporcionarse a **EpiEstim** como un vector, un dataframe o un objeto de incidencia del paquete de **incidencia** original. Incluso se puede distinguir entre infecciones importadas y adquiridas localmente; consulte la documentación en ?estimate\_R para más detalles.

Crearemos la entrada utilizando **incidence2**. Consulte la página sobre [curvas epidémicas](#epidemic-curves) para ver más ejemplos con el paquete **incidence2**. Dado que ha habido actualizaciones en el paquete **incidence2** que no se alinean completamente con la entrada esperada de estimateR(), hay algunos pasos adicionales menores necesarios. El objeto de incidencia consiste en un tibble con fechas y sus respectivos recuentos de casos. Usamos complete() de **tidyr** para asegurarnos de que se incluyen todas las fechas (incluso las que no tienen casos), y luego renombramos() las columnas para alinearlas con lo que espera estimate\_R() en un paso posterior.

El paquete proporciona varias opciones para especificar el intervalo en serie, cuyos detalles se proporcionan en la documentación en ?estimate\_R. Aquí cubriremos dos de ellas.

#### Utilizando estimaciones de intervalos de serie de la literatura

Utilizando la opción method = "parametric\_si", podemos especificar manualmente la media y la desviación estándar del intervalo en serie en un objeto config creado con la función make\_config. Utilizamos una media y una desviación estándar de 12,0 y 5,2, respectivamente, definidas en [este documento](https://bmcmedicine.biomedcentral.com/articles/10.1186/s12916-014-0196-0):

Entonces podemos estimar Rt con la función estimate\_R:

y trazar un resumen de los resultados:

#### Utilización de estimaciones de intervalos de serie a partir de los datos

Como tenemos datos sobre las fechas de inicio de los síntomas y los vínculos de transmisión, también podemos estimar el intervalo de serie a partir del listado calculando el retraso entre las fechas de inicio de los pares infector-infectado. Como hicimos en la sección **EpiNow2**, utilizaremos la función get\_pairwise del paquete epicontacts, que nos permite calcular las diferencias por pares de las propiedades del listado entre los pares de transmisión. Primero creamos un objeto epicontacts (ver la página de cadenas [de transmisión](#transmission-chains) para más detalles):

A continuación, ajustamos la diferencia de fechas de inicio entre los pares de transmisión, calculada mediante get\_pairwise, a una distribución gamma. Utilizamos el práctico fit\_disc\_gamma del paquete **epitrix** para este procedimiento de ajuste, ya que necesitamos una distribución discreta.

A continuación, pasamos esta información al objeto config, ejecutamos de nuevo **EpiEstim** y trazamos los resultados:

#### Especificación de las ventanas de tiempo de estimación

Estas opciones por defecto proporcionarán una estimación deslizante semanal y podrían actuar como una advertencia de que está estimando Rt demasiado pronto en el brote para una estimación precisa. Puedes cambiar esto estableciendo una fecha de inicio posterior para la estimación, como se muestra a continuación. Lamentablemente, **EpiEstim** sólo proporciona una forma muy tosca de especificar estos tiempos de estimación, ya que tiene que proporcionar un vector de **enteros** que se refieran a las fechas de inicio y fin de cada ventana temporal.

Ahora volvemos a ejecutar **EpiEstim** y podemos ver que las estimaciones sólo comienzan a partir de junio:

#### Análisis de los resultados

Se puede acceder a los principales resultados a través de $R. Como ejemplo, crearemos un gráfico de Rt y una medida de "potencial de transmisión" dada por el producto de Rt y el número de casos notificados en ese día; esto representa el número esperado de casos en la siguiente generación de infección.

## Proyección de la incidencia

### EpiNow2

Además de la estimación de Rt, **EpiNow2** también admite la previsión de Rt y las proyecciones del número de casos mediante la integración con el paquete **EpiSoon** bajo el capó. Todo lo que hay que hacer es especificar el argumento del horizonte en la llamada a la función epinow, indicando cuántos días se quiere proyectar en el futuro; véase la sección de **EpiNow2** en la sección "Estimación de Rt" para obtener detalles sobre cómo poner en marcha **EpiNow2**. En esta sección, sólo vamos a trazar los resultados de ese análisis, almacenados en el objeto epinow\_res.

### proyecciones

El paquete de **proyecciones** desarrollado por RECON hace que sea muy fácil hacer previsiones de incidencia a corto plazo, requiriendo sólo el conocimiento del número de reproducción efectivo Rt y el intervalo serial. Aquí cubriremos cómo utilizar las estimaciones del intervalo de serie de la literatura y cómo utilizar nuestras propias estimaciones del listado.

#### Utilizando estimaciones de intervalos de serie de la literatura

Las **proyecciones** requieren una distribución de intervalos seriales discretizados del tipo distcrete del paquete **distcrete**. Utilizaremos una distribución gamma con una media de 12,0 y una desviación estándar de 5,2 definida en [este documento](https://bmcmedicine.biomedcentral.com/articles/10.1186/s12916-014-0196-0). Para convertir estos valores en los parámetros de forma y escala necesarios para una distribución gamma, utilizaremos la función gamma\_mucv2shapescale del paquete **epitrix**.

Aquí tenemos una comprobación rápida para asegurarnos de que el intervalo de la serie parece correcto. Accedemos a la densidad de la distribución gamma que acabamos de definir mediante $d, lo que equivale a llamar a dgamma:

#### Utilización de estimaciones de intervalos de serie a partir de los datos

Como tenemos datos sobre las fechas de inicio de los síntomas y los vínculos de transmisión, también podemos estimar el intervalo de serie a partir del listado calculando el retraso entre las fechas de inicio de los pares infector-infectado. Como hicimos en la sección **EpiNow2**, utilizaremos la función get\_pairwise del paquete epicontacts, que nos permite calcular las diferencias por pares de las propiedades del listado entre los pares de transmisión. Primero creamos un objeto epicontacts (ver la página de cadenas [de transmisión](#transmission-chains) para más detalles):

A continuación, ajustamos la diferencia de fechas de inicio entre los pares de transmisión, calculada mediante get\_pairwise, a una distribución gamma. Utilizamos el práctico fit\_disc\_gamma del paquete **epitrix** para este procedimiento de ajuste, ya que necesitamos una distribución discreta.

#### Proyección de la incidencia

Para proyectar la incidencia futura, todavía tenemos que proporcionar la incidencia histórica en forma de un objeto de incidencia, así como una muestra de valores de Rt plausibles. Generaremos estos valores utilizando las estimaciones de Rt generadas por **EpiEstim** en la sección anterior (en "Estimación de Rt") y almacenadas en el objeto epiestim\_res\_emp. En el código siguiente, extraemos las estimaciones de la media y la desviación estándar de Rt para la última ventana temporal del brote (utilizando la función tail para acceder al último elemento de un vector), y simulamos 1000 valores a partir de una distribución gamma utilizando rgamma. También puede proporcionar su propio vector de valores de Rt que desee utilizar para las proyecciones a futuro.

A continuación, utilizamos la función project() para realizar la previsión real. Especificamos para cuántos días queremos proyectar mediante los argumentos n\_days, y especificamos el número de simulaciones utilizando el argumento n\_sim.

A continuación, podemos trazar fácilmente la incidencia y las proyecciones utilizando las funciones plot() y add\_projections(). Podemos fácilmente subconjuntar el objeto de incidencia para mostrar sólo los casos más recientes utilizando el operador de corchetes.

También puede extraer fácilmente las estimaciones brutas del número de casos diarios convirtiendo la salida en un dataframe.

## Recursos

* [Aquí está el documento](https://www.sciencedirect.com/science/article/pii/S1755436519300350) que describe la metodología implementada en **EpiEstim**.
* [Aquí está el documento](https://wellcomeopenresearch.org/articles/5-112/v1) que describe la metodología implementada en **EpiNow2**.
* [Aquí hay un documento](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008409) que describe varias consideraciones metodológicas y prácticas para estimar el Rt.

# **#Rastreo** de contactos

{#contact-tracing-1}

Esta página muestra el análisis descriptivo de los datos de rastreo de contactos, abordando algunas consideraciones clave y enfoques exclusivos de este tipo de datos.

Esta página hace referencia a muchas de las competencias básicas de gestión y visualización de datos de R tratadas en otras páginas (por ejemplo, limpieza de datos, pivoteo, tablas, análisis de series temporales), pero destacaremos ejemplos específicos del rastreo de contactos que han sido útiles para la toma de decisiones operativas. Por ejemplo, esto incluye la visualización de los datos de seguimiento del rastreo de contactos a lo largo del tiempo o a través de áreas geográficas, o la producción de tablas limpias de Indicadores Clave de Rendimiento (KPI) para los supervisores del rastreo de contactos.

Para la demostración utilizaremos datos de rastreo de contactos de la plataforma [Go.Data](https://www.who.int/tools/godata). Los principios que aquí se exponen son válidos para los datos de rastreo de contactos de otras plataformas, sólo que puede ser necesario realizar diferentes pasos de preprocesamiento de datos en función de la estructura de los mismos.

Puedes leer más sobre el proyecto Go.Data en el [sitio de documentación de Github](https://worldhealthorganization.github.io/godata/) o en la [Comunidad de Práctica](https://community-godata.who.int/).

## Preparación

### Cargar paquetes

Este trozo de código muestra la carga de los paquetes necesarios para los análisis. En este manual destacamos p\_load() de **pacman**, que instala el paquete si es necesario y lo carga para su uso. También puede cargar los paquetes instalados con library() de R **base.** Consulte la página sobre [los fundamentos de R](#r-basics) para obtener más información sobre los paquetes de R.

### Importar datos

Importaremos conjuntos de datos de muestra de contactos y de su "seguimiento". Estos datos se han recuperado y desanidado de la API Go.Data y se han almacenado como archivos ".rds".

Puedes descargar todos los datos de ejemplo de este manual en la página de [descarga de manuales y datos](#download-handbook-and-data).

Si desea descargar los datos de seguimiento de contactos de ejemplo específicos de esta página, utiliza los tres enlaces de descarga que aparecen a continuación:

[Clica para descargar los datos de la investigación del caso (archivo .rds)](https://github.com/WorldHealthOrganization/godata/blob/master/analytics/r-reporting/data/cases_clean.rds?raw=true)

[Clica para descargar los datos de registro de los contactos (archivo .rds)](https://github.com/WorldHealthOrganization/godata/blob/master/analytics/r-reporting/data/contacts_clean.rds?raw=true)

[Clica para descargar los datos de seguimiento de los contactos (archivo .rds)](https://github.com/WorldHealthOrganization/godata/blob/master/analytics/r-reporting/data/followups_clean.rds?raw=true)

En su forma original en los archivos descargables, los datos reflejan los datos proporcionados por la API de Go.Data (aprenda sobre [las API aquí](#import_api)). A modo de ejemplo, aquí limpiaremos los datos para que sean más fáciles de leer en esta página. Si está utilizando una instancia de Go.Data, puede ver las instrucciones completas sobre cómo recuperar sus datos [aquí](https://github.com/WorldHealthOrganization/godata/tree/master/analytics/r-reporting).

A continuación, los conjuntos de datos se importan utilizando la función import() del paquete **rio**. Consulte la página sobre [importación y exportación](#import-and-export) para conocer las distintas formas de importar datos. Utilizamos here() para especificar la ruta del archivo - debes escribir la ruta del archivo específica de su ordenador. A continuación, utilizamos select() para seleccionar sólo ciertas columnas de los datos, para simplificar a efectos de demostración.

#### Datos del caso

Estos datos son una tabla de los casos, y la información sobre ellos.

Aquí están los casos de nrow(cases):

#### Datos de contacto

Estos datos son una tabla de todos los contactos e información sobre ellos. De nuevo, proporcione su propia ruta de acceso al archivo. Después de la importación, realizamos algunos pasos preliminares de limpieza de datos que incluyen:

* Establezca age\_class como factor e invierta el orden de los niveles para que las edades más jóvenes sean las primeras
* Seleccionar sólo una columna determinada, renombrando una de ellas
* Asignar artificialmente a "Djembe" las filas a las que les falta el nivel 2 de administración, para mejorar la claridad de algunas visualizaciones de ejemplo

Aquí están las filas nrow(contacts) de los datos de contactos:

#### Datos de seguimiento

Estos datos son registros de las interacciones de "seguimiento" con los contactos. Se supone que cada contacto tiene un encuentro diario durante los 14 días siguientes a su exposición.

Importamos y realizamos algunos pasos de limpieza. Seleccionamos ciertas columnas y también convertimos una columna de caracteres a todos los valores en minúsculas.

Aquí están las primeras 50 filas de los datos nrow(followups)-row followups (cada fila es una interacción de seguimiento, con el estado del resultado en la columna followup\_status):

#### Datos de las relaciones

Aquí importamos datos que muestran la relación entre casos y contactos. Seleccionamos cierta columna para mostrar.

A continuación se muestran las primeras 50 filas de los datos de relaciones, que registra todas las relaciones entre casos y contactos.

## Análisis descriptivos

Puedes utilizar las técnicas tratadas en otras páginas de este manual para realizar análisis descriptivos de tus casos, contactos y sus relaciones. A continuación se ofrecen algunos ejemplos.

### Datos demográficos

Como se ha demostrado en la página dedicada a [las pirámides demográficas, se](#demographic-pyramids-and-likert-scales) puede visualizar la distribución por edades y por sexos (aquí utilizamos el paquete **apyramid**).

#### Edad y sexo de los contactos

La pirámide que se muestra a continuación compara la distribución de la edad de los contactos, por género. Observe que los contactos a los que les falta la edad se incluyen en su propia barra en la parte superior. Puedes cambiar este comportamiento por defecto, pero entonces considere listar el número que falta en una leyenda.

Con la estructura de datos Go.Data, los datos de relaciones contiene las edades tanto de los casos como de los contactos, por lo que podría utilizar ese conjunto de datos y crear una pirámide de edades que muestre las diferencias entre estos dos grupos de personas. El dataframe de relaciones será mutado para transformar las columnas numéricas de edad en categorías (véase la página de [limpieza de datos y funciones básicas](#cleaning-data-and-core-functions)). También pivotamos el dataframe más largo para facilitar el trazado con **ggplot2** (ver [Pivotando datos](#pivoting-data)).

Ahora podemos trazar este conjunto de datos transformado con age\_pyramid() como antes, pero sustituyendo el género por la categoría (contacto, o caso).

También podemos ver otras características como el desglose profesional (por ejemplo, en forma de gráfico circular).

### Contactos por caja

El número de contactos por caso puede ser una métrica importante para evaluar la calidad de la enumeración de los contactos y la conformidad de la población con la respuesta de salud pública.

Dependiendo de su estructura de datos, esto puede evaluarse con unos datos que contenga todos los casos y contactos. En los conjuntos de datos de Go.Data, los vínculos entre los casos ("fuentes") y los contactos ("objetivos") se almacenan en los datos de relaciones.

En este conjunto de datos, cada fila es un contacto, y el caso de origen aparece en la fila. No hay contactos que tengan relaciones con múltiples casos, pero si esto existe, puede ser necesario tenerlos en cuenta antes de trazar (¡y explorarlos también!).

Comenzamos contando el número de filas (contactos) por caso de origen. Esto se guarda como un dataframe.

Utilizamos geom\_histogram() para trazar estos datos como un histograma.

## Seguimiento de contactos

Los datos de rastreo de contactos suelen contener datos de "seguimiento", que registran los resultados de los controles diarios de los síntomas de las personas en cuarentena. El análisis de estos datos puede servir de base para la estrategia de respuesta e identificar a los contactos con riesgo de pérdida de seguimiento o con riesgo de desarrollar la enfermedad.

### Limpieza de datos

Estos datos pueden existir en una variedad de formatos. Pueden existir como una hoja de Excel de formato "ancho" con una fila por contacto y una columna por "día" de seguimiento. Consulte [Pivotear datos](#pivoting-data) para ver las descripciones de los datos "largos" y "anchos" y cómo pivotar los datos más anchos o más largos.

En nuestro ejemplo de Go.Data, estos datos se almacenan en el dataframe de seguimiento, que tiene un formato "largo" con una fila por interacción de seguimiento. Las primeras 50 filas tienen este aspecto:

**PRECAUCIÓN:** Tenga cuidado con los duplicados al tratar los datos de seguimiento, ya que podría haber varios seguimientos erróneos en el mismo día para un contacto determinado. Tal vez parezca un error, pero refleja la realidad: por ejemplo, un rastreador de contactos podría enviar un formulario de seguimiento a primera hora del día cuando no pudo contactar con el contacto, y enviar un segundo formulario cuando se le pudo contactar más tarde. Dependerá del contexto operativo la forma en que desee gestionar los duplicados, pero asegúrese de documentar claramente su enfoque.

Veamos cuántos casos de filas "duplicadas" tenemos:

En nuestros datos de ejemplo, los únicos registros a los que se aplica esto son los que carecen de ID. Podemos eliminarlos. Pero, a efectos de demostración, mostraremos los pasos para la eliminación de la duplicación de modo que sólo haya un registro de seguimiento por persona y por día. Para más detalles, consulte la página de [desduplicación](#de-duplication). Asumiremos que el registro de encuentro más reciente es el correcto. También aprovechamos la oportunidad para limpiar la columna followup\_number (el "día" de seguimiento que debe ir de 1 a 14).

Para cada encuentro de seguimiento, tenemos un estado de seguimiento (como si el encuentro se produjo y, si es así, el contacto tuvo síntomas o no). Para ver todos los valores podemos ejecutar un tabyl() rápido (desde el **conserje**) o table() (desde **la base** R) (ver [Tablas descriptivas](#descriptive-tables)) por followup\_status para ver la frecuencia de cada uno de los resultados.

En este conjunto de datos, "seen\_not\_ok" significa "visto con síntomas", y "seen\_ok" significa "visto sin síntomas".

### Trazado en el tiempo

Como los datos de las fechas son continuos, utilizaremos un histograma para representarlos con date\_of\_followup asignado al eje x. Podemos conseguir un histograma "apilado" especificando un argumento fill = dentro de aes(), que asignamos a la columna followup\_status. En consecuencia, se puede establecer el título de la leyenda utilizando el argumento fill = de labs().

Podemos ver que los contactos se identificaron en oleadas (presumiblemente correspondientes a las oleadas epidémicas de casos), y que la finalización del seguimiento no parece haber mejorado a lo largo de la epidemia.

**PRECAUCIÓN:** Si estás preparando muchos gráficos (por ejemplo, para múltiples jurisdicciones) querrás que las leyendas aparezcan de forma idéntica incluso con diferentes niveles de finalización o composición de los datos. Puede haber gráficos para los cuales no todos los estados de seguimiento están presentes, pero todavía quieres que esas categorías aparezcan en las leyendas. En ggplots (como arriba), puede especificar el argumento drop = FALSE de scale\_fill\_discrete(). En las tablas, utiliza tabyl() que muestra los recuentos de todos los niveles de los factores, o si utiliza count() de **dplyr** añada el argumento .drop = FALSE para incluir los recuentos de todos los niveles de los factores.

### Seguimiento individual diario

Si su brote es lo suficientemente pequeño, es posible que quiera mirar a cada contacto individualmente y ver su estado a lo largo de su seguimiento. Afortunadamente, este conjunto de datos de seguimiento ya contiene una columna con el "número" de día de seguimiento (1-14). Si no existe en sus datos, puede crearla calculando la diferencia entre la fecha de encuentro y la fecha en la que el seguimiento debía comenzar para el contacto.

Un mecanismo de visualización conveniente (si el número de casos no es demasiado grande) puede ser un gráfico de calor, hecho con geom\_tile(). Vea más detalles en la página [heat plot].

### Analizar por grupos

Tal vez estos datos de seguimiento se consulten diaria o semanalmente para la toma de decisiones operativas. Es posible que desee desgloses más significativos por zona geográfica o por equipo de seguimiento de contactos. Podemos hacerlo ajustando las columnas proporcionadas a group\_by().

## Tablas de KPI

Hay una serie de indicadores clave de rendimiento (KPI) que pueden calcularse y seguirse a distintos niveles de desagregación y a lo largo de diferentes períodos de tiempo para supervisar el rendimiento del rastreo de contactos. Una vez que se tienen los cálculos y el formato básico de la tabla, es bastante fácil cambiar los diferentes KPI.

Existen numerosas fuentes de KPI de rastreo de contactos, como ésta de [ResolveToSaveLives.org](https://contacttracingplaybook.resolvetosavelives.org/checklists/metrics). La mayor parte del trabajo consistirá en recorrer su estructura de datos y pensar en todos los criterios de inclusión/exclusión. A continuación mostramos algunos ejemplos, utilizando la estructura de metadatos de Go.Data:

| **Categoría** | **Indicador** | **Go.Data Numerador** | **Go.Data Denominador** |
| --- | --- | --- | --- |
| Indicador de proceso - Velocidad de rastreo de contactos | % de casos entrevistados y aislados en las 24 horas siguientes a la notificación del caso | COUNT OF case\_id WHERE (date\_of\_reporting - date\_of\_data\_entry) < 1 day AND (isolation\_startdate - date\_of\_data\_entry) < 1 day | COUNT OF case\_id |
| Indicador de proceso - Velocidad de rastreo de contactos | Porcentaje de contactos notificados y puestos en cuarentena en las 24 horas siguientes a la solicitud | COUNT OF contact\_id WHERE followup\_status == "SEEN\_NOT\_OK" OR "SEEN\_OK" AND date\_of\_followup - date\_of\_reporting < 1 day | COUNT OF contact\_id |
| Indicador de proceso - Completitud de las pruebas | Porcentaje de nuevos casos sintomáticos examinados y entrevistados en los 3 días siguientes al inicio de los síntomas | COUNT OF case\_id WHERE (date\_of\_reporting - date\_of\_onset) < =3 days | COUNT OF case\_id |
| Indicador de resultado - Global | % de nuevos casos entre la lista de contactos existente | COUNT OF case\_id WHERE was\_contact == "TRUE" | COUNT OF case\_id |

A continuación veremos un ejercicio de ejemplo para crear una bonita tabla visual para mostrar el seguimiento de los contactos en las áreas de administración. Al final, lo haremos apto para la presentación con el paquete **formattable** (pero podrías usar otros paquetes como **flextable** - ver [Tablas para la presentación](#tables-for-presentation)).

La forma de crear una tabla como ésta dependerá de la estructura de sus datos de seguimiento de contactos. Utiliza la página de [tablas descriptivas](#descriptive-tables) para aprender a resumir los datos utilizando las funciones de **dplyr**.

Crearemos una tabla que será dinámica y cambiará a medida que cambien los datos. Para que los resultados sean interesantes, estableceremos una fecha\_de\_informe que nos permita simular la ejecución de la tabla en un día determinado (elegimos el 10 de junio de 2020). Los datos se filtran a esa fecha.

Ahora, basándonos en nuestra estructura de datos, haremos lo siguiente:

1. Comience con los datos de seguimiento y resúmalos para contener, para cada contacto único:

* La fecha del último registro (sin importar el estado del encuentro)
* La fecha del último encuentro en el que el contacto fue "visto"
* El estado del encuentro en ese último encuentro "visto" (por ejemplo, con síntomas, sin síntomas)

1. Uniremos estos datos a los de los contactos, que contienen otra información como el estado general del contacto, la fecha de la última exposición a un caso, etc. También calcularemos las métricas de interés para cada contacto, como los días desde la última exposición
2. Agrupamos los datos de contacto mejorados por región geográfica (admin\_2\_nombre) y calculamos las estadísticas resumidas por región
3. Por último, damos un buen formato a la tabla para su presentación

Primero resumimos los datos de seguimiento para obtener la información de interés:

Así es como se ven estos datos:

Ahora añadiremos esta información al conjunto de datos de contactos y calcularemos algunas columnas adicionales.

Así es como se ven estos datos. Observe la columna de contactos a la derecha, y la nueva columna calculada en el extremo derecho.

A continuación, resumimos los datos de los contactos por región, para conseguir un dataframe conciso de columnas de estadísticas resumidas.

Y ahora aplicamos el estilo de los paquetes **formattable** y **knitr**, incluyendo una nota a pie de página que muestra la fecha "a partir de".

## Matrices de transmisión

Como se discutió en la página de [gráficos de calor](#heat-plots), puede crear una matriz de "quién infectó a quién" utilizando geom\_tile().

Cuando se crean nuevos contactos, Go.Data almacena esta información de relación en el punto final de la API de relaciones; y podemos ver las primeras 50 filas de este conjunto de datos a continuación. Esto significa que podemos crear un gráfico de calor con relativamente pocos pasos, dado que cada contacto ya está unido a su caso de origen.

Al igual que en el caso de la pirámide de edad que compara casos y contactos, podemos seleccionar las pocas variables que necesitamos y crear columnas con agrupaciones categóricas de edad tanto para las fuentes (casos) como para los objetivos (contactos).

Como se ha descrito anteriormente, creamos una tabulación cruzada;

convertir en formato largo con proporciones;

y crear un mapa de calor para la edad.

## Recursos

<https://github.com/WorldHealthOrganization/godata/tree/master/analytics/r-reporting>

<https://worldhealthorganization.github.io/godata/>

<https://community-godata.who.int/>

# #Análisis de encuestas

{#survey-analysis}

## Resumen

Esta página muestra el uso de varios paquetes para el análisis de encuestas.

La mayoría de los paquetes R de encuestas se basan en el [paquete de**encuestas**](https://cran.r-project.org/web/packages/survey/index.html) para realizar análisis ponderados. Utilizaremos **survey**, así como [**srvyr**](https://cran.r-project.org/web/packages/srvyr/index.html) (una envoltura para **survey que permite la codificación** al estilo tidyverse) y [**gtsummary**](https://cran.r-project.org/web/packages/gtsummary/index.html) (una envoltura para **survey** que permite obtener tablas listas para su publicación). Aunque el paquete original de **la encuesta** no permite la codificación al estilo tidyverse, tiene la ventaja añadida de permitir modelos lineales generalizados ponderados por la encuesta (que se añadirán a esta página más adelante). También demostraremos el uso de una función del paquete [**sitrep**](https://github.com/R4EPI/sitrep) para crear ponderaciones de muestreo (n.b. este paquete no está todavía en CRAN, pero se puede instalar desde github).

La mayor parte de esta página se basa en el trabajo realizado para el [proyecto "](https://r4epis.netlify.app/)[R4Epis"](https://github.com/R4EPI/sitrep); para ver el código detallado y las plantillas R-markdown, consulte la [página github de "R4Epis"](https://github.com/R4EPI/sitrep). Parte del código basado en el paquete de encuestas se basa en las primeras versiones de los [estudios de caso de EPIET](https://github.com/EPIET/RapidAssessmentSurveys).

Actualmente, esta página no aborda el cálculo del tamaño de la muestra ni el muestreo. Para una calculadora del tamaño de la muestra fácil de usar, consulte [OpenEpi](https://www.openepi.com/Menu/OE_Menu.htm). La página de [fundamentos del SIG](https://epirhandbook.com/gis-basics.html) del manual tendrá eventualmente una sección sobre muestreo aleatorio espacial, y esta página tendrá eventualmente una sección sobre marcos de muestreo así como cálculos del tamaño de la muestra.

1. Datos de la encuesta
2. Tiempo de observación
3. Ponderación
4. Objetos de diseño de la encuesta
5. Análisis descriptivo
6. Proporciones ponderadas
7. Tipos ponderados

## Preparación

### Paquetes

Este trozo de código muestra la carga de los paquetes necesarios para los análisis. En este manual destacamos p\_load() de **pacman**, que instala el paquete si es necesario y lo carga para su uso. También se pueden cargar paquetes con library() desde **el** R **base.** Consulte la página sobre [los fundamentos de R](#r-basics) para obtener más información sobre los paquetes de R.  
Aquí también demostramos el uso de la función p\_load\_gh() de **pacman** para instalar y cargar un paquete de github que aún no ha sido publicado en CRAN.

### Datos de carga

El set de datos de ejemplo utilizado en esta sección:

* datos de la encuesta de mortalidad ficticia.
* recuentos de población ficticios para la zona de la encuesta.
* diccionario de datos para los datos de la encuesta de mortalidad ficticia.

Se basa en la encuesta pre-aprobada por la junta de revisión ética de MSF OCA. Los datos ficticios se produjeron como parte del [proyecto "R4Epis"](https://r4epis.netlify.app/). Todo ello se basa en los datos recopilados mediante [KoboToolbox,](https://www.kobotoolbox.org/) un software de recopilación de datos basado en [Open Data Kit](https://opendatakit.org/).

Kobo le permite exportar tanto los datos recogidos como el diccionario de datos para ese conjunto de datos. Recomendamos encarecidamente hacer esto, ya que simplifica la limpieza de los datos y es útil para buscar variables/preguntas.

**CONSEJO:** El diccionario de datos de Kobo tiene nombres de variables en la columna "nombre" de la hoja de la encuesta. Los valores posibles para cada variable se especifican en la hoja de opciones. En la hoja de opciones, "name" tiene el valor acortado y las columnas "label::english" y "label::french" tienen las versiones largas correspondientes. Si utiliza la función msf\_dict\_survey() del paquete **epidict** para importar un archivo excel del diccionario Kobo, éste se reformulará para que pueda utilizarse fácilmente para recodificar.

**ATENCIÓN:** El conjunto de datos de ejemplo no es lo mismo que una exportación (ya que en Kobo se exportan los diferentes niveles del cuestionario de forma individual) - vea la sección de datos de la encuesta más abajo para fusionar los diferentes niveles.

El set de datos se importa mediante la función import() del paquete **rio**. Consulte la página sobre [importación y exportación](https://epirhandbook.com/import-and-export.html) para conocer las distintas formas de importar datos.

A continuación se muestran las 10 primeras filas de la encuesta.

También queremos importar los datos de la población de muestreo para poder elaborar las ponderaciones adecuadas. Estos datos pueden estar en diferentes formatos, sin embargo sugerimos tenerlos como se ve a continuación (esto puede ser simplemente escrito en un Excel).

A continuación se muestran las 10 primeras filas de la encuesta.

En el caso de las encuestas por conglomerados, es posible que desee añadir ponderaciones de la encuesta a nivel de conglomerado. Puedes introducir estos datos como se indica más arriba. Alternativamente, si sólo hay unos pocos recuentos, éstos podrían introducirse como se indica a continuación en un tríptico. En cualquier caso, tendrá que tener una columna con un identificador de conglomerado que coincida con los datos de su encuesta, y otra columna con el número de hogares en cada conglomerado.

### Datos limpios

A continuación se asegura de que la columna de fechas tenga el formato adecuado. Hay varias otras maneras de hacer esto (ver la página [Trabajar con fechas](https://epirhandbook.com/working-with-dates.html) para más detalles), sin embargo, usar el diccionario para definir las fechas es rápido y fácil.

También creamos una variable de grupo de edad utilizando la función age\_categories() de **epikit** - véase la sección del manual de [limpieza de datos](https://epirhandbook.com/cleaning-data-and-core-functions.html" \l "num_cats) para más detalles. Además, creamos una variable de carácter que define en qué distrito se encuentran las distintas agrupaciones.

Por último, recodificamos todas las variables sí/no en variables VERDADERO/FALSO, ya que de lo contrario no pueden ser utilizadas por las funciones de proporción de **la encuesta**.

## Datos de la encuesta

Existen numerosos diseños de muestreo que pueden utilizarse para las encuestas. Aquí demostraremos el código para: - Estratificado - Conglomerado - Estratificado y conglomerado

Como se ha descrito anteriormente (dependiendo de cómo diseñe su cuestionario) los datos de cada nivel se exportarían como unos datos separado de Kobo. En nuestro ejemplo hay un nivel para los hogares y un nivel para los individuos dentro de esos hogares.

Estos dos niveles están vinculados por un identificador único. Para unos datos de Kobo, esta variable es "\_index" en el nivel del hogar, que coincide con "\_parent\_index" en el nivel individual. Esto creará nuevas filas para el hogar con cada individuo que coincida, véase la sección del manual sobre la [unión](https://epirhandbook.com/joining-data.html) para más detalles.

## Tiempo de observación

En el caso de las encuestas de mortalidad, queremos saber cuánto tiempo ha estado presente cada individuo en el lugar para poder calcular una tasa de mortalidad adecuada para nuestro periodo de interés. Esto no es relevante para todas las encuestas, pero en particular para las encuestas de mortalidad es importante, ya que se realizan con frecuencia entre poblaciones móviles o desplazadas.

Para ello, primero definimos nuestro periodo de interés, también conocido como periodo de recuerdo (es decir, el tiempo sobre el que se pide a los participantes que informen al responder a las preguntas). A continuación, podemos utilizar este periodo para establecer las fechas inadecuadas como ausentes, es decir, si las muertes se notifican fuera del periodo de interés.

Entonces podemos utilizar nuestras variables de fecha para definir las fechas de inicio y fin de cada individuo. Podemos utilizar la función find\_start\_date() de **sitrep** para afinar las causas de las fechas y luego utilizarla para calcular la diferencia entre días (persona-tiempo).

Fecha de inicio: Evento de llegada más temprano dentro de su período de retirada O bien el inicio de su período de retirada (definidas de antemano), o una fecha posterior al inicio de la retirada, si procede (por ejemplo, llegadas o nacimientos)

Fecha de finalización: Evento de salida más temprano apropiado dentro de su periodo de retirada O bien el final de su periodo de retirada, o una fecha anterior al final de la retirada si procede (por ejemplo, salidas, fallecimientos)

## Ponderación

Es importante que elimine las observaciones erróneas antes de añadir los pesos de la encuesta. Por ejemplo, si tiene observaciones con tiempo de observación negativo, tendrá que comprobarlas (puede hacerlo con la función assert\_positive\_timespan() de **sitrep**. Otra cosa es si quiere eliminar las filas vacías (por ejemplo, con drop\_na(uid)) o eliminar los duplicados (véase la sección del manual sobre [Desduplicación](#de-duplication) para más detalles). También hay que eliminar las que no tienen consentimiento.

En este ejemplo, filtramos los casos que queremos eliminar y los almacenamos en un dataframe separado, de forma que podamos describir los que fueron excluidos de la encuesta. A continuación, utilizamos la función anti\_join() de **dplyr** para eliminar estos casos descartados de los datos de nuestra encuesta.

**PELIGRO:** No puede tener valores faltantes en su variable de peso, ni en ninguna de las variables relevantes para el diseño de su encuesta (por ejemplo, edad, sexo, estratos o variables de agrupación).

Como se ha mencionado anteriormente, demostramos cómo añadir ponderaciones para tres diseños de estudio diferentes (estratificado, conglomerado y conglomerado estratificado). Estos requieren información sobre la población de origen y/o los conglomerados encuestados. Utilizaremos el código de conglomerado estratificado para este ejemplo, pero utiliza el que sea más apropiado para su diseño de estudio.

## Objetos de diseño de la encuesta

Cree un objeto de encuesta de acuerdo con el diseño de su estudio. Se utiliza de la misma manera que los dataframes para calcular las proporciones de peso, etc. Asegúrese de que todas las variables necesarias están creadas antes de esto.

Hay cuatro opciones, comente las que no utiliza: - Aleatorio simple - Estratificado - Conglomerado - Conglomerado estratificado

Para esta plantilla, supondremos que agrupamos las encuestas en dos estratos distintos (distritos sanitarios A y B). Por lo tanto, para obtener las estimaciones globales necesitamos haber combinado las ponderaciones de los grupos y de los estratos.

Como se ha mencionado anteriormente, hay dos paquetes disponibles para hacer esto. El clásico es **survey** y luego hay un paquete envolvente llamado **srvyr** que hace objetos y funciones amigables con tidyverse. Demostraremos ambos, pero ten en cuenta que la mayor parte del código de este capítulo utilizará objetos basados en srvyr. La única excepción es que el paquete **gtsummary** sólo acepta objetos de **encuesta**.

### Paquete de ****encuestas****

El paquete de **encuestas** utiliza efectivamente la codificación **básica** de R, por lo que no es posible utilizar tuberías (%>%) u otra sintaxis de dplyr. Con el paquete de **encuestas** utilizamos la función svydesign() para definir un objeto de encuesta con clusters, pesos y estratos adecuados.

**NOTA:** necesitamos utilizar la tilde (~) delante de las variables, esto es porque el paquete utiliza la sintaxis **base de** R de asignar variables basadas en fórmulas.

### Paquete ****Srvyr****

Con el paquete **srvyr** podemos utilizar la función as\_survey\_design(), que tiene los mismos argumentos que la anterior pero permite los tuberías (%>%), por lo que no es necesario utilizar la tilde (~).

## Análisis descriptivo

El análisis descriptivo básico y la visualización se tratan extensamente en otros capítulos del manual, por lo que no nos detendremos en ellos aquí. Para más detalles, consulte los capítulos sobre [tablas descriptivas](https://epirhandbook.com/descriptive-tables.html), [pruebas estadísticas](https://epirhandbook.com/simple-statistical-tests.html), [tablas para la presentación](https://epirhandbook.com/tables-for-presentation.html), [fundamentos de ggplot](https://epirhandbook.com/ggplot-basics.html) e [informes R markdown](https://epirhandbook.com/r-markdown-reports.html).

En este apartado nos centraremos en cómo investigar el sesgo de la muestra y visualizarlo. También veremos cómo visualizar el flujo de la población en un entorno de encuesta utilizando diagramas aluviales/sanitarios.

En general, debe considerar incluir los siguientes análisis descriptivos:

* Número final de agrupaciones, hogares e individuos incluidos
* Número de personas excluidas y motivos de la exclusión
* Mediana (rango) del número de hogares por grupo y de individuos por hogar

### Sesgo de muestreo

Compare las proporciones de cada grupo de edad entre su muestra y la población de origen. Esto es importante para poder resaltar el posible sesgo del muestreo. También puedes repetir esta operación para ver las distribuciones por sexo.

Ten en cuenta que estos valores p son sólo indicativos, y que una discusión descriptiva (o la visualización con las pirámides de edad que aparecen a continuación) de las distribuciones en su muestra de estudio en comparación con la población de origen es más importante que la prueba binomial en sí. Esto se debe a que el aumento del tamaño de la muestra suele dar lugar a diferencias que pueden ser irrelevantes después de ponderar los datos.

### Pirámides demográficas

Las pirámides demográficas (o de edad y sexo) son una forma sencilla de visualizar la distribución de la población de la encuesta. También vale la pena considerar la creación de [tablas descriptivas](https://epirhandbook.com/descriptive-tables.html) de edad y sexo por estratos de la encuesta. Demostraremos el uso del paquete **apyramid**, ya que permite las proporciones ponderadas utilizando nuestro objeto de diseño de la encuesta creado anteriormente. Otras opciones para crear [pirámides demográficas](https://epirhandbook.com/demographic-pyramids-and-likert-scales.html) se tratan ampliamente en ese capítulo del manual. También utilizaremos una función envolvente de **sitrep** llamada plot\_age\_pyramid() que ahorra algunas líneas de codificación para producir un gráfico con proporciones.

Al igual que con la prueba binomial formal de la diferencia, vista anteriormente en la sección de sesgo de muestreo, aquí estamos interesados en visualizar si nuestra población muestreada es sustancialmente diferente de la población de origen y si la ponderación corrige esta diferencia. Para ello, utilizaremos el paquete **patchwork** para mostrar nuestras visualizaciones **ggplot** una al lado de la otra; para más detalles, consulte la sección sobre la combinación de gráficos en el capítulo de [consejos de ggplot](https://epirhandbook.com/ggplot-tips.html?q=patch" \l "combine-plots) del manual. Visualizaremos nuestra población de origen, nuestra población de encuesta no ponderada y nuestra población de encuesta ponderada. También puede considerar la posibilidad de visualizar por cada estrato de su encuesta - en nuestro ejemplo aquí sería utilizando el argumento stack\_by = "health\_district" (ver ?plot\_age\_pyramid para más detalles).

**NOTA:** Los ejes x e y están invertidos en las pirámides

### Diagrama de aluvión/sangría

Visualizar los puntos de partida y los resultados de los individuos puede ser muy útil para obtener una visión general. Su aplicación es bastante obvia en el caso de las poblaciones móviles, pero hay muchas otras aplicaciones, como las cohortes o cualquier otra situación en la que haya transiciones de estados para los individuos. Estos diagramas tienen varios nombres diferentes, como conjuntos aluviales, sankey y paralelos; los detalles se encuentran en el capítulo del manual sobre [diagramas y gráficos](https://epirhandbook.com/diagrams-and-charts.html" \l "alluvialsankey-diagrams).

## Proporciones ponderadas

Esta sección detallará cómo producir tablas para recuentos y proporciones ponderadas, con los intervalos de confianza asociados y el efecto del diseño. Hay cuatro opciones diferentes que utilizan funciones de los siguientes paquetes: **survey**, **srvyr**, **sitrep** y **gtsummary**. Para una codificación mínima que produzca una tabla de estilo epidemiológico estándar, recomendaríamos la función sitrep - que es una envoltura para el código srvyr; Ten en cuenta, sin embargo, que esto no está todavía en CRAN y puede cambiar en el futuro. Por lo demás, es probable que el código de la **encuesta** sea el más estable a largo plazo, mientras que **srvyr** se adaptará mejor a los flujos de trabajo de tidyverse. Aunque las funciones de **gtsummary** tienen mucho potencial, parecen ser experimentales e incompletas en el momento de escribir este artículo.

### Paquete de ****encuestas****

Podemos utilizar la función svyciprop() de la **encuesta** para obtener las proporciones ponderadas y los correspondientes intervalos de confianza del 95%. Se puede extraer un efecto de diseño apropiado utilizando la función svymean() en lugar de svyprop(). Cabe señalar que svyprop() sólo parece aceptar variables entre 0 y 1 (o TRUE/FALSE), por lo que las variables categóricas no funcionarán.

**NOTA:** Las funciones de la **encuesta** también aceptan objetos de diseño **srvyr**, pero aquí hemos utilizado el objeto de diseño de **la encuesta** sólo por coherencia

Podemos combinar las funciones de **survey** mostradas arriba en una función que definimos nosotros mismos a continuación, llamada svy\_prop; y podemos entonces usar esa función junto con map() del paquete purrr para iterar sobre varias variables y crear una tabla. Consulte el capítulo de [iteración](https://epirhandbook.com/iteration-loops-and-lists.html) del manual para obtener más detalles sobre **purrr**.

### Paquete ****Srvyr****

Con **srvyr** podemos utilizar la sintaxis **de dplyr** para crear una tabla. Observe que se utiliza la función survey\_mean() y se especifica el argumento de la proporción, y también que se utiliza la misma función para calcular el efecto del diseño. Esto se debe a que **srvyr** envuelve las dos funciones del paquete de **encuestas** svyciprop() y svymean(), que se utilizan en la sección anterior.

**NOTA:** Tampoco parece posible obtener proporciones a partir de variables categóricas utilizando **srvyr**, si lo necesita, consulte la sección siguiente utilizando **sitrep**

Aquí también podríamos escribir una función para iterar sobre múltiples variables utilizando el paquete purrr. Consulte el capítulo de [iteración](https://epirhandbook.com/iteration-loops-and-lists.html) del manual para obtener más detalles sobre **purrr**.

### Paquete ****Sitrep****

La función tab\_survey() de **sitrep** es una envoltura para **srvyr**, que permite crear tablas ponderadas con una codificación mínima. También permite calcular proporciones ponderadas para variables categóricas.

### Paquete ****Gtsummary****

Con **gtsummary** no parece haber todavía funciones incorporadas para añadir intervalos de confianza o efecto de diseño. Aquí mostramos cómo definir una función para añadir intervalos de confianza y luego añadir intervalos de confianza a una tabla gtsummary creada con la función tbl\_svysummary().

## Ratios ponderados

Del mismo modo, para los ratios ponderados (como los ratios de mortalidad) puede utilizar el paquete **survey** o **srvyr**. También se pueden escribir funciones (similares a las anteriores) para iterar sobre varias variables. También podría crear una función para **gtsummary** como la anterior, pero actualmente no tiene una funcionalidad incorporada.

### Paquete de ****encuestas****

### Paquete ****Srvyr****

## Recursos

[Página de estadísticas de la UCLA](https://stats.idre.ucla.edu/r/seminars/survey-data-analysis-with-r/)

[Analizar los datos de la encuesta gratis](http://asdfree.com/)

[srvyr packge](http://gdfe.co/srvyr/)

[paquete gtsummary](http://www.danieldsjoberg.com/gtsummary/reference/index.html)

[Estudios de caso de la encuesta EPIET](https://github.com/EPIET/RapidAssessmentSurveys)

# #Análisis de supervivencia

{#survival-analysis}

## Resumen

El análisis de supervivencia se centra en la descripción, para un individuo o grupo de individuos determinado, de un punto definido de acontecimiento denominado **fracaso** (aparición de una enfermedad, curación de una enfermedad, muerte, recaída tras la respuesta al tratamiento...) que se produce tras un periodo de tiempo denominado tiempo de **fracaso** (o **tiempo de seguimiento** en los estudios basados en cohortes/poblaciones) durante el cual se observa a los individuos. Para determinar el tiempo de fracaso, es necesario entonces definir un tiempo de origen (que puede ser la fecha de inclusión, la fecha de diagnóstico...).

El objetivo de la inferencia para el análisis de supervivencia es entonces el tiempo entre un origen y un evento. En la investigación médica actual, se utiliza ampliamente en los estudios clínicos para evaluar el efecto de un tratamiento, por ejemplo, o en la epidemiología del cáncer para evaluar una gran variedad de medidas de supervivencia del cáncer.

Suele expresarse mediante la **probabilidad de supervivencia**, que es la probabilidad de que el suceso de interés no haya ocurrido en una duración t.

**Censura**: La censura se produce cuando al final del seguimiento, algunos de los individuos no han tenido el evento de interés, y por lo tanto su verdadero tiempo hasta el evento es desconocido. Aquí nos centraremos principalmente en la censura derecha, pero para más detalles sobre la censura y el análisis de supervivencia en general, puede consultar las referencias.

## Preparación

### Cargar paquetes

Para realizar análisis de supervivencia en R, uno de los paquetes más utilizados es el de **supervivencia**. Primero lo instalamos y luego lo cargamos, así como los demás paquetes que se utilizarán en esta sección:

En este manual destacamos p\_load() de **pacman**, que instala el paquete si es necesario y lo carga para su uso. También puede cargar los paquetes instalados con library() de R **base.** Consulte la página sobre [los fundamentos de R](#r-basics) para obtener más información sobre los paquetes de R.

Esta página explora los análisis de supervivencia utilizando linelist utilizada en la mayoría de las páginas anteriores y sobre la que aplicamos algunos cambios para tener unos datos de supervivencia adecuados.

### Importar los datos

Importamos los datos de casos de una epidemia de ébola simulada. Si quiere seguir el proceso, [clica para descargar linelist "limpia"](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/case_linelists/linelist_cleaned.rds) (como archivo .rds). Importe los datos con la función import() del paquete **rio** (maneja muchos tipos de archivos como .xlsx, .csv, .rds - vea la página de [importación y exportación](#import-and-export) para más detalles).

### Gestión y transformación de datos

En resumen, los datos de supervivencia pueden describirse con las tres características siguientes:

1. la variable dependiente o respuesta es el tiempo de espera hasta la ocurrencia de un evento bien definido,
2. las observaciones están censuradas, en el sentido de que para algunas unidades el evento de interés no ha ocurrido en el momento en que se analizan los datos, y
3. existen predictores o variables explicativas cuyo efecto sobre el tiempo de espera queremos evaluar o controlar.

Así, crearemos las diferentes variables necesarias para respetar esa estructura y ejecutaremos el análisis de supervivencia.

Lo definimos:

* un nuevo dataframe linelist\_surv para este análisis
* nuestro evento de interés como "la muerte" (por lo tanto, nuestra probabilidad de supervivencia será la probabilidad de estar vivo después de un cierto tiempo después del momento de origen),
* el tiempo de seguimiento (futime) como el tiempo transcurrido entre el momento del inicio y el momento del desenlace en días,
* Los pacientes censurados son aquellos que se recuperaron o para los que no se conoce el resultado final, es decir, no se observó el evento "muerte" (evento=0).

**ATENCIÓN:** Dado que en un estudio de cohortes real, la información sobre el momento de origen y el final del seguimiento se conoce dado que los individuos son observados, eliminaremos las observaciones en las que se desconozca la fecha de inicio o la fecha de desenlace. También se eliminarán los casos en los que la fecha de inicio sea posterior a la fecha de desenlace, ya que se consideran erróneos.

**SUGERENCIA:** Dado que el filtrado a mayor (>) o menor (<) de una fecha puede eliminar las filas con valores faltantes, la aplicación del filtro en las fechas incorrectas también eliminará las filas con fechas faltantes.

A continuación, utilizamos case\_when() para crear una columna age\_cat\_small en la que sólo hay 3 categorías de edad.

**CONSEJO:** Podemos verificar las nuevas columnas que hemos creado haciendo un resumen sobre el futime y una tabulación cruzada entre el evento y el resultado a partir del cual se ha creado. Además de esta verificación, es un buen hábito comunicar la mediana del tiempo de seguimiento al interpretar los resultados del análisis de supervivencia.

Ahora cruzamos la nueva var de age\_cat\_small y la antigua col de age\_cat para asegurarnos de que las asignaciones son correctas

Ahora revisamos las 10 primeras observaciones de los datos de linelist\_surv mirando las variables específicas (incluyendo las de nueva creación).

También podemos cruzar las columnas age\_cat\_small y gender para tener más detalles sobre la distribución de esta nueva columna por género. Utilizamos tabyl() y las funciones de adorno de **janitor** como se describe en la página de [tablas descriptivas](#descriptive-tables).

## Fundamentos del análisis de supervivencia

### Construir un objeto de tipo superviviente

Primero utilizaremos Surv() de **survival** para construir un objeto de supervivencia a partir de las columnas de tiempo de seguimiento y evento.

El resultado de este paso es producir un objeto de tipo Surv que condensa la información de tiempo y si el evento de interés (muerte) fue observado. Este objeto se utilizará en última instancia en el lado derecho de las fórmulas posteriores del modelo (véase [la documentación](https://cran.r-project.org/web/packages/survival/vignettes/survival.pdf)).

Para revisar, aquí están las primeras 10 filas de los datos de linelist\_surv, viendo sólo algunas columnas importantes.

Y aquí están los primeros 10 elementos de survobj. Se imprime como esencialmente un vector de tiempo de seguimiento, con "+" para representar si una observación fue censurada a la derecha. Vea cómo los números se alinean arriba y abajo.

### Realización de los primeros análisis

A continuación, iniciamos nuestro análisis utilizando la función survfit() para producir un objeto survfit, que se ajusta a los cálculos por defecto para las estimaciones de **Kaplan Meier** (KM) de la curva de supervivencia global (marginal), que son de hecho una función escalonada con saltos en los tiempos de los eventos observados. El objeto survfit final contiene una o más curvas de supervivencia y se crea utilizando el objeto Surv como variable de respuesta en la fórmula del modelo.

**NOTA:** La estimación de Kaplan-Meier es una estimación no paramétrica de máxima verosimilitud (MLE) de la función de supervivencia. . (ver recursos para más información).

El resumen de este objeto survfit dará lo que se llama una tabla de vida. Para cada paso de tiempo del seguimiento (tiempo) en el que ocurrió un evento (en orden ascendente):

* el número de personas que estaban en riesgo de desarrollar el evento (personas que aún no tenían el evento ni estaban censuradas: n.riesgo)
* los que sí desarrollaron el evento (n.evento)
* y de lo anterior: la probabilidad de no desarrollar el evento (probabilidad de no morir, o de sobrevivir más allá de ese tiempo específico)
* por último, se obtienen y muestran el error estándar y el intervalo de confianza de esa probabilidad

Ajustamos las estimaciones de KM mediante la fórmula en la que el objeto Surv "survobj" anterior es la variable de respuesta. "~ 1" precisa que ejecutamos el modelo para la supervivencia global.

Al utilizar summary() podemos añadir la opción tiempos y especificar ciertos tiempos en los que queremos ver la información de supervivencia

También podemos utilizar la función print(). El argumento print.rmean = TRUE se utiliza para obtener el tiempo medio de supervivencia y su error estándar (se).

**NOTA:** El tiempo medio de supervivencia restringido (RMST) es una medida de supervivencia específica cada vez más utilizada en el análisis de supervivencia del cáncer y que suele definirse como el área bajo la curva de supervivencia, dado que observamos a los pacientes hasta el tiempo restringido T (más detalles en la sección Recursos).

**TIP:** Podemos crear el objeto surv directamente en la función survfit() y ahorrarnos una línea de código. Esto se verá como: linelistsurv\_quick <- survfit(Surv(futime, event) ~ 1, data=linelist\_surv).

### Peligro acumulado

Además de la función summary(), también podemos utilizar la función str() que da más detalles sobre la estructura del objeto survfit(). Se trata de una lista de 16 elementos.

Entre estos elementos hay uno importante: el cumhaz, que es un vector numérico. Se puede trazar para mostrar el **peligro acumulado,** siendo el **peligro** la **tasa instantánea de ocurrencia del evento** (ver referencias).

### Trazado de curvas de Kaplan-Meir

Una vez ajustadas las estimaciones de KM, podemos visualizar la probabilidad de estar vivo a lo largo de un tiempo determinado utilizando la función básica plot() que dibuja la "curva de Kaplan-Meier". En otras palabras, la curva de abajo es una ilustración convencional de la experiencia de supervivencia en todo el grupo de pacientes.

Podemos verificar rápidamente el tiempo de seguimiento mínimo y máximo en la curva.

Una forma fácil de interpretarlo es decir que en el momento cero, todos los participantes siguen vivos y la probabilidad de supervivencia es entonces del 100%. Esta probabilidad disminuye con el tiempo a medida que los pacientes mueren. La proporción de participantes que sobreviven más allá de los 60 días de seguimiento se sitúa en torno al 40%.

El intervalo de confianza de las estimaciones de supervivencia de KM también se traza por defecto y puede descartarse añadiendo la opción conf.int = FALSE al comando plot().

Dado que el evento de interés es la "muerte", dibujar una curva que describa los complementos de las proporciones de supervivencia llevará a dibujar las proporciones de mortalidad acumulada. Esto puede hacerse con lines(), que añade información a un gráfico existente.

## Comparación de las curvas de supervivencia

Para comparar la supervivencia dentro de los diferentes grupos de nuestros participantes o pacientes observados, es posible que tengamos que observar primero sus respectivas curvas de supervivencia y luego realizar pruebas para evaluar la diferencia entre grupos independientes. Esta comparación puede referirse a grupos basados en el género, la edad, el tratamiento, la comorbilidad...

### Prueba de rango logarítmico

La prueba de rango logarítmico es una prueba popular que compara toda la experiencia de supervivencia entre dos o más grupos independientes y puede considerarse como una prueba de si las curvas de supervivencia son idénticas (se superponen) o no (hipótesis nula de no diferencia de supervivencia entre los grupos). La función survdiff() del **paquete de supervivencia** permite ejecutar la prueba de rango logarítmico cuando especificamos rho = 0 (que es el valor predeterminado). Los resultados de la prueba dan un estadístico chi-cuadrado junto con un valor p, ya que el estadístico log rank se distribuye aproximadamente como un estadístico de prueba chi-cuadrado.

En primer lugar, tratamos de comparar las curvas de supervivencia por grupos de género. Para ello, primero intentamos visualizarlo (comprobar si las dos curvas de supervivencia se superponen). Se creará un nuevo objeto survfit con una fórmula ligeramente diferente. Luego se creará el objeto survdiff.

Al suministrar ~ género como lado derecho de la fórmula, ya no trazamos la supervivencia global sino por género.

Ahora podemos trazar las curvas de supervivencia por género. Observe el orden de los niveles de los estratos en la columna de género antes de definir los colores y la leyenda.

Y ahora podemos calcular la prueba de la diferencia entre las curvas de supervivencia utilizando survdiff()

Vemos que la curva de supervivencia de las mujeres y la de los hombres se superponen y la prueba de rango logarítmico no da pruebas de una diferencia de supervivencia entre mujeres y hombres.

Algunos otros paquetes de R permiten ilustrar curvas de supervivencia para diferentes grupos y probar la diferencia de una sola vez. Utilizando la función ggsurvplot() del paquete **survminer**, también podemos incluir en nuestra curva las tablas de riesgo impresas para cada grupo, así como el valor p de la prueba log-rank.

**ATENCIÓN:** las funciones **de survminer** requieren que especifique el objeto de supervivencia y que vuelva a especificar los datos utilizados para ajustar el objeto de supervivencia. Recuerde hacer esto para evitar mensajes de error no específicos.

También podemos comprobar si hay diferencias en la supervivencia según la fuente de infección (fuente de contaminación).

En este caso, la prueba de rango logarítmico da pruebas suficientes de una diferencia en las probabilidades de supervivencia a alfa= 0,005. Las probabilidades de supervivencia de los pacientes que se infectaron en los funerales son mayores que las de los pacientes que se infectaron en otros lugares, lo que sugiere un beneficio para la supervivencia.

## Análisis de regresión de Cox

La regresión de riesgos proporcionales de Cox es una de las técnicas de regresión más populares para el análisis de supervivencia. También se pueden utilizar otros modelos, ya que el modelo de Cox requiere supuestos importantes que deben verificarse para un uso adecuado, como el supuesto de riesgos proporcionales: véanse las referencias.

En un modelo de regresión de riesgos proporcionales de Cox, la medida del efecto es la tasa de **riesgo (**HR), que es el riesgo de fracaso (o el riesgo de muerte en nuestro ejemplo), dado que el participante ha sobrevivido hasta un momento específico. Normalmente, estamos interesados en comparar grupos independientes con respecto a sus riesgos, y utilizamos una razón de riesgo, que es análoga a una razón de probabilidades en el entorno del análisis de regresión logística múltiple. La función cox.ph() del paquete de supervivencia se utiliza para ajustar el modelo. La función cox.zph() del paquete de **supervivencia** puede utilizarse para probar la suposición de riesgos proporcionales para un ajuste del modelo de regresión de Cox.

**NOTA:** Una probabilidad debe estar en el rango de 0 a 1. Sin embargo, el peligro representa el número esperado de eventos por una unidad de tiempo.

* Si la razón de riesgo de un predictor es cercana a 1, entonces ese predictor no afecta a la supervivencia,
* si el HR es inferior a 1, entonces el predictor es protector (es decir, está asociado a una mejor supervivencia),
* y si el HR es mayor que 1, entonces el predictor se asocia a un mayor riesgo (o a una menor supervivencia).

### Ajuste de un modelo de Cox

Primero podemos ajustar un modelo para evaluar el efecto de la edad y el sexo en la supervivencia. Con sólo imprimir el modelo, tenemos la información sobre:

* los coeficientes de regresión estimados coef que cuantifican la asociación entre los predictores y el resultado,
* su exponencial (para su interpretación, exp(coef)) que produce la razón de riesgo,
* su error estándar se(coef),
* la puntuación z: cuántos errores estándar se aleja el coeficiente estimado de 0,
* y el valor p: la probabilidad de que el coeficiente estimado sea 0.

La función summary() aplicada al objeto del modelo cox ofrece más información, como el intervalo de confianza de la HR estimada y las diferentes puntuaciones de la prueba.

El efecto de la primera covariable, el género, se presenta en la primera fila. Se imprime genderm (masculino), lo que implica que el primer nivel de estrato ("f"), es decir, el grupo femenino, es el grupo de referencia para el género. Por lo tanto, la interpretación del parámetro de la prueba es la de los hombres en comparación con las mujeres. El valor p indica que no hay pruebas suficientes de un efecto del género sobre el peligro esperado o de una asociación entre el género y la mortalidad por todas las causas.

La misma falta de pruebas se observa en relación con el grupo de edad.

Fue interesante ejecutar el modelo y observar los resultados, pero un primer vistazo para verificar si se respetan los supuestos de riesgos proporcionales podría ayudar a ahorrar tiempo.

**NOTA: Se** puede especificar un segundo argumento llamado método cuando se calcula el modelo cox, que determina cómo se manejan los empates. El valor por defecto es "efron", y las otras opciones son "breslow" y "exact".

En otro modelo añadimos más factores de riesgo, como el origen de la infección y el número de días entre la fecha de inicio y el ingreso. Esta vez, primero verificamos la hipótesis de riesgos proporcionales antes de seguir adelante.

En este modelo, hemos incluido un predictor continuo (days\_onset\_hosp). En este caso, interpretamos las estimaciones de los parámetros como el aumento del logaritmo esperado del riesgo relativo por cada aumento de una unidad en el predictor, manteniendo los demás predictores constantes. Primero verificamos el supuesto de riesgos proporcionales.

La verificación gráfica de esta suposición puede realizarse con la función ggcoxzph() del paquete **survminer**.

Los resultados del modelo indican que existe una asociación negativa entre la duración del inicio del ingreso y la mortalidad por todas las causas. El riesgo esperado es 0,9 veces menor en una persona que ingresa un día más tarde que otra, manteniendo el género constante. O, en una explicación más directa, un aumento de una unidad en la duración del inicio al ingreso se asocia con una disminución del 10,7% (coef \*100) en el riesgo de muerte.

Los resultados muestran también una asociación positiva entre la fuente de infección y la mortalidad por todas las causas. Es decir, hay un mayor riesgo de muerte (1,21 veces) para los pacientes que tuvieron una fuente de infección distinta de los funerales.

Podemos comprobar esta relación con una tabla:

Habría que considerar e investigar por qué existe esta asociación en los datos. Una posible explicación podría ser que los pacientes que viven lo suficiente como para ser ingresados más tarde tenían una enfermedad menos grave para empezar. Otra explicación, quizá más probable, es que, dado que utilizamos unos datos falsos simulados, este patrón no refleja la realidad.

### Gráficos forestales

A continuación, podemos visualizar los resultados del modelo cox utilizando los prácticos gráficos de bosque con la función ggforest() del **paquete survminer**.

## Covariables dependientes del tiempo en los modelos de supervivencia

Algunas de las siguientes secciones han sido adaptadas con permiso de una excelente [introducción al análisis de supervivencia en R](https://www.emilyzabor.com/tutorials/survival_analysis_in_r_tutorial.html) por [la Dra. Emily Zabor](https://www.emilyzabor.com/)

En la última sección hemos tratado el uso de la regresión de Cox para examinar las asociaciones entre las covariables de interés y los resultados de supervivencia, pero estos análisis dependen de que la covariable se mida en la línea de base, es decir, antes de que comience el tiempo de seguimiento del evento.

¿Qué ocurre si está interesado en una covariable que se mide **después del** tiempo de seguimiento? O, ¿qué pasa si tiene una covariable que puede cambiar con el tiempo?

Por ejemplo, tal vez esté trabajando con datos clínicos en los que repite medidas de valores de laboratorio del hospital que pueden cambiar con el tiempo. Este es un ejemplo de una **covariable dependiente del tiempo**. Para abordar esto se necesita una configuración especial, pero afortunadamente el modelo cox es muy flexible y este tipo de datos también puede ser modelado con herramientas del paquete de **supervivencia**.

### Configuración de covariables dependientes del tiempo

El análisis de covariables dependientes del tiempo en R requiere la configuración de unos datos especial. Si está interesado, vea el documento más detallado sobre esto por el autor del paquete de **supervivencia** [Using Time Dependent Covariates and Time Dependent Coefficients in the Cox Model](https://cran.r-project.org/web/packages/survival/vignettes/timedep.pdf).

Para ello, utilizaremos un nuevo conjunto de datos del paquete SemiCompRisks denominado BMT, que incluye datos de 137 pacientes de trasplante de médula ósea. Las variables en las que nos centraremos son:

* T1 - tiempo (en días) hasta la muerte o el último seguimiento
* delta1 - indicador de muerte; 1-Muerto, 0-Vivo
* TA - tiempo (en días) hasta la enfermedad de injerto contra huésped aguda
* deltaA - indicador de la enfermedad de injerto contra huésped aguda;
  + 1 - Desarrolló la enfermedad de injerto contra huésped aguda
  + 0 - Nunca desarrolló la enfermedad de injerto contra huésped aguda

Cargaremos este conjunto de datos del paquete **survival** utilizando el comando **base de** R data(), que puede utilizarse para cargar datos que ya están incluidos en un paquete de R que se ha cargado. El dataframe BMT aparecerá en su entorno R.

#### Añadir un identificador único de paciente

No hay una columna de identificación única en los datos de BMT, que es necesaria para crear el tipo de conjunto de datos que queremos. Así que utilizamos la función rowid\_to\_column() del paquete **tibble de tidyverse** para crear una nueva columna de identificación llamada my\_id (añade una columna al principio del dataframe con identificadores de fila secuenciales, empezando por el 1). Llamamos al dataframe bmt.

