RANDALL PRUIM, NICHOLAS J. HORTON, AND

DANIEL T. KAPLAN

EMPIECE A ENSEÑAR CON R

Randall Pruim
Nicholas J. Horton
Daniel T. Kaplan

Start Teaching with R

Project MOSAIC

Edition 1.1, November 2015

Cover Photo: Maya Hanna

Índice general

Acerca de estas notas

Aquí presentamos un abordaje a la enseñanza introductoria e intermedia en cursos de estadística, la cual está acoplada importantemente con la computación general, y con R y RStudio RStudio en particular. Estas actividades y ejemplos tienen la intención de resaltar el abordaje moderno de la enseñanza estadística que se concentra en el modelaje, la inferencia basada en el re-muestreo y las técnicas de gráficos multivariados. Una meta secundaria es facilitar los cálculos con datos mediante el uso de pequeños estudios de simulaciones y el flujo de trabajo de análisis estadístico apropiado. Esto sigue la filosofía descrita por Nolan y Temple Lang1[1]. La importancia de la computación moderna en la educación estadística consiste en que esta es un componente principal de los lineamientos de currículo recientemente adoptados de la Asociación Americana de Estadística[2].

En este libro (y en sus volúmenes acompañantes), introduciremos múltiples actividades, algunas apropiadas para un curso introductorio, otras maleables para niveles superiores, que demuestran conceptos claves de estadística y modelaje, mientras también ofrecen ayuda al material de base de cursos más tradicionales.

[1] D. Nolan and D. Temple Lang. Computing in the statistics curriculum. *The American Statistician*, 64(2):97–107, 2010

[2] ASA Undergraduate Guidelines Workgroup. 2014 curriculum guidelines for undergraduate programs in statistical science. Technical report, American Statistical Association, November 2014. `http://www.amstat.org/education/curriculumguidelines.cfm`

Un trabajo en progreso

Estos materiales fueron construidos para un taller llamado *Teaching Statistics Using R* (Enseñando estadística con R) anterior a la United States Conference on Teaching Statistics y fue corregido para USCOTS 2011, USCOTS 2013, eCOTS 2014, ICOTS 9, and USCOTS 2015.

Organizamos todos estos talleres para ayudar a los instructores a integrar R R (así como otras tecnologías relacionadas) en los cursos de estadística de todos los

Precaución!
A pesar de nuestros mejores esfuerzos, VA a encontrar fallas en este documento y en nuestro código. Por favor háganos saber cuándo las encuentre para poder resolverlas..

niveles.

Recibimos una gran retroalimentación y bastantes buenas ideas por parte de participantes y por parte de aquellos con los que compartimos este material en los talleres.

Considere las notas como un trabajo en progreso. Apreciaremos cualquier retroalimentación que esté dispuesto a compartirnos, conforme continuamos el trabajo en estos materiales y el acompañante del material, el paquete `mosaic`. Envíenos un email a `pis@mosaic-web.org` con cualquier comentario, sugerencia, corrección, etc

Versiones actualizadas del paquete serán publicadas en `http://mosaic-web.org`.

Dos audiencias

Inicialmente desarrollamos estos materiales para instructores y profesores de estadística a nivel universitario. Otra audiencia para la cual desarrollamos el material son los estudiantes a quién estos instructores o profesores enseñan. Algunas de las secciones, algunos de los ejemplos y algunos ejercicios están pensados para una u otra de estas audiencias, con una mayor claridad en el énfasis que se le da a la audiencia. Esto significa que:

1. Algunos materiales son utilizados esencialmente para los estudiantes.

2. Algunos materiales apuntan a equipar al instructor a explotar sus propias capacidades en R and RStudio para desarrollar su propio material de enseñanza.

Aunque la distinción puede volverse confusa, y lo que funciona .ªsí.en una condición puede no funcionar .ªsí.en otras, intentaremos indicarle cuales partes se ajustan a cual audiencia conforme vayamos por ella.

R, RStudio y los paquetes de R

R puede ser obtenido `http://cran.r-project.org/`. La descarga e instalación es abierta y sencilla para máquinas con MAC, PC o linux

RStudio es una interfaz de desarrollo integrada (IDE por sus siglas en inglés) que facilita el uso de R tanto para usuarios sin experiencia como usuarios expertos. Lo

hemos adaptado como nuestra interfaz standard de enseñanza puesto que simplifica drásticamente el uso de R para instructores y estudiantes.

RStudio está disponible en `http://www.rstudio.org/`.

RStudio . RStudio puede instalarse como aplicación de escritorio o como una aplicación en un servidor accesible a los usuarios vía Internet.

Además de R y RStudio haremos uso de algunos paquetes que necesitan ser instalados y cargados separadamente. El paquete `mosaic` (y sus dependencias) serán utilizados en todo momento. Otros paquetes aparecerán esporádicamente.

MÁS INFO
Algunas cosas sólo pueden ser ejecutadas en RStudio, como la función `manipulate()` y la herramienta de RStudio para la investigación reproducible.

La versión en un servidor web de RStudio funciona bien con estudiantes que están iniciando. Lo único que se necesita es un navegador web, evadiendo todos los potenciales problemas con particularidades de la computadora personal de cada estudiante.

Notas marginales

Las notas marginales aparecerán por aquí y por allá En ocasiones estas son comentarios extra que deseábamos hacer, pero no queríamos interrumpir el flujo de la lectura y mencionarlos en el texto principal. Otras proveen consejos para la enseñanza o precauciones acerca de trampas, caídas y 'jugadas'.

¿Tiene alguna sugerencia para una nota marginal? Compártala con nosotros.

Lo que es nuestro, es suyo, hasta cierto punto

Este material se encuentra en copyright por los autores, bajo 'Creative Commons Attribution 3.0 Unported License'. Está libre para *compartirlo* (copiarlo, distribuirlo y transferir el trabajo) y *mezclarlo* (adaptar el trabajo) si nos atribuye el trabajo realizado. Para información más detallada sobre esto, está disponible en el siguiente sitio web: `http://www.mosaic-web.org/go/teachingRlicense.html`.

Creación del documento

El documento original fue creado el 12 de Agosto de 2016, utilizando: knitr, versión 1.13.2 mosaic, versión 0.14.9000 *R version 3.3.0 (2016-05-03)

Inevitablemente, cada una de estas va ser actualizada en ciertas ocasiones. Si usted encuentra cosas que se ven diferente en su computadora, asegúrese que su versión de R y sus paquetes estén actualizados y revise versiones más nuevas de este documento

CAVANDO HONDO
Si usted conoce LaTeX tan bien como R, entonces el paquete knitr ofrece una buena solución para mezclar ambos. Nosotros utilizamos este sistema para producir este libro. También lo utilizamos para nuestra propia investigación y para introducir a estudiantes de nivel superior a métodos de análisis reproducible. Para principiantes, introducimos knitr con RMarkdown, el cual produce archivos en formato PDF, HTML o Word utilizando sintaxis sencilla

Honores y agradecimientos a Joseph Cappelleri por sus comentarios útiles en los primeros borradores de este material

1
Proyecto MOSAIC

Este libro es producto del Proyecto MOSAIC, una comunidad de educadores trabajando para desarrollar nuevas formas de introducir las matemáticas, la estadística, la computación y el modelaje de datos a estudiantes universitarios.

La meta del proyecto MOSAIC es ayudar a compartir ideas y recursos para mejorar la enseñanza, y desarrollar una estructura curricular y valorativa para dar apoyo a la diseminación y evaluación de estos esfuerzos. Nuestra meta es proveer un amplio acercamiento a los estudios cuantitativos que brindan una mejor herramienta al trabajo en la ciencia y tecnología. El proyecto resalta e integra diversos aspectos del trabajo cuantitativo que los estudiantes de ciencia, tecnología e ingeniería necesitarán en sus vidas profesionales, pero que actualmente es usual que sean enseñados de forma aislada, si es que acaso se enseñan. En particular, nos concentramos en:

En particular nos enfocamos en:

Modelaje La habilidad de crear, manipular e investigar representaciones matemáticas útiles e informativas de situaciones del mundo real.

Estadística El análisis de variabilidad que se esboza de nuestra habilidad de cuantificar la incertidumbre y generar inferencias lógicas de observación y experimentación.

Computación La capacidad de pensar algorítmicamente, de manejar datos en gran escala, visualizar e interac-

tuar con modelos, y automatizar tareas para la eficiencia, precisión y reproducibilidad.

Cálculo El punto de entrada tradicional para estudiantes universitarios y un tema que todavía hoy tiene la capacidad de dotar de nociones importantes a los estudiantes

Tomando el apoyo de la Fundación Nacional de Ciencia de Estados Unidos (NSF DUE-0920350), Proyecto MOSAIC apoya algunas iniciativas para ayudar a lograr estas metas, incluyendo:

Desarrollo de facultades y oportunidad de entrenamiento así como en USCOTS 2011, USCOTS 2013, eCOTS 2014, e ICOTS9 en los talleres *'Teaching Statistics Using* R *y* RStudio, en 2010 nuestro Proyecto MOSAIC inició talleres en el Instituto de Matemática y sus aplicaciones, y nuestro taller *Modeling: Early and Often in Undergraduate Calculus* AMS PREP fueron ofrecidos en 2012, 2013, and 2015.

M-casts, una serie de seminarios web programados regularmente, dados vía Internet, en el que se creó foro para que los instructores puedan compartir sus observaciones e innovaciones y desarrollar colaboraciones para refinar y desarrollar las mismas. Las grabaciones de los M-casts están disponibles en el sitio web del Proyecto MOSAIC, `http://mosaic-web.org`.

La construcción de un programa y materiales para cursos que enseña MOSAIC con tópicos integrados de la mejor manera. Estos Estos cursos y materiales podrían ser completamente nuevas construcciones, o podrían ser modificaciones de recursos existentes que se aproximen a las conexiones entre los tópicos de MOSAIC.

Más detalles pueden ser encontrados en `http://www.mosaic-web.org`. Le damos la bienvenida e incitamos su participación en todas estas iniciativas.

Estadística computacional

Hay al menos dos formas en las cuales se puede introducir un software estadístico en un curso de estadística. En el primer abordaje, el curso debe ser enseñado esencialmente como se hacía anterior a la utilización de los software estadísticos, pero utilizando la computadora para hacer más rápidos algunos cálculos y generar gráficos de alta calidad displays. PProbablemente el tamaño de los conjuntos de datos también deba ser incrementado. Nos referiremos a este enfoque como **computación estadística** puesto que la computadora funciona primariamente como una herramienta de cálculo para reemplazar los cálculos en lápiz y papel, además del dibujo manual de los gráficos.

En el segundo abordaje, cambios más importantes en el curso resultan de la introducción de la computadora. Algunos temas nuevos son cubiertos, algunos temas anteriores son omitidos. Algunos temas anteriores son tratados en formas muy diferentes, y quizás en diferentes etapas en el curso. A este abordaje nos referiremos como **estadística computacional** bporque la disponibilidad de la computación está moldeando como la estadística es construida y enseñada. La estadística computacional es un componente clave de la **ciencia de los datos**, definida como la habilidad de usar datos para responder preguntas y comunicar resultados.

En la práctica, la mayoría de los cursos van a incorporar elementos de ambas, la estadística computacional y la computación estadística, pero las proporciones relativas pueden diferir drásticamente de curso a curso. En qué parte del espectro se encuentra el curso va depender de muchos factores, incluyendo las metas del curso, la disponibilidad de tecnología para el uso de los estudiantes, la perspectiva del libro utilizado y el nivel de 'comfort'que

Los estudiantes necesitan ver los aspectos de la computación y la ciencia de los datos de manera pronta y cotidiana para desarrollar habilidades profundas. Establecer bases en cursos introductorios ayuda a que inicien de mejor manera.

sienta el instructor con la estadística y la computación.

Entre los varios paquetes de software estadístico disponible, R está incrementando su popularidad. La reciente adición de RStudio ha hecho R más poderoso y accesible. Como R y RStudio son gratis, ambos se han vuelto ampliamente utilizados en la investigación y la industria. Entrenar en R y RStudio es normalmente visto como una habilidad adicional importante que un curso de estadística puede desarrollar. Por esto, un alto número de instructores están utilizando R para su propio trabajo estadístico; por lo que es natural que inicien a integrarlo en la enseñanza también. Al mismo tiempo, el desarrollo de R y RStudio (una interfaz opcional e integrada al desarrollo de un entorno para R) están haciendo cada vez más fácil iniciar la utilización de R.

Desarrollamos el paquete de R `mosaic` (disponible en CRAN), para hacer ciertos aspectos de la computación estadística y la estadística computacional más simple para inexpertos, sin limitar su habilidad a utilizar características del lenguaje. El paquete `mosaic` incluye un acercamiento al modelaje, que utiliza la misma sintaxis general para calcular estadísticas descriptivas, crear gráficos y ajustar modelos lineales.

La información sobre el paquete `mosaic`, incluyendo ejemplos de características y material suplementario (así como este libro) puede encontrarla en `https://cran.r-project.org/web/packages/mosaic`.

2
Algunos Consejos para empezar con R

Aprender R es un proceso gradual, arrancar de buena forma asegura que al final del camino todo sea un éxito. En este capítulo discutimos sobre estrategias y técnicas para empezar a enseñar estadística con R. En los capítulos siguientes, le brindaremos más detalles acerca de (los pocos) comandos que los estudiantes necesitan saber, además de alguna información necesaria que es útil que los instructores sepan. En el camino, presentamos algunos de nuestros ejemplos favoritos que resalta el uso de R incluyendo algunos que pueden ser utilizados muy pronto en el curso.

El paquete `mosaic` incluye una viñeta resumiendo una serie de comandos de R para enseñar un curso introductorio.

2.1 Estrategias

Cada instructor elige como empezar su curso, sin embargo, nosotros ofrecemos las siguientes estrategias (seguidas por tácticas y ejemplos) que pueden servir como guía para empezar el curso de forma que prepare a los estudiantes exitosamente con R.

1. Empiece lo más pronto posible.

   Haga algo con él en el día 1. Haga algo más en el día 2. Haga que los estudiantes hagan algo para el final de la semana 1.

2. Ilustre frecuentemente.

   Tenga R ejecutándose en todo momento de clase y utilícelo conforme sea necesario durante el curso, así los estudiantes pueden ver lo que R hace. Presente los temas antes de pedirle a los estudiantes hacer algo.

Consejo de Enseñanza RMarkdown proporciona una forma fácil de crear documentos o diapositivas para sus estudiantes. Para más información sobre la integración de RMarkdown en su curso, vea *R Markdown: Integrating a Reproducible Analysis Tool into Introductory Statistics* por B Baumer *et al*. Para aquellos que ya esten familiarizados con LaTeX, también existe una forma de integrar `knitr`/LaTeXen RStudio.

3. Enseñe R como un lenguaje (Pero no abuse de ello).

   Hay un poco de sintaxis que se debe aprender – entonces enséñela de forma explícita

   - Enfatice que la utilización de mayúsculas y minúsculas, así como la ortografía cuentan
   - Explique con mucho cuidado (y repetidamente) la sintaxis de las funciones.

     Afortunadamente la sintaxis es muy directa. Esta consiste en el nombre de una función, seguido por un paréntesis abierto, seguido por listas de argumento, seguido por el paréntesis de cierre

     ```
     functionname ( name1=arg1, name2=arg2, ... )
     ```

     Haga que los estudientes piensen que hace la función y que necesita saber para que lo haga. Generalmente el nombre de la función indica qué hace la función. Los argumentos le dan a la función la información necesaria para que haga su tarea.
   - Hay diferentes tipos de objeto en R. Pregunte frecuentemente: *¿Este objeto de qué tipo es?*

     Los estudiantes necesitan entender la diferencia entre una variable y un conjunto de datos, además de que existen variables (por ejemplo: `factor` para categórica, `numeric` para datos numéricos). Los instructores y los alumnos avanzados usualmente van a querer saber sobre los objetos de tipo `vector` y `list`.

   De más detalles en cursos de niveles más avanzados.

   Los estudiantes de niveles avanzados deberían aprender más sobre las funciones definidas por el usuario y las estructuras de control del lenguaje, como los loops y los condicionales. No es necesario saber tanto del lenguaje para estudiantes en cursos introductorios.