El conjunto de datos tiene ahora este aspecto:

#### Ampliar las filas de pacientes

A continuación, utilizaremos la función tmerge() con las funciones de ayuda event() y tdc() para crear el conjunto de datos reestructurado. Nuestro objetivo es reestructurar el conjunto de datos para crear una fila separada para cada paciente por cada intervalo de tiempo en el que tengan un valor diferente de deltaA. En este caso, cada paciente puede tener como máximo dos filas dependiendo de si desarrollaron la enfermedad aguda de injerto contra huésped durante el periodo de recogida de datos. Llamaremos a nuestro nuevo indicador para el desarrollo de la enfermedad aguda de injerto contra huésped agvhd.

* tmerge() crea unos datos largo con múltiples intervalos de tiempo para los diferentes valores de las covariables de cada paciente
* event() crea el nuevo indicador de eventos para que vaya con los intervalos de tiempo recién creados
* tdc() crea la columna de covarianza dependiente del tiempo, agvhd, para que vaya con los intervalos de tiempo recién creados

Para ver qué hace esto, veamos los datos de los 5 primeros pacientes individuales.

Las variables de interés en los datos originales tenían este aspecto:

El nuevo conjunto de datos para estos mismos pacientes tiene el siguiente aspecto:

Ahora algunos de nuestros pacientes tienen dos filas en el conjunto de datos correspondientes a intervalos en los que tienen un valor diferente de nuestra nueva variable, agvhd. Por ejemplo, el paciente 1 tiene ahora dos filas con un valor de agvhd de cero desde el tiempo 0 hasta el tiempo 67, y un valor de 1 desde el tiempo 67 hasta el tiempo 2081.

### Regresión de Cox con covariables dependientes del tiempo

Ahora que hemos remodelado nuestros datos y añadido la nueva variable aghvd dependiente del tiempo, vamos a ajustar un modelo de regresión cox simple de una sola variable. Podemos utilizar la misma función coxph() que antes, sólo tenemos que cambiar nuestra función Surv() para especificar tanto el tiempo de inicio como el de finalización de cada intervalo utilizando los argumentos time1 = y time2 =.

De nuevo, visualizaremos los resultados de nuestro modelo cox utilizando la función ggforest() del **paquete survminer**.:

Como se puede ver en el gráfico de bosque, el intervalo de confianza y el valor p, no parece haber una fuerte asociación entre la muerte y la enfermedad de injerto contra huésped aguda en el contexto de nuestro modelo simple.

## Recursos

[Análisis de supervivencia Parte I: Conceptos básicos y primeros análisis](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2394262/)

[Análisis de supervivencia en R](https://www.emilyzabor.com/tutorials/survival_analysis_in_r_tutorial.html)

[Análisis de supervivencia en la investigación de enfermedades infecciosas: La descripción de eventos en el tiempo](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2954271/)

[Capítulo sobre modelos de supervivencia avanzados Princeton](https://data.princeton.edu/wws509/notes/c7.pdf)

[Uso de covariables y coeficientes dependientes del tiempo en el modelo de Cox](https://cran.r-project.org/web/packages/survival/vignettes/timedep.pdf)

[Hoja de trucos de análisis de supervivencia R](https://publicifsv.sund.ku.dk/~ts/survival/survival-cheat.pdf)

[Hoja de trucos de Survminer](https://paulvanderlaken.files.wordpress.com/2017/08/survminer_cheatsheet.pdf)

[Documento sobre diferentes medidas de supervivencia para datos de registros de cáncer con Rcode proporcionado como material suplementario](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6322561/)

# #Conceptos básicos de los SIG

{#gis-basics}

## Resumen

Los aspectos espaciales de sus datos pueden proporcionar mucha información sobre la situación del brote, y responder a preguntas como:

* ¿Dónde están los focos actuales de la enfermedad?
* ¿Cómo han cambiado los puntos conflictivos con el tiempo?
* ¿Cómo es el acceso a las instalaciones sanitarias? ¿Se necesitan mejoras?

El enfoque actual de esta página de los SIG (GIS, Sistemas de información geográfico) de es abordar las necesidades de la epidemiología aplicada en su respuesta a los brotes. Exploraremos los métodos básicos de visualización de datos espaciales utilizando los paquetes **tmap** y **ggplot2**. También recorreremos algunos de los métodos básicos de gestión y consulta de datos espaciales con el paquete **sf**. Por último, abordaremos brevemente conceptos de estadística espacial como las relaciones espaciales, la autocorrelación espacial y la regresión espacial utilizando el paquete **spdep**.

## Términos clave

A continuación presentamos algunos términos clave. Para una introducción completa a los SIG y al análisis espacial, le sugerimos que revise uno de los tutoriales o cursos más largos que aparecen en la sección de Referencias.

**Sistema de Información Geográfica (SIG)** - Un SIG es un marco o entorno para recopilar, gestionar, analizar y visualizar datos espaciales.

### Software GIS

Algunos de los programas de SIG más conocidos permiten la interacción "apuntar y clicar" para el desarrollo de mapas y el análisis espacial. Estas herramientas tienen ventajas como no tener que aprender código y la facilidad de seleccionar y colocar manualmente los iconos y características en un mapa. He aquí dos de los más populares:

**ArcGIS** - Un software comercial de SIG desarrollado por la empresa ESRI, que es muy popular pero bastante caro

**QGIS** - Un software SIG gratuito de código abierto que puede hacer casi todo lo que ArcGIS puede hacer. Puedes [descargar QGIS aquí](https://qgis.org/en/site/forusers/download.html)

El uso de R como SIG puede parecer más intimidante al principio porque, en lugar de "apuntar y clicar", tiene una "interfaz de línea de comandos" (hay que codificar para adquirir el resultado deseado). Sin embargo, esto es una gran ventaja si necesita producir mapas repetidamente o crear un análisis que sea reproducible.

### Datos espaciales

Las dos formas principales de datos espaciales utilizadas en los SIG son los datos vectoriales y los ráster:

**Datos vectoriales** - El formato más común de datos espaciales utilizado en los SIG, los datos vectoriales se componen de características geométricas de vértices y trayectorias. Los datos espaciales vectoriales pueden dividirse a su vez en tres tipos ampliamente utilizados:

* Puntos - Un punto consiste en un par de coordenadas (x,y) que representa una ubicación específica en un sistema de coordenadas. Los puntos son la forma más básica de datos espaciales, y pueden utilizarse para denotar un caso (por ejemplo, el domicilio de un paciente) o una ubicación (por ejemplo, un hospital) en un mapa.
* Líneas - Una línea está compuesta por dos puntos conectados. Las líneas tienen una longitud y pueden utilizarse para indicar cosas como carreteras o ríos.
* Polígonos - Un polígono está compuesto por al menos tres segmentos de línea conectados por puntos. Las características de los polígonos tienen una longitud (es decir, el perímetro del área) así como una medida de área. Los polígonos pueden utilizarse para señalar una zona (por ejemplo, un pueblo) o una estructura (por ejemplo, la superficie real de un hospital).

**Datos ráster** - Un formato alternativo para los datos espaciales, los datos ráster son una matriz de celdas (por ejemplo, píxeles) en la que cada celda contiene información como la altura, la temperatura, la pendiente, la cubierta forestal, etc. Suelen ser fotografías aéreas, imágenes de satélite, etc. Los rásteres también pueden utilizarse como "mapas base" debajo de los datos vectoriales.

### Visualización de datos espaciales

Para representar visualmente los datos espaciales en un mapa, el software SIG requiere que se proporcione suficiente información sobre dónde deben estar las diferentes características, en relación unas con otras. Si se utilizan datos vectoriales, como ocurre en la mayoría de los casos, esta información suele almacenarse en un archivo shapefile:

**Shapefiles** - Un shapefile es un formato de datos común para almacenar datos espaciales "vectoriales" consistentes en líneas, puntos o polígonos. Un shapefile es en realidad una colección de al menos tres archivos - .shp, .shx y .dbf. Todos estos archivos subcomponentes deben estar presentes en un determinado directorio (carpeta) para que el shapefile sea legible. Estos archivos asociados pueden comprimirse en una carpeta ZIP para enviarlos por correo electrónico o descargarlos de un sitio web.

El shapefile contendrá información sobre las características en sí mismas, así como su ubicación en la superficie de la Tierra. Esto es importante porque, aunque la Tierra es un globo terráqueo, los mapas suelen ser bidimensionales; las decisiones sobre cómo "aplanar" los datos espaciales pueden tener un gran impacto en el aspecto y la interpretación del mapa resultante.

**Sistemas de referencia de coordenadas (SRC)** - Un SRC es un sistema basado en coordenadas que se utiliza para localizar accidentes geográficos en la superficie de la Tierra. Tiene unos cuantos componentes clave:

* Sistema de coordenadas - Hay muchos sistemas de coordenadas diferentes, así que asegúrate de saber de qué sistema son tus coordenadas. Los grados de latitud/longitud son comunes, pero también podría ver coordenadas [UTM](https://www.maptools.com/tutorials/utm/quick_guide).
* Unidades - Saber cuáles son las unidades de su sistema de coordenadas (por ejemplo, grados decimales, metros)
* Datum - Una versión particular modelada de la Tierra. Estos han sido revisados a lo largo de los años, así que asegúrate de que tus capas de mapa utilizan el mismo datum.
* Proyección - Referencia a la ecuación matemática que se utilizó para proyectar la tierra realmente redonda sobre una superficie plana (mapa).

Recuerde que puede resumir los datos espaciales sin utilizar las herramientas cartográficas que se muestran a continuación. A veces basta con una simple tabla por zonas geográficas (por ejemplo, distrito, país, etc.).

## Introducción a los SIG

Hay un par de elementos clave que deberá tener y en los que deberá pensar para hacer un mapa. Entre ellos están:

* unos datos: puede estar en un formato de datos espaciales (como los shapefiles, como se ha indicado anteriormente) o puede no estar en un formato espacial (por ejemplo, sólo como un csv).
* Si su conjunto de datos no está en formato espacial, también necesitará unos datos **de referencia**. Los datos de referencia consisten en la representación espacial de los datos y los **atributos** relacionados, que incluirían el material que contiene la información de ubicación y dirección de características específicas.
  + Si está trabajando con límites geográficos predefinidos (por ejemplo, regiones administrativas), los archivos shape de referencia suelen estar disponibles para su descarga de forma gratuita desde una agencia gubernamental u organización de intercambio de datos. En caso de duda, un buen punto de partida es buscar en Google "[regiones] shapefile"
  + Si dispone de información de direcciones, pero no de latitud y longitud, puede que tenga que utilizar un **motor de geocodificación** para obtener los datos de referencia espacial de sus registros.
* Una idea sobre **cómo quiere presentar** la información de sus conjuntos de datos a su público objetivo. Hay muchos tipos diferentes de mapas, y es importante pensar qué tipo de mapa se ajusta mejor a sus necesidades.

### Tipos de mapas para visualizar sus datos

**Mapa coroplético**: tipo de mapa temático en el que se utilizan colores, sombreados o patrones para representar regiones geográficas en relación con su valor de un atributo. Por ejemplo, un valor mayor podría indicarse con un color más oscuro que un valor menor. Este tipo de mapa es particularmente útil cuando se visualiza una variable y cómo cambia a través de regiones o áreas geopolíticas definidas.

**Mapa térmico de densidad de casos**: un tipo de mapa temático en el que se utilizan colores para representar la intensidad de un valor, pero que no utiliza regiones definidas ni límites geopolíticos para agrupar los datos. Este tipo de mapa se suele utilizar para mostrar "puntos calientes" o zonas con una alta densidad o concentración de puntos.

**Mapa de densidad de puntos**: un tipo de mapa temático que utiliza puntos para representar los valores de los atributos en sus datos. Este tipo de mapa es el más adecuado para visualizar la dispersión de los datos y buscar clusters visualmente.

Mapa de **símbolos proporcionales (mapa de símbolos graduados)**: es un mapa temático similar a un mapa coroplético, pero en lugar de utilizar el color para indicar el valor de un atributo, utiliza un símbolo (normalmente un círculo) en relación con el valor. Por ejemplo, un valor mayor podría indicarse con un símbolo mayor que un valor menor. Este tipo de mapa se utiliza mejor cuando se quiere visualizar el tamaño o la cantidad de los datos en las distintas regiones geográficas.

También puede combinar varios tipos de visualizaciones diferentes para mostrar patrones geográficos complejos. Por ejemplo, los casos (puntos) del mapa siguiente están coloreados según su centro de salud más cercano (véase la leyenda). Los círculos rojos grandes muestran las zonas de captación de los centros de salud de un determinado radio, y los puntos rojos brillantes los que estaban fuera de cualquier zona de captación:

Nota: El enfoque principal de esta página del SIG se basa en el contexto de la respuesta a los brotes de campo. Por lo tanto, el contenido de la página cubrirá las manipulaciones, visualizaciones y análisis básicos de datos espaciales.

## Preparación

### Cargar paquetes

Este trozo de código muestra la carga de los paquetes necesarios para los análisis. En este manual destacamos p\_load() de **pacman**, que instala el paquete si es necesario y lo carga para su uso. También puede cargar los paquetes instalados con library() de R **base.** Consulte la página sobre [los fundamentos de R](#r-basics) para obtener más información sobre los paquetes de R.

Puedes ver un resumen de todos los paquetes de R que se ocupan de los datos espaciales en la ["Vista de tareas espaciales" de CRAN](https://cran.r-project.org/web/views/Spatial.html).

### Datos del caso de muestra

Para fines de demostración, trabajaremos con una muestra aleatoria de 1000 casos del dataframe del listado de la epidemia de ébola simulada (computacionalmente, trabajar con menos casos es más fácil de mostrar en este manual). Si quiere seguir el proceso, [clica para descargar linelist "limpia"](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/case_linelists/linelist_cleaned.rds) (como archivo .rds).

Dado que estamos tomando una muestra aleatoria de los casos, sus resultados pueden ser ligeramente diferentes de los que se muestran aquí cuando ejecutes el código por tu cuenta.

Importar datos con la función import() del paquete **rio** (maneja muchos tipos de archivos como .xlsx, .csv, .rds - ver la página de [importación y exportación](#import-and-export) para más detalles).

A continuación, seleccionamos una muestra aleatoria de 1000 filas utilizando sample() de la **base** R.

Ahora queremos convertir esta lista de líneas, que es de tipo dataframe, en un objeto de tipo "sf" (spatial features). Dado que linelist tiene dos columnas "lon" y "lat" que representan la longitud y latitud de la residencia de cada caso, esto será fácil.

Utilizamos el paquete **sf** (spatial features) y su función st\_as\_sf() para crear el nuevo objeto que llamamos linelist\_sf. Este nuevo objeto tiene esencialmente el mismo aspecto que linelist, pero las columnas lon y lat han sido designadas como columnas de coordenadas, y se ha asignado un sistema de referencia de coordenadas (CRS) para cuando se muestren los puntos. El 4326 identifica nuestras coordenadas como basadas en el [Sistema Geodésico Mundial 1984 (WGS84)](https://gisgeography.com/wgs84-world-geodetic-system/) - que es el estándar para las coordenadas GPS.

Este es el aspecto del dataframe original del listado. En esta demostración, sólo utilizaremos la columna date\_onset y geometry (que se construyó a partir de los campos de longitud y latitud anteriores y es la última columna del dataframe).

### Archivos shapefiles de los límites de la administración

**Sierra Leona: Archivos shapefiles de los límites administrativos**

De antemano, hemos descargado todos los límites administrativos de Sierra Leona del [sitio web de](https://data.humdata.org/dataset/sierra-leone-all-ad-min-level-boundaries) Humanitarian Data Exchange (HDX) [aquí](https://data.humdata.org/dataset/sierra-leone-all-ad-min-level-boundaries). Como alternativa, puede descargar estos y todos los demás datos de ejemplo para este manual a través de nuestro paquete R, como se explica en la página de [descarga del manual y los datos](#download-handbook-and-data).

Ahora vamos a hacer lo siguiente para guardar el shapefile del nivel 3 de administración en R:

1. Importar el shapefile
2. Limpiar los nombres de las columnas
3. Filtrar las filas para mantener sólo las áreas de interés

Para importar un shapefile utilizamos la función read\_sf() de **sf**. Se le proporciona la ruta del archivo a través de here(). - En nuestro caso el archivo se encuentra dentro de nuestro proyecto R en las subcarpetas "data", "gis" y "shp", con nombre de archivo "sle\_adm3.shp" (para más información ver las páginas sobre [Importación y exportación](#import-and-export) y [proyectos R](#r-projects)). Tendrá que proporcionar su propia ruta de archivo.

A continuación utilizamos clean\_names() del paquete **janitor** para estandarizar los nombres de las columnas del shapefile. También utilizamos filter() para mantener sólo las filas con admin2name de "Western Area Urban" o "Western Area Rural".

A continuación puede ver el aspecto del shapefile después de la importación y la limpieza. Desplácese a la derecha para ver cómo hay columnas con el nivel de administración 0 (país), el nivel de administración 1, el nivel de administración 2 y, finalmente, el nivel de administración 3. Cada nivel tiene un nombre de carácter y un identificador único "pcode". El pcode se expande con cada nivel de administración creciente, por ejemplo, SL (Sierra Leona) -> SL04 (Occidental) -> SL0410 (Zona Occidental Rural) -> SL040101 (Koya Rural).

### Datos de población

**Sierra Leona: Población por ADM3**

Estos datos pueden descargarse de nuevo de HDX (enlace [aquí](https://data.humdata.org/dataset/sierra-leone-population)) o a través de nuestro paquete R **epirhandbook** como se explica [en esta página](#download-handbook-and-data). Utilizamos import() para cargar el archivo .csv. También pasamos el archivo importado a clean\_names() para estandarizar la sintaxis de los nombres de las columnas.

Este es el aspecto del archivo de población. Desplácese a la derecha para ver cómo cada jurisdicción tiene columnas con la población masculina, la población femenina, la población total y el desglose de la población en columnas por grupos de edad.

### Instalaciones sanitarias

**Sierra Leona: Datos de los centros de salud de OpenStreetMap**

Una vez más, hemos descargado las ubicaciones de los centros de salud desde el HDX [aquí](https://data.humdata.org/dataset/hotosm_sierra_leone_health_facilities) o mediante las instrucciones de la página de [descarga de manuales y datos](#download-handbook-and-data).

Importamos el shapefile de puntos de las instalaciones con read\_sf(), limpiamos de nuevo los nombres de las columnas y filtramos para mantener sólo los puntos etiquetados como "hospital", "clínica" o "médicos".

Aquí está el dataframe resultante - desplácese a la derecha para ver el nombre de la instalación y las coordenadas geométricas.

## Trazado de coordenadas

La forma más sencilla de trazar las coordenadas X-Y (longitud/latitud, puntos), en este caso de los casos, es dibujarlas como puntos directamente desde el objeto linelist\_sf que creamos en la sección de preparación.

El paquete **tmap** ofrece capacidades de mapeo simples tanto para el modo estático (modo "plot") como para el interactivo (modo "view") con sólo unas pocas líneas de código. La sintaxis de tmap es similar a la de **ggplot2**, de forma que los comandos se añaden unos a otros con +. Lea más detalles en esta [viñeta](https://cran.r-project.org/web/packages/tmap/vignettes/tmap-getstarted.html).

1. Establezca el modo **tmap**. En este caso utilizaremos el modo "plot", que produce salidas estáticas.

A continuación, los puntos se trazan solos. tm\_shape() se proporciona con los objetos linelist\_sf. A continuación, añadimos puntos mediante tm\_dots(), especificando el tamaño y el color. Como linelist\_sf es un objeto sf, ya hemos designado las dos columnas que contienen las coordenadas lat/long y el sistema de referencia de coordenadas (CRS):

Por sí solos, los puntos no nos dicen mucho. Así que también deberíamos trazar los límites administrativos:

De nuevo utilizamos tm\_shape() (ver [documentación](https://www.rdocumentation.org/packages/tmap/versions/3.3/topics/tm_shape)) pero en lugar de proporcionar el shapefile de los puntos del caso, proporcionamos el shapefile de los límites administrativos (polígonos).

Con el argumento bbox = (bbox significa "bounding box") podemos especificar los límites de las coordenadas. Primero mostramos la visualización del mapa sin bbox, y luego con él.

Y ahora tanto los puntos como los polígonos juntos:

Para leer una buena comparación de las opciones de mapeo en R, consulte esta [entrada del blog](https://rstudio-pubs-static.s3.amazonaws.com/324400_69a673183ba449e9af4011b1eeb456b9.html).

## Uniones espaciales

Es posible que esté familiarizado con la unión de datos de unos datos a otro. En la página de [unión de datos](#joining-data) de este manual se tratan varios métodos. Una unión espacial tiene un propósito similar, pero aprovecha las relaciones espaciales. En lugar de confiar en los valores comunes de las columnas para hacer coincidir correctamente las observaciones, puede utilizar sus relaciones espaciales, como que una característica esté dentro de otra, o que sea la vecina más cercana de otra, o que esté dentro de un búfer de un determinado radio de otra, etc.

El paquete **sf** ofrece varios métodos para las uniones espaciales. Consulte más documentación sobre el método st\_join y los tipos de uniones espaciales en esta [referencia](https://r-spatial.github.io/sf/reference/geos_binary_pred.html).

### Puntos en el polígono

**Asignación espacial de unidades administrativas a los casos**

Aquí se plantea un interesante dilema: la lista de casos no contiene ninguna información sobre las unidades administrativas de los casos. Aunque lo ideal es recoger dicha información durante la fase inicial de recogida de datos, también podemos asignar unidades administrativas a los casos individuales basándonos en sus relaciones espaciales (es decir, el punto se cruza con un polígono).

A continuación, intersecaremos espacialmente las ubicaciones de nuestros casos (puntos) con los límites del ADM3 (polígonos):

1. Comience con linelist (puntos)
2. Unión espacial a los límites, estableciendo el tipo de unión en "st\_intersects"
3. Utiliza select() para mantener sólo algunas de las nuevas columnas de los límites administrativos

Todas las columnas de sle\_adms se han añadido a linelist. Cada caso tiene ahora columnas que detallan los niveles administrativos a los que pertenece. En este ejemplo, sólo queremos mantener dos de las nuevas columnas (nivel administrativo 3), así que seleccionamos() los nombres de las columnas antiguas y sólo las dos adicionales de interés:

A continuación, sólo a efectos de visualización, puede ver los diez primeros casos y que sus jurisdicciones de nivel de administración 3 (ADM3) que se han adjuntado, basado en donde el punto espacialmente se cruzó con las formas de polígono.

Ahora podemos describir nuestros casos por unidad administrativa, algo que no podíamos hacer antes de la unión espacial.

También podemos crear un gráfico de barras de los recuentos de casos por unidad administrativa.

En este ejemplo, comenzamos el ggplot() con el linelist\_adm, de modo que podemos aplicar funciones de factor como fct\_infreq() que ordena las barras por frecuencia (véase la página sobre [Factores](#factors) para obtener consejos).

### Vecino más cercano

**Encontrar el centro de salud/zona de captación más cercano**

Podría ser útil saber dónde se encuentran los centros de salud en relación con los focos de la enfermedad.

Podemos utilizar el método st\_nearest\_feature join de la función st\_join() (paquete **sf**) para visualizar el centro sanitario más cercano a los casos individuales.

1. Comenzamos con el shapefile linelist linelist\_sf
2. Unimos espacialmente con sle\_hf, que es la ubicación de los centros de salud y las clínicas (puntos)

Podemos ver a continuación (primeras 50 filas) que cada caso tiene ahora datos sobre la clínica/hospital más cercano

Podemos ver que "Den Clinic" es el centro de salud más cercano para aproximadamente el 30% de los casos.

Para visualizar los resultados, podemos utilizar **tmap** - esta vez en modo interactivo para facilitar la visualización

### Búferes

También podemos explorar cuántos casos se encuentran a menos de 2,5 km (~30 minutos) de distancia a pie del centro de salud más cercano.

Nota: Para un cálculo más preciso de la distancia, es mejor reproyectar su objeto sf al respectivo sistema de proyección cartográfica local, como UTM (Tierra proyectada sobre una superficie plana). En este ejemplo, para simplificar, nos ceñiremos al sistema de coordenadas geográficas del Sistema Geodésico Mundial (WGS84) (la Tierra representada en una superficie esférica / redonda, por lo que las unidades están en grados decimales). Utilizaremos una conversión general de: 1 grado decimal = ~111km.

Vea más información sobre proyecciones cartográficas y sistemas de coordenadas en este [artículo de esri](https://www.esri.com/arcgis-blog/products/arcgis-pro/mapping/gcs_vs_pcs/). Este [blog](http://www.geo.hunter.cuny.edu/~jochen/gtech201/lectures/lec6concepts/map coordinate systems/how to choose a projection.htm) habla de los diferentes tipos de proyecciones cartográficas y de cómo se puede elegir una proyección adecuada en función del área de interés y del contexto de su mapa/análisis.

**En primer lugar**, se crea un buffer circular con un radio de ~2,5km alrededor de cada centro de salud. Esto se hace con la función st\_buffer() de **tmap**. Como la unidad del mapa está en grados decimales lat/long, así es como se interpreta "0,02". Si su sistema de coordenadas del mapa está en metros, el número debe proporcionarse en metros.

A continuación, trazamos las zonas de amortiguación propiamente dichas, con el :

En segundo lugar, intersectamos estos buffers con los casos (puntos) utilizando *st\_join()* y el tipo de unión st\_intersects\*. Es decir, los datos de los buffers se unen a los puntos con los que se cruzan.

Ahora podemos contar los resultados: nrow(linelist\_sf\_hf\_2k[is.na(linelist\_sf\_hf\_2k$osm\_id.y),]) de 1000 casos no se cruzan con ningún buffer (falta ese valor), y por tanto viven a más de 30 minutos a pie del centro de salud más cercano.

Podemos visualizar los resultados de forma que los casos que no se cruzan con ningún búfer aparecen en rojo.

### Otras uniones espaciales

Los valores alternativos para el argumento join incluyen (de la [documentación](https://r-spatial.github.io/sf/reference/st_join.html))

* st\_contiene\_propiamente
* st\_contiene
* st\_covered\_by
* st\_covers
* st\_crosses
* st\_disjoint
* st\_equals\_exact
* st\_equals
* st\_está\_dentro\_de\_la\_distancia
* st\_nearest\_feature
* st\_overlaps
* st\_touches
* st\_within

## Mapas coropléticos

Los mapas coropléticos pueden ser útiles para visualizar los datos por áreas predefinidas, normalmente unidades administrativas o áreas de salud. En la respuesta a los brotes, esto puede ayudar a dirigir la asignación de recursos a zonas específicas con altas tasas de incidencia, por ejemplo.

Ahora que tenemos los nombres de las unidades administrativas asignados a todos los casos (véase la sección sobre uniones espaciales, más arriba), podemos empezar a mapear los recuentos de casos por zonas (mapas coropléticos).

Como también tenemos datos de población por ADM3, podemos añadir esta información a la tabla case\_adm3 creada anteriormente.

Comenzamos con el dataframe creado en el paso anterior case\_adm3, que es una tabla resumen de cada unidad administrativa y su número de casos.

1. Los datos de la población sle\_adm3\_pop se unen utilizando un left\_join() de **dplyr sobre la** base de valores comunes a través de la columna admin3pcod en el dataframe case\_adm3, y la columna adm\_pcode en el dataframe sle\_adm3\_pop. Véase la página sobre la [unión de datos](#joining-data)).
2. select() se aplica al nuevo dataframe, para mantener sólo las columnas útiles - total es la población total
3. Los casos por cada 10.000 habitantes se calculan como una nueva columna con mutate()

Unir esta tabla con el shapefile de polígonos ADM3 para la cartografía

Trazado de los resultados

También podemos mapear las tasas de incidencia

## Mapeo con ggplot2

Si ya está familiarizado con el uso de **ggplot2**, puede utilizar ese paquete para crear mapas estáticos de sus datos. La función geom\_sf() dibujará diferentes objetos en función de las características (puntos, líneas o polígonos) que haya en sus datos. Por ejemplo, puede utilizar geom\_sf() en un ggplot() utilizando datos sf con geometría de polígonos para crear un mapa coroplético.

Para ilustrar cómo funciona esto, podemos empezar con el archivo shape de polígonos ADM3 que hemos utilizado antes. Recordemos que se trata de regiones de nivel administrativo 3 en Sierra Leona:

Podemos utilizar la función left\_join() de **dplyr** para añadir los datos que queremos mapear al objeto shapefile. En este caso, vamos a utilizar el dataframe case\_adm3 que creamos anteriormente para resumir los recuentos de casos por región administrativa; sin embargo, podemos utilizar este mismo enfoque para mapear cualquier dato almacenado en un dataframe.

Para hacer un gráfico de columnas de los recuentos de casos por región, utilizando **ggplot2,** podríamos llamar a geom\_col() de la siguiente manera:

Si queremos utilizar **ggplot2** para hacer un mapa de coropletas de los recuentos de casos, podemos utilizar una sintaxis similar para llamar a la función geom\_sf():

A continuación, podemos personalizar la apariencia de nuestro mapa utilizando una gramática que sea consistente en **ggplot2**, por ejemplo:

Para los usuarios de R que se sientan cómodos trabajando con **ggplot2**, geom\_sf() ofrece una implementación simple y directa que es adecuada para las visualizaciones básicas de mapas. Para saber más, lea la [viñeta de geom\_sf()](https://ggplot2.tidyverse.org/reference/ggsf.html) o el [libro de ggplot2](https://ggplot2-book.org/maps.html).

## Mapas base

### OpenStreetMap

A continuación describimos cómo conseguir un mapa base para un mapa **ggplot2** utilizando las características de OpenStreetMap. Métodos alternativos incluyen el uso de **ggmap** que requiere el registro gratuito con Google ([detalles](https://www.earthdatascience.org/courses/earth-analytics/lidar-raster-data-r/ggmap-basemap/)).

[**OpenStreetMap**](https://en.wikipedia.org/wiki/OpenStreetMap) es un proyecto de colaboración para crear un mapa editable y gratuito del mundo. Los datos de geolocalización subyacentes (por ejemplo, ubicaciones de ciudades, carreteras, características naturales, aeropuertos, escuelas, hospitales, caminos, etc.) se consideran el resultado principal del proyecto.

Primero cargamos el paquete **OpenStreetMap**, del que obtendremos nuestro mapa base.

A continuación, creamos el objeto mapa, que definimos mediante la función openmap() del paquete **OpenStreetMap** ([documentación](https://www.rdocumentation.org/packages/OpenStreetMap/versions/0.3.4/topics/openmap)). Proporcionamos lo siguiente:

* upperLeft y lowerRight Dos pares de coordenadas que especifican los límites del mosaico del mapa base
  + En este caso hemos puesto los máximos y mínimos de las filas del listado, para que el mapa responda dinámicamente a los datos
* zoom = (si es nulo se determina automáticamente)
* tipo = qué tipo de mapa base - aquí hemos enumerado varias posibilidades y el código utiliza actualmente la primera ([1]) "osm"
* mergeTiles = elegimos TRUE para que los basetiles se fusionen en uno solo

Si trazamos este mapa base ahora mismo, usando autoplot.OpenStreetMap() del paquete **OpenStreetMap**, verás que las unidades en los ejes no son coordenadas de latitud/longitud. Se está utilizando un sistema de coordenadas diferente. Para mostrar correctamente las residencias del caso (que se almacenan en lat/long), esto debe ser cambiado.

Así, queremos convertir el mapa a latitud/longitud con la función openproj() del paquete **OpenStreetMap**. Proporcionamos el mapa base y también el Sistema de Referencia de Coordenadas (CRS) que queremos. Lo hacemos proporcionando la cadena de caracteres "proj.4" para la proyección WGS 1984, pero también se puede proporcionar el CRS de otras maneras. (vea [esta página](https://www.earthdatascience.org/courses/earth-analytics/spatial-data-r/understand-epsg-wkt-and-other-crs-definition-file-types/) para entender mejor qué es una cadena proj.4)

Ahora cuando creamos el gráfico vemos que a lo largo de los ejes están las coordenadas de latitud y longitud. El sistema de coordenadas ha sido convertido. Ahora nuestros casos se trazarán correctamente si se superponen.

Consulte los tutoriales [aquí](http://data-analytics.net/cep/Schedule_files/geospatial.html) y [aquí](https://www.rdocumentation.org/packages/OpenStreetMap/versions/0.3.4/topics/autoplot.OpenStreetMap) para obtener más información.

## Mapas térmicos de densidad contorneada

A continuación describimos cómo conseguir un mapa de calor de densidad contorneada de casos, sobre un mapa base, comenzando con un listado (una fila por caso).

1. Crear un mosaico de mapa base a partir de OpenStreetMap, como se ha descrito anteriormente
2. Trazar los casos del listado utilizando las columnas de latitud y longitud
3. Convierta los puntos en un mapa de calor de densidad con stat\_density\_2d() de **ggplot2**,

Cuando tenemos un mapa base con coordenadas de latitud y longitud, podemos trazar nuestros casos encima utilizando las coordenadas de latitud y longitud de su residencia.

Partiendo de la función autoplot.OpenStreetMap() para crear el mapa base, las funciones de **ggplot2 se añaden** fácilmente encima, como se muestra con geom\_point() a continuación:

El mapa anterior puede ser difícil de interpretar, especialmente con los puntos superpuestos. Por lo tanto, puede trazar un mapa de densidad en 2d utilizando la función **ggplot2** stat\_density\_2d(). Se siguen utilizando las coordenadas lat/lon del listado, pero se realiza una estimación de la densidad del núcleo en 2D y los resultados se muestran con líneas de contorno - como un mapa topográfico. Lea la [documentación](https://ggplot2.tidyverse.org/reference/geom_density_2d.html) completa [aquí](https://ggplot2.tidyverse.org/reference/geom_density_2d.html).

### Mapa de calor de las series temporales

El mapa de calor de la densidad anterior muestra los casos acumulados. Podemos examinar el brote a lo largo del tiempo y del espacio haciendo un facetado del mapa de calor basado en el mes de inicio de los síntomas, como se deriva del listado.

Comenzamos en linelist, creando una nueva columna con el Año y el Mes de inicio. La función format() de la **base** R cambia la forma en que se muestra una fecha. En este caso queremos "AAAA-MM".

Ahora, simplemente introducimos el facetado a través de **ggplot2** en el mapa de calor de la densidad. Se aplica facet\_wrap(), utilizando la nueva columna como filas. Fijamos el número de columnas de facetas en 3 para mayor claridad.

## Estadísticas espaciales

La mayor parte de nuestra discusión hasta ahora se ha centrado en la visualización de datos espaciales. En algunos casos, también puede interesarle utilizar la estadística espacial para cuantificar las relaciones espaciales de los atributos de sus datos. En esta sección se ofrece una breve visión general de algunos conceptos clave de la estadística espacial y se sugieren algunos recursos que resultarán útiles si desea realizar análisis espaciales más exhaustivos.

### Relaciones espaciales

Antes de poder calcular cualquier estadística espacial, tenemos que especificar las relaciones entre las características de nuestros datos. Hay muchas formas de conceptualizar las relaciones espaciales, pero un modelo sencillo y comúnmente aplicable es el de la adyacencia, es decir, que esperamos una relación geográfica entre las zonas que comparten una frontera o son "vecinas" unas de otras.

Podemos cuantificar las relaciones de adyacencia entre los polígonos de las regiones administrativas en los datos sle\_adm3 que hemos estado utilizando con el paquete **spdep**. Especificaremos la contigüidad de reinas, lo que significa que las regiones serán vecinas si comparten al menos un punto a lo largo de sus fronteras. La alternativa sería la contigüidad de la torre, que requiere que las regiones compartan un borde - en nuestro caso, con polígonos irregulares, la distinción es trivial, pero en algunos casos la elección entre reina y torre puede ser influyente.

La matriz impresa arriba muestra las relaciones entre las 9 regiones de nuestros datos sle\_adm3. Una puntuación de 0 indica que dos regiones no son vecinas, mientras que cualquier valor distinto de 0 indica una relación de vecindad. Los valores de la matriz se han escalado para que cada región tenga un peso total de 1 en la fila.

Una mejor manera de visualizar estas relaciones de vecindad es trazarlas:

Hemos utilizado un enfoque de adyacencia para identificar los polígonos vecinos; los vecinos que identificamos también se denominan a veces **vecinos por contigüidad**. Pero ésta es sólo una forma de elegir qué regiones se espera que tengan una relación geográfica. Los enfoques alternativos más comunes para identificar las relaciones geográficas generan vecinos basados en la **distancia**; brevemente, estos son:

* **K-vecinos más cercanos** - Basándose en la distancia entre los centroides (el centro ponderado geográficamente de cada región poligonal), selecciona las n regiones más cercanas como vecinas. También se puede especificar un umbral de proximidad de distancia máxima. En **spdep, se** puede utilizar knearneigh() (ver [documentación](https://r-spatial.github.io/spdep/reference/knearneigh.html)).
* **Vecinos de umbral de distancia** - Selecciona todos los vecinos dentro de un umbral de distancia. En **spdep**, estas relaciones de vecindad pueden ser identificadas usando dnearneigh() (ver [documentación](https://www.rdocumentation.org/packages/spdep/versions/1.1-7/topics/dnearneigh)).

### Autocorrelación espacial

La tan citada primera ley de la geografía de Tobler afirma que "todo está relacionado con todo lo demás, pero las cosas cercanas están más relacionadas que las lejanas". En epidemiología, esto suele significar que el riesgo de un determinado resultado sanitario en una región determinada es más similar al de sus regiones vecinas que al de las lejanas. Este concepto se ha formalizado como **autocorrelación espacial**: la propiedad estadística de que las características geográficas con valores similares se agrupan en el espacio. Las medidas estadísticas de autocorrelación espacial pueden utilizarse para cuantificar el alcance de la agrupación espacial en sus datos, localizar dónde se produce la agrupación e identificar patrones compartidos de autocorrelación espacial entre distintas variables en sus datos. Esta sección ofrece una visión general de algunas medidas comunes de autocorrelación espacial y cómo calcularlas en R.

I de **Moran** - Se trata de una estadística de resumen global de la correlación entre el valor de una variable en una región y los valores de la misma variable en las regiones vecinas. La estadística I de Moran suele oscilar entre -1 y 1. Un valor de 0 indica que no hay ningún patrón de correlación espacial, mientras que los valores más cercanos a 1 o -1 indican una mayor autocorrelación espacial (valores similares cercanos) o dispersión espacial (valores disímiles cercanos), respectivamente.

Como ejemplo, calcularemos un estadístico I de Moran para cuantificar la autocorrelación espacial en los casos de Ébola que hemos mapeado antes (recordemos que se trata de un subconjunto de casos del dataframe del listado de la epidemia simulada). El paquete **spdep** tiene una función, moran.test, que puede hacer este cálculo por nosotros:

La salida de la función moran.test() nos muestra un estadístico I de Moran de round(moran\_i$estimate[1],2). Esto indica la presencia de autocorrelación espacial en nuestros datos, en concreto, que es probable que las regiones con un número similar de casos de ébola estén próximas entre sí. El valor p proporcionado por moran.test() se genera mediante la comparación con la expectativa bajo la hipótesis nula de ausencia de autocorrelación espacial, y puede utilizarse si se necesita informar de los resultados de una prueba de hipótesis formal.

I de **Moran local** - Podemos descomponer el estadístico I de Moran (global) calculado anteriormente para identificar la autocorrelación espacial localizada; es decir, para identificar grupos específicos en nuestros datos. Este estadístico, que a veces se denomina **indicador local de asociación espacial (LISA)**, resume el grado de autocorrelación espacial alrededor de cada región individual. Puede ser útil para encontrar puntos "calientes" y "fríos" en el mapa.

Para mostrar un ejemplo, podemos calcular y mapear la I de Moran local para los recuentos de casos de Ébola utilizados anteriormente, con la función local\_moran() de **spdep**:

**Getis-Ord Gi\*** - Esta es otra estadística que se utiliza comúnmente para el análisis de puntos calientes; en gran parte, la popularidad de esta estadística se relaciona con su uso en la herramienta de análisis de puntos calientes en ArcGIS. Se basa en la suposición de que, normalmente, la diferencia del valor de una variable entre regiones vecinas debería seguir una distribución normal. Utiliza un enfoque de puntuación z para identificar las regiones que tienen valores significativamente más altos (punto caliente) o significativamente más bajos (punto frío) de una variable específica, en comparación con sus vecinos.

Podemos calcular y asignar la estadística Gi\* utilizando la función localG() de **spdep**:

Como puede ver, el mapa de Getis-Ord Gi\* tiene un aspecto ligeramente diferente del mapa de Moran local que elaboré anteriormente. Esto refleja que el método utilizado para calcular estas dos estadísticas es ligeramente diferente; cuál de ellas debe utilizar depende de su caso de uso específico y de la pregunta de investigación de interés.

**Prueba L de Lee** - Es una prueba estadística de correlación espacial bivariada. Permite comprobar si el patrón espacial de una determinada variable x es similar al patrón espacial de otra variable, y, que se supone que está relacionada espacialmente con x.

Para dar un ejemplo, vamos a probar si el patrón espacial de los casos de ébola de la epidemia simulada está correlacionado con el patrón espacial de la población. Para empezar, necesitamos tener una variable de población en nuestros datos sle\_adm3. Podemos utilizar la variable total del dataframe sle\_adm3\_pop que hemos cargado anteriormente.

Podemos visualizar rápidamente los patrones espaciales de las dos variables una al lado de la otra, para ver si se parecen:

Visualmente, los patrones parecen disímiles. Podemos utilizar la función lee.test() de **spdep** para comprobar estadísticamente si el patrón de autocorrelación espacial de las dos variables está relacionado. El estadístico L será cercano a 0 si no hay correlación entre los patrones, cercano a 1 si hay una fuerte correlación positiva (es decir, los patrones son similares), y cercano a -1 si hay una fuerte correlación negativa (es decir, los patrones son inversos).

La salida anterior muestra que el estadístico L de Lee para nuestras dos variables fue round(lee\_test$estimate[1],2), lo que indica una débil correlación negativa. Esto confirma nuestra evaluación visual de que el patrón de los casos y la población no están relacionados entre sí, y proporciona pruebas de que el patrón espacial de los casos no es estrictamente un resultado de la densidad de población en las zonas de alto riesgo.

El estadístico L de Lee puede ser útil para hacer este tipo de inferencias sobre la relación entre variables distribuidas espacialmente; sin embargo, para describir la naturaleza de la relación entre dos variables con más detalle, o ajustar por confusión, se necesitarán técnicas de regresión espacial. Éstas se describen brevemente en la siguiente sección.

### Regresión espacial

Es posible que desee hacer inferencias estadísticas sobre las relaciones entre las variables de sus datos espaciales. En estos casos, es útil considerar las técnicas de regresión espacial, es decir, los enfoques de regresión que consideran explícitamente la organización espacial de las unidades en sus datos. Algunas de las razones por las que puede necesitar considerar modelos de regresión espacial, en lugar de modelos de regresión estándar como los GLM, incluyen:

* Los modelos de regresión estándar suponen que los residuos son independientes entre sí. En presencia de una fuerte autocorrelación espacial, es probable que los residuos de un modelo de regresión estándar también estén autocorrelacionados espacialmente, violando así este supuesto. Esto puede dar lugar a problemas de interpretación de los resultados del modelo, en cuyo caso sería preferible un modelo espacial.
* Los modelos de regresión también suelen suponer que el efecto de una variable x es constante en todas las observaciones. En el caso de la heterogeneidad espacial, los efectos que deseamos estimar pueden variar a lo largo del espacio, y podemos estar interesados en cuantificar esas diferencias. En este caso, los modelos de regresión espacial ofrecen más flexibilidad para estimar e interpretar los efectos.

Los detalles de los enfoques de regresión espacial están fuera del alcance de este manual. En su lugar, esta sección ofrece una visión general de los modelos de regresión espacial más comunes y sus usos, y remite a referencias que pueden ser útiles si se desea profundizar en este ámbito.

**Modelos de error espacial** - Estos modelos suponen que los términos de error entre unidades espaciales están correlacionados, en cuyo caso los datos violarían los supuestos de un modelo OLS estándar. Los modelos de error espacial también se denominan a veces **modelos autorregresivos simultáneos (SAR)**. Pueden ajustarse utilizando la función errorsarlm() del paquete **spatialreg** (funciones de regresión espacial que solían formar parte de **spdep**).

**Modelos de rezago espacial** - Estos modelos suponen que la variable dependiente de una región i está influida no sólo por el valor de las variables independientes en i, sino también por los valores de esas variables en las regiones vecinas a i. Al igual que los modelos de error espacial, los modelos de rezago espacial también se describen a veces como **modelos autorregresivos simultáneos (SAR)**. Pueden ajustarse utilizando la función lagsarlm() del paquete **spatialreg**.

El paquete **spdep** contiene varias pruebas de diagnóstico útiles para decidir entre los modelos OLS estándar, de retardo espacial y de error espacial. Estas pruebas, denominadas diagnósticos del multiplicador de Lagrange, pueden utilizarse para identificar el tipo de dependencia espacial en sus datos y elegir el modelo más apropiado. La función lm.LMtests() puede utilizarse para calcular todas las pruebas del multiplicador de Lagrange. Anselin (1988) también proporciona una útil herramienta de diagrama de flujo para decidir qué modelo de regresión espacial utilizar basándose en los resultados de las pruebas del multiplicador de Lagrange:

**Modelos jerárquicos bayesianos**: los enfoques bayesianos se utilizan habitualmente para algunas aplicaciones del análisis espacial, sobre todo para la [cartografía de enfermedades](https://pubmed.ncbi.nlm.nih.gov/15690999/). Se prefieren en los casos en los que los datos de los casos están escasamente distribuidos (por ejemplo, en el caso de un resultado raro) o son estadísticamente "ruidosos", ya que pueden utilizarse para generar estimaciones "suavizadas" del riesgo de enfermedad al tener en cuenta el proceso espacial latente subyacente. Esto puede mejorar la calidad de las estimaciones. También permiten que el investigador especifique previamente (mediante la elección de la prioridad) los complejos patrones de correlación espacial que pueden existir en los datos, que pueden dar cuenta de la variación espacialmente dependiente e independiente en las variables independientes y dependientes. En R, los modelos jerárquicos bayesianos pueden ajustarse utilizando el paquete **CARbayes** (véase [la viñeta](https://cran.r-project.org/web/packages/CARBayes/vignettes/CARBayes.pdf)) o R-INLA (véase [el sitio web](https://www.r-inla.org/home) y el [libro de texto](https://becarioprecario.bitbucket.io/inla-gitbook/)). R también puede utilizarse para llamar a software externo que realiza estimaciones bayesianas, como JAGS o WinBUGS.

## Recursos

* Funciones simples de R y [viñeta del](https://cran.r-project.org/web/packages/sf/vignettes/sf1.html) paquete sf
* [Viñeta del](https://cran.r-project.org/web/packages/tmap/vignettes/tmap-getstarted.html) paquete R tmap
* ggmap: [Visualización espacial con ggplot2](https://journal.r-project.org/archive/2013-1/kahle-wickham.pdf)
* [Introducción a la elaboración de mapas con R, visión general de los diferentes paquetes](https://bookdown.org/nicohahn/making_maps_with_r5/docs/introduction.html)
* Datos espaciales en R [(curso EarthLab)](https://www.earthdatascience.org/courses/earth-analytics/spatial-data-r/)
* [Libro de texto](https://link.springer.com/book/10.1007/978-1-4614-7618-4) Applied Spatial Data Analysis in R
* **SpatialEpiApp** - una [aplicación Shiny que se puede descargar como un paquete de R](https://github.com/Paula-Moraga/SpatialEpiApp), lo que le permite proporcionar sus propios datos y llevar a cabo la cartografía, el análisis de conglomerados y las estadísticas espaciales.
* [Taller de](http://www.econ.uiuc.edu/~lab/workshop/Spatial_in_R.html) introducción a la econometría espacial en R

# Visualización de datos

# **#Tablas** para presentaciones

{#tables-for-presentation}

Esta página muestra cómo convertir dataframes de resumen en tablas listas para su presentación con el paquete **flextable**. Estas tablas pueden insertarse en diapositivas de PowerPoint, páginas HTML, documentos PDF o Word, etc.

Comprenda que antes de utilizar **la tabla flexible**, debe crear la tabla resumen como un dataframe. Utiliza los métodos de las páginas [Tablas descriptivas](#descriptive-tables) y [Pivoteo de datos](#pivoting-data), como tabulaciones, tabulaciones cruzadas, pivoteo y cálculo de estadísticas descriptivas. El dataframe resultante puede pasarse a **flextable** para el formato de visualización.

Hay muchos otros paquetes de R que se pueden utilizar para elaborar tablas para su presentación - hemos elegido destacar **flextable** en esta página. Un ejemplo que utiliza el paquete **knitr** y su función kable() se puede encontrar en la página [Seguimiento de contactos](#contact-tracing-1). Asimismo, el paquete **DT** se destaca en la página [Dashboards with Shiny](#dashboards-with-shiny). Otros como **GT** y **huxtable** se mencionan en la página de [Paquetes recomendados](#suggested-packages-1).

## Preparación

### Cargar paquetes

Instalar y cargar **flextable**. En este manual destacamos p\_load() de **pacman**, que instala el paquete si es necesario y lo carga para su uso. También puede cargar paquetes con library() desde R **base.** Consulte la página sobre [los fundamentos de R](#r-basics) para obtener más información sobre los paquetes de R.

### Importar datos

Para empezar, importamos la lista de casos limpia de una epidemia de ébola simulada. Si quieres seguir el proceso, clica aquí pa[ra descargar el listado "limpio"](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/case_linelists/linelist_cleaned.rds) (como archivo .rds). Importe los datos con la función import() del paquete **rio** (maneja muchos tipos de archivos como .xlsx, .csv, .rds - vea la página de [importación y exportación](#import-and-export) para más detalles).

A continuación se muestran las primeras 50 filas del listado.

### Preparar la **tabla**

Antes de empezar a utilizar **la tabla flexible** tendrá que crear su tabla como un dataframe. Consulte la página sobre [Tablas descriptivas](#descriptive-tables) y [Pivotar datos](#pivoting-data) para aprender a crear un dataframe utilizando paquetes como **janitor** y **dplyr**. Debe organizar el contenido en filas y columnas tal y como quiere que se muestre. Luego, el dataframe se pasará a **flextable** para mostrarlo con colores, encabezados, fuentes, etc.

A continuación se muestra un ejemplo de la página de [tablas descriptivas](#descriptive-tables) para convertir la lista de casos en un dataframe que resume los resultados de los pacientes y los valores de TC por hospital, con una fila de totales en la parte inferior. El resultado se guarda como tabla.

## **Tabla** flexible básica

### Crear una tabla flexible

Para crear y gestionar los objetos de **la tabla flexible**, primero pasamos el dataframe por la función flextable(). Guardamos el resultado como mi\_tabla.

Después de hacer esto, podemos canalizar progresivamente el objeto mi\_tabla a través de más funciones de formato **de la tabla flexible**.

En esta página, para mayor claridad, guardaremos la tabla en los pasos intermedios como mi\_tabla, añadiendo las funciones **de la tabla flexible** poco a poco. Si quieres ver todo el código de principio a fin escrito en un solo trozo, visita la sección [Todo el código junto](#tbl_pres_all) más abajo.

La sintaxis general de cada línea de código **de la tabla flexible** es la siguiente:

* function(table, i = X, j = X, part = "X"), where:
  + La "función" puede ser una de muchas funciones diferentes, como width() para determinar el ancho de las columnas, bg() para establecer los colores de fondo, align() para establecer si el texto está alineado al centro/derecha/izquierda, etc.
  + table = es el nombre del dataframe, aunque no es necesario indicarlo si el dataframe se introduce en la función.
  + part = se refiere a la parte de la tabla a la que se aplica la función. Por ejemplo, "cabecera", "cuerpo" o "todo".
  + i = especifica la fila a la que se aplicará la función, donde 'X' es el número de fila. Si se trata de varias filas, por ejemplo de la primera a la tercera, se puede especificar: i = c(1:3). Ten en cuenta que si se selecciona "cuerpo", la primera fila empieza por debajo de la sección de cabecera.
  + j = especifica la columna a la que se aplicará la función, donde 'x' es el número o nombre de la columna. Si hay varias columnas, por ejemplo la quinta y la sexta, se puede especificar: j = c(5,6).

Puedes encontrar la lista completa de la función de formato de **la tabla flexible** [aquí](https://davidgohel.github.io/flextable/reference/index.html) o revisar la documentación introduciendo ?flextable.

### Ancho de columna

Podemos utilizar la función autofit(), que estira la tabla de forma que cada celda sólo tiene una fila de texto. La función qflextable() es una abreviatura conveniente para flextable() y autofit().

Sin embargo, esto podría no ser siempre apropiado, especialmente si hay valores muy largos dentro de las celdas, lo que significa que la tabla podría no caber en la página.

En cambio, podemos especificar el ancho con la función width(). Puede ser necesario jugar un poco para saber qué valor de anchura poner. En el ejemplo siguiente, especificamos diferentes anchos para la columna 1, la columna 2 y las columnas 4 a 8.

### Encabezados de columna

Queremos encabezados más claros para facilitar la interpretación del contenido de la tabla.

Para esta tabla, querremos añadir una segunda capa de cabecera para que las columnas que cubren los mismos subgrupos puedan agruparse. Lo hacemos con la función add\_header\_row() con top = TRUE. Proporcionamos el nuevo nombre de cada columna a values =, dejando los valores vacíos "" para las columnas que sabemos que vamos a fusionar más tarde.

También renombramos los nombres de las cabeceras en la ahora segunda cabecera en un comando separado set\_header\_labels().

Por último, para "combinar" ciertas cabeceras de columna en la cabecera superior utilizamos merge\_at() para fusionar las cabeceras de columna en la fila de la cabecera superior.

### Fronteras y fondos

Puedes ajustar los bordes, las líneas internas, etc. con varias funciones **de la tabla flexible**. A menudo es más fácil empezar eliminando todos los bordes existentes con border\_remove().

A continuación, puede aplicar los temas de borde por defecto pasando la tabla a theme\_box(), theme\_booktabs() o theme\_alafoli().

Puedes añadir líneas verticales y horizontales con una variedad de funciones. hline() y vline() añaden líneas a una fila o columna especificada, respectivamente. Dentro de cada una, debe especificar la parte = como "todo", "cuerpo" o "cabecera". Para las líneas verticales, especifique la columna a j =, y para las líneas horizontales la fila a i =. Otras funciones como vline\_right(), vline\_left(), hline\_top(), y hline\_bottom() añaden líneas sólo a los lados.

En todas estas funciones, el propio estilo de línea debe especificarse a border = y debe ser la salida de un comando separado utilizando la función fp\_border() del paquete **oficial**. Esta función le ayuda a definir el ancho y el color de la línea. Puedes definirlo sobre los comandos de la tabla, como se muestra a continuación.

### Fuente y alineación

Alineamos en el centro todas las columnas, excepto la más a la izquierda, con los nombres de los hospitales, utilizando la función align() de **flextable**.

Además, podemos aumentar el tamaño de la fuente de la cabecera y cambiarla a negrita. También podemos cambiar la fila total a negrita.

Podemos asegurar que las columnas de proporción muestren sólo un decimal utilizando la función colformat\_num(). Ten en cuenta que esto también podría haberse hecho en la fase de gestión de datos con la función round().

### Fusionar celdas

Al igual que fusionamos celdas horizontalmente en la fila de la cabecera, también podemos fusionar celdas verticalmente utilizando merge\_at() y especificando las filas (i) y la columna (j). Aquí fusionamos los valores "Hospital" y "Total de casos con resultado conocido" verticalmente para darles más espacio.

### Color de fondo

Para distinguir el contenido de la tabla de las cabeceras, es posible que queramos añadir un formato adicional, por ejemplo, cambiando el color de fondo. En este ejemplo cambiamos el cuerpo de la tabla a gris.

## Formato condicional

Podemos resaltar todos los valores de una columna que cumplan una determinada regla, por ejemplo, que más del 55% de los casos hayan muerto. Basta con poner el criterio en el argumento i = o j =, precedido de una tilde ~. Haga referencia a la columna en el dataframe, no a los valores del encabezamiento de la pantalla.

O bien, podemos resaltar toda la fila que cumpla un determinado criterio, como un hospital de interés. Para ello, basta con eliminar la especificación de la columna (j) para que los criterios se apliquen a todas las columnas.

## Todo el código junto

A continuación mostramos todo el código de las secciones anteriores juntas.

## Guardar la mesa

Hay diferentes maneras de integrar la tabla en su salida.

### Guardar tabla individual

Puedes exportar las tablas a Word, PowerPoint o HTML o como archivos de imagen (PNG). Para ello, utiliza una de las siguientes funciones:

* save\_as\_docx()
* save\_as\_pptx()
* save\_as\_image()
* save\_as\_html()

Por ejemplo, a continuación guardamos nuestra tabla como un documento de Word. Ten en cuenta la sintaxis del primer argumento - puede proporcionar simplemente el nombre de su objeto flextable, por ejemplo, mi\_tabla, o puede darle un "nombre" como se muestra a continuación (el nombre es "mi tabla"). Si el nombre, este aparecerá como el título de la tabla en Word. También demostramos el código para guardar como imagen PNG.

Ten en cuenta que los paquetes webshot o webshot2 son necesarios para guardar una tabla flexible como imagen. Las imágenes pueden salir con fondos transparentes.

Si desea ver una versión "en vivo" de la salida de la tabla **flexible** en el formato de documento previsto, utiliza print() y especifique uno de los siguientes para previsualizar =. El documento se "abrirá" en su ordenador en el programa de software especificado, pero no se guardará. Esto puede ser útil para comprobar si la tabla cabe en una página/diapositiva o para poder copiarla rápidamente en otro documento, puede utilizar el método de impresión con el argumento vista previa establecido en "pptx" o "docx".

### Imprimir tabla en R markdown

Esta tabla puede integrarse en un documento automatizado, una salida de R markdown, si el objeto tabla se llama dentro del chunk de R markdown. Esto significa que la tabla puede actualizarse como parte de un informe en el que los datos podrían cambiar, por lo que los números pueden actualizarse.

Vea los detalles en la página de [Informes con R Markdown](#reports-with-r-markdown) de este manual.

## Recursos

[El](https://ardata-fr.github.io/flextable-book/) libro completo de la **mesa flexible está** aquí: <https://ardata-fr.github.io/flextable-book/>El sitio Github está [aquíUn](https://davidgohel.github.io/flextable/)   
manual de todas las funciones de **la mesa flexible** puede encontrarse [aquí](https://davidgohel.github.io/flextable/reference/index.html)

Puedes acceder a una galería de bonitos ejemplos de tablas **flexibles** con código [aquí](https://ardata-fr.github.io/flextable-gallery/gallery/)

# #Conceptos básicos de ggplot

**{#ggplot-basics}**

**ggplot2** es el paquete de R más popular para la visualización de datos. Su función ggplot() es el núcleo de este paquete, y todo este enfoque se conoce coloquialmente como "ggplot", con las figuras resultantes a veces llamadas cariñosamente "ggplots". El "gg" en estos nombres refleja la "gramática de los gráficos" utilizada para construir las figuras. **ggplot2** se beneficia de una amplia variedad de paquetes R complementarios que mejoran aún más su funcionalidad.

La sintaxis es significativamente diferente de **la de** R, y tiene una curva de aprendizaje asociada. El uso de **ggplot2** generalmente requiere que el usuario formatee sus datos de una manera que sea altamente compatible con **tidyverse**, lo que en última instancia hace que el uso de estos paquetes juntos sea muy eficaz.

En esta página cubriremos los fundamentos del trazado con **ggplot2**. Vea la página de [consejos de ggplot](#ggplot-tips) para sugerencias y técnicas avanzadas para hacer que sus gráficos se vean realmente bien.

Hay varios tutoriales extensos de **ggplot2** enlazados en la sección de recursos. También puede descargar esta [hoja de trucos de visualización de datos con ggplot](https://github.com/rstudio/cheatsheets/raw/master/data-visualization-2.1.pdf) desde el sitio web de RStudio. Si quieres inspirarte en formas de visualizar tus datos de forma creativa, te sugerimos que revises sitios web como la [galería de gráficos de R](https://www.r-graph-gallery.com/) y [Data-to-viz](https://www.data-to-viz.com/caveats.html).

## Preparación

### Cargar paquetes

Este trozo de código muestra la carga de los paquetes necesarios para los análisis. En este manual destacamos p\_load() de **pacman**, que instala el paquete si es necesario y lo carga para su uso. También puede cargar los paquetes instalados con library() de R **base.** Consulte la página sobre [los fundamentos de R](#r-basics) para obtener más información sobre los paquetes de R.

### Importar datos

Importamos el conjunto de datos de casos de una epidemia de ébola simulada. Si quiere seguir el proceso, [clica para descargar linelist "limpia"](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/case_linelists/linelist_cleaned.rds) (como archivo .rds). Importe sus datos con la función import() del paquete **rio** (acepta muchos tipos de archivos como .xlsx, .rds, .csv - vea la página de [importación y exportación](#import-and-export) para más detalles).

A continuación se muestran las primeras 50 filas del listado. Nos centraremos en las variables continuas edad, wt\_kg (peso en kilos), ct\_blood (valores de CT) y days\_onset\_hosp (diferencia entre la fecha de inicio y la hospitalización).

### Limpieza general

Cuando se preparan los datos para trazarlos, lo mejor es hacer que los datos se adhieran a [los estándares de datos "ordenados" tanto como](https://r4ds.had.co.nz/tidy-data.html) sea posible. En las páginas de gestión de datos de este manual, como [Limpieza de datos y funciones básicas, se explica](#cleaning-data-and-core-functions) cómo conseguirlo.

Algunas formas sencillas de preparar nuestros datos para que sean mejores para el trazado pueden incluir la mejora del contenido de los datos para su visualización, lo que no equivale necesariamente a una mejor manipulación de los datos. Por ejemplo:

* Sustituir los valores NA de una columna de caracteres por la cadena de caracteres "Desconocido"
* Considere la posibilidad de convertir la columna en de tipo factor para que sus valores tengan niveles ordinales prescritos
* Limpiar algunas columnas para que sus valores "amigables con los datos" con barra baja, etc., se cambien a texto normal o a mayúsculas y minúsculas (ver [Caracteres y cadenas](#characters-and-strings))

He aquí algunos ejemplos de esto en acción:

### Pivotar más tiempo

Como una cuestión de estructura de datos, para **ggplot2** a menudo también queremos pivotar nuestros datos en formatos más largos. Lea más sobre esto en la página de [Pivoteo de datos](#pivoting-data).

Por ejemplo, digamos que queremos trazar datos que están en un formato "amplio", como por ejemplo para cada caso en linelist y sus síntomas. A continuación creamos una minilista llamada datos\_síntomas que contiene sólo las columnas case\_id y symptoms.

Así es como se ven las primeras 50 filas de esta mini-lista - vea cómo están formateadas "a lo ancho" con cada síntoma como una columna:

Si quisiéramos trazar el número de casos con síntomas específicos, estamos limitados por el hecho de que cada síntoma es una columna específica. Sin embargo, podemos hacer pivotar las columnas de síntomas a un formato más largo como este:

Aquí están las primeras 50 filas. Observe que el caso tiene 5 filas - una para cada síntoma posible. Las nuevas columnas symptom\_name y symptom\_is\_present son el resultado del pivote. Ten en cuenta que este formato puede no ser muy útil para otras operaciones, pero es útil para trazar.

## Fundamentos de ggplot

**"Gramática de los gráficos" - ggplot2**

El trazado con **ggplot2** se basa en "añadir" capas de trazado y elementos de diseño unos sobre otros, añadiendo cada comando a los anteriores con un símbolo de suma (+). El resultado es un objeto de trazado multicapa que se puede guardar, modificar, imprimir, exportar, etc.

Los objetos ggplot pueden ser muy complejos, pero el orden básico de las capas suele ser el siguiente:

1. Comience con el comando ggplot() de la línea de base - esto "abre" el ggplot y permite que las funciones subsecuentes sean agregadas con +. Normalmente, el conjunto de datos también se especifica en este comando
2. Añadir capas "geom" - estas funciones visualizan los datos como geometrías (formas), por ejemplo, como un gráfico de barras, un gráfico de líneas, un gráfico de dispersión, un histograma (¡o una combinación!). Todas estas funciones comienzan con geom\_ como prefijo.
3. Añada elementos de diseño al gráfico, como etiquetas de ejes, título, fuentes, tamaños, esquemas de color, leyendas o rotación de ejes.

Un ejemplo sencillo de código esqueleto es el siguiente. Explicaremos cada componente en las secciones siguientes.