4. "Menos vólumen, más creatividad"[Mike McCarthy, entrenador, Green Bay Packers]

   Use frecuentemente pocos métodos y los estudiantes van a aprender a usarlos bien, flexiblemente e incluso creativamente. Enfóquese en una cantidad pequeña de tipos de datos: numéricos, vectores, cadenas de signos

NOTA

Esta es una de las motivaciones primarias detrás de nuestro paquete, `mosaic`, el cual busca hacer las cosas más simples y más similares; de esta forma los estudiantes pueden fácilmente volverse usuarios de R creativos e independientes. Incluso si usted decide no optar por hacer las cosas exactamente como las recomendamos, utilice el "Menos volumen, más creatividadçomo guía. .

y "dataframes". Elija funciones que empleen estructuras y estilos similares, para aumentar la habilidad de los estudiantes de transferir conocimiento de una situación a otra.

5. Encuentre una forma de tener computadores disponibles para exámenes.

   Esto hace que el examen vaya de acuerdo al resto del curso y motiva a los estudiantes a aprender R. Esto también cambia lo que se puede preguntar en los exámenes.

   Uno de nosotros hizo esto primero por la petición de un estudiante de un curso de estadística introductoria que pidió utilizar la computadora durante el examen "*puesto que así hacía toda la tarea*". Ahora, él hace a sus estudiantes traer las laptops para los exámenes. Otro de nosotros utiliza los componentes de en clase (sin computadora) y fuera de clase (con computadora) para la evaluación

6. Piense nuevamente su curso.

   Si usted siempre ha enseñado sin computadores, o con relativamente poco uso de estos en el pasado, tal vez sea necesario que lo piense nuevamente. Con la extensión computacional, algunas cosas desaparecen del curso

   - Usar tablas estadísticas.

     ¿Alguien todavía consulta tablas para los valores de seno o log Todos nos hemos negado al uso de tabulaciones de valores críticos de una distribución (nosotros no lo usamos en nuestro trabajo profesional. ¿Por qué entonces es necesario enseñárselo a estudiantes?)
   - Fórmulas de cálculo.

     Reemplácelas con computación. Enseñe solo las fórmulas más intuitivas. Enfóquese en como llevan a la intuición y el entendimiento, *no* el cálculo
   - (Casi) todos los cálculos a mano.

   Al mismo tiempo, otras cosas se volvieron posibles que no eran posibles antes:

   - Conjuntos de datos grandes

NOTA
Uno de los principales usos de la calculadora en los exámenes de AP Statistics es el cálculo de valores p y sus cuantilos relacionados.

- Gráficos agradables
- Simulaciones y métodos basados en la aleatorización y remuestreo.
- Cálculos rápidos
- Centralización en conceptos, en lugar de cálculos

Haga a sus estudiantes pensar que la computadores es parte de cómo ahora se hace estadística, en lugar de solamente un complemento.

7. Mantenga el mensaje lo más simple posible y mantenga los comandos de la manera más simple

   Particularmente cuando esté haciendo gráficos, tenga cuidado de distraer a los estudiantes con detalles complejos de embellecimiento de publicaciones. Si el predeterminado es lo suficientemente bueno, déjelo así.

Es importante no empezar cosas muy complicadas de manera apresurada. En un inicio, es mejor utilizar los ajustes en predeterminado y centrarse en las ideas principales. Más tarde, se puede introducir opciones más finas conforme los estudiantes se sienten a gusto con las cosas simples (y usualmente están pidiendo más)

8. Anticipe a estudiantes con problemas con la computadora, pero sea confiado de que usted los guiará por el camino correcto.

   Algunos estudiantes entienden R muy fácil. En todos los cursos habrá estudiantes que tendrán problemas. Para ayudarlos, enfóquese en diagnosticar lo que ellos no saben y cómo ayudarlos a solucionarlo

   La experiencia nos dice que la computadora es un chivo expiatorio de otras cosas que el estudiante no entiende. Como la computadora da respuestas inmediatas, revela estos malos entendidos. Por ejemplo, si los estudiantes están confundidos sobre la distinción de variable, estadística y unidades observaciones, va tener un rato complicado para brindarle la información correcta a una función de gráficos. Los estudiantes pueden culpar a R, pero eso no es el componente primario de la dificultad. Si tiene la capacidad de diagnosticar el verdadero problema, va mejorar el entendimiento de la estadística y arreglar dificultades con R simultáneamente.

   Incluso estudiantes con un entendimiento sólido de conceptos estadísticos se van a encontrar con errores de R que no pueden depurar. Dígales a los estudiantes que copien y peguen los mensajes de error en un

CONSEJO DE ENSEÑANZA
Cuando introduzca el código de R a los estudiantes, enfatice las siguientes preguntas: *¿Qué quiere que* R *haga por usted?* y *¿Qué información debe proveerle a* R *para que logre hacer eso?* La primera pregunta generalmente determina la función que se va utilizar. La segunda determina los elementos de entrada que necesita la función.

CONSEJO DE ENSEÑANZA
Dígale a sus estudiantes que copien y pegue mensajes de error en un email en lugar de describirlos vagamente. Es un gran salvador de tiempo.

email cuando tienen problemas. Cuando les conteste, explíquele como el mensaje de error le ayudó a diagnosticar su problema y ayúdelo a generalizarlo para solucionar otras situaciones. Vea el capítulo 8 para algunos de los mensajes de error más comunes y lo que podrían indicar.

2.2 *Tácticas*

Los estudiantes deben aprender a ver antes de ver para aprender

1. Introduzca los gráficos tempranamente

   Introduzca gráficos de forma muy pronta, para que los estudiantes vean como pueden conseguir salidas impresionantes de comandos simples. Intente eliminar expectativas anteriores de que hay una "curva empinada de conocimiento."

   Acepte los predeterminados – no se preocupe por la belleza (los ejes, tamaños de la clase del histograma, colores) demasiado pronto. Déjelos sentirse cómodos con los comandos de gráficos básicos y después juegue (asegúrese de que se siente como jugar) haciendo las cosas más finas.

   Tenga en mente que sólo porque los gráficos son fáciles de generar en la computadora no significa que los estudiantes sepan cómo leer los gráficos. Use ejemplos que ayuden a los estudiantes a generar buenos hábitos de visualización de datos.

2. Introduzca el muestreo y la aleatorización prontamente .

   Puesto que el muestreo dirige una buena porción de la lógica estadística, introduzca la idea de las muestras aleatorias tempranamente y haga a los estudiantes construir sus propias muestras aleatorias. Este fenómeno de las distribuciones muestrales puede ser introducido de forma intuitiva, ajustándolo como un tema para discusión tardía y análisis.

Para mantener este consejo, la mayoría de los ejemplos en este libro caen en el área del análisis de datos exploratorio. La organización de esto se eligió para desarrollar gradualmente un entendimiento de R. Vea el volumen acompañante, *Una guía para estudiantes de* R para un recorrido de comandos utilizados en los primeros dos cursos de un pregrado de estadística. El volumen acompañante está organizado por tipos de análisis de datos y supone cierta familiaridad con el lenguaje R.

3
Iniciando con RStudio

RStudio es un entorno integrado de desarrollo (IDE por sus siglas en inglés) para R que proporciona una alternativa de interface de R, que tiene algunas ventajas sobre la interfaz predeterminada de R.

- RStudio es ejecutable en máquinas Mac, PC o Linux y brinda una interfaz simplificada que se ve y se siente *idéntico en todas ellas*

  La interfaz predeterminada de R es un poco diferente en las distintas plataformas. Esto es un distractor para los estudiantes y agrega una responsabilidad extra de conocimiento al instructor.

- RStudio puede ejecutarse desde un navegador web

  Además de la versión para escritorio, RStudio puede ser utilizado y configurado como una aplicación en un servidor al que se tiene acceso vía internet. La instlación es sencilla para cualquiera que sepa utilizar un sistema Linux. Una vez que el servidor se haya ajustado en su institución, los estudiantes puede empezar a usar RStudio simplemente abriendo el sitio web e iniciando sesión. No es necesaria ninguna instalación adicional.

  La interfaz web es prácticamente idéntica a la versión de escritorio. Como con otros servicios web, el usuario debe hacer un inicio de sesión para acceder a su cuenta. Si los estudiantes cierran sesión y abren sesión después, incluso en una máquina diferente, la sesión es restaurada y pueden continuar sus análisis justo donde los dejaron. Con un poco de ajustes avanzados, los instructores pueden salvar el registro de su utilización de

NOTA
Usar RStudio en un navegador es como Facebook para estadística. Cada vez que el usuario vuelve, la sesión previa es restaurada y puede continuar el trabajo donde lo dejó. Los usuarios pueden iniciar sesión desde cualquier dispositivo con acceso a internet.

PRECAUCIÓN!
La versión de escritorio y la versión del servidor de RStudio son tan similares que si las ejecuta al mismo tiempo, tendrá que prestar mucha atención y asegurarse de que esté trabajan o en la que tenía intención de trabajar

R en clase y pueden cargar estos archivos de registro en su propio entorno.

- RStudio brinda apoyo para la investigación reproducible.

  RStudiofacilita incluir texto, análisis estadístico (código de R y salidas de R), y gráficos, todo en un mismo documento. El sistema de RMarkdown proporciona una adición en el lenguaje y produce documentos en HTML. Elsistema knitr/LaTeX permite a los usuarios combinar Ry LaTeXen el mismo documento. La recompensa por aprender este sistema más complicado es un control más fino sobre el formato del documento de salida. Dependiendo del nivel del curso, los estudiantes pueden usar esto para tareas y proyectos.

  RStudio provee una interfaz funcional de gráficos para el usuario. RStudio no es un GUI para R, pero este sí brinda una GUI que simplifica cosas como instalar y actualizar los paquetes; monitorear, guardar y cargar entornos; importar y exportar datos; navegar y exportar gráficos; y navegar por archivos y documentación.

- RStudio ofrece una opción integrada para editar y ejecutar código de R y documentos

- RStudio provee una interfaz funcional de gráficos para el usuario.

  RStudio no es un GUI para R, pero este sí brinda una GUI que simplifica cosas como instalar y actualizar los paquetes; monitorear, guardar y cargar entornos; importar y exportar datos; navegar y exportar gráficos; y navegar por archivos y documentación.

- RStudio permite acceso al paquete `manipulate`.

  El paquete `manipulate` ofrece una forma fácil de crear aplicaciones gráficas que sean interactivas, fáciles y rápidas

Realmente se puede usar R sin RStudio, RStudio hace una cantidad considerable de cosas más sencillas, recomendamos vigorosamente la utilización de RStudio. Además puesto que RStudio está en desarrollo activo, tenemos la expectativa de más características útiles en el futuro.

NOTA

Para usar Markdown o `knitr`/LaTeXse requiere que el paquete `knitr` esté instalado en su sistema.

3.1 Ajustando R y RStudio

RStudio puede ser obtenido en `http://cran.r-project.org/`. La descarga y la instalación están dirigidas a dispositivos Mac, PC o Linux. RStudio está disponible en `http://www.rstudio.org/`. RStudio puede ser instalado como una aplicación de escritorio o una aplicación de servidor.

3.1.1 RStudio en la nube

Nosotros primariamente usamos una versión en línea de RStudio. RStudio es una interfaz innovadora y poderosa de Rque se ejecuta en un navegador web o en el dispositivo local. Ejecutarlo en el navegador web tiene la ventaja de que no se tiene que instalar o configurar nada. Sólo se debe iniciar sesión y listo. Entonces, RStudio va recordar"lo que usted estaba haciendo cada vez que inicie sesión (incluso en dispositivos diferentes) lo puede retomar justo donde lo dejó. Esto es R en la nubeƶ funciona un poco como GoogleDocs o Facebook para R.

Es posible que el administrador de su sistema necesite ajustar la instalación de RStudio para su institución, pero le aseguramos que este proceso es sumamente sencillo y facilitará a los estudiantes y la facultad el uso.

3.1.2 RStudio en su computadora

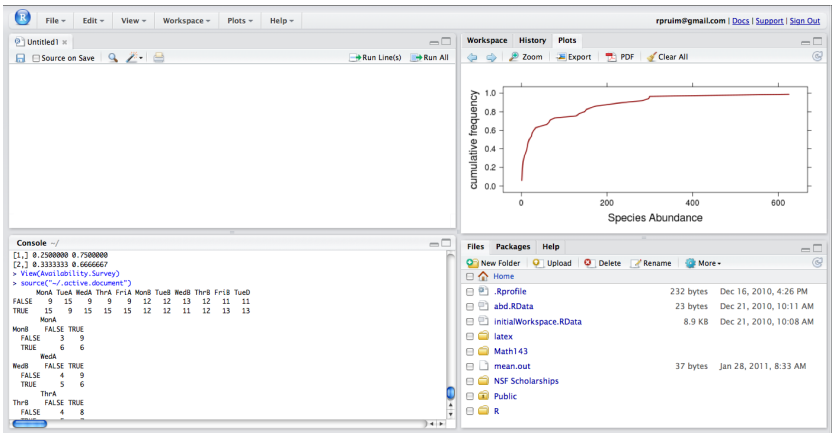
Hay también una versión autónoma del entorno de RStudio que se puede instalar en su computadora. Esta puede ser descargada de `http://www.rstudio.org/`. Se supone que usted tiene una versión de R en su computadora (observe más abajo instrucciones para descargarlo de CRAN). Incluso si sus estudiantes están primaria o exclusivamente utilizando en la edición del web de RStudio, los instructores deberían tener una manta de seguridad de la versión que no requiere acceso a internet. Sea precavido, las dos versiones se ven tan similares que ocasionalmente se puede encontrar a usted mismo trabajando en una cuando tenía la intención de trabajar en la otra.

3.1.3 *Consiguiendo R de CRAN*

CRAN es el Comprehensive R Archive Network (`http://cran.r-project.org/`). Puede descargar versiones gratuitas de R para pc, Mac y Linux de CRAN. (Si usa la versión autónoma de RStudio, necesita instalar R de esta forma primero). Todas las instrucciones para instalarlo están en CRAN. Sólo siga las instrucciones apropiadas para su plataforma.

3.1.4 *Ejecutar RStudio por la primera vez*

Una vez que haya iniciado la versión de escritorio o haya iniciado sesión en el servidor de RStudio, usted verá algo como lo siguiente:



Observe como RStudio divide su universo en cuatro paneles. Algunos paneles después son subdivididos en múltiples viñetas. Qué viñetas aparecen y cuales paneles aparecen puede ser personalizado por el usuario.

Consejo de Enseñanza
Nosotros encontramos conveniente poner la consola en la parte superior derecha en lugar de en su ubicación predeterminada (parte superior izquierda) así los estudiantes pueden ver mejor cuando proyectamos R en clase

3.2 *Usando la consola de R como calculadora*

R puede hacer mucho más que una simple calculadora, y vamos a introducir estas características en su debido tiempo. Pero representar cálculos simples en R es una buena forma de comenzar a aprender las características de RStudio

Los comandos ingresados en la viñeta de console son inmediatamente ejecutados por R. Una buena forma de familiarizarse con la consola es hacer cálculos simples. Intente escribir los siguientes comandos en el panel de la consola.

```
5 + 3

## [1] 8

15.3 * 23.4

## [1] 358

sqrt(16)              # square root

## [1] 4
```

Este último ejemplo demuestra como las funciones son llamadas dentro de R como también el uso de comentarios. Los comentarios son anticipados por el símbolo #. Los comentarios pueden ser de mucha ayuda cuando se escriben scripts con múltiples comandos o para anotar código de ejemplo para sus estudiantes.

Usted puede guardar valores en variables nombradas para reutilización posterior.

```
product = 15.3 * 23.4   # save result
product                 # display the result

## [1] 358

product <- 15.3 * 23.4  # <- instead of =
product

## [1] 358
```

Una vez que las variables son definidas, pueden ser referenciadas en otras operaciones y funciones.

```
0.5 * product        # half of the product

## [1] 179

log(product)         # (natural) log of the product
```

Consejo de Enseñanza
Probablemente es mejor establecer el uso de uno u otro operador de asignación en lugar de estar cambiándolo. Personalmente preferimos el operador de la flecha, porque representa visualmente lo que está pasando en la asignación además porque deja clara la distinción entre el operador de asignación, el uso de = en este caso es para dar valores a los argumentos de funciones, y el uso de == para probar la igualdad.

```
## [1] 5.88

log10(product)       # base 10 log of the product

## [1] 2.55

log2(product)        # base 2 log of the product

## [1] 8.48

log(product, base=2) # another way for base 2 log

## [1] 8.48
```

El punto y coma puede ser utilizado para ubicar comandos múltiples en una sola línea. Un uso frecuente de esto puede ser para salvar y después imprimir un valor, en una sola línea:

```
# store and show result
product <- 15.3 * 23.4; product

## [1] 358
```

3.3 *Trabajando con archivos*

3.3.1 *Archivos* R *script*

Como alternativa, los comandos de R pueden ser almacenados en un archivo. RStudio ofrece un editor integrado para editar este tipo de documentos y facilita ejecutar algunos o todos los comandos. Para crear un archivo, seleccione en el menú de RStudio: File, después New File, después R Script. Una ventana del editor de archivos se va abrir en el panel de Source. El código de R puede ser escrito ahí, y botones e ítems de menú son provistos para correr todo el código (llamado abastecer el documento) o ejecutar el código en una sola línea o en una sección seleccionada del documento.