## ggplot()

El comando de apertura de cualquier gráfico ggplot2 es ggplot(). Este comando simplemente crea un lienzo en blanco sobre el que añadir capas. Se "abre" el camino para añadir más capas con un símbolo +.

Normalmente, el comando ggplot() incluye el argumento data = para el gráfico. Esto establece el conjunto de datos por defecto que se utilizará para las capas posteriores del gráfico.

Este comando terminará con un + después de su paréntesis de cierre. Esto deja el comando "abierto". El ggplot sólo se ejecutará/aparecerá cuando el comando completo incluya una capa final sin un + al final.

## Geoms

Un lienzo en blanco no es suficiente: necesitamos crear geometrías (formas) a partir de nuestros datos (por ejemplo, gráficos de barras, histogramas, gráficos de dispersión, gráficos de caja).

Esto se hace añadiendo capas "geoms" al comando inicial ggplot(). Hay muchas funciones de **ggplot2** que crean "geoms". Cada una de estas funciones comienza con "geom\_", por lo que nos referiremos a ellas genéricamente como geom\_XXXX(). Hay más de 40 geoms en **ggplot2** y muchos otros creados por aficionados. Véalos en la [galería de ggplot2](https://exts.ggplot2.tidyverse.org/gallery/). Algunos geoms comunes se enumeran a continuación:

* Histogramas - geom\_histograma()
* Gráficos de barras - geom\_bar() o geom\_col() (véase [la sección "Gráfico de barras"](#ggplot_basics_bars))
* Gráficos de caja - geom\_boxplot()
* Puntos (por ejemplo, gráficos de dispersión) - geom\_point()
* Gráficos de líneas - geom\_line() o geom\_path()
* Líneas de tendencia - geom\_smooth()

En un gráfico se pueden mostrar uno o varios geoms. Cada uno se añade a los comandos anteriores **de ggplot2** con un +, y se trazan secuencialmente de manera que los geoms posteriores se trazan encima de los anteriores.

## Asignación de datos al gráfico

A la mayoría de las funciones geom hay que decirles qué utilizar para crear sus formas, por lo que hay que indicarles cómo deben asignar las columnas de los datos a los componentes del gráfico, como los ejes, los colores de las formas, los tamaños de las formas, etc. Para la mayoría de las funciones geom, los componentes esenciales que deben asignarse a las columnas de los datos son el eje x y (si es necesario) el eje y.

Este "mapeo" se produce con el argumento mapping =. Los mapeos que proporcione a mapping deben estar envueltos en la función aes(), por lo que escribiría algo como mapping = aes(x = col1, y = col2), como se muestra a continuación.

A continuación, en el comando ggplot() los datos se establecen como linelist de casos. En el argumento mapping = aes() la columna edad se asigna al eje x, y la columna wt\_kg se asigna al eje y.

Después de un +, los comandos de trazado continúan. Se crea una forma con la función "geom" geom\_point(). Este geom hereda los mapeos del comando ggplot() anterior - conoce las asignaciones eje-columna y procede a visualizar esas relaciones como puntos en el lienzo.

Como otro ejemplo, los siguientes comandos utilizan los mismos datos, un mapeo ligeramente diferente y un geom diferente. La función geom\_histograma() sólo requiere una columna mapeada en el eje x, ya que el eje y de las cuentas se genera automáticamente.

### Estética de la trama

En la terminología de ggplot, la "estética" de un gráfico tiene un significado específico. Se refiere a una propiedad visual de los datos trazados. Ten en cuenta que "estética" aquí se refiere a los datos que se trazan en geoms / formas - no la pantalla circundante, tales como títulos, etiquetas de los ejes, el color de fondo, que se podría asociar con la palabra "estética" en Inglés común. En ggplot esos detalles se llaman "temas" y se ajustan dentro de un comando theme() (ver [esta sección](#ggplot_basics_themes)).

Por lo tanto, la estética del objeto de ploteo puede ser colores, tamaños, transparencias, colocación, etc. de los datos ploteados. No todos los geoms tendrán las mismas opciones estéticas, pero muchas pueden ser utilizadas por la mayoría de los geoms. He aquí algunos ejemplos:

* shape = Mostrar un punto con geom\_point() como punto, estrella, triángulo o cuadrado...
* fill = El color interior (por ejemplo, de una barra o boxplot)
* color = La línea exterior de una barra, boxplot, etc., o el color del punto si se utiliza geom\_point()
* size = Tamaño (por ejemplo, grosor de línea, tamaño de punto)
* alpha = Transparencia (1 = opaco, 0 = invisible)
* binwidth = Ancho de los bins del histograma
* anchura = Anchura de las columnas del "diagrama de barras"
* linetype = Tipo de línea (por ejemplo, sólida, discontinua, punteada)

A esta estética de los objetos de la trama se le pueden asignar valores de dos maneras:

1. Se asigna un valor estático (por ejemplo, color = "azul") que se aplica a todas las observaciones trazadas
2. Se asigna a una columna de los datos (por ejemplo, color = hospital) de manera que la visualización de cada observación depende de su valor en esa columna

### Establecer un valor estático

Si se desea que la estética del objeto de trazado sea estática, es decir, que sea la misma para cada observación de los datos, se escribe su asignación dentro del geom pero fuera de cualquier sentencia mapping = aes(). Estas asignaciones podrían ser como size = 1 o color = "blue". Aquí hay dos ejemplos:

* En el primer ejemplo, el mapeo = aes() está en el comando ggplot() y los ejes se asignan a las columnas de edad y peso en los datos. La estética del gráfico color =, tamaño =, y alfa = (transparencia) se asignan a valores estáticos. Para mayor claridad, esto se hace en la función geom\_point(), ya que se pueden añadir otros geoms después que tomarían valores diferentes para su estética de trazado.
* En el segundo ejemplo, el histograma requiere sólo el eje x mapeado a una columna. La anchura del histograma =, el color =, el relleno = (color interno), y el alfa = se establecen de nuevo dentro del geom a valores estáticos.

### Escalado a los valores de la columna

La alternativa es escalar la estética del objeto de trazado por los valores de una columna. En este enfoque, la visualización de esta estética dependerá del valor de esa observación en esa columna de los datos. Si los valores de la columna son continuos, la escala de visualización (leyenda) para esa estética será continua. Si los valores de la columna son discretos, la leyenda mostrará cada valor y los datos trazados aparecerán como claramente "agrupados" (lea más en la sección de [agrupación](#ggplotgroups) de esta página).

Para conseguirlo, se asigna esa estética de gráfico a un nombre de columna (sin comillas). Esto debe hacerse dentro de una función *mapping = aes()* (nota: hay varios lugares en el código donde puedes hacer estas asignaciones de mapeo, como se discute [a continuación](##ggplot_basics_map_loc)).

A continuación, dos ejemplos.

* En el primer ejemplo, el color = estético (de cada punto) está mapeado a la columna edad - ¡y ha aparecido una escala en una leyenda! Por ahora sólo hay que tener en cuenta que la escala existe - mostraremos cómo modificarla en secciones posteriores.
* En el segundo ejemplo, dos nuevas estéticas de trazado también se asignan a columnas (color = y tamaño =), mientras que las estéticas de trazado forma = y alfa = se asignan a valores estáticos fuera de cualquier función de mapeo = aes().

Nota: Las asignaciones de los ejes siempre se asignan a las columnas de los datos (no a los valores estáticos), y esto se hace siempre dentro de mapping = aes().

Es importante mantener un seguimiento de las capas y la estética de los gráficos cuando se hacen gráficos más complejos, por ejemplo, gráficos con múltiples geom. En el ejemplo siguiente, el tamaño = estético se asigna dos veces - una para geom\_point() y otra para geom\_smooth() - ambas veces como un valor estático.

### Dónde hacer las asignaciones cartográficas

El mapeo estético dentro de mapping = aes() puede escribirse en varios lugares en sus comandos de trazado e incluso puede escribirse más de una vez. Esto puede ser escrito en el comando ggplot() superior, y/o para cada geom individual debajo. Los matices incluyen:

* Las asignaciones de mapeo realizadas en el comando ggplot() superior se heredarán por defecto en cualquier geom a continuación, al igual que se heredan x = e y =
* Las asignaciones cartográficas realizadas dentro de un geom se aplican sólo a ese geom

Del mismo modo, los datos = especificados en la parte superior de ggplot() se aplicarán por defecto a cualquier geom de abajo, pero también se podrían especificar datos para cada geom (pero esto es más difícil).

Así, cada uno de los siguientes comandos creará el mismo gráfico:

### Grupos

Puedes agrupar fácilmente los datos y "trazar por grupo". De hecho, ¡ya lo has hecho!

Asignar la columna "agrupación" a la estética de la gráfica adecuada, dentro de un mapeo = aes(). Arriba, hemos demostrado esto usando valores continuos cuando asignamos el tamaño del punto = a la columna edad. Sin embargo, esto funciona de la misma manera para las columnas discretas/categóricas.

Por ejemplo, si quiere que los puntos se muestren por género, deberá establecer mapeo = aes(color = género). Automáticamente aparecerá una leyenda. Esta asignación puede hacerse dentro de mapping = aes() en el comando ggplot() superior (y ser heredado por el geom), o podría establecerse en un mapping = aes() separado dentro del geom. Ambos enfoques se muestran a continuación:

Ten en cuenta que dependiendo de la geom, tendrá que utilizar diferentes argumentos para agrupar los datos. Para geom\_point() lo más probable es que utiliza color =, forma = o tamaño =. Mientras que para geom\_bar() es más probable que utiliza fill =. Esto sólo depende del geom y de la estética del gráfico que desee para reflejar las agrupaciones.

Para su información - la forma más básica de agrupar los datos es utilizando sólo el argumento group = dentro de mapping = aes(). Sin embargo, esto por sí mismo no cambiará los colores, el relleno o las formas. Tampoco creará una leyenda. Sin embargo, los datos están agrupados, por lo que las visualizaciones estadísticas pueden verse afectadas.

Para ajustar el orden de los grupos en un gráfico, consulte la página de [consejos de ggplot](#ggplot-tips) o la página sobre [Factores](#factors). Hay muchos ejemplos de gráficos agrupados en las secciones siguientes sobre el trazado de datos continuos y categóricos.

## Facetas / Múltiplos pequeños

Las facetas, o "pequeños múltiplos", se utilizan para dividir un gráfico en una figura de varios paneles, con un panel ("faceta") por grupo de datos. El mismo tipo de gráfico se crea varias veces, cada una de ellas utilizando un subgrupo del mismo conjunto de datos.

El facetado es una funcionalidad que viene con **ggplot2**, por lo que las leyendas y los ejes de los "paneles" facetados se alinean automáticamente. Hay otros paquetes discutidos en la página de [consejos de ggplot](#ggplot-tips) que se utilizan para combinar gráficos completamente diferentes (**cowplot** y **patchwork**) en una figura.

El facetado se realiza con una de las siguientes funciones de **ggplot2**:

1. facet\_wrap() Para mostrar un panel diferente para cada nivel de una sola variable. Un ejemplo de esto podría ser mostrar una curva de epidemia diferente para cada hospital de una región. Las facetas se ordenan alfabéticamente, a menos que la variable sea un factor con otro orden definido.

* Puedes invocar ciertas opciones para determinar la disposición de las facetas, por ejemplo, nrow = 1 o ncol = 1 para controlar el número de filas o columnas en las que se organizan los gráficos con facetas.

1. facet\_grid() Se utiliza cuando se quiere introducir una segunda variable en la disposición de las facetas. Aquí cada panel de una cuadrícula muestra la intersección entre los valores de dos columnas. Por ejemplo, las curvas epidémicas para cada combinación hospital-grupo de edad con los hospitales en la parte superior (columnas) y los grupos de edad en los lados (filas).

* nrow y ncol no son relevantes, ya que los subgrupos se presentan en una cuadrícula

Cada una de estas funciones acepta una sintaxis de fórmula para especificar la(s) columna(s) para el facetado. Ambas aceptan hasta dos columnas, una a cada lado de la tilde ~.

* Para facet\_wrap() lo más frecuente es escribir una sola columna precedida de una tilde ~ como facet\_wrap(~hospital). Sin embargo, puede escribir dos columnas facet\_wrap(resultado ~hospital) - cada combinación única se mostrará en un panel separado, pero no se organizarán en una cuadrícula. Los encabezados mostrarán los términos combinados y éstos no tendrán una lógica específica para las columnas frente a las filas. Si sólo proporciona una variable de facetado, se utiliza un punto . como marcador de posición en el otro lado de la fórmula - vea los ejemplos de código.
* Para facet\_grid() también puede especificar una o dos columnas a la fórmula (filas de la cuadrícula ~ columnas). Si sólo quiere especificar una, puede colocar un punto . al otro lado de la tilde como facet\_grid(. ~ hospital) o facet\_grid(hospital ~ .).

Las facetas pueden contener rápidamente una cantidad abrumadora de información, por lo que conviene asegurarse de no tener demasiados niveles de cada variable por la que se elija la faceta. He aquí algunos ejemplos rápidos con el conjunto de datos sobre la malaria (véase [Descargar el manual y los datos](#download-handbook-and-data)), que consiste en el recuento diario de casos de malaria en los centros, por grupos de edad.

A continuación importamos y hacemos algunas modificaciones rápidas para simplificar:

A continuación se muestran las primeras 50 filas de los datos sobre la malaria. Obsérvese que hay una columna malaria\_tot, pero también columnas para los recuentos por grupo de edad (que se utilizarán en el segundo ejemplo de facet\_grid()).

### facet\_wrap()

Por el momento, vamos a centrarnos en las columnas malaria\_tot y Distrito. Ignoremos por ahora las columnas de recuento por edad. Trazaremos las curvas epidémicas con geom\_col(), que produce una columna para cada día a la altura del eje y especificada en la columna malaria\_tot (los datos ya son recuentos diarios, por lo que utilizamos geom\_col() - véase [la sección "Diagrama de barras" más adelante](#ggplot_basics_bars)).

Cuando añadimos el comando facet\_wrap(), especificamos una tilde y a continuación la columna sobre la que hacer la faceta (Distrito en este caso). Puedes colocar otra columna a la izquierda de la tilde, - esto creará una faceta para cada combinación - pero le recomendamos que lo haga con facet\_grid() en su lugar. En este caso de uso, se crea una faceta para cada valor único de Distrito.

### facet\_grid()

Podemos utilizar un enfoque de facet\_grid() para cruzar dos variables. Digamos que queremos cruzar Distrito y edad. Bien, necesitamos hacer algunas transformaciones de datos en las columnas de edad para poner estos datos en el formato "largo" preferido por ggplot. Los grupos de edad tienen sus propias columnas - los queremos en una sola columna llamada age\_group y otra llamada num\_cases. Consulte la página sobre [Pivoteo de datos](#pivoting-data) para obtener más información sobre este proceso.

Ahora las primeras 50 filas de datos tienen este aspecto:

Cuando se pasan las dos variables a facet\_grid(), lo más fácil es utilizar la notación de fórmula (por ejemplo, x ~ y) donde x son filas e y son columnas. Aquí está el gráfico, utilizando facet\_grid() para mostrar los gráficos para cada combinación de las columnas age\_group y District.

### Ejes libres o fijos

Las escalas de los ejes que se muestran cuando se hacen las facetas son por defecto las mismas (fijas) en todas las facetas. Esto es útil para las comparaciones cruzadas, pero no siempre es apropiado.

Al utilizar facet\_wrap() o facet\_grid(), podemos añadir scales = "free\_y" para "liberar" o liberar los ejes y de los paneles para que se escalen adecuadamente a su subconjunto de datos. Esto es particularmente útil si los recuentos reales son pequeños para una de las subcategorías y las tendencias son difíciles de ver de otra manera. En lugar de "free\_y" también podemos escribir "free\_x" para hacer lo mismo con el eje x (por ejemplo, para las fechas) o "free" para ambos ejes. Ten en cuenta que en facet\_grid, las escalas de y serán las mismas para las facetas en la misma fila, y las escalas de x serán las mismas para las facetas en la misma columna.

Cuando se utiliza facet\_grid solamente, podemos añadir space = "free\_y" o space = "free\_x" para que la altura o la anchura real de la faceta sea ponderada a los valores de la figura en su interior. Esto sólo funciona si ya se ha aplicado la escala = "free" (y o x).

### Orden del nivel de los factores en las facetas

Consulte esta [entrada](https://juliasilge.com/blog/reorder-within/) sobre cómo reordenar los niveles de los factores dentro de las facetas.

## Almacenamiento de gráficos

### Guardar los gráficos

Por defecto, cuando se ejecuta un comando ggplot(), el gráfico se imprimirá en el panel de RStudio Plots. Sin embargo, también puede guardar el gráfico como un objeto utilizando el operador de asignación <- y dándole un nombre. Entonces no se imprimirá a menos que se ejecute el propio nombre del objeto. También puede imprimirlo envolviendo el nombre del trazado con print(), pero esto sólo es necesario en ciertas circunstancias, como si el trazado se crea dentro de un bucle for utilizado para imprimir múltiples trazados a la vez (véase la página [Iteración, bucles y listas](#iteration-loops-and-lists)).

### Modificación de gráficos guardados

Una cosa buena de **ggplot2** es que puedes definir un gráfico (como arriba), y luego añadirle capas empezando por su nombre. No tiene que repetir todos los comandos que crearon el gráfico original.

Por ejemplo, para modificar el gráfico age\_by\_wt que se definió anteriormente, para incluir una línea vertical a la edad de 50 años, sólo tendríamos que añadir un + y empezar a añadir capas adicionales al gráfico.

### Exportación de gráficos

La exportación de ggplots es fácil con la función ggsave() de **ggplot2**. Puedes funcionar de dos maneras, ya sea:

* Especifique el nombre del objeto de la gráfica, a continuación, la ruta del archivo y el nombre con la extensión
  + Por ejemplo: ggsave(my\_plot, here("plots", "my\_plot.png"))
* Ejecuta el comando con sólo una ruta de archivo, para guardar el último gráfico que se imprimió
  + Por ejemplo: ggsave(here("plots", "my\_plot.png"))

Puedes exportar como png, pdf, jpeg, tiff, bmp, svg, o varios otros tipos de archivos, especificando la extensión del archivo en la ruta del mismo.

También puede especificar los argumentos ancho =, alto = y unidades = (ya sea "in", "cm" o "mm"). También puede especificar dpi = con un número para la resolución del trazado (por ejemplo, 300). Consulte los detalles de la función introduciendo ?ggsave o leyendo la [documentación en línea](https://ggplot2.tidyverse.org/reference/ggsave.html).

Recuerde que puede utilizar la sintaxis here() para proporcionar la ruta de archivo deseada. Consulte la página de [importación y exportación](#import-and-export) para obtener más información.

## Etiquetas

Seguramente querrá añadir o ajustar las etiquetas del gráfico. Esto se hace más fácilmente dentro de la función labs() que se añade al gráfico con + al igual que los geoms.

Dentro de labs() puede proporcionar cadenas de caracteres a estos argumentos:

* x = e y = El título del eje x y del eje y (etiquetas)
* title = El título de la trama principal
* subtítulo = El subtítulo de la gráfica, en texto más pequeño debajo del título
* caption = El título del gráfico, en la parte inferior derecha por defecto

Aquí está un gráfico que hicimos antes, pero con etiquetas más bonitas:

Observe cómo en la asignación del título hemos utilizado str\_glue() del paquete **stringr** para implantar código R dinámico dentro del texto de la cadena. El título mostrará la fecha "Datos a partir de:" que refleja la fecha máxima de hospitalización en linelist. Lea más sobre esto en la página sobre [Caracteres y cadenas](#characters-and-strings).

Una nota sobre la especificación del título de la leyenda: No hay un argumento "título de la leyenda", ya que podría tener múltiples escalas en su leyenda. Dentro de labs(), puede escribir el argumento de la estética del gráfico utilizado para crear la leyenda, y proporcionar el título de esta manera. Por ejemplo, arriba asignamos color = edad para crear la leyenda. Por lo tanto, proporcionamos color = a labs() y asignamos el título de la leyenda deseado ("Edad" con A mayúscula). Si se crea la leyenda con aes(fill = COLUMN), entonces en labs() se escribiría fill = para ajustar el título de esa leyenda. La sección sobre escalas de color en la página [de consejos de ggplot](#ggplot-tips) proporciona más detalles sobre la edición de leyendas, y un enfoque alternativo utilizando las funciones scales\_().

## Temas

Una de las mejores partes de **ggplot2** es la cantidad de control que tienes sobre el gráfico - ¡puedes definir cualquier cosa! Como se mencionó anteriormente, el diseño del gráfico que no está relacionado con las formas/geometrías de los datos se ajustan dentro de la función theme(). Por ejemplo, el color de fondo del gráfico, la presencia/ausencia de líneas de cuadrícula, y la fuente/tamaño/color/alineación del texto (títulos, subtítulos, leyendas, texto de los ejes...). Estos ajustes pueden realizarse de dos maneras:

* Añade una función theme\_() [completa](https://ggplot2.tidyverse.org/reference/ggtheme.html) para realizar ajustes de barrido - incluye theme\_classic(), theme\_minimal(), theme\_dark(), theme\_light() theme\_grey(), theme\_bw() entre otras
* Ajuste cada pequeño aspecto de la trama individualmente dentro de theme()

### Temas completos

Como son bastante sencillas, demostraremos las funciones del tema completo a continuación y no las describiremos más aquí. Ten en cuenta que cualquier microajuste con theme() debe hacerse después de utilizar un tema completo.

Escríbalos con paréntesis vacíos.

### Modificar el tema

La función theme() puede tomar un gran número de argumentos, cada uno de los cuales edita un aspecto muy específico del gráfico. No hay manera de que podamos cubrir todos los argumentos, pero describiremos el patrón general para ellos y le mostraremos cómo encontrar el nombre del argumento que necesita. La sintaxis básica es esta:

1. Dentro de theme() escribe el nombre del argumento del elemento de la trama que quieres editar, como plot.title =
2. Proporcionar una función element\_() al argumento

* Lo más habitual es utilizar element\_text(), pero también element\_rect() para los colores de fondo del lienzo, o element\_blank() para eliminar los elementos de la trama

1. Dentro de la función element\_(), escriba las asignaciones de argumentos para realizar los ajustes finos que desee

Esa descripción era bastante abstracta, así que aquí hay algunos ejemplos.

El siguiente gráfico parece bastante tonto, pero sirve para mostrarle una variedad de formas en las que puede ajustar su gráfico.

* Comenzamos con el gráfico age\_by\_wt definido anteriormente y añadimos theme\_classic()
* Para realizar ajustes más finos, añadimos theme() e incluimos un argumento por cada elemento de la trama a ajustar

Puedes ser bueno organizar los argumentos en secciones lógicas. A continuación se describen algunos de los utilizados:

* legend.position = es único en el sentido de que acepta valores simples como "abajo", "arriba", "izquierda" y "derecha". Pero, por lo general, los argumentos relacionados con el texto requieren que se coloquen los detalles dentro de element\_text().
* Tamaño del título con element\_text(size = 30)
* La alineación horizontal del título con element\_text(hjust = 0) (de derecha a izquierda)
* El subtítulo está en cursiva con element\_text(face = "italic")

Aquí hay algunos argumentos de theme() especialmente comunes. Reconocerá algunos patrones, como añadir .x o .y para aplicar el cambio sólo a un eje.

| **argumento de theme()** | **Lo que ajusta** |
| --- | --- |
| plot.title = element\_text() | El título |
| plot.subtitle = element\_text() | El subtítulo |
| plot.caption = element\_text() | La leyenda (familia, cara, color, tamaño, ángulo, vjust, hjust...) |
| axis.title = element\_text() | Títulos de los ejes (tanto x como y) (tamaño, cara, ángulo, color...) |
| axis.title.x = element\_text() | Título del eje x solamente (use .y para el eje y solamente) |
| axis.text = element\_text() | Texto de los ejes (x e y) |
| axis.text.x = element\_text() | Texto del eje x solamente (use .y para el eje y solamente) |
| axis.ticks = element\_blank() | Eliminar las garrapatas del eje |
| axis.line = element\_line() | Líneas del eje (color, tamaño, tipo de línea: sólida, punteada, etc.) |
| strip.text = element\_text() | Texto de la tira de facetas (color, cara, tamaño, ángulo...) |
| strip.background = element\_rect() | tira de facetas (relleno, color, tamaño...) |

Pero ¡hay tantos argumentos temáticos! ¿Cómo podría recordarlos todos? No te preocupes, es imposible recordarlos todos. Por suerte, hay algunas herramientas que le ayudarán:

La documentación de **tidyverse** sobre la [modificación del tema](https://ggplot2.tidyverse.org/reference/theme.html), que tiene una lista completa.

**CONSEJO:** Ejecuta theme\_get() de **ggplot2** para imprimir una lista de los más de 90 argumentos de theme() en la consola.

**TIP:** Si alguna vez quieres eliminar un elemento de un gráfico, también puedes hacerlo a través de theme(). Basta con pasar element\_blank() a un argumento para que desaparezca por completo. Para las leyendas, establezca legend.position = "none".

## Colores

Consulte esta [sección sobre las escalas de color de la página de consejos de ggplot](#ggplot_tips_colors).

## Canalización en ****ggplot2****

Cuando se utilizan tuberías para limpiar y transformar los datos, es fácil pasar los datos transformados a ggplot().

Las tuberías que pasan el conjunto de datos de función a función pasarán a + una vez que se llame a la función ggplot(). Ten en cuenta que, en este caso, no es necesario especificar el argumento data =, ya que éste se define automáticamente como el conjunto de datos canalizado.

Así es como podría verse:

## Trazar datos continuos

A lo largo de esta página, ya has visto muchos ejemplos de trazado de datos continuos. Aquí los consolidamos brevemente y presentamos algunas variaciones.   
Las visualizaciones que aquí se cubren incluyen:

* Gráficos para una variable continua:
  + **Histograma**, un gráfico clásico para presentar la distribución de una variable continua.
  + **Gráfico de caja** (también llamado de caja y bigotes), para mostrar los percentiles 25, 50 y 75, los extremos de la cola de la distribución y los valores atípicos ([limitaciones importantes](https://www.data-to-viz.com/caveat/boxplot.html)).
  + **Gráfico de fluctuación**, para mostrar todos los valores como puntos que se "fluctúan" para que se puedan ver (en su mayoría) todos, incluso cuando dos tienen el mismo valor.
  + **Gráfico del violín**, muestra la distribución de una variable continua en función de la anchura simétrica del "violín".
  + Los **gráficos de Sina**, son una combinación de los gráficos de jitter y de violín, donde se muestran los puntos individuales pero con la forma simétrica de la distribución (a través del paquete **ggforce**).
* **Gráfico de dispersión** para dos variables continuas.
* **Gráficos de calor** para tres variables continuas (enlazado a la página de [gráficos de calor](#heat-plots))

### Histogramas

Los histogramas pueden parecerse a los gráficos de barras, pero son distintos porque miden la distribución de una variable continua. No hay espacios entre las "barras", y sólo se proporciona una columna a geom\_histogram().

A continuación se muestra el código para generar **histogramas**, que agrupan datos continuos en rangos y se muestran en barras adyacentes de altura variable. Esto se hace utilizando geom\_histogram(). Consulte la [sección "Diagrama de barras"](#ggplot_basics_bars) de la página de fundamentos de ggplot para entender la diferencia entre geom\_histograma(), geom\_bar() y geom\_col().

Vamos a mostrar la distribución de las edades de los casos. Dentro de mapping = aes() especifique la columna de la que quiere ver la distribución. Puedes asignar esta columna al eje x o al eje y.

Las filas serán asignadas a "bins" basados en su edad numérica, y estos bins serán representados gráficamente por barras. Si se especifica un número de bins con la estética de gráfico bins =, los puntos de ruptura se espacian uniformemente entre los valores mínimos y máximos del histograma. Si no se especifica bins =, se adivinará un número apropiado de bins y se mostrará este mensaje después del gráfico:

## `stat\_bin()` usando `bins = 30`. Elija un valor mejor con `binwidth`.

Si no quiere especificar un número de bins a bins =, puede especificar alternativamente binwidth = en las unidades del eje. Damos algunos ejemplos que muestran diferentes bins y anchos de bins:

Para obtener proporciones suavizadas, puede utilizar geom\_density():

Para obtener un histograma "apilado" (de una columna continua de datos), puede hacer una de las siguientes cosas:

1. Utiliza geom\_histogram() con el argumento fill = dentro de aes() y asignado a la columna de agrupación, o
2. Utiliza geom\_freqpoly(), que probablemente sea más fácil de leer (aún puedes establecer binwidth =)
3. Para ver las proporciones de todos los valores, establezca el y = after\_stat(density) (utiliza esta sintaxis exactamente - no se ha modificado para sus datos). Nota: estas proporciones se mostrarán por grupo.

Cada uno se muestra a continuación (\*nótese el uso de color = frente a relleno = en cada uno):

Si quieres divertirte un poco, prueba con geom\_density\_ridges del paquete **ggridges** ([vignette aquí](https://cran.r-project.org/web/packages/ggridges/vignettes/introduction.html).

Lea más en detalle sobre los histogramas en la [página de](https://ggplot2.tidyverse.org/reference/geom_histogram.html) **tidyverse** [sobre geom\_histogram()](https://ggplot2.tidyverse.org/reference/geom_histogram.html).

### Gráficos de caja

Los gráficos de caja son habituales, pero tienen importantes limitaciones. Pueden ocultar la distribución real - por ejemplo, una distribución bimodal. Consulte esta [galería de gráficos de R](https://www.r-graph-gallery.com/boxplot.html) y este [artículo sobre la conversión de datos en imágenes](https://www.data-to-viz.com/caveat/boxplot.html) para obtener más detalles. Sin embargo, muestran muy bien el rango intercuartil y los valores atípicos, por lo que pueden superponerse a otros tipos de gráficos que muestran la distribución con más detalle.

A continuación le recordamos los distintos componentes de un boxplot:

Cuando se utiliza geom\_boxplot() para crear un gráfico de caja, generalmente se asigna sólo un eje (x o y) dentro de aes(). El eje especificado determina si los gráficos son horizontales o verticales.

En la mayoría de los geoms, se crea un gráfico por grupo asignando una estética como color = o relleno = a una columna dentro de aes(). Sin embargo, en el caso de los gráficos de caja, esto se consigue asignando la columna de agrupación al eje no asignado (x o y). A continuación se muestra el código para un diagrama de caja de todos los valores de edad en el conjunto de datos, y en segundo lugar el código para mostrar un diagrama de caja para cada género (no ausente) en el conjunto de datos. Ten en cuenta que los valores NA (que faltan) aparecerán como un gráfico de caja separado a menos que se eliminen. En este ejemplo, también hemos establecido el relleno del resultado de la columna para que cada gráfico tenga un color diferente, pero esto no es necesario.

Para ver el código para añadir un gráfico de caja a los bordes de un gráfico de dispersión (gráficos "marginales"), consulte la página [ggplot tips](#ggplot-tips).

### Gráficos de violín, jitter y sina

A continuación se muestra el código para crear gráficos de **violín** (geom\_violin) y **gráficos de fluctuación** (geom\_jitter) para mostrar distribuciones. Se puede especificar que el relleno o el color también estén determinados por los datos, insertando estas opciones dentro de aes().

Puedes combinar los dos usando la función geom\_sina() del paquete **ggforce**. La sina traza los puntos de jitter en la forma del gráfico de violín. Cuando se superpone al gráfico de violín (ajustando las transparencias) puede ser más fácil de interpretar visualmente.

### Dos variables continuas

Siguiendo una sintaxis similar, geom\_point() le permitirá trazar dos variables continuas entre sí en un **gráfico de dispersión**. Esto es útil para mostrar los valores reales en lugar de sus distribuciones. En (A) se muestra un gráfico de dispersión básico de la edad frente al peso. En (B) volvemos a utilizar facet\_grid() para mostrar la relación entre dos variables continuas en linelist.

### Tres variables continuas

Puedes mostrar tres variables continuas utilizando el argumento fill = para crear un gráfico de calor. El color de cada "celda" reflejará el valor de la tercera columna continua de datos. Consulte la página de [consejos de ggplot](#ggplot-tips) y la página de [gráficos de calor](#heat-plots) para obtener más detalles y varios ejemplos.

Hay formas de hacer gráficos en 3D en R, pero para la epidemiología aplicada suelen ser difíciles de interpretar y, por tanto, menos útiles para la toma de decisiones.

## Trazar datos categóricos

Los datos categóricos pueden ser valores de carácter, pueden ser lógicos (TRUE/FALSE), o factores (ver la página de [Factores](#factors)).

### Preparación

#### Estructura de datos

Lo primero que hay que entender sobre sus datos categóricos es si existen como observaciones brutas, como una lista de casos, o como un dataframe de resumen o agregado que contiene recuentos o proporciones. El estado de sus datos afectará a la función de trazado que utilice:

* Si sus datos son observaciones en bruto con una fila por observación, es probable que utiliza geom\_bar()
* Si sus datos ya están agregados en recuentos o proporciones, es probable que utiliza geom\_col()

#### Tipo de columna y ordenación de valores

A continuación, examine el tipo de las columnas que desea trazar. Miramos a hospital, primero con class() de R **base**, y con tabyl() de **janitor**.

Podemos ver que los valores dentro son caracteres, ya que son nombres de hospitales, y por defecto están ordenados alfabéticamente. Hay valores "otros" y "desaparecidos", que preferiríamos que fueran las últimas subcategorías al presentar los desgloses. Así que cambiamos esta columna por un factor y la reordenamos. Esto se trata con más detalle en la página de [Factores](#factors).

### geom\_bar()

Utiliza geom\_bar() si desea que la altura de las barras (o la altura de los componentes de las barras apiladas) refleje el número de filas relevantes de los datos. Estas barras tendrán huecos entre ellas, a menos que se ajuste la estética de width = plot.

* Proporcione sólo una asignación de columna de eje (normalmente el eje x). Si proporciona x e y, obtendrá Error: stat\_count() sólo puede tener una estética x o y.
* Puedes crear barras apiladas añadiendo una asignación de relleno = columna dentro de mapping = aes()
* El eje opuesto se llamará por defecto "recuento", ya que representa el número de filas

A continuación, hemos asignado el resultado al eje Y, pero podría estar fácilmente en el eje X. Si tiene valores de caracteres más largos, a veces puede parecer mejor invertir las barras de lado y poner la leyenda en la parte inferior. Esto puede afectar a la forma en que se ordenan los niveles de los factores - en este caso los invertimos con fct\_rev() para poner los faltantes y otros en la parte inferior.

### geom\_col()

Utiliza geom\_col() si desea que la altura de las barras (o la altura de los componentes de las barras apiladas) refleje valores precalculados que existen en los datos. A menudo, se trata de recuentos sumarios o "agregados", o de proporciones.

Proporcione las asignaciones de columna para ambos ejes a geom\_col(). Normalmente la columna del eje x es discreta y la del eje y es numérica.

Digamos que tenemos los resultados de este conjunto de datos:

A continuación se muestra un código que utiliza geom\_col para crear gráficos de barras sencillos que muestren la distribución de los resultados de los pacientes con ébola. Con geom\_col, es necesario especificar tanto x como y. Aquí x es la variable categórica a lo largo del eje x, e y es la proporción de la columna de proporciones generada.

Para mostrar los desgloses por hospital, necesitaríamos que nuestra tabla contuviera más información, y que estuviera en formato "largo". Creamos esta tabla con las frecuencias de las categorías combinadas resultado y hospital (véase la página de [Agrupar datos](#grouping-data) para obtener consejos de agrupación).

A continuación, creamos el ggplot con algún formato añadido:

* **Cambio de eje**: Intercambiamos los ejes con coord\_flip() para poder leer los nombres de los hospitales.
* **Columnas de lado a lado**: Se ha añadido un argumento de posición = "esquivar" para que las barras de muerte y recuperación se presenten una al lado de la otra en lugar de apiladas. Ten en cuenta que las barras apiladas son las predeterminadas.
* **Ancho de columna**: Se especifica "ancho", por lo que las columnas son la mitad del ancho posible.
* **Orden de las columnas**: Se ha invertido el orden de las categorías en el eje y para que "Otros" y "Falta" estén en la parte inferior, con scale\_x\_discrete(limits=rev). Ten en cuenta que usamos eso en lugar de scale\_y\_discrete porque hospital se indica en el argumento x de aes(), aunque visualmente esté en el eje y. Hacemos esto porque Ggplot parece presentar las categorías al revés a menos que le digamos que no lo haga.
* **Otros detalles**: Etiquetas/títulos y colores añadidos dentro de labs y scale\_fill\_color respectivamente.

Ten en cuenta que las proporciones son binarias, por lo que podemos preferir omitir "recuperar" y mostrar sólo la proporción que murió. Esto es sólo a título ilustrativo.

Si se utiliza geom\_col() con datos de fechas (por ejemplo, una epicurva a partir de datos agregados) - querrá ajustar el argumento width = para eliminar las líneas de "hueco" entre las barras. Si se utilizan datos diarios, ajuste el ancho = 1. Si se trata de datos semanales, la anchura es de 7. Los meses no son posibles porque cada mes tiene un número diferente de días.

### geom\_histograma()

Los histogramas pueden parecerse a los gráficos de barras, pero son distintos porque miden la distribución de una variable continua. No hay espacios entre las "barras" y sólo se proporciona una columna a geom\_histogram(). Hay argumentos específicos para los histogramas como bin\_width = y breaks = para especificar cómo se deben dividir los datos. La sección anterior sobre datos continuos y la página sobre [curvas epidémicas](#epidemic-curves) proporcionan detalles adicionales.

## Recursos

Hay una gran cantidad de ayuda en línea, especialmente con ggplot. Ver:

* [Hoja de trucos de ggplot2](http://r-statistics.co/ggplot2-cheatsheet.html)
* [otra hoja de trucos](https://biostats.w.uib.no/the-ggplot2-cheat-sheet-by-rstudio/)
* [página de fundamentos de tidyverse ggplot](https://ggplot2.tidyverse.org/reference/)
* [trazado de variables continuas](http://www.sthda.com/english/articles/32-r-graphics-essentials/131-plot-two-continuous-variables-scatter-graph-and-alternatives/)
* Páginas de R for Data Science sobre [visualización de datos](https://r4ds.had.co.nz/data-visualisation.html)
* [gráficos para la comunicación](https://r4ds.had.co.nz/graphics-for-communication.html)

# #Consejos de ggplot

{#ggplot-tips}

En esta página cubriremos consejos y trucos para hacer que tus ggplots sean nítidos y elegantes. Consulte la página sobre [los fundamentos de ggplot](#ggplot-basics) para conocer los fundamentos.

Hay varios [tutoriales](https://ggplot2.tidyverse.org/) extensos de [**ggplot2**](https://ggplot2.tidyverse.org/) enlazados en la sección de Recursos. También puede descargar esta [hoja de trucos de visualización de datos con ggplot](https://rstudio.com/resources/cheatsheets/) desde el sitio web de RStudio. Recomendamos encarecidamente que busques inspiración en la [galería de gráficos de R](https://www.r-graph-gallery.com/) y en [Data-to-viz](https://www.data-to-viz.com/caveats.html).

## Preparación

### Cargar paquetes

Este trozo de código muestra la carga de los paquetes necesarios para los análisis. En este manual destacamos p\_load() de **pacman**, que instala el paquete si es necesario y lo carga para su uso. También puede cargar los paquetes instalados con library() de R **base.** Consulte la página sobre [los fundamentos de R](#r-basics) para obtener más información sobre los paquetes de R.

### Importar datos

Para esta página, importamos el conjunto de datos de casos de una epidemia de ébola simulada. Si quiere seguir el proceso, [clica para descargar linelist "limpia"](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/case_linelists/linelist_cleaned.rds) (como archivo .rds). Importe los datos con la función import() del paquete **rio** (maneja muchos tipos de archivos como .xlsx, .csv, .rds - vea la página de [importación y exportación](#import-and-export) para más detalles).

A continuación se muestran las primeras 50 filas del listado.

## Escalas de color, relleno, ejes, etc.

En **ggplot2**, cuando la estética de los datos trazados (por ejemplo, el tamaño, el color, la forma, el relleno, el eje de trazado) se asigna a las columnas de los datos, la visualización exacta se puede ajustar con el correspondiente comando "scale". En esta sección explicamos algunos ajustes de escala comunes.

### Esquemas de color

Una cosa que puede ser inicialmente difícil de entender con **ggplot2** es el control de los esquemas de color. Ten en cuenta que esta sección discute el color de los objetos de trazado (geoms/formas) como puntos, barras, líneas, mosaicos, etc. Para ajustar el color del texto accesorio, los títulos o el color de fondo, consulte la sección [Temas de](#ggplot_basics_themes) la página de [fundamentos de ggplot](#ggplot-basics).

Para controlar el "color" de los objetos de la gráfica se ajustará el color = estético (el color exterior) o el relleno = estético (el color interior). Una excepción a este patrón es geom\_point(), donde realmente sólo se llega a controlar color =, que controla el color del punto (interior y exterior).

Al establecer el color o el relleno puede utilizar nombres de colores reconocidos por R como "rojo" (ver [lista completa](http://sape.inf.usi.ch/quick-reference/ggplot2/colour) o introducir ?colores), o un color hexadecimal específico como "#ff0505".

Como se explica en la sección de [fundamentos de ggplot](#ggplot-basics) sobre la [asignación de datos al gráfico](#ggplot_basics_mapping), la estética como el relleno = y el color = puede definirse fuera de una sentencia mapping = aes() o dentro de ella. Si está fuera de aes(), el valor asignado debe ser estático (por ejemplo, color = "azul") y se aplicará a todos los datos trazados por el geom. Si está dentro, la estética debe asignarse a una columna, como color = hospital, y la expresión variará según el valor de esa fila en los datos. Algunos ejemplos:

### Escalas

Una vez que se asigna una columna a una estética de la gráfica (por ejemplo, x =, y =, relleno =, color =...), su gráfica ganará una escala / leyenda. Vea arriba cómo la escala puede ser continua, discreta, de fecha, etc. dependiendo del tipo de la columna asignada. Si tiene múltiples estéticas asignadas a las columnas, su gráfico tendrá múltiples escalas.

Puedes controlar las escalas con la función scales\_() apropiada. Las funciones de escala de **ggplot()** tienen 3 partes que se escriben así: scale\_AESTHETIC\_METHOD().

1. La primera parte, scale\_(), es fija.
2. La segunda parte, la ESTÉTICA, debe ser la estética para la que desea ajustar la escala (\_relleno\_, \_forma\_, \_color\_, \_tamaño\_, \_alfa\_...) - las opciones aquí también incluyen \_x\_ e \_y\_.
3. La tercera parte, el MÉTODO, será \_discreto(), continuo(), \_fecha(), \_gradiente(), o \_manual() dependiendo del tipo de la columna y de cómo se quiera controlar. Hay otros, pero estos son los más utilizados.

Asegúrese de utilizar la función correcta para la escala. De lo contrario, su comando de escala no parecerá cambiar nada. Si tiene varias escalas, puede utilizar varias funciones de escala para ajustarlas. Por ejemplo:

### Argumentos de la escala

Cada tipo de escala tiene sus propios argumentos, aunque hay cierto solapamiento. Consulte la función como ?scale\_color\_discrete en la consola de R para ver la documentación de los argumentos de la función.

Para escalas continuas, utiliza breaks = para proporcionar una secuencia de valores con seq() (tomar a =, desde =, y por = como se muestra en el ejemplo siguiente. Establezca expand = c(0,0) para eliminar el espacio de relleno alrededor de los ejes (esto se puede utilizar en cualquier escala \_x\_ o \_y\_.

En el caso de las escalas discretas, puede ajustar el orden de aparición de los niveles con los saltos =, y cómo se muestran los valores con el argumento labels =. Proporcione un vector de caracteres a cada uno de ellos (véase el ejemplo siguiente). También puede dejar de lado NA fácilmente estableciendo na.translate = FALSE.

Los matices de las escalas de fechas se tratan más ampliamente en la página de las [curvas epidémicas](#epidemic-curves).

### Ajustes manuales

Uno de los trucos más útiles es utilizar las funciones de escalado "manual" para asignar explícitamente los colores como se desee. Son funciones con la sintaxis scale\_xxx\_manual() (por ejemplo, scale\_colour\_manual() o scale\_fill\_manual()). Cada uno de los argumentos siguientes se demuestra en el ejemplo de código que sigue.

* Asignar colores a los valores de los datos con el argumento values =
* Especifica un color para NA con na.value =
* Cambia cómo se escriben los valores en la leyenda con el argumento labels =
* Cambiar el título de la leyenda con el nombre =

A continuación, creamos un gráfico de barras y mostramos cómo aparece por defecto, y luego con tres escalas ajustadas - la escala continua del eje y, la escala discreta del eje x, y el ajuste manual del relleno (color interior de la barra).

### Escalas de ejes continuos

Cuando los datos se mapean a los ejes del gráfico, éstos también pueden ajustarse con comandos de escalas. Un ejemplo común es el ajuste de la visualización de un eje (por ejemplo, el eje Y) que se asigna a una columna con datos continuos.

Es posible que queramos ajustar los descansos o la visualización de los valores en el ggplot utilizando scale\_y\_continuous(). Como se indicó anteriormente, utiliza el argumento breaks = para proporcionar una secuencia de valores que servirán como "saltos" a lo largo de la escala. Estos son los valores en los que se mostrarán los números. A este argumento, puede proporcionar un vector c() que contenga los valores de ruptura deseados, o puede proporcionar una secuencia regular de números utilizando la función **base** de R seq(). Esta función seq() acepta hasta =, desde =, y por =.

#### Mostrar porcentajes

Si sus valores de datos originales son proporciones, puede mostrarlos fácilmente como porcentajes con "%" proporcionando etiquetas = escalas::porcentaje en su comando de escalas, como se muestra a continuación.

Aunque una alternativa sería convertir los valores en caracteres y añadir un carácter "%" al final, este enfoque causará complicaciones porque sus datos ya no serán valores numéricos continuos.

#### Escala logarítmica

Para transformar un eje continuo a escala logarítmica, añada trans = "log2" al comando de escala. A modo de ejemplo, creamos un dataframe de regiones con sus respectivos valores de índice de preparación y casos acumulados.

Los casos acumulados de la región "I" son mucho mayores que los de las demás regiones. En circunstancias como ésta, puede optar por mostrar el eje Y utilizando una escala logarítmica para que el lector pueda ver las diferencias entre las regiones con menos casos acumulados.

### Escalas de gradiente

Las escalas de degradado de relleno pueden implicar matices adicionales. Los valores predeterminados suelen ser bastante agradables, pero es posible que desee ajustar los valores, los cortes, etc.

Para demostrar cómo ajustar una escala de colores continua, utilizaremos unos datos de la página de [seguimiento de contactos](#contact-tracing-1) que contiene las edades de los casos y de sus casos de origen.

A continuación, producimos un gráfico de densidad de baldosas de calor "rasterizado". No vamos a elaborar cómo (ver el enlace en el párrafo anterior), pero nos centraremos en cómo podemos ajustar la escala de colores. Lea más sobre la función **ggplot2** stat\_density2d() [aquí](https://ggplot2.tidyverse.org/reference/geom_density_2d.html). Observe cómo la escala de relleno es continua.

Ahora mostramos algunas variaciones en la escala de relleno:

Ahora mostramos algunos ejemplos de cómo ajustar realmente los puntos de ruptura de la escala:

* scale\_fill\_gradient() acepta dos colores (alto/bajo)
* scale\_fill\_gradientn() acepta un vector de cualquier longitud de colores a valores = (los valores intermedios serán interpolados)
* Utiliza [scales::rescale()](https://www.rdocumentation.org/packages/scales/versions/0.4.1/topics/rescale) para ajustar la posición de los colores a lo largo del gradiente; reescala su vector de posiciones para que esté entre 0 y 1.

### Paletas

#### Colorbrewer y Viridis

En general, si desea paletas predefinidas, puede utilizar las funciones scale\_xxx\_brewer o scale\_xxx\_viridis\_y.

Las funciones 'brewer' pueden dibujar desde las paletas de [colorbrewer.org](../../../../../C:/Users/Neale/OneDrive%20-%20Neale%20Batra/Documents/Analytic%20Software/R/Projects/R%20handbook/Epi_R_handbook/colorbrewer.org).

Las funciones "viridis" se basan en las paletas viridis (¡difíciles para los daltónicos!), que "proporcionan mapas de color que son perceptivamente uniformes tanto en color como en blanco y negro. También están diseñadas para ser percibidas por espectadores con formas comunes de daltonismo". (lea más [aquí](https://ggplot2.tidyverse.org/reference/scale_viridis.html) y [aquí](https://bids.github.io/colormap/)). Defina si la paleta es discreta, continua o con bines especificando esto al final de la función (por ejemplo, discreta es scale\_xxx\_viridis\_d).

Se aconseja que pruebe su esquema en este [simulador de daltonismo](https://www.color-blindness.com/coblis-color-blindness-simulator/). Si tiene un esquema de color rojo/verde, pruebe en su lugar un esquema "caliente-frío" (rojo-azul) como se describe [aquí](https://www.visualisingdata.com/2019/08/five-ways-to-design-for-red-green-colour-blindness/" \l ":~:text=The pink-red through to,green hues used by default.)

Aquí hay un ejemplo de la página de [fundamentos de ggplot](#ggplot-basics), utilizando varios esquemas de color.

## Cambiar el orden de las variables discretas

Cambiar el orden en que aparecen las variables discretas es a menudo difícil de entender para las personas que son nuevas en los gráficos de ggplot2. Sin embargo, es fácil de entender cómo hacer esto una vez que se entiende cómo ggplot2 maneja las variables discretas bajo el capó. En general, si se utiliza una variable discreta, se convierte automáticamente en un tipo de factor - que ordena los factores por orden alfabético por defecto. Para manejar esto, simplemente tiene que reordenar los niveles de los factores para reflejar el orden en que le gustaría que aparecieran en el gráfico. Para obtener información más detallada sobre cómo reordenar los objetos de factor, consulte la sección de factores de la guía.

Podemos ver un ejemplo común utilizando los grupos de edad - por defecto el grupo de 5 a 9 años se colocará en medio de los grupos de edad (dado el orden alfanumérico), pero podemos moverlo detrás del grupo de 0 a 4 años del gráfico renivelando los factores.

#### ****ggthemr****

También considere el uso del paquete **ggthemr**. Puedes descargar este paquete desde Github usando las instrucciones [aquí](https://github.com/Mikata-Project/ggthemr). Ofrece paletas que son muy agradables estéticamente, pero ten en cuenta que estas suelen tener un número máximo de valores que puede ser limitante si quieres más de 7 u 8 colores.

## Contornos

Los gráficos de contorno son útiles cuando se tienen muchos puntos que pueden cubrirse unos a otros ("sobretrazado"). Los datos de la fuente de casos utilizados anteriormente se trazan de nuevo, pero de forma más sencilla utilizando stat\_density2d() y stat\_density2d\_filled() para producir niveles de contorno discretos - como un mapa topográfico. Lea más sobre las estadísticas [aquí](https://ggplot2.tidyverse.org/reference/geom_density_2d.html).

## Distribuciones marginales

Para mostrar las distribuciones en los bordes de un gráfico de dispersión geom\_point(), puede utilizar el paquete **ggExtra** y su función ggMarginal(). Guarde su ggplot original como un objeto, y páselo a ggMarginal() como se muestra a continuación. Estos son los argumentos clave:

* Debe especificar el tipo = como "histograma", "densidad" "boxplot", "violín" o "densigrama".
* Por defecto, los gráficos marginales aparecerán para ambos ejes. Puedes establecer márgenes = a "x" o "y" si sólo quiere uno.
* Otros argumentos opcionales son fill = (color de la barra), color = (color de la línea), size = (tamaño del gráfico en relación con el tamaño del margen, por lo que un número mayor hace que el gráfico marginal sea más pequeño).
* Puedes proporcionar otros argumentos específicos del eje a xparams = e yparams =. Por ejemplo, para tener diferentes tamaños de cubos de histograma, como se muestra a continuación.

Puedes hacer que los gráficos marginales reflejen grupos (columnas a las que se les ha asignado un color = en su estética mapeada de ggplot()). Si este es el caso, establezca el argumento ggMarginal() groupColour = o groupFill = a TRUE, como se muestra a continuación.

Lea más en [esta viñeta](https://cran.r-project.org/web/packages/ggExtra/vignettes/ggExtra.html), en la [galería de gráficos de R](https://www.r-graph-gallery.com/277-marginal-histogram-for-ggplot2.html) o en la documentación de la función R ?ggMarginal.

Para añadir histogramas marginales utiliza type = "histograma". Opcionalmente puede establecer groupFill = TRUE para obtener histogramas apilados.

Gráfico de densidad marginal con valores agrupados/coloreados:

Establezca el argumento size = para ajustar el tamaño relativo del gráfico marginal. Un número más pequeño hace un gráfico marginal más grande. También se establece el color =. A continuación se muestra un boxplot marginal, con la demostración de los márgenes = argumento por lo que aparece en un solo eje:

## Etiquetado inteligente

En **ggplot2**, también es posible añadir texto a los gráficos. Sin embargo, esto viene con la notable limitación de que las etiquetas de texto a menudo chocan con los puntos de datos en un gráfico, haciendo que se vean desordenados o difíciles de leer. No hay una manera ideal de lidiar con esto en el paquete base, pero hay un complemento de **ggplot2**, conocido como **ggrepel** que hace que esto sea muy simple.

El paquete **ggrepel** proporciona dos nuevas funciones, geom\_label\_repel() y geom\_text\_repel(), que sustituyen a geom\_label() y geom\_text(). Basta con utilizar estas funciones en lugar de las funciones base para producir etiquetas ordenadas. Dentro de la función, mapee la estética() como siempre, pero incluya el argumento label = al que le proporciona un nombre de columna que contenga los valores que desea mostrar (por ejemplo, id de paciente, o nombre, etc.). Puedes hacer etiquetas más complejas combinando columnas y nuevas líneas (\n) dentro de str\_glue() como se muestra a continuación.

Algunos consejos:

* Utiliza min.segment.length = 0 para dibujar siempre segmentos de línea, o min.segment.length = Inf para no dibujarlos nunca
* Utiliza size = fuera de aes() para establecer el tamaño del texto
* Utiliza fuerza = para cambiar el grado de repulsión entre las etiquetas y sus respectivos puntos (por defecto es 1)
* Incluir fill = dentro de aes() para tener la etiqueta coloreada por el valor
  + Puedes aparecer una letra "a" en la leyenda - añada guides(fill = guide\_legend(override.aes = aes(color = NA)))+ para eliminarla

Para más información, consulte este [tutorial](https://ggrepel.slowkow.com/articles/examples.html) en profundidad.

Puedes etiquetar sólo un subconjunto de los puntos de datos - utilizando la sintaxis estándar de ggplot() para proporcionar diferentes datos = para cada capa geomática del gráfico. A continuación, se trazan todos los casos, pero sólo se etiquetan algunos.

## Ejes temporales

Trabajar con ejes de tiempo en ggplot puede parecer desalentador, pero se hace muy fácil con unas pocas funciones clave. Recuerde que cuando trabaje con el tiempo o la fecha debe asegurarse de que las variables correctas están formateadas como tipo date o datetime - vea la página [Trabajar con fechas](#working-with-dates-1) para más información sobre esto, o la página [Curvas epidémicas](#epidemic-curves) (sección ggplot) para ver ejemplos.

El conjunto de funciones más útil para trabajar con fechas en ggplot2 son las funciones de escala (scale\_x\_date(), scale\_x\_datetime(), y sus funciones afines del eje y). Estas funciones le permiten definir la frecuencia de las etiquetas de los ejes, y cómo formatear las etiquetas de los ejes. Para saber cómo dar formato a las fechas, vuelva a ver la sección de trabajo con fechas. Puedes utilizar los argumentos date\_breaks y date\_labels para especificar el aspecto de las fechas:

1. date\_breaks permite especificar la frecuencia con la que se producen las rupturas de los ejes - se puede pasar aquí una cadena (por ejemplo, "3 meses", o "2 días")
2. date\_labels permite definir el formato en el que se muestran las fechas. Puedes pasar una cadena de formato de fecha a estos argumentos (por ejemplo, "%b-%d-%Y"):

## Destacando

Resaltar elementos específicos en un gráfico es una forma útil de llamar la atención sobre una instancia específica de una variable, a la vez que se proporciona información sobre la dispersión de los datos. Aunque esto no es fácil de hacer en **ggplot2** base, hay un paquete externo que puede ayudar a hacer esto conocido como **gghighlight**. Es fácil de usar dentro de la sintaxis de ggplot.

El paquete **gghighlight** utiliza la función gghighlight() para lograr este efecto. Para usar esta función, suministre una declaración lógica a la función - esto puede tener resultados bastante flexibles, pero aquí mostraremos un ejemplo de la distribución de edad de los casos en nuestra lista de líneas, resaltándolos por resultado.

Esto también funciona bien con las funciones de facetas - ¡permite al usuario producir gráficos de facetas con los datos de fondo resaltados que no se aplican a la faceta! A continuación, contamos los casos por semana y trazamos las curvas de epidemia por hospital (color = y facet\_wrap() ajustado a la columna del hospital).

## Trazado de múltiples conjuntos de datos

Ten en cuenta que alinear correctamente los ejes para trazar desde múltiples conjuntos de datos en el mismo gráfico puede ser difícil. Considere una de las siguientes estrategias:

* Fusionar los datos antes de trazarlos y convertirlos al formato "largo" con una columna que refleje el conjunto de datos
* Utiliza **Cowplot** o un paquete similar para combinar dos gráficos (véase más abajo)

## Combinar gráficos

Dos paquetes que son muy útiles para combinar gráficos son **cowplot** y **patchwork**. En esta página nos centraremos principalmente en **cowplot**, con el uso ocasional de **patchwork**.

Aquí está la [introducción](https://cran.r-project.org/web/packages/cowplot/vignettes/introduction.html) en línea [a cowplot](https://cran.r-project.org/web/packages/cowplot/vignettes/introduction.html). Puedes leer la documentación más extensa de cada función en línea [aquí](https://www.rdocumentation.org/packages/cowplot/versions/1.1.1). A continuación cubriremos algunos de los casos de uso y funciones más comunes.

El paquete **cowplot** funciona en tándem con **ggplot2** - esencialmente, se utiliza para organizar y combinar ggplots y sus leyendas en figuras compuestas. También puede aceptar gráficos **básicos de** R.

Mientras que las facetas (descritas en la página de [fundamentos de ggplot](#ggplot-basics)) son un enfoque conveniente para el trazado, a veces no es posible obtener los resultados que desea de su enfoque relativamente restrictivo. En este caso, se puede optar por combinar los gráficos pegándolos en un gráfico más grande. Hay tres paquetes bien conocidos que son excelentes para esto - **cowplot**, **gridExtra**, y **patchwork**. Sin embargo, estos paquetes hacen en gran medida las mismas cosas, por lo que nos centraremos en **cowplot** para esta sección.

### plot\_grid()

El paquete **cowplot** tiene una gama bastante amplia de funciones, pero el uso más fácil de él se puede lograr mediante el uso de plot\_grid(). Se trata de una forma de organizar los gráficos predefinidos en una cuadrícula. Podemos trabajar a través de otro ejemplo con el conjunto de datos de la malaria - aquí podemos trazar el total de casos por distrito, y también mostrar la curva epidémica en el tiempo.

### Combinar leyendas

Si sus gráficos tienen la misma leyenda, combinarlos es relativamente sencillo. Simplemente utiliza el método de **cowplot** anterior para combinar los gráficos, pero elimine la leyenda de uno de ellos (desduplique).

Si sus gráficos tienen leyendas diferentes, debe utilizar un enfoque alternativo:

1. Cree y guarde sus gráficos sin leyendas utilizando theme(legend.position = "none")
2. Extraiga las leyendas de cada gráfica utilizando get\_legend() como se muestra a continuación - pero extraiga las leyendas de los gráficos modificadas para mostrar realmente la leyenda
3. Combinar las leyendas en un panel de leyendas
4. Combinar el panel de gráficos y leyendas

Para la demostración mostramos las dos gráficos por separado, y luego dispuestas en una cuadrícula con sus propias leyendas mostrando (uso feo e ineficiente del espacio):

Así es como se ven los dos gráficos cuando se combinan usando plot\_grid() sin combinar sus leyendas:

Y ahora mostramos cómo combinar las leyendas. Esencialmente lo que hacemos es definir cada gráfica sin su leyenda (theme(legend.position = "none"), y luego definimos la leyenda de cada gráfica por separado, utilizando la función get\_legend() de **cowplot**. Cuando extraemos la leyenda del gráfico guardado, tenemos que añadir + la leyenda de nuevo, incluyendo la especificación de la colocación ("derecha") y pequeños ajustes para la alineación de las leyendas y sus títulos. A continuación, combinamos las leyendas verticalmente, y luego combinamos los dos gráficos con las leyendas recién combinadas. Y ya está.

Esta solución fue aprendida de [este post](https://stackoverflow.com/questions/52060601/ggplot-multiple-legends-arrangement) con un arreglo menor para alinear las leyendas de [este post](https://github.com/wilkelab/cowplot/issues/33).

**CONSEJO:** Nota divertida: la "vaca" en **cowplot** viene del nombre del creador: Claus O. Wilke.

### Gráficos de inserción

Puedes insertar una gráfica en otra utilizando **cowplot**. Aquí hay cosas que hay que tener en cuenta:

* Defina el gráfico principal con theme\_half\_open() de **cowplot**; puede ser mejor tener la leyenda arriba o abajo
* Defina el gráfico de inserción. Lo mejor es tener un gráfico en el que no se necesite una leyenda. Puedes eliminar los elementos del tema del gráfico con element\_blank() como se muestra a continuación.
* Combínelos aplicando ggdraw() al gráfico principal, y luego añadiendo draw\_plot() en el gráfico de inserción y especificando las coordenadas (x e y de la esquina inferior izquierda), la altura y la anchura como proporción de todo el gráfico principal.

Esta técnica se explica mejor en estas dos viñetas:

[Laboratorio Wilke](https://wilkelab.org/cowplot/articles/drawing_with_on_plots.html)  
[documentación de draw\_plot()](https://www.rdocumentation.org/packages/cowplot/versions/1.1.1/topics/draw_plot)

## Ejes dobles

Un eje Y secundario es a menudo una adición solicitada a un gráfico ggplot2. Aunque existe un fuerte debate sobre la validez de estos gráficos en la comunidad de visualización de datos, y a menudo no se recomiendan, es posible que su gestor los quiera. A continuación, presentamos un método para conseguirlos: usar el paquete **cowplot** para combinar dos gráficos separados.

Este enfoque implica la creación de dos gráficos separadas - una con un eje y a la izquierda, y la otra con un eje y a la derecha. Ambos utilizarán un tema específico de theme\_cowplot() y deben tener el mismo eje x. Luego, en un tercer comando, los dos gráficos se alinean y se superponen. Las funcionalidades de **cowplot**, de las que ésta es sólo una, se describen en profundidad en este [sitio](https://wilkelab.org/cowplot/articles/aligning_plots.html).

Para demostrar esta técnica, superpondremos la curva epidémica con una línea del porcentaje semanal de pacientes fallecidos. Utilizamos este ejemplo porque la alineación de las fechas en el eje x es más compleja que, por ejemplo, alinear un gráfico de barras con otro gráfico. Hay que tener en cuenta algunas cosas:

* La epicurva y la línea se agregan en semanas antes de trazarlas y los date\_breaks y date\_labels son idénticos - lo hacemos para que los ejes x de los dos gráficos sean los mismos cuando se superponen.
* El eje y se mueve a la derecha para el gráfico 2 con el argumento position = de scale\_y\_continuous().
* Ambos gráficos hacen uso de theme\_cowplot()

Obsérvese que hay otro ejemplo de esta técnica en la página de [curvas epidémicas](#epidemic-curves): la superposición de la incidencia acumulada sobre la epicurva.

**Make plot**   
1Esto es esencialmente la epicurva. Usamos geom\_area() sólo para demostrar su uso (área bajo una línea, por defecto)

**Haga el gráfico 2Cree**   
el segundo gráfico mostrando una línea del porcentaje semanal de casos que murieron.

Ahora alineamos el gráfico utilizando la función align\_plots(), especificando la alineación horizontal y vertical ("hv", también podría ser "h", "v", "none"). También especificamos la alineación de todos los ejes (superior, inferior, izquierdo y derecho) con "tblr". La salida es de tipo lista (2 elementos).

Luego dibujamos los dos gráficos juntos usando ggdraw() (de **cowplot**) y referenciando las dos partes del objeto aligned\_plots.

## Paquetes para ayudarle

Hay algunos paquetes de R muy buenos diseñados específicamente para ayudarle a navegar por **ggplot2**:

### Apuntar y clicar en ****ggplot2**** con ****equisse****

"Este complemento le permite explorar interactivamente sus datos visualizándolos con el paquete ggplot2. Le permite dibujar gráficos de barras, curvas, gráficos de dispersión, histogramas, boxplot y objetos sf, y luego exportar el gráfico o recuperar el código para reproducir el gráfico."

Instale y lance el complemento a través del menú de RStudio o con esquisse::esquisser().

Ver la [página de Github](https://github.com/dreamRs/esquisse)

[Documentación](https://dreamrs.github.io/esquisse/index.html)

## Varios

### Pantalla numérica

Puedes desactivar la notación científica ejecutando este comando antes de trazar.

O aplicar number\_format() del paquete de **escalas** a un valor o columna específicos, como se muestra a continuación.

Utiliza las funciones de las **escalas** del paquete para ajustar fácilmente la forma en que se muestran los números. Pueden aplicarse a las columnas del dataframe, pero se muestran en los números individuales a modo de ejemplo.

## Recursos

Inspiración [ggplot graph gallery](https://www.tidyverse.org/blog/2018/07/ggplot2-3-0-0/)

Presentación de los datos Centro Europeo para la Prevención y el Control de las Enfermedades [Directrices para la presentación de los datos de vigilancia](https://ecdc.europa.eu/sites/portal/files/documents/Guidelines for presentation of surveillance data-final-with-cover-for-we....pdf)

Facetas y etiquetadoras [Utilización de etiquetadoras para tiras de facetas](http://www.cookbook-r.com/Graphs/Facets_(ggplot2)/" \l "modifying-facet-label-text) [Etiquetadoras](https://ggplot2.tidyverse.org/reference/labellers.html)

Ajuste del orden con factores [fct\_reorder](https://forcats.tidyverse.org/reference/fct_reorder.html)  
[fct\_inorder](https://forcats.tidyverse.org/reference/fct_inorder.html)  
[Cómo reordenar un boxplot](https://cmdlinetips.com/2019/02/how-to-reorder-a-boxplot-in-r/)  
[Reordenar una variable en ggplot2](https://www.r-graph-gallery.com/267-reorder-a-variable-in-ggplot2.html)  
[R for Data Science - Factores](https://r4ds.had.co.nz/factors.html)

Leyendas  
[Ajustar el orden de las leyendas](https://stackoverflow.com/questions/38425908/reverse-stacking-order-without-affecting-legend-order-in-ggplot2-bar-charts)

Leyendas [Alineación de las leyendas](https://stackoverflow.com/questions/64701500/left-align-ggplot-caption)

Etiquetas  
[ggrepel](https://ggrepel.slowkow.com/articles/examples.html)

Hojas de trucos  
[Trazado bonito con ggplot2](http://zevross.com/blog/2014/08/04/beautiful-plotting-in-r-a-ggplot2-cheatsheet-3/)

# #Curvas epidémicas

{#epidemic-curves}

Una curva epidémica (también conocida como "curva epi") es un gráfico epidemiológico básico que se suele utilizar para visualizar el patrón temporal de aparición de la enfermedad entre un grupo o epidemia de casos.

El análisis de la epicurva puede revelar tendencias temporales, valores atípicos, la magnitud del brote, el periodo de exposición más probable, los intervalos de tiempo entre las generaciones de casos, e incluso puede ayudar a identificar el modo de transmisión de una enfermedad no identificada (por ejemplo, fuente puntual, fuente común continua, propagación de persona a persona). En el sitio web de [los CDC de EE.UU. se](https://www.cdc.gov/training/quicklearns/epimode/index.html) puede encontrar una lección en línea sobre la interpretación de las curvas de epi.

En esta página demostramos dos enfoques para producir epicurvas en R:

* El paquete **incidence2**, que puede producir una curva epi con simples comandos
* El paquete **ggplot2**, que permite una personalización avanzada mediante comandos más complejos

También se abordan casos de uso específicos como:

* Trazado de datos de recuento agregados
* Facetado o producción de múltiplos pequeños
* Aplicación de medias móviles
* Mostrar qué datos son "provisionales" o están sujetos a retrasos en la presentación de informes
* Superposición de la incidencia de casos acumulados mediante un segundo eje

## Preparación

### Paquetes

Este trozo de código muestra la carga de los paquetes necesarios para los análisis. En este manual destacamos p\_load() de **pacman**, que instala el paquete si es necesario y lo carga para su uso. También puede cargar los paquetes instalados con library() de R **base.** Consulte la página sobre [los fundamentos de R](#r-basics) para obtener más información sobre los paquetes de R.

### Importar datos

En esta sección se utilizan dos conjuntos de datos de ejemplo:

* Lista de casos individuales de una epidemia simulada
* Recuentos agregados por hospital de la misma epidemia simulada

Los conjuntos de datos se importan mediante la función import() del paquete **rio**. Consulte la página sobre [importación y exportación](#import-and-export) para conocer las distintas formas de importar datos.

**Lista de casos**

Importamos el conjunto de datos de casos de una epidemia de ébola simulada. Si desea descargar los datos para seguirlos paso a paso, consulte las instrucciones en la página de [descarga de manuales y datos](#download-handbook-and-data). Asumimos que el archivo está en el directorio de trabajo, por lo que no se especifican subcarpetas en esta ruta de archivo.

A continuación se muestran las primeras 50 filas.

**Recuentos de casos agregados por hospital**

A efectos del manual, el conjunto de datos de recuentos semanales agregados por hospital se crea a partir del listado con el siguiente código.

A continuación se muestran las primeras 50 filas:

### Establecer parámetros

Para la producción de un informe, es posible que desee establecer parámetros editables como la fecha para la que los datos son actuales (la "fecha de datos"). A continuación, puede hacer referencia al objeto fecha\_datos en su código cuando aplique filtros o en subtítulos dinámicos.

### Verificar las fechas

Verifique que cada columna de fecha relevante es de tipo Date y tiene un rango de valores apropiado. Puedes hacerlo simplemente utilizando hist() para histogramas, o range() con na.rm=TRUE, o con ggplot() como se indica a continuación.

## Epicurves con paquete de ****incidencia2****

A continuación mostramos cómo hacer epicurvas utilizando el paquete **incidence2**. Los autores de este paquete han intentado que el usuario pueda crear y modificar epicurvas sin necesidad de conocer la sintaxis **de ggplot2**. Gran parte de esta página está adaptada de las viñetas del paquete, que se pueden encontrar en la [página de github de](https://github.com/reconhub/incidence2) **incidence2**.

### Ejemplo sencillo

**Se requieren 2 pasos para trazar una curva epidémica con el paquete incidence2:**

1. **Crear** un objeto de incidencia (utilizando la función incidencia())
   * Proporcionar los datos
   * Especifica la columna de fecha a date\_index =
   * Especifique el intervalo = en el que deben agregarse los casos (diario, semanal, mensual..)
   * Especifique cualquier columna de agrupación (por ejemplo, sexo, hospital, resultado)
2. **Trazar** el objeto de incidencia
   * Especifique etiquetas, colores, títulos, etc.

A continuación, cargamos el paquete **incidence2**, creamos el objeto de incidencia a partir del listado en la columna date\_onset y agregamos los casos por día. A continuación, imprimimos un resumen del objeto de incidencia.

El objeto **incidencia2** tiene el aspecto de un tibble (como una trama de datos) y puede imprimirse o manipularse como una trama de datos.

Este es el aspecto que tiene cuando se imprime. Tiene una columna date\_index y una columna count.

También puede imprimir un resumen del objeto:

Para trazar el objeto de incidencia, utiliza plot() en el nombre del objeto de incidencia. En segundo plano, se llama a la función plot.incidence2(), por lo que para leer la documentación **específica de la incidencia2 se** ejecutaría ?plot.incidence2.

Si nota muchas líneas blancas verticales diminutas, intente ajustar el tamaño de su imagen. Por ejemplo, si exporta su gráfico con ggsave(), puede proporcionar números a la anchura = y a la altura =. Si amplía el gráfico esas líneas pueden desaparecer.

### Cambiar el intervalo de tiempo de la agregación de casos

El argumento intervalo = de incidence() define cómo se agrupan las observaciones en barras verticales.

**Especifique el intervalo**

**incidence2** proporciona flexibilidad y una sintaxis comprensible para especificar cómo quiere agregar sus casos en barras epicúreas. Proporcione un valor como los siguientes al argumento intervalo =. Puedes escribir cualquiera de los siguientes en plural (por ejemplo, "**semanas**"), y puede añadir números antes (por ejemplo, "3 meses").

| **Opción de argumento** | **Más explicaciones** |
| --- | --- |
| Número (1, 7, 13, 14, etc.) | Número de días por intervalo |
| "semana" | nota: el día de inicio del lunes es el predeterminado |
| "2 semanas" | o 3, 4, 5... |
| "Domingo de la semana" | semanas que comienzan en domingo (también se puede utilizar el jueves, etc.) |
| "2 semanas de domingo" | o 3, 4, 5... |
| "MMWRweek" | la semana comienza en domingo - ver US CDC |
| "mes" | El 1 de mes |
| "cuarto" | 1 del mes del trimestre |
| "2 meses" | o 3, 4, 5... |
| "año" | Primer día del año natural |

A continuación se muestran ejemplos de cómo se ven los diferentes intervalos cuando se aplican a linelist. Observe cómo el formato por defecto y la frecuencia de las etiquetas de fecha en el eje x cambian a medida que cambia el intervalo de fecha.

**Primera cita**

Puedes especificar opcionalmente un valor de tipo Date (por ejemplo, as.Date("2016-05-01")) a firstdate = en el comando incidence(). Si se da, los datos se recortarán a este rango y los intervalos comenzarán en esta fecha.

### Grupos

Los grupos se especifican en el comando incidence(), y pueden utilizarse para colorear las barras o para facetar los datos. Para especificar los grupos en sus datos, proporcione el nombre de la(s) columna(s) al argumento groups = en el comando incidence() (sin comillas alrededor del nombre de la columna). Si especifica varias columnas, ponga sus nombres dentro de c().

Puedes especificar que los casos con valores faltantes en las columnas de agrupación sean listados como un grupo NA distinto estableciendo na\_as\_group = TRUE. De lo contrario, se excluirán del gráfico.

* Para colorear las barras por una columna de agrupación, debe proporcionar de nuevo el nombre de la columna para rellenar = en el comando plot().
* Para crear una faceta basada en una columna de agrupación, consulte la sección siguiente sobre facetas con **incidencia2**.

En el ejemplo siguiente, los casos de todo el brote se agrupan por su categoría de edad. Los valores faltantes se incluyen como grupo. El intervalo de la epicurva es de semanas.

**CONSEJO:** Cambie el título de la leyenda añadiendo + el comando **ggplot2** labs(fill = "su título") a su gráfico de **incidencia2**.

También puede hacer que las barras agrupadas se muestren una al lado de la otra estableciendo stack = FALSE en plot(), como se muestra a continuación:

Puedes establecer el argumento na\_as\_group = a FALSE en el comando incidence() para eliminar las filas con valores faltantes del gráfico.

### Datos filtrados

Para trazar la epicurva de un subconjunto de datos:

1. Filtrar los datos del listado
2. Proporcionar los datos filtrados al comando incidence()
3. Trazar el objeto de incidencia

El ejemplo siguiente utiliza datos filtrados para mostrar sólo los casos del Hospital Central.

### Recuentos agregados

Si sus datos originales son agregados (recuentos), proporcione el nombre de la columna que contiene los recuentos de casos al argumento count = cuando cree el objeto de incidencia con incidence().

Por ejemplo, este dataframe count\_data es linelist agregadas en recuentos diarios por hospital. Las primeras 50 filas tienen este aspecto:

Si comienza su análisis con datos de recuento diario como el conjunto de datos anterior, su comando incidence() para convertirlo en una epicurva semanal por hospital tendría el siguiente aspecto:

### Facetas/pequeños múltiplos

Facetar los datos por grupos (es decir, producir "pequeños múltiplos"):

1. Especifique la columna de facetas a grupos = cuando cree el objeto de incidencia
2. Utiliza el comando facet\_plot() en lugar de plot()
3. Especificar qué columnas de agrupación utilizar como relleno = y cuáles utilizar como facetas =

A continuación, establecemos las columnas hospital y resultado como columnas de agrupación en el comando incidence(). A continuación, en facet\_plot() trazamos la epicurva, especificando que queremos una epicurva diferente para cada hospital y que dentro de cada epicurva las barras deben estar apiladas y coloreadas por resultado.

Ten en cuenta que el paquete **ggtree** (utilizado para mostrar árboles filogenéticos) también tiene una función facet\_plot() - por eso especificamos incidence2::facet\_plot() arriba.

### Modificaciones con plot()

Una epicurva producida por **incidence2** puede ser modificada a través de estos argumentos dentro de la función *plot()*.

**Aquí están los argumentos de plot() que modifican la apariencia de las barras:**

| **Argumento** | **Descripción** | **Ejemplos** |
| --- | --- | --- |
| llenar = | Color de la barra. Un nombre de color o un nombre de columna previamente especificado a los grupos = en el comando incidence() | relleno = "rojo", o relleno = género |
| color = | Colorea alrededor de cada barra, o alrededor de cada agrupación dentro de una barra | borde = "blanco" |
| leyenda = | Ubicación de la leyenda | Una de las opciones "abajo", "arriba", "izquierda", "derecha" o "ninguna" |
| alfa = | Transparencia de las barras/cajas | 1 es totalmente opaco, 0 es totalmente transparente |
| ancho = | Valor entre 0 y 1 que indica el tamaño relativo de las barras a su intervalo de tiempo | ancho = .7 |
| show\_cases = | Lógico; si es TRUE, cada caso se muestra como una caja. Se muestra mejor en brotes pequeños. | show\_cases = TRUE |

**Aquí están los argumentos de plot() que modifican el eje de la fecha:**

| **Argumento(s)** | **Descripción** |
| --- | --- |
| centre\_dates = | TRUE/FALSE en cuanto a si la fecha aparece bajo el centro de las barras, o al principio de las mismas |
| formato\_fecha = | Ajuste el formato de visualización de la fecha utilizando la sintaxis strptime ("%"). Sólo funciona si centre\_dates = FALSE (detalles más abajo). |
| n.breaks = | Número aproximado de interrupciones de la etiqueta del eje x deseadas. |
| ángulo = | Ángulo de las etiquetas de fecha del eje x (número de grados) |
| tamaño = | Tamaño del texto en puntos |

Ten en cuenta que el argumento date\_breaks = sólo funciona si centre\_dates = FALSE. Proporcione un valor de carácter entre comillas utilizando la sintaxis strptime que se indica a continuación, como se detalla en la página [Trabajar con fechas](#working-with-dates-1). Puedes utilizar \n para una "nueva línea".

%d = Número de día del mes (5, 17, 28, etc.)  
%j = Número de día del año (día juliano 001-366)  
%a = Día de la semana abreviado (lunes, martes, miércoles, etc.)  
%A = Día de la semana completo (lunes, martes, etc.))%w = Número del día de la semana   
(0-6, el domingo es 0)  
%u = Número del día de la semana (1-7, el lunes es 1)  
%W = Número de la semana (00-53, el lunes es el comienzo de la semana)  
%U = Número de la semana (01-53, el domingo es el comienzo de la semana)  
%m = Número del mes (e.%b   
= Mes abreviado (enero, febrero, etc.  
)  
%B = Mes completo (enero, febrero, etc.)%y   
= Año de 2 dígitos (por ejemplo, 89)  
%Y = Año de 4 dígitos (por ejemplo, 1989)  
  
%h = horas (reloj de 24 horas)  
%m = minutos%s   
= segundos%z = desviación   
de GMT%Z   
= Zona horaria (carácter)

**Estos son los argumentos de plot() que modifican las etiquetas de los gráficos:**

| **Argumento(s)** | **Descripción** |
| --- | --- |
| título = | Título de la trama |
| xlab = | Título del eje x |
| ylab = | Título del eje Y |
| tamaño = | Tamaño del texto del eje x en pts (utiliza ggplot's theme() para ajustar otros tamaños) |

Un ejemplo que utiliza muchos de los argumentos anteriores:

Para ajustar aún más la apariencia del gráfico, consulte la sección siguiente sobre modificaciones con ggplot().

### Modificaciones con ggplot2

Puedes modificar aún más un gráfico de **incidencia2** añadiendo modificaciones de **ggplot2** con un + después del cierre de la función de incidencia plot(), como se demuestra a continuación.

A continuación, el gráfico de **incidencia2** termina y luego se utilizan los comandos **de ggplot2** para modificar los ejes, añadir una leyenda y ajustar la fuente en negrita y el tamaño del texto.

Ten en cuenta que si añade scale\_x\_date(), la mayor parte del formato de fecha de plot() se sobrescribirá. Consulte la sección de epicurvas de ggplot() y la página del Manual [ggplot tips](#ggplot-tips) para más opciones.

### Cambiar los colores

#### Especifique una paleta

Proporcione el nombre de una paleta predefinida al argumento col\_pal = en plot(). El paquete **incidence2** viene con 2 paletas predefinidas: "vibrant" y "muted". En "vibrante" los primeros 6 colores son distintos y en "apagado" los primeros 9 colores son distintos. Después de estos números, los colores son interpolaciones/intermediarios de otros colores. Estas paletas predefinidas se pueden encontrar en [este sitio web](https://personal.sron.nl/~pault/" \l "sec:qualitative). Las paletas excluyen el gris, que está reservado para los datos que faltan (utiliza na\_color = para cambiar este valor por defecto).

También puede utilizar una de las paletas **base** de R (ponga el nombre de la paleta sin comillas).

También puede añadir una paleta de colores del paquete **viridis** o del paquete **RColorBrewer**. Primero hay que cargar esos paquetes, y luego añadir sus respectivas funciones scale\_fill\_\*() con un +, como se muestra a continuación.

#### Especificar manualmente

Para especificar los colores manualmente, añada la función **ggplot2** scale\_fill\_manual() a la función plot() con un + y proporcione el vector de nombres de colores o códigos HEX al argumento values =. El número de colores listados debe ser igual al número de grupos. Ten en cuenta si los valores faltantes son un grupo - pueden ser convertidos a un valor de carácter como "Missing" durante la preparación de los datos con la función fct\_explicit\_na() como se explica en la página sobre [Factores](#factors).

Como se menciona en la página de [consejos de ggplot](#ggplot-tips), puede crear sus propias paletas utilizando colorRampPalette() sobre un vector de colores y especificando el número de colores que desea a cambio. Esta es una buena manera de obtener muchos colores en una rampa especificando unos pocos.

### Ajustar el orden de los niveles

Para ajustar el orden de aparición de los grupos (en el gráfico y en la leyenda), la columna de agrupación debe ser del tipo Factor. Consulte la página sobre [Factores](#factors) para obtener más información.

En primer lugar, veamos una epicurva semanal por hospital con la ordenación por defecto:

Ahora, para ajustar el orden de manera que "Falta" y "Otros" estén en la parte superior de la epicurva podemos hacer lo siguiente:

* Cargar el paquete **forcats**, para trabajar con factores
* Ajustar el conjunto de datos - en este caso vamos a definir un nuevo conjunto de datos (plot\_data) en el que:
  + la columna de género se define como un factor el orden de los niveles se establecen con fct\_relevel() de manera que "Otros" y "Falta" son los primeros, por lo que aparecen en la parte superior de las barras
* El objeto de incidencia se crea y se traza como antes
* Añadimos modificaciones de **ggplot2** 
  + scale\_fill\_manual() para asignar manualmente los colores para que "Missing" sea gris y "Other" sea beige

**CONSEJO:** Si desea invertir el orden de la leyenda solamente, añada este comando **ggplot2** guides(fill = guide\_legend(reverse = TRUE)).

### Líneas de cuadrícula verticales

Si traza con la configuración predeterminada **de la incidencia2**, puede observar que las líneas de cuadrícula verticales aparecen en cada etiqueta de fecha y una vez entre cada etiqueta de fecha. Esto puede dar lugar a que las líneas de cuadrícula se crucen con la parte superior de algunas barras.

Puedes eliminar todas las líneas de la cuadrícula añadiendo el comando **ggplot2** theme\_classic().

Ten en cuenta, sin embargo, que si utiliza semanas, los argumentos date\_breaks y date\_minor\_breaks sólo funcionan para las semanas del lunes. Si sus semanas son por otro día de la semana tendrá que proporcionar manualmente un vector de fechas a los argumentos breaks = y minor\_breaks = en su lugar. Consulte la sección de **ggplot2** para ver ejemplos de esto utilizando seq.Date().

### Incidencia acumulada

Puedes producir fácilmente un gráfico de incidencia acumulada pasando el objeto de incidencia al comando **incidence2** cumulate() y luego a plot(). Esto también funciona con facet\_plot().

Ver la sección más abajo en esta página para el método alternativo para trazar la incidencia acumulativa con **ggplot2** - por ejemplo para superponer una línea de incidencia acumulativa sobre una epicurva.

### Media móvil

Puedes añadir una media móvil a un gráfico de **incidencia2** fácilmente con add\_rolling\_average() del paquete **i2extras**. Pase su objeto incidence2 a esta función, y luego a plot(). Establezca before = como el número de días anteriores que desea incluir en la media móvil (por defecto es 2). Si sus datos están agrupados, la media móvil se calculará por grupo.

Para aprender a aplicar las medias móviles de forma más general sobre los datos, consulte la página del Manual sobre [medias móviles](#moving-averages-1).

## Epicurvas con ggplot2

El uso de ggplot() para construir su epicurva permite más flexibilidad y personalización, pero requiere más esfuerzo y comprensión de cómo funciona ggplot().

A diferencia de lo que ocurre con el paquete **incidence2**, hay que controlar manualmente la agregación de los casos por tiempo (en semanas, meses, etc.) y los intervalos de las etiquetas en el eje de fechas. Esto debe gestionarse cuidadosamente.

Estos ejemplos utilizan un subconjunto de los datos del listado: sólo los casos del Hospital Central.

Para producir una epicurva con ggplot() hay tres elementos principales:

* Un histograma, con los casos del listado agregados en "franjas" distinguidas por puntos de "ruptura" específicos
* Escalas para los ejes y sus etiquetas
* Temas para la apariencia de la trama, incluyendo títulos, etiquetas, subtítulos, etc.

### Especifique las ubicaciones de las cajas

Aquí mostramos cómo especificar cómo se agregarán los casos en los intervalos del histograma ("barras"). Es importante reconocer que la agregación de los casos en los intervalos del histograma **no son** necesariamente los mismos intervalos que las fechas que aparecerán en el eje x.

A continuación se muestra el código más sencillo para producir epicurvas diarias y semanales.

En el comando general ggplot() se proporciona el conjunto de datos a data =. Sobre esta base, se añade la geometría de un histograma con un +. Dentro del geom\_histograma(), mapeamos la estética de tal manera que la columna date\_onset se mapea al eje x. También dentro de geom\_histogram() pero no dentro de aes() establecemos el binwidth = de los bins del histograma, en días. Si esta sintaxis de **ggplot2** es confusa, revisa la página sobre [los fundamentos de ggplot](#ggplot-basics).

**PRECAUCIÓN: El** trazado de casos semanales mediante el uso de binwidth = 7 inicia el primer bin de 7 días en el primer caso, ¡que podría ser cualquier día de la semana! Para crear semanas específicas, véase la sección siguiente.

Observemos que el primer caso de este conjunto de datos del Hospital Central tuvo un inicio de síntomas el:

**Para especificar manualmente las pausas del histograma, no utiliza el argumento binwidth =, y en su lugar suministre un vector de fechas a las pausas =.**

Cree el vector de fechas con la función **base de** R seq.Date(). Esta función espera argumentos a =, desde = y por =. Por ejemplo, el comando siguiente devuelve fechas mensuales que comienzan en el 15 de enero y terminan en el 28 de junio.

Este vector puede proporcionarse a geom\_histogram() como saltos =:

Una simple secuencia de fechas semanales puede ser devuelta estableciendo por = "semana". Por ejemplo:

Una alternativa a la provisión de fechas específicas de inicio y fin es escribir un código dinámico para que los intervalos semanales comiencen el lunes anterior al primer caso. **Utilizaremos estos vectores de fechas a lo largo de los ejemplos siguientes.**

Descompongamos el código anterior, que es bastante desalentador:

* El valor "desde" (fecha más temprana de la secuencia) se crea de la siguiente manera: el valor de la fecha mínima (min() con na.rm=TRUE) en la columna date\_onset se introduce en floor\_date() del paquete **lubridate.** floor\_date() ajustado a "week" devuelve la fecha de inicio de la "semana" de esos casos, dado que el día de inicio de cada semana es un lunes (week\_start = 1).
* Asimismo, el valor "to" (fecha final de la secuencia) se crea utilizando la función inversa ceiling\_date() para devolver el lunes posterior al último caso.
* El argumento "por" de seq.Date() puede establecerse en cualquier número de días, semanas o meses.
* Utiliza week\_start = 7 para las semanas de domingo

Como vamos a utilizar estos vectores de fechas a lo largo de esta página, también definimos uno para todo el brote (el anterior es sólo para el Hospital Central).

Estas salidas de seq.Date() pueden utilizarse para crear los saltos de las casillas del histograma, pero también los saltos de las etiquetas de fecha, que pueden ser independientes de las casillas. Lea más sobre las etiquetas de fecha en secciones posteriores.

**SUGERENCIA:** Para un comando ggplot() más sencillo, guarde los saltos de cubo y los saltos de etiqueta de fecha como vectores con nombre por adelantado, y simplemente proporcione sus nombres a breaks =.

### Ejemplo de epicurva semanal

**A continuación se muestra un código de ejemplo detallado para producir epicurvas semanales para las semanas del lunes, con barras alineadas, etiquetas de fecha y líneas de cuadrícula verticales.** Esta sección es para el usuario que necesita el código rápidamente. Para entender cada aspecto (temas, etiquetas de fecha, etc.) en profundidad, continúe con las secciones siguientes. Es importante tener en cuenta:

* Las pausas del histograma se definen con seq.Date(), como se ha explicado anteriormente, para comenzar el lunes anterior al caso más antiguo y terminar el lunes posterior al último caso
* El intervalo de las etiquetas de fecha se especifica mediante date\_breaks = dentro de scale\_x\_date()
* El intervalo de líneas verticales menores entre etiquetas de fecha se especifica a date\_minor\_breaks =
* expand = c(0,0) en las escalas x e y elimina el exceso de espacio a cada lado de los ejes, lo que también asegura que las etiquetas de fecha comiencen desde la primera barra.

#### Semanas dominicales

Para conseguir el gráfico anterior para las semanas de los domingos son necesarias algunas modificaciones, ya que los date\_breaks = "weeks" sólo funcionan para las semanas de los lunes.

* Los puntos de ruptura de las franjas del histograma deben fijarse en los domingos (week\_start = 7)
* Dentro de scale\_x\_date(), los saltos de fecha similares deben proporcionarse a breaks = y minor\_breaks = para asegurar que las etiquetas de fecha y las líneas verticales de la cuadrícula se alineen los domingos.

Por ejemplo, el comando scale\_x\_date() para las semanas del domingo podría tener este aspecto:

### Agrupar/colorear por valor

Las barras del histograma pueden colorearse por grupos y "apilarse". Para designar la columna de agrupación, haga los siguientes cambios. Consulte la página de [fundamentos de ggplot](#ggplot-basics) para más detalles.

* Dentro del mapeo estético del histograma aes(), asigne el nombre de la columna a los argumentos group = y fill =
* Elimine cualquier argumento fill = fuera de aes(), ya que anulará el de dentro
* Los argumentos dentro de aes() se aplicarán por grupo, mientras que los de fuera se aplicarán a todas las barras (por ejemplo, es posible que quiera color = outside, para que cada barra tenga el mismo borde)

Este es el aspecto que tendría el comando aes() para agrupar y colorear las barras por género:

Aquí se aplica:

### Ajustar los colores

* Para establecer manualmente el relleno de cada grupo, utiliza scale\_fill\_manual() (nota: scale\_color\_manual() es diferente).
  + Utiliza el argumento values = para aplicar un vector de colores.
  + Utiliza na.value = para especificar un color para los valores NA.
  + Utiliza el argumento labels = para cambiar el texto de los elementos de la leyenda. Para estar seguro, proporcione como un vector con nombre como c("viejo" = "nuevo", "viejo" = "nuevo") o ajuste los valores en los propios datos.
  + Utiliza nombre = para dar un título adecuado a la leyenda
* Para obtener más consejos sobre las escalas y paletas de colores, consulte la página sobre [los fundamentos de ggplot](#ggplot-basics).

### Ajustar el orden de los niveles

El orden en que se apilan las barras agrupadas se ajusta mejor clasificando la columna de agrupación como tipo Factor. A continuación, puede designar el orden de los niveles de los factores (y sus etiquetas de visualización). Consulte la página sobre [Factores](#factors) o [los consejos de ggplot](#ggplot-tips) para obtener más detalles.

Antes de realizar el gráfico, utiliza la función fct\_relevel() del paquete **forcats** para convertir la columna de agrupación en de tipo factor y ajustar manualmente el orden de los niveles, como se detalla en la página sobre [Factores](#factors).

En el siguiente gráfico, las únicas diferencias con respecto al anterior es que la columna hospital se ha consolidado como en el caso anterior, y utilizamos guides() para invertir el orden de la leyenda, de modo que "Missing" se encuentra en la parte inferior de la leyenda.

**CONSEJO:** Para invertir el orden de la leyenda solamente, añada este comando **ggplot2**: guides(fill = guide\_legend(reverse = TRUE)).

### Ajustar la leyenda

Lea más sobre las leyendas y las escalas en la página de [consejos de ggplot](#ggplot-tips). Aquí hay algunos puntos destacados:

* Edite el título de la leyenda, ya sea en la función de escala o con labs(fill = "Legend title") (si está usando color = estético, entonces use labs(color = ""))
* theme(legend.title = element\_blank()) para no tener título de leyenda
* theme(legend.position = "top") ("bottom", "left", "right", o "none" para eliminar la leyenda)
* theme(legend.direction = "horizontal") leyenda horizontal
* guides(fill = guide\_legend(reverse = TRUE)) para invertir el orden de la leyenda

### Barras de lado a lado

La visualización lado a lado de las barras de grupo (en lugar de apiladas) se especifica dentro de geom\_histogram() con position = "dodge" colocado fuera de aes().

Si hay más de dos grupos de valores, éstos pueden resultar difíciles de leer. Considere la posibilidad de utilizar un gráfico facetado (múltiplos pequeños). Para mejorar la legibilidad en este ejemplo, se han eliminado los valores de género que faltan.

### Límites del eje

Hay dos maneras de limitar la extensión de los valores del eje.

Por lo general, la forma preferida es utilizar el comando coord\_cartesian(), que acepta xlim = c(min, max) y ylim = c(min, max) (donde proporcionas los valores mínimos y máximos). Esto actúa como un "zoom" sin eliminar realmente ningún dato, lo que es importante para las estadísticas y las medidas de resumen.

Alternativamente, puede establecer valores de fecha máximos y mínimos utilizando limits = c() dentro de scale\_x\_date(). Por ejemplo:

Asimismo, si desea que el eje x se extienda hasta una fecha concreta (por ejemplo, la fecha actual), aunque no se hayan notificado nuevos casos, puede utilizar

**PELIGRO:** Tenga cuidado al establecer los cortes o límites de la escala del eje Y (por ejemplo, de 0 a 30 por 5: seq(0, 30, 5)). Tales números estáticos pueden cortar su gráfica demasiado corto si los datos cambian para superar el límite!

### Etiquetas/cuadrículas del eje de la fecha

**SUGERENCIA:** Recuerde que **las etiquetas de** los ejes de fecha son independientes de la agregación de los datos en barras, pero visualmente puede ser importante alinear las franjas, las etiquetas de fecha y las líneas verticales de la cuadrícula.

Para **modificar las etiquetas de fecha y las líneas de la cuadrícula**, utiliza scale\_x\_date() de una de estas maneras:

* **Si los intervalos de su histograma son días, el lunes serán semanas, meses o años**:
  + Utiliza date\_breaks = para especificar el intervalo de las etiquetas y las líneas principales de la cuadrícula (por ejemplo, "día", "semana", "3 semanas", "mes" o "año")
  + Utiliza date\_minor\_breaks = para especificar el intervalo de las líneas verticales menores (entre las etiquetas de fecha)
  + Añade expand = c(0,0) para comenzar las etiquetas en la primera barra
  + Use date\_labels = para especificar el formato de las etiquetas de fecha - vea la página de Fechas para consejos (use \n para una nueva línea)
* **Si los bines de su histograma son semanas de domingo**:
  + Use breaks = y minor\_breaks = proporcionando una secuencia de saltos de fecha para cada
  + Puedes seguir utilizando date\_labels = y expand = para formatear como se ha descrito anteriormente

Algunas notas:

* Consulte la sección de apertura de ggplot para obtener instrucciones sobre cómo crear una secuencia de fechas utilizando seq.Date().
* Consulte [esta página](https://rdrr.io/r/base/strptime.html) o la página [Trabajar con fechas](#working-with-dates-1) para obtener consejos sobre la creación de etiquetas con fechas.

#### Demostraciones

A continuación se muestra una demostración de los gráficos en los que los intervalos y las etiquetas de los gráficos/líneas de la cuadrícula están alineados y no alineados:

### Datos agregados

A menudo, en lugar de un listado, se comienza con recuentos agregados de instalaciones, distritos, etc. Se puede hacer una epicurva con ggplot() pero el código será ligeramente diferente. Esta sección utilizará el conjunto de datos count\_data que fue importado anteriormente, en la sección de preparación de datos. Este conjunto de datos es linelist agregada a los recuentos de día-hospital. A continuación se muestran las primeras 50 filas.

#### Trazado de recuentos diarios

Podemos trazar una epicurva diaria a partir de estos recuentos diarios. Aquí están las diferencias con el código:

* Dentro del mapeo estético aes(), especifique y = como columna de recuento (en este caso, el nombre de la columna es n\_cases)
* Añadir el argumento stat = "identity" dentro de geom\_histogram(), que especifica que la altura de la barra debe ser el valor y =, no el número de filas como es el valor por defecto
* Añada el argumento ancho = para evitar las líneas blancas verticales entre las barras. Para los datos diarios, establezca el valor 1. Para los datos semanales, establezca 7. Para los datos de recuento mensual, las líneas blancas son un problema (cada mes tiene un número diferente de días) - considere la posibilidad de transformar su eje x en un factor categórico ordenado (meses) y utilizar geom\_col().

#### Trazado de recuentos semanales

Si sus datos ya son recuentos de casos por semana, podrían parecerse a este conjunto de datos (llamado count\_data\_weekly):

A continuación se muestran las primeras 50 filas de count\_data\_weekly. Puedes ver que los recuentos se han agregado en semanas. Cada semana se muestra por el primer día de la semana (lunes por defecto).

Ahora trace de manera que x = la columna epiweek. Recuerde añadir y = la columna de recuentos al mapeo estético, y añadir stat = "identity" como se ha explicado anteriormente.

### Medias móviles

Consulte la página sobre [medias móviles](#moving-averages-1) para obtener una descripción detallada y varias opciones. A continuación se muestra una opción para calcular medias móviles con el paquete **deslizante**. En este enfoque, la media móvil se calcula en el conjunto de datos antes de su trazado:

1. Agregue los datos en recuentos según sea necesario (diario, semanal, etc.) (véase la página de [Agrupar datos](#grouping-data))
2. Crea una nueva columna para contener la media móvil, creada con slide\_index() del paquete **slider**
3. Trazar la media móvil como una geom\_line() encima (después) del histograma de la epicurva

Vea la útil [viñeta](https://cran.r-project.org/web/packages/slider/vignettes/slider.html) en línea [para elpaquete de**deslizadores**](https://cran.r-project.org/web/packages/slider/vignettes/slider.html)

### Facetas/pequeños múltiplos

Al igual que con otros ggplots, puede crear gráficos facetados ("pequeños múltiplos"). Como se explica en la página de [consejos de ggplot](#ggplot-tips) de este manual, puede utilizar facet\_wrap() o facet\_grid(). Aquí demostramos con facet\_wrap(). Para las epicurvas, facet\_wrap() es típicamente más fácil, ya que es probable que sólo necesite facetar una columna.

La sintaxis general es facet\_wrap(rows ~ cols), donde a la izquierda de la tilde (~) está el nombre de una columna que se extiende a través de las "filas" del gráfico facetado, y a la derecha de la tilde está el nombre de una columna que se extiende a través de las "columnas" del gráfico facetado. Lo más sencillo es utilizar un nombre de columna, a la derecha de la tilde: facet\_wrap(~age\_cat).

**Ejes**   
libresTendrá que decidir si las escalas de los ejes para cada faceta son "fijas" a las mismas dimensiones (por defecto), o "libres" (lo que significa que cambiarán en función de los datos dentro de la faceta). Haga esto con el argumento scales = dentro de facet\_wrap() especificando "free\_x" o "free\_y", o "free".

**Número de coles y filas de las facetasSe**   
puede especificar con ncol = y nrow = dentro de facet\_wrap().

**Orden de los panelesPara**   
cambiar el orden de aparición, cambie el orden subyacente de los niveles de la columna de factores utilizada para crear las facetas.

**EstéticaEl**   
tamaño y la cara de **la fuente**  
, el color de la franja, etc. se pueden modificar a través de theme() con argumentos como

* strip.text = element\_text() (tamaño, color, cara, ángulo...)
* strip.background = element\_rect() (por ejemplo, element\_rect(fill="grey"))
* strip.position = (posición de la tira "abajo", "arriba", "izquierda" o "derecha")

**Etiquetas de bandaLas etiquetas de**   
los gráficos de facetas pueden modificarse a través de las "etiquetas" de la columna como factor, o mediante el uso de un "etiquetador".

Haga un etiquetador como este, usando la función as\_labeller() de **ggplot2**. A continuación, proporcione el etiquetador al argumento labeller = de facet\_wrap() como se muestra a continuación.

**Un ejemplo de gráfico facetado** - facetado por la columna age\_cat.

Consulte este [enlace](https://ggplot2.tidyverse.org/reference/labellers.html) para obtener más información sobre las etiquetadoras.

#### Epidemia total en el fondo de la faceta

Para mostrar la epidemia total en el fondo de cada faceta, añada la función gghighlight() con paréntesis vacíos al ggplot. Esto es del paquete **gghighlight**. Observe que el máximo del eje Y en todas las facetas se basa ahora en el pico de toda la epidemia. Hay más ejemplos de este paquete en la página [de consejos de ggplot](#ggplot-tips).

#### Una faceta con datos

Si quiere tener una caja de facetas que contenga todos los datos, duplique todo el conjunto de datos y trate los duplicados como un solo valor de facetas. Una función de "ayuda" CreateAllFacet() a continuación puede ayudar con esto (gracias a esta [entrada del blog](https://stackoverflow.com/questions/18933575/easily-add-an-all-facet-to-facet-wrap-in-ggplot2)). Cuando se ejecuta, el número de filas se duplica y habrá una nueva columna llamada facet en la que las filas duplicadas tendrán el valor "all", y las filas originales tienen el valor original de la columna facetada. Ahora sólo tienes que hacer la faceta en la columna de la faceta.

Aquí está la función de ayuda. Ejecútala para que esté disponible para ti.

Ahora aplique la función de ayuda al conjunto de datos, en la columna age\_cat:

Los cambios más importantes en el comando ggplot() son:

* Los datos utilizados son ahora central\_data2 (el doble de filas, con la nueva columna "facet")
* La etiquetadora tendrá que ser actualizada, si se utiliza
* Opcional: para conseguir facetas apiladas verticalmente: la columna de la faceta se mueve al lado de las filas de la ecuación y a la derecha se sustituye por "." (facet\_wrap(facet~.)), y ncol = 1. También puede ser necesario ajustar la anchura y la altura de la imagen png guardada (ver ggsave() en [los consejos de ggplot](#ggplot-tips)).

## Datos provisionales

Los datos más recientes que se muestran en las epicurvas deben marcarse a menudo como provisionales, o sujetos a retrasos en los informes. Esto puede hacerse añadiendo una línea vertical y/o un rectángulo sobre un número determinado de días. Aquí hay dos opciones:

1. Utiliza annotate():
   * Para una línea utiliza annotate(geom = "segment"). Proporcione x, xend, y, yend. Ajuste el tamaño, el tipo de línea (lty) y el color.
   * Para un rectángulo utiliza annotate(geom = "rect"). Proporcionar xmin/xmax/ymin/ymax. Ajuste el color y el alfa.
2. Agrupar los datos por estado tentativo y colorear esas barras de forma diferente

**ATENCIÓN:** Puedes intentar geom\_rect() para dibujar un rectángulo, pero el ajuste de la transparencia no funciona en un contexto de lista de líneas. Esta función superpone un rectángulo para cada observación/fila!. Utiliza un alfa muy bajo (por ejemplo, 0,01), u otro enfoque.

### Uso de annotate()

* Dentro de annotate(geom = "rect"), los argumentos xmin y xmax deben tener entradas del tipo Date.
* Ten en cuenta que, como estos datos se agregan en barras semanales, y la última barra se extiende hasta el lunes siguiente al último punto de datos, la región sombreada puede parecer que abarca 4 semanas
* Este es un [ejemplo de](https://ggplot2.tidyverse.org/reference/annotate.html) annotate() [en línea](https://ggplot2.tidyverse.org/reference/annotate.html)

La misma línea vertical negra se puede conseguir con el código de abajo, pero usando geom\_vline() se pierde la capacidad de controlar la altura:

### Color de las barras

Un enfoque alternativo podría ser ajustar el color o la visualización de las propias barras de datos tentativos. Podría crear una nueva columna en la etapa de preparación de los datos y utilizarla para agrupar los datos, de manera que el aes(fill = ) de los datos tentativos pueda tener un color o un alfa diferente al de las otras barras.

## Etiquetas de fecha de varios niveles

Si desea etiquetas de fecha de varios niveles (por ejemplo, mes y año) sin duplicar los niveles de etiquetas inferiores, considere uno de los enfoques siguientes:

Recuerde - puede utilizar herramientas como \n dentro de los argumentos date\_labels o labels para poner partes de cada etiqueta en una nueva línea inferior. Sin embargo, el código de abajo le ayuda a tomar años o meses (por ejemplo) en una línea inferior y sólo una vez. Algunas notas sobre el código de abajo:

* Los recuentos de casos se agregan en semanas por motivos estéticos. Véase la página de Epicurves (pestaña de datos agregados) para más detalles.
* Se utiliza una línea geom\_area() en lugar de un histograma, ya que el enfoque de facetas que se presenta a continuación no funciona bien con los histogramas.

**Agregue a los recuentos semanales**

**Hacer gráficos**

Las técnicas anteriores fueron adaptadas de [este](https://stackoverflow.com/questions/44616530/axis-labels-on-two-lines-with-nested-x-variables-year-below-months) y [este](https://stackoverflow.com/questions/20571306/multi-row-x-axis-labels-in-ggplot-line-chart) post en stackoverflow.com.

## Doble eje

Aunque hay fuertes discusiones sobre la validez de los ejes duales dentro de la comunidad de visualización de datos, muchos supervisores de epi todavía quieren ver una epicurva o un gráfico similar con un porcentaje superpuesto con un segundo eje. Esto se discute más ampliamente en la página de [consejos de ggplot](#ggplot-tips), pero a continuación se muestra un ejemplo utilizando el método **cowplot**:

* Se hacen dos gráficos distintos y luego se combinan con el paquete **cowplot**.
* Los gráficos deben tener exactamente el mismo eje x (límites establecidos) o de lo contrario los datos y las etiquetas no se alinearán
* Cada uno de ellos utiliza theme\_cowplot() y uno de ellos tiene el eje Y desplazado a la derecha del gráfico

Ahora utiliza **cowplot** para superponer los dos gráficos. Se ha prestado atención a la alineación del eje x, al lado del eje y y al uso de theme\_cowplot().

## Incidencia acumulada

Nota: Si utiliza **incidence2**, consulte la sección sobre cómo puede producir la incidencia acumulada con una función simple. Esta página abordará cómo calcular la incidencia acumulada y trazarla con ggplot().

Si se empieza con una lista de casos, cree una nueva columna que contenga el número acumulado de casos por día en un brote utilizando cumsum() de la **base** R:

A continuación se muestran las 10 primeras filas:

Esta columna acumulativa puede entonces ser trazada contra date\_onset, usando geom\_line():

También se puede superponer a la epicurva, con doble eje utilizando el método **cowplot** descrito anteriormente y en la página de [consejos de ggplot](#ggplot-tips):

Ahora utiliza **cowplot** para superponer los dos gráficos. Se ha prestado atención a la alineación del eje x, al lado del eje y y al uso de theme\_cowplot().

## Recursos

# #Pirámides demográficas y escalas de Likert

{#demographic-pyramids-and-likert-scales}

Las pirámides demográficas son útiles para mostrar distribuciones de edad y género. Se puede utilizar un código similar para visualizar los resultados de las preguntas de las encuestas tipo Likert (por ejemplo, "Muy de acuerdo", "De acuerdo", "Neutral", "En desacuerdo", "Muy en desacuerdo"). En esta página cubrimos lo siguiente:

* Pirámides rápidas y sencillas con el paquete apyramid
* Más pirámides personalizables con ggplot()
* Visualización de datos demográficos "de referencia" en el fondo de la pirámide
* Utilización de gráficos piramidales para mostrar otros tipos de datos (por ejemplo, respuestas a preguntas de encuestas **tipo Likert**)

## Preparación

### Cargar paquetes

Este trozo de código muestra la carga de los paquetes necesarios para los análisis. En este manual destacamos p\_load() de **pacman**, que instala el paquete si es necesario y lo carga para su uso. También puede cargar los paquetes instalados con library() de R **base.** Consulte la página sobre [los fundamentos de R](#r-basics) para obtener más información sobre los paquetes de R.

### Importar datos

Para empezar, importamos la lista de casos limpia de una epidemia de ébola simulada. Si quieres seguir el proceso, [clica aquí para descargar el listado "limpio"](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/case_linelists/linelist_cleaned.rds) (como archivo .rds). Importe los datos con la función import() del paquete **rio** (maneja muchos tipos de archivos como .xlsx, .csv, .rds - vea la página de [importación y exportación](#import-and-export) para más detalles).

A continuación se muestran las primeras 50 filas del listado.

### Limpieza

Para hacer una pirámide demográfica tradicional de edad/género, primero hay que limpiar los datos de la siguiente manera:

* La columna de género debe ser limpiada.
* Dependiendo de su método, la edad debe ser almacenada como un número o en una columna de categoría de edad.

Si se utilizan categorías de edad, los valores de las columnas deben corregirse ordenados, ya sea por defecto alfanumérico o intencionadamente al convertirlo en de tipo factor.

A continuación utilizamos tabyl() de **janitor** para inspeccionar las columnas gender y age\_cat5.

También realizamos un histograma rápido en la columna de edad para asegurarnos de que está limpia y correctamente clasificada:

## paquete ****apirámide****

El paquete **apyramid** es un producto del proyecto [R4Epis](https://r4epis.netlify.com/). Puedes leer más sobre este paquete [aquí](https://cran.r-project.org/web/packages/apyramid/vignettes/intro.html). Le permite hacer rápidamente una pirámide de edad. Para situaciones más matizadas, véase la sección sobre [el uso deggplot()](#demo_pyr_gg). Puedes leer más sobre el paquete **apyramid** en su página de ayuda introduciendo ?age\_pyramid en su consola de R.

### Datos del listado

Utilizando el conjunto de datos del listado limpiado, podemos crear una pirámide de edad con un simple comando age\_pyramid(). En este comando:

* El argumento data = se establece como el dataframe del listado
* El argumento age\_group = (para el eje Y) se establece con el nombre de la columna de edad categórica (entre comillas)
* El argumento split\_by = (para el eje x) se establece en la columna de género

La pirámide puede mostrarse con el porcentaje de todos los casos en el eje x, en lugar de los recuentos, incluyendo proporcional = TRUE.

Cuando se utiliza el paquete **agepyramid**, si la columna split\_by es binaria (por ejemplo, hombre/mujer, o sí/no), el resultado aparecerá como una pirámide. Sin embargo, si hay más de dos valores en la columna split\_by (sin incluir NA), la pirámide aparecerá como un gráfico de barras facetadas con barras grises en el "fondo" que indican el rango de los datos no facetados para ese grupo de edad. En este caso, los valores de split\_by = aparecerán como etiquetas en la parte superior de cada panel de facetas. Por ejemplo, a continuación se muestra lo que ocurre si a split\_by = se le asigna la columna hospital.

#### Valores faltantes

Las filas que tienen valores faltantes NA en las columnas split\_by = o age\_group =, si se codifican como NA, no activarán el faceting mostrado arriba. Por defecto, estas filas no se mostrarán. Sin embargo, puede especificar que aparezcan, en un gráfico de barras adyacente y como un grupo de edad separado en la parte superior, especificando na.rm = FALSE.

#### Proporciones, colores y estética

Por defecto, las barras muestran los recuentos (no el %), se muestra una línea media discontinua para cada grupo y los colores son verde/morado. Cada uno de estos parámetros puede ajustarse, como se muestra a continuación:

También puede añadir comandos adicionales de ggplot() al gráfico utilizando la sintaxis estándar de ggplot() "+", como temas estéticos y ajustes de etiquetas:

### Datos agregados

Los ejemplos anteriores suponen que sus datos están en formato de lista de líneas, con una fila por observación. Si sus datos ya están agregados en recuentos por categoría de edad, puede seguir utilizando el paquete **apyramid**, como se muestra a continuación.

Para la demostración, agregamos los datos del listado en recuentos por categoría de edad y género, en un formato "amplio". Esto simulará como si sus datos estuvieran en recuentos para empezar. Aprenda más sobre [Agrupar datos](#grouping-data) y [Pivotar datos](#pivoting-data) en sus respectivas páginas.

...lo que hace que el conjunto de datos tenga el siguiente aspecto: con columnas para la categoría de edad, y recuentos de hombres, recuentos de mujeres y recuentos faltantes.

Para configurar estos datos para la pirámide de edad, pivotaremos los datos para que sean "largos" con la función pivot\_longer() de **dplyr**. Esto se debe a que ggplot() generalmente prefiere datos "largos", y **apyramid** está utilizando ggplot().

A continuación, utiliza los argumentos split\_by = y count = de age\_pyramid() para especificar las respectivas columnas de los datos:

Observe en lo anterior, que el orden de los factores "m" y "f" es diferente (pirámide invertida). Para ajustar el orden debe redefinir el género en los datos agregados como un Factor y ordenar los niveles como se desee. Consulte la página de [Factores](#factors).

## ggplot()

El uso de ggplot() para construir su pirámide de edad permite más flexibilidad, pero requiere más esfuerzo y comprensión de cómo funciona ggplot(). También es más fácil cometer errores accidentalmente.

Para usar ggplot() para hacer pirámides demográficas, se crean dos gráficos de barras (uno para cada género), se convierten los valores de un gráfico en negativo y, finalmente, se invierten los ejes x e y para mostrar los gráficos de barras verticalmente, con sus bases encontrándose en el centro del gráfico.

### Preparación

Este enfoque utiliza la columna numérica de la edad, no la columna categórica de age\_cat5. Así que comprobaremos que el tipo de esta columna es efectivamente numérica.

Podría utilizar la misma lógica que se indica a continuación para construir una pirámide a partir de datos categóricos utilizando geom\_col() en lugar de geom\_histogram().

### Construcción de la trama

En primer lugar, hay que entender que para hacer una pirámide de este tipo utilizando ggplot() el planteamiento es el siguiente:

* Dentro de ggplot(), cree **dos** histogramas utilizando la columna numérica de la edad. Cree uno para cada uno de los dos valores de agrupación (en este caso los géneros masculino y femenino). Para ello, los datos para cada histograma se especifican dentro de sus respectivos comandos geom\_histogram(), con los respectivos filtros aplicados a linelist.
* Un gráfico tendrá valores de recuento positivos, mientras que el otro tendrá sus recuentos convertidos a valores negativos - esto crea la "pirámide" con el valor 0 en el centro del gráfico. Los valores negativos se crean utilizando un término especial **de ggplot2** ..count.. y multiplicando por -1.
* El comando coord\_flip() cambia los ejes X e Y, lo que hace que los gráficos se vuelvan verticales y se cree la pirámide.
* Por último, hay que modificar las etiquetas de los valores del eje de recuento para que aparezcan como recuentos "positivos" en ambos lados de la pirámide (a pesar de que los valores subyacentes en un lado sean negativos).

A continuación se muestra una versión **sencilla** de esto, utilizando geom\_histograma():

**PELIGRO:** Si los **límites** de su eje de recuentos son demasiado bajos, y una barra de recuentos los sobrepasa, la barra desaparecerá por completo o se acortará artificialmente. Tenga cuidado con esto si analiza datos que se actualizan de forma rutinaria. Evítelo haciendo que los límites del eje de recuentos se ajusten automáticamente a sus datos, como se indica a continuación.

Hay muchas cosas que puedes cambiar/añadir a esta sencilla versión, entre ellas:

* Ajuste automáticamente la escala del eje de recuentos a sus datos (evite los errores que se comentan en la advertencia que aparece a continuación)
* Especificar manualmente los colores y las etiquetas de las leyendas

**Convertir recuentos en porcentajes**

Para convertir los recuentos en porcentajes (del total), hágalo en sus datos antes de trazarlos. A continuación, obtenemos los recuentos de edad-género, luego desagrupamos(), y luego mutamos() para crear nuevas columnas de porcentajes. Si quiere porcentajes por género, omita el paso de desagrupación.

Es importante que guardemos los valores máximo y mínimo para saber cuáles deben ser los límites de la escala. Estos se utilizarán en el comando ggplot() a continuación.

Finalmente hacemos el ggplot() sobre los datos porcentuales. Especificamos scale\_y\_continuous() para extender las longitudes predefinidas en cada dirección (positiva y "negativa"). Usamos floor() y ceiling() para redondear los decimales en la dirección apropiada (abajo o arriba) para el lado del eje.

### Comparación con la línea de base

Con la flexibilidad de ggplot(), puede tener una segunda capa de barras en el fondo que represente la pirámide de población "verdadera" o "de referencia". Esto puede proporcionar una buena visualización para comparar lo observado con la línea de base.

Importe y visualice los datos de la población (véase la página de [descarga del manual y de los datos](#download-handbook-and-data)):

En primer lugar, algunos pasos de gestión de datos:

Aquí registramos el orden de las categorías de edad que queremos que aparezcan. Debido a algunas peculiaridades de la forma en que se implementa ggplot(), en este escenario específico es más fácil almacenar estos como un vector de caracteres y utilizarlos más tarde en la función de trazado.

Combinar los datos de la población y de los casos mediante la función **dplyr** bind\_rows():

* En primer lugar, asegúrese de que los nombres de las columnas, los valores de las categorías de edad y los valores del género son exactamente los mismos
* Haz que tengan la misma estructura de datos: columnas de categoría de edad, sexo, recuentos y porcentaje del total
* Agruparlas, una encima de la otra (bind\_rows())

Revisar el conjunto de datos de la población modificada

Ahora implemente lo mismo para la lista de casos. Ligeramente diferente porque comienza con las filas de casos, no con los recuentos.

Revisar el conjunto de datos de los casos modificados

Ahora los dos dataframes están combinados, uno encima del otro (tienen los mismos nombres de columna). Podemos "nombrar" cada uno de los dataframes, y utilizar el argumento .id = para crear una nueva columna "data\_source" que indicará de qué dataframe se originó cada fila. Podemos utilizar esta columna para filtrar en el ggplot().

Almacena los valores porcentuales máximo y mínimo, utilizados en la función de trazado para definir la extensión del gráfico (¡y no acortar ninguna barra!)

Ahora el gráfico se hace con ggplot():

* Un gráfico de barras de los datos de población (barras más anchas y transparentes)
* Un gráfico de barras de los datos del caso (barras pequeñas y más sólidas)

## Escala Likert

Las técnicas utilizadas para hacer una pirámide de población con ggplot() también se pueden utilizar para hacer gráficos de datos de encuestas en escala Likert.

Importe los datos (consulte la página de [descarga de manuales y datos](#download-handbook-and-data) si lo desea).

Empiece con datos que tengan este aspecto, con una clasificación categórica de cada encuestado (estado) y sus respuestas a 8 preguntas en una escala tipo Likert de 4 puntos ("Muy pobre", "Pobre", "Bueno", "Muy bueno").

En primer lugar, algunos pasos de gestión de datos:

* Pivotar los datos por más tiempo
* Crear una nueva dirección de columna en función de si la respuesta fue generalmente "positiva" o "negativa"
* Establezca el orden del nivel de factor para la columna de estado y la columna de respuesta
* Almacenar el valor de recuento máximo para que los límites del gráfico sean los adecuados

Ahora haga el gráfico. Como en las pirámides de edad anteriores, estamos creando dos gráficos de barras e invirtiendo los valores de uno de ellos a negativo.

Utilizamos geom\_bar() porque nuestros datos son una fila por observación, no recuentos agregados. Utilizamos el término especial **de ggplot2** ..count.. en uno de los gráficos de barras para invertir los valores en negativo (\*-1), y establecemos position = "stack" para que los valores se apilen unos encima de otros.

## Recursos

[documentación de la apirámide](https://cran.r-project.org/web/packages/apyramid/vignettes/intro.html)

# #Gráficos de calor

{#heat-plots}

Los gráficos de calor, también conocidos como "mapas de calor" o "mosaicos de calor", pueden ser visualizaciones útiles cuando se trata de mostrar 3 variables (eje x, eje y y relleno). A continuación mostramos dos ejemplos:

* Una matriz visual de eventos de transmisión por edad ("quién infectó a quién")
* Seguimiento de las métricas de información en muchas instalaciones/jurisdicciones a lo largo del tiempo

## Preparación

### Cargar paquetes

Este trozo de código muestra la carga de los paquetes necesarios para los análisis. En este manual destacamos p\_load() de **pacman**, que instala el paquete si es necesario y lo carga para su uso. También puede cargar los paquetes instalados con library() de R **base.** Consulte la página sobre [los fundamentos de R](#r-basics) para obtener más información sobre los paquetes de R.

**Conjuntos de datos**

Esta página utiliza la lista de casos de un brote simulado para la sección de la matriz de transmisión, y unos datos separado de recuentos diarios de casos de malaria por instalación para la sección de seguimiento de métricas. Se cargan y limpian en sus secciones individuales.

## Matriz de transmisión

Los mosaicos de calor pueden ser útiles para visualizar matrices. Un ejemplo es la visualización de "quién-infectó-quién" en un brote. Esto supone que se tiene información sobre los eventos de transmisión.

Ten en cuenta que la página de [Rastreo de contactos](#contact-tracing-1) contiene otro ejemplo de elaboración de una matriz de contactos de mosaico de calor, utilizando unos datos diferente (quizás más sencillo) en el que las edades de los casos y sus fuentes están perfectamente alineadas en la misma fila del dataframe. Estos mismos datos se utilizan para hacer un mapa de densidad en la página [de consejos ggplot](#ggplot-tips). Este ejemplo comienza a partir de una lista de casos y, por lo tanto, implica una considerable manipulación de los datos antes de lograr un dataframe ploteable. Así que hay muchos escenarios para elegir...

Partimos de la lista de casos de una epidemia de ébola simulada. Si quiere seguir el proceso, [clica para descargar la lista de casos "limpia"](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/case_linelists/linelist_cleaned.rds) (como archivo .rds). Importe sus datos con la función import() del paquete **rio** (acepta muchos tipos de archivos como .xlsx, .rds, .csv - vea la página de [importación y exportación](#import-and-export) para más detalles).

A continuación se muestran las primeras 50 filas del listado para su demostración:

En esta lista:

* Hay una fila por caso, identificada por case\_id
* Hay una columna posterior infector que contiene el case\_id del infector, que también es un caso en linelist

### Preparación de los datos

**Objetivo**: Necesitamos conseguir un dataframe de estilo "largo" que contenga una fila por cada posible ruta de transmisión entre edades, con una columna numérica que contenga la proporción de esa fila de todos los eventos de transmisión observados en linelist.

Esto requerirá varios pasos de manipulación de datos para lograrlo:

#### Hacer casos dataframe

Para empezar, creamos un dataframe de los casos, sus edades y sus infectadores - llamamos al dataframe case\_ages. Las primeras 50 filas se muestran a continuación.

#### Hacer un dataframe de infectores

A continuación, creamos un dataframe de los infectores, que por el momento consta de una sola columna. Se trata de las identificaciones de los infectores del listado. No todos los casos tienen un infector conocido, por lo que eliminamos los valores que faltan. A continuación se muestran las primeras 50 filas.

A continuación, utilizamos las uniones para obtener las edades de los infectores. Esto no es sencillo, ya que en linelist, las edades de los infectores no aparecen como tales. Conseguimos este resultado uniendo la lista de casos con los infectores. Comenzamos con los infectores, y left\_join() (añadimos) la lista de casos de tal manera que la columna de ID del infector del lado izquierdo del dataframe "base" se une a la columna de ID del caso en el dataframe de la lista de casos del lado derecho.

Así, los datos del registro de casos del infector en linelist (incluida la edad) se añaden a la fila del infector. A continuación se muestran las 50 primeras filas.

A continuación, combinamos los casos y sus edades con los infectores y sus edades. Cada uno de estos dataframes tiene la columna infector, por lo que se utiliza para la unión. Las primeras filas se muestran a continuación:

A continuación, una simple tabulación cruzada de los recuentos entre los grupos de edad de los casos y de los infectantes. Etiquetas añadidas para mayor claridad.

Podemos convertir esta tabla en un dataframe con data.frame() de la **base** R, que también la convierte automáticamente al formato "long", que es el deseado para el ggplot(). Las primeras filas se muestran a continuación.

Ahora hacemos lo mismo, pero aplicamos prop.table() de **base** R a la tabla para que en lugar de recuentos obtengamos proporciones del total. Las primeras 50 filas se muestran a continuación.

### Crear un gráfico de calor

Ahora, finalmente, podemos crear el gráfico de calor con el paquete **ggplot2**, utilizando la función geom\_tile(). Consulte la página de [consejos de ggplot](#ggplot-tips) para conocer más ampliamente las escalas de color/relleno, especialmente la función scale\_fill\_gradient().

* En la estética aes() de geom\_tile() establece la x y la y como la edad del caso y la edad del infector
* También en aes() establezca el argumento fill = en la columna Freq - este es el valor que se convertirá en un color de mosaico
* Establezca un color de escala con scale\_fill\_gradient() - puede especificar los colores altos/bajos
  + Ten en cuenta que scale\_color\_gradient() es diferente. En este caso quiere que el relleno
* Dado que el color se hace a través de "fill", puede utilizar el argumento fill = en labs() para cambiar el título de la leyenda

## Informar sobre las métricas a lo largo del tiempo

A menudo, en el ámbito de la salud pública, uno de los objetivos es evaluar las tendencias a lo largo del tiempo de muchas entidades (instalaciones, jurisdicciones, etc.). Una forma de visualizar esas tendencias a lo largo del tiempo es un gráfico de calor en el que el eje de abscisas es el tiempo y en el eje de ordenadas están las numerosas entidades.

### Preparación de los datos

Comenzamos importando unos datos de informes diarios sobre la malaria procedentes de muchos centros. Los informes contienen una fecha, una provincia, un distrito y el recuento de paludismo. Consulte la página de [descarga de manuales y datos](#download-handbook-and-data) para saber cómo descargar estos datos. A continuación se muestran las primeras 30 filas:

#### Agregar y resumir

**El objetivo de este ejemplo** es transformar los recuentos diarios del total de casos de malaria del centro (vistos en la pestaña anterior) en estadísticas resumidas semanales del rendimiento de los informes del centro, en este caso la proporción de días por semana en que el centro notificó algún dato. Para este ejemplo mostraremos los datos sólo para el **distrito de Spring**.