3.3.2 *RMarkdown, y knitr/LaTeX*

Una tercera alternativa es tomar ventaja del apoyo que brinda RStudio para la investigación reproducible. Si usted ya conoce LaTeX, va a querer investigar las capacidades de `knitr`/LaTeX. Para aquellos que no conocen LaTeXtodavía, el sistema de RMarkdown es más sencillo y proporciona una entrada fácil en el mundo de los métodos de la investigación reproducible. Además brinda una buena estructura para que los estudiantes creen sus tareas y reportes que incluyen texto, código de R, salidas de Ry gráficos.

Para crear un nuevo archivo de RMarkdown, seleccione la pestaña de File, después New File, después RMarkdown.El archivo va ser abierto con un pequeño documento de plantilla que ilustra la especificación que tiene el lenguaje. Si puede hace click en From template, antes de crear el documento, se le dará una lista de documentos de plantilla disponibles en paquetes. Si el paquete `mosaic` está cargado, esta lista va incluir plantillas que se aseguran que el paquete `mosaic` este cargado y cambie los tamaños predeterminados para los gráficos, para que sean un poco más pequeños que los genericos de RStudio. La versión fancy muestra muchas de las características de RMarkdown. (El sitio de RStudio incluye un tutorial extensivo de como usar RMarkdown que demuestra un rango de características más amplio). La plantilla plain es diseñada para crear rapidamente nuevos documentos empezando casi en blanco.

El proceso de ejecutar código de R y combinar texto, código de R, salidas, y gráficos es un mismo archivo se llama “knitting". Presione en knit para convertir el documento RMarkdown en un archivo HTML, PDF, o Word.

Es importante recordar que contrario a los scripts de R que son ejecutados en la consola y tiene acceso al entorno de la consola, los archivos de RMarkdown y knitr/LaTeX no tienen acceso al entorno de la consola. Esto es una buena característica puesto que obliga a los archivos a ser auto-contenidos, lo cual los hace transferibles y respeta las buenas prácticas de la investigación reproducible. Pero en el caso de los principiantes, especialmente si adoptan una estrategia de probar cosas en la consola, y copiar y pegar código de la consola en su documento, crean archi-

vos que son incompletos y por esto no compilan correctamente.

Una buena estrategia para que los estudiantes usen RMarkdown es proporcionarles un documento de ejmemplo que incluya la plantilla que usted desee que utilicen, cargue cualquier paquete de R que necesiten, ajuste las opciones del paquete `knitr` u opciones de R de la forma que usted las prefire, y tiene lugares para poner el trabajo donde usted quiere que lo realicen.

3.4 Los otros paneles y viñetas

3.4.1 La viñeta de historial (History)

Como los comandos son ingresados en la consola, así aparecen en la viñeta de History. Estas "historias.º historiales pueden ser guardados y cargados, hay una opción de búsqueda para encontrar comandos previos, y líneas individuales previas o secciones pueden ser transferidas de vuelta a la consola. Mantener la viñeta de History abierta le permite ir atrás y ver varios comandos anteriores. Esto puede ser especialmente útil cuando ciertos comandos producen una salida grande entonces la pantalla se mueve hacia abajo rápidamente.

Una alternativa es producir documentos RMarkdown con instrucciones en el código.

3.4.2 Comunicación entre viñetas

RStudio brinda varias formas de mover código de Rentre las viñetas. Presionar el botón de Run en el panel de edición para un script de Ro `RMarkdown` o algún otro documento va copiar líneas de código en la consola y ejecutarlas.

3.4.3 La viñeta de archivos

La viñeta de archivos provee un manejador de datos simples. En esta se puede navegar de forma amigable y puede ser utilizada para abrir, renombrar y borrar archivos. En la versión de navegador web de RStudio, la pestaña de archivos también ofrece una utilidad para subir los

archivos de la máquina local al servidor. En archivos de `RMarkdown` y `knitr` también se puede ejecutar código en algún çhunk"del archivo o en todos los çhunks"del archivo. Cada una de estas características simplifica el intentar escribir código .[en] vivo"mientras se crea un documento que deja un historial del código.

En la dirección inversa, el código del historial puede ser copiado de nuevo en la consola para ejecutarlo de nuevo (después de un poco de edición) o en una de las viñetas de edición, para la inclusión en un archivo.

3.4.4 The Help Tab

La viñeta de ayuda (Help) es donde RStudio despliega los archivos de ayuda de R. Se puede abrir un archivo de ayuda usando el operador `?` en la consola.

Por ejemplo, el siguiente comando va mostrarnos el archivo de ayuda para la función de logaritmo.

```
?log
```

3.4.5 La pestaña de Entorno (Environment)

La viñeta de entorno (Environment) muestra los objetos disponibles para la consola. Estos están subdivididos en datos, valores (no son objetos de tipo conjunto de datos, ni objetos de tipo función) y funciones.

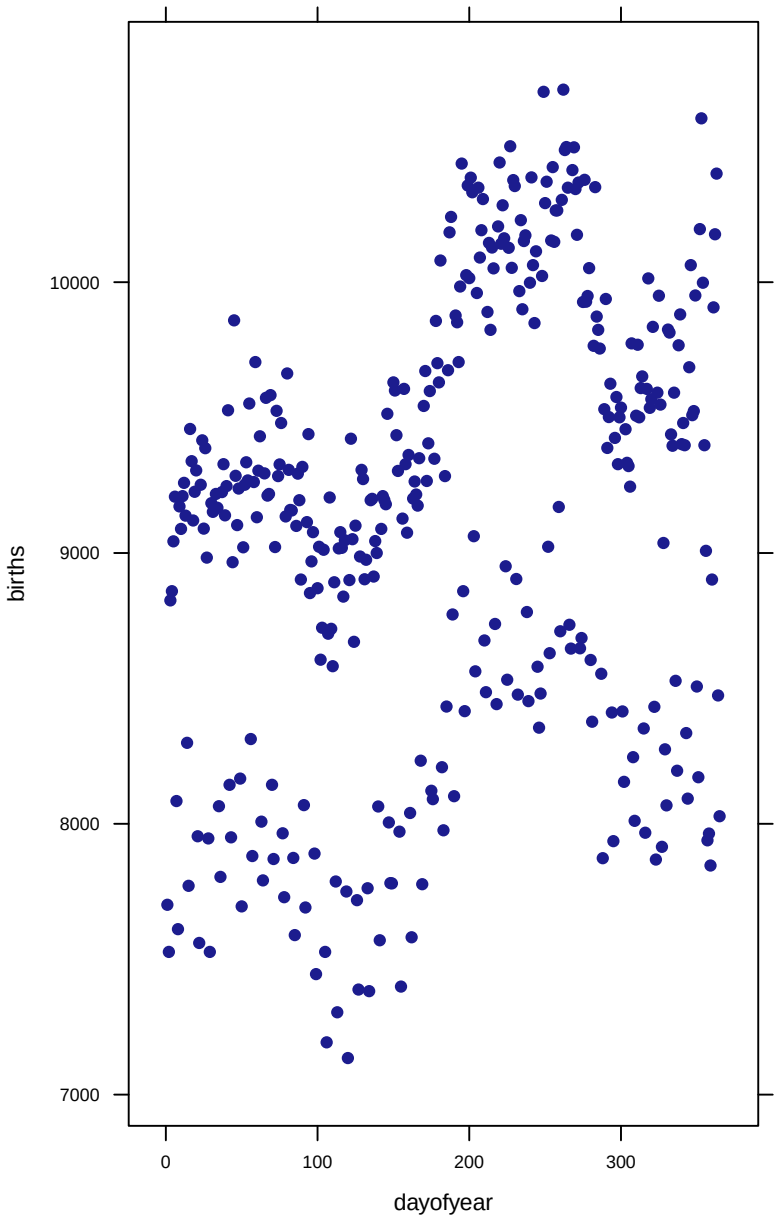
El ícono de la escoba puede ser utilizado para remover todos los objetos del entorno, y es bueno hacer esto de vez en cuando, especialmente cuando se está ejecutando RStudio en el servidor o si decide salvar el entorno cuando cierra RStudio, puesto que en estos casos los objetos se pueden quedar en el entorno indefinidamente.

3.4.6 La viñeta de gráficos

Los gráficos creados en la consola son desplegados en la viñeta de gráficos (Plots). Por ejemplo, los siguientes comandos despliegan el número de nacimientos en Estados Unidos cada día en 1978.

Si no ha estado ingresando los comandos que mostramos anteriormente, se le recomienda devolverse e ingresarlos.

```
#Esto hace los gráficos lattice disponibles
require(mosaic)
xyplot( births ~ dayofyear, data=Births78)
```



En la viñeta de gráficos, se puede navegar a gráficos anteriores y también exportar gráficos en varios formatos, después de interactivamente manipular las dimensiones.

3.4.7 La viñeta de paquetes

Mucha de la funcionalidad de R está localizada en paquetes, muchos de los cuales se pueden obtener de la casa matriz llamada CRAN (Comprehensive R Archive Network).

La viñeta de paquetes (packages) facilita instalar y cargar paquetes. También le permite buscar paquetes que han sido actualizados desde que usted los instaló.

4
Usando R tempranamente en cursos

Este capítulo incluye algunas de nuestras actividades favoritas para iniciar los cursos. Estas actividades simultáneamente les dan a los estudiantes su primer vistazo a R y a la introducción de algunos temas importantes del curso. Si lo vemos de esta forma, no es necesario que los estudiantes entiendan detalles del código de R. En lugar de eso, los centramos en las preguntas y como los resultados presentados les dan un çhispazo"de cómo responder estas preguntas.

4.1 Monedas y tasas: La dama y la degustación de té

Esta sección es una versión levemente modificada de un folleto que uno de los autores le da a sus estudiantes de introducción a la estadística en el Día 1 después al final de la actividad para generar una discusión en la clase

Hay un famoso relato sobre una dama que afirmaba que el té tenía un sabor diferente dependiendo de si la leche era se la agregaba té o si se le agregaba el té a la leche. El relato es famosos por el escenario en que ella hizo esta afirmación. Ella estaba en una fiesta en Cambridge, Inglaterra, en los años 1920s. A la fiesta también se presentaron varios catedráticos universitarios y sus esposas. Los científicos que asistieron se burlaron de la afirmación. ¿Qué, después de todo, era la diferencia?

Todos los científicos, excepto uno. En lugar de simplemente ignorar la afirmación de la dama, propuso que decidieran cómo se debía hacer una prueba de la afirmación. El tono de la conversación cambió después de la sugerencia del científico, y este mismo empezó a discutir como la afirmación debía probarse. En unos minutos, tasas de té con leche fueron preparadas y presentadas a la

mujer para que las probara.

En este punto, usted podría estar preguntándose quién fue el científico innovador y cuáles fueron los resultados del experimento. El científico fue R.A Fisher, quien primeramente describió la situación como un ejemplo pedagógico en su libro de metodología [1] estadística en 1925. Fisher desarrolló métodos estadísticos que son parte de los más importantes y mayormente utilizados a la fecha, y la mayoría de sus aplicaciones son biológicas.

[1] R. A. Fisher. *Statistical Methods for Research Workers*. Oliver & Boyd, 1925

Usted podría también tener curiosidad sobre cómo el experimento terminó. ¿Cuántas copas de té fueron preparadas? ¿Cuántas identificó la dama correctamente? ¿Cuál fue la conclusión?

Fisher nunca lo dice. En su libro, él está interesado en el método, no en los resultados en particular. Sin embargo, podemos usar estos ajustes para introducir algunas claves de estadística.

Supongamos que decidimos probar a la dama con diez tazas de té. Vamos a lanzar una moneda para decidir de qué forma preparar las tasas. Si la lanzamos y un escudo, tenemos que poner la leche primero; si es corona, ponemos primero el té. Después presentamos las diez tasas a la dama y le pedimos que nos diga cuales fueron hechas de qué manera.

Es sencillo darle la puntuación (9 de 10, 7 de 10, o lo que sea que sucede) Es un poco más confuso saber qué hacer con el puntaje. Incluso si ella está adivinando y no tiene ni idea, puede tener suerte y tener algunas correctas – incluso todas las 10. ¿Pero, qué tan verosímil es eso?

CONSEJO DE ENSEÑANZA
El puntaje es un ajuste para la idea de una prueba estadística, pero no es necesario introducir esa terminología en el día 1

Intentemos un experimento. Yo lanzo 10 monedas. Usted adivina cuales son escudos y cuales coronas, y vemos cómo lo hace.

CONSEJO DE ENSEÑANZA
Haga que cada estudiante haga su suposición escribiendo una secuencia de 10 escudos y 10 coronas mientras usted lanza las 10 monedas detrás de una barrera para que los estudiantes no puedan ver los resultados.

Comparando con los compañeros de clases, vamos a ver sin duda que algunos adivinaron mejor que otros.

Ahora suponga que la dama dice 9 de 10 correctamente Esto no es perfecto, pero es mejor de lo esperado para alguien que estaba adivinando. Por otro lado, no es imposible acertar 9 de 10 adivinando. Entonces aquí está la gran idea de Fisher: Averigüemos que tan difícil es acertar 9 de 10 veces adivinando. Si no es tan díficil de hacer, entonces puede ser que eso haya ocurrido, entonces no

deberíamos estar muy impresionados de la capacidad de degustar té de la mujer. Por otro lado, si es realmente inusual conseguir 9 de 10 aciertos adivinando, entonces tenemos evidencia que puede ser que nos diga algo.

¿Pero cómo nos damos cuenta de qué tan inusual es acertar 9 de 10 solamente adivinando? Ya aprenderemos otro método, pero por ahora sólo tiremos un montón de monedas y mantengamos un control de estas. Si ella está adivinando, podrá también estar tirando una moneda.

Entonces este es el plan. Vamos a lanzar 10 monedas. Vamos a llamar los escudos aciertos y las coronas fallos. Ahora vamos a tirar 10 monedas más, y 10 más, y 10 más y . . . Eso se volvería bastante tedioso. Afortunadamente las computadoras son buenas con las cosas tediosas, así que permitimos a la computadora tirarlas por nosotros.

La función `rflip()` puede tirar una moneda

```
require(mosaic)
rflip()

##
## Flipping 1 coin [ Prob(Heads) = 0.5 ] ...
##
## T
##
## Number of Heads: 0 [Proportion Heads: 0]
```

O un número de monedas

```
rflip(10)

##
## Flipping 10 coins [ Prob(Heads) = 0.5 ] ...
##
## H T H H T H H H T H
##
## Number of Heads: 7 [Proportion Heads: 0.7]
```

Escribir `rflip(10)` muchas veces es casi tan tedioso como tirar las monedas. Pero no es tan difícil decirle a R que haga `do()` una serie de veces.

NOTA

Hay un cambio sutil aquí. Antes le preguntábamos a los estudiantes cuantas H's y cuantas T's acertaron de la moneda tirada. Ahora estamos usando H para simular los aciertos al adivinar y las T los desaciertos al adivinar. Esto hace simular más fácil.

NOTA

Note que `do()` es inteligente sabiendo que información guardar. En lugar de guardar todo lo que individualmente da, sólo guarda el número de tiradas, el número de escudos y de coronas

```
do(3) * rflip(10)

##    n heads tails prop
## 1 10     8     2  0.8
## 2 10     4     6  0.4
## 3 10     1     9  0.1
```

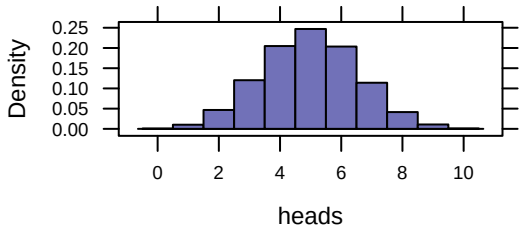
Ahora hagamos que R haga `do()` para nosotros 100000 veces y hagamos una tabla de resultados

```
# Guarde los resultados de 10000 damas simuladas
random.ladies <- do(10000) * rflip(10)
```

```
tally(~heads, data=random.ladies)

## heads
##    0    1    2    3    4    5    6    7    8    9   10
##    5  102  467 1203 2048 2470 2035 1140  415  108    7

# También podemos desplegar una
#tabla usando porcentajes
tally(~heads, data=random.ladies, format="prop")

## heads
##      0      1      2      3      4      5      6      7
## 0.0005 0.0102 0.0467 0.1203 0.2048 0.2470 0.2035 0.1140
##      8      9     10
## 0.0415 0.0108 0.0007
```

Podemos desplegar esta tabla gráficamente con un gráfico llamado histograma y con clases de ancho 1.

```
histogram(~ heads, data=random.ladies, width=1)
```



Usted puede estar sorprendido de ver que el número correcto de aciertos igual a 5 (la mitad de los 10 intentos) sucede solo 25 % de las veces. Pero la mayoría de los resultados están cerca de 5 correctas. Por ejemplo 67 % de los resultados son 4, 5 o 6. Aproximadamente 90 % de los

Consejo de Enseñanza

Siempre está la pregunta de cuantas simulaciones hacer. Este es un punto medio entre la velocidad y la precisión. Para cosas simples, uno puede fácilmente ejecutar 100000 o más simulaciones en clase. Para cosas más complicadas (que pueden requerir ajustar un modelo y extraer información de cada iteración) puede ser que prefiera un número menor de iteraciones para demostraciones en clase

Cuando se cubra inferencia para una proporción, es buena idea usar estos métodos para revisar la pregunta de cuantas replicaciones son necesarias en ese contexto.

Nota

El paquete `mosaic` le agrega algunas características adicionales a la función `histogram()`. En particular, los argumentos de ancho del intervalo y del centro, que hace más fácil el control de las clases; esto sólo está disponible si está usando el paquete `mosaic`.

resultados están entre 3 y 7 (incluyéndolo). Pero tener 8 correctas es inusual, y 9 o 10 todavía más inusual.

¿Entonces, qué concluimos? Es posible que la dama tuviera 9 de 10 correctas adivinando, pero no es verosímil (sólo paso 1.2 % de nuestras simulaciones). Entonces una de dos cosas es verdad:

- La dama es inusualmente suertuda, o
- La dama no está adivinando.

¡Aunque Fisher no dice como resultó el experimento, otros dicen que la dama identificó correctamente las 10 tasas! [2]

[2] D. Salsburg. *The Lady Tasting Tea: How statistics revolutionized science in the twentieth century*. W.H. Freeman, New York, 2001

Un diseño diferente

Suponga que en lugar de lo anterior, vamos a preparar cinco copas de cada forma (y que la dama que las probará supiera esto) le damos cinco cartas que dicen "leche primeroz ella debe ponerlas junto a las tasas en las que la leche se sirvió primero. ¿Cómo este diseño cambia las cosas?

Podemos simular barajas de 10 cartas y sacar cinco de ellas.

Nota

El uso de `factor()` aquí permite a R saber que los posibles valores son 'M'y 'T', incluso cuando uno o el otro aparezca en una muestra aleatoria.

```
cards <-
  factor(c("M","M","M","M","M","T","T","T","T","T"))
tally(~deal(cards, 5))

## deal(cards, 5)
## M T
## 3 2
```

```
results <- do(10000) * tally(~deal(cards, 5))
tally(~ M, data=results)

## M
##    0    1    2    3    4    5
##   44  993 3966 3927 1028   42

tally(~ M, data=results, format="prop")

## M
##      0      1      2      3      4      5
## 0.0044 0.0993 0.3966 0.3927 0.1028 0.0042
```

```
tally(~ M, data=results, format="perc")
```

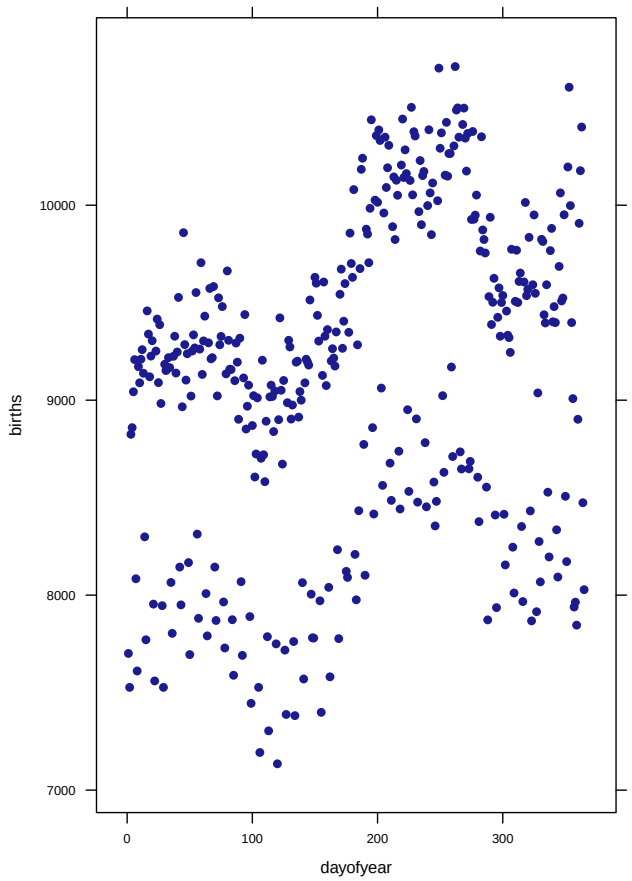
```
## M
##     0     1     2     3     4     5
##  0.44  9.93 39.66 39.27 10.28  0.42
```

4.2 *Nacimientos por día*

El conjunto de datos `Births78` contiene el número de nacimientos en Estados Unidos para cada día de 1978. Un gráfico de dispersión de los nacimientos por día del año revela algunos patrones interesantes. Veamos como el número de nacimientos depende del día del año.

NOTA
El uso de la frase "depende de.[es] intencional. Más adelante vamos a enfatizar en como `y ~ x` puede ser interpretado como "y depende de x"

```
xyplot(births ~ dayofyear, data=Births78)
```



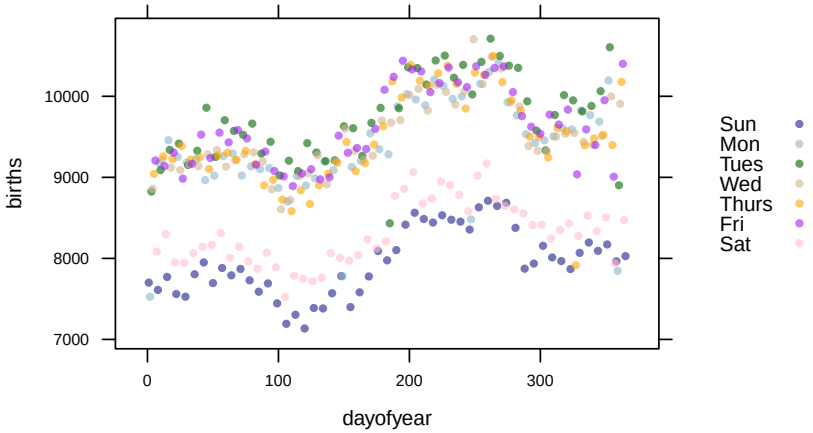
Cuando se les muestra esta imagen, los estudiantes deberían estar listos para describir los dos patrones en los datos; deberían notar que ambas bajadas y subidas en el curso del año y las dos .ºndas paralelas". Hacer que los estudiantes conjeturen acerca de estos rápidamente revela si están interpretando correctamente un gráfico. Many students will be able to come up with conjectures about the peaks and valleys, but they often struggle to correctly interpret the parallel waves. Having them make conjectures about this will quickly reveal whether they are correctly interpreting the plot.

Una conjetura acerca del paralelismo de las ondas puede ser revisada usando los datos disponibles. Si desplegamos cada día de la semana con diferente símbolo o color, podemos ver que hay menor cantidad de nacimientos los fines de semana. Hay algunas excepciones que se pueden ver, los días libres o feriados.

CONSEJO DE ENSEÑANZA
El gráfico puede también ser construido usando la variable `date`. Para propósitos generales, este puede ser un mejor gráfico a construir, pero usar `dayofyear` fuerza a los estudiantes a pensar más en lo que el eje x significa

CONSEJO DE ENSEÑANZA
Esto puede hacer una buena actividad de pensar y compartir. Haga que los estudiantes hagan explicaciones posibles, y que después discuta estas explicaciones con un compañero. Finalmente, haga a algunas parejas compartir sus explicaciones con toda la clase

```
require(mosaicData)       # Carga los conjuntos de datos
xyplot(births ~ dayofyear, data=Births78,
       groups=wday,
       auto.key=list(space="right"))
```



CONSEJO DE ENSEÑANZA
Esta cantidad de excepciones es fácil de ver si çonectamos los puntos", vea sección 4.6.1

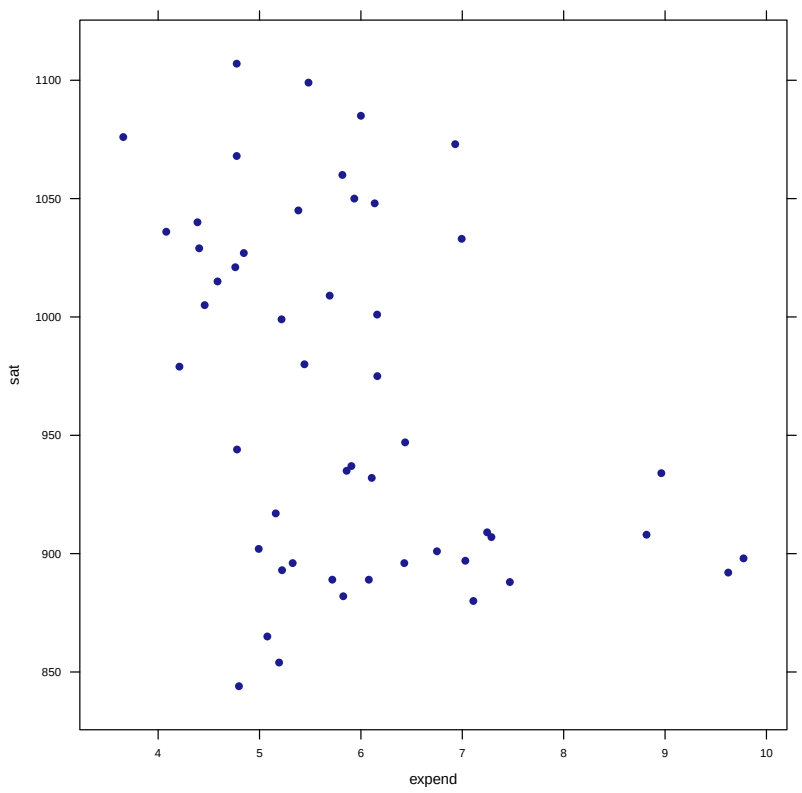
Una discusión de este u otro conjunto de datos que pueden ser explorados mediante exposiciones gráficas estadísticas es una buena forma de demostrar çuriosidad estadística", para ilustrar el poder de R para crear gráficos, y para introducir la importancia de las co-variables en el análisis estadístico.

A la visualización la han llamado la "droga de entrada.ª la estadística. Puede ser una gran forma de ingresar a los estudiantes en la estadística – y lejos de sus calculadoras con graficadores

4.3 *SAT y cofundación*

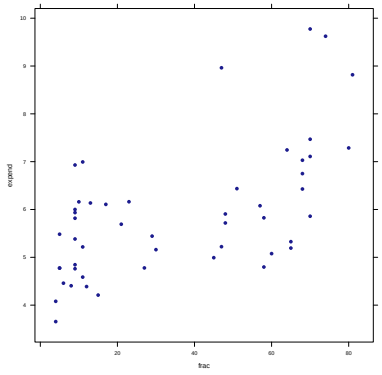
El conjunto de datos `SAT` contiene información acerca de la relación entre los puntajes obtenidos en el SAT y las medidas de gasto educativo. Los estudiantes frecuentemente se sorprenden de ver que estados que gastan más en educación tienen peores resultados en los SAT.

```
xyplot(sat ~ expend, data=SAT)
```

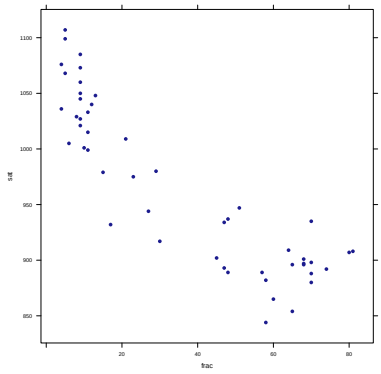


La implicación, que gastar menos puede dar mejor resultado es injustificado. Los gastos son confundidos con la proporción de estudiantes que toman el examen, y las notas son mayores en estados donde una menor cantidad de estudiantes hacen el examen.

```
xyplot(expend ~ frac, data=SAT)
```



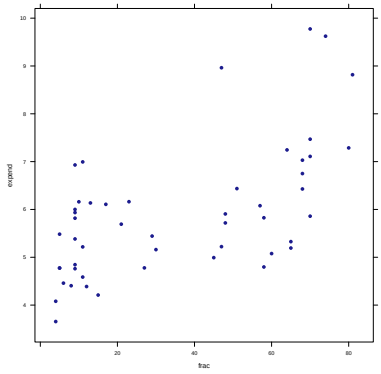
```
xyplot(sat ~ frac, data=SAT)
```



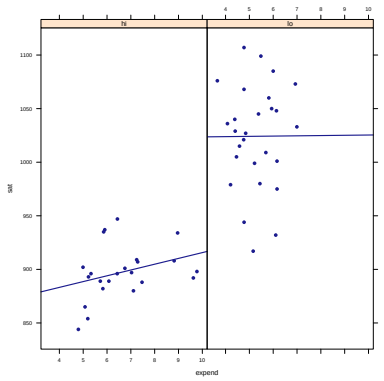
Es interesante ver el gráfico original si ubicamos a los estados en dos grupos dependiendo si tienen más o menos 40

```
SAT <- mutate(SAT,
        fracGroup = derivedFactor(
          hi = (frac > 40),
          lo = (frac <=40) ))
```

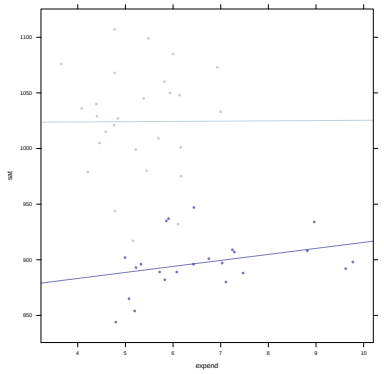
```
xyplot(expend ~ frac, data=SAT)
```




```
xyplot( sat ~ expend | fracGroup , data=SAT,
        type=c("p","r") )
```




```
xyplot( sat ~ expend, groups = fracGroup , data=SAT,
        type=c("p","r") )
```



Este ejemplo puede usarse para advertirnos contra interpretar relaciones causales e ilustrar la importancia de considerar co-variables.

4.4 *Enfermedad de Wilts y los ácaros*

Este ejemplo muestra cómo construir inferencia estadística desde principios primarios

Los investigadores sospechan que el ataque de una planta por otro organismo induce la resistencia al ataque subsecuente por un organismo diferente. Unas plantas de algodón en macetas individuales fueron aleatoriamente distribuidas en dos grupos: invadidas por arañas enanas y sin invasión. Después de dos semanas, las arañas fueron removidas responsablemente por un asistente de la

investigación, y ambos grupos fueron tratados con Verticillium, un hongo que causa la enfermedad de Wilt. Los investigadores estaban esperando que los datos dieran luz para responder la siguiente pregunta:

¿Hay una relación entre la plaga y la enfermedad de Wilt?