Para ello, realizaremos los siguientes pasos de gestión de datos:

1. Filtrar los datos según convenga (por lugar, fecha)
2. Crear una columna de semana utilizando floor\_date() del paquete **lubridate** 
   * Esta función devuelve la fecha de inicio de la semana de una fecha dada, utilizando una fecha de inicio especificada de cada semana (por ejemplo, "lunes")
3. Los datos se agrupan por columnas "ubicación" y "semana" para crear unidades de análisis de "instalación-semana"
4. La función summarise() crea nuevas columnas para reflejar las estadísticas resumidas por grupo de semanas de la instalación:
   * Número de días por semana (7 - un valor estático)
   * Número de informes recibidos de la semana de la instalación (¡podrían ser más de 7!)
   * Suma de los casos de paludismo notificados por el centro-semana (sólo por interés)
   * Número de días únicos en la semana de la instalación para los que hay datos reportados
   * **Porcentaje de los 7 días por instalación-semana para los que se comunicaron datos**
5. El dataframe se une con right\_join() a una lista exhaustiva de todas las posibles combinaciones de semanas de instalaciones, para que el conjunto de datos esté completo. La matriz de todas las combinaciones posibles se crea aplicando expand() a esas dos columnas del dataframe tal y como se encuentra en ese momento en la cadena de tuberías (representada por . ). Como se utiliza un right\_join(), se mantienen todas las filas del dataframe de expand() y se añaden a agg\_weeks si es necesario. Estas nuevas filas aparecen con valores resumidos NA (missing).

A continuación le mostramos el paso a paso:

Ahora el conjunto de datos tiene nrow(agg\_weeks) filas, cuando antes tenía nrow(facility\_count\_data).

A continuación creamos una columna de semana que refleje la fecha de inicio de la semana para cada registro. Esto se consigue con el paquete **lubridate** y la función floor\_date(), que se establece como "semana" y para que las semanas comiencen los lunes (día 1 de la semana - los domingos serían 7). Las filas superiores se muestran a continuación.

La columna de la nueva semana puede verse en el extremo derecho del dataframe

Ahora agrupamos los datos en semanas de instalaciones y los resumimos para producir estadísticas por semanas de instalaciones. Consulte la página sobre [tablas descriptivas](#descriptive-tables) para obtener consejos. La agrupación en sí misma no cambia el dataframe, pero afecta a la forma en que se calculan las estadísticas de resumen posteriores.

A continuación se muestran las filas superiores. Observe cómo las columnas han cambiado completamente para reflejar las estadísticas de resumen deseadas. Cada fila refleja una instalación-semana.

Por último, ejecutamos el siguiente comando para asegurarnos de que TODAS las semanas posibles de las instalaciones están presentes en los datos, incluso si antes no estaban.

Estamos utilizando un right\_join() sobre sí mismo (el conjunto de datos está representado por ".") pero habiéndose expandido para incluir todas las combinaciones posibles de las columnas semana y nombre\_localidad. Véase la documentación sobre la función expand() en la página sobre [Pivoting]. Antes de ejecutar este código, el conjunto de datos contiene nrow(agg\_weeks) filas.

Aquí está expanded\_weeks:

Antes de ejecutar este código, agg\_weeks contiene nrow(agg\_weeks) filas.

Después de ejecutar este código, agg\_weeks contiene nrow(agg\_weeks) filas.

### Crear un gráfico de calor

El ggplot() se realiza utilizando geom\_tile() del paquete **ggplot2**:

* Las semanas en el eje x se transforman en fechas, lo que permite utilizar scale\_x\_date()
* location\_name en el eje y mostrará todos los nombres de las instalaciones
* El relleno es p\_days\_reported, el rendimiento para ese establecimiento-semana (numérico)
* scale\_fill\_gradient() se utiliza en el relleno numérico, especificando los colores para el alto, el bajo y el NA
* scale\_x\_date() se utiliza en el eje x especificando las etiquetas cada 2 semanas y su formato
* Los temas de visualización y las etiquetas pueden ajustarse según sea necesario

### Básico

A continuación se produce un gráfico de calor básico, utilizando los colores, escalas, etc., por defecto. Como se ha explicado anteriormente, dentro de aes() para geom\_tile() debe proporcionar una columna del eje x, una columna del eje y **y** una columna para el relleno =. El relleno es el valor numérico que se presenta como color del mosaico.

### **G**ráfica limpia

Podemos hacer que este gráfico se vea mejor añadiendo funciones adicionales **de ggplot2**, como se muestra a continuación. Consulte la página de [consejos de ggplot](#ggplot-tips) para más detalles.

### Eje Y ordenado

Actualmente, las instalaciones están ordenadas "alfanuméricamente" de abajo a arriba. Si desea ajustar el orden de las instalaciones del eje Y, conviértalas en de tipo factor y proporcione el orden. Consulte la página sobre [Factores](#factors) para obtener consejos.

Como hay muchas instalaciones y no queremos escribirlas todas, intentaremos otro enfoque: ordenar las instalaciones en un dataframe y utilizar la columna de nombres resultante como orden de los niveles del factor. A continuación, la columna location\_name se convierte en un factor, y el orden de sus niveles se establece en función del número total de días de notificación presentados por el centro en todo el período de tiempo.

Para ello, creamos un dataframe que representa el número total de informes por instalación, ordenados de forma ascendente. Podemos utilizar este vector para ordenar los niveles del factor en el gráfico.

Véase el dataframe más abajo:

Ahora utiliza una columna del dataframe anterior (facility\_order$location\_name) para que sea el orden de los niveles del factor location\_name en el dataframe agg\_weeks:

Y ahora los datos se vuelven a trazar, con location\_name como factor ordenado:

### Mostrar valores

Puedes añadir una capa geom\_text() encima de los mosaicos, para mostrar los números reales de cada mosaico. Ten en cuenta que esto puede no parecer bonito si tiene muchos azulejos pequeños.

Se ha añadido el siguiente código: geom\_text(aes(label = p\_days\_reported)). Esto añade texto en cada mosaico. El texto que se muestra es el valor asignado al argumento label =, que en este caso se ha establecido en la misma columna numérica p\_days\_reported que también se utiliza para crear el gradiente de color.

## Recursos

[scale\_fill\_gradient()](https://ggplot2.tidyverse.org/reference/scale_gradient.html)

[Galería de gráficos R - mapa de calor](https://ggplot2.tidyverse.org/reference/scale_gradient.html)

# #Diagramas y gráficos

{#diagrams-and-charts}

Esta página cubre el código para producir:

* Diagramas de flujo utilizando **DiagrammeR** y el lenguaje DOT
* Diagramas aluviales/sanitarios
* Calendario de eventos

## Preparación

### Cargar paquetes

Este trozo de código muestra la carga de los paquetes necesarios para los análisis. En este manual destacamos p\_load() de **pacman**, que instala el paquete si es necesario y lo carga para su uso. También puede cargar los paquetes instalados con library() de R **base.** Consulte la página sobre [los fundamentos de R](#r-basics) para obtener más información sobre los paquetes de R.

### Importar datos

La mayor parte del contenido de esta página no requiere unos datos. Sin embargo, en la sección del diagrama de Sankey, utilizaremos la lista de casos de una epidemia de ébola simulada. Si quiere seguir esta parte, [clica para descargar linelist "limpia"](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/case_linelists/linelist_cleaned.rds) (como archivo .rds). Importe los datos con la función import() del paquete **rio** (maneja muchos tipos de archivos como .xlsx, .csv, .rds - vea la página de [importación y exportación](#import-and-export) para más detalles).

A continuación se muestran las primeras 50 filas del listado.

## Diagramas de flujo

Se puede utilizar el paquete R **DiagrammeR** para crear gráficos/gráficos de flujo. Pueden ser estáticos o ajustarse de forma dinámica en función de los cambios en unos datos.

**Herramientas**

La función grViz() se utiliza para crear un diagrama "Graphviz". Esta función acepta una cadena de caracteres de entrada que contiene las instrucciones para hacer el diagrama. Dentro de esa cadena, las instrucciones están escritas en un lenguaje diferente, llamado [DOT](https://graphviz.org/doc/info/lang.html) - es bastante fácil aprender lo básico.

**Estructura básica**

1. Abre las instrucciones grViz("
2. Especifique la dirección y el nombre del gráfico, y abra los paréntesis, por ejemplo, digraph my\_flow\_chart {
3. Enunciado del gráfico (diseño, dirección del rango)
4. Declaraciones de nodos (crear nodos)
5. Declaraciones de las aristas (da enlaces entre nodos)
6. Cierra las instrucciones }")

### Ejemplos sencillos

A continuación, dos sencillos ejemplos

Un ejemplo mínimo:

Un ejemplo con un contexto de salud pública quizás más aplicado:

### Sintaxis

**Sintaxis básica**

Los nombres de los nodos, o las declaraciones de las aristas, pueden separarse con espacios, punto y coma o nuevas líneas.

**Dirección del rango**

Se puede reorientar un gráfico para que se mueva de izquierda a derecha ajustando el argumento rankdir dentro de la sentencia del gráfico. El valor predeterminado es TB (de arriba a abajo), pero puede ser LR (de izquierda a derecha), RL o BT.

**Nombres de los nodos**

Los nombres de los nodos pueden ser palabras sueltas, como en el sencillo ejemplo anterior. Para utilizar nombres de varias palabras o caracteres especiales (por ejemplo, paréntesis, guiones), ponga el nombre del nodo entre comillas simples (' '). Puedes ser más fácil tener un nombre de nodo corto, y asignar una etiqueta, como se muestra a continuación entre corchetes [ ]. Si desea tener una nueva línea dentro del nombre del nodo, debe hacerlo a través de una etiqueta - utiliza \n en la etiqueta del nodo entre comillas simples, como se muestra a continuación.

**SubgruposEn**   
las declaraciones de aristas, se pueden crear subgrupos a ambos lados de la arista con corchetes ({ }). La arista se aplica entonces a todos los nodos en el corchete - es una forma abreviada.

**Diseños**

* dot (establecer rankdir en TB, LR, RL, BT, )
* neato
* twopi
* circo

**Nodos - atributos editables**

* label (texto, entre comillas simples si es de varias palabras)
* fillcolor (muchos colores posibles)
* color de la fuente
* alfa (transparencia 0-1)
* forma (elipse, óvalo, diamante, huevo, texto plano, punto, cuadrado, triángulo)
* estilo
* lados
* periferias
* tamaño fijo (h x w)
* altura
* ancho
* distorsión
* penwidth (ancho del borde de la forma)
* x (desplazamiento a la izquierda/derecha)
* y (desplazamiento arriba/abajo)
* nombre de la fuente
* tamaño de letra
* icono

**Bordes - atributos editables**

* tamaño de la flecha
* punta de flecha (normal, caja, cuervo, curva, diamante, punto, inv, ninguno, tee, vee)
* cola de flecha
* dir (dirección, )
* estilo (guiones, ...)
* color
* alfa
* puerta de entrada (texto delante de la punta de la flecha)
* tailport (texto detrás de la cola de flecha)
* nombre de la fuente
* tamaño de letra
* color de la fuente
* penwidth (anchura de la flecha)
* minlen (longitud mínima)

**Nombres de los colores**: valores hexadecimales o nombres de colores "X11", véase [aquí para los detalles de X11](http://rich-iannone.github.io/DiagrammeR/graphviz_and_mermaid.html)

### Ejemplos complejos

El siguiente ejemplo amplía el diagrama de vigilancia, añadiendo nombres de nodos complejos, aristas agrupadas, colores y estilos

DiagrammeR::grViz(" # Todas las instrucciones están dentro de una cadena de caracteres grandes

digraph surveillance\_diagram { # 'digraph' significa 'gráfico direccional', luego el nombre del gráfico

# declaración gráfica

#################

gráfico [layout = dot,

rankdir = TB, # diseño de arriba a abajo

fontsize = 10]

# nodos (círculos)

#################

nodo [forma = círculo, # forma = círculo

fixedsize = true

anchura = 1,3]

Primaria [label = 'Primary\nFacility']

Secundaria [label = 'Secondary\nFacility']

Terciario [label = 'Terciario\nInstalación']

SC [label = 'Surveillance\nCoordination',

fontcolor = darkgreen]

# bordes

#######

Primario -> Secundario [label = ' case transfer',

fontcolor = rojo,

color = rojo]

Secundario -> Terciario [label = ' case transfer',

fontcolor = rojo,

color = rojo]

# borde agrupado

{Primario Secundario Terciario} -> SC [etiqueta = 'notificación de casos',

fontcolor = darkgreen,

color = verde oscuro,

estilo = punteado]

}

")

**Agrupaciones de subgráficos**

Para agrupar los nodos en clústeres de cajas, póngalos dentro del mismo subgrafo con nombre (nombre del subgrafo {}). Para que cada subgrafo se identifique dentro de una caja delimitadora, comience el nombre del subgrafo con "cluster", como se muestra con las 4 cajas de abajo.

DiagrammeR::grViz(" # Todas las instrucciones están dentro de una cadena de caracteres grandes

digraph surveillance\_diagram { # 'digraph' significa 'gráfico direccional', luego el nombre del gráfico

# declaración gráfica

#################

gráfico [layout = dot,

rankdir = TB,

solapamiento = verdadero,

fontsize = 10]

# nodos (círculos)

#################

nodo [forma = círculo, # forma = círculo

fixedsize = true

anchura = 1,3] # anchura de los círculos

subgrafo cluster\_pasivo {

Primaria [label = 'Primary\nFacility']

Secundaria [label = 'Secondary\nFacility']

Terciario [label = 'Terciario\nInstalación']

SC [label = 'Surveillance\nCoordination',

fontcolor = darkgreen]

}

# nodos (cajas)

###############

nodo [forma = caja, # forma del nodo

fontname = Helvetica] # fuente de texto en el nodo

subgrafo cluster\_activo {

Activo [label = 'Active\nSurveillance']

HCF\_active [label = 'HCF\nBúsqueda activa']

}

subgrafo cluster\_EBD {

EBS [label = 'Vigilancia basada en eventos (EBS)']

Medios de comunicación social

Radio

}

subgrafo cluster\_CBS {

CBS [label = 'Vigilancia comunitaria (CBS)']

RECOs

}

# bordes

#######

{Primario Secundario Terciario} -> SC [label = 'case reporting']

Primario -> Secundario [etiqueta = 'transferencia de caso',

fontcolor = rojo]

Secundario -> Terciario [etiqueta = 'transferencia de casos',

fontcolor = rojo]

HCF\_active -> Activo

{'Social Media' Radio} -> EBS

RECOs -> CBS

}

")

**Formas de los nodos**

El siguiente ejemplo, tomado de [este tutorial](http://rich-iannone.github.io/DiagrammeR/), muestra las formas de los nodos aplicados y una abreviatura de las conexiones de los bordes en serie

### Salidas

Cómo manejar y guardar las salidas

* Las salidas aparecerán en el panel del Visor de RStudio, por defecto en la parte inferior derecha junto a Archivos, Gráficos, Paquetes y Ayuda.
* Para exportar puedes "Guardar como imagen" o "Copiar al portapapeles" desde el Visor. El gráfico se ajustará al tamaño especificado.

### Cifras parametrizadas

[Esta](https://mikeyharper.uk/flowcharts-in-r-using-diagrammer/) es una cita de este tutorial: <https://mikeyharper.uk/flowcharts-in-r-using-diagrammer/>

"Figuras parametrizadas": Una gran ventaja de diseñar figuras dentro de R es que podemos conectar las figuras directamente con nuestro análisis leyendo los valores de R directamente en nuestros diagramas de flujo. Por ejemplo, suponga que ha creado un proceso de filtrado que elimina valores después de cada etapa de un proceso, puede hacer que una figura muestre el número de valores que quedan en el conjunto de datos después de cada etapa de su proceso. Para hacer esto, puede utilizar el símbolo @@X directamente dentro de la figura, y luego hacer referencia a esto en el pie de página del gráfico utilizando [X]:, donde X es el índice numérico único".

Le animamos a revisar este tutorial si la parametrización es algo que le interesa.

## Diagramas de aluvión/sangría

### Cargar paquetes

Este trozo de código muestra la carga de los paquetes necesarios para los análisis. En este manual destacamos p\_load() de **pacman**, que instala el paquete si es necesario y lo carga para su uso. También puede cargar los paquetes instalados con library() de R **base.** Consulte la página sobre [los fundamentos de R](#r-basics) para obtener más información sobre los paquetes de R.

Cargamos el paquete **networkD3** para producir el diagrama, y también **tidyverse** para los pasos de preparación de datos.

### Trazado de los datos

Trazado de las conexiones en unos datos. A continuación demostramos el uso de este paquete en linelist del caso. Aquí hay un [tutorial en línea](https://www.r-graph-gallery.com/321-introduction-to-interactive-sankey-diagram-2.html).

Comenzamos obteniendo los recuentos de casos para cada combinación única de categoría de edad y hospital. Hemos eliminado los valores con categoría de edad ausente para mayor claridad. También reetiquetamos las columnas hospital y age\_cat como origen y destino respectivamente. Estos serán los dos lados del diagrama aluvial.

El conjunto de datos tiene ahora este aspecto:

Ahora creamos un dataframe de todos los nodos del diagrama, bajo el nombre de la columna. Esto consiste en todos los valores de hospital y age\_cat. Obsérvese que nos aseguramos de que todos son caracteres de tipo antes de combinarlos. y ajustamos las columnas ID para que sean números en lugar de etiquetas:

A continuación editamos el dataframe de los enlaces, que hemos creado anteriormente con count(). Añadimos dos columnas numéricas IDsource e IDtarget que reflejarán/crearán los enlaces entre los nodos. Estas columnas contendrán los números de ruta (posición) de los nodos de origen y destino. Se resta 1 para que estos números de posición comiencen en 0 (no en 1).

El conjunto de datos de enlaces tiene ahora este aspecto:

Ahora traza el diagrama Sankey con sankeyNetwork(). Puedes leer más sobre cada argumento ejecutando ?sankeyNetwork en la consola. Ten en cuenta que a menos que establezca iteraciones = 0 el orden de los nodos puede no ser el esperado.

Este es un ejemplo en el que también se incluye el resultado del paciente. Obsérvese que en el paso de preparación de los datos tenemos que calcular los recuentos de casos entre la edad y el hospital, y por separado entre el hospital y el resultado - y luego unir todos estos recuentos con bind\_rows().

<https://www.displayr.com/sankey-diagrams-r/>

## Calendario de eventos

Para hacer una línea de tiempo que muestre eventos específicos, puedes utilizar el paquete vistime.

Vea esta [viñeta](https://cran.r-project.org/web/packages/vistime/vignettes/vistime-vignette.html" \l "ex.-2-project-planning)

Este es el conjunto de datos de eventos con el que comenzamos:

## DAGs

Puedes construir un DAG manualmente utilizando el paquete **DiagammeR** y el lenguaje DOT como se ha descrito anteriormente.

Como alternativa, existen paquetes como **ggdag** y **daggity**

[Introducción a los DAGs ggdag vignette](https://cran.r-project.org/web/packages/ggdag/vignettes/intro-to-dags.html)

[Inferencia causal con dags en R](https://www.r-bloggers.com/2019/08/causal-inference-with-dags-in-r/" \l ":~:text=In a DAG all the,for drawing and analyzing DAGs.)

## Recursos

Gran parte de lo anterior sobre el lenguaje DOT está adaptado del tutorial [de este sitio](https://mikeyharper.uk/flowcharts-in-r-using-diagrammer/)

Otro [tutorial](http://rich-iannone.github.io/DiagrammeR/) más detallado [sobre DiagammeR](http://rich-iannone.github.io/DiagrammeR/)

Esta página sobre [los diagramas de Sankey](https://www.displayr.com/sankey-diagrams-r/)

# #Análisis de combinaciones

{#combinations-analysis}

Este análisis traza la frecuencia de diferentes **combinaciones** de valores/respuestas. En este ejemplo, se traza la frecuencia con la que los casos mostraron varias combinaciones de síntomas.

Este análisis también se suele llamar:

* **"Análisis de respuesta múltiple"**
* **"Análisis de conjuntos"**
* **"Análisis de combinaciones"**

En el ejemplo de gráfico anterior, se muestran cinco síntomas. Debajo de cada barra vertical hay una línea y puntos que indican la combinación de síntomas que refleja la barra de arriba. A la derecha, las barras horizontales reflejan la frecuencia de cada síntoma individual.

El primer método que mostramos utiliza el paquete **ggupset**, y el segundo utiliza el paquete **UpSetR**.

## Preparación

### Cargar paquetes

Este trozo de código muestra la carga de los paquetes necesarios para los análisis. En este manual destacamos p\_load() de **pacman**, que instala el paquete si es necesario y lo carga para su uso. También puede cargar los paquetes instalados con library() de R **base.** Consulte la página sobre [los fundamentos de R](#r-basics) para obtener más información sobre los paquetes de R.

### Importar datos

Para empezar, importamos la lista de casos limpia de una epidemia de ébola simulada. Si quieres seguir el proceso, clica aquí [para descargar el listado "limpio"](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/case_linelists/linelist_cleaned.rds) (como archivo .rds). Importe los datos con la función import() del paquete **rio** (maneja muchos tipos de archivos como .xlsx, .csv, .rds - vea la página de [importación y exportación](#import-and-export) para más detalles).

Esta lista de líneas incluye cinco variables "sí/no" sobre los síntomas declarados. Tendremos que transformar un poco estas variables para utilizar el paquete **ggupset** para hacer nuestro gráfico. Vea los datos (desplácese a la derecha para ver las variables de los síntomas).

### Reformular los valores

Para alinearse con el formato esperado por **ggupset**, convertimos el "sí" y el "no" en el nombre real del síntoma, utilizando case\_when() de **dplyr**. Si "no", establecemos el valor en blanco, por lo que los valores son eiter NA o el síntoma.

Ahora hacemos dos columnas finales:

1. Concatenar (pegar) todos los síntomas del paciente (una columna de caracteres)
2. Convierte la columna anterior en una de tipo lista, para que pueda ser aceptada por **ggupset** para hacer la trama

Consulte la página sobre [Caracteres y cadenas](#characters-and-strings) para saber más sobre la función unite() de **stringr**

Vea los nuevos datos. Observe las dos columnas del extremo derecho: los valores combinados pegados y la lista

## ****ggupset****

Cargar el paquete

Crear el gráfico. Comenzamos con un ggplot() y geom\_bar(), pero luego añadimos la función especial scale\_x\_upset() del **ggupset**.

Puedes encontrar más información sobre ggupset [en línea](https://rdrr.io/cran/ggupset/man/scale_x_upset.html) o fuera de línea en la documentación del paquete en su pestaña de Ayuda de RStudio ?ggupset.

## UpSetR

El paquete **UpSetR** permite una mayor personalización de la trama, pero puede ser más difícil de ejecutar:

**Paquete de carga**

**Limpieza de datos**

Debemos convertir los valores de los síntomas del listado en 1 / 0.

Ahora haga el gráfico usando la función personalizada upset() - usando sólo las columnas de síntomas. Debe designar qué "conjuntos" comparar (los nombres de las columnas de síntomas). Alternativamente, utiliza nsets = y order.by = "freq" para mostrar sólo las X combinaciones principales.

## Recursos

[La página de github de UpSetR](https://github.com/hms-dbmi/UpSetR)

[Una versión de Shiny App: puedes cargar tus propios datos](https://gehlenborglab.shinyapps.io/upsetr/)

[\*documentación - difícil de interpretar](https://cran.r-project.org/web/packages/UpSetR/UpSetR.pdf)

# #Cadenas de transmisión

{#transmission-chains}

## Resumen

La principal herramienta para manejar, analizar y visualizar las cadenas de transmisión y los datos de rastreo de contactos es el paquete **epicontacts**, desarrollado por la gente de RECON. Pruebe el gráfico interactivo que se muestra a continuación pasando el cursor por encima de los nodos para obtener más información, arrastrándolos para moverlos y clicando sobre ellos para resaltar los casos posteriores.

## Preparación

### Cargar paquetes

Primero cargue los paquetes estándar necesarios para la importación y manipulación de datos. En este manual destacamos p\_load() de **pacman**, que instala el paquete si es necesario y lo carga para su uso. También se pueden cargar paquetes con library() desde **el** R **base.** Consulte la página sobre [los fundamentos de R](#r-basics) para obtener más información sobre los paquetes de R.

Necesitará la versión de desarrollo de **epicontacts**, que puede instalarse desde github utilizando la función p\_install\_github() de **pacman**. Sólo necesita ejecutar este comando a continuación una vez, no cada vez que utiliza el paquete (a partir de entonces, puede utilizar p\_load() como de costumbre).

### Importar datos

Importamos el conjunto de datos de casos de una epidemia de ébola simulada. Si desea descargar los datos para seguirlos paso a paso, consulte las instrucciones en la página de [descarga de manuales y datos](#download-handbook-and-data). El conjunto de datos se importa utilizando la función import() del paquete **rio**. Consulte la página sobre [importación y exportación](#import-and-export) para conocer las distintas formas de importar datos.

A continuación se muestran las primeras 50 filas del listado. Son especialmente interesantes las columnas case\_id, generation, infector y source.

### Creación de un objeto epicontactos

A continuación, tenemos que crear un objeto **epicontactos**, que requiere dos tipos de datos:

* un listado que documenta los casos en los que las columnas son variables y las filas corresponden a casos únicos
* una lista de aristas que definen los vínculos entre los casos sobre la base de sus identificadores únicos (pueden ser contactos, eventos de transmisión, etc.)

Como ya tenemos un listado, sólo tenemos que crear una lista de aristas entre los casos, más concretamente entre sus ID. Podemos extraer los enlaces de transmisión del listado vinculando la columna infector con la columna case\_id. En este punto también podemos añadir "propiedades de borde", con lo que nos referimos a cualquier variable que describa el vínculo entre los dos casos, no los casos en sí. Por ejemplo, añadiremos una variable de localización que describa la ubicación del evento de transmisión, y una variable de duración que describa la duración del contacto en días.

En el código siguiente, la función **dplyr** transmutar es similar a mutar, excepto que sólo mantiene las columnas que hemos especificado dentro de la función. La función drop\_na filtrará cualquier fila en la que las columnas especificadas tengan un valor NA; en este caso, sólo queremos mantener las filas en las que se conoce el infector.

Ahora podemos crear el objeto **epicontactos** utilizando la función make\_epicontacts. Necesitamos especificar qué columna del listado apunta al identificador único del caso, así como qué columnas de los contactos apuntan a los identificadores únicos de los casos involucrados en cada enlace. Estos enlaces son direccionales en el sentido de que la infección va del infector al caso, por lo que necesitamos especificar los argumentos desde y hacia en consecuencia. Por lo tanto, también establecemos el argumento dirigido a TRUE, que afectará a las operaciones futuras.

Al examinar los objetos **epicontactos**, podemos ver que la columna case\_id del listado ha sido renombrada a id y las columnas case\_id e infector de los contactos han sido renombradas a from y to. Esto garantiza la coherencia en las operaciones posteriores de manipulación, visualización y análisis.

## Manejando

### Subconjunto

El método subset() para los objetos epicontactos permite, entre otras cosas, filtrar las redes en función de las propiedades del listado ("atributos de nodos") y de la base de datos de contactos ("atributos de aristas"). Estos valores deben pasarse como listas con nombre al argumento respectivo. Por ejemplo, en el código que sigue mantenemos en linelist sólo los casos masculinos que tienen una fecha de infección entre abril y julio de 2014 (las fechas se especifican como rangos), y los enlaces de transmisión que se produjeron en el hospital.

Podemos utilizar la función thin para filtrar linelist para incluir los casos que se encuentran en los contactos estableciendo el argumento what = "linelist", o filtrar los contactos para incluir los casos que se encuentran en linelist estableciendo el argumento what = "contacts". En el código siguiente, filtramos aún más el objeto epicontactos para mantener sólo los enlaces de transmisión que implican los casos masculinos infectados entre abril y julio que habíamos filtrado anteriormente. Podemos ver que sólo dos enlaces de transmisión conocidos se ajustan a esa especificación.

Además de la subdivisión por atributos de nodos y aristas, las redes pueden podarse para incluir sólo los componentes que están conectados a ciertos nodos. El argumento cluster\_id toma un vector de IDs de casos y devuelve linelist de individuos que están vinculados, directa o indirectamente, a esos IDs. En el código siguiente, podemos ver que un total de 13 casos del listado están involucrados en los clusters que contienen 2ae019 y 71577a.

El método subset() para los objetos epicontactos también permite filtrar por tamaño de cluster usando los argumentos cs, cs\_min y cs\_max. En el código siguiente, estamos manteniendo sólo los casos vinculados a clusters de 10 casos o más, y podemos ver que 271 casos del listado están involucrados en tales clusters.

### Acceso a las identificaciones

La función get\_id() recupera información sobre los ID de los casos en el conjunto de datos, y puede parametrizarse como sigue:

* lista de **líneas**: IDs en los datos del listado
* **contactos**: IDs en el conjunto de datos de los contactos ("desde" y "hasta" combinados)
* **from**: IDs en la columna "from" de los datos del contacto
* **a** los identificadores de la columna "a" de los datos de los contactos
* **todos**: Las identificaciones que aparecen en cualquier parte de cualquiera de los conjuntos de datos
* **común**: identificaciones que aparecen tanto en el conjunto de datos de contactos como en linelist

Por ejemplo, ¿cuáles son las diez primeras identificaciones de los datos de contactos?

¿Cuántas identificaciones se encuentran tanto en linelist como en los contactos?

## Visualización

### Trazado básico

Todas las visualizaciones de los objetos **epicontactos** son manejadas por la función plot. En primer lugar, filtraremos el objeto **epicontactos** para incluir solo los casos con fechas de inicio en junio de 2014 utilizando la función de subconjunto, y solo incluiremos los contactos vinculados a esos casos utilizando la función thin.

A continuación, podemos crear el gráfico básico e interactivo de la siguiente manera:

Puedes mover los nodos arrastrándolos, pasar por encima de ellos para obtener más información y clicar en ellos para resaltar los casos conectados.

Hay un gran número de argumentos para modificar este gráfico. Aquí cubriremos los principales, pero consulte la documentación a través de ?vis\_epicontacts (la función a la que se llama cuando se utiliza plot en un objeto **epicontacts**) para obtener una descripción completa de los argumentos de la función.

#### Visualización de los atributos de los nodos

El color del nodo, la forma del nodo y el tamaño del nodo se pueden asignar a una columna determinada en linelist utilizando los argumentos node\_color, node\_shape y node\_size. Esto es similar a la sintaxis aes que puede reconocer de **ggplot2**.

Los colores, formas y tamaños específicos de los nodos pueden especificarse de la siguiente manera:

* **Los colores** a través del argumento col\_pal, ya sea proporcionando una lista de nombres para la especificación manual de cada color como se hace a continuación, o proporcionando una función de paleta de colores como colorRampPalette(c("negro", "rojo", "naranja")), que proporcionaría un gradiente de colores entre los especificados.
* **Formas** pasando una lista con nombre al argumento shapes, especificando una forma para cada elemento único en la columna del listado especificada por el argumento node\_shape. Ver codeawesome para las formas disponibles.
* **Tamaño** pasando un rango de tamaño de los nodos al argumento size\_range.

Aquí un ejemplo, donde el color representa el resultado, la forma el género y el tamaño la edad:

#### Visualización de los atributos de los bordes

El color, la anchura y el tipo de línea de los bordes pueden asignarse a una columna determinada del dataframe de los contactos utilizando los argumentos edge\_color, edge\_width y edge\_linetype. Los colores y anchos específicos de los bordes se pueden especificar como sigue:

* **Colores** a través del argumento edge\_col\_pal, de la misma manera que se utiliza para col\_pal.
* **Anchos** pasando un rango de tamaño de los nodos al argumento width\_range.

Aquí un ejemplo:

### Eje temporal

También podemos visualizar la red a lo largo de un eje temporal asignando el argumento eje\_x a una columna del listado. En el ejemplo siguiente, el eje x representa la fecha de inicio de los síntomas. También hemos especificado el argumento arrow\_size para asegurarnos de que las flechas no son demasiado grandes, y hemos establecido label = FALSE para que la figura esté menos recargada.

Hay un gran número de argumentos adicionales para especificar aún más cómo se visualiza esta red a lo largo de un eje temporal, que puede comprobar a través de ?vis\_temporal\_interactive (la función que se llama cuando se utiliza plot en un objeto **epicontactos** con el eje x especificado). A continuación veremos algunos.

#### Especificación de la forma del árbol de transmisión

Hay dos formas principales que puede adoptar el árbol de transmisión, especificadas mediante el argumento network\_shape. La primera es una forma ramificada, como se muestra arriba, en la que un borde recto conecta dos nodos cualesquiera. Esta es la representación más intuitiva, pero puede dar lugar a la superposición de aristas en una red densamente conectada. La segunda forma es el rectángulo, que produce un árbol parecido a una filogenia. Por ejemplo:

A cada nodo del caso se le puede asignar una posición vertical única mediante el argumento position\_dodge. La posición de los casos no conectados (es decir, sin contactos reportados) se especifica utilizando el argumento unlinked\_pos.

La posición del nodo padre respecto a los nodos hijos puede especificarse mediante el argumento parent\_pos. La opción por defecto es colocar el nodo padre en el centro, sin embargo puede colocarse en la parte inferior (parent\_pos = 'bottom') o en la parte superior (parent\_pos = 'top').

#### Cómo guardar los gráficos y las cifras

Puedes guardar un gráfico como un archivo html interactivo y autónomo con la función visSave del paquete **VisNetwork**:

Guardar estas salidas de red como una imagen es desafortunadamente menos fácil y requiere que guardes el archivo como un html y luego tomes una captura de pantalla de este archivo usando el paquete webshot. En el código siguiente, estamos convirtiendo el archivo html guardado anteriormente en un PNG:

### Líneas de tiempo

También se pueden incluir líneas de tiempo en la red, que se representan en el eje de abscisas de cada caso. Esto puede servir para visualizar la ubicación de los casos, por ejemplo, o el tiempo hasta el resultado. Para generar una línea de tiempo, tenemos que crear un data.frame de al menos tres columnas que indiquen el ID del caso, la fecha de inicio del "evento" y la fecha de finalización del "evento". También se puede añadir cualquier número de otras columnas que luego se pueden asignar a las propiedades de los nodos y aristas de la línea de tiempo. En el código siguiente, generamos una línea de tiempo que va desde la fecha de inicio de los síntomas hasta la fecha del desenlace, y mantenemos las variables de desenlace y hospital que utilizamos para definir la forma y el color de los nodos. Ten en cuenta que puede tener más de una fila/evento de la línea de tiempo por caso, por ejemplo si un caso es transferido entre varios hospitales.

A continuación, pasamos el elemento de la línea de tiempo al argumento de la línea de tiempo. Podemos mapear los atributos de la línea de tiempo a los colores, formas y tamaños de los nodos de la línea de tiempo de la misma manera definida en las secciones anteriores, excepto que tenemos dos nodos: el nodo de inicio y el nodo final de cada línea de tiempo, que tienen argumentos separados. Por ejemplo, tl\_start\_node\_color define qué columna de la línea de tiempo se asigna al color del nodo inicial, mientras que tl\_end\_node\_shape define qué columna de la línea de tiempo se asigna a la forma del nodo final. También podemos asignar el color, la anchura, el tipo de línea y las etiquetas al borde de la línea de tiempo mediante los argumentos tl\_edge\_\*.

Consulte ?vis\_temporal\_interactive (la función a la que se llama cuando se traza un objeto epicontacto) para obtener documentación detallada sobre los argumentos. Cada argumento está anotado también en el código de abajo:

## Análisis

### Resumiendo

Podemos obtener una visión general de algunas de las propiedades de la red utilizando la función de resumen.

Por ejemplo, podemos ver que sólo el 57% de los contactos tienen ambos casos en linelist; esto significa que no tenemos datos del listado sobre un número significativo de casos involucrados en estas cadenas de transmisión.

### Características de los pares

La función get\_pairwise() permite procesar la(s) variable(s) del listado según cada par de los datos de contactos. En el siguiente ejemplo, la fecha de inicio de la enfermedad se extrae del listado para calcular la diferencia entre la fecha de inicio de la enfermedad para cada par. El valor que se obtiene de esta comparación representa el **intervalo de serie (si)**.

La función get\_pairwise() interpretará el tipo de la columna que se utiliza para la comparación, y ajustará su método de comparación de los valores en consecuencia. Para los números y las fechas (como en el ejemplo de **si**), la función restará los valores. Cuando se aplica a columnas que son caracteres o categóricas, get\_pairwise() pegará los valores. Dado que la función también permite un procesamiento arbitrario (véase el argumento "f"), estas combinaciones discretas pueden ser fácilmente tabuladas y analizadas.

En este caso, se observa una asociación significativa entre los vínculos de transmisión y el género.

### Identificación de grupos de empresas

La función get\_clusters() puede utilizarse para identificar componentes conectados en un objeto epicontacto. En primer lugar, la utilizamos para recuperar un data.frame que contenga la información de los clusters:

Veamos los clusters más grandes. Para ello, añadimos la información de los clústeres al objeto epicontactos y luego lo subconjuntamos para mantener sólo los clústeres más grandes:

### Cálculo de grados

El grado de un nodo corresponde a su número de aristas o conexiones con otros nodos. get\_degree() proporciona un método sencillo para calcular este valor para las redes de epicontactos. Un grado alto en este contexto indica un individuo que estuvo en contacto con muchos otros. El argumento type indica que queremos contar tanto el grado de entrada como el de salida, el argumento only\_linelist indica que sólo queremos calcular el grado para los casos del listado.

¿Qué personas son las que tienen más contactos?

¿Cuál es el número medio de contactos?

## Recursos

La [página de epicontacts](https://www.repidemicsconsortium.org/epicontacts/index.html) ofrece una visión general de las funciones del paquete e incluye algunas viñetas más detalladas.

La [página de github](http://github.com/reconhub/epicontacts) puede utilizarse para plantear problemas y solicitar funciones.

# #Árboles filogenéticos

{#phylogenetic-trees-1}

## Resumen

Los **árboles filogenéticos** se utilizan para visualizar y describir el parentesco y la evolución de los organismos a partir de la secuencia de su código genético.

Pueden construirse a partir de secuencias genéticas utilizando métodos basados en la distancia (como el método de unión de vecinos) o métodos basados en los caracteres (como el método de máxima verosimilitud y el método Bayesiano Markov Chain Monte Carlo). La secuenciación de nueva generación (NGS) se ha vuelto más asequible y se está utilizando cada vez más en la sanidad pública para describir los patógenos causantes de enfermedades infecciosas. Los dispositivos de secuenciación portátiles reducen el tiempo de respuesta y prometen hacer que los datos estén disponibles para apoyar la investigación de brotes en tiempo real. Los datos de NGS pueden utilizarse para identificar el origen o la fuente de una cepa de un brote y su propagación, así como para determinar la presencia de genes de resistencia a los antimicrobianos. Para visualizar el parentesco genético entre las muestras se construye un árbol filogenético.

En esta página aprenderemos a utilizar el paquete **ggtree**, que permite la visualización combinada de árboles filogenéticos con datos de muestra adicionales en forma de dataframe. Esto nos permitirá observar patrones y mejorar la comprensión de la dinámica de los brotes.

## Preparación

### Cargar paquetes

Este trozo de código muestra la carga de los paquetes necesarios. En este manual destacamos p\_load() de **pacman**, que instala el paquete si es necesario y lo carga para su uso. También puede cargar los paquetes instalados con library() de R **base.** Consulte la página sobre [los fundamentos de R](#r-basics) para obtener más información sobre los paquetes de R.

### Importar datos

Los datos de esta página pueden descargarse con las instrucciones de la página de [descarga de manuales y datos](#download-handbook-and-data).

Hay varios formatos diferentes en los que se puede almacenar un árbol filogenético (por ejemplo, Newick, NEXUS, Phylip). Uno de los más comunes es el formato de archivo Newick (.nwk), que es el estándar para representar árboles en forma legible por ordenador. Esto significa que un árbol completo puede expresarse en un formato de cadena como "((t2:0,04,t1:0,34):0,89,(t5:0,37,(t4:0,03,t3:0,67):0,9):0,59);", enumerando todos los nodos y puntas y su relación (longitud de rama) entre sí.

Nota: Es importante entender que el archivo del árbol filogenético en sí mismo no contiene datos de secuenciación, sino que es simplemente el resultado de las distancias genéticas entre las secuencias. Por lo tanto, no podemos extraer datos de secuenciación de un archivo de árbol.

En primer lugar, utilizamos la función read.tree() del paquete **ape** para importar un archivo de árbol filogenético de Newick en formato .txt, y lo almacenamos en un objeto tipo lista "phylo". Si es necesario, utiliza la función here() del paquete **here** para especificar la ruta relativa del archivo.

Nota: En este caso el árbol newick se guarda como un archivo .txt para facilitar su manejo y descarga desde Github.

tree <- ape::read.tree("Shigella\_tree.txt")

Inspeccionamos nuestro objeto árbol y vemos que contiene 299 puntas (o muestras) y 236 nodos.

En segundo lugar, importamos una tabla almacenada en un archivo .csv con información adicional para cada muestra secuenciada, como el sexo, el país de origen y los atributos de resistencia a los antimicrobianos, utilizando la función import() del paquete **rio**:

sample\_data <- import("sample\_data\_Shigella\_tree.csv")

A continuación se muestran las primeras 50 filas de los datos:

### Limpiar e inspeccionar

Limpiamos e inspeccionamos nuestros datos: Para asignar los datos de muestra correctos al árbol filogenético, los valores de la columna Sample\_ID en el dataframe sample\_data deben coincidir con los valores de tip.labels en el archivo del árbol:

Comprobamos el formato de los tip.labels en el archivo de árbol mirando las 6 primeras entradas usando con head() de R **base**.

También nos aseguramos de que la primera columna de nuestro dataframe sample\_data sea Sample\_ID. Miramos los nombres de las columnas de nuestro dataframe utilizando colnames() de R **base**.

Miramos los Sample\_IDs en el dataframe para asegurarnos de que el formato es el mismo que en el tip.label (por ejemplo, las letras son todas mayúsculas, no hay barras bajas adicionales \_ entre las letras y los números, etc.)

También podemos comparar si todas las muestras están presentes en el archivo de árbol y viceversa, generando un vector lógico de TRUE o FALSE donde coinciden o no. Estos no se imprimen aquí, para simplificar.

Podemos utilizar estos vectores para mostrar cualquier ID de muestra que no esté en el árbol (no hay ninguna).

Al inspeccionar podemos ver que el formato de Sample\_ID en el dataframe corresponde al formato de los nombres de las muestras en el tip.labels. No es necesario que estén clasificados en el mismo orden para que coincidan.

Estamos listos para ir!

## Visualización simple de un árbol

### Diferentes disposiciones de los árboles

**ggtree** ofrece muchos formatos de presentación diferentes y algunos pueden ser más adecuados para su propósito específico que otros. A continuación se muestran algunas demostraciones. Para otras opciones, consulte este [libro en línea](http://yulab-smu.top/treedata-book/chapter4.html).

A continuación, algunos ejemplos de disposición de árboles:

### Árbol simple con datos de muestra

El operador **%<+%** se utiliza para conectar el dataframe sample\_data con el archivo del árbol. La anotación más sencilla de su árbol es la adición de los nombres de las muestras en las puntas, así como la coloración de los puntos de las puntas y, si se desea, de las ramas:

Este es un ejemplo de árbol circular:

Puedes exportar el gráfico de árbol con ggsave() como lo harías con cualquier otro objeto ggplot. Escrito de esta manera, ggsave() guarda la última imagen producida en la ruta de archivo que especifiques. Recuerde que puede utilizar here() y rutas de archivo relativas para guardar fácilmente en subcarpetas, etc.

## Manipulación de árboles

A veces puede tener un árbol filogenético muy grande y sólo le interesa una parte del árbol. Por ejemplo, si ha producido un árbol que incluye muestras históricas o internacionales para obtener una gran visión general de dónde puede encajar su conjunto de datos en el panorama general. Pero luego, para ver más de cerca sus datos, quiere inspeccionar sólo esa parte del árbol más grande.

Dado que el archivo del árbol filogenético es sólo el resultado del análisis de los datos de secuenciación, no podemos manipular el orden de los nodos y las ramas en el propio archivo. Estos ya han sido determinados en análisis anteriores a partir de los datos NGS en bruto. Sin embargo, podemos ampliar partes, ocultar partes e incluso subconjuntar partes del árbol.

### Acercarse

Si no quieres "cortar" tu árbol, sino sólo inspeccionar una parte de él más de cerca, puedes acercarte para ver una parte específica.

En primer lugar, trazamos todo el árbol en formato lineal y añadimos etiquetas numéricas a cada nodo del árbol.

Para acercarse a una rama en particular (que sobresale a la derecha), utiliza viewClade() en el objeto ggtree p y proporcione el número de nodo para verlo más de cerca:

### Ramas colapsadas

Sin embargo, podemos querer ignorar esta rama y podemos colapsarla en ese mismo nodo (nodo 452) utilizando collapse(). Este árbol se define como p\_collapsed.

Para mayor claridad, cuando imprimimos p\_collapsed, añadimos un geom\_point2() (un diamante azul) en el nodo de la rama colapsada.

### Subconjunto de un árbol

Si queremos hacer un cambio más permanente y crear un nuevo árbol reducido con el que trabajar, podemos subconjuntar parte de él con tree\_subset(). Luego se puede guardar como un nuevo archivo de árbol o archivo .txt.

En primer lugar, inspeccionamos los nodos del árbol y las etiquetas de las puntas para decidir qué se va a subconjuntar.

Ahora, digamos que hemos decidido hacer un subconjunto del árbol en el nodo 528 (mantener sólo las puntas dentro de esta rama después del nodo 528) y lo guardamos como un nuevo objeto sub\_tree1:

Veamos el subconjunto del árbol 1:

También puede hacer un subconjunto basado en una muestra en particular, especificando cuántos nodos "hacia atrás" quiere incluir. Vamos a subconjuntar la misma parte del árbol basándonos en una muestra, en este caso S17BD07692, retrocediendo 9 nodos y lo guardamos como un nuevo objeto sub\_tree2:

Veamos el subconjunto del árbol 2:

También puede guardar su nuevo árbol como un tipo Newick o incluso un archivo de texto utilizando la función write.tree() del paquete **ape**:

### Rotación de nodos en un árbol

Como ya hemos dicho, no podemos cambiar el orden de las puntas o de los nodos en el árbol, ya que éste se basa en su parentesco genético y no está sujeto a manipulación visual. Pero podemos rotar las ramas alrededor de los nodos si eso facilita nuestra visualización.

En primer lugar, trazamos nuestro nuevo subconjunto de árbol 2 con las etiquetas de los nodos para elegir el nodo que queremos manipular y lo almacenamos en un objeto ggtree plot p.

Luego podemos manipular los nodos aplicando ggtree**::rotate()** o **ggtree::flip()**: Nota: para ilustrar qué nodos estamos manipulando aplicamos primero la función **geom\_hilight()** de **ggtree** para resaltar las muestras de los nodos que nos interesan y almacenamos ese objeto de trazado de ggtree en un nuevo objeto p1.

Ahora podemos rotar el nodo 37 en el objeto p1 para que las muestras del nodo 38 se muevan hacia arriba. Almacenamos el árbol rotado en un nuevo objeto p2.

O podemos usar el comando flip para rotar el nodo 36 en el objeto p1 y cambiar el nodo 37 a la parte superior y el nodo 39 a la parte inferior. Almacenamos el árbol volteado en un nuevo objeto p3.

### Ejemplo de subárbol con anotación de datos de muestra

Digamos que estamos investigando el grupo de casos con expansión clonal que se produjo en 2017 y 2018 en el nodo 39 de nuestro subárbol. Añadimos el año de aislamiento de la cepa, así como el historial de viajes y el color por país para ver el origen de otras cepas estrechamente relacionadas:

Nuestra observación apunta a un evento de importación de cepas procedentes de Asia, que luego circularon en Bélgica a lo largo de los años y parecen haber causado nuestro último brote.

## Árboles más complejos: añadir mapas térmicos de datos de muestra

Podemos añadir información más compleja, como la presencia categórica de genes de resistencia a los antimicrobianos y valores numéricos para la resistencia realmente medida a los antimicrobianos en forma de mapa de calor utilizando la función **ggtree::gheatmap()**.

Primero necesitamos trazar nuestro árbol (puede ser lineal o circular) y almacenarlo en un nuevo objeto ggtree plot p: Utilizaremos el subárbol de la parte 3).

En segundo lugar, preparamos nuestros datos. Para visualizar las diferentes variables con nuevos esquemas de color, subestablecimos nuestro dataframe a la variable deseada. Es importante añadir el Sample\_ID como rownames de lo contrario no puede coincidir los datos con el árbol tip.labels:

En nuestro ejemplo, queremos observar el género y las mutaciones que podrían conferir resistencia a la ciprofloxacina, un importante antibiótico de primera línea utilizado para tratar las infecciones por Shigella.

Creamos un dataframe para el género:

Creamos un dataframe para las mutaciones en el gen gyrA, que confieren resistencia a la ciprofloxacina:

Creamos un dataframe para la concentración inhibitoria mínima (CIM) medida para la Ciprofloxacina del laboratorio:

Creamos un primer gráfico añadiendo un mapa de calor binario para el género al árbol filogenético y almacenándolo en un nuevo objeto de gráfico ggtree h1:

A continuación, añadimos información sobre las mutaciones en el gen gyrA, que confieren resistencia a la ciprofloxacina:

Nota: La presencia de mutaciones puntuales cromosómicas en los datos de WGS se determinó previamente utilizando la herramienta PointFinder desarrollada por Zankari et al. (véase la referencia en la sección de referencias adicionales)

En primer lugar, asignamos un nuevo esquema de colores a nuestro objeto de trazado existente h1 y lo almacenamos en un objeto ahora h2. Esto nos permite definir y cambiar los colores para nuestra segunda variable en el mapa de calor.

A continuación, añadimos la segunda capa del mapa de calor a h2 y almacenamos los gráficos combinados en un nuevo objeto h3:

Repetimos el proceso anterior, añadiendo primero una nueva capa de escala de colores a nuestro objeto existente h3, y luego añadiendo los datos continuos sobre la concentración inhibitoria mínima (CIM) de Ciprofloxacina para cada cepa al objeto resultante h4 para producir el objeto final h5:

Podemos hacer el mismo ejercicio para un árbol lineal:

Primero añadimos el género:

A continuación, añadimos las mutaciones de resistencia a la ciprofloxacina después de añadir otra capa de colores:

A continuación, añadimos la concentración mínima inhibitoria determinada por el laboratorio (MIC):

## Recursos

[http://hydrodictyon.eeb.uconn.edu/eebedia/index.php/Ggtree#](http://hydrodictyon.eeb.uconn.edu/eebedia/index.php/Ggtree)  Clade\_Colors <https://bioconductor.riken.jp/packages/3.2/bioc/vignettes/ggtree/inst/doc/treeManipulation.html><https://guangchuangyu.github.io/ggtree-book/chapter-ggtree.html><https://bioconductor.riken.jp/packages/3.8/bioc/vignettes/ggtree/inst/doc/treeManipulation.html>

Ea Zankari, Rosa Allesøe, Katrine G Joensen, Lina M Cavaco, Ole Lund, Frank M Aarestrup, PointFinder: una novedosa herramienta web para la detección basada en WGS de la resistencia a los antimicrobianos asociada a mutaciones puntuales cromosómicas en patógenos bacterianos, Journal of Antimicrobial Chemotherapy, Volume 72, Issue 10, October 2017, Pages 2764-2768, <https://doi.org/10.1093/jac/dkx217>

# #Gráficos interactivos

{#interactive-plots}

Cada vez se exige más que la visualización de datos sea interrogativa para el público. Por ello, es cada vez más habitual crear gráficos interactivos. Hay varias formas de incluirlos, pero las dos más comunes son **plotly** y **shiny**.

En esta página nos centraremos en convertir un gráfico ggplot() existente en un gráfico interactivo con **plotly**. Puedes leer más sobre **shiny** en la página [Dashboards with Shiny](#dashboards-with-shiny). Lo que vale la pena mencionar es que los gráficos interactivos sólo se pueden utilizar en documentos R markdown en formato HTML, no en documentos PDF o Word.

A continuación se muestra una epicurva básica que se ha transformado para que sea interactiva utilizando la integración de **ggplot2** y **plotly** (pase el cursor por encima del gráfico, amplía la imagen o clica en los elementos de la leyenda).

## Preparación

### Cargar paquetes

Este trozo de código muestra la carga de los paquetes necesarios para los análisis. En este manual destacamos p\_load() de **pacman**, que instala el paquete si es necesario y lo carga para su uso. También puede cargar los paquetes instalados con library() de R **base.** Consulte la página sobre [los fundamentos de R](#r-basics) para obtener más información sobre los paquetes de R.

### Comienza con un ggplot()

En esta página asumimos que comienzas con un gráfico ggplot() que deseas convertir para que sea interactivo. Construiremos varios de estos gráficos en esta página, utilizando linelist de casos utilizada en muchas páginas de este manual.

### Importar datos

Para empezar, importamos la lista de casos limpia de una epidemia de ébola simulada. Si quieres seguir el proceso, clica aquí [para descargar el listado "limpio"](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/case_linelists/linelist_cleaned.rds) (como archivo .rds). Importe los datos con la función import() del paquete **rio** (maneja muchos tipos de archivos como .xlsx, .csv, .rds - vea la página de [importación y exportación](#import-and-export) para más detalles).

A continuación se muestran las primeras 50 filas del listado.

## Trazar con ggplotly()

La función ggplotly() del paquete **plotly** facilita la conversión de un ggplot() para que sea interactivo. Simplemente guarde su ggplot() y luego páselo a la función ggplotly().

A continuación, trazamos una línea simple que representa la proporción de casos que murieron en una semana determinada:

Comenzamos creando unos datos resumidos de cada semana epidemiológica y el porcentaje de casos con un resultado conocido que murieron.

Aquí están las primeras 50 filas de los datos weekly\_deaths.

Luego creamos el gráfico con **ggplot2**, utilizando geom\_line().

Podemos hacer esto interactivo simplemente pasando este gráfico a ggplotly(), como abajo. Pasa el cursor por encima de la línea para mostrar los valores x e y. Puedes ampliar el gráfico y arrastrarlo. También puede ver los iconos en la parte superior derecha de la gráfica. En orden, le permiten:

* Descargar la vista actual como imagen PNG
* Acercarse con un cuadro de selección
* "Pan", o moverse a través de la gráfica clicando y arrastrando la gráfica
* Acercar, alejar o volver al zoom por defecto
* Restablecer los ejes por defecto
* Activar/desactivar las "líneas de pico" que son líneas punteadas desde el punto interactivo que se extienden a los ejes x e y
* Ajustes para que los datos se muestren cuando no se está sobre la línea

Los datos agrupados también funcionan con ggplotly(). A continuación, se realiza una epicurva semanal agrupada por resultado. Las barras apiladas son interactivas. Prueba a clicar en los diferentes elementos de la leyenda (aparecerán/desaparecerán).

# Trazar interactivamente

p %>% plotly::ggplotly()

## Modificaciones

### Tamaño del archivo

Cuando se exporta en un HTML generado por R Markdown (¡como este libro!) se desea que el gráfico tenga el menor tamaño de datos posible (sin efectos secundarios negativos en la mayoría de los casos). Para ello, sólo hay que canalizar el gráfico interactivo a partial\_bundle(), también desde **plotly**.

### Botones

Algunos de los botones de un plotly estándar son superfluos y pueden distraer, por lo que puede eliminarlos. Puedes hacer esto simplemente canalizando la salida en config() de **plotly** y especificando qué botones eliminar. En el siguiente ejemplo especificamos por adelantado los nombres de los botones a eliminar, y los proporcionamos al argumento modeBarButtonsToRemove =. También establecemos displaylogo = FALSE para eliminar el logo de plotly.

## Baldosas térmicas

Puedes hacer que casi cualquier gráfico de ggplot() sea interactivo, incluidos los gráficos de calor. En la página sobre [gráficos de calor](#heat-plots) puede leer cómo hacer el siguiente gráfico, que muestra la proporción de días a la semana en que determinadas instalaciones comunicaron datos a su provincia.

Aquí está el código, aunque no lo describiremos en profundidad aquí.

A continuación, lo convertimos en interactivo y lo modificamos para que los botones sean sencillos y el tamaño del archivo.

–>

## Recursos

Plotly no es sólo para R, sino que también funciona bien con Python (y realmente con cualquier lenguaje de ciencia de datos, ya que está construido en JavaScript). Puedes leer más sobre él en el [sitio web de plotly](https://plotly.com/r/)

# Informes y Dashboards

# #Informes con R Markdown

{#reports-with-r-markdown}

R Markdown es una herramienta ampliamente utilizada para crear resultados automatizados, reproducibles y dignos de compartir, como los informes. Puedes generar resultados estáticos o interactivos, en Word, pdf, html, powerpoint y otros formatos.

Un script de R Markdown intercala el código R y el texto de tal manera que el script se convierte en su documento de salida. Puedes crear un documento completo con formato, incluyendo texto narrativo (puede ser dinámico para cambiar en función de sus datos), tablas, figuras, viñetas/números, bibliografías, etc.

Estos documentos pueden producirse para actualizarlos de forma rutinaria (por ejemplo, informes de vigilancia diarios) y/o ejecutarse sobre subconjuntos de datos (por ejemplo, informes para cada jurisdicción).

Otras páginas de este manual amplían este tema:

* La página [Organización de los informes de rutina](#organizing-routine-reports) muestra cómo rutinizar la producción de informes con carpetas autogeneradas con marca de tiempo.
* La página [Dashboardscon R Markdown](#dashboards-with-r-markdown) explica cómo formatear un informe de R Markdown como un cuadro de mando o tablero de control.

Cabe destacar que el proyecto [R4Epis](https://r4epis.netlify.app/) ha desarrollado plantillas de scripts R Markdown para los escenarios de brotes y encuestas más comunes que se encuentran en las ubicaciones de los proyectos de MSF.

## Preparación

**Antecedentes de R Markdown**

Explicar algunos de los conceptos y paquetes involucrados:

* **Markdown** es un "lenguaje" que permite escribir un documento en texto plano, que puede ser convertido a html y otros formatos. No es específico de R. Los archivos escritos en Markdown tienen una extensión '.md'.
* R **Markdown**: es una variación de markdown que es específica de R - le permite escribir un documento usando markdown para producir texto y para incrustar código R y mostrar sus resultados. Los archivos R Markdown tienen la extensión '.Rmd'.
* **rmarkdown - el paquete**: Esto es usado por R para convertir el archivo .Rmd en la salida deseada. Su objetivo es convertir la sintaxis markdown (texto), por lo que también necesitamos...
* **knitr**: Este paquete de R leerá los trozos de código, los ejecutará y los "tejerá" de nuevo en el documento. Así es como se incluyen las tablas y los gráficos junto al texto.
* **Pandoc**: Por último, pandoc convierte la salida en word/pdf/powerpoint, etc. Es un software independiente de R, pero se instala automáticamente con RStudio.

En resumen, el proceso que ocurre en segundo plano (¡no es necesario que conozcas todos estos pasos!) consiste en alimentar el archivo .Rmd a **knitr**, que ejecuta los trozos de código R y crea un nuevo archivo .md (markdown) que incluye el código R y su salida renderizada. El archivo .md es entonces procesado por pandoc para crear el producto final: un documento de Microsoft Word, un archivo HTML, un documento powerpoint, un pdf, etc.

(fuente: [https:](https://rmarkdown.rstudio.com/authoring_quick_tour.html)//rmarkdown.rstudio.com/authoring\_quick\_tour.html):

**Instalación**

Para crear una salida de R Markdown, necesita tener instalado lo siguiente:

* El paquete **rmarkdown** (**knitr** también se instalará automáticamente)
* Pandoc, que debería venir instalado con RStudio. Si no utiliza RStudio, puede descargar Pandoc aquí: [http:](http://pandoc.org/)//pandoc.org.
* Si quiere generar una salida en PDF (un poco más complicado), necesitará instalar LaTeX. Para los usuarios de R Markdown que no hayan instalado LaTeX antes, recomendamos que instalen TinyTeX [(](https://yihui.name/tinytex/)https://yihui.name/tinytex/). Puedes utilizar los siguientes comandos:

## Cómo empezar

### Instalar el paquete R rmarkdown

Instale el paquete R **rmarkdown**. En este manual destacamos p\_load() de **pacman**, que instala el paquete si es necesario y lo carga para su uso. También puede cargar los paquetes instalados con library() de R **base.** Consulte la página sobre [los fundamentos de R](#r-basics) para obtener más información sobre los paquetes de R.

### Iniciar un nuevo archivo Rmd

En RStudio, abra un nuevo archivo R markdown, comenzando con 'Archivo', luego 'Nuevo archivo' y luego 'R markdown...'.

R Studio le dará algunas opciones de salida para elegir. En el ejemplo siguiente seleccionamos "HTML" porque queremos crear un documento html. El título y los nombres de los autores no son importantes. Si el tipo de documento de salida que desea no es uno de estos, no se preocupe - puede elegir cualquiera y cambiarlo en el script más tarde.

Esto abrirá un nuevo script .Rmd.

### Es importante saber

**El directorio de trabajo**

El directorio de trabajo de un archivo markdown es el lugar donde se guarda el propio archivo Rmd. Por ejemplo, si el proyecto R está dentro de ~/Documents/projectX y el archivo Rmd en sí está en una subcarpeta ~/Documents/projectX/markdownfiles/markdown.Rmd, el código read.csv("data.csv") dentro del markdown buscará un archivo csv en la carpeta markdownfiles, y no en la carpeta raíz del proyecto donde los scripts dentro de los proyectos normalmente buscarían automáticamente.

Para referirse a los archivos en otro lugar, tendrá que utilizar la ruta completa del archivo o utilizar el paquete **here**. El paquete **here** establece el directorio de trabajo en la carpeta raíz del proyecto R y se explica en detalle en las páginas de [proyectos R](#r-projects) e [importación y exportación](#import-and-export) de este manual. Por ejemplo, para importar un archivo llamado "data.csv" desde la carpeta projectX, el código sería import(here("data.csv")).

Ten en cuenta que no se recomienda el uso de setwd() en los scripts de R Markdown - sólo se aplica al trozo de código en el que está escrito.

**Trabajar en una unidad frente a tu ordenador**

Debido a que R Markdown puede tener problemas con pandoc cuando se ejecuta en una unidad de red compartida, se recomienda que su carpeta esté en su máquina local, por ejemplo, en un proyecto dentro de 'Mis Documentos'. Si utilizas Git (¡muy recomendable!), esto te resultará familiar. Para más detalles, vea las páginas del manual sobre [R en unidades de red](#r-on-network-drives) y [Errores y ayuda].

## Componentes de R Markdown

Un documento R Markdown puede ser editado en RStudio igual que un script estándar de R. Cuando se inicia un nuevo script de R Markdown, RStudio intenta ser útil mostrando una plantilla que explica las diferentes secciones de un script de R Markdown.

Lo que aparece a continuación es lo que aparece al iniciar un nuevo script Rmd destinado a producir una salida html (según la sección anterior).

Como puede ver, hay tres componentes básicos en un archivo Rmd: YAML, texto Markdown y trozos de código R.

Estos crearán y se convertirán en la salida de su documento. Consulte el siguiente diagrama:

### Metadatos YAML

Denominado "metadatos YAML" o simplemente "YAML", se encuentra en la parte superior del documento R Markdown. Esta sección del script le dirá a su archivo Rmd qué tipo de salida producir, preferencias de formato y otros metadatos como el título del documento, el autor y la fecha. Hay otros usos que no se mencionan aquí (pero a los que se hace referencia en "Producción de una salida"). Ten en cuenta que la sangría es importante; los tabuladores no se aceptan, pero los espacios sí.

Esta sección debe comenzar con una línea que contenga sólo tres guiones --- y debe cerrar con una línea que contenga sólo tres guiones ---. Los parámetros YAML vienen en pares clave:valor. La colocación de los dos puntos en YAML es importante - los pares clave:valor están separados por dos puntos (¡no por signos de igualdad!).

El YAML debe comenzar con los metadatos del documento. El orden de estos parámetros YAML primarios (sin sangría) no importa. Por ejemplo:

título: "Mi documento"

autor: "Yo"

fecha: "2021-06-04"

Puedes utilizar código R en valores YAML escribiéndolo como código en línea (precedido por r entre comillas) pero también entre comillas (véase el ejemplo anterior para date:).

En la imagen de arriba, porque hemos clicado en que nuestra salida por defecto sea un archivo html, podemos ver que el YAML dice salida: html\_document. Sin embargo, también podemos cambiar esto para decir powerpoint\_presentation o word\_document o incluso pdf\_document.

### Texto

Esta es la narrativa de su documento, incluyendo los títulos y encabezados. Está escrito en el lenguaje "markdown", que se utiliza en muchos programas diferentes.

A continuación se presentan las formas principales de escribir este texto. Consulte una documentación más extensa disponible en R Markdown "cheatsheet" en el [sitio web de RStudio](https://rstudio.com/resources/cheatsheets/).

#### Nuevas líneas

Únicamente en R Markdown, para iniciar una nueva línea, introduzca \*dos espacios\*\* al final de la línea anterior y luego Enter/Return.

#### Caso

Rodee su texto normal con estos caracteres para cambiar su apariencia en la salida.