The accompanying table shows a cross tabulation the number of plants that developed symptoms of Wilt disease.

```
tally(outcome ~ treatment, data = Mites, margins = TRUE)
```

```
##          treatment
## outcome   mites no mites
##   no wilt    15        4
##   wilt       11       17
##   Total      26       21
```

Algunas preguntas para los estudiantes:

1. ¿Cuál piensa que es la variable predictora? ¿La variable respuesta?
2. ¿Qué proporción de las plantas del estudio que tuvo el problema de arañas desarrolló enfermedad de Wilt?
3. ¿Qué proporción de plantas en el estudio que tuvo el problema de arañas desarrolló enfermedad de Wilt?
4. El riesgo relativo es el radio de los proporciones de riesgo. ¿Cuál es el riesgo relativo de desarrollar una enfermedad de Wilt, comparando aquellos que tuvieron arañas y los que no?
5. ¿Si no hubiera asociación entre las arañas y la enfermedad de Wilt, cuál sería el riesgo relativo (en la población)? ¿Qué tan cerca está este riesgo relativo calculado de los datos de este valor?
6. Sea X el número de plantas en el grupo que no tuvo arañas que no desarrolló enfermedad de Wilt. ¿Cuáles son los posibles valores de X?
7. Asumiendo una población con riesgo relativo 1, dele dos posibles valores a X que puedan ser más inusuales que los valores para estos datos

Las preguntas 6-7 pueden ser vistas usando cartas
Los estudiantes pueden juntar sus resultados al ponerlos en la pizarra al frente del salón. Después los estudiantes pueden procesar los resultados contestando las siguientes preguntas.

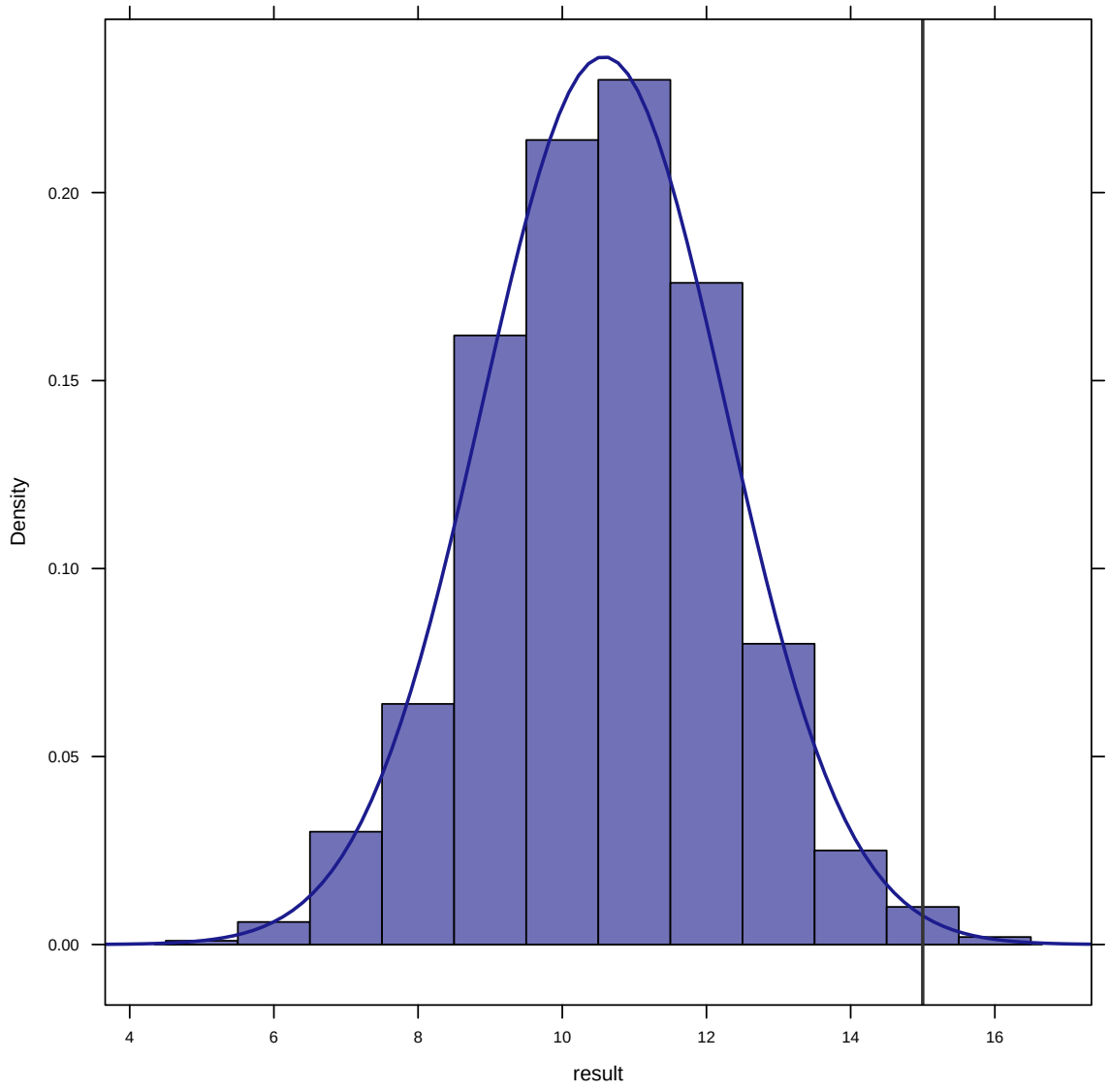
8. ¿Cuantas cartas negras esperamos (en promedio)?
9. ¿Qué observamos?

10. ¿Cómo resumimos estos resultados? ¿Cuál es la gran idea?

Una vez que la simulación con cartas ha sido completada, podemos usar R para hacer muchísimas más simulaciones rápido

Computational Simulation

```
tally(outcome ~ treatment, data=Mites)

##          treatment
## outcome   mites no mites
##   no wilt    15        4
##   wilt       11       17

X <- tally(outcome ~ treatment, data=Mites)[1,1]; X

## [1] 15

nullDist <- do(1000) *
  tally(outcome ~ shuffle(treatment), data=Mites)[1,1]
histogram(~ result, data=nullDist, width=1,
          type="density", fit="normal", v=15)
```



5
Menos vólumen, más creatividad

> Muchas veces uno termina poniendo muchísimo más volumen, porque está enseñando fundamentos y porque está enseñando conceptos que necesita agregar, pero usted tal vez no los use necesariamente porque son bloques de construcción para otros conceptos y variaciones que van a aparecer de esto... Cuando se está fuera de temporada, se tiene la oportunidad de tomar un paso atrás y adaptarlo más específicamente a su equipo y a sus jugadores.
>
> – Mike McCarthy, Entrenador principal, Green Bay Packers

> La perfección se logra, no cuando no hay nada más que agregar, pero cuando no hay nada más que quitar
>
> – Antoine de Saint-Exupery, escritor, poeta, aviador

Una forma clave de introducir R exitosamente es encontrar un conjunto de comandos que sea

- Pequeño,
- coherente, y
- poderoso.

Este capítulo proporciona un ejemplo extenso de este abordaje de "Menos vólumen, más creatividad". El paquete `mosaic` (combinado con el paquete `lattice` y otras funciones del corazón de R) nos provee una poderosa y simple estructura que equipa a los estudiantes para producir todos los

- Resúmenes numéricos,

Mike McCarthy, entrenador del equipo de fútbol americano Green Bay Packers usa "Menos volumen, más creatividadçomo un mantra con su staff de entrenamiento conforme preparan los planes de juego cada semana. Como ilustración del principio de trabajo, cuando un fanático del equipo le pregunta cuantos pases prepara el equipo para un oponente dado, el entrenador contestó, Çuando entré a la NFL, hacíamos 150 pases en nuestro plan de juego. He puesto un cartel en todas las puertas de coordinadores del equipo - Menos volumen, más creatividad. Funcionamos con más conceptos, con menos volumen. [Ahora] Hacemos alrededor de 50[pases] en un plan de juego"

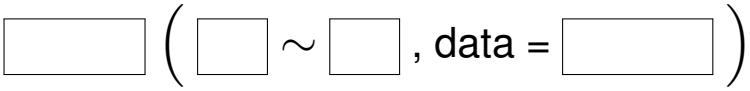
- Resúmenes gráficos, y
- Modelos lineales

necesarios en cursos introductorios. Presentando esto como una plantilla maestra con variaciones, le damos énfasis a la similitud entre estos comandos y así reducir la carga cognitiva para los estudiantes. En nuestra experiencia, esto ha hecho R mucho más accesible y disfrutable para los estudiantes e instructores.

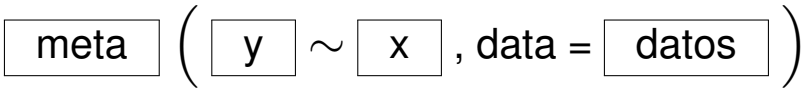
5.1 *El paquete `mosaic` y la notación de fórmula*

Mucho del primer trabajo del paquete `mosaic` está centrado en producir un conjunto de comandos de Rmínimo que pudiera brindar a los estudiantes todo lo que necesitan para cursos introductorios de estadística; no sobrecargarlos con demasiados comandos. Una de las viñetas del paquete `mosaic` incluye un documento describiendo tal conjunto de comandos.

Muchos de estos están construidos a partir de la siguiente notación, que es utilizada frecuentemente.



Esta plantilla se puede utilizar llenando las cajas. Sirve darle a cada caja un nombre::



La notación tiene un poco más de flexibilidad que la que se ha indicado. A veces y no es necesario

```
goal( ~ x, data=mydata )
```

The formula may also include a third part

```
goal( y ~ x | z , data=mydata )
```

La fórmula puede incluir una tercera parte

Consejo de Enseñanza
Después de introducir esta plantilla, tal vez debería evaluar a los estudiantes para saber si la han aprendido. Esto enfatiza su importancia.

```
goal( formula , data=mydata )
```

Esta notación puede ser aplicada para crear resúmenes numéricos, resúmenes gráficos o ajustes de modelos haciéndose dos preguntas y usando las preguntas para llenar los espacios de la notación (plantilla).

1. ¿Qué quiere que haga R?

   Esta, es la meta

2. ¿Qué debe saber R para saber eso?

   Estas son las entradas que de la función. Para resúmenes numéricos, resúmenes gráficos y ajustes de modelos, típicamente necesitamos especificar las variables envueltas y el conjunto de datos en el que están guardadas.

5.2 *Resúmenes gráficos de datos*

Los resúmenes gráficos son una importante y vistosa forma de demostrar el poder y flexibilidad de nuestra notación. Nos gusta introducir a los estudiantes a resúmenes gráficos prontamente en el curso. Esto les da a los estudiantes acceso a una funcionalidad de R que destaca (y es muchísimo mejor que una calculadora a la mano). También empieza a desarrollar su habilidad de interpretar representaciones gráficas de datos, pensar en distribuciones y plantear preguntas estadísticas.

Hay bastantes formas de hacer gráficos en R. Un abordaje es el sistema de los gráficos llamados `lattice`. Siempre que se carga el paquete `mosaic`, el paquete `lattice` es cargado también. Uno de sus aspectos atractivos es que usa la misma notación (plantilla) que usamos para resúmenes y modelos lineales.

5.2.1 *Resúmenes gráficos de dos variables*

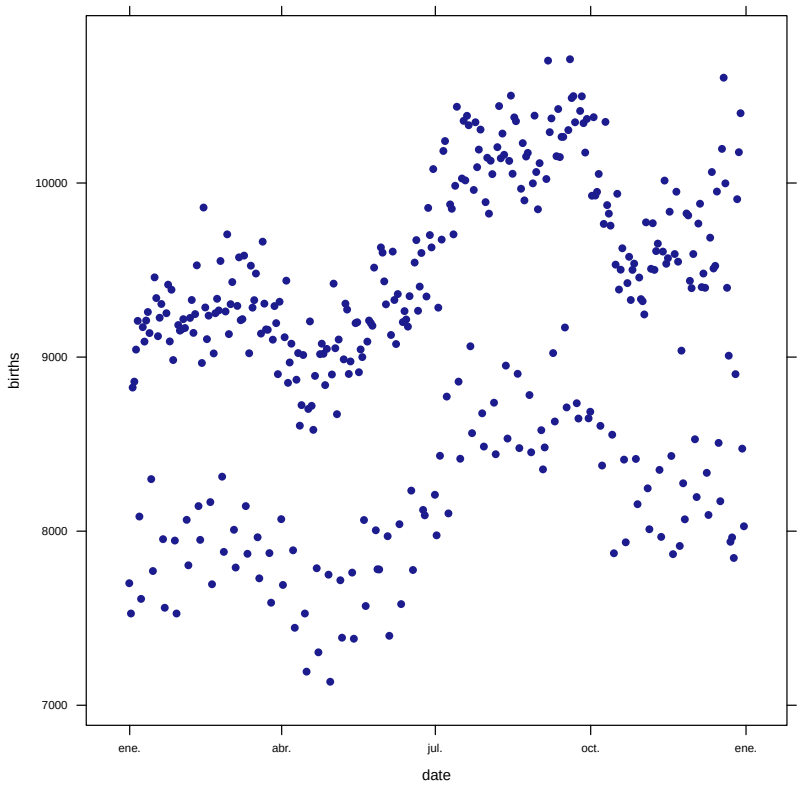
Un primer ejemplo: haciendo un gráfico de dispersión

Más Info

Seguido nos preguntan sobre otros sistemas gráficos, especialmente los gráficos `ggplot2`. En nuestra experiencia, `lattice` hace más fácil para principiantes crear una amplia variedad de gráficos más o menos estandarizados – incluyendo la habilidad de representar varias variables al mismo tiempo. `ggplot2`, por otra parte, hace más fácil generar gráficos personalizados o combinar componentes de gráficos. Cada una tiene su lugar, y nosotros utilizamos ambos sistemas. Pero para principiantes, típicamente enfatizamos `lattice`.

El nuevo paquete `ggvis`, por los autores de `ggplot2` agrega interactividad y velocidad a los gráficos para fortalecer `ggplot2`.

Como ejemplo. Creemos el siguiente gráfico, que muestra el número de nacimientos en Estados Unidos para cada día en 1978.



Consejo de Enseñanza
Este gráfico puede generar una discusión interesante al inicio del curso. Pídale a los estudiantes hacer conjeturas sobre patrones en los gráficos. Sus respuestas revelarán sin están interpretando bien los gráficos.

1. ¿Cuál es la meta?

   Queremos un gráfico de dispersión. La función que crea gráficos de dispersión se llama xyplot(), entonces esta meta llena una parte de nuestra plantilla.

2. ¿Qué necesita saber R ?

   R necesita saber cuál variable va donde y donde encontrar las variables. En este caso, los datos están guardados en el conjunto de datos `Births78`.

```
head(Births78)

##         date births dayofyear  wday
## 1 1978-01-01   7701         1   Sun
## 2 1978-01-02   7527         2   Mon
## 3 1978-01-03   8825         3  Tues
```

```
## 4 1978-01-04   8859         4   Wed
## 5 1978-01-05   9043         5 Thurs
## 6 1978-01-06   9208         6   Fri
```

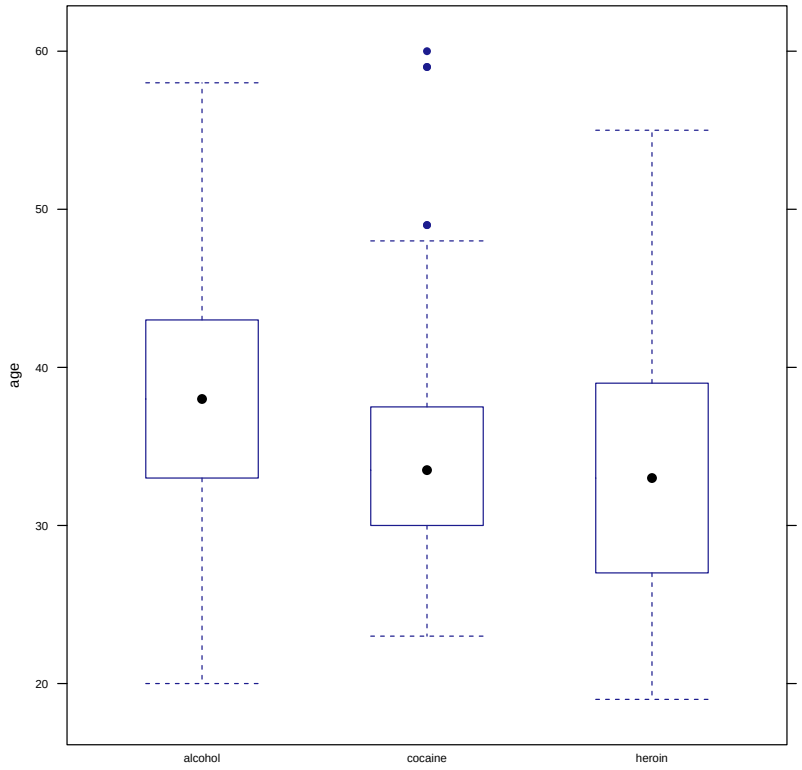
Queremos poner el número de nacimientos (`births`) en el eje y, y el día del año (`date`) en el eje x.

Poniendo todo junto, genera el siguiente comando

```
xyplot(births ~ date, data=Births78)
```

Otro ejemplo: Diagramas de cajas

Ahora generemos este gráfico, que muestra diagramas de cajas para cada una de las tres sustancias de abuso utilizadas por participantes en los ensayos clínicos aleatorizados de *Health Evaluation and Linkage to Primary Care*



Los datos que necesitamos están en el conjunto de datos `HELPrct`, del cual queremos desplegar las variables año y sustancias en los ejes `y` y `x`. De acuerdo a nuestra notación, el comando para crear este gráfico tiene la forma

```
meta(age ~ substance, data=HELPrct)
```

La única información adicional que necesitamos es el nombre de la función que crea los diagramas de caja. Esta función es `bwplot()`. Entonces podemos crear el gráfico con

Más Info

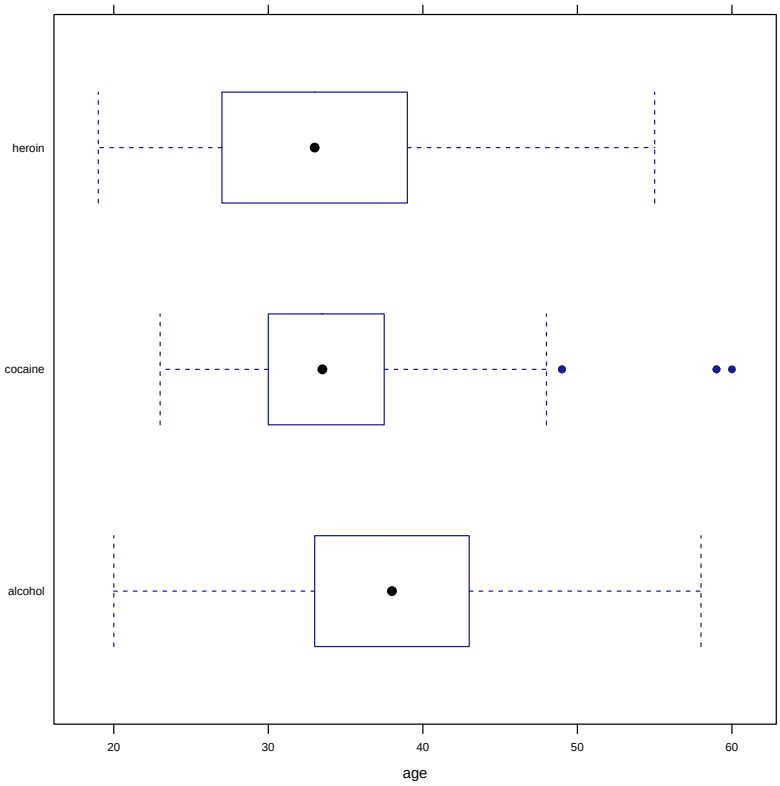
Puede encontrar más sobre información del conjunto de datos `HELPrct` usando el comando de ayuda: `?HELPrct`. Este le brindará ayuda con el libro de código para los datos y enlaces de la fuente original.

También hay unas cuantas funciones que permiten inspeccionar contenidos de un conjunto de datos. Entre nuestros favoritos están `inspect()`, `glimpse()`, y `head()`.

```
bwplot(age ~ substance, data=HELPrct)
```

Para hacer los gráficos de caja horizontales en lugar de verticales, invierta la posición de `age` y `substance`:

```
bwplot(substance ~ age, data=HELPrct)
```



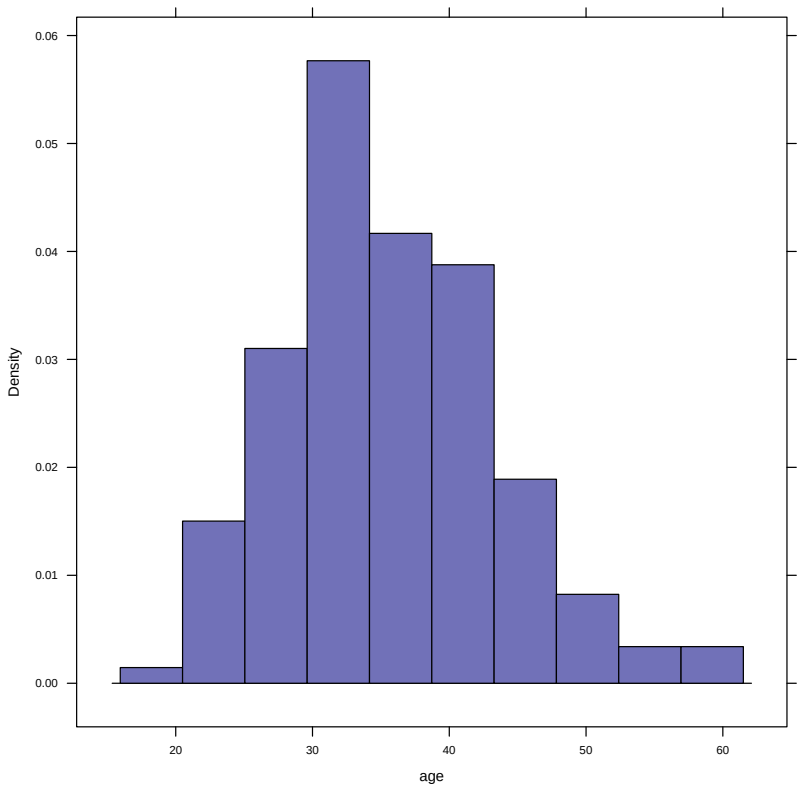
5.2.2 *Resúmenes gráficos de una variable*

Si queremos hacer un gráfico que incluya solamente una variable, simplemente omitimos la parte de `y` de la fórmula. Por ejemplo, un histograma como

MÁS INFO

Podrá estar preguntándose sobre gráficos para dos variables categóricas. Un gráfico comúnmente usado para esto es el gráfico de barras segmentadas. Vamos a tratar esto como una versión aumentada de un gráfico de barras simple, que es un resumen gráfico de una variable categórica.

Otro gráfico que puede usar para desplegar dos (o más) variables categóricas es un gráfico `mosaic()`. El paquete `lattice` no incluye los gráficos mosaic, sin embargo, el paquete `vcd` proporciona una función `mosaic()` que crea gráficos mosaic.



puede ser hecho con

```
histogram( ~ age, data=HELPrct)
```

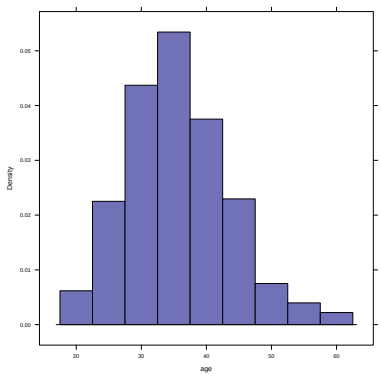
El paquete `mosaic` agrega alguna otra funcionalidad a `histogram()`, para hacer más fácil especificar las clases pendientes. En particular, las opciones del ancho y del centro(el predeterminado es 0) puede ser usado para definir el ancho de las clases o el centro de una de las clases. Por ejemplo, para crear un histograma con clases que sean de un ancho de 5 años, se puede usar `width=5`, y podemos cambiarlas de izquierda a derecha cambiando el valor del centro.

```
histogram( ~ age, data=HELPrct, width=5)
```

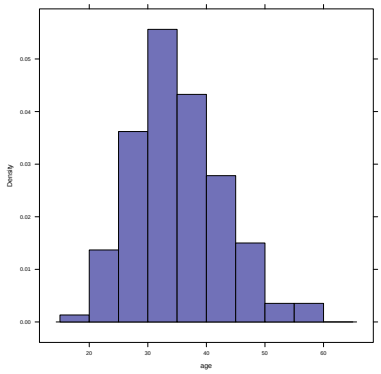
¡Precaución!
Es importante notar que cuando hay solamente una variable, esta tiene que estar *a la derecha de la fórmula*.

Consejo de Enseñanza
Dígale a los estudiantes que dado que R está computando los valores de `Y`, nosotros no necesitamos proporcionárselos. Esta no es exactamente la razón por la cual esto es de esta forma, pero les va ayudar a recordar.

Introducir `width` y `center` aquí es, en efecto, una violación a nuestra política usual de aceptar predeterminados y guardando opciones para más tarde. Pero es importante que las cajas del histograma sean elegidas apropiadamente, y el predeterminado algoritmico no funciona bien para todos los conjuntos de datos. Motivamos a los estudiantes a hacer algunos histogramas y experimentar con el centro y especialmente con el ancho.




```
histogram( ~ age, data=HELPrct, width=5,
           center=2.5)
```

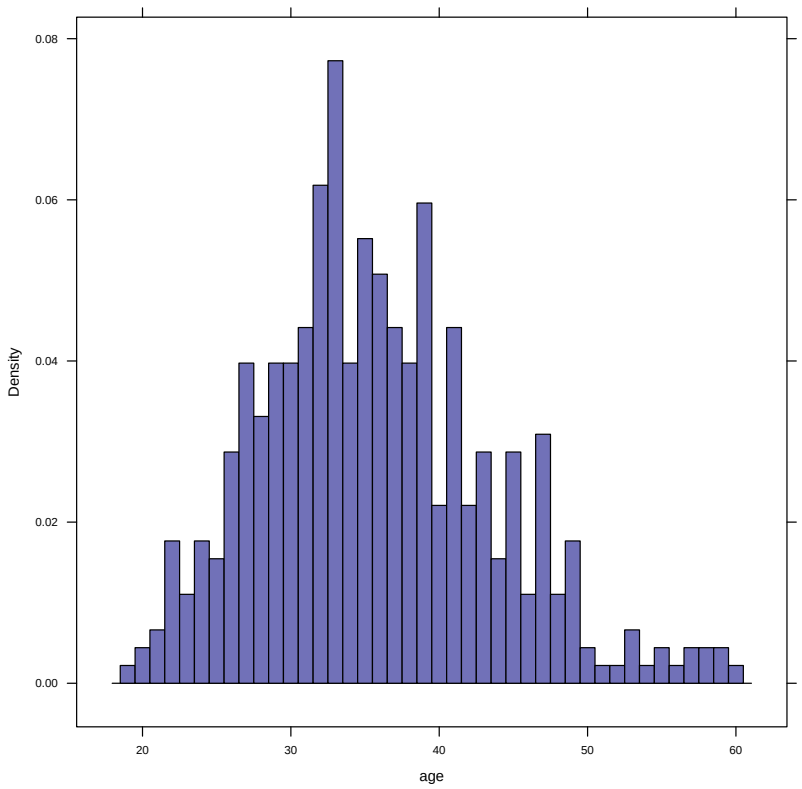


Hay datos suficientes para utilizar una barra para cada número entero si queremos. Como el valor predeterminado de centro es 0, ajustar el ancho a 1 centra las clases en los enteros, evitando confusiones potenciales de cual límite es incluido en la barra.

```
histogram( ~ age, data=HELPrct, width=1)
```

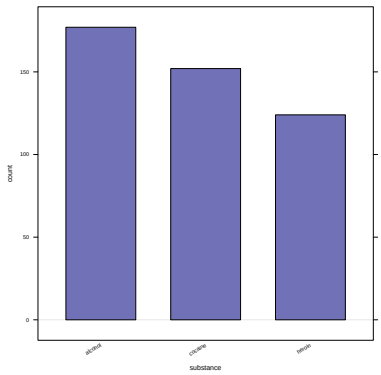
NOTA

El centro no necesita ser contenido en las clases que son desplegadas. Entonces consiga clases con límites .en 0's y 5's", podemos hacer el centro 2.5, aunque no esté en el rango de datos.



Para una única variable categórica, podemos hacer un gráfico de barras, usando `bargraph()` en lugar de `histogram()`. Puesto que las fórmulas son requeridas al lado derecho, los gráficos de barra horizontales son producidos utilizando `horizontal = TRUE`

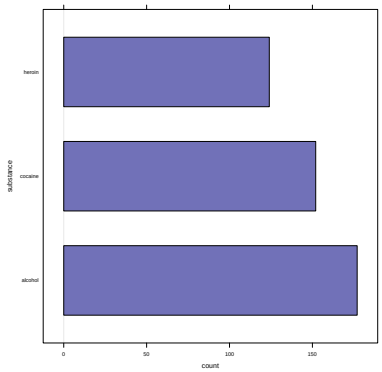
```
bargraph( ~ substance, data=HELPrct)
```



Más Info

La función `bargraph()` no está en el paquete `lattice`, pero está en el paquete `mosaic`. La funcion de `lattice` `barchat()` crea gráficos de barra para datos *resumidos*; `bargraph()` ya se encarga de crear este resumen de datos y después usar `barchat()` para crear el gráfico.

```
bargraph( ~ substance, data=HELPrct, horizontal=TRUE)
```

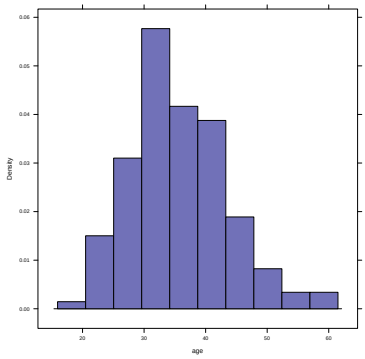


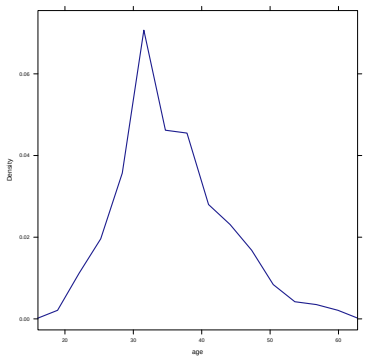
5.2.3 *La gama de gráficos*

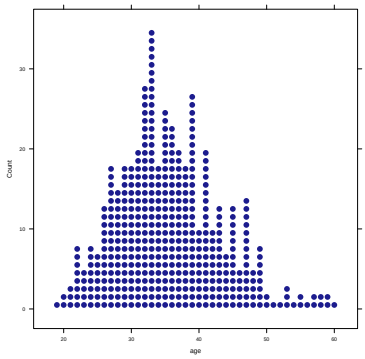
El poder de la notación (plantilla) es que ahora podemos hacer diferentes tipos de gráficos imitando los ejemplos anteriores pero cambiando la meta.

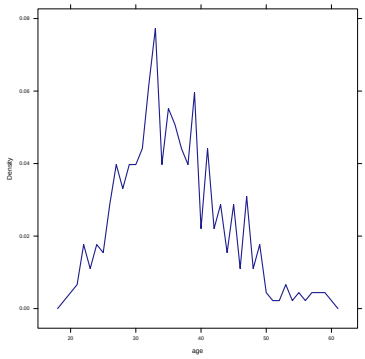
Más Info
Si usted no está familiarizado con algunos gráficos, como los ashplots y los polígonos de frecuencia, siga leyendo. Tenemos más que decir de ellos pronto.