* Guiones bajos (\_texto\_) o asterisco simple (\*texto\*) para poner en cursiva
* Doble asterisco (\*\*texto\*\*) para el **texto en negrita**
* Tildes (texto) para mostrar el texto como código

El aspecto real de la fuente puede establecerse utilizando plantillas específicas (especificadas en los metadatos YAML; véase el ejemplo de las pestañas).

#### Color

No existe un mecanismo sencillo para cambiar el color del texto en R Markdown. Una solución, si su salida es un archivo HTML, es añadir una línea HTML en el texto de Markdown. El siguiente código HTML imprimirá una línea de texto en negrita roja.

<span style="color: rojo;"> \*\*\_DANGER:\_\*\* Esto es una advertencia. </span>

**PELIGRO:** Esto es una advertencia.

#### Títulos y encabezamientos

Un símbolo hash en una porción de texto de un script de R Markdown crea un encabezado. Esto es diferente que en un trozo de código R en el script, en el que un símbolo hash es un mecanismo para comentar/anotar/desactivar, como en un script normal de R.

Los distintos niveles de encabezamiento se establecen con diferentes números de símbolos de almohadilla al comienzo de una nueva línea. Un símbolo de almohadilla es un título o encabezamiento primario. Dos símbolos hash son un encabezamiento de segundo nivel. Los encabezamientos de tercer y cuarto nivel pueden hacerse con más símbolos hash sucesivamente.

# Encabezamiento / título de primer nivel

## Rúbrica de segundo nivel

### Rúbrica de tercer nivel

#### Viñetas y numeración

Utiliza asteriscos (\*) para crear una lista de viñetas. Termine la frase anterior, introduzca dos espacios, Enter/Return dos veces, y luego comience sus viñetas. Incluya un espacio entre el asterisco y el texto de su viñeta. Después de cada viñeta, introduzca dos espacios y luego Enter/Return. Las sub viñetas funcionan de la misma manera pero con sangría. Los números funcionan de la misma manera, pero en lugar de un asterisco, escriba 1), 2), etc. A continuación se muestra cómo podría ser el texto de tu script de R Markdown.

Aquí están mis viñetas (hay dos espacios después de los dos puntos):

\* Viñeta 1 (seguida de dos espacios y Enter/Return)

\* Viñeta 2 (seguida de dos espacios y Enter/Return)

\* Subbalanceo 1 (seguido de dos espacios y Enter/Return)

\* Subbalanceo 2 (seguido de dos espacios y Enter/Return)

#### Comentar el texto

Puedes "comentar" el texto de R Markdown del mismo modo que puede utilizar el "#" para comentar una línea de código R en un chunk de R. Simplemente resalta el texto y clica Ctrl+Mayús+c (Cmd+Mayús+c para Mac). El texto estará rodeado de flechas y se volverá verde. No aparecerá en su salida.

### Trozos de código

Las secciones del script que se dedican a ejecutar el código R se denominan "chunks". Aquí es donde se pueden cargar paquetes, importar datos y realizar la gestión y visualización de datos propiamente dicha. Puedes haber muchos "chunks" de código, por lo que pueden ayudarle a organizar su código R en partes, quizás intercaladas con texto. Para tener en cuenta: estos "trozos" parecerán tener un color de fondo ligeramente diferente al de la parte narrativa del documento.

Cada trozo se abre con una línea que comienza con tres marcas de retroceso y corchetes que contienen parámetros para el trozo ({ }). El chunk termina con otros tres back-ticks.

Puedes crear un nuevo fragmento escribiéndolo tú mismo, utilizando el atajo de teclado "Ctrl + Alt + i" (o Cmd + Shift + r en Mac), o clicando en el icono verde 'insertar un nuevo fragmento de código' en la parte superior de tu editor de scripts.

Algunas notas sobre el contenido de las llaves { }:

* Empiezan por "r" para indicar que el nombre del idioma dentro del chunk es R
* Después de la r puedes escribir opcionalmente un "nombre" de chunk - no es necesario pero puede ayudarte a organizar tu trabajo. Ten en cuenta que si nombras tus chunks, debes usar SIEMPRE nombres únicos o de lo contrario R se quejará cuando intentes renderizarlos.
* Las llaves pueden incluir también otras opciones, escritas como tag=value, como por ejemplo
* eval = FALSE para no ejecutar el código R
* echo = FALSE para no imprimir el código fuente de R del chunk en el documento de salida
* warning = FALSE para no imprimir las advertencias producidas por el código R
* message = FALSE para no imprimir ningún mensaje producido por el código R
* include = TRUE/FALSE si se incluyen los resultados de los trozos (por ejemplo, los gráficos) en el documento
* out.width = y out.height = - proporcionar en estilo out.width = "75%"
* fig.align = "center" ajustar cómo se alinea una figura en la página
* fig.show='hold' si tu chunk imprime múltiples figuras y quieres que se impriman una al lado de la otra (par con out.width = c("33%", "67%"). También puede establecer como fig.show='asis' para mostrarlas debajo del código que las genera, 'hide' para ocultarlas, o 'animate' para concatenar varias en una animación.
* La cabecera de un trozo debe escribirse en una sola línea
* Intente evitar los puntos, las barras bajas y los espacios. Utiliza guiones ( - ) en su lugar si necesita un separador.

Lea más extensamente sobre las opciones de **knitr** [aquí](https://yihui.org/knitr/options/).

Algunas de las opciones anteriores pueden configurarse con apuntar y clicar mediante los botones de configuración situados en la parte superior derecha del chunk. Aquí puedes especificar qué partes del chunk quieres que incluya el documento renderizado, es decir, el código, las salidas y las advertencias. Esto aparecerá como preferencias escritas dentro de los corchetes, por ejemplo, echo=FALSE si especifica que quiere 'Mostrar sólo la salida'.

También hay dos flechas en la parte superior derecha de cada trozo, que son útiles para ejecutar el código dentro de un trozo, o todo el código en trozos anteriores. Pasa el cursor por encima de ellas para ver lo que hacen.

Para que las opciones globales se apliquen a todos los chunks del script, puede configurar esto dentro de su primer chunk de código R en el script. Por ejemplo, para que sólo se muestren las salidas de cada trozo de código y no el propio código, puede incluir este comando en el trozo de código R:

#### Código R en el texto

También puede incluir un mínimo de código R dentro de los back-ticks. Dentro de los back-ticks, comience el código con "r" y un espacio, para que RStudio sepa que debe evaluar el código como código R. Vea el ejemplo siguiente.

El ejemplo siguiente muestra múltiples niveles de encabezamiento, viñetas, y utiliza el código R para la fecha actual (Sys.Date()) para evaluar en una fecha impresa.

El ejemplo anterior es sencillo (muestra la fecha actual), pero utilizando la misma sintaxis puede mostrar valores producidos por un código R más complejo (por ejemplo, para calcular el mínimo, la mediana o el máximo de una columna). También puede integrar objetos R o valores que fueron creados en trozos de código R anteriormente en el script.

Como ejemplo, el siguiente script calcula la proporción de casos que tienen menos de 18 años, utilizando funciones **tidyverse**, y crea los objetos less18, total y less18prop. Este valor dinámico se inserta en el texto posterior. Vemos cómo queda cuando se teje en un documento de Word.

### Imágenes

Puedes incluir imágenes en su R Markdown de dos maneras:

Si lo anterior no funciona, prueba a utilizar knitr::include\_graphics()

(recuerde que la ruta de su archivo puede ser escrita usando el paquete **aquí**)

### Tablas

Cree una tabla utilizando guiones ( - ) y barras ( | ). El número de guiones antes/entre las barras permite el número de espacios en la celda antes de que el texto comience a envolverse.

Columna 1 |Columna 2 |Columna 3

---------|----------|--------

Celda A |Celda B |Celda C

Celda D |Celda E |Celda F

El código anterior produce la siguiente tabla:

| **Columna 1** | **Columna 2** | **Columna 3** |
| --- | --- | --- |
| Celda A | Celda B | Celda C |
| Celda D | Celda E | Celda F |

### Secciones con pestañas

Para las salidas HTML, puede organizar las secciones en "pestañas". Basta con añadir .tabset en las llaves { } que se colocan después de un encabezamiento. Todos los subtítulos debajo de ese encabezado (hasta otro encabezado del mismo nivel) aparecerán como pestañas en las que el usuario puedes clicar. Lee más [aquí](https://bookdown.org/yihui/rmarkdown-cookbook/html-tabs.html)

Puedes añadir una opción adicional .tabset-pills después de .tabset para dar a las propias pestañas una apariencia "en forma de píldora". Ten en cuenta que al ver la salida HTML con pestañas, la funcionalidad de búsqueda Ctrl+f sólo buscará en las pestañas "activas", no en las ocultas.

## Estructura de los archivos

Hay varias maneras de estructurar su R Markdown y cualquier script de R asociado. Cada una tiene ventajas y desventajas:

* R Markdown autónomo: todo lo necesario para el informe se importa o se crea dentro de R Markdown
  + Fuente de otros archivos - Puedes ejecutar scripts R externos con el comando source() y utilizar sus salidas en el Rmd
  + Scripts hijos - un mecanismo alternativo para source()
* Utilizar un "archivo de ejecución" - Ejecutar comandos en un script de R antes de renderizar el R Markdown

### Rmd autónomo

Para un informe relativamente sencillo, puede optar por organizar tu script de R Markdown de manera que sea "autocontenido" y no implique ningún script externo.

Todo lo que necesita para ejecutar el R Markdown se importa o se crea dentro del archivo Rmd, incluyendo todos los trozos de código y la carga de paquetes. Este enfoque "autocontenido" es apropiado cuando no necesita hacer mucho procesamiento de datos (por ejemplo, trae un archivo de datos limpio o semilimpio) y la representación del R Markdown no tomará demasiado tiempo.

En este escenario, una organización lógica del script de R Markdown podría ser:

1. Establecer las opciones globales de **knitr**
2. Cargar paquetes
3. Importar datos
4. Datos del proceso
5. Producir resultados (tablas, gráficos, etc.)
6. Guarde los resultados, si procede (.csv, .png, etc.)

#### Fuente de otros archivos

Una variación del enfoque "autocontenido" es hacer que los trozos de código R Markdown "originen" (ejecuten) otros scripts R. Esto puede hacer que su script de R Markdown sea menos desordenado, más simple y más fácil de organizar. También puede ayudar si quiere mostrar las cifras finales al principio del informe. En este enfoque, el script de R Markdown final simplemente combina las salidas preprocesadas en un documento.

Una forma de hacerlo es proporcionando los scripts de R (ruta del archivo y nombre con extensión) al comando **base de** R source().

Ten en cuenta que cuando se utiliza source() dentro de R Markdown, los archivos externos se seguirán ejecutando durante el curso de la representación de su archivo Rmd. Por lo tanto, cada script se ejecuta cada vez que se renderiza el informe. Por lo tanto, tener estos comandos source() dentro del R Markdown no acelera el tiempo de ejecución, ni ayuda mucho a la depuración, ya que el error producido todavía se imprimirá al producir el R Markdown.

Una alternativa es utilizar la opción child = **knitr**. EXPLICAR MÁS PARA HACER

Debe ser consciente de los distintos entornos de R. Los objetos creados dentro de un entorno no estarán necesariamente disponibles para el entorno utilizado por R Markdown.

### Archivo de ejecución

Este enfoque implica utilizar el script de R que contiene el comando(s) render() para preprocesar los objetos que se introducen en el markdown de R.

Por ejemplo, puede cargar los paquetes, cargar y limpiar los datos, e incluso crear los gráficos de interés antes de render(). Estos pasos pueden ocurrir en el script de R, o en otros scripts que se originan. Siempre y cuando estos comandos ocurran en la misma sesión de RStudio y los objetos se guarden en el entorno, los objetos pueden ser llamados dentro del contenido de Rmd. Entonces el propio R markdown sólo se utilizará para el paso final - para producir la salida con todos los objetos pre-procesados. Esto es mucho más fácil de depurar si algo va mal.

Este enfoque es útil por las siguientes razones:

* Mensajes de error más informativos - estos mensajes serán generados por el script de R, no por el R Markdown. Los errores de R Markdown tienden a decirle qué trozo tuvo un problema, pero no le dirán qué línea.
* Si procede, puede ejecutar pasos de procesamiento largos antes del comando render() - se ejecutarán sólo una vez.

En el ejemplo siguiente, tenemos un script de R separado en el que preprocesamos un objeto de datos en el entorno de R y luego renderizamos el "create\_output.Rmd" usando render().

### Estructura de la carpeta

El flujo de trabajo también se refiere a la estructura general de las carpetas, como tener una carpeta de "salida" para los documentos y figuras creados, y carpetas de "datos" o "entradas" para los datos depurados. No entramos en más detalles aquí, pero echa un vistazo a la página de [organización de los informes de rutina](#organizing-routine-reports).

## Elaboración del documento

Puedes elaborar el documento de las siguientes maneras:

* Manualmente clicando sobre el botón "Knit" en la parte superior del editor de scripts de RStudio (rápido y fácil)
* Ejecuta el comando render() (ejecutado fuera del script de R Markdown)

### Opción 1: botón "Knit" (tejer)

Cuando tengas el archivo Rmd abierto, clica el icono/botón 'Knit' en la parte superior del archivo.

R Studio le mostrará el progreso dentro de una pestaña 'R Markdown' cerca de su consola R. El documento se abrirá automáticamente cuando esté completo.

El documento se guardará en la misma carpeta que su script de R markdown, y con el mismo nombre de archivo (aparte de la extensión). Obviamente, esto no es ideal para el control de versiones (se sobreescribirá cada vez que se haga un punto, a menos que se mueva manualmente), ya que entonces puede que tengas que renombrar el archivo (por ejemplo, añadir una fecha).

Este es el botón de acceso directo de RStudio para la función render() de **rmarkdown**. Este enfoque sólo es compatible con un R markdown autocontenido, donde todos los componentes necesarios existen o se originan dentro del archivo.

### Opción 2: comando render()

Otra forma de producir su salida de R Markdown es ejecutar la función render() (del paquete **rmarkdown**). Debe ejecutar este comando fuera del script de R Markdown, ya sea en un script de R separado (a menudo llamado "archivo de ejecución"), o como un comando independiente en la consola de R.

Al igual que con "knit", la configuración por defecto guardará la salida Rmd en la misma carpeta que el script Rmd, con el mismo nombre de archivo (aparte de la extensión del archivo). Por ejemplo, "mi\_informe.Rmd" cuando se teje creará "mi\_informe.docx" si se teje en un documento de Word. Sin embargo, al usar render() tiene la opción de usar diferentes configuraciones. render() puede aceptar argumentos que incluyen:

* output\_format = Este es el formato de salida al que se va a convertir (por ejemplo, "html\_document", "pdf\_document", "word\_document", o "all"). También puede especificar esto en el YAML dentro del script de R Markdown.
* output\_file = Este es el nombre del archivo de salida (y la ruta del archivo). Se puede crear a través de funciones de R como here() o str\_glue() como se demuestra a continuación.
* output\_dir = Este es un directorio de salida (carpeta) para guardar el archivo. Esto le permite elegir una alternativa distinta al directorio en el que se guarda el archivo Rmd.
* output\_options = Puedes proporcionar una lista de opciones que anulen las del YAML del script (por ejemplo )
* output\_yaml = Puedes proporcionar la ruta a un archivo .yml que contenga las especificaciones YAML
* params = Ver la sección de parámetros más abajo
* Vea la lista completa [aquí](https://pkgs.rstudio.com/rmarkdown/reference/render.html)

Como ejemplo, para mejorar el control de versiones, el siguiente comando guardará el archivo de salida dentro de una subcarpeta 'outputs', con la fecha actual en el nombre del archivo. Para crear el nombre del archivo, se utiliza la función str\_glue() del paquete **stringr** para "pegar" las cadenas estáticas (escritas sin formato) con el código dinámico de R (escrito entre llaves). Por ejemplo, si es 10 de abril de 2021, el nombre del archivo de abajo será "Informe\_2021-04-10.docx". Consulte la página sobre [Caracteres y cadenas](#characters-and-strings) para obtener más detalles sobre str\_glue().

A medida que el archivo se renderiza, la consola de RStudio le mostrará el progreso de la renderización hasta el 100%, y un mensaje final para indicar que la renderización se ha completado.

### Opción 3: paquete ****informativo****

El paquete de R **reportfactory** ofrece un método alternativo de organización y compilación de informes R Markdown para situaciones en las que se ejecutan informes de forma rutinaria (por ejemplo, diariamente, semanalmente...). Facilita la compilación de múltiples archivos R Markdown y la organización de sus resultados. En esencia, proporciona una "fábrica" desde la que se pueden ejecutar los informes R Markdown, obtener automáticamente carpetas con fecha y hora para los resultados, y tener un control de versiones "ligero".

Lea más sobre este flujo de trabajo en la página sobre la [organización de los informes de rutina](#organizing-routine-reports).

## Informes parametrizados

Puedes utilizar la parametrización para hacer que un informe sea dinámico, de forma que pueda ejecutarse con una configuración específica (por ejemplo, una fecha o lugar concretos o con determinadas opciones de tejido). A continuación, nos centramos en los aspectos básicos, pero hay más [detalles en línea](https://bookdown.org/yihui/rmarkdown/parameterized-reports.html) sobre los informes parametrizados.

Utilizando linelist del ébola como ejemplo, digamos que queremos ejecutar un informe de vigilancia estándar para cada hospital cada día. Mostramos cómo se puede hacer esto usando parámetros.

Importante: los informes dinámicos también son posibles sin la estructura formal de parámetros (sin *params:*), utilizando simples objetos R en un script adyacente de R. Esto se explica al final de esta sección.

### Ajuste de los parámetros

Tienes varias opciones para especificar los valores de los parámetros para tu salida de R Markdown.

#### Opción 1: Establecer parámetros dentro de YAML

Edite el YAML para incluir una opción params:, con declaraciones sangradas para cada parámetro que desee definir. En este ejemplo creamos los parámetros date y hospital, para los que especificamos valores. Estos valores están sujetos a cambios cada vez que se ejecuta el informe. Si utiliza el botón "Knit" para producir la salida, los parámetros tendrán estos valores por defecto. Del mismo modo, si utiliza render() los parámetros tendrán estos valores por defecto a menos que se especifique lo contrario en el comando render().

---

título: Informe de vigilancia

salida: documento\_html

params:

fecha: 2021-04-10

hospital: Hospital Central

---

En el fondo, estos valores de los parámetros están contenidos en una lista de sólo lectura llamada params. Así, puedes insertar los valores de los parámetros en el código de R como lo haría con otro objeto/valor de R en tu entorno. Simplemente escriba params$ seguido del nombre del parámetro. Por ejemplo params$hospital para representar el nombre del hospital ("Hospital Central" por defecto).

Ten en cuenta que los parámetros también pueden tener valores verdaderos o falsos, y por lo tanto estos pueden ser incluidos en sus opciones de **knitr** para un chunk de R. Por ejemplo, puedes establecer {r, eval=params$run} en lugar de {r, eval=FALSE}, y ahora si el chunk se ejecuta o no depende del valor de un parámetro run:.

Ten en cuenta que para los parámetros que son fechas, serán introducidos como una cadena. Por lo tanto, para que params$date se interprete en el código de R, es probable que tenga que ser envuelto con as.Date() o una función similar para convertir al tipo Date.

#### Opción 2: Establecer los parámetros dentro de render()

Como se ha mencionado anteriormente, como alternativa a clicar el botón "Knit" para producir la salida es ejecutar la función render() desde un script independiente. En este último caso, se pueden especificar los parámetros a utilizar en ese renderizado al argumento params = de render().

Ten en cuenta que los valores de los parámetros proporcionados aquí sobrescribirán sus valores por defecto si se escriben en el YAML. Escribimos los valores entre comillas ya que en este caso deben ser definidos como valores de carácter/cadena.

El siguiente comando renderiza "surveillance\_report.Rmd", especifica un nombre de archivo de salida dinámico y una carpeta, y proporciona una lista() de dos parámetros y sus valores al argumento params =.

#### Opción 3: Configurar los parámetros mediante una interfaz gráfica de usuario

Para obtener una sensación más interactiva, también puede utilizar la interfaz gráfica de usuario (GUI) para seleccionar manualmente los valores de los parámetros. Para ello, podemos clicar en el menú desplegable situado junto al botón "Tejer" y elegir "Tejer con parámetros".

Aparecerá una ventana emergente que le permitirá introducir los valores de los parámetros establecidos en el YAML del documento.

Puedes lograr lo mismo a través de un comando render() especificando params = "ask", como se demuestra a continuación.

Sin embargo, la introducción de valores en esta ventana emergente está sujeta a errores y faltas de ortografía. Es posible que prefiera añadir restricciones a los valores que se pueden introducir a través de los menús desplegables. Puedes hacerlo añadiendo en el YAML varias especificaciones para cada entrada params:.

* label: es como el título para ese menú desplegable en particular
* valor: es el valor por defecto (inicial)
* entrada: establecer la selección para el menú desplegable
* opciones: Indique los valores elegibles en el menú desplegable

A continuación, estas especificaciones están escritas para el parámetro hospitalario.

---

título: Informe de vigilancia

salida: documento\_html

params:

fecha: 2021-04-10

hospital:

etiqueta: "Ciudad:"

valor: Hospital Central

entrada: seleccionar

elecciones: [Hospital Central, Hospital Militar, Hospital del Puerto, Hospital de Maternidad de San Marcos (SMMH)]

---

Al tejer (ya sea a través del botón "tejer con parámetros" o mediante render()), la ventana emergente tendrá opciones desplegables para seleccionar.

### Ejemplo parametrizado

El siguiente código crea parámetros para la fecha y el hospital, que se utilizan en el R Markdown como params$date y params$hospital, respectivamente.

En la salida del informe resultante, vea cómo los datos se filtran al hospital específico, y el título del gráfico se refiere al hospital y a la fecha correctos. En este caso utilizamos el archivo "linelist\_cleaned.rds", pero sería especialmente adecuado que la propia lista de líneas tuviera también un sello de fecha para alinearse con la fecha parametrizada.

Si se teje esto se obtiene la salida final con la fuente y el diseño por defecto.

### Parametrización sin parámetros

Si está renderizando un archivo R Markdown con render() desde un script separado, puede realmente crear el impacto de la parametrización sin usar la funcionalidad params:.

Por ejemplo, en el script de R que contiene el comando render(), puedes simplemente definir hospital y fecha como dos objetos R (valores) antes del comando render(). En el R Markdown, no sería necesario tener una sección params: en el YAML, y nos referiríamos al objeto date en lugar de params$date y a hospital en lugar de params$hospital.

Seguir este enfoque significa que no se puede "tejer con parámetros", ni utilizar la interfaz gráfica de usuario, ni incluir opciones de tejido dentro de los parámetros. Sin embargo, permite simplificar el código, lo que puede ser ventajoso.

## Informes en bucle

Es posible que queramos ejecutar un informe varias veces, variando los parámetros de entrada, para producir un informe para cada jurisdicción/unidad. Esto puede hacerse utilizando herramientas para la iteración, que se explican en detalle en la página sobre [Iteración, bucles y listas](#iteration-loops-and-lists). Las opciones incluyen el paquete **purrr**, o el uso de un bucle for como se explica a continuación.

A continuación, utilizamos un simple bucle for para generar un informe de vigilancia para todos los hospitales de interés. Esto se hace con un solo comando (en lugar de cambiar manualmente el parámetro del hospital uno por uno). El comando para generar los informes debe existir en un script separado fuera del informe Rmd. Este script también contendrá objetos definidos para "hacer un bucle" - la fecha de hoy, y un vector de nombres de hospitales para hacer un bucle.

A continuación, introducimos estos valores uno a uno en el comando render() mediante un bucle, que ejecuta el comando una vez por cada valor del vector de hospitales. La letra i representa la posición del índice (del 1 al 4) del hospital que se está utilizando en esa iteración, de modo que hospital\_list[1] sería "Hospital Central". Esta información se suministra en dos lugares en el comando render():

1. Al nombre del archivo, de forma que el nombre del archivo de la primera iteración si se produce el 10 de abril de 2021 sería "Informe-Hospital Central\_2021-04-10.docx", guardado en la subcarpeta 'output' del directorio de trabajo.
2. A params = de manera que el Rmd utiliza el nombre del hospital internamente siempre que se llame al valor params$hospital (por ejemplo, para filtrar de los datos sólo al hospital concreto). En este ejemplo, se crearían cuatro archivos, uno por cada hospital.

## Plantillas

Utilizando un documento de plantilla que contenga cualquier formato deseado, puede ajustar la estética del aspecto de la salida Rmd. Puedes crear, por ejemplo, un archivo de MS Word o Powerpoint que contenga páginas/diapositivas con las dimensiones, marcas de agua, fondos y fuentes deseadas.

### Documentos de Word

Para crear una plantilla, inicie un nuevo documento de Word (o utiliza uno ya existente con el formato que le convenga), y edite las fuentes definiendo los Estilos. En el Estilo, los encabezados 1, 2 y 3 se refieren a los distintos niveles de encabezado de markdown (# Encabezado 1, ## Encabezado 2 y ### Encabezado 3 respectivamente). Clica con el botón derecho en el estilo y clica en "modificar" para cambiar el formato de la fuente, así como el párrafo (por ejemplo, puede introducir saltos de página antes de ciertos estilos que pueden ayudar con el espaciado). Otros aspectos del documento de Word, como los márgenes, el tamaño de la página, los encabezados, etc., pueden modificarse como un documento de Word normal en el que se trabaja directamente.

### Documentos Powerpoint

Como en el caso anterior, cree un nuevo conjunto de diapositivas o utiliza un archivo PowerPoint existente con el formato deseado. Para seguir editando, clica en "Ver" y "Patrón de diapositivas". Desde aquí puede cambiar la apariencia de la diapositiva "maestra" editando el formato del texto en los cuadros de texto, así como las dimensiones del fondo/página para la página en general.

Desgraciadamente, la edición de archivos PowerPoint es algo menos flexible:

* Una cabecera de primer nivel (# Cabecera 1) se convertirá automáticamente en el título de una nueva diapositiva,
* El texto de la Cabecera 2 no aparecerá como subtítulo, sino como texto dentro del cuadro de texto principal de la diapositiva (a menos que encuentre una manera de maniplar la vista del Patrón).
* Los gráficos y las tablas resultantes irán automáticamente a nuevas diapositivas. Tendrá que combinarlos, por ejemplo con la función patchwork para combinar ggplots, para que aparezcan en la misma página. Vea esta [entrada del blog](https://mattherman.info/blog/ppt-patchwork/) sobre el uso del paquete patchwork para poner múltiples imágenes en una diapositiva.

En el [paquete **oficial**](https://davidgohel.github.io/officer/) encontrará una herramienta para trabajar más a fondo con las presentaciones de PowerPoint.

### Integración de plantillas en el YAML

Una vez preparada la plantilla, el detalle de la misma puede añadirse en el YAML del Rmd debajo de la línea 'output' y debajo de donde se especifica el tipo de documento (que va a una línea separada en sí). Ten en cuenta que reference\_doc se puede utilizar para las plantillas de diapositivas de PowerPoint.

Lo más fácil es guardar la plantilla en la misma carpeta en la que está el archivo Rmd (como en el ejemplo siguiente), o en una subcarpeta dentro de ella.

---

título: Informe de vigilancia

de salida:

palabra\_documento:

reference\_docx: "plantilla.docx"

params:

fecha: 2021-04-10

hospital: Hospital Central

plantilla:

---

### Formateo de archivos HTML

Los archivos HTML no utilizan plantillas, pero pueden tener los estilos configurados dentro del YAML. Los HTML son documentos interactivos, y son particularmente flexibles. Aquí cubrimos algunas opciones básicas.

* Tabla de contenidos: Podemos añadir una tabla de contenidos con toc: true a continuación, y también especificar que permanezca visible ("flotante") al desplazarse, con toc\_float: true.
* Temas: Podemos referirnos a algunos temas prefabricados, que provienen de una biblioteca de temas de Bootswatch. En el siguiente ejemplo utilizamos cerulean. Otras opciones son: journal, flatly, darkly, readable, spacelab, united, cosmo, lumen, paper, sandstone, simplex y yeti.
* Resaltar: Configurando esto se cambia el aspecto del texto resaltado (por ejemplo, el código dentro de los trozos que se muestran). Los estilos soportados son default, tango, pygments, kate, monochrome, espresso, zenburn, haddock, breezedark y textmate.

He aquí un ejemplo de cómo integrar las opciones anteriores en el YAML.

---

título: "Ejemplo de HTML"

de salida:

html\_documento:

toc: true

toc\_float: true

tema: cerúleo

destacar: kate

---

A continuación se muestran dos ejemplos de salidas HTML que tienen ambas tablas de contenido flotantes, pero diferentes estilos de tema y resaltado seleccionados:

## Contenido dinámico

En una salida HTML, el contenido de tu informe puede ser dinámico. A continuación, algunos ejemplos:

### Tablas

En un informe HTML, se puede imprimir un dataframe/tablo de manera que el contenido sea dinámico, con filtros y barras de desplazamiento. Hay varios paquetes que ofrecen esta capacidad.

Para hacer esto con el paquete **DT**, como se utiliza a lo largo de este manual, se puede insertar un trozo de código como este:

La función datatable() imprimirá el dataframe proporcionado como una tabla dinámica para el lector. Puedes establecer rownames = FALSE para simplificar el extremo izquierdo de la tabla. filter = "top" proporciona un filtro sobre cada columna. En el argumento option() proporciona una lista de otras especificaciones. A continuación incluimos dos: pageLength = 5 establece el número de filas que aparecen como 5 (las filas restantes se pueden ver paginando a través de flechas), y scrollX=TRUE habilita una barra de desplazamiento en la parte inferior de la tabla (para las columnas que se extienden demasiado a la derecha).

Si tu conjunto de datos es muy grande, considere la posibilidad de mostrar sólo las X filas superiores envolviendo el dataframe en head().

### Widgets HTML

Los [widgets HTML para R](http://www.htmlwidgets.org/) son un tipo especial de paquetes de R que permiten una mayor interactividad utilizando bibliotecas de JavaScript. Puedes incrustarlos en salidas HTML R Markdown.

Algunos ejemplos comunes de estos widgets son:

* Plotly (utilizado en la página de este manual y en la página de [Plots interativos])
* visNetwork (utilizado en la página de [cadenas de transmisión de](#transmission-chains) este manual)
* Folleto (utilizado en la página de [fundamentos del SIG](#gis-basics) de este manual)
* dygraphs (útil para mostrar interactivamente los datos de las series temporales)
* DT (datatable()) (se utiliza para mostrar tablas dinámicas con filtro, ordenación, etc.)

La función ggplotly() de **plotly** es particularmente fácil de usar. Consulte la página de [gráficos interactivos](#interactive-plots-1).

## Recursos

Puedes encontrar más información en:

* <https://bookdown.org/yihui/rmarkdown/>
* <https://rmarkdown.rstudio.com/articles_intro.html>

Una buena explicación de markdown vs knitr vs Rmarkdown está aquí: <https://stackoverflow.com/questions/40563479/relationship-between-r-markdown-knitr-pandoc-and-bookdown>

# #Organización de informes rutinarios

{#organizing-routine-reports}

Esta página cubre el paquete **reportfactory**, que es un complemento para el uso de R Markdown para los informes.

En situaciones en las que se ejecutan reportes de forma rutinaria (diariamente, semanalmente, etc.), facilita la compilación de múltiples archivos R Markdown y la organización de sus resultados. En esencia, proporciona una "fábrica" desde la que se pueden ejecutar los informes R Markdown, obtener automáticamente carpetas con fecha y hora para los resultados, y tener un control de versiones "ligero".

**reportfactory** es uno de los paquetes desarrollados por RECON (R Epidemics Consortium). Aquí está su [sitio web](https://www.repidemicsconsortium.org/) y [Github](https://github.com/reconverse).

## Preparación

### Cargar paquetes

Desde RStudio, instale la última versión del paquete **reportfactory** desde Github.

Puedes hacerlo a través del paquete **pacman** con p\_load\_current\_gh() que forzará la instalación de la última versión desde Github. Proporcione la cadena de caracteres "reconverse/reportfactory", que especifica la organización de Github (reconverse) y el repositorio (reportfactory). También puede utilizar install\_github() del paquete **remotes**, como alternativa.

## Nueva fábrica

Para crear una nueva fábrica, ejecuta la función new\_factory(). Esto creará una nueva carpeta de proyecto R autocontenida. Por defecto:

* La fábrica se añadirá a tu directorio de trabajo
* El nombre del proyecto R de la fábrica se llamará "nueva\_fábrica.Rproj"
* Tu sesión de RStudio se "trasladará" a este proyecto R

Mirando dentro de la fábrica, se puede ver que las subcarpetas y algunos archivos se crearon automáticamente.

* La carpeta report\_sources contendrá sus scripts R Markdown, que generan sus informes
* La carpeta de resultados contendrá los resultados del informe (por ejemplo, HTML, Word, PDF, etc.)
* La carpeta de scripts puede utilizarse para almacenar otros scripts de R (por ejemplo, los que se originan en sus scripts de Rmd)
* La carpeta de datos puede utilizarse para guardar sus datos (se incluyen las subcarpetas "raw" y "clean")
* Un archivo .here, para que puedas utilizar el paquete **here** para llamar a los archivos de las subcarpetas por su relación con esta carpeta raíz (véase la página de [proyectos de R](#r-projects) para más detalles)
* Se ha creado un archivo gitignore en caso de que se vincule este proyecto R a un repositorio de Github (ver [Control de versiones y colaboración con Github])
* Un archivo README vacío, para si usas un repositorio de Github

**ATENCIÓN:** dependiendo de la configuración de tu ordenador, los archivos como ".here" pueden existir pero ser invisibles.

De los ajustes por defecto, a continuación hay varios que puede querer ajustar dentro del comando new\_factory():

* factory = - Proporcionar un nombre para la carpeta de fábrica (por defecto es "new\_factory")
* path = - Designa una ruta de archivo para la nueva fábrica (por defecto es el directorio de trabajo)
* report\_sources = Proporcione un nombre alternativo para la subcarpeta que contiene los scripts R Markdown (por defecto es "report\_sources")
* outputs = Proporcione un nombre alternativo para la carpeta que contiene los resultados del informe (por defecto es "outputs")

Véase ?new\_factory para una lista completa de los argumentos.

Cuando se crea la nueva fábrica, tu sesión de R se transfiere al nuevo proyecto R, por lo que debe cargar de nuevo el paquete **reportfactory**.

Ahora puede ejecutar el comando factory\_overview() para ver la estructura interna (todas las carpetas y archivos) de la fábrica.

El siguiente "árbol" de las carpetas y archivos de la fábrica se imprime en la consola de R. Observe que en la carpeta "data" hay subcarpetas para los datos "raw" y "clean", y datos CSV de ejemplo. También hay "example\_report.Rmd" en la carpeta "report\_sources".

## Crear un informe

Desde el proyecto R de fábrica, cree un informe R Markdown como lo haría normalmente, y guárdelo en la carpeta "report\_sources". Consulte la página de R Markdown para obtener instrucciones. A modo de ejemplo, hemos añadido lo siguiente a la fábrica:

* Un nuevo script de R markdown titulado "daily\_sitrep.Rmd", guardado dentro de la carpeta "report\_sources".
* Datos para el informe ("linelist\_cleaned.rds"), guardados en la subcarpeta "clean" dentro de la carpeta "data"

Podemos ver usando factory\_overview() nuestro R Markdown en la carpeta "report\_sources" y el archivo de datos en la carpeta de datos "clean" (resaltado):

A continuación se muestra una captura de pantalla del comienzo del R Markdown "daily\_sitrep.Rmd". Puedes ver que el formato de salida está configurado para ser HTML, a través de la salida de la cabecera YAML: html\_document.

En este sencillo script, hay comandos para:

* Cargar los paquetes necesarios
* Importe los datos del listado utilizando una ruta de archivo del paquete **aquí** (lea más en la página sobre [importación y exportación](#import-and-export))
* Imprime una tabla resumen de los casos, y la exporta con export() como un archivo .csv
* Imprimir una epicurva, y exportarla con ggsave() como un archivo .png

Puedes revisar sólo la lista de informes R Markdown en la carpeta "report\_sources" con este comando:

## Compilar

En una fábrica de informes, "compilar" un informe R Markdown significa que se ejecutará el script .Rmd y se producirá la salida (como se especifica en el script YAML, por ejemplo, como HTML, Word, PDF, etc.).

La fábrica creará automáticamente una carpeta con fecha y hora para las salidas en la carpeta "outputs".

El propio informe y cualquier archivo exportado producido por el script (por ejemplo, csv, png, xlsx) se guardarán en esta carpeta. Además, el propio script Rmd se guardará en esta carpeta, para que tenga un registro de esa versión del script.

Esto contrasta con el comportamiento normal de un R Markdown "tejido", que guarda las salidas en la ubicación del script Rmd. Este comportamiento por defecto puede resultar en carpetas abarrotadas y desordenadas. El objetivo de la fábrica es mejorar la organización cuando uno necesita ejecutar informes con frecuencia.

### Compilar por nombre

Puedes compilar un informe específico ejecutando compile\_reports() y proporcionando el nombre del script Rmd (sin la extensión .Rmd) a reports =. Para simplificar, puede omitir reports = y simplemente escribir el nombre R Markdown entre comillas, como se indica a continuación.

Este comando compilaría sólo el informe "daily\_sitrep.Rmd", guardando el informe HTML, y las exportaciones de la tabla .csv y la epicurva .png en una subcarpeta con fecha y hora específicas del informe, dentro de la carpeta "outputs".

Ten en cuenta que si elige proporcionar la extensión .Rmd, debe escribir correctamente la extensión tal y como se guarda en el nombre del archivo (.rmd vs. .Rmd).

Ten en cuenta también que, al compilar, es posible que aparezcan temporalmente varios archivos en la carpeta "report\_sources", pero pronto desaparecerán al ser transferidos a la carpeta "outputs" correcta.

### Compilación por número

También puede especificar el script Rmd a compilar proporcionando un número o vector de números a reports =. Los números deben alinearse con el orden en que aparecen los informes cuando se ejecuta list\_reports().

### Compilar todos los

Puedes compilar todos los informes R Markdown en la carpeta "report\_sources" estableciendo el argumento reports = a TRUE.

### Compilar desde la subcarpeta

Puedes añadir subcarpetas a la carpeta "report\_sources". Para ejecutar un informe R Markdown desde una subcarpeta, simplemente proporcione el nombre de la carpeta a subcarpeta =. A continuación se muestra un ejemplo de código para compilar un informe Rmd que vive en una subcarpeta de "report\_sources".

Puedes compilar todos los informes Rmd dentro de una subcarpeta proporcionando el nombre de la subcarpeta a reports =, con una barra al final, como se indica a continuación.

### Parametrización

Como se indicó en la página sobre [Informes con R Markdown](#reports-with-r-markdown), puede ejecutar informes con parámetros especificados. Puedes pasar estos parámetros como una lista a compile\_reports() a través del argumento params =. Por ejemplo, en este informe ficticio hay tres parámetros proporcionados a los informes de R Markdown.

### Utilizar un "archivo de ejecución"

Si tiene varios informes que ejecutar, considere la posibilidad de crear un script de R que contenga todos los comandos compile\_reports(). Un usuario puede simplemente ejecutar todos los comandos en este script de R y todos los informes se compilarán. Puedes guardar este "archivo de ejecución" en la carpeta "scripts".

## Salidas

Después de haber compilado los informes unas cuantas veces, la carpeta "outputs" podría tener este aspecto (los resaltados se han añadido para mayor claridad):

* Dentro de "outputs", se han creado subcarpetas para cada informe Rmd
* Dentro de ellas, se han creado otras subcarpetas para cada compilación única
  + Están marcados con fecha y hora ("2021-04-23\_T11-07-36" significa 23 de abril de 2021 a las 11:07:36)
  + Puedes editar el formato de la fecha/hora. Ver compilación de informes
* Dentro de cada carpeta compilada de fecha/hora, se almacena el resultado del informe (por ejemplo, HTML, PDF, Word) junto con el script Rmd (¡control de versiones!) y cualquier otro archivo exportado (por ejemplo, table.csv, epidemic\_curve.png)

Esta es una vista dentro de una de las carpetas con fecha/hora, para el informe "daily\_sitrep". La ruta del archivo está resaltada en amarillo para enfatizar.

Por último, a continuación se muestra una captura de pantalla de la salida del informe HTML.

Puedes utilizar list\_outputs() para revisar una lista de las salidas.

## Varios

### Tejido

Si lo desea, puede "tejer" uno de sus informes R Markdown clicando el botón "Knit". Si hace esto, como por defecto, las salidas aparecerán en la carpeta donde se guarda el Rmd - la carpeta "report\_sources". En versiones anteriores de **reportfactory**, tener cualquier archivo que no sea Rmd en "report\_sources" impediría la compilación, pero esto ya no es así. Puedes ejecutar compile\_reports() y no se producirá ningún error.

### Scripts

Te animamos a utilizar la carpeta "scripts" para almacenar "archivos de ejecución" o scripts .R que se originan en tus scripts .Rmd. Consulta la página sobre [R Markdown](#reports-with-r-markdown) para obtener consejos sobre cómo estructurar tu código en varios archivos.

### Extras

* Con **reportfactory**, puede utilizar la función list\_deps() para listar todos los paquetes requeridos en todos los informes de toda la fábrica.
* Hay un paquete de acompañamiento en desarrollo llamado **rfextras** que ofrece más funciones de ayuda para asistirle en la construcción de informes, tales como:
  + load\_scripts() - carga todos los scripts .R en una carpeta determinada (la carpeta "scripts" por defecto)
  + find\_latest() - encuentra la última versión de un archivo (por ejemplo, el último conjunto de datos)

## Recursos

Consulte la [página de Github del](https://github.com/reconverse/reportfactory) paquete **reportfactory**

Consulte la [página de Github del](https://github.com/reconhub/rfextras) paquete **rfextras**

# #Dashboards con R Markdown

{#dashboards-with-r-markdown}

Esta página cubrirá el uso básico del paquete **flexdashboard**. Este paquete permite formatear fácilmente la salida de R Markdown como un tablero de instrumentos con paneles y páginas. El contenido del panel puede ser texto, figuras/tablas estáticas o gráficos interactivos.

Ventajas de **flexdashboard**:

* Requiere una codificación mínima de R no estándar - con muy poca práctica puede crear rápidamente un panel de control
* El Dashboard puede enviarse por correo electrónico a los compañeros como un archivo HTML autónomo, sin necesidad de servidor
* Puedes combinar **flexdashboard** con **shiny**, **ggplotly** y otros "widgets html" para añadir interactividad

Desventajas de **flexdashboard**:

* Menos personalización en comparación con el uso de **Shiny** para crear un panel de control

En la sección de Recursos se pueden encontrar tutoriales muy completos sobre el uso de **flexdashboard** que sirvieron de base a esta página. A continuación describimos las características principales y damos un ejemplo de construcción de un tablero para explorar un brote, utilizando los datos de la lista de casos.

## Preparación

### Cargar paquetes

En este manual destacamos p\_load() de **pacman**, que instala el paquete si es necesario y lo carga para su uso. También puede cargar los paquetes instalados con library() de R **base.** Consulte la página sobre [los fundamentos de R](#r-basics) para obtener más información sobre los paquetes de R.

### Importar datos

Importamos los datos de casos de una epidemia de ébola simulada. Si quiere seguir el proceso, [clica para descargar linelist "limpia"](https://github.com/epirhandbook/Epi_R_handbook/raw/master/data/case_linelists/linelist_cleaned.rds) (como archivo .rds). Importe los datos con la función import() del paquete **rio** (maneja muchos tipos de archivos como .xlsx, .csv, .rds - vea la página de [importación y exportación](#import-and-export) para más detalles).

A continuación se muestran las primeras 50 filas del listado.

## Crear un nuevo R Markdown

Una vez instalado el paquete, crea un nuevo archivo R Markdown haciendo clicando en File > New file > R Markdown.

En la ventana que se abre, selecciona "Desde la plantilla" y selecciona la plantilla "Flex Dashboard". A continuación, se le pedirá que nombre el documento. En el ejemplo de esta página, nombraremos nuestro R Markdown como "outbreak\_dashboard.Rmd".

## El script

El script es un script de R Markdown, y por lo tanto tiene los mismos componentes y organización que se describen en la página sobre [Informes con R Markdown](#reports-with-r-markdown). Volvemos a revisar brevemente estos y destacamos las diferencias con otros formatos de salida de R Markdown.

### YAML

En la parte superior del script está la cabecera "YAML". Este debe comenzar con tres guiones --- y debe cerrar con tres guiones ---. Los parámetros YAML vienen en pares clave:valor. La **sangría y la colocación de los dos puntos en YAML es importante** - los pares clave:valor están separados por dos puntos (¡no por signos de igualdad!).

El YAML debe comenzar con los metadatos del documento. El orden de estos parámetros YAML primarios (sin sangría) no importa. Por ejemplo:

Puedes utilizar el código R en los valores YAML poniéndolo como código en línea (precedido por r entre comillas) pero también entre comillas (véase más arriba para la fecha).

Un parámetro YAML necesario es output:, que especifica el tipo de archivo que se producirá (por ejemplo, documento\_html, documento\_pdf, documento\_palabra o presentación\_powerpoint). En el caso de **flexdashboard** el valor de este parámetro es un poco confuso - debe establecerse como output:flexdashboard::flex\_dashboard. Ten en cuenta los dos puntos simples y dobles, y el guión bajo. Este parámetro de salida YAML suele ir seguido de dos puntos adicionales y de subparámetros con sangría (ver parámetros orientation: y vertical\_layout: más abajo).

Como se muestra arriba, se utilizan sangrías (2 espacios) para los subparámetros. En este caso, no olvide poner dos puntos adicionales después del primario, como clave:valor:.

Si procede, los valores lógicos deben indicarse en YAML en minúsculas (true, false, null). Si los dos puntos forman parte del valor (por ejemplo, en el título), ponga el valor entre comillas. Vea los ejemplos en las secciones siguientes.

### Trozos de código

Un script de R Markdown puede contener múltiples "trozos" de código - estas son áreas del script donde se puede escribir código R de varias líneas y funcionan como mini scripts R.

Los trozos de código se crean con tres signos de retroceso y corchetes con una "r" minúscula dentro. El fragmento se cierra con tres puntos suspensivos. Puedes crear un nuevo fragmento escribiéndolo tú mismo, utilizando el atajo de teclado "Ctrl + Alt + i" (o Cmd + Shift + r en Mac), o clicando en el icono verde 'insertar un nuevo fragmento de código' en la parte superior de tu editor de scripts. A continuación se ofrecen muchos ejemplos.

### Texto narrativo

Fuera de un "trozo" de código R, puede escribir texto narrativo. Como se describe en la página sobre [Informes con R Markdown](#reports-with-r-markdown), puede poner el texto en cursiva rodeándolo con un asterisco (\*), o en negrita rodeándolo con dos asteriscos (\*\*). Recuerde que las viñetas y los esquemas de numeración son sensibles a las nuevas líneas, a la sangría y a terminar una línea con dos espacios.

También puede insertar código R en línea en el texto, como se describe en la página [Informes con R Markdown](#reports-with-r-markdown), rodeando el código con puntos suspensivos y comenzando el comando con "r": ` 1+1` (véase el ejemplo con la fecha anterior).

### Rúbricas

Los diferentes niveles de encabezamiento se establecen con diferentes números de símbolos hash, como se describe en la página [Informes con R Markdown](#reports-with-r-markdown).

En **flexdashboard**, un encabezado primario (#) crea una "página" del dashboard. Los encabezados de segundo nivel (##) crean una columna o una fila dependiendo de su parámetro de orientación: (ver detalles más abajo). Los encabezados de tercer nivel (###) crean paneles para gráficos, diagramas, tablas, texto, etc.

# Título de primer nivel (página)

## Rúbrica de segundo nivel (fila o columna)

### Rúbrica de tercer nivel (panel para el trazado, el gráfico, etc.)

## Atributos de la sección

Al igual que en un R Markdown normal, puedes especificar los atributos que se aplicarán a las partes del cuadro de mando incluyendo las opciones key=value después de un encabezado, entre llaves { }. Por ejemplo, en un típico informe HTML R Markdown podrías organizar los sub-encabezados en pestañas con ## My heading {.tabset}.

Ten en cuenta que estos atributos se escriben después de un título en una parte de texto del script. Son diferentes a las opciones de **knitr** insertadas dentro en la parte superior de los trozos de código R, como out.height =.

Los atributos de sección específicos de **flexdashboard** incluyen:

* {orientación de los datos=} Establece la orientación de las filas o de las columnas. Si su tablero tiene varias páginas, añade este atributo a cada una de ellas para indicar la orientación (se explica con más detalle en [la sección de diseño](#layout)).
* {data-width=} y {data-height=} establecen el tamaño relativo de los gráficos, columnas y filas dispuestos en la misma dimensión (horizontal o vertical). Los tamaños absolutos se ajustan para llenar mejor el espacio en cualquier dispositivo de visualización gracias al motor [flexbox](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Using_CSS_flexible_boxes).
  + La altura de las figuras también depende de si se establece el parámetro YAML vertical\_layout: fill o vertical\_layout: scroll. Si se establece en scroll, la altura de la figura reflejará la opción tradicional fig.height = en el fragmento de código de R.
  + Consulte la documentación completa sobre el tamaño en el [sitio web de flexdashboard](https://rmarkdown.rstudio.com/flexdashboard/using.html" \l "sizing)
* {.hidden} Utiliza esto para excluir una página específica de la barra de navegación
* {data-navbar=} Utilícelo en un encabezado a nivel de página para anidarlo dentro de un menú desplegable de la barra de navegación. Indique el nombre (entre comillas) del menú desplegable. Véase el ejemplo siguiente.

## Diseño

Ajusta el diseño de tu panel de control de las siguientes maneras:

* Añadir páginas, columnas/filas y gráficos con encabezados R Markdown (por ejemplo, #, ## o ###)
* Ajuste la orientación de los parámetros YAML: a filas o columnas
* Especifica si el diseño llena el navegador o permite el desplazamiento
* Añadir pestañas a un título de sección concreto

### Páginas

Los encabezados de primer nivel (#) en el R Markdown representarán las "páginas" del cuadro de mando. Por defecto, las páginas aparecerán en una barra de navegación a lo largo de la parte superior del tablero.

Puedes agrupar las páginas en un "menú" dentro de la barra de navegación superior añadiendo el atributo {data-navmenu=} al título de la página. Tenga cuidado: no incluya espacios alrededor del signo de igualdad, de lo contrario no funcionará.

Esto es lo que produce el script:

También puedes convertir una página o una columna en una "barra lateral" en el lado izquierdo del panel de control añadiendo el atributo {.sidebar}. Puedes contener texto (visible desde cualquier página) o, si has integrado una interactividad **Shiny,** puede ser útil para contener controles de entrada del usuario, como deslizadores o menús desplegables.

Esto es lo que produce el script:

### Orientación

Establezca el parámetro orientation: yaml para indicar cómo deben interpretarse sus encabezados de segundo nivel (##) de R Markdown - como orientación: columnas u orientación: filas.

Los encabezados de segundo nivel (##) se interpretarán como nuevas columnas o filas en función de este ajuste de orientación.

Si establece la orientación: columnas, las cabeceras de segundo nivel crearán nuevas columnas en el tablero. El siguiente tablero tiene una página, que contiene dos columnas, con un total de tres paneles. Puedes ajustar el ancho relativo de las columnas con {data-width=} como se muestra a continuación.

Esto es lo que produce el script:

Si establece la orientación: filas, los encabezados de segundo nivel crearán nuevas filas en lugar de columnas. A continuación se muestra el mismo script que el anterior, pero con la orientación: filas para que los encabezados de segundo nivel produzcan filas en lugar de columnas. Puedes ajustar la altura relativa de las filas con {data-height=} como se muestra a continuación.

Esto es lo que produce el script:

Si tu tablero tiene varias páginas, puede designar la orientación para cada página específica añadiendo el atributo {data-orientation=} la cabecera de cada página (especifique filas o columnas sin comillas).

### Pestañas

Puedes dividir el contenido en pestañas con el atributo {.tabset}, como en otras salidas HTML R Markdown.

Simplemente añada este atributo después del título deseado. Los subtítulos bajo ese encabezado se mostrarán como pestañas. Por ejemplo, en el script de ejemplo que aparece a continuación, la columna 2 de la derecha (##) se modifica para que la curva epidémica y los paneles de la tabla (###) se muestren en pestañas.

Puedes hacer lo mismo con las filas si su orientación es de filas.

Esto es lo que produce el script:

## Añadir contenido

Comencemos a construir un panel de control. Nuestro sencillo panel de control tendrá 1 página, 2 columnas y 4 paneles. Construiremos los paneles pieza por pieza para la demostración.

Puedes incluir fácilmente salidas estándar de R, como texto, ggplots y tablas (véase la página [Tablas para la presentación](#tables-for-presentation)). Simplemente codifíquelos dentro de un fragmento de código R como lo haría con cualquier otro script de R Markdown.

Nota: puede descargar el script Rmd terminado y el resultado del Dashboard en HTML - vea la página de [descarga del manual y los datos](#download-handbook-and-data).

### Texto

Puedes escribir el texto de Markdown e incluir el código en línea como para cualquier otra salida de R Markdown. Consulte la página [Informes con R Markdown](#reports-with-r-markdown) para obtener más detalles.

En este cuadro de mando incluimos un panel de texto resumido que incluye un texto dinámico que muestra la última fecha de hospitalización y el número de casos notificados en el brote.

### Tablas

Puedes incluir trozos de código R que impriman salidas como tablas. Pero la salida se verá mejor y responderá al tamaño de la ventana si utiliza la función kable() de **knitr** para mostrar sus tablas. Las funciones de **flextable** pueden producir tablas acortadas / cortadas.

Por ejemplo, a continuación alimentamos el linelist() a través de un comando count() para producir una tabla resumen de casos por hospital. Finalmente, la tabla se canaliza a knitr::kable() y el resultado tiene una barra de desplazamiento a la derecha. Puedes leer más sobre la personalización de la tabla con kable() y **kableExtra** [aquí](https://cran.r-project.org/web/packages/kableExtra/vignettes/awesome_table_in_html.html).

Esto es lo que produce el script:

Si deseas mostrar una tabla dinámica que permita al usuario filtrar, ordenar y/o clicar a través de las "páginas" del dataframe, utiliza el paquete **DT** y su función datatable(), como en el código siguiente.

En el código de ejemplo que sigue, se imprime linelist del dataframe. Se puede establecer rownames = FALSE para conservar el espacio horizontal, y filter = "top" para tener filtros en la parte superior de cada columna. Se puede proporcionar una lista de otras especificaciones a options =. A continuación, establecemos pageLength = para que aparezcan 5 filas y scrollX = para que el usuario pueda utilizar una barra de desplazamiento en la parte inferior para desplazarse horizontalmente. El argumento class = 'white-space: nowrap' asegura que cada fila sea sólo una línea (no varias líneas). Puedes consultar otros argumentos y valores posibles [aquí](https://rstudio.github.io/DT/?_ga=2.2810736.1321860763.1619286819-369061888.1601594705) o introduciendo ?datatable

### Gráficos

Puedes imprimir gráficos en un panel de control como lo haría en un script de R. En nuestro ejemplo, utilizamos el paquete **incidence2** para crear una "epicurva" por grupo de edad con dos simples comandos (véase la página de [curvas epidémicas](#epidemic-curves)). Sin embargo, podría utilizar ggplot() e imprimir un gráfico de la misma manera.

Esto es lo que produce el script:

### Gráficos interactivos

También puedes pasar un ggplot estándar u otro objeto de gráfico a ggplotly() del paquete **plotly** (véase la página de [gráficos interactivos](#interactive-plots-1)). Esto hará que el gráfico sea interactivo, permitirá al lector hacer un "zoom", y mostrará sobre el tablero el valor de cada punto de datos (en este escenario el número de casos por semana y el grupo de edad en la curva).

Esto es lo que parece en el tablero de mandos (gif). Esta funcionalidad interactiva seguirá funcionando incluso si envías por correo electrónico el Dashboard como un archivo estático (no en línea en un servidor).

### Widgets HTML

Los [widgets HTML para R](http://www.htmlwidgets.org/) son un tipo especial de paquetes R que aumentan la interactividad utilizando bibliotecas JavaScript. Se pueden incrustar en salidas R Markdown (como un flexdashboard) y en dashboards de Shiny.

Algunos ejemplos comunes de estos widgets son:

* Plotly (utilizado en la página de este manual y en la página de [Plots interativos])
* visNetwork (utilizado en la página de [cadenas de transmisión de](#transmission-chains) este manual)
* Folleto (utilizado en la página de [fundamentos del SIG](#gis-basics) de este manual)
* dygraphs (útil para mostrar interactivamente los datos de las series temporales)
* DT (datatable()) (utilizado para mostrar tablas dinámicas con filtro, ordenación, etc.)

A continuación demostramos la adición de una cadena de transmisión de epidemias que utiliza visNetwork al tablero. El guión muestra sólo el nuevo código añadido a la sección "Columna 2" del guión R Markdown. Puedes encontrar el código en la página de cadenas de transmisión de este manual.

Esto es lo que produce el script:

## Organización del código

Puedes elegir tener todo el código dentro del script de R Markdown **flexdashboard**. Alternativamente, para tener un script de cuadro de mando más limpio y conciso, puede elegir llamar al código/figuras que están alojadas o creadas en scripts R externos. Esto se describe con mayor detalle en la página [Informes con R Markdown](#reports-with-r-markdown).

## Shiny

La integración del paquete R **shiny** puede hacer que sus Dashboards sean aún más reactivos a la entrada del usuario. Por ejemplo, puede hacer que el usuario selecciona una jurisdicción, o un rango de fechas, y hacer que los paneles reaccionen a su elección (por ejemplo, filtrar los datos mostrados). Para incrustar la reactividad **de shiny** en **el flexdashboard**, sólo tienes que hacer unos pocos cambios en tu script de R Markdown del **flexdashboard**.

También se puede utilizar **shiny** para producir aplicaciones/tableros sin flexdashboard. La página del manual sobre [Tableros de Control con Shiny](#dashboards-with-shiny) ofrece una visión general de este enfoque, incluyendo consejos sobre la sintaxis de **Shiny**, la estructura de los archivos de la aplicación y las opciones para compartir/publicar (incluyendo opciones de servidor gratuito). Esta sintaxis y los consejos generales se traducen también en el contexto de **flexdashboard**.

La incrustación de **shiny** en **el flexdashboard supone**, sin embargo, un cambio fundamental en tu flexdashboard. Ya no producirá una salida HTML que pueda enviar por correo electrónico y que cualquiera pueda abrir y ver. En su lugar, será una "aplicación". El botón "Knit" en la parte superior del script será reemplazado por un icono "Run document", que abrirá una instancia del dashboard interactivo localmente en tu ordenador.

Para compartir tu panel de control, ahora será necesario que:

* Enviar el script Rmd al espectador, ellos lo abren en R en tu ordenador, y ejecutan la aplicación, o
* La aplicación/panel de control se aloja en un servidor accesible para el espectador

Por lo tanto, la integración de **shiny** tiene ventajas, pero también complicaciones. Si la facilidad de compartir por correo electrónico es una prioridad y no necesita las capacidades reactivas de shiny, considere la reducida interactividad que ofrece ggplotly() como se ha demostrado anteriormente.

A continuación damos un ejemplo muy sencillo utilizando el mismo "outbreak\_dashboard.Rmd" que el anterior. Una amplia documentación sobre la integración de Shiny en **flexdashboard** está disponible en línea [aquí](https://rmarkdown.rstudio.com/flexdashboard/shiny.html).

### Ajustes

Habilitar **shiny** en un **flexdashboard** añadiendo el parámetro YAML runtime: shiny en el mismo nivel de sangría que output:, como se indica a continuación:

---

título: "Tablero de mandos del brote (Demo Shiny)"

de salida:

flexdashboard::flex\_dashboard:

orientación: columnas

diseño\_vertical: relleno

tiempo de ejecución: Shiny

---

También es conveniente habilitar una "barra lateral" para albergar los Shinys widgets de entrada que recogerán la información del usuario. Como se explicó anteriormente, cree una columna e indique la opción {.sidebar} para crear una barra lateral en el lado izquierdo. Dentro de esta columna se pueden añadir trozos de texto y R que contengan los comandos de entrada de **shiny**.

Si tu aplicación/panel está alojado en un servidor y puede tener varios usuarios simultáneos, nombre el primer trozo de código R como global. Incluye los comandos para importar/cargar tus datos en este chunk. Este chunk con nombre especial es tratado de manera diferente, y los datos importados dentro de él sólo se importan una vez (no continuamente) y están disponibles para todos los usuarios. Esto mejora la velocidad de arranque de la aplicación.

### Ejemplo trabajado

Aquí adaptamos el script flexdashboard "outbreak\_dashboard.Rmd" para incluir **shiny**. Añadiremos la capacidad de que el usuario selecciona un hospital de un menú desplegable, y que la curva epidémica refleje sólo los casos de ese hospital, con un título de gráfico dinámico. Hacemos lo siguiente:

* Añadir runtime: shiny al YAML
* Renombrar el chunk de configuración como global
* Crear una barra lateral que contenga:
  + Código para crear un vector de nombres únicos de hospitales
  + Un comando selectInput() (menú desplegable **Shiny**) con la elección de los nombres de los hospitales. La selección se guarda como hospital\_choice, a la que se puede hacer referencia en código posterior como input$hospital\_choice
* El código de la curva epidémica (columna 2) está envuelto dentro de renderPlot({ }), incluyendo:
  + Un filtro en los datos que restringe la columna hospital al valor actual de input$hospital\_choice
  + Un título de gráfico dinámico que incorpora input$hospital\_choice

Ten en cuenta que cualquier código que haga referencia a un valor de input$ debe estar dentro de una función render({}) (para ser reactiva).

Aquí está la parte superior del script, incluyendo el YAML, el chunk global y la barra lateral:

Aquí está la Columna 2, con el gráfico de la epicurva reactiva:

Y aquí está el Dashboard:

### Otros ejemplos

Para leer un ejemplo relacionado con la salud de un Shiny-flexdashboard que utiliza la interactividad **de Shiny** y el widget de mapeo **de folletos**, consulte este capítulo del libro en línea [Geospatial Health Data: Modeling and Visualization with R-INLA and Shiny](https://www.paulamoraga.com/book-geospatial/sec-dashboardswithshiny.html).

## Compartir

Los Dashboards que no contengan elementos Shiny producirán un archivo HTML (.html), que puede enviarse por correo electrónico (si el tamaño lo permite). Esto es útil, ya que puede enviar el informe del "cuadro de mando" y no tener que configurar un servidor para alojarlo como un sitio web.

Si ha incrustado **shiny**, no podrá enviar una salida por correo electrónico, pero puede enviar el propio script a un usuario de R, o alojar el Dashboard en un servidor como se ha explicado anteriormente.

## Recursos

A continuación se pueden encontrar excelentes tutoriales que informaron esta página. Si los revisas, lo más probable es que en una hora puedas tener tu propio panel de control.

<https://bookdown.org/yihui/rmarkdown/dashboards.html>

<https://rmarkdown.rstudio.com/flexdashboard/>

<https://rmarkdown.rstudio.com/flexdashboard/using.html>

<https://rmarkdown.rstudio.com/flexdashboard/examples.html>

# #Dashboards con Shiny

{#dashboards-with-shiny}

Los Dashboards (cuadros de mando o tableros de control) suelen ser una buena forma de compartir los resultados de los análisis con otras personas. Elaborar un cuadro de mando con **shiny** requiere un conocimiento relativamente avanzado del lenguaje R, pero ofrece una personalización y unas posibilidades increíbles.

Se recomienda que alguien que esté aprendiendo a usar Dashboards con **shiny** tenga buenos conocimientos de transformación y visualización de datos, y se sienta cómodo depurando código y escribiendo funciones. Trabajar con dashboards no es intuitivo cuando se empieza, y es difícil de entender a veces, pero es una gran habilidad para aprender y se hace mucho más fácil con la práctica.

Esta página dará una breve visión general de cómo hacer Dashboards con **shiny** y sus extensiones. Para un método alternativo de hacer dashboards que es más rápido, más fácil, pero quizás menos personalizable, vea la página sobre **flextable** ([Dashboards with R Markdown](#dashboards-with-r-markdown)).

## Preparación

### Cargar paquetes

En este manual destacamos p\_load() de **pacman**, que instala el paquete si es necesario y lo carga para su uso. También puede cargar los paquetes instalados con library() de R **base.** Consulte la página sobre [los fundamentos de R](#r-basics) para obtener más información sobre los paquetes de R.