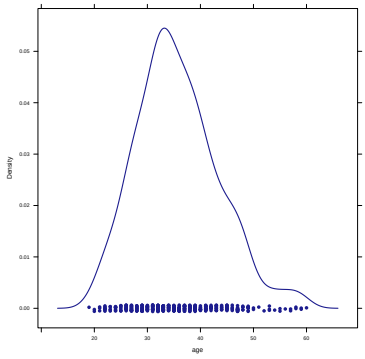
```
histogram(~age, data = HELPrct)
freqpolygon(~age, data = HELPrct)
dotPlot(~age, data = HELPrct, width = 1)
ashplot(~age, data = HELPrct, width = 1)
densityplot(~age, data = HELPrct)
qqmath(~age, data = HELPrct)
bwplot(~age, data = HELPrct)
bwplot(~age, data = HELPrct, pch = "|")
```

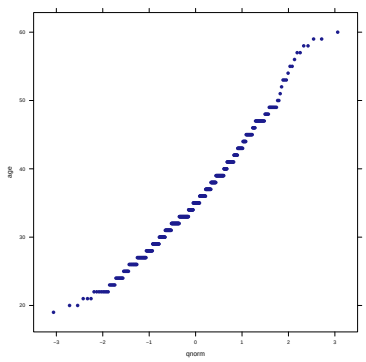


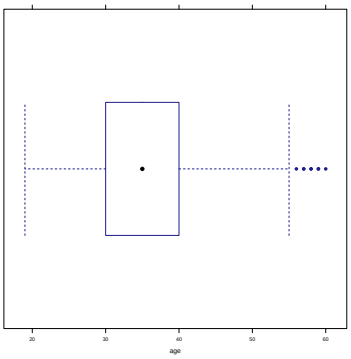

Density
0.00
0.02
0.04
0.06
20
30
40
50
60
age

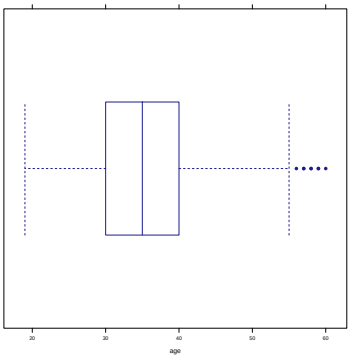

Count
0
10
20
30
20
30
40
50
60
age


Density
0.00
0.02
0.04
0.06
0.08
20
30
40
50
60
age


Density
0.00
0.01
0.02
0.03
0.04
0.05
20
30
40
50
60
age





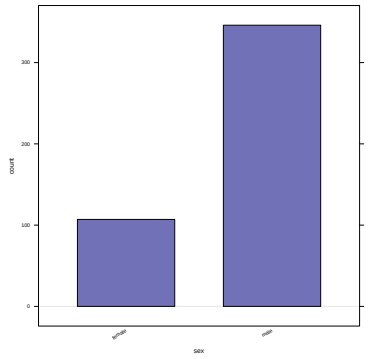


Algunas personas prefieren el diagrama de cajas desplegado de una forma más tradicional, con una línea en la mediana en lugar de un punto. Podemos hacer esto un comportamiento predeterminado usando.

```
trellis.par.set(box.dot = list(pch = "|"))
```

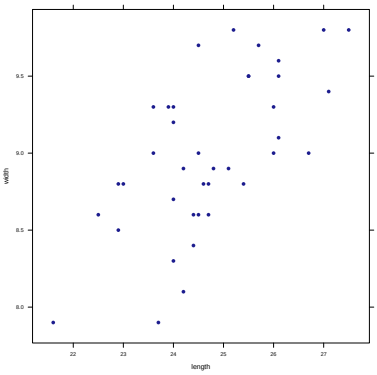
Para una variable categorica, podemos usar un gráfico de barras

```
bargraph( ~ sex, data=HELPrct)
```



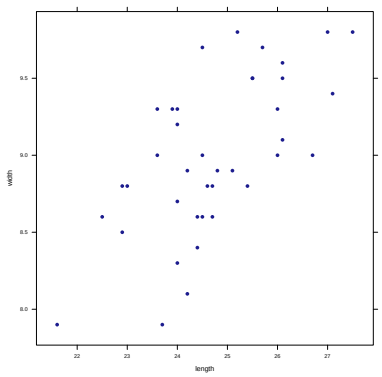

```
 # Variable categórica
```

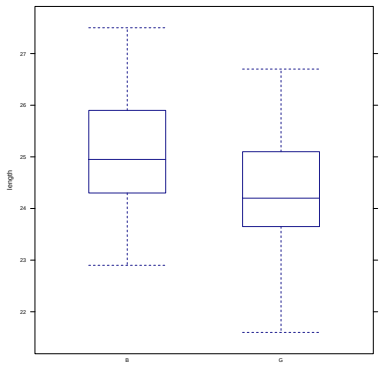
```
    xyplot( width ~ length, data=KidsFeet)  # 2 variables cuantitativas
plotPoints( width ~ length, data=KidsFeet)  # Una alternativa a mosaic
    bwplot(length ~ sex,    data=KidsFeet)  # 1 categórica, 1 cuantitativa
    bwplot(   sex ~ length, data=KidsFeet)  # roles inversos
```

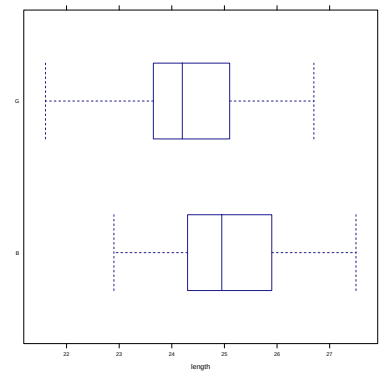


NOTA

El paquete `lattice` no suministra una función para crear gráficos de tipo pastel. Esto no es una gran pérdida puesto que es generalmente más difícil hacer comparaciones con estos gráficos.



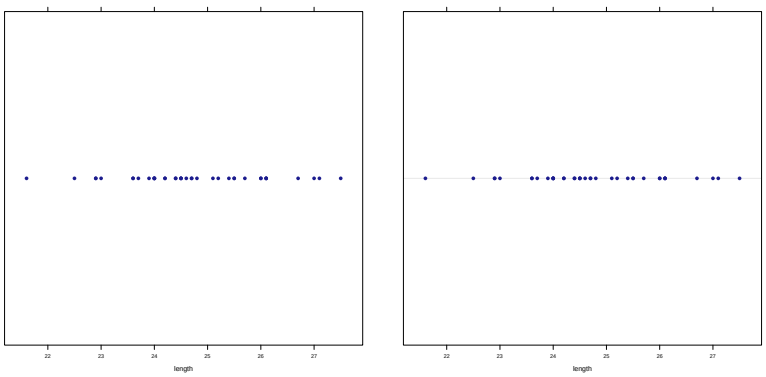




El paquete `lattice` también ofrece las funciones `stripplot()` y `dotplot()` pueden ser usadas para un gráfico de dispersión unidimensional. Esto trabaja razonablemente bien para conjuntos de datos pequeños, pero tiene una utilidad limitada para conjuntos de datos grandes.

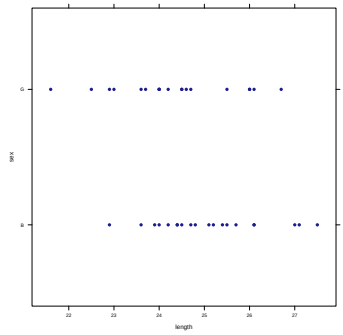
¡Precaución! Hay una función llamada `dotPlot()` (con P mayúscula). Note que `dotplot()` produce un resultado muy diferente al gráfico que produce `dotPlot()`

```
stripplot( ~ length, data=KidsFeet)
  dotplot( ~ length, data=KidsFeet)
```

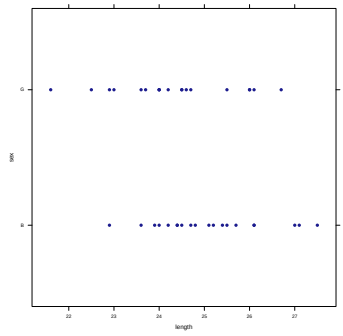


Estos `xyplot()` o `plotPoints()` pueden también ser usados con una variable cuantitativa y una variable categorica

```
xyplot(sex ~ length, data=KidsFeet)
```

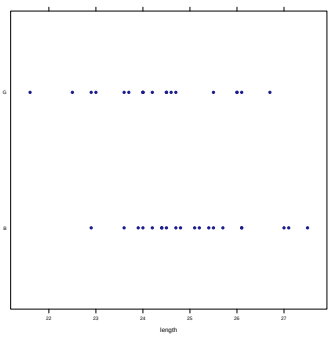



```
plotPoints(sex ~ length, data=KidsFeet)
```

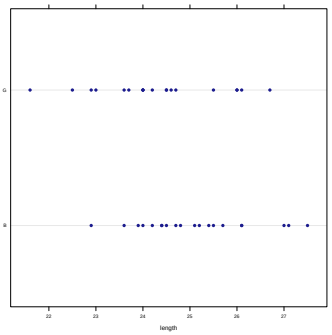



```
stripplot(sex ~ length, data=KidsFeet)
```

CONSEJO DE ENSEÑANZA
Usualmente no introducimos la función `dotplot()` y la función `stripplot()` a los estudiantes, simplemente usamos la función `xyplot.()`



```
dotplot(sex ~ length, data=KidsFeet)
```



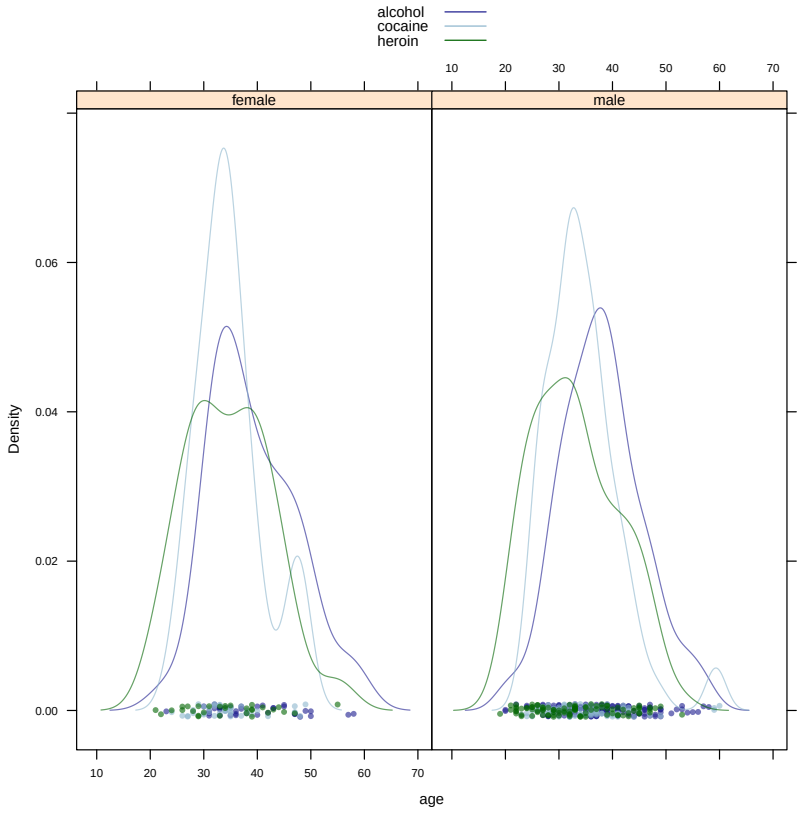
5.2.4 *Grupos y sub-gráficos*

Podemos agregar variables adicionales a nuestros gráficos tanto trasponiendo múltiples gráficos como ubicando múltiples gráficos uno al lado del otro en la cuadrícula. Para trasponer gráficos, agregamos un argumentos extra a nuestra plantilla, usando `groups =` ; y para crear sub-gráficos (llamados paneles en `lattice` y facetas en `ggplot2`) usando la fórmula

```
y ~ x | z
```

Por ejemplo, podemos sobreponer un gráfico de densidad para la edad para cada grupo de sustancia en paneles separados para cada sexo:

```
densityplot( ~ age | sex, data=HELPrct,
              groups=substance,
              auto.key=TRUE)
```



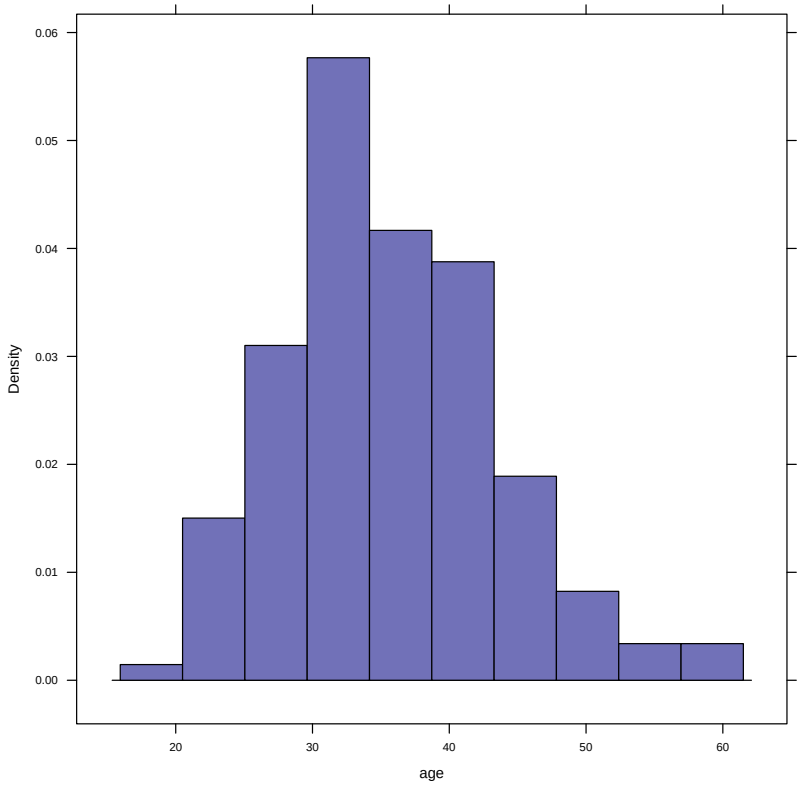
`auto.key=TRUE` agrega una leyenda simple, la cual podemos ver sobre las curvas, diciéndonos cual es cual.

5.3 *Resúmenes numéricos*

Los resúmenes numéricos pueden ser creados de la misma forma, simplemente tenemos que reemplazar el nombre del gráfico con el nombre de algún resumen numérico que deseamos. Nada más cambia; una media y un histograma resume una única variable, entonces cambiando `histogram()` por `mean()` nos da el resumen que queremos.

Nota
Lo importante que se debe resaltar en esta sección es lo poco que hay que aprender luego de que se aprende a hacer gráficos. Simplemente se cambia el nombre del gráfico al de una estadística de resumen y está listo.

```
histogram( ~ age, data=HELPrct)
```




```
     mean( ~ age, data=HELPrct)

## [1] 35.7
```

Más Info
Para ver la lista completa de estos resúmenes numéricos, use `help(favstats)`.