Comenzamos instalando el paquete R **Shiny**:

### Importar datos

Si quieres seguir esta página, consulta esta sección del [Manual de descarga y datos](#data_shiny). Hay enlaces para descargar los scripts de R y los archivos de datos que producen la aplicación final de Shiny.

Si intenta reconstruir la aplicación utilizando estos archivos, Ten en cuenta la estructura de carpetas del proyecto R que se crea en el transcurso de la demostración (por ejemplo, carpetas para "data" y para "funcs").

## La estructura de una aplicación Shiny

### Estructuras básicas de archivos

Para entender Shiny, primero tenemos que entender cómo funciona la estructura de archivos de una aplicación. Deberíamos crear un nuevo directorio antes de empezar. Esto puede hacerse más fácil eligiendo Nuevo proyecto en Rstudio, y eligiendo Aplicación Web Shiny. Esto creará la estructura básica de una aplicación shiny para ti.

Al abrir este proyecto, notarás que ya hay un archivo .R llamado app.R. Es esencial que tengamos una de las dos estructuras básicas de archivos:

1. Un archivo llamado app.R, o
2. Dos archivos, uno llamado ui.R y el otro server.R

En esta página, utilizaremos el primer enfoque de tener un archivo llamado app.R. Aquí hay un script de ejemplo:

Si abres este archivo, te darás cuenta de que hay dos objetos definidos: uno llamado ui y otro llamado server. Estos objetos deben ser definidos en todas las aplicaciones shiny y son fundamentales para la estructura de la propia aplicación. De hecho, la única diferencia entre las dos estructuras de archivos descritas anteriormente es que en la estructura 1, tanto ui como server están definidos en un solo archivo, mientras que en la estructura 2 están definidos en archivos separados. Nota: también podemos (y deberíamos si tenemos una aplicación más grande) tener otros archivos .R en nuestra estructura que podemos source() en nuestra aplicación.

### El servidor y la ui

A continuación, tenemos que entender lo que hacen realmente los objetos servidor y ui. En pocas palabras, se trata de dos objetos que interactúan entre sí cada vez que el usuario interactúa con la shiny app.

El elemento de interfaz de usuario de una aplicación Shiny es, en un nivel básico, el código R que crea una interfaz HTML. Esto significa todo lo que se muestra en la UI de una app. Esto generalmente incluye:

* "Widgets" - menús desplegables, casillas de verificación, deslizadores, etc. con los que puede interactuar el usuario
* Gráficos, tablas, etc. - resultados que se generan con el código R
* Aspectos de la navegación de una aplicación: pestañas, paneles, etc.
* Texto genérico, hipervínculos, etc.
* Elementos HTML y CSS (abordados más adelante)

Lo más importante que hay que entender sobre la UI es que recibe entradas del usuario y muestra salidas del servidor. No hay código activo que se ejecute en la UI en ningún momento - todos los cambios que se ven en la UI pasan por el servidor (más o menos). Así que tenemos que hacer nuestros gráficos, descargas, etc en el servidor

El servidor de la shiny app es donde se ejecuta todo el código una vez que la aplicación se inicia. La forma en que esto funciona es un poco confusa. La función del servidor reaccionará efectivamente a la interfaz del usuario con la UI, y ejecutará trozos de código en respuesta. Si las cosas cambian en el servidor, estas serán pasadas de vuelta a la UI, donde los cambios pueden ser vistos. Es importante destacar que el código en el servidor se ejecutará de forma no consecutiva (o es mejor pensarlo así). Básicamente, cada vez que una entrada de la ui afecte a un trozo de código en el servidor, éste se ejecutará automáticamente, y se producirá y mostrará esa salida.

Probablemente todo esto suene muy abstracto por ahora, así que tendremos que sumergirnos en algunos ejemplos para tener una idea clara de cómo funciona realmente.

### Antes de empezar a crear una aplicación

Antes de empezar a construir una aplicación, es muy útil saber qué quieres construir. Dado que tu interfaz de usuario estará escrita en código, no puedes visualizar realmente lo que estás construyendo a menos que tengas como objetivo algo específico. Por esta razón, es inmensamente útil mirar muchos ejemplos de aplicaciones Shinys para tener una idea de lo que puedes hacer - ¡incluso mejor si puedes mirar el código fuente detrás de estas aplicaciones! Algunos de los mejores recursos para ello son:

* La [galería de aplicaciones de Rstudio](https://shiny.rstudio.com/gallery/)

Una vez que tengas una idea de lo que es posible, también es útil hacer un mapa de cómo quieres que sea la tuya; puedes hacerlo en papel o en cualquier software de dibujo (PowerPoint, MS paint, etc.). Es útil empezar con algo sencillo para tu primera aplicación. Tampoco hay que avergonzarse de utilizar el código que encuentres en Internet de una buena aplicación como plantilla para tu trabajo: es mucho más fácil que construir algo desde cero.

## Construir una interfaz de usuario

Cuando construimos nuestra aplicación, es más fácil trabajar en la interfaz de usuario primero para que podamos ver lo que estamos haciendo, y no arriesgarnos a que la aplicación falle debido a cualquier error del servidor. Como se mencionó anteriormente, a menudo es bueno utilizar una plantilla cuando se trabaja en la interfaz de usuario. Hay una serie de diseños estándar que se pueden utilizar con shiny que están disponibles en el paquete base de shiny, pero vale la pena señalar que también hay una serie de extensiones del paquete como shinydashboard. Utilizaremos un ejemplo del paquete base de shiny para empezar.

Una interfaz de usuario Shiny se define generalmente como una serie de funciones anidadas, en el siguiente orden

1. Una función que define el diseño general (la más básica es fluidPage(), pero hay más disponibles)
2. Paneles dentro del diseño como:
   * una barra lateral (sidebarPanel())
   * un panel "principal" (mainPanel())
   * una pestaña (tabPanel())
   * una "columna" genérica (column())
3. Widgets y salidas: pueden conferir entradas al servidor (widgets) o salidas del servidor (salidas)
   * Los widgets suelen tener el estilo de xxxInput(), por ejemplo, selectInput()
   * Las salidas suelen tener el estilo de xxxOutput(), por ejemplo, plotOutput()

Vale la pena repetir que estos datos no se pueden visualizar fácilmente de forma abstracta, por lo que es mejor ver un ejemplo. Consideremos la posibilidad de crear una aplicación básica que visualice nuestros datos de recuento de instalaciones de malaria por distrito. Estos datos tienen muchos parámetros diferentes, por lo que sería estupendo que el usuario final pudiera aplicar algunos filtros para ver los datos por grupo de edad/distrito según tu criterio. Podemos utilizar un diseño Shiny muy simple para empezar - el diseño de la barra lateral. Se trata de un diseño en el que los widgets se colocan en una barra lateral a la izquierda, y el gráfico se coloca a la derecha.

Planifiquemos nuestra aplicación: podemos empezar con un selector que nos permita elegir el distrito donde queremos visualizar los datos, y otro que nos permita visualizar el grupo de edad que nos interesa. Con estos filtros pretendemos mostrar una epicurva que refleje estos parámetros. Para ello necesitamos:

1. Dos menús desplegables que nos permiten elegir el distrito que queremos y el grupo de edad que nos interesa.
2. Un área donde podemos mostrar nuestra epicurva resultante.

Esto podría ser algo así:

Cuando app.R se ejecuta con el código de interfaz de usuario anterior (sin código activo en la parte del servidor de app.R), el diseño aparece con el siguiente aspecto: ten en cuenta que no habrá ningún gráfico si no hay un servidor que lo represente, ¡pero nuestras entradas están funcionando!

Esta es una buena oportunidad para discutir cómo funcionan los widgets - nota que cada widget está aceptando un inputId, una etiqueta, y una serie de otras opciones que son específicas para el tipo de widget. Este inputId es extremadamente importante - estos son los IDs que se utilizan para pasar la información de la UI al servidor. Por esta razón, deben ser únicos. Deberías hacer un esfuerzo para nombrarlos algo sensato, y específico a lo que están interactuando en casos de aplicaciones más grandes.

Deberías leer la documentación cuidadosamente para conocer todos los detalles sobre lo que hace cada uno de estos widgets. Los widgets pasarán tipos específicos de datos al servidor dependiendo del tipo de widget, y esto debe ser entendido completamente. Por ejemplo, selectInput() pasará un tipo de carácter al servidor:

* Si seleccionamos Spring para el primer widget aquí, pasará el objeto carácter "Spring" al servidor.
* Si seleccionamos dos elementos del menú desplegable, aparecerán como un vector de caracteres (por ejemplo, c("Primavera", "Bolo")).

Otros widgets pasarán diferentes tipos de objetos al servidor. Por ejemplo:

* numericInput() pasará un objeto de tipo numérico al servidor
* checkboxInput() pasará un objeto de tipo lógico al servidor (TRUE o FALSE)

También vale la pena notar el vector con nombre que usamos para los datos de edad aquí. Para muchos widgets, el uso de un vector con nombre como las opciones mostrará los nombres del vector como las opciones de visualización, pero pasará el valor seleccionado del vector al servidor. Por ejemplo, aquí alguien puede seleccionar "15+" en el menú desplegable, y la interfaz de usuario pasará "malaria\_rdt\_15" al servidor, que resulta ser el nombre de la columna que nos interesa.

Hay un montón de widgets que puedes utilizar para hacer muchas cosas con tu aplicación. Los widgets también permiten cargar archivos en la aplicación y descargar resultados. También hay algunas excelentes extensiones de shiny que te dan acceso a más widgets que el shiny base - el paquete **shinyWidgets** es un gran ejemplo de esto. Para ver algunos ejemplos puedes consultar los siguientes enlaces:

* [base de la galería de widgets Shinys](https://shiny.rstudio.com/gallery/widget-gallery.html)
* [Galería de shinyWidgets](https://github.com/dreamRs/shinyWidgets)

## Carga de datos en nuestra aplicación

El siguiente paso en el desarrollo de nuestra aplicación es poner en marcha el servidor. Para ello, sin embargo, tenemos que conseguir algunos datos en nuestra aplicación, y averiguar todos los cálculos que vamos a hacer. Una aplicación Shiny no es fácil de depurar, ya que a menudo no está claro de dónde provienen los errores, por lo que es ideal para obtener todo nuestro procesamiento de datos y código de visualización de trabajo antes de empezar a hacer el propio servidor.

Así que dado que queremos hacer una aplicación que muestre curvas epi que cambien en base a la entrada del usuario, deberíamos pensar en qué código necesitaríamos para ejecutar esto en un script normal de R. Necesitaremos:

1. Cargar nuestros paquetes
2. Cargar nuestros datos
3. Transformar nuestros datos
4. Desarrollar una función para visualizar nuestros datos en función de las entradas del usuario

Esta lista es bastante sencilla, y no debería ser demasiado difícil de hacer. Ahora es importante pensar qué partes de este proceso deben hacerse una sola vez y qué partes deben ejecutarse en respuesta a las entradas del usuario. Esto se debe a que las aplicaciones Shinys generalmente ejecutan algún código antes de ejecutarse, que sólo se realiza una vez. Ayudará al rendimiento de nuestra aplicación si la mayor parte de nuestro código puede ser trasladado a esta sección. Para este ejemplo, sólo necesitamos cargar nuestros datos/paquetes y hacer transformaciones básicas una vez, así que podemos poner ese código fuera del servidor. Esto significa que lo único que necesitaremos en el servidor es el código para visualizar nuestros datos. Vamos a desarrollar todos estos componentes en un script primero. Sin embargo, ya que estamos visualizando nuestros datos con una función, también podemos poner el código de la función fuera del servidor para que nuestra función esté en el entorno cuando la aplicación se ejecute.

Primero vamos a cargar nuestros datos. Ya que estamos trabajando con un nuevo proyecto, y queremos limpiarlo, podemos crear un nuevo directorio llamado data, y añadir nuestros datos de malaria allí. Podemos ejecutar este código de abajo en un script de prueba que eventualmente borraremos cuando limpiemos la estructura de nuestra aplicación.

pacman::p\_load("tidyverse", "lubridate")

# leer datos

malaria\_data <- rio::import(here::here("data", "malaria\_facility\_count\_data.rds")) %>%

as\_tibble()

print(datos\_del\_paludismo)

Será más fácil trabajar con estos datos si utilizamos estándares de datos ordenados, por lo que también debemos transformarlos en un formato de datos más largo, donde el grupo de edad es una columna, y los casos son otra columna. Podemos hacer esto fácilmente usando lo que hemos aprendido en la página de [Pivoteo de datos](#pivoting-data).

malaria\_data <- malaria\_data %>%

select(-newid) %>%

pivot\_longer(cols = starts\_with("malaria\_"), names\_to = "age\_group", values\_to = "cases\_reported")

print(datos\_del\_paludismo)

Y con esto hemos terminado de preparar nuestros datos! Esto tacha los puntos 1, 2 y 3 de nuestra lista de cosas a desarrollar para nuestro "script de prueba de R". La última tarea, y la más difícil, será construir una función para producir una epicurva basada en parámetros definidos por el usuario. Como se mencionó anteriormente, se recomienda encarecidamente que cualquier persona que aprenda a brillar primero mire la sección sobre la programación funcional ([Escribir funciones](#writing-functions-1)) para entender cómo funciona esto!

Al definir nuestra función, puede ser difícil pensar en los parámetros que queremos incluir. Para la programación funcional con shiny, cada parámetro relevante tendrá generalmente un widget asociado a él, así que pensar en esto suele ser bastante fácil. Por ejemplo, en nuestra aplicación actual, queremos ser capaces de filtrar por distrito, y tener un widget para ello, por lo que podemos añadir un parámetro de distrito para reflejar esto. No tenemos ninguna funcionalidad de la aplicación para filtrar por centro (por ahora), así que no necesitamos añadir esto como parámetro. Empecemos haciendo una función con tres parámetros:

1. los datos básicos
2. El distrito de elección
3. El grupo de edad elegido

No entraremos en grandes detalles sobre esta función, ya que su funcionamiento es relativamente sencillo. Una cosa a tener en cuenta, sin embargo, es que manejamos los errores devolviendo NULL cuando de otro modo daría un error. Esto se debe a que cuando un servidor Shiny produce un objeto NULL en lugar de un objeto gráfico, ¡no se mostrará nada en la interfaz de usuario! Esto es importante, ya que de lo contrario los errores a menudo harán que tu aplicación deje de funcionar.

Otra cosa a tener en cuenta es el uso del operador %in% cuando se evalúa la entrada del distrito. Como se mencionó anteriormente, esto podría llegar como un vector de caracteres con múltiples valores, por lo que el uso de %in% es más flexible que, por ejemplo, ==.

Vamos a probar nuestra función!

plot\_epicurve(datos\_paludismo, distrito = "Bolo", grupo de edad = "paludismo\_rdt\_0-4")

Con nuestra función funcionando, ahora tenemos que entender cómo todo esto va a encajar en nuestra Shiny aplicación. Hemos mencionado el concepto de código de inicio antes, pero vamos a ver cómo podemos incorporar esto en la estructura de nuestra aplicación. Hay dos maneras de hacerlo.

1. Escribe este código en tu archivo app.R al principio del script (por encima de la interfaz de usuario), o
2. Crea un nuevo archivo en el directorio de tu aplicación llamado global.R, y pon el código de inicio en este archivo.

Vale la pena señalar en este punto que generalmente es más fácil, especialmente con aplicaciones más grandes, utilizar la segunda estructura de archivos, ya que permite separar su estructura de archivos de una manera sencilla. Vamos a desarrollar completamente este script global.R ahora. Esto es lo que podría parecer:

Fácil! Una gran característica de shiny es que entenderá para qué sirven los archivos llamados app.R, server.R, ui.R y global.R, por lo que no es necesario conectarlos entre sí mediante ningún código. Así que sólo con tener este código en global.R en el directorio se ejecutará antes de que iniciemos nuestra app!

También debemos tener en cuenta que mejoraría la organización de nuestra aplicación si movemos la función de trazado a tu propio archivo - esto será especialmente útil a medida que las aplicaciones se hacen más grandes. Para hacer esto, podríamos hacer otro directorio llamado funcs, y poner esta función en un archivo llamado plot\_epicurve.R. Podríamos entonces leer esta función a través del siguiente comando en global.R

Ten en cuenta que siempre debes especificar local = TRUE en las aplicaciones shiny, ya que afectará a la obtención de recursos cuando/si la aplicación se publica en un servidor.

## Desarrollar un servidor de aplicaciones

Ahora que tenemos la mayor parte de nuestro código, sólo tenemos que desarrollar nuestro servidor. Esta es la pieza final de nuestra aplicación, y es probablemente la más difícil de entender. El servidor es una gran función de R, pero es útil pensar en él como una serie de funciones más pequeñas, o tareas que la aplicación puede realizar. Es importante entender que estas funciones no se ejecutan en un orden lineal. Hay un orden en ellas, pero no es necesario entenderlo del todo cuando se empieza con Shiny. A un nivel muy básico, estas tareas o funciones se activarán cuando haya un cambio en las entradas del usuario que las afecte, a menos que el desarrollador las haya configurado para que se comporten de forma diferente. De nuevo, todo esto es bastante abstracto, pero vamos a repasar primero los tres tipos básicos de objetos shiny

1. Fuentes reactivas - este es otro término para las entradas del usuario. El servidor shiny tiene acceso a las salidas de la UI a través de los widgets que hemos programado. Cada vez que los valores de estos se cambian, esto se pasa al servidor.
2. Conductores reactivos - estos son objetos que existen sólo dentro del servidor Shiny. En realidad no los necesitamos para aplicaciones simples, pero producen objetos que sólo pueden ser vistos dentro del servidor, y utilizados en otras operaciones. Generalmente dependen de fuentes reactivas.
3. Puntos finales: son las salidas que se pasan del servidor a la interfaz de usuario. En nuestro ejemplo, esto sería la curva epi que estamos produciendo.

Con esto en mente vamos a construir nuestro servidor paso a paso. Vamos a mostrar nuestro código de interfaz de usuario de nuevo aquí sólo para referencia:

De este código UI tenemos:

* Dos entradas:
  + Selector de distrito (con un inputId de select\_district)
  + Selector de grupo de edad (con un inputId de select\_agegroup)
* Una salida:
  + La epicurva (con un outputId de malaria\_epicurve)

Como hemos dicho anteriormente, estos nombres únicos que hemos asignado a nuestras entradas y salidas son cruciales. Deben ser únicos y se utilizan para pasar información entre la ui y el servidor. En nuestro servidor, accedemos a nuestras entradas a través de la sintaxis input$inputID y a las salidas y las pasamos a la ui a través de la sintaxis output$output\_name ¡Veamos un ejemplo, porque de nuevo esto es difícil de entender de otra manera!

El servidor para una aplicación simple como esta es en realidad bastante sencillo. Te darás cuenta de que el servidor es una función con tres parámetros - entrada, salida y sesión - esto no es tan importante para entender por ahora, pero es importante seguir esta configuración. En nuestro servidor sólo tenemos una tarea - esta renderiza un gráfico basado en nuestra función que hicimos antes, y las entradas del servidor. Fíjate en que los nombres de los objetos de entrada y salida se corresponden exactamente con los de la interfaz de usuario.

Para entender los fundamentos de cómo el servidor reacciona a las entradas del usuario, debe tener en cuenta que la salida sabrá (a través del paquete subyacente) cuando las entradas cambian, y volver a ejecutar esta función para crear un gráfico cada vez que cambian. Ten en cuenta que también utilizamos la función renderPlot() aquí - esta es una de una familia de funciones específicas del tipo que pasan esos objetos a una salida ui. Hay una serie de funciones que se comportan de manera similar, pero hay que asegurarse de que la función utilizada coincide con el tipo de objeto que se está pasando a la ui. Por ejemplo:

* renderText() - enviar texto a la ui
* renderDataTable - envía una tabla interactiva a la ui.

Recuerde que estos también necesitan coincidir con la función de salida utilizada en la ui - así que renderPlot() se empareja con plotOutput(), y renderText() se empareja con textOutput().

Así que finalmente hemos hecho una aplicación que funciona! Podemos ejecutarla clicando el botón Ejecutar aplicación en la parte superior derecha de la ventana de script en Rstudio. Debes tener en cuenta que puedes elegir ejecutar tu aplicación en tu navegador por defecto (en lugar de Rstudio), lo que reflejará con mayor precisión el aspecto que tendrá la aplicación para otros usuarios.

Es divertido observar que en la consola R, la aplicación está "escuchando". Hablando de reactividad!

## Añadir más funcionalidad

En este punto tenemos finalmente una aplicación en funcionamiento, pero tenemos muy poca funcionalidad. Tampoco hemos rascado la superficie de lo que shiny puede hacer, ¡así que hay mucho más que aprender! Vamos a seguir construyendo nuestra aplicación actual añadiendo algunas características adicionales. Algunas cosas que podría ser bueno añadir podrían ser:

1. Algunos textos explicativos
2. Un botón de descarga para nuestra gráfica - esto proporcionaría al usuario una versión de alta calidad de la imagen que está generando en la aplicación
3. Un selector de instalaciones específicas
4. Otra página del panel de control: podría mostrar una tabla con nuestros datos.

Esto es mucho para agregar, pero podemos usarlo para aprender sobre un montón de diferentes características de Shiny en el camino. Hay mucho que aprender sobre Shiny (puede ser muy avanzado, pero es de esperar que una vez que los usuarios tienen una mejor idea de cómo usarlo pueden llegar a ser más cómodo usando fuentes de aprendizaje externas también).

### Añadir texto estático

Vamos a hablar primero de la adición de texto estático a nuestra aplicación Shiny. Añadir texto a nuestra aplicación es extremadamente fácil, una vez que se tiene un conocimiento básico de la misma. Dado que el texto estático no cambia en la aplicación shiny (si quieres que cambie, puedes utilizar las funciones de renderizado de texto en el servidor), todo el texto estático de shiny se añade generalmente en la interfaz de usuario de la aplicación. No vamos a entrar en detalles, pero puedes añadir un número de elementos diferentes a su ui (e incluso personalizados) mediante la interfaz de R con HTML y css.

HTML y css son lenguajes que intervienen explícitamente en el diseño de la interfaz de usuario. No es necesario entenderlos demasiado bien, pero HTML crea objetos en la interfaz de usuario (como un cuadro de texto, o una tabla), y css se utiliza generalmente para cambiar el estilo y la estética de esos objetos. Shiny tiene acceso a una gran variedad de etiquetas HTML - éstas están presentes para los objetos que se comportan de una manera específica, como los encabezados, los párrafos de texto, los saltos de línea, las tablas, etc. Podemos utilizar algunos de estos ejemplos así:

* h1() - esta es una etiqueta de encabezado, que hará que el texto adjunto sea automáticamente más grande, y cambiará los valores predeterminados en cuanto a la fuente, el color, etc. (dependiendo del tema general de tu aplicación). Puedes acceder a subtítulos cada vez más pequeños con h2() hasta h6() también. El uso es así:
  + h1("mi cabecera - sección 1")
* p() - esta es una etiqueta de párrafo, que hará que el texto encerrado sea similar al texto de un cuerpo de texto. Este texto se envolverá automáticamente, y será de un tamaño relativamente pequeño (los pies de página podrían ser más pequeños, por ejemplo.) Piense en ello como el cuerpo de texto de un documento de Word. El uso es así:
  + p("Este es un cuerpo de texto más grande donde explico la función de mi aplicación")
* tags$b() y tags$i() - se utilizan para crear etiquetas$b() en negrita y etiquetas$i() en cursiva con el texto que se incluya.
* tags$ul(), tags$ol() y tags$li() - son etiquetas utilizadas para crear listas. Todas ellas se utilizan dentro de la sintaxis siguiente, y permiten al usuario crear una lista ordenada (tags$ol(); es decir, numerada) o desordenada (tags$ul(), es decir, con viñetas). tags$li() se utiliza para denotar los elementos de la lista, independientemente del tipo de lista que se utilice. p. ej:
* br() y hr() - estas etiquetas crean saltos de línea y líneas horizontales (con un salto de línea) respectivamente. Utilízalas para separar las secciones de tu aplicación y el texto. No es necesario pasar ningún elemento a estas etiquetas (los paréntesis pueden permanecer vacíos).
* div() - esta es una etiqueta genérica que puede contener cualquier cosa, y puede tener cualquier nombre. Una vez que avances en el diseño de la interfaz de usuario, puedes utilizarlas para compartimentar tu interfaz de usuario, dar estilos específicos a determinadas secciones y crear interacciones entre el servidor y los elementos de la interfaz de usuario. No vamos a entrar en detalles, pero vale la pena conocerlos.

Ten en cuenta que se puede acceder a cada uno de estos objetos a través de tags$... o para algunos, sólo la función. Estos son efectivamente sinónimos, pero puede ayudar a utilizar el estilo tags$... si prefiere ser más explícito y no sobrescribir las funciones accidentalmente. Esta no es en absoluto una lista exhaustiva de etiquetas disponibles. Hay una lista completa de todas las etiquetas disponibles en shiny [aquí](https://shiny.rstudio.com/articles/tag-glossary.html) e incluso se pueden utilizar más insertando HTML directamente en su ui!

Si te sientes seguro, también puedes añadir cualquier elemento de estilo css a tus etiquetas HTML con el argumento style en cualquiera de ellas. No vamos a entrar en detalles sobre cómo funciona esto, pero un consejo para probar los cambios estéticos en una interfaz de usuario es utilizar el modo de inspector de HTML en Chrome (de tu aplicación Shiny que está ejecutando en el navegador), y editar el estilo de los objetos tu mismo!

Vamos a añadir algo de texto a nuestra aplicación

### Añadir un enlace

Para añadir un enlace a una página web, utiliza tags$a() con el enlace y el texto a mostrar como se muestra a continuación. Para tener como un párrafo independiente, póngalo dentro de p(). Para tener sólo algunas palabras de una frase enlazada, divida la frase en partes y utiliza tags$a() para la parte hipervinculada. Para que el enlace se abra en una nueva ventana del navegador, añada target = "\_blank" como argumento.

### Añadir un botón de descarga

Pasemos a la segunda de las tres características. Un botón de descarga es una cosa bastante común para añadir a una aplicación y es bastante fácil de hacer. Tenemos que añadir otro Widget a nuestra ui, y tenemos que añadir otra salida a nuestro servidor para adjuntarlo. También podemos introducir conductores reactivos en este ejemplo!

Vamos a actualizar nuestra interfaz de usuario primero - esto es fácil ya que Shiny viene con un widget llamado downloadButton() - vamos a darle un inputId y una etiqueta.

Observe que también hemos añadido una etiqueta hr() - esto añade una línea horizontal que separa nuestros widgets de control de nuestros widgets de descarga. Esta es otra de las etiquetas HTML que hemos discutido anteriormente.

Ahora que tenemos nuestra ui lista, necesitamos añadir el componente del servidor. Las descargas se realizan en el servidor con la función downloadHandler(). De manera similar a nuestra trama, necesitamos adjuntarla a una salida que tenga el mismo inputId que el botón de descarga. Esta función toma dos argumentos - nombre del archivo y contenido - ambos son funciones. Como podrás adivinar, filename se utiliza para especificar el nombre del archivo descargado, y content se utiliza para especificar lo que debe ser descargado. content contiene una función que usarías para guardar los datos localmente - así que si estuvieras descargando un archivo csv podrías usar rio::export(). Como estamos descargando un gráfico, usaremos ggplot2::ggsave(). Veamos cómo programaríamos esto (aún no lo añadiremos al servidor).

Observe que la función de contenido siempre toma un argumento de archivo, que ponemos donde se especifica el nombre del archivo de salida. También puedes notar que estamos repitiendo código aquí - estamos usando nuestra función plot\_epicurve() dos veces en este servidor, una para la descarga y otra para la imagen mostrada en la aplicación. Aunque esto no afecta masivamente al rendimiento, significa que el código para generar este gráfico tendrá que ejecutarse cuando el usuario cambie los widgets que especifican el distrito y el grupo de edad, y de nuevo cuando quiera descargar el gráfico. En aplicaciones más grandes, decisiones subóptimas como ésta ralentizarán cada vez más las cosas, así que es bueno aprender a hacer nuestra aplicación más eficiente en este sentido. Lo que tendría más sentido es si tuviéramos una forma de ejecutar el código de la epicurva cuando los distritos/grupos de edad son cambios, y dejar que eso sea utilizado por las funciones renderPlot() y downloadHandler(). Aquí es donde entran los conductores reactivos!

Los conductores reactivos son objetos que se crean en el servidor shiny de forma reactiva, pero no se emiten - sólo pueden ser utilizados por otras partes del servidor. Hay varios tipos de conductores reactivos, pero vamos a repasar los dos básicos.

1.reactive() - este es el conductor reactivo más básico - reaccionará siempre que cualquier entrada utilizada dentro de él cambie (por lo que nuestros widgets de distrito/grupo de edad)  
2. eventReactive()- este conductor rectivo funciona igual que reactive(), excepto que el usuario puede especificar qué entradas hacen que se vuelva a ejecutar. Esto es útil si tu conductor reactivo tarda mucho en procesar, pero esto se explicará más adelante.

Veamos los dos ejemplos:

Cuando usamos la configuración de eventReactive(), podemos especificar qué entradas hacen que se ejecute este trozo de código - esto no nos es muy útil por el momento, así que podemos dejarlo por ahora. Ten en cuenta que puede incluir múltiples entradas con c()

Veamos cómo podemos integrar esto en el código de nuestro servidor:

Puedes ver que sólo estamos llamando a la salida de nuestro reactivo que hemos definido en nuestras funciones de descarga y representación gráfica. Una cosa que hay que tener en cuenta y que suele confundir a la gente es que hay que utilizar las salidas de los reactivos como si fueran funciones, por lo que hay que añadir paréntesis vacíos al final de los mismos (es decir, malaria\_plot() es correcto, y malaria\_plot no lo es). Ahora que hemos añadido esta solución nuestra aplicación es un poco más ordenada, más rápida y más fácil de cambiar ya que todo nuestro código que ejecuta la función epicurve está en un solo lugar.

### Añadir un selector de instalaciones

Pasemos a nuestra siguiente función: un selector para instalaciones específicas. Implementaremos otro parámetro en nuestra función para poder pasarlo como argumento desde nuestro código. Vamos a ver cómo hacer esto primero - sólo funciona con los mismos principios que los otros parámetros que hemos establecido. Actualicemos y probemos nuestra función.

plot\_epicurve <- function(data, district = "All", ageegroup = "malaria\_tot", facility = "All") {

if (! ("All" %in% district)) {

datos <- datos %>%

filter(Distrito %en% distrito)

plot\_title\_district <- stringr::str\_glue("{paste0(district, collapse = ', ')} districts")

} si no {

plot\_title\_district <- "todos los distritos"

}

# si no hay datos restantes, devuelve NULL

si (nrow(data) == 0) {

return(NULL)

}

datos <- datos %>%

filter(grupo\_de\_edad == grupo\_de\_edad)

# si no hay datos restantes, devuelve NULL

si (nrow(data) == 0) {

return(NULL)

}

if (agegroup == "malaria\_tot") {

agegroup\_title <- "Todas las edades"

} si no {

agegroup\_title <- stringr::str\_glue("{str\_remove(agegroup, 'malaria\_rdt')} years")

}

if (! ("All" %in% facility)) {

datos <- datos %>%

filter(nombre\_de\_localización == instalación)

plot\_title\_facility <- facility

} si no {

plot\_title\_facility <- "todas las instalaciones"

}

# si no hay datos restantes, devuelve NULL

si (nrow(data) == 0) {

return(NULL)

}

ggplot(data, aes(x = data\_date, y = cases\_reported)) +

geom\_col(anchura = 1, relleno = "rojo oscuro") +

theme\_minimal() +

laboratorios(

x = "fecha",

y = "número de casos",

title = stringr::str\_glue("Casos de malaria - {plot\_title\_district}; {plot\_title\_facility}"),

subtítulo = ageegroup\_title

)

}

Vamos a probarlo:

Con todas las instalaciones en nuestros datos, no está muy claro qué instalaciones corresponden a qué distritos, y el usuario final tampoco lo sabrá. Esto puede hacer que el uso de la aplicación sea poco intuitivo. Por esta razón, debemos hacer que las opciones de instalaciones en la interfaz de usuario cambien dinámicamente a medida que el usuario cambia de distrito, de modo que una filtra a la otra. Dado que tenemos tantas variables que estamos utilizando en las opciones, también podríamos querer generar algunas de nuestras opciones para la ui en nuestro archivo global.R a partir de los datos. Por ejemplo, podemos añadir este trozo de código a global.R después de haber leído nuestros datos:

Vamos a verlos:

Podemos pasar estas nuevas variables a la ui sin ningún problema, ya que son visibles globalmente tanto por el servidor como por la ui. Actualicemos nuestra UI:

Fíjate en que ahora pasamos variables para nuestras elecciones en lugar de codificarlas en la interfaz de usuario. Esto también puede hacer que nuestro código sea más compacto. Por último, tendremos que actualizar el servidor. Será fácil actualizar nuestra función para incorporar nuestra nueva entrada (sólo tenemos que pasarla como argumento a nuestro nuevo parámetro), pero debemos recordar que también queremos que la ui se actualice dinámicamente cuando el usuario cambie el distrito seleccionado. Es importante entender aquí que podemos cambiar los parámetros y el comportamiento de los widgets mientras la aplicación se está ejecutando, pero esto debe hacerse en el servidor. Tenemos que entender una nueva forma de salida al servidor para aprender a hacer esto.

Las funciones que necesitamos para entender cómo hacer esto se conocen como funciones de observador, y son similares a las funciones reactivas en cuanto a su comportamiento. Sin embargo, tienen una diferencia clave:

* Las funciones reactivas no afectan directamente a las salidas, y producen objetos que pueden verse en otros lugares del servidor
* Las funciones de los observadores pueden afectar a las salidas del servidor, pero lo hacen a través de los efectos secundarios de otras funciones. (También pueden hacer otras cosas, pero esta es su función principal en la práctica)

Al igual que las funciones reactivas, hay dos tipos de funciones de observador, y se dividen por la misma lógica que divide las funciones reactivas:

1. observe() - esta función se ejecuta cada vez que cambian las entradas utilizadas dentro de ella
2. observeEvent() - esta función se ejecuta cuando cambia una entrada especificada por el usuario

También necesitamos entender las funciones proporcionadas por Shiny que actualizan los widgets. Estas son bastante sencillas de ejecutar - primero toman el objeto de sesión de la función del servidor (esto no necesita ser entendido por ahora), y luego el inputId de la función a ser cambiada. Luego pasamos las nuevas versiones de todos los parámetros que ya son tomados por selectInput() - estos serán actualizados automáticamente en el widget.

Veamos un ejemplo aislado de cómo podríamos utilizar esto en nuestro servidor. Cuando el usuario cambia de distrito, queremos filtrar nuestra lista de instalaciones por distrito, y actualizar las opciones para que sólo reflejen las que están disponibles en ese distrito (y una opción para todas las instalaciones)

Y ya está, podemos añadirlo a nuestro servidor, y ese comportamiento ya funcionará. Este es el aspecto que debería tener nuestro nuevo servidor:

### Añadir otra pestaña con una tabla

Ahora pasaremos al último componente que queremos añadir a nuestra aplicación. Querremos separar nuestra ui en dos pestañas, una de las cuales tendrá una tabla interactiva donde el usuario podrá ver los datos con los que está haciendo la curva epidémica. Para ello, podemos utilizar los elementos de ui empaquetados que vienen con shiny relevantes para las pestañas. En un nivel básico, podemos encerrar la mayor parte de nuestro panel principal en esta estructura general:

Apliquemos esto a nuestra ui. También vamos a querer utilizar el paquete **DT** aquí - este es un gran paquete para hacer tablas interactivas a partir de datos preexistentes. Podemos ver que se utiliza para DT::datatableOutput() en este ejemplo.

Ahora nuestra aplicación está organizada en pestañas! Hagamos también las modificaciones necesarias en el servidor. Dado que no necesitamos manipular nuestro conjunto de datos antes de renderizarlo, esto es muy sencillo: ¡sólo tenemos que renderizar los datos malaria\_data a través de DT::renderDT() en la interfaz de usuario!

## Compartir aplicaciones Shinys

Ahora que has desarrollado tu aplicación, probablemente quieras compartirla con los demás, ¡al fin y al cabo esta es la principal ventaja de shiny! Podemos hacerlo compartiendo el código directamente, o podemos publicarlo en un servidor. Si compartimos el código, otros podrán ver lo que has hecho y construir sobre él, pero esto anulará una de las principales ventajas de shiny: puede eliminar la necesidad de que los usuarios finales mantengan una instalación de R. Por esta razón, si estás compartiendo tu aplicación con usuarios que no se sienten cómodos con R, es mucho más fácil compartir una aplicación que ha sido publicada en un servidor.

Si prefieres compartir el código, puedes hacer un archivo .zip de la aplicación, o mejor aún, publicar tu aplicación en github y añadir colaboradores. Puedes consultar la sección de github para más información aquí.

Sin embargo, si vamos a publicar la aplicación en línea, tenemos que hacer un poco más de trabajo. En última instancia, queremos que se pueda acceder a tu aplicación a través de una URL web para que otros puedan acceder a ella de forma rápida y sencilla. Desafortunadamente, para publicar tu aplicación en un servidor, necesitas tener acceso a un servidor donde publicarla. Hay varias opciones de alojamiento en este sentido:

* shinyapps.io: es el lugar más sencillo para publicar aplicaciones shiny, ya que es el que menos trabajo de configuración necesita, y tiene algunas licencias gratuitas, pero limitadas.
* RStudio Connect: es una versión mucho más potente de un servidor de R, que puede realizar muchas operaciones, incluida la publicación de aplicaciones Shinys. Sin embargo, es más difícil de usar y menos recomendable para los usuarios noveles.

Para los propósitos de este documento, utilizaremos shinyapps.io, ya que es más fácil para los usuarios de primera vez. Puedes hacer una cuenta gratuita aquí para empezar - también hay diferentes planes de precios para los licesnses de los servidores si es necesario. Cuantos más usuarios esperes tener, más caro tendrá que ser tu plan de precios, así que tenlo en cuenta. Si quieres crear algo para un pequeño grupo de personas, una licencia gratuita puede ser perfectamente adecuada, pero una aplicación de cara al público puede necesitar más licencias.

Primero debemos asegurarnos de que nuestra aplicación es adecuada para publicar en un servidor. En tu aplicación, debes reiniciar tu sesión de R, y asegurarte de que se ejecuta sin ejecutar ningún código extra. Esto es importante, ya que una aplicación que requiere la carga de paquetes, o la lectura de datos no definidos en el código de tu aplicación no se ejecutará en un servidor. También ten en cuenta que no puedes tener rutas de archivo explícitas en tu aplicación - éstas serán inválidas en la configuración del servidor - el uso del paquete here resuelve muy bien este problema. Por último, si estás leyendo datos de una fuente que requiere autenticación de usuario, como los servidores de tu organización, esto no funcionará generalmente en un servidor. Tendrás que ponerte en contacto con tu departamento de TI para averiguar cómo poner en la lista blanca el Shiny servidor here.

registro de la cuenta

Una vez que tengas tu cuenta, puedes navegar a la página de tokens en Cuentas. Aquí querrás añadir un nuevo token, que se utilizará para desplegar tu aplicación.

A partir de aquí, debes tener en cuenta que la url de tu cuenta reflejará el nombre de tu app - así que si tu app se llama mi\_app, la url se añadirá como xxx.io/mi\_app/. Elige bien el nombre de tu aplicación. Ahora que está todo listo, clica en desplegar - si tiene éxito esto ejecutará su aplicación en la url web elegida.

¿algo sobre la creación de aplicaciones en documentos?

## Más información

Hasta ahora, hemos cubierto muchos aspectos de shiny, y apenas hemos arañado la superficie de lo que ofrece shiny. Aunque esta guía sirve de introducción, hay mucho más que aprender para entender completamente shiny. Deberías empezar a crear aplicaciones y añadir gradualmente más y más funcionalidad

## Paquetes de extensión recomendados

A continuación se presenta una selección de extensiones de brillo de alta calidad que pueden ayudarle a sacar mucho más provecho del brillo. Sin ningún orden en particular:

* **shinyWidgets** - este paquete ofrece muchos más widgets que pueden ser utilizados en tu aplicación. Ejecuta shinyWidgets::shinyWidgetsGallery() para ver una selección de los widgets disponibles con este paquete. Vea los ejemplos [aquí](https://github.com/dreamRs/shinyWidgets)
* **shinyjs** - este es un excelente paquete que da al usuario la capacidad de ampliar en gran medida la utilidad de shiny a través de una serie de javascript. Las aplicaciones de este paquete van desde las más sencillas hasta las más avanzadas, pero es posible que quieras utilizarlo primero para manipular la interfaz de usuario de forma sencilla, como ocultar/mostrar elementos, o activar/desactivar botones. Para más información[, consulte](https://deanattali.com/shinyjs/basic)
* **shinydashboard** - este paquete expande masivamente la ui disponible que puede ser usada en shiny, específicamente permitiendo al usuario crear un dashboard complejo con una variedad de diseños complejos. Vea más [aquí](https://rstudio.github.io/shinydashboard/)
* **shinydashboardPlus**: ¡obtenga aún más funciones del marco de trabajo **de shinydashboard**! Vea más [aquí](https://rinterface.github.io/shinydashboardPlus/articles/shinydashboardPlus.html)
* **shinythemes** - ¡cambia el tema css por defecto de tu shiny app con una amplia gama de plantillas preestablecidas! Vea más [aquí](https://rstudio.github.io/shinythemes/)

También hay una serie de paquetes que pueden utilizarse para crear resultados interactivos compatibles con Shiny.

* **DT** está semi-incorporado en base-shiny, pero proporciona un gran conjunto de funciones para crear tablas interactivas.
* **plotly** es un paquete para crear gráficos interactivos que el usuario puede manipular en la aplicación. También puede convertir sus gráficos en versiones interactivas mediante plotly::ggplotly(). Como alternativas, **dygraphs** y **highcharter** son también excelentes.

## Recursos recomendados

# Miscelánea

# #Funciones de escritura

{#writing-functions-1}

## Preparación

### Cargar paquetes

Este trozo de código muestra la carga de los paquetes necesarios para los análisis. En este manual destacamos p\_load() de **pacman**, que instala el paquete si es necesario y lo carga para su uso. También puede cargar los paquetes instalados con library() de R **base.** Consulte la página sobre [los fundamentos de R](#r-basics) para obtener más información sobre los paquetes de R.

### Importar datos

Importamos los datos de casos de una epidemia de ébola simulada. Si desea descargar los datos para seguirlos paso a paso, consulte las instrucciones en la página [Descargar libro y datos]. Los datos se importan mediante la función import() del paquete **rio**. Consulte la página sobre [importación y exportación](#import-and-export) para conocer las distintas formas de importar datos.

También utilizaremos en la última parte de esta página algunos datos sobre la gripe H7N9 de 2013.

## Funciones

Las funciones son útiles en la programación, ya que permiten hacer códigos más fáciles de entender, de alguna manera más cortos y menos propensos a errores (dado que no hay errores en la propia función).

Si has llegado hasta este manual, significa que te has encontrado con un sinfín de funciones ya que en R, cada operación es una llamada a una función +, for, if, [, $, { .... Por ejemplo x + y es lo mismo que'+'(x, y)

R es uno de los lenguajes que más posibilidades ofrece para trabajar con funciones y da suficientes herramientas al usuario para escribirlas fácilmente. No debemos pensar en las funciones como algo fijo en la cima o al final de la cadena de programación, R ofrece la posibilidad de utilizarlas como si fueran vectores e incluso utilizarlas dentro de otras funciones, listas...

Existen muchos recursos muy avanzados sobre programación funcional y aquí sólo daremos una visión para ayudarte a empezar con la programación funcional con breves ejemplos prácticos. A continuación, te animamos a visitar los enlaces de las referencias para leer más sobre el tema.

## ¿Por qué utilizar una función?

Antes de responder a esta pregunta, es importante tener en cuenta que ya has tenido consejos para llegar a escribir tus primeras funciones R en la página sobre [Iteración, bucles y listas](#iteration-loops-and-lists) de este manual. De hecho, el uso de "if/else" y bucles suele ser una parte fundamental de muchas de nuestras funciones, ya que ayudan fácilmente a ampliar la aplicación de nuestro código permitiendo múltiples condiciones o a iterar códigos para repetir tareas.

* ¿Estoy repitiendo varias veces el mismo bloque de código para aplicarlo a una variable o dato diferente?
* Deshacerse de él, ¿acortará sustancialmente mi código general y hará que se ejecute más rápido?
* ¿Es posible que el código que he escrito se utiliza de nuevo pero con un valor diferente en muchos lugares del código?

Si la respuesta a una de las preguntas anteriores es "SÍ", es probable que tenga que escribir una función

## ¿Cómo construye R las funciones?

Las funciones en R tienen tres componentes principales:

* los formales() que es la lista de argumentos que controla cómo podemos llamar a la función
* el cuerpo() que es el código dentro de la función, es decir, dentro de los paréntesis o después del paréntesis, dependiendo de cómo lo escribamos

y,

* el entorno() que ayudará a localizar las variables de la función y determina cómo la función encuentra el valor.

Una vez que hayas creado tu función, puedes verificar cada uno de estos componentes llamando a la función asociada.

## Sintaxis y estructura básicas

* Una función tendrá que ser nombrada adecuadamente para que su trabajo sea fácilmente comprensible tan pronto como leamos su nombre. En realidad, este es el caso de la mayoría de la arquitectura básica de R. Funciones como mean(), print(), summary() tienen nombres muy sencillos
* Una función necesitará argumentos, como los datos sobre los que trabajar y otros objetos que pueden ser valores estáticos entre otras opciones
* Y finalmente una función dará una salida basada en su tarea principal y en los argumentos que se le han dado. Normalmente utilizaremos las funciones incorporadas como print(), return()... para producir la salida. La salida puede ser un valor lógico, un número, un carácter, un dataframe... en definitiva cualquier tipo de objeto de R.

Básicamente se trata de la composición de una función:

Podemos crear nuestra primera función que se llamará contain\_covid19().

A continuación, podemos verificar los componentes de nuestra función recién creada.

Ahora vamos a probar nuestra función. Para llamar a nuestra función escrita, la usas como usas todas las funciones de R, es decir, escribiendo el nombre de la función y añadiendo los argumentos necesarios.

Podemos volver a escribir el nombre de cada argumento por precaución. Pero sin especificarlos, el código debería funcionar ya que R tiene en memoria la posición de cada argumento. Así que mientras pongas los valores de los argumentos en el orden correcto, puedes omitir escribir los nombres de los argumentos al llamar a las funciones.

A continuación, veamos qué ocurre si uno de los valores es "no" o **no "**sí".

Si proporcionamos un argumento que no es reconocido, obtenemos un error:

Error en contain\_covid19(barrier\_gest = "sometimes", wear\_mask = "yes", : no se pudo encontrar la función "contain\_covid19"

**NOTA:** Algunas funciones (la mayoría de las veces muy cortas y sencillas) pueden no necesitar un nombre y pueden ser utilizadas directamente en una línea de código o dentro de otra función para realizar una tarea rápida. Se llaman **funciones anónimas**.

Por ejemplo, a continuación se muestra una primera función anónima que mantiene sólo las variables de carácter de los datos.

A continuación, otra función que selecciona una de cada dos observaciones de nuestro conjunto de datos (puede ser relevante cuando tenemos datos longitudinales con muchos registros por paciente, por ejemplo, después de haber ordenado por fecha o visita). En este caso, la función adecuada que se escribe fuera de dplyr sería function (x) (x%2 == 0) para aplicarla al vector que contiene todos los números de fila.

Un posible código R base para la misma tarea sería:

**PRECAUCIÓN:** Aunque es cierto que el uso de funciones puede ayudarnos con nuestro código, puede llevar mucho tiempo escribir algunas funciones o arreglar una si no ha sido pensada a fondo, escrita adecuadamente y está devolviendo errores como resultado. Por esta razón, a menudo se recomienda escribir primero el código en R, asegurarse de que hace lo que pretendemos, y luego transformarlo en una función con sus tres componentes principales, como se ha indicado anteriormente.

## Ejemplos

### Devuelve tablas de proporciones para varias columnas

Sí, ya disponemos de bonitas funciones en muchos paquetes que permiten resumir la información de una manera muy fácil y agradable. Pero aún así intentaremos hacer las nuestras, en nuestros primeros pasos para acostumbrarnos a escribir funciones.

En este ejemplo queremos mostrar cómo la escritura de una función simple le evitaría copiar y pegar el mismo código varias veces.

**CONSEJO:** Como se ha indicado anteriormente, es muy importante comentar las funciones como se haría en la programación general. Ten en cuenta que el objetivo de una función es hacer un código fácil de leer, más corto y más eficiente. Entonces uno debería ser capaz de entender lo que hace la función con sólo leer su nombre y debería tener más detalles leyendo los comentarios.

Una segunda opción es utilizar esta función en otra a través de un bucle para hacer el proceso a la vez:

Una forma más sencilla podría ser utilizar la base R "apply" en lugar de un "bucle for" como se expresa a continuación:

**CONSEJO:** R se define a menudo como un lenguaje de programación funcional y casi siempre que ejecutas una línea de código estás utilizando algunas funciones incorporadas. Un buen hábito para sentirse más cómodo con la escritura de funciones es echar a menudo un vistazo interno a cómo están construidas las funciones básicas que utiliza a diario. El atajo para hacerlo es seleccionar el nombre de la función y luego clicar en Ctrl+F2 o fn+F2 o Cmd+F2 (dependiendo de tu ordenador) .

## Uso de ****purrr****: escribir funciones que se pueden aplicar de forma iterativa

### Modificar el tipo de varias columnas en unos datos

Digamos que muchas variables de carácter en los datos originales del listado necesitan ser cambiadas a "factor" para propósitos de análisis y trazado. En lugar de repetir el paso varias veces, podemos utilizar simplemente lapply() para realizar la transformación de todas las variables afectadas en una sola línea de código.

**ATENCIÓN:** lapply() devuelve una lista, por lo que su uso puede requerir una modificación adicional como último paso.

El mismo paso puede realizarse utilizando la función map\_if() del paquete **purrr**

### Elaborar de forma iterativa gráficos para diferentes niveles de una variable

Produciremos aquí un gráfico circular para ver la distribución del resultado de los pacientes en China durante el brote de H7N9 para cada provincia. En lugar de repetir el código para cada una de ellas, nos limitaremos a aplicar una función que crearemos.

### Producir iterativamente tablas para diferentes niveles de una variable

Aquí crearemos tres indicadores para resumirlos en una tabla y nos gustaría elaborar esta tabla para cada una de las provincias. Nuestros indicadores son el retraso entre el inicio y la hospitalización, el porcentaje de recuperación y la edad media de los casos.

## Consejos y mejores prácticas para el buen funcionamiento de las funciones

La programación funcional está pensada para aliviar el código y facilitar su lectura. Debería producir lo contrario. Los siguientes consejos le ayudarán a tener un código limpio y fácil de leer.

### Nombres y sintaxis

* Evitar el uso de caracteres que podrían haber sido fácilmente tomados por otras funciones ya existentes en su entorno
* Se recomienda que el nombre de la función sea corto y sencillo de entender para otro lector
* Es preferible utilizar verbos como nombre de la función y sustantivos para los nombres de los argumentos.

### Nombres de columnas y evaluación ordenada

Si quiere saber cómo referenciar nombres de columnas que se proporcionan a su código como argumentos, lea esta [guía de programación de tidyverse](https://dplyr.tidyverse.org/articles/programming.html). Entre los temas tratados están la evaluación ordenada y el uso del abrazo {{ }} "llaves dobles"

Por ejemplo, aquí hay un esqueleto rápido de código de plantilla del tutorial de la página mencionada anteriormente:

### Pruebas y tratamiento de errores

Cuanto más complicada sea la tarea de una función, mayor será la posibilidad de errores. Por lo tanto, a veces es necesario añadir alguna verificación dentro de la función para ayudar a entender rápidamente de dónde proviene el error y encontrar una manera de solucionarlo.

* Puede ser más que recomendable introducir una comprobación sobre la ausencia de un argumento utilizando missing(argumento). Esta simple comprobación puede devolver el valor "TRUE" o "FALSE".
* Utiliza stop() para errores más detectables.
* Como vemos cuando ejecutamos la mayoría de las funciones incorporadas, hay mensajes y advertencias que pueden aparecer en ciertas condiciones. Podemos integrarlos en nuestras funciones escritas utilizando las funciones message() y warning().
* Podemos manejar los errores también usando safely() que toma una función como argumento y la ejecuta de forma segura. De hecho, la función se ejecutará sin detenerse si encuentra un error. safely() devuelve como salida una **lista** con dos objetos que son los resultados y el error que se ha "saltado".

Podemos verificarlo ejecutando primero la media() como función, y luego ejecutarla con safely().

Como se ha dicho anteriormente, comentar bien nuestros códigos ya es una buena forma de tener documentación en nuestro trabajo.

## Recursos

[Enlace a R for Data Science](https://r4ds.had.co.nz/functions.html)

[Cheatsheet advance R programming](https://www.rstudio.com/wp-content/uploads/2016/02/advancedR.pdf)

[Paquete Cheatsheet purr](https://purrr.tidyverse.org/)

[Vídeo-ACM charla de Hadley Wickham: La alegría de la programación funcional (cómo funciona map\_dbl)](https://youtube.videoken.com/embed/bzUmK0Y07ck)

# #Interacciones con directorios

{#directory-interactions}

En esta página cubrimos los escenarios comunes en los que se crea, se interactúa, se guarda y se importa con directorios (carpetas).

## Preparación

### paquete ****fs****

El paquete fs es un paquete **tidyverse** que facilita las interacciones con los directorios, mejorando algunas de las funciones de R **base**. En las secciones siguientes utilizaremos a menudo funciones de **fs**.

### Imprimir el directorio como un árbol de dendrogramas

Utiliza la función dir\_tree() de **fs**.

Proporcione la ruta de la carpeta a path = y decida si quiere mostrar sólo un nivel (recurse = FALSE) o todos los archivos en todos los subniveles (recurse = TRUE). A continuación utilizamos here() como abreviatura del proyecto R y especificamos su subcarpeta "data", que contiene todos los datos utilizados para este manual de R. Lo configuramos para que muestre todos los archivos dentro de "data" y sus subcarpetas (por ejemplo, "cache", "epidemic models", "population", "shp" y "weather").

## Listar los archivos de un directorio

Para listar sólo los nombres de los archivos de un directorio puede utilizar dir() desde **la base de R.** Por ejemplo, este comando lista los nombres de los archivos de la subcarpeta "population" de la carpeta "data" en un proyecto R. La ruta relativa de los archivos se proporciona utilizando here() (sobre la que puede leer más en la página de [importación y exportación](#import-and-export)).

Para listar las rutas completas de los archivos del directorio, puede utilizar dir\_ls() de **fs**. Una alternativa **básica** de R es list.files().

Para obtener toda la información de los metadatos de cada archivo en un directorio, (por ejemplo, la ruta, la fecha de modificación, etc.) puede utilizar dir\_info() de **fs**.

Esto puede ser especialmente útil si quiere extraer la última hora de modificación del archivo, por ejemplo si quiere importar la versión más reciente de un archivo. Para ver un ejemplo de esto, consulte la página de [importación y exportación](#import-and-export).

Aquí está el dataframe devuelto. Desplácese a la derecha para ver todas las columnas.

## Información sobre el archivo

Para extraer información de metadatos sobre un archivo específico, puede utilizar file\_info() de **fs** (o file.info() de R **base**).

Aquí usamos el $ para indexar el resultado y devolver sólo el valor de modification\_time.

## Comprobar si existe

### Objetos R

Puedes utilizar exists() de R **base** para comprobar si un objeto R existe dentro de R (proporcione el nombre del objeto entre comillas).

Ten en cuenta que algunos paquetes **base de** R utilizan nombres de objetos genéricos como "data" entre bastidores, que aparecerán como TRUE a menos que se especifique inherit = FALSE. Esta es una razón para no nombrar su conjunto de datos como "data".

Si está escribiendo una función, debería utilizar missing() de R **base** para comprobar si un argumento está presente o no, en lugar de exists().

### Directorios

Para comprobar si un directorio existe, proporcione la ruta del archivo (y el nombre del archivo) a is\_dir() de **fs**. Desplácese a la derecha para ver que se imprime TRUE.

Una alternativa es file.exists() de la **base** R.

### Archivos

Para comprobar si un archivo específico existe, utiliza is\_file() de **fs**. Desplácese a la derecha para ver que se imprime TRUE.

Una alternativa **básica** de R es file.exists().

## Crear

### Directorios

Para crear un nuevo directorio (carpeta) puede utilizar dir\_create() de **fs**. Si el directorio ya existe, no se sobrescribirá y no se devolverá ningún error.

Una alternativa es dir.create() de la **base** R, que mostrará un error si el directorio ya existe. En cambio, dir\_create() en este escenario será silencioso.

### Archivos

Puedes crear un archivo (vacío) con file\_create() de **fs**. Si el archivo ya existe, no se sobreescribirá ni se modificará.

Una alternativa **básica** de R es file.create(). Pero si el archivo ya existe, esta opción lo truncará. Si se utiliza file\_create() el archivo se dejará sin cambios.

### Crear si no existe

EN CONSTRUCCIÓN

## Borrar

### Objetos R

Utiliza rm() desde **la base** R para eliminar un objeto R.

### Directorios

Utiliza dir\_delete() de **fs**.

### Archivos

Puedes eliminar archivos con file\_delete() de **fs**.

## Ejecución de otros archivos

### fuente()

Para ejecutar un script de R desde otro script de R, puede utilizar el comando source() (de R **base**).

Esto equivale a ver el script de R anterior y clicar en el botón "Source" en la parte superior derecha del script. Esto ejecutará el script pero lo hará de forma silenciosa (sin salida a la consola de R) a menos que se pretenda específicamente. Consulte la página sobre [Consola interactiva] para ver ejemplos de uso de source() para interactuar con un usuario a través de la consola de R en modo de pregunta y respuesta.

### renderizar()

render() es una variación de source() que se utiliza más a menudo para los scripts de R markdown. Tu proporcionas la entrada = que es el archivo R markdown, y también el formato\_de\_salida = (típicamente "documento\_html", "documento\_pdf", "documento\_palabra", "")

Vea la página sobre [Informes con R Markdown](#reports-with-r-markdown) para más detalles. También consulte la documentación de render() [aquí](https://rmarkdown.rstudio.com/docs/reference/render.html) o introduciendo ?render.

### Ejecutar archivos en un directorio

Puedes crear un bucle for y utilizarlo para source() cada archivo en un directorio, como se identifica con dir().

Si sólo quieres ejecutar determinados scripts, puedes identificarlos por su nombre de la siguiente manera:

A continuación, una [comparación](https://cran.r-project.org/web/packages/fs/vignettes/function-comparisons.html) de las funciones **fs** y **base** R.

### Importar archivos en un directorio

Consulte la página sobre [importación y exportación](#import-and-export) para importar y exportar archivos individuales.

Consulte también la página de [importación y exportación](#import-and-export) para conocer los métodos para importar automáticamente el archivo más reciente, basándose en una fecha del nombre del archivo o mirando los metadatos del mismo.

Consulte la página sobre [Iteración, bucles y listas](#iteration-loops-and-lists) para ver un ejemplo con el paquete **purrr** demostrando:

* Dividir un dataframe y guardarlo como múltiples archivos CSV
* Dividir un dataframe y guardar cada parte como una hoja separada dentro de un libro de Excel
* Importar varios archivos CSV y combinarlos en un dataframe
* Importar un libro de Excel con varias hojas y combinarlas en un dataframe

## ****base**** R

Vea a continuación las funciones list.files() y dir(), que realizan la misma operación de listar archivos dentro de un directorio especificado. Puedes especificar ignore.case = o un patrón específico para buscar.

Si un archivo está actualmente "abierto", se mostrará en su carpeta con una tilde delante, como "~$hospital\_linelists.xlsx".

## Recursos

<https://cran.r-project.org/web/packages/fs/vignettes/function-comparisons.html>

# #Control de versiones y colaboración con Git y Github

{#version-control-and-collaboration-with-git-and-github}

Este capítulo presenta una visión general del uso de Git para colaborar con otros. Puedes encontrar tutoriales más extensos al final en la sección de Recursos.

## ¿Qué es Git?

Git es un software de **control de versiones** que permite seguir los cambios realizados en una carpeta. Se puede utilizar como la opción "control de cambios" de Word, LibreOffice o Google docs, pero para todo tipo de archivos. Es una de las opciones más potentes y más utilizadas para el control de versiones.

**¿Por qué nunca he oído hablar de Git? -** Mientras que las personas con formación como desarrollador aprenden habitualmente a utilizar un software de control de versiones (Git, Mercurial, Subversion u otros), a pocas personas de las disciplinas cuantitativas se nos enseñan estas habilidades. En consecuencia, la mayoría de profesionales de la epidemiología nunca han oído hablar sobre esto en sus estudios, y tienen que aprenderlo sobre la marcha.

**Espera, he oído hablar de Github, ¿es lo mismo?** - No exactamente, pero a menudo se utilizan juntos, y veremos aquí cómo hacerlo. En resumen:

* **Git** es el sistema de control de versiones, una pieza de software. Se puede utilizar localmente en el ordenador o para sincronizar una carpeta con un **sitio web** anfitrión. Por defecto, se utiliza una ventana de terminal para escribir las instrucciones de Git en la línea de comandos.
* Se puede utilizar un **cliente/interfaz Git** para evitar la línea de comandos y realizar las mismas acciones (al menos para las más sencillas y supercomunes).
* Si se quiere almacenar una carpeta en un **sitio web** para colaborar con otros, se puede utilizar una cuenta en Github, Gitlab, Bitbucket u otros.

Así se puede utilizar el cliente/interfaz **Github Desktop**, que utiliza **Git** en segundo plano para gestionar los archivos, tanto localmente en el ordenador, como remotamente en un servidor de **Github**.

## ¿Por qué utilizar la combinación de Git y Github?

El uso de **Git** facilita:

1. Archivar versiones documentadas con cambios incrementales de forma que permite volver fácilmente a cualquier estado anterior
2. Tener ramas paralelas, es decir, versiones de desarrollo/"trabajo" con formas estructuradas de integrar los cambios después de la revisión

Esto también se puede hacer localmente en tu ordenador, incluso si no colaboras con otras personas. Alguna vez ….:

* ¿te has arrepentido de haber eliminado una sección de código, para darte cuenta dos meses después de que realmente la necesitabas?
* ¿has vuelto a un proyecto que había estado en pausa e intentado recordar si habías hecho esa complicada modificación en uno de los modelos?
* ¿tenías un archivo modelo\_1.R y otro archivo modelo\_1\_prueba.R y un archivo modelo\_1\_no\_funciona.R para probar las cosas?
* ¿tenías un archivo report.Rmd, un archivo report\_full.Rmd, un archivo report\_true\_final.Rmd, un archivo report\_final\_20210304.Rmd, un archivo report\_final\_20210402.Rmd y maldecías tus habilidades de almacenamiento?

Git puede ayudar con todo eso, y vale la pena aprenderlo sólo por eso.

Sin embargo, se vuelve aún más potente cuando se utiliza con un repositorio en línea como Github para apoyar **proyectos de colaboración**. Esto facilita:

* Colaboración: otros pueden revisar, comentar y aceptar o rechazar los cambios
* Compartir el código, los datos y los resultados, e invitar a hacer comentarios al público (o en privado, con tu equipo)

y evitar:

* "Uy, me olvidé de enviar la última versión y ahora tienes que rehacer el trabajo de dos días en este nuevo archivo"
* Mina, Henry y Oumar trabajaron al mismo tiempo en un script y necesitan fusionar manualmente sus cambios
* Dos personas intentan modificar el mismo archivo en Dropbox y Sharepoint y esto crea un error de sincronización.

### Esto suena complicado, yo no soy un programador

Puede ser. Los ejemplos de usos avanzados pueden ser bastante aterradores. Sin embargo, al igual que ocurre con R, o incluso con Excel, no es necesario convertirse en un experto para aprovechar las ventajas de la herramienta. El aprendizaje de un pequeño número de funciones y nociones te permite seguir sus cambios, sincronizar los archivos en un repositorio en línea y colaborar con los colegas en muy poco tiempo.

Debido a la curva de aprendizaje, el contexto de emergencia puede no ser el mejor momento para aprender estas herramientas. Pero el aprendizaje puede hacerse por pasos. Una vez que adquieras un par de nociones, tu flujo de trabajo puede ser bastante eficiente y rápido. Si no estás trabajando en un proyecto en el que la colaboración con personas a través de Git sea una necesidad, ... **en realidad es un buen momento para adquirir confianza en su uso** en solitario antes de sumergirte en ello en un proyecto colaborativo.

## Configuración

### Instalar Git

Git es el motor detrás del escenario en la computadora, rastrea los cambios, las ramas (versiones), las fusiones y las reversiones. **Primero debes instalar Git desde https://git-scm.com/downloads.**

### Instalar una interfaz (opcional pero recomendable)

Git tiene su propio lenguaje de comandos, que se pueden escribir en la línea de comandos de un terminal. Sin embargo, hay muchos clientes/interfaces que proporcionan una buena visualización de las modificaciones de archivos o ramas. Esto es recomendable ya que personas que no son desarrolladoras, en su uso diario, rara vez necesitarán interactuar directamente con Git.

Existen muchas opciones, en todos los sistemas operativos, desde las amigables para los principiantes hasta las más complejas. Unas buenas opciones para principiantes son el panel Git de RStudio y [Github Desktop](https://desktop.github.com/), que mostraremos en este capítulo. Las opciones intermedias (más potentes, pero más complejas) incluyen Source Tree, Gitkracken, Smart Git y otras.

Explicación rápida sobre [los clientes Git](../../../../../C:/Users/Neale/OneDrive%20-%20Neale%20Batra/Documents/Analytic%20Software/R/Projects/R%20handbook/Epi_R_handbook/-%09https:/happygitwithr.com/git-client.html" \l "git-client).

Nota: dado que todas las interfaces utilizan Git internamente, puedes probar varias de ellas, cambiar de una a otra en un proyecto determinado, utilizar la consola puntualmente para una acción que tu interfaz no soporta, o incluso realizar una serie de acciones online en Github.

Como se indica más adelante, es posible que ocasionalmente tengas que escribir comandos Git en un terminal como en la pestaña “terminal” de RStudio (una pestaña adyacente a la consola de R) o la terminal Git Bash.

### Cuenta de Github

Regístrate para obtener una cuenta gratuita en [github.com](../../../../../C:/Users/Neale/OneDrive%20-%20Neale%20Batra/Documents/Analytic%20Software/R/Projects/R%20handbook/Epi_R_handbook/github.com).

Es posible que se te ofrezca configurar la autenticación de dos pasos con una aplicación en tu teléfono. Lee más en [los documentos de ayuda](https://docs.github.com/en/github/authenticating-to-github/securing-your-account-with-two-factor-authentication-2fa) de Github.

Si usas Github Desktop, puedes introducir tus credenciales de Github después de la instalación siguiendo estos [pasos](https://docs.github.com/en/desktop/installing-and-configuring-github-desktop/authenticating-to-github). Si no lo haces, las credenciales se te pedirán más tarde cuando intentes clonar un proyecto desde Github.

## Vocabulario, conceptos y funciones básicas

Al igual que cuando se aprende R, hay que recordar un poco de vocabulario para entender Git. Aquí están los [conceptos básicos para empezar](https://www.freecodecamp.org/news/an-introduction-to-git-for-absolute-beginners-86fa1d32ff71/) / [tutorial interactivo](../../../../../C:/Users/Neale/OneDrive%20-%20Neale%20Batra/Documents/Analytic%20Software/R/Projects/R%20handbook/Epi_R_handbook/learngitbranching.js.org). En las próximas secciones, mostraremos cómo usar las interfaces, pero es bueno tener el vocabulario y los conceptos en mente, para construir tu modelo mental, ya que lo necesitarás cuando más tarde, aunque uses las interfaces de los programas.

### Repositorio

Un repositorio Git ("repo") es una carpeta que contiene todas las subcarpetas y archivos de tu proyecto (datos, código, imágenes, etc.) y sus historiales de revisión. Cuando empieces a seguir los cambios en el repositorio con él, Git creará una carpeta oculta que contiene toda la información de seguimiento. Un repositorio típico de Git es la carpeta de tu proyecto R (ver la página del manual sobre [proyectos R](#r-projects)).

Mostraremos cómo crear (inicializar) un repositorio Git desde Github, Github Desktop o Rstudio en las siguientes secciones.

### Commits (Consolidaciones)

Cuando realices un cambio en el proyecto, hay que realizar un nuevo commit para consolidar estos cambios (el delta) realizados en tus archivos. Por ejemplo, quizás hayas editado algunas líneas de código y actualizado unos datos relacionados. Una vez guardados los cambios, puedes agrupar estos cambios en un solo "commit".

Cada Consolidación (commit) tiene un ID único (un hash). Para el control de versiones, puedes revertir tu proyecto hacia atrás en base a estas consolidaciones, así que es mejor mantenerlas relativamente pequeñas y coherentes. También realizarás una breve descripción de los cambios llamada "mensaje de consolidación". En cierto modo, cada commit es una **instantánea** del proyecto en un momento dado.

¿Cambios por etapas? Poner etapas en los cambios es añadirlos a la zona de preparación para la siguiente consolidación. La idea es que puedas decidir con precisión qué cambios incluir en un determinado commit. Por ejemplo, si trabajas en la especificación del modelo en un script, y más tarde en una figura en otro script, tendría sentido tener dos commits diferentes (sería más fácil en caso de que quisieras revertir los cambios en la figura pero no en el modelo).

### Ramas

Una rama representa una línea independiente de cambios en su repo, una versión paralela y alternativa de los archivos del proyecto.

Las ramas son útiles para probar los cambios antes de incorporarlos a la rama principal, que suele ser la versión primaria/final/"viva" de tu proyecto. Cuando termines de experimentar en una rama, puedes incorporar los cambios a tu rama principal, fusionándola, o eliminarla, si los cambios no fueron tan exitosos.

Nota: no es necesario colaborar con otras personas para utilizar las ramas, ni es necesario tener un repositorio remoto en línea.

### Repositorios locales y remotos

Clonar es crear una copia de un repositorio Git en otro lugar.

Por ejemplo, puedes clonar un repositorio online de Github localmente en tu ordenador, o empezar con un repositorio local y clonarlo online en Github.

Cuando se ha clonado un repositorio, los archivos del proyecto existen en dos lugares:

* el repositorio LOCAL en el ordenador físico. Aquí es donde se hacen los cambios reales a los archivos/código.
* el repositorio REMOTO, en línea: las versiones de los archivos del proyecto en el repositorio Github (o en cualquier otro alojamiento web).

Para sincronizar estos repositorios, utilizaremos más funciones. En efecto, a diferencia de Sharepoint, Dropbox u otro software de sincronización, Git no actualiza automáticamente el repositorio local en base a lo que está en línea, o viceversa. Tú eliges cuándo y cómo sincronizar.

* git fetch descarga los nuevos cambios del repositorio remoto pero no cambia el repositorio local. Piensa en ello para una comprobación del estado del repositorio remoto.
* git pull descarga los nuevos cambios de los repositorios remotos y actualiza el repositorio local.
* Cuando hayas hecho uno o varios commits localmente, puedes hacer git push de los commits al repositorio remoto. Esto envía tus cambios a Github para que otras personas puedan verlos y extraerlos si lo desean.

## Empezar: crear un nuevo repositorio

Hay muchas formas de crear nuevos repositorios. Puedes hacerlo desde la consola, desde Github, desde una interfaz.

Hay dos enfoques generales para la puesta en marcha:

* Crear un nuevo proyecto R a partir de un repositorio de Github existente o nuevo (preferible para los principiantes), o
* Crear un repositorio Github para un proyecto R existente

### Archivos de inicio

Cuando se crea un nuevo repositorio, se pueden crear todos los archivos siguientes, o pueden añadirse a su repositorio en una etapa posterior. Normalmente estarán en la carpeta "raíz" del repositorio.

* Un archivo README es un archivo que alguien puede leer para entender por qué existe tu proyecto y qué más deben saber para usarlo. Al principio estará vacío, pero deberías completarlo más adelante.
* Un archivo .gitignore es un archivo de texto donde cada línea contendría carpetas o archivos que Git debería ignorar (no rastrear los cambios). Lee más sobre esto y mira ejemplos [aquí](https://www.freecodecamp.org/news/gitignore-what-is-it-and-how-to-add-to-repo/).
* Puedes elegir un tipo de licencia para el trabajo, de modo que otras personas sepan en qué condiciones pueden utilizar o reproducir tu obra. Para más información, consulta las [licencias Creative Commons](https://creativecommons.org/licenses/).

### Crear un nuevo repositorio en Github

Para crear un nuevo repositorio, entra en Github y busca el botón verde para crear un nuevo repositorio. Este repositorio, ahora vacío, puede ser clonado localmente en tu ordenador (ver la siguiente sección).

Debes elegir si quieres que tu repositorio sea **público** (visible para todo el mundo en Internet) o **privado** (sólo visible para aquellos con permiso). Esto tiene importantes implicaciones si tus datos son sensibles. Si tu repositorio es privado te encontrarás con algunas cuotas en circunstancias especiales avanzadas, como por ejemplo si estás usando acciones de Github para ejecutar automáticamente tu código en la nube.

### Clonar desde un repositorio de Github

Puedes clonar un repositorio de Github existente para crear un nuevo proyecto R local en tu ordenador.

El repositorio de Github puede ser uno que ya existe y tiene contenido, o puede ser un repositorio vacío que acabas de crear. En este último caso, básicamente estás creando el repositorio de Github y el proyecto local de R al mismo tiempo (ver las instrucciones anteriores).

Nota: si no tienes derechos de contribución en un repositorio de Github, es posible primero bifurcar (fork) el repositorio a tu perfil, y luego proceder con las otras acciones. La bifurcación se explica al final de este capítulo, pero recomendamos que lea primero las otras secciones.

Paso 1: Navega en Github hasta el repositorio, clica en el botón verde "**Code**" y copia la **URL del clon HTTPS** (ver imagen inferior)

El siguiente paso se puede realizar en cualquier interfaz. Lo ilustraremos con Rstudio y Github desktop.

#### En Rstudio

En RStudio, inicia un nuevo proyecto R clicando en File>New project > Version control > Git) (Archivo > Nuevo proyecto > Control de versiones > Git)

* Cuando se le pida la "URL del repositorio", pegue la URL HTTPS de Github
* Asigne al proyecto R un nombre corto e informativo
* Elija dónde se guardará el nuevo proyecto R localmente
* Marque "Abrir en una nueva sesión" y clica en "Crear proyecto".

Ahora estás en un nuevo proyecto local de RStudio que es un clon del repositorio de Github. Este proyecto local y el repositorio de Github están ahora vinculados.

#### En Github Desktop

* Clica en File>Clone repository (Archivo > Clonar un repositorio)
* Selecciona la pestaña URL
* Pegua la URL HTTPS de Github en la primera casilla
* Selecciona la carpeta en la que deseas tener tu repositorio local
* Clica en "CLONE"

### Nuevo repositorio de Github a partir de un proyecto R existente

Un escenario alternativo de configuración es que ya tengas un proyecto R con contenido, y quieras crear un repositorio Github para él.

1. Crear un nuevo repositorio de Github vacío para el proyecto (ver instrucciones anteriores)
2. Clona este repositorio localmente (ver las instrucciones de HTTPS más arriba)
3. Copia todo el contenido de tu proyecto R preexistente (códigos, datos, etc.) en este nuevo repositorio local vacío (por ejemplo, utiliza copiar y pegar).
4. Abre tu nuevo proyecto en RStudio, y ve al panel Git. Los nuevos archivos deberían registrarse como cambios de archivo, ahora rastreados por Git. Por lo tanto, puedes agrupar estos cambios como un commit y empujarlos a Github. Una vez empujado, el repositorio en Github reflejará todos los archivos.

Consulta la sección de flujo de trabajo de Github para obtener más detalles sobre este proceso.

### ¿Qué aspecto tiene ahora?

#### En RStudio

Una vez que hayas clonado un repositorio de Github a un nuevo proyecto R, ahora verás en RStudio una pestaña "Git". Esta pestaña aparece en el mismo panel de RStudio que tu Entorno R:

Ten en cuenta los botones marcados con un círculo en la imagen anterior, ya que se hará referencia a ellos más adelante (de izquierda a derecha):

* Botón para confirmar los cambios del archivo guardado en la rama local (se abrirá una nueva ventana)
* Flecha azul para tirar (actualiza tu versión local de la rama con los cambios realizados en la versión remota/Github de esa rama)
* Flecha verde para empujar (enviar cualquier commits/cambio de tu versión local de la rama a la versión remota/Github de esa rama)
* La pestaña Git en RStudio
* Botón para crear una NUEVA rama usando la rama local que se muestra a la derecha como base. Casi siempre querrá bifurcarse desde la rama principal (después de haber tirado primero para actualizar la rama principal)
* La Rama en la que trabaja actualmente
* Los cambios que haya realizado en el código o en otros archivos aparecerán a continuación

#### En Github Desktop

Github Desktop es una aplicación independiente que te permite gestionar todos tus repositorios. Cuando la abres, la interfaz te permite elegir el repositorio en el que quieres trabajar, y luego realizar acciones básicas de Git desde allí.

## Flujo de trabajo Git + Github

### Resumen del proceso

Una vez que hayas completado la configuración (descrita anteriormente), tendrás un repo de Github que está conectado (clonado) a un proyecto local de R. La rama principal (creada por defecto) es la llamada versión "viva" de todos los archivos. Cuando quieras hacer modificaciones, es una buena práctica crear una nueva rama a partir de la rama principal (como "Make a Copy"). Este es un flujo de trabajo típico en Git porque crear una rama es fácil y rápido.

Un flujo de trabajo típico es el siguiente:

1. Asegúrate de que tu repositorio local está actualizado, actualízalo si no es así
2. Ve a la rama en la que estabas trabajando anteriormente, o crea una nueva rama para probar algunas cosas
3. Trabaja en los archivos localmente en tu ordenador, haga uno o varios commits en esta rama
4. Actualiza la versión remota de la rama con tus cambios (push)
5. Cuando esté satisfecho con su rama, puedes fusionar la versión en línea de la rama de trabajo con la rama "principal" en línea para transferir los cambios

Otros miembros del equipo pueden estar haciendo lo mismo con sus propias ramas, o quizás contribuyendo con commits en su rama de trabajo también.

A continuación, repasamos el proceso anterior paso a paso con más detalle. Es un esquema que hemos desarrollado - está en el formato de una tabla de dos x dos, por lo que debería ayudarnos a entenderlo.

Aquí hay [otro diagrama](https://build5nines.com/introduction-to-git-version-control-workflow/).

Nota: hasta hace poco, se utilizaba el término rama "master" (maestra), pero ahora se denomina rama "main" (principal).

[Fuente](https://build5nines.com/introduction-to-git-version-control-workflow/) de la imagen

## Crear una nueva rama

Cuando seleccionas una rama para trabajar, **Git restablece tu directorio de trabajo tal y como estaba la última vez que estuviste en esta rama**.

### En el panel Git de Rstudio

Asegúrate que te encuentras en la rama “main” (principal) y, a continuación, clica en el icono morado para crear una nueva rama (véase la imagen anterior).

* Pedirá un nombre descriptivo para esa rama, de una palabra (se pueden usar barras bajas si es necesario).
* Verás que localmente, sigues en el mismo proyecto R, pero ya no estás trabajando en la rama "principal".
* Una vez creada, la nueva rama también aparecerá en el sitio web de Github como una rama.

Puedes visualizar las ramas en el panel Git de Rstudio tras clicar en "Historial"

### En Github Desktop

El proceso es muy similar, se pide que dé un nombre a su rama. Después, pedirá que "publique su rama en Github" para que la nueva rama aparezca también en el repositorio remoto.

### En la consola

Lo que realmente ocurre entre bastidores es que creas una nueva rama con git branch, y luego vas a la rama con git checkout (es decir, le dices a Git que tus próximos commits se producirán allí). Desde tu repositorio git:

Para más información sobre el uso de la consola, consulta la sección sobre comandos Git al final.

## Consolidar los cambios (Commit)

Ahora puedes editar el código, añadir nuevos archivos, actualizar conjuntos de datos, etc.

Cada uno de tus cambios es rastreado, una vez que el archivo respectivo es guardado. Los archivos modificados aparecerán en la pestaña Git de RStudio, en Github Desktop, o utilizando el comando git status en el terminal (ver más abajo).

Siempre que hagas cambios sustanciales (por ejemplo, añadir o actualizar una sección de código), haz una pausa y consolida esos cambios (Commit). Piensa en una Consolidación como un "lote" de cambios relacionados con un propósito común. Siempre puedes seguir revisando un archivo después de haber confirmado los cambios en él.

Consejo sobre los commits: en general, es mejor hacer consolidaciones pequeñas, que puedan revertirse fácilmente si surge un problema, para Consolidar juntas modificaciones relacionadas con un propósito común. Para lograr esto, verás que debes hacer commits a menudo. Al principio, es probable que te olvides de hacer commits a menudo, pero luego el hábito se impone.

### En Rstudio

El ejemplo siguiente muestra que, desde la última consolidación, el script de R Markdown "collaboration.Rmd" ha cambiado, y se han añadido varias imágenes PNG.

Puedes que te preguntes qué representan los cuadrados amarillo, azul, verde y rojo que aparecen junto a los nombres de los archivos. Aquí hay una captura de la [hoja de trucos de RStudio](https://www.rstudio.com/wp-content/uploads/2016/01/rstudio-IDE-cheatsheet.pdf) que explica su significado. Ten en cuenta que los cambios con el amarillo "?" aún pueden ser puestos en escena, confirmados y enviados.

* Clica el botón "Commit" en la pestaña Git, que abrirá una nueva ventana (mostrada a continuación)
* Clica en un nombre de archivo en el cuadro superior izquierdo
* Revise los cambios que ha realizado en ese archivo (resaltados en verde o rojo)
* "Poner en escena" el archivo, lo que incluirá esos cambios en la consolidación. Para ello, marca la casilla situada junto al nombre del archivo. También puedes marcar varios nombres de archivo y clicar en "Stage".
* Escribe un mensaje de consolidación breve pero descriptivo (obligatorio)
* Clica el botón "Confirmar". Aparecerá un cuadro emergente mostrando el éxito o un mensaje de error.

Ahora puedes hacer más cambios y más commits, tantas veces como quieras

### En Github Desktop

Puedes ver la lista de los archivos que fueron modificados a la izquierda. Si selecciona un archivo de texto, verá un resumen de las modificaciones que se hicieron en el panel derecho (la vista no funcionará en archivos más complejos como .docs o .xlsx).

Para poner en escena los cambios, basta con marcar la pequeña casilla situada junto a los nombres de los archivos. Cuando hayas seleccionado los archivos que quieres añadir a esta consolidación, dale un nombre a la consolidación, opcionalmente una descripción y luego clica en el botón de **commit**.

### En la consola

Las dos funciones que se utilizan entre bastidores son git add para seleccionar/poner en escena los archivos y git commit para hacer realmente el commit.

### Modificar una consolidación anterior

¿Qué sucede si confirmas algunos cambios, sigues trabajando y te das cuenta de que hiciste cambios que deberían "pertenecer" a la consolidación anterior (en tu opinión)? No temas! Puedes añadir estos cambios a tu consolidación anterior.

En Rstudio, debería ser bastante obvio, ya que hay una casilla "Amend previous commit" en la misma línea que el botón COMMIT.

Por alguna razón poco clara, la funcionalidad no se ha implementado como tal en Github Desktop, pero hay una forma (conceptualmente incómoda pero fácil) de hacerlo. Si has confirmado **pero** aún **no has enviado** tus cambios, aparece un botón "UNDO" justo debajo del botón COMMIT. Clica en él y revertirá tu consolidación (pero mantendrá sus archivos en etapa y tu mensaje de consolidación). Guarde sus cambios, añada nuevos archivos a la consolidación si es necesario y vuelva a confirmar.

En la consola:

Nota: piensa antes de modificar los commits que ya son públicos y compartidos con tus colaboradores.

## Sube y empuja los cambios a Github

"Primero tirar, luego empujar"

Es una buena práctica obtener y tirar antes de empezar a trabajar en tu proyecto, para actualizar la versión de la rama en tu equipo local con los cambios que se han hecho en la versión remota/Github.

Tira a menudo. No dudes. Tira siempre antes de empujar.

Cuando los cambios estén hechos y confirmados y estés contento con el estado de tu proyecto, puedes enviar tus confirmaciones a la versión remota/Github de tu rama.

Repite y repite mientras trabajas en el repositorio.

**Nota:** es mucho más fácil revertir los cambios que fueron confirmados pero no empujados (es decir, siguen siendo locales) que revertir los cambios que fueron empujados al repositorio remoto (y tal vez ya sacados por otra persona), por lo que es mejor empujar cuando haya terminado de introducir cambios en la tarea en la que estaba trabajando.

#### En Rstudio

TIRAR - En primer lugar, clica en el icono "Tirar" (flecha hacia abajo) que busca y tira al mismo tiempo.

PUSH – Clicando en el icono verde "Pull" (flecha hacia arriba). Es posible que se le pida que introduzca tu nombre de usuario y contraseña de Github. La primera vez que se le pida, es posible que tenga que introducir dos líneas de comando Git en el Terminal:

* **git config -global user.email "you@example.com"** (tu dirección de correo electrónico de Github), y
* **git config -global user.name "Tu nombre de usuario de Github"**

Para saber más sobre cómo introducir estos comandos, consulta la sección siguiente sobre comandos Git.

**SUGERENCIA:** ¿Te piden la contraseña muy a menudo? Consulta los capítulos 10 y 11 de este tutorial [para](https://happygitwithr.com/credential-caching.html" \l "credential-caching) conectarse a un repositorio usando una clave SSH (más complicado)

#### En Github Desktop

Clica en el botón "Fetch origin" para comprobar si hay nuevos commits en el repositorio remoto.

Si Git encuentra nuevos commits en el repositorio remoto, el botón cambiará a un botón "Pull". Dado que el mismo botón se utiliza para empujar y tirar, no puedes empujar tus cambios si no tiras antes.

Puedes ir a la pestaña "Historial" (cerca de la pestaña "Cambios") para ver todos los commits (los tuyos y los de los demás). Esta es una buena manera de conocer lo que hicieron tus colaboradores. Puedes leer el mensaje de consolidación, la descripción si la hay, y comparar el código de los dos archivos usando el panel de diferencias.

Una vez que se han extraído todos los cambios remotos y se ha consignado al menos un cambio local, se puede empujar clicando en el mismo botón.

#### Consola

Sin sorpresa, las órdenes son buscar, tirar y empujar.

### Quiero tirar pero tengo trabajo local

Esto puede ocurrir a veces: has hecho algunos cambios en tu repositorio local, pero el repositorio remoto tiene confirmaciones que no has sacado.

Git se negará a hacer un pull porque podría sobrescribir tus cambios. Hay varias estrategias para guardar tus cambios, bien descritas en [Happy Git with R](https://happygitwithr.com/pull-tricky.html), entre las cuales las dos principales son - Confirmar tus cambios, obtener los cambios remotos, extraerlos, resolver los conflictos si es necesario (ver la sección más abajo), y empujar todo en línea - almacenar tus cambios, lo que en cierto modo los guarda a un lado, tirar, desalmacenar (restaurar), y luego confirmar, resolver cualquier conflicto, y empujar.

Si los archivos afectados por los cambios remotos y los archivos afectados por tus cambios locales no se solapan, Git puede resolver los conflictos automáticamente.

En Github Desktop, esto se puede hacer con botones. Para almacenar, ve a Rama > Almacenar todos los cambios.

## Combinar la rama con la principal

Si ha terminado de hacer cambios, puede comenzar el proceso de fusión de esos cambios en la rama principal. Dependiendo de su situación, esto puede ser rápido, o puede tener pasos deliberados de revisión y aprobación que involucren a compañeros de equipo.

### Localmente en Github Desktop

Uno puede fusionar ramas localmente usando Github Desktop. Primero, ve a (checkout) la rama que será la destinataria de los commits, es decir, la rama que quieres actualizar. A continuación, ve al menú Rama > Fusionar en la rama actual y haz clic. Un cuadro te permitirá seleccionar la rama desde la que quieres importar.

### En la consola

Primero, vuelve a la rama que será la destinataria de los cambios. Normalmente es la rama maestra, pero puede ser otra rama. Luego fusiona tu rama de trabajo con la maestra.

[Esta página](https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging) muestra un ejemplo más avanzado de bifurcación y explica un poco lo que ocurre entre bastidores.

### En Github: envío de pull requests

Aunque es totalmente posible fusionar dos ramas localmente, o sin informar a nadie, una fusión puede ser discutida o investigada por varias personas antes de ser integrada en la rama maestra. Para ayudar en el proceso, Github ofrece algunas funciones de discusión en torno a la fusión: el **pull request**.

Un pull request (un "PR") es una solicitud para fusionar una rama con otra (en otras palabras, una solicitud para que tu rama de trabajo se incorpore a la rama "principal"). Una solicitud de extracción suele incluir varias confirmaciones. Un pull request suele iniciar un proceso de conversación y revisión antes de que sea aceptado y la rama sea fusionada. Por ejemplo, puedes leer las discusiones sobre pull requests en [el github de dplyr](https://github.com/tidyverse/dplyr/pulls).

Puedes enviar una solicitud de extracción (PR) directamente desde el sitio web (como se ilustra a continuación) o desde Github Desktop.

* Ir al repositorio Github (en línea)
* Ve a la pestaña "Pull Requests" y clica en el botón "New pull request".
* Selecciona en el menú desplegable para fusionar su rama en la principal
* Escribe un comentario detallado sobre la solicitud de extracción y clica en "Crear solicitud de extracción".

En la imagen siguiente, se ha seleccionado la rama "bosques" para fusionarla con la "principal":

Ahora debería poder ver el pull request (imagen de ejemplo abajo):

* Revisa la pestaña "Archivos cambiados" para ver cómo cambiaría la rama "principal" si se fusionara la rama.
* A la derecha, puedes solicitar una revisión a los miembros de tu equipo etiquetando su ID de Github. Si quieres, puedes configurar el repositorio para que se requiera una revisión de aprobación para poder fusionarlo con el principal.
* Una vez aprobada la solicitud de extracción, se activará un botón para "fusionar la solicitud de extracción". Clica en él.
* Una vez completado, elimine su rama como se explica a continuación.

### Resolución de conflictos

Cuando dos personas modifican la(s) misma(s) línea(s) al mismo tiempo, surge un conflicto de fusión. De hecho, Git se niega a tomar una decisión sobre qué versión mantener, pero te ayuda a encontrar dónde está el conflicto. **NO TE ASUSTES**. La mayoría de las veces, es bastante sencillo de resolver.

Por ejemplo, en Github:

Después de que la fusión haya planteado un conflicto, abre el archivo en tu editor favorito. El conflicto se indicará con una serie de caracteres:

El texto entre <<<<<<< HEAD y ======= proviene de tu repositorio local, y el que está entre ======= y >>>>>>> de la otra rama (que puede ser origin, master o cualquier rama de tu elección).

Tienes que decidir qué versión del código prefieres (o incluso escribir una tercera, incluyendo los cambios de ambas partes si es pertinente), borrar el resto y eliminar todas las marcas que Git ha añadido (<<<<<<< HEAD, =======, >>>>>>> origin/master/your\_branch\_name).

A continuación, guarda el archivo, escénalo y haz un commit: este es el commit que hace que la versión fusionada sea "oficial". No te olvides de hacer push después.

Cuanto más a menudo tiren y empujen tú y tus colaboradores, menores serán los conflictos.

Nota: Si te sientes cómodo con la consola, existen [*opciones de fusión*](https://git-scm.com/book/en/v2/Git-Tools-Advanced-Merging) más [*avanzadas*](https://git-scm.com/book/en/v2/Git-Tools-Advanced-Merging) (por ejemplo, ignorar los espacios en blanco, dar prioridad a un colaborador, etc.).

### Borrar una rama

Una vez que una rama se ha fusionado con la maestra y ya no es necesaria, puedes eliminarla.

#### Github + Rstudio

Ve al repositorio en Github y clica en el botón para ver todas las ramas (junto al desplegable para seleccionar ramas). Ahora busca tu rama y clica en el icono de la papelera junto a ella. Lee más detalles sobre cómo eliminar una rama [aquí](https://docs.github.com/en/free-pro-team@latest/github/collaborating-with-issues-and-pull-requests/creating-and-deleting-branches-within-your-repository" \l "deleting-a-branch).

Asegúrate de eliminar también la rama localmente en tu ordenador. Esto no ocurrirá automáticamente.

* Desde RStudio, asegúrese de estar en la rama principal
* Cambie a escribir los comandos Git en la "Terminal" de RStudio (la pestaña adyacente a la consola de R), y escribe: **git branch -d nombre\_de\_rama**, donde "nombre\_de\_rama" es el nombre de la rama a eliminar
* Actualiza tu pestaña Git y la rama debería desaparecer

#### En Github Desktop

Sólo tienes que comprobar la rama que quieres eliminar, e ir al menú Rama > Eliminar.

### Bifurcación

Puedes hacer un fork de un proyecto si quieres contribuir a él pero no tienes los derechos para hacerlo, o si sólo quieres modificarlo para tu uso personal. Puedes encontrar una breve descripción de la bifurcación [aquí](https://guides.github.com/activities/forking/).

En Github, clica en el botón "Fork":

Esto clonará el repositorio original, pero en tu propio perfil. Así que ahora hay dos versiones del repositorio **en Github**: la original, que no puedes modificar, y la versión clonada en tu perfil.

Entonces, puedes clonar tu versión del repositorio en línea localmente en tu ordenador, utilizando cualquiera de los métodos descritos en las secciones anteriores. Luego, puede crear una nueva rama, hacer cambios, confirmarlos y empujarlos a tu repositorio remoto.

Una vez que estés contento con el resultado, puedes crear un Pull Request desde Github o Github Desktop para iniciar la conversación con los propietarios/mantenedores del repositorio original.

**¿Y si necesitas algunos commits más recientes del repositorio oficial?**

Imagina que alguien hace una modificación crítica en el repositorio oficial, que quieres incluir en tu versión clonada. Es posible sincronizar tu fork con el repositorio oficial. Implica usar el terminal, pero no es demasiado complicado. Principalmente necesitas recordar que - upstream = el repositorio oficial, el que no has podido modificar - origin = tu versión del repositorio en tu perfil de Github

Puedes leer [este tutorial](https://docs.github.com/en/github/collaborating-with-issues-and-pull-requests/syncing-a-fork) o seguirlo a continuación:

Primero, escribe en tu terminal Git (dentro de tu repo):

Si aún no ha configurado el repositorio upstream debería ver dos líneas, que comienzan por origen. Muestran el repositorio remoto al que apuntan fetch y push. Recuerda que origen es el apodo convencional para tu propia versión del repositorio en Github. Por ejemplo:

Ahora, añada un nuevo repositorio remoto:

Aquí la dirección es la que genera Github cuando clonas un repositorio (ver sección de clonación). Ahora tendrás cuatro punteros remotos:

Ahora que la configuración está hecha, siempre que quieras obtener los cambios del repositorio original (upstream), sólo tienes que ir (checkout) a la rama que quieres actualizar y teclear:

Si hay conflictos, tendrá que resolverlos, tal y como se explica en la sección Resolución de conflictos.

**Resumen**: forking es clonación, pero en el lado del servidor de Github. El resto de las acciones son las típicas del flujo de trabajo de colaboración (clonar, empujar, tirar, confirmar, fusionar, enviar solicitudes de extracción...).

Nota: aunque la bifurcación es un concepto, no un comando de Git, también existe en otros hosts web, como [*Bitbucket*](https://www.atlassian.com/git/tutorials/comparing-workflows/forking-workflow).

## Lo que hemos aprendido

Has aprendido a hacerlo:

* Configurar Git para rastrear las modificaciones en tus carpetas,
* Conectar tu repositorio local a un repositorio remoto en línea,
* Confirmar los cambios,
* Sincronizar tus repositorios locales y remotos.

Todo esto debería ayudar a ponerte en marcha y ser suficiente para la mayoría de tus necesidades de análisis epidemiológico. Normalmente no tenemos un uso tan avanzado como los desarrolladores.

Sin embargo, debes saber que si quieres (o necesitas) ir más allá, Git ofrece más potencia para simplificar los historiales de commit, revertir uno o varios commits, hacer cherry-pick de commits, etc. Algunas cosas pueden parecer pura magia, pero ahora que tienes los fundamentos, es más fácil construir sobre ellos.

Ten en cuenta que mientras el panel Git en Rstudio y Github Desktop son buenos para los principiantes / uso diario en nuestra línea de trabajo, no ofrecen una interfaz para algunas de las funciones intermedias / avanzadas de Git. Algunas interfaces más completas permiten hacer más cosas con apuntar y clicar (normalmente a costa de un diseño más complejo).

Recuerda que, dado que puedes utilizar cualquier herramienta en cualquier momento para realizar el seguimiento de tu repositorio, puedes instalar muy fácilmente una interfaz para probarla a veces, o para realizar alguna tarea compleja menos común ocasionalmente, mientras prefieres una interfaz simplificada para el resto del tiempo (por ejemplo, utilizando Github Desktop la mayor parte del tiempo, y cambiando a SourceTree o Gitbash para algunas tareas específicas).

## Comandos Git

### Aprendizaje recomendado

Para aprender los comandos de Git en un tutorial interactivo, consulte [este sitio web](https://learngitbranching.js.org/).

### Dónde introducir los comandos

Se introducen comandos en un shell Git.

Opción 1 Puedes abrir una nueva Terminal en RStudio. Esta pestaña está al lado de la Consola R. Si no puede escribir ningún texto en ella, clica en el menú desplegable debajo de "Terminal" y selecciona "Nueva terminal". Escriba los comandos en el espacio parpadeante delante del signo de dólar "$".

Opción 2 También puede abrir un shell (un terminal para introducir comandos) clicando en el icono azul de "engranajes" en la pestaña Git (cerca del entorno de RStudio). Selecciona "Shell" en el menú desplegable. Se abrirá una nueva ventana en la que puedes escribir los comandos después del signo de dólar "$".

Opción 3 Clica con el botón derecho para abrir "Git Bash aquí" que abrirá el mismo tipo de terminal, o abra Git Bash desde tu lista de aplicaciones. [Más información para principiantes sobre Git Bash](https://happygitwithr.com/shell.html), cómo encontrarlo y algunos comandos bash que necesitarás.

### Ejemplos de comandos

A continuación presentamos algunos comandos git comunes. Cuando los uses, ten en cuenta qué rama está activa (check-out), ¡ya que eso cambiará la acción!

En los comandos de abajo, representa un nombre de rama. representa el hash ID de un commit específico. representa un número. No escriba los símbolos < o >.

| **Comando Git** | **Acción** |
| --- | --- |
| git branch <nombre> | Crear una nueva rama con el nombre |
| git checkout <nombre> | Cambiar la rama actual a |
| git checkout -b <nombre> | Acceso directo para crear una nueva rama y cambiar a ella |
| estado de git | Ver los cambios no rastreados |
| git add <fichero> | Poner en escena un archivo |
| git commit -m <mensaje> | Confirmar los cambios actualmente en fase a la rama actual con el mensaje |
| git fetch | Obtener los commits del repositorio remoto |
| git pull | Tirar de los commits del repositorio remoto en la rama actual |
| git push | Empujar los commits locales al directorio remoto |
| interruptor git | Una alternativa a git checkout que se está introduciendo en Git |
| git merge <nombre> | Fusionar rama en la rama actual |
| git rebase <nombre> | Añadir los commits de la rama actual a la rama |

## Recursos

Gran parte de esta página fue informada por [este sitio web "Happy Git with R"](https://happygitwithr.com/) de Jenny Bryan. Hay una sección muy útil de este sitio web que te ayuda a solucionar errores comunes relacionados con Git y R.

La [documentación y la guía de inicio de Github.com](https://docs.github.com/en/github).

La [hoja de trucos de](https://www.rstudio.com/wp-content/uploads/2016/01/rstudio-IDE-cheatsheet.pdf) RStudio ["IDE"](https://www.rstudio.com/wp-content/uploads/2016/01/rstudio-IDE-cheatsheet.pdf) que incluye consejos sobre Git con RStudio.

<https://ohi-science.org/news/github-going-back-in-time>

**Comandos Git para principiantes**

Un [tutorial interactivo](../../../../../C:/Users/Neale/OneDrive%20-%20Neale%20Batra/Documents/Analytic%20Software/R/Projects/R%20handbook/Epi_R_handbook/learngitbranching.js.org) para aprender los comandos de Git.

https://www.freecodecamp.org/news/an-introduction-to-git-for-absolute-beginners-86fa1d32ff71/: bueno para aprender lo más básico para rastrear los cambios en una carpeta en tu propio ordenador.

Buenos esquemas para entender las ramas: <https://speakerdeck.com/alicebartlett/git-for-humans>

**Tutoriales que cubren tanto los temas básicos como los más avanzados**

<https://tutorialzine.com/2016/06/learn-git-in-30-minutes>

[https://dzone.com/articles/git-tutorial-commands-and-operations-in-git](https://rsjakob.gitbooks.io/git/content/chapter1.html) <https://swcarpentry.github.io/git-novice/>(curso corto) <https://rsjakob.gitbooks.io/git/content/chapter1.html>

El [libro Pro Git](https://git-scm.com/book/en/v2) se considera una referencia oficial. Aunque algunos capítulos están bien, suele ser un poco técnico. Probablemente sea un buen recurso una vez que hayas usado un poco Git y quieras aprender un poco más precisamente lo que sucede y cómo ir más allá.

# #Errores comunes

{#common-errors}

Esta página incluye una lista actualizada de los errores más comunes y sugiere soluciones para solucionarlos.

## Interpretación de los mensajes de error

Los mensajes de error de R pueden ser crípticos a veces, así que Google es tu amigo. Busca el mensaje de error con "R" y busca publicaciones recientes en [StackExchange.com](../../../../../C:/Users/Neale/OneDrive%20-%20Neale%20Batra/Documents/Analytic%20Software/R/Projects/R%20handbook/Epi_R_handbook/StackExchange.com), [stackoverflow.com](../../../../../C:/Users/Neale/OneDrive%20-%20Neale%20Batra/Documents/Analytic%20Software/R/Projects/R%20handbook/Epi_R_handbook/stackoverflow.com), [community.rstudio.com](../../../../../C:/Users/Neale/OneDrive%20-%20Neale%20Batra/Documents/Analytic%20Software/R/Projects/R%20handbook/Epi_R_handbook/community.rstudio.com), twitter (#rstats) y otros foros utilizados por los programadores para archivar preguntas y respuestas. Intenta encontrar publicaciones recientes que hayan resuelto problemas similares.

Si después de mucho buscar no encuentras una respuesta a tu problema, considera la posibilidad de crear un ejemplo reproducible ("reprex") y publicar tú mismo la pregunta. Consulta la página sobre [Cómo obtener ayuda](#getting-help) para obtener consejos sobre cómo crear y publicar un ejemplo reproducible en los foros.

## Errores comunes

A continuación, enumeramos algunos errores comunes y posibles explicaciones/soluciones. Algunos de ellos se han tomado prestados de Noam Ross, que analizó los mensajes más comunes del foro en Stack Overflow sobre los mensajes de error de R (véase el análisis [aquí](https://github.com/noamross/zero-dependency-problems/blob/master/misc/stack-overflow-common-r-errors.md))

### Errores tipográficos

Error: símbolo inesperado en:

" geom\_histograma(stat = "identity")+

tidyquant::geom\_ma(n=7, tamaño = 2, color = "rojo" lty"

Si aparece "símbolo inesperado", compruebe si faltan comas

### Errores del paquete

no pudo encontrar la función "x"...

Esto probablemente significa que has escrito mal el nombre de la función, o que has olvidado instalar o cargar un paquete.

Error en select(data, var) : argumento no utilizado (var)

Crees que estás usando dplyr::select() pero la función select() ha sido enmascarada por MASS::select() - especifica dplyr:: o reordena la carga de tu paquete para que dplyr esté después de todos los demás.

Otros errores de enmascaramiento comunes provienen de: plyr::summarise() y stats::filter(). Considere la posibilidad de utilizar el [paquete**en conflicto**](https://www.tidyverse.org/blog/2018/06/conflicted/).

Error en install.packages : ERROR: no se ha podido bloquear el directorio 'C:\Nsers\Name\Documents\R\win-library\4.0' para modificar

Intenta eliminar 'C:\Nsers\Nnombre\Documents\R\win-library\4.0/00LOCK'

Si recibes un error diciendo que necesita eliminar un archivo "00LOCK", ve a tu biblioteca "R" en el directorio de tu ordenador (por ejemplo, R/win-library/) y busca una carpeta llamada "00LOCK". Elimíneaa manualmente e intenta instalar el paquete de nuevo. Es probable que un proceso de instalación anterior se haya interrumpido, provocando este error.

### Errores en los objetos

No hay tal archivo o directorio:

Si ves un error como este cuando intentas exportar o importar: Compruebe la ortografía del archivo y de la ruta de acceso, y si la ruta contiene barras inclinadas, asegúrese de que son hacia delante / y no hacia atrás. Asegúrese también de que ha utilizado la extensión de archivo correcta (por ejemplo, .csv, .xlsx).

objeto 'x' no encontrado

Esto significa que el objeto al que se hace referencia no existe. ¿Quizá el código anterior no se ha ejecutado correctamente?

Error en 'x': subíndice fuera de los límites

Esto significa que has intentado acceder a algo (un elemento de un vector o una lista) que no estaba allí.

### Errores de sintaxis de las funciones

# ejecuta el recodificado sin reponer la variable x en mutate(x = recodificado(x, OLD = NEW)

Error: Problema con la entrada `mutate()` de `hospital`.

Falta el argumento ".x", sin que haya un valor por defecto

i La entrada `hospital` es `recode(...)`.

Este error de arriba (falta el argumento .x, sin valor por defecto) es común en mutate() si está suministrando una función como recode() o replace\_na() donde se espera que proporcione el nombre de la columna como primer argumento. Esto es fácil de olvidar.

### Errores lógicos

Error en si

Esto probablemente significa que se aplicó una sentencia if a algo que no era TRUE o FALSE.

### Errores de los factores

#Trató de añadir un valor ("Falta") a un factor (con replace\_na operando en un factor)

Problema con la entrada `mutate()` de `age\_cat`.

i nivel de factor no válido, generado por NA

i La entrada `age\_cat` es `replace\_na(age\_cat, "Missing")`.nivel de factor no válido, NA generado

Si ve este error sobre niveles de factor no válidos, es probable que tenga una columna de tipo Factor (que contiene niveles predefinidos) y haya intentado añadirle un nuevo valor. Conviértala al tipo Carácter antes de añadir un nuevo valor.

### Errores de trazado

Error: Valores insuficientes en la escala manual. Se necesitan 3 pero sólo se han proporcionado 2. ggplot() scale\_fill\_manual() values = c("orange", "purple") ... insuficiente para el número de niveles del factor ... considerar si NA es ahora un nivel del factor...

No se puede añadir un objeto x

Probablemente tienes un + extra al final de un comando ggplot que necesitas eliminar.

### Errores de R Markdown

Si el mensaje de error contiene algo como Error en options[[sprintf("fig.%s", i)]], comprueba que tus opciones knitr en la parte superior de cada chunk utilizan correctamente el out.width = o out.height = y no fig.width= y fig.height=.

### Varios

Considera si has reordenado los verbos **dplyr** y no has reemplazado una tubería en el medio, o no has eliminado una tubería del final después de reordenar.

## Recursos

Esta es otra entrada del blog que enumera los [errores](https://www.r-bloggers.com/2016/06/common-r-programming-errors-faced-by-beginners/) comunes [de programación en R a los que se enfrentan los principiantes](https://www.r-bloggers.com/2016/06/common-r-programming-errors-faced-by-beginners/)

# #Cómo obtener ayuda

{#getting-help}

Esta página explica cómo obtener ayuda publicando una incidencia en Github o publicando un ejemplo reproducible ("reprex") en un foro en línea.

## Problemas en Github

Muchos paquetes y proyectos de R tienen su código alojado en el sitio web Github.com. Puedes comunicarte directamente con los autores a través de este sitio web publicando un "Issue".

Lee más sobre cómo almacenar tu trabajo en Github en la página [Colaboración y Github].

En Github, cada proyecto está contenido en un repositorio. Cada repositorio contiene código, datos, resultados, documentación de ayuda, etc. También hay un vehículo para comunicarse con los autores llamado "Issues".

Vea a continuación la página de Github para el paquete **incidence2** (utilizado para hacer curvas epidémicas). Puedes ver la pestaña "Issues" resaltada en amarillo. Puedes ver que hay 5 temas abiertos.

Una vez en la pestaña de problemas, podrá ver los problemas abiertos. Revíselas para asegurarse de que tu problema no ha sido ya tratado. Puedes abrir una nueva incidencia clicando en el botón verde de la derecha. Necesitarás una cuenta de Github para hacerlo.

En tu edición, siga las instrucciones que aparecen a continuación para proporcionar un ejemplo mínimo y reproducible. Y, por favor, ¡sea cortés! La mayoría de las personas que desarrollan paquetes y proyectos de R lo hacen en su tiempo libre (¡como este manual!).

Para leer más materiales avanzados sobre el manejo de problemas en tu propio repositorio de Github, consulta la [documentación de](https://guides.github.com/features/issues/) Github [sobre Problemas](https://guides.github.com/features/issues/).

## Ejemplo reproducible

Proporcionar un ejemplo reproducible ("reprex") es la clave para obtener ayuda cuando se publica en un foro o en una cuestión de Github. La gente quiere ayudarte, pero tienes que darles un ejemplo con el que puedan trabajar en su propio ordenador. El ejemplo debe:

* Demuestre el problema que ha encontrado
* Ser mínimo, en el sentido de que incluya sólo los datos y el código necesarios para reproducir el problema
* Ser reproducible, de manera que se incluyan todos los objetos (por ejemplo, los datos), las llamadas al paquete (por ejemplo, library() o p\_load())

Además, ¡asegúrese de no publicar ningún dato sensible con el reprex! Puedes crear dataframes de ejemplo, o utilizar uno de los dataframes incorporados en R (introduzca data() para abrir una lista de estos conjuntos de datos).

### El paquete ****reprex****

El paquete **reprex** puede ayudarte a crear un ejemplo reproducible:

1. **reprex** se instala con **tidyverse**, así que carga cualquiera de los dos paquetes
2. Inicia un script de R que cree el problema, paso a paso, empezando por la carga de paquetes y datos.

Copia todo el código en tu portapapeles y ejecuta el siguiente comando:

Verás que aparece una salida HTML en el panel del visor de RStudio. Contendrá todo tu código y cualquier advertencia, error o salida de gráficos. Esta salida también se copia en el portapapeles, por lo que puede publicarla directamente en una cuestión de Github o en un mensaje del foro.

* Si establece session\_info = TRUE se incluirá la salida de sessioninfo::session\_info() con sus versiones de R y del paquete R
* Puedes proporcionar un directorio de trabajo a wd =
* Puedes leer más sobre los argumentos y las posibles variaciones en la documentación o introduciendo ?reprex

En el ejemplo anterior, el comando ggplot() no se ejecutó porque el argumento date\_format = no es correcto - debería ser date\_labels =.

### Datos mínimos

Los ayudantes tienen que ser capaces de utilizar sus datos - idealmente tienen que ser capaces de crearlos con código.

Para crear unos datos mínimo, considere la posibilidad de anonimizar y utilizar sólo un subconjunto de las observaciones.

EN CONSTRUCCIÓN - también puede utilizar la función dput() para crear unos datos mínimo.

## Publicar en un foro

Lee muchos mensajes del foro. Comprende qué mensajes están bien escritos y cuáles no.

1. En primer lugar, decida si va a formular la pregunta. Has revisado a fondo el sitio web del foro, probando con varios términos de búsqueda, para ver si tu pregunta ya ha sido formulada?
2. Dale a tu pregunta un título informativo (no "¡Ayuda! esto no funciona").
3. Escribe tu pregunta:

* Presenta la situación y tu problema
* Enlaza con posts de temas similares y explica cómo no responden a tu pregunta
* Incluya cualquier información relevante para ayudar a alguien que no conozca el contexto de tu trabajo
* Dé un ejemplo mínimo reproducible con la información de tu sesión de R
* Utiliza la ortografía, la gramática y la puntuación adecuadas, y divide tu pregunta en párrafos para que sea más fácil de leer

1. Supervisa tu pregunta una vez publicada para responder a cualquier solicitud de aclaración. Sea cortés y amable: a menudo las personas que responden están ofreciendo tu tiempo para ayudarle. Si tiene una pregunta de seguimiento, considere si debe ser una pregunta publicada por separado.
2. Marca la pregunta como respondida, si obtienes una respuesta que satisfaga la petición original. Esto ayuda a que otros reconozcan más tarde rápidamente la solución.

Lee estos posts sobre [cómo hacer una buena pregunta](https://stackoverflow.com/help/how-to-ask) el [código de conducta de Stack overflow](https://stackoverflow.com/conduct).

## Recursos

Página de Tidyverse sobre cómo [obtener ayuda](https://www.tidyverse.org/help/" \l ":~:text=When you want to make,to load the reprex package.&text=Enter reprex() in the,preview of your rendered reprex.)

Consejos para [elaborar unos datos mínimo](https://xiangxing98.github.io/R_Learning/R_Reproducible.nb.html" \l "producing-a-minimal-dataset)

Documentación de la [función dput](https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/dput)

# #R en redes locales

{#r-on-network-drives}

## Resumen

El uso de R en unidades compartidas de la red o de la "empresa" puede presentar desafíos adicionales. Esta página contiene enfoques, errores comunes y sugerencias sobre la solución de problemas obtenidas a partir de nuestra experiencia trabajando con estos problemas. Se incluyen consejos para las situaciones especialmente delicadas relacionadas con R Markdown.

**Uso de R en unidades de red: Principios generales**

1. Debe obtener acceso de administrador para tu ordenador. Configure RStudio específicamente para que se ejecute como administrador.
2. Guarde los paquetes en una biblioteca de una unidad con letras (por ejemplo, "C:") cuando sea posible. Utiliza lo menos posible una biblioteca de paquetes cuya ruta empiece por "\N".
3. el paquete **rmarkdown no** debe estar en una librería de paquetes "\N", ya que entonces no puede conectarse a TinyTex o Pandoc.

## RStudio como administrador

Cuando clica en el icono de RStudio para abrirlo, hazlo clicando con el botón derecho. Dependiendo de tu máquina, puede ver una opción para "Ejecutar como administrador". De lo contrario, puede ver una opción para seleccionar Propiedades (entonces debería aparecer una ventana con la opción "Compatibilidad", y puede seleccionar una casilla de verificación "Ejecutar como administrador").

## Comandos útiles

A continuación se presentan algunos comandos útiles cuando se trata de solucionar problemas utilizando R en unidades de red.

Puedes devolver la(s) ruta(s) a las bibliotecas de paquetes que R está utilizando. Serán listadas en el orden que R está usando para instalar/cargar/buscar paquetes. Por lo tanto, si quiere que R utiliza una biblioteca diferente por defecto, puede cambiar el orden de estas rutas (ver más abajo).

Es posible que desee cambiar el orden de las bibliotecas de paquetes utilizados por R. Por ejemplo, si R está recogiendo una ubicación de la biblioteca que comienza con "\" y uno que comienza con una letra, por ejemplo, "D:". Puedes ajustar el orden de .libPaths() con el siguiente código.

Si tiene dificultades para que R Markdown se conecte a Pandoc, comience con este código para averiguar dónde cree RStudio que está tu instalación de Pandoc.

Si quieres ver de qué biblioteca se está cargando un paquete, prueba con el siguiente código:

## Solución de errores comunes

**"Fallo al compilar...tex en rmarkdown"**

* Compruebe la instalación de TinyTex, o instale TinyTex en la ubicación C:. Consulte la página de [fundamentos de R](#r-basics) sobre cómo instalar TinyTex.

**No se pueden cargar las rutinas de Internet**

Por ejemplo, Error en tools::startDynamicHelp() : no se pueden cargar las rutinas de Internet

* Intente seleccionar la versión de 32 bits de RStudio a través de Herramientas/Opciones Globales.
  + nota: si la versión de 32 bits no aparece en el menú, asegúrese de que no está utilizando RStudio v1.2.
* Alternativamente, intente desinstalar R y volver a instalarlo con una versión de bits diferente (32 en lugar de 64)

**La biblioteca C: no aparece como opción cuando intento instalar los paquetes manualmente**

* Ejecuta RStudio como administrador, entonces aparecerá esta opción.
* Para configurar RStudio para que se ejecute siempre como administrador (lo que resulta ventajoso cuando se utiliza un proyecto R en el que no se clica en el icono de RStudio para abrirlo)... clica con el botón derecho en el icono de Rstudio

La imagen siguiente muestra cómo puede seleccionar manualmente la biblioteca en la que instalar un paquete. Esta ventana aparece cuando se abre el panel de paquetes de RStudio y se clica en "Instalar".

**Error de Pandoc 1**

Si aparece el error "pandoc error 1" al tejer R Markdowns scripts en unidades de red:

* De las múltiples ubicaciones de las bibliotecas, que aparezca en primer lugar la que tenga una unidad de disco con letras (véanse los códigos anteriores)
* La solución anterior funcionó cuando se tejió en la unidad local, pero mientras que en una conexión a Internet en red
* Vea más consejos aquí: <https://ciser.cornell.edu/rmarkdown-knit-to-html-word-pdf/>

**Error Pandoc 83**

El error será algo así: no se puede encontrar el archivo...rmarkdown...lua.... Esto significa que no se ha podido encontrar este archivo.

[Ver https://stackoverflow.com/questions/58830927/rmarkdown-unable-to-locate-lua-filter-when-knitting-to-word](https://stackoverflow.com/questions/58830927/rmarkdown-unable-to-locate-lua-filter-when-knitting-to-word)

Posibilidades:

1. El paquete Rmarkdown no está instalado
2. El paquete Rmarkdown no se encuentra
3. Un problema de derechos de administración.

Es posible que R no sea capaz de encontrar el archivo del paquete rmarkdown, así que compruebe en qué biblioteca vive el paquete **rmarkdown** (vea el código anterior). Si el paquete está instalado en una biblioteca inaccesible (por ejemplo, comienza con "\") considere moverlo manualmente a C: o a otra biblioteca con nombre. Ten en cuenta que el paquete **rmarkdown** tiene que ser capaz de conectarse a la instalación de TinyTex, por lo que no puede vivir en una biblioteca en una unidad de red.

**Error Pandoc 61**

Por ejemplo: Error: la conversión del documento pandoc ha fallado con el error 61 o Could not fetch...

* Prueba a ejecutar RStudio como administrador (clica con el botón derecho en el icono, selecciona ejecutar como administrador, vea las instrucciones anteriores)
* Vea también si el paquete específico que no pudo ser alcanzado puede ser movido a la biblioteca C:.

**Error de LaTex (ver más abajo)**

Un error como: ! Paquete pdftex.def Error: Archivo 'cict\_qm2\_2020-06-29\_files/figure-latex/unnamed-chunk-5-1.png' no encontrado: usando la configuración de borrador. o Error: LaTeX falló al compilar nombre\_de\_archivo.tex.

* [Consulte https://yihui.org/tinytex/r/#debugging](https://yihui.org/tinytex/r/" \l "debugging) para obtener consejos de depuración.
* Ver nombre\_de\_archivo.log para más información.

**Error de Pandoc 127**

Podría tratarse de un problema de RAM (espacio). Reinicia tu sesión de R e inténtelo de nuevo.

**Asignación de unidades de red**

Mapear una unidad de red puede ser arriesgado. Consulte con tu departamento de TI antes de intentarlo.

Un consejo tomado de esta [discusión del foro](https://stackoverflow.com/questions/48161177/r-markdown-openbinaryfile-does-not-exist-no-such-file-or-directory/55616529?noredirect=1" \l "comment97966859_55616529):

¿Cómo se abre un archivo "a través de una unidad de red asignada"?

* En primer lugar, tendrás que conocer la ubicación de la red a la que intentas acceder.
* A continuación, en el administrador de archivos de Windows, deberá clicar con el botón derecho en "Este PC" en el panel de la derecha, y seleccionar "Asignar una unidad de red".
* Vaya a través del diálogo para definir la ubicación de red de antes como una unidad de letra.
* Ahora tienes dos maneras de llegar al archivo que estás abriendo. Usar la ruta de la letra de la unidad debería funcionar.

**Error en install.packages()**

Si obtiene un error que incluya la mención de un directorio de "bloqueo", por ejemplo Error en install.packages : ERROR: failed to lock directory...

Busca en tu biblioteca de paquetes y verá una carpeta cuyo nombre empieza por "00LOCK". Pruebe los siguientes consejos:

* Elimine manualmente el directorio de la carpeta "00LOCK" de tu biblioteca de paquetes. Intente instalar el paquete de nuevo.
* También puede probar el comando pacman::p\_unlock() (también puede poner este comando en el Rprofile para que se ejecute cada vez que se abra el proyecto). Luego intente instalar el paquete de nuevo. Puedes tomar varios intentos.
* Pruebe a ejecutar RStudio en modo de administrador e intente instalar los paquetes uno por uno.
* Si todo lo demás falla, instale el paquete en otra biblioteca o carpeta (por ejemplo, Temp) y luego copie manualmente la carpeta del paquete en la biblioteca deseada.

# #data.table

{#data-table}

El manual se centra en las funciones "verbales" **de dplyr** y en el operador de tubería %>% **de magrittr como** método para limpiar y agrupar datos, pero el paquete **data.table** ofrece un método alternativo que puedes encontrar en tu carrera en R.

## Introducción a data.table

Una tabla de datos es una estructura de datos bidimensional como un dataframe que permite realizar operaciones de agrupación complejas. La sintaxis de data.table está estructurada de forma que se puedan realizar operaciones sobre filas, columnas y grupos.

La estructura es **DT[i, j, by]**, separada por 3 partes; los argumentos i**, j** y **by**. El argumento **i** permite subconjuntar las filas necesarias, el argumento **j permite** operar sobre las columnas y el argumento **by** permite operar sobre las columnas por grupos.

En esta página se tratarán los siguientes temas:

* Importación de datos y uso de fread() y fwrite()
* Selección y filtrado de filas mediante el argumento **i**
* Uso de las funciones de ayuda %like%, %chin%, %between%
* Selección y cálculo de columnas con el argumento **j**
* Cálculo por grupos utilizando el argumento **by**
* Añadir y actualizar datos a las tablas de datos utilizando :=

## Cargar paquetes e importar datos

### Cargar paquetes

Utilizando la función p\_load() de **pacman**, cargamos (e instalamos si es necesario) los paquetes necesarios para este análisis.

### Importar datos

Esta página explorará algunas de las funciones principales de **data.table** utilizando la lista de casos referenciados a lo largo del manual.

Importamos los datos de casos de una epidemia de ébola simulada. Si desea descargar los datos para seguirlos paso a paso, consulte las instrucciones en la página [Descargar libro y datos]. Los datos se importa mediante la función import() del paquete **rio**. Consulte la página sobre [importación y exportación](#import-and-export) para conocer las distintas formas de importar datos. A partir de aquí utilizamos data.table() para convertir el dataframe en una tabla de datos.

La función fread() se utiliza para importar directamente archivos delimitados regulares, como los archivos .csv, directamente a un formato de tabla de datos. Esta función, y su homóloga, fwrite(), utilizada para escribir tablas de datos como archivos delimitados regulares, son opciones muy rápidas y eficientes desde el punto de vista computacional para bases de datos de gran tamaño.

Las primeras 20 filas del listado:

Los comandos básicos de R, como dim(), que se utilizan para los dataframes, también pueden utilizarse para las tablas de datos

## El argumento i: seleccionar y filtrar filas

Recordando la estructura DT**[i, j, by]**, podemos filtrar filas utilizando números de fila o expresiones lógicas. El argumento i es el primero; por tanto, se puede utilizar la sintaxis **DT[i]** o **DT[i,]**.

El primer ejemplo recupera las 5 primeras filas de la tabla de datos, el segundo ejemplo subsume los casos de 18 años o más, y el tercer ejemplo subsume los casos de 18 años o más pero no diagnosticados en el Hospital Central:

El uso de .N en el argumento i representa el número total de filas en la tabla de datos. Esto se puede utilizar para subconjuntar los números de las filas:

### Uso de funciones de ayuda para el filtrado

La tabla de datos utiliza funciones de ayuda que facilitan el subconjunto de filas. La función %like% se utiliza para coincidir con un patrón en una columna, %chin% se utiliza para coincidir con un carácter específico, y la función de ayuda %between% se utiliza para coincidir con columnas numéricas dentro de un rango preestablecido.

En los siguientes ejemplos: \* filtramos las filas en las que la variable hospital contiene "Hospital" \* filtramos las filas en las que el resultado es "Recuperación" o "Muerte" \* filtramos las filas en el rango de edad 40-60

## El argumento j: seleccionar y calcular en columnas

Utilizando la estructura DT**[i, j, by]**, podemos seleccionar columnas utilizando números o nombres. El argumento **j** es el segundo; por lo tanto, se utiliza la sintaxis **DT[, j]**. Para facilitar los cálculos sobre el argumento **j**, la columna se envuelve utilizando list() o .().

### Selección de columnas

El primer ejemplo recupera la primera, tercera y quinta columnas de la tabla de datos, el segundo ejemplo selecciona todas las columnas excepto las de altura, peso y sexo. El tercer ejemplo utiliza la envoltura .() para seleccionar las columnas **case\_id** y **outcome**.

### Cálculo en columnas

Combinando los argumentos **i** y **j es posible filtrar filas** y calcular en las columnas. El uso de **.N** en el argumento **j** también representa el número total de filas en la tabla de datos y puede ser útil para devolver el número de filas después del filtrado de filas.

En los siguientes ejemplos: \* Contar el número de casos que permanecieron más de 7 días en el hospital \* Calcular la edad media de los casos que murieron en el hospital militar \* Calcular la desviación estándar, la mediana, la edad media de los casos que se recuperaron en el hospital central

Recuerde que el uso de la envoltura .() en el argumento j facilita el cálculo, devuelve una tabla de datos y permite nombrar las columnas.

## El argumento by: el cálculo por grupos

El argumento **by** es el tercer argumento de la estructura **DT[i, j, by]**. El argumento **by** acepta tanto un vector de caracteres como la sintaxis list() o .(). El uso de la sintaxis .() en el argumento **by** permite renombrar las columnas sobre la marcha.

En los siguientes ejemplos:  
\* agrupamos el número de casos por hospital \* en los casos de 18 años o más, calculamos la media de altura y peso de los casos según el sexo y si se recuperaron o murieron \* en los ingresos que duraron más de 7 días, contamos el número de casos según el mes en que ingresaron y el hospital en el que lo hicieron

Data.table también permite encadenar expresiones de la siguiente manera:

En estos ejemplos estamos siguiendo la suposición de que una fila en la tabla de datos es igual a un nuevo caso, y por lo tanto podemos utilizar el **.N** para representar el número de filas en la tabla de datos. Otra función útil para representar el número de casos únicos es uniqueN(), que devuelve el número de valores únicos en una entrada dada. Esto se ilustra aquí:

La respuesta es 3, ya que los valores únicos de la columna de género son m, f y N/A. Compárelo con la función base de R unique(), que devuelve todos los valores únicos en una entrada dada:

Para hallar el número de casos únicos en un mes determinado escribiríamos lo siguiente:

## Añadir y actualizar a las tablas de datos

El operador := se utiliza para añadir o actualizar datos en una tabla de datos. La adición de columnas a la tabla de datos puede hacerse de las siguientes maneras:

Las agregaciones más complejas están fuera del alcance de este capítulo introductorio, pero la idea es proporcionar una alternativa popular y viable a **dplyr** para agrupar y limpiar datos. El paquete **data.table** es un gran paquete que permite un código ordenado y legible.

## Recursos

A continuación, algunos recursos útiles para obtener más información: \* <https://cran.r-project.org/web/packages/data.table/vignettes/datatable-intro.html>\* <https://github.com/Rdatatable/data.table>\* [https://s3.amazonaws.com/assets.datacamp.com/img/blog/data+tabla+cheat+sheet.pdf](https://s3.amazonaws.com/assets.datacamp.com/img/blog/data+table+cheat+sheet.pdf) \* <https://www.machinelearningplus.com/data-manipulation/datatable-in-r-complete-guide/>\* <https://www.datacamp.com/community/tutorials/data-table-r-tutorial>

Puedes realizar cualquier función de resumen sobre datos agrupados; consulte la hoja de trucos aquí para obtener más información: <https://s3.amazonaws.com/assets.datacamp.com/blog_assets/datatable_Cheat_Sheet_R.pdf>