El paquete `mosaic` incluye versiones de varios resúmenes numéricos con notación de fórmula, incluyendo

El paquete `mosaic` incluye versiones de varios resúmenes numéricos con notación de fórmula, incluyendo `mean()`, `sd()`, `var()`, `min()`, `max()`, `sum()`, `IQR()`. Además de `favstats()`, una función que computa varias de nuestras estadísticas favoritas:

```
favstats( ~ age, data=HELPrct)

##  min Q1 median Q3 max mean   sd   n missing
##   19 30     35 40  60 35.7 7.71 453       0
```

La función `tally()` puede ser usada para contar casos

```
tally( ~ sex, data=HELPrct)

## sex
## female   male
##    107    346

tally( ~ substance, data=HELPrct)

## substance
## alcohol cocaine  heroin
##     177     152     124
```

En ocasiones es conveniente desplegar proporciones o porcentajes.

```
tally( ~ substance, data=HELPrct, format="percent")

## substance
## alcohol cocaine  heroin
##    39.1    33.6    27.4

tally( ~ substance, data=HELPrct, format="proportion")

## substance
## alcohol cocaine  heroin
##   0.391   0.336   0.274
```

Las estadísticas de resumen pueden ser computadas separadamente para múltiples sub-conjuntos de datos. Esta forma es análoga a la graficación de múltiples variables y puede ser pensada de 3 formas. Cada una tiene el mismo valor.

```
# Edad dependiente de sustancia
sd(   age ~ substance, data=HELPrct)

## alcohol cocaine  heroin
##    7.65    6.69    7.99

# Edad separada por sustancia
sd( ~ age | substance, data=HELPrct)

## alcohol cocaine  heroin
##    7.65    6.69    7.99
```

```
# Edad agrupada por sustancia
sd( ~ age, groups=substance, data=HELPrct)

## alcohol cocaine  heroin
##    7.65    6.69    7.99
```

La función `favstats()` puede computar varios resúmenes para cada sub-conjunto.

```
favstats(age ~ substance, data=HELPrct)

##   substance min Q1 median   Q3 max mean   sd   n missing
## 1   alcohol  20 33   38.0 43.0  58 38.2 7.65 177       0
## 2   cocaine  23 30   33.5 37.2  60 34.5 6.69 152       0
## 3    heroin  19 27   33.0 39.0  55 33.4 7.99 124       0
```

Similarmente, podemos crear tablas de dos por dos que muestren conteos o proporciones.

```
tally(sex ~ substance, data=HELPrct)

##         substance
## sex      alcohol cocaine heroin
##   female      36      41     30
##   male       141     111     94

tally( ~ sex + substance, data=HELPrct)

##         substance
## sex      alcohol cocaine heroin
##   female      36      41     30
##   male       141     111     94
```

Los totales marginales pueden ser agregados con `margins=TRUE`

```
tally(sex ~ substance, data=HELPrct, margins=TRUE)

##         substance
## sex      alcohol cocaine heroin
##   female      36      41     30
##   male       141     111     94
##   Total      177     152    124
```

```
tally( ~ sex + substance, data=HELPrct, margins=TRUE)
```

```
##         substance
## sex      alcohol cocaine heroin Total
##   female      36      41     30   107
##   male       141     111     94   346
##   Total      177     152    124   453
```

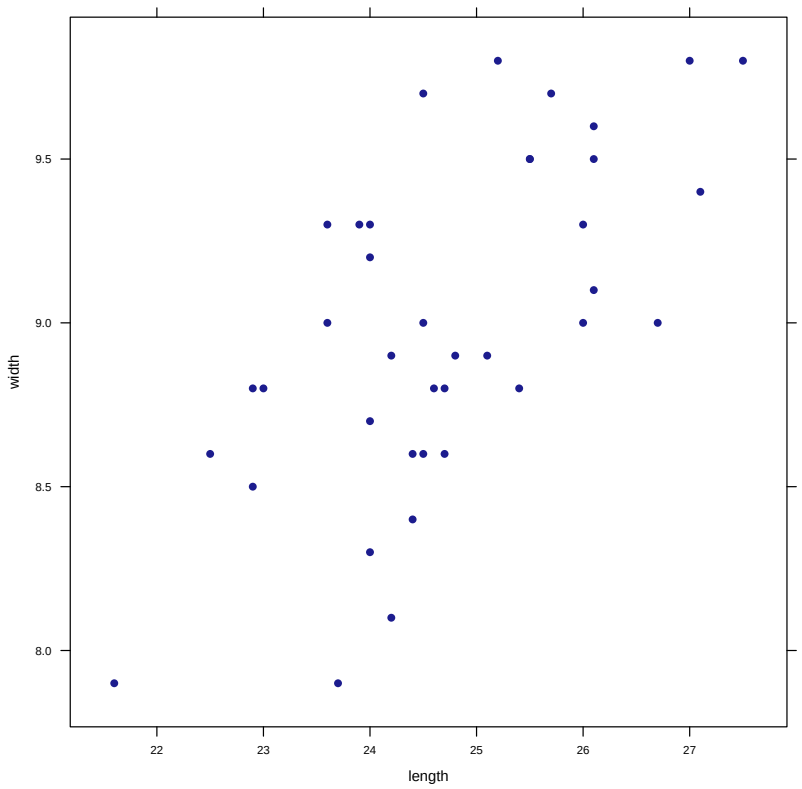
5.4 *Modelos lineales*

Aunque no hemos mencionado los modelos lineales aún, estos son una motivación importante para el patrón de abordaje de los resúmenes gráficos y numéricos. El sistema gráfico `lattice` y el paquete `mosaic` hace posible hacer resúmenes numéricos y resúmenes gráficos con la misma notación. Introduciendo a los estudiantes a la notación para gráficos y resúmenes numéricos, hay muy poco que enseñarles cuando tengan que aprender a ajustar un modelo.

Probablemente usted está pensando que no necesitamos esperar tanto para introducir el modelaje en los cursos de estadística introductoria. Nosotros pensamos lo mismo. Vea el volumen acompañante, *Start Modeling in R*.

Por ejemplo, suponemos que queremos saber cómo el ancho de los pies de los niños, depende del largo del pie. Podemos usar un gráfico de dispersión y podemos construir un modelo lineal usando la misma notación.

```
xyplot(width ~ length, data=KidsFeet)
```




```
    lm(width ~ length, data=KidsFeet)

##
## Call:
## lm(formula = width ~ length, data = KidsFeet)
##
## Coefficients:
## (Intercept)       length
##       2.862        0.248
```

Vamos a tener más que decir sobre el modelaje en otras partes. Por ahora, el punto es que nuestro uso de la notación para resúmenes gráficos y numéricos prepara a los estudiantes para preguntarse cómo `Y` depende de `X` y concretar modelos de dos o más variables cuando sea el momento.

5.5 *Otras pruebas*

Algunos cursos introductorios de estadística introducen a los estudiantes pruebas de hipótesis para una o dos muestras, tanto para medias como para proporciones. El paquete `mosaic` permite utilizar esta notación también.

```
t.test( ~ length, data=KidsFeet)

## ~length
##
##  One Sample t-test
##
## data:  length
## t = 100, df = 40, p-value <2e-16
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  24.3 25.2
## sample estimates:
## mean of x
##      24.7
```

La salida que generan estas funciones incluye más de lo que realmente necesitamos. El paquete `mosaic` nos brinda las funciones `pval()` y `confint()` para extraer p-values e intervalos de confianza:

```
pval(t.test( ~ length, data=KidsFeet))

## ~length
##  p.value
## 3.06e-50

confint(t.test( ~ length, data=KidsFeet))

## ~length
##   mean of x lower upper level
## 1      24.7  24.3  25.2  0.95
```

Más Info

Para un tratamiento más exhaustivo de cómo usar R para temas importantes en un curso tradicional de estadística introductoria, vea *Una guía de estudiantes para R*

Más Info

La prueba de ji-cuadrado puede ser realizada usando la función `chisq.test()`. Esta función es un poco diferente pues opera en datos tabulados, como del tipo producidos por la función `tally()` en lugar de con los datos. Entonces, el uso de la notación pasa en `tally()` en lugar de en `chisq.test()`

```
confint(t.test(length ~ sex, data=KidsFeet))

## length ~ sex
##   mean in group B mean in group G  lower upper level
## 1            25.1            24.3 -0.045  1.61  0.95
```

```
# Usando la distribución binomial
confint(binom.test( ~ sex, data=HELPrct))

##   probability of success lower upper level
## 1                  0.236 0.198 0.278  0.95
```

```
# Usando la aproximación normal de la distribución binomial
confint(prop.test( ~ sex, data=HELPrct))

##       p lower upper level
## 1 0.236 0.198 0.279  0.95

confint(prop.test(sex ~ homeless, data=HELPrct))

##   prop 1 prop 2  lower    upper level
## 1  0.191  0.275 -0.165 -0.00143  0.95
```

5.6 *Las exquisiteces de `lattice`*

En los gráficos que le hemos mostrado, nos hemos enfocado en crear una variedad de gráficos útiles y hemos aceptado (la mayor parte del tiempo) la forma presentación predeterminada de estos. El sistema de gráficos `lattice` proporciona muchas exquisiteces que pueden ser introducidas una vez que las plantillas de gráficos han sido bien aprendidas. Argumentos opcionales de las funciones graficas pueden ser utilizados para agregar o modificar lo siguiente

- La ventana en la que se ve,
- título,
- nombre de los ejes

- colores, formas, tamaños y tipos de linea,
- transparencia,
- tipo de letra

y otras características de los gráficos

Nuestro consejo es guardar esas exquisiteces para cuando un estudiante pregunte o un análisis lo requiera.

5.6.1 *Ejemplo: Número de nacimientos por día.*

Hemos visto los datos de `Births78` en la sección 3.2. Los gráficos abajo toman ventaja de argumentos adicionales para mejorar el gráfico. El primer gráfico ilustra una de las características más importantes de este conjunto de datos – hay usualmente menos nacimientos en dos días de la semana y más en los otros cinco. FDe lo que podemos estar seguros es que es que el año en 1978 inició un domingo.

Más Info

`%%` lleva a cabo un modular aritmético, en este caso dando 7 grupos, cada uno de ellos es un día de la semana

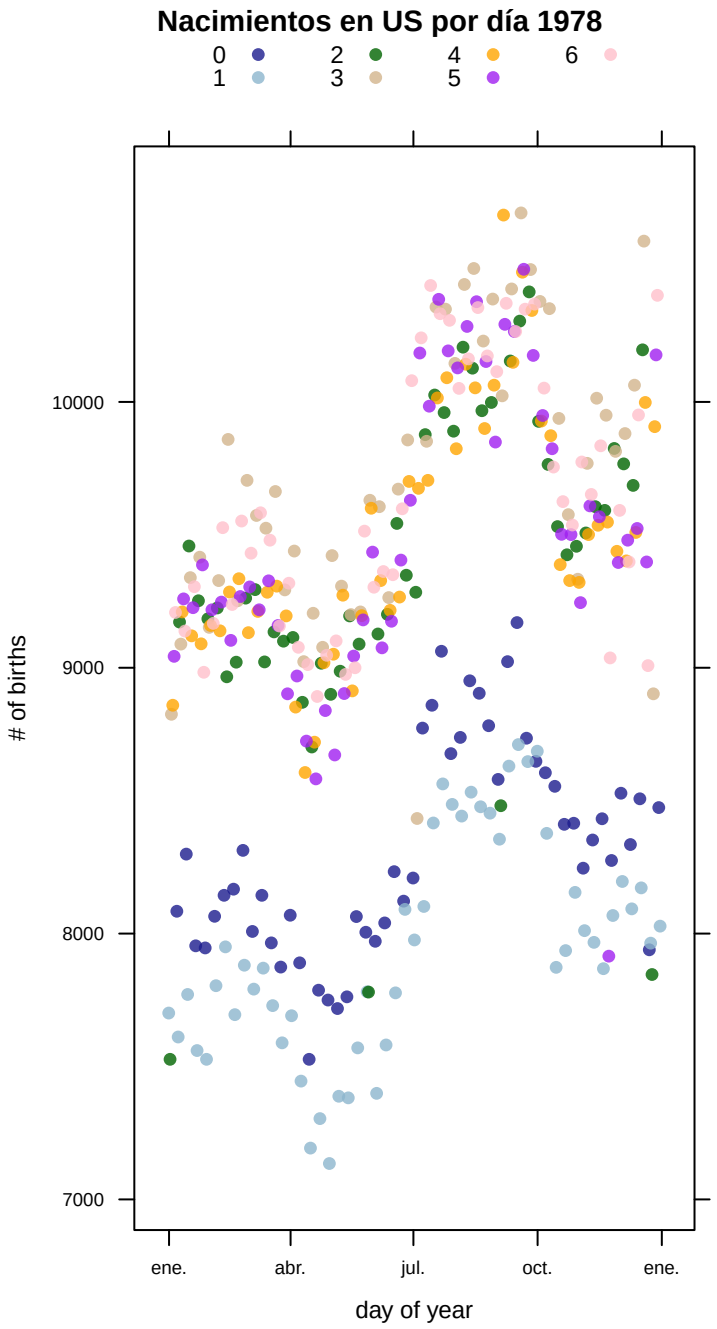
```
xyplot(births ~ date, data=Births78,
        groups=dayofyear %% 7,
        auto.key=list(columns=4),
        main="Nacimientos en US por día 1978",
        xlab="day of year",
        ylab="# of births",
        par.settings=list(
          superpose.symbol=list(pch=16,
                                cex=.8,
                                alpha=.8))
)
```

Más Info

Algunos argumentos aquí usan listas(list). Este es uno de los "tipos contenedores"fundamentales de R. Los instructores pueden obtener beneficios de saber reconocerlos. Vamos a hablar de ellos en el capítulo 7.

Más Info

También podemos usar la función `wday()` en el paquete `lubridate` para conseguir el día de se la semana directamente de `date`.



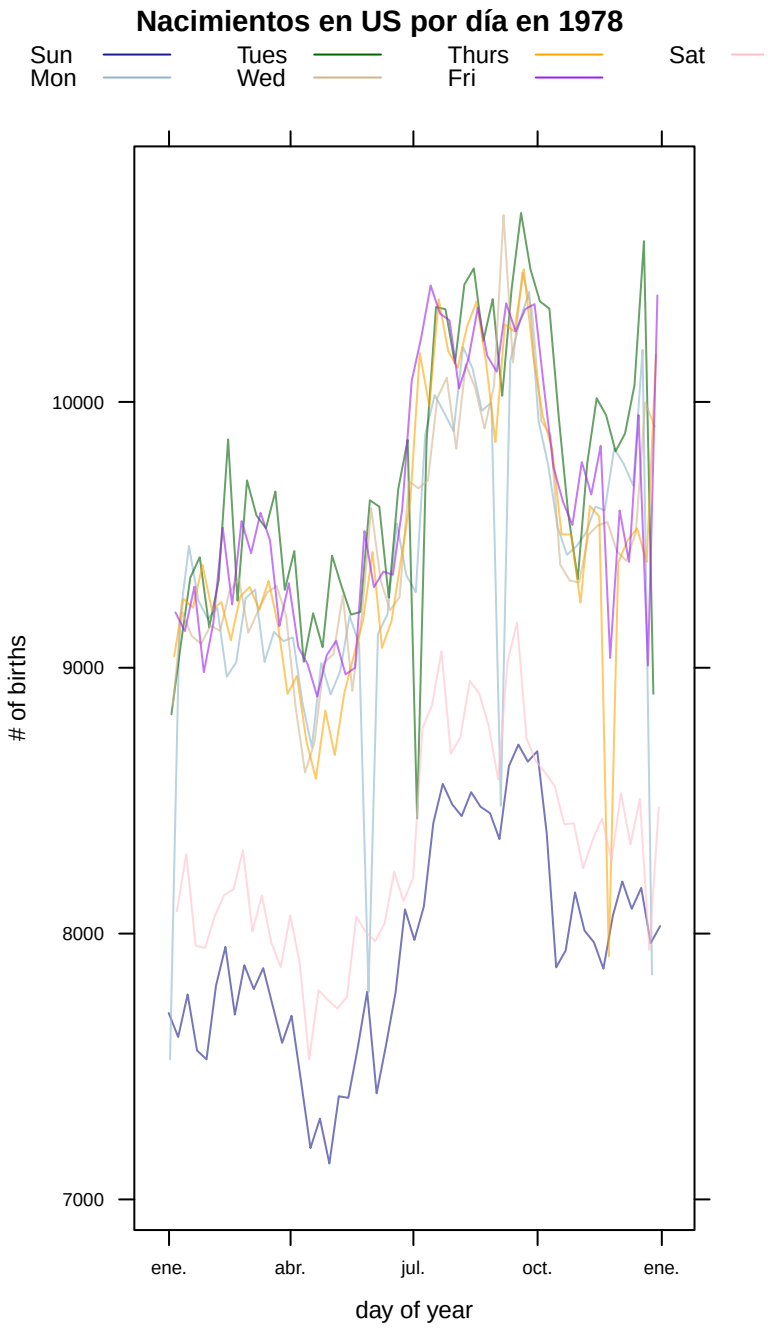
Aquí, hemos usado

- `auto.key` para el control del esquema de la leyenda (4 columnas en lugar de 1)
- `main` , para ponerle título al gráfico
- `xlab` y `ylab` para nombrar los ejes del gráfico

- `par.settings` para ajustar los simbolos o caracteres del gráfico (`pch`), el tamaño de estos (`cex`), y la opacidad (`alpha`) para los gráficos traslapados (`superpose.symbol`).

El siguiente gráfico usa lineas en lugar de puntos, lo cual hace más fácil localizar parte de las observaciones inusuales.

```
xyplot(births ~ date, data=Births78,
  groups=wday, type='l',
  main="Nacimientos en US por día en 1978",
  auto.key=list(columns=4, lines=TRUE, points=FALSE),
  xlab="day of year",
  ylab="# of births"
)
```

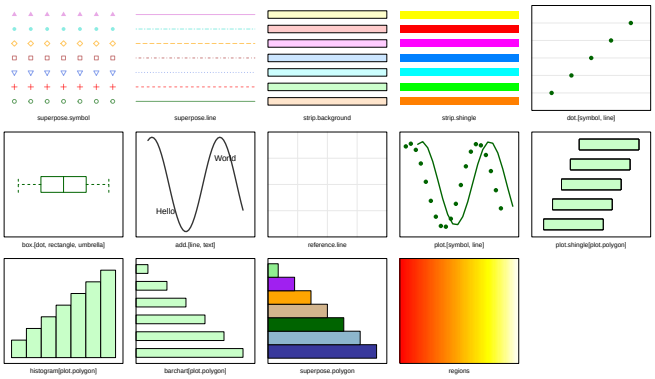


5.6.2 *Temas*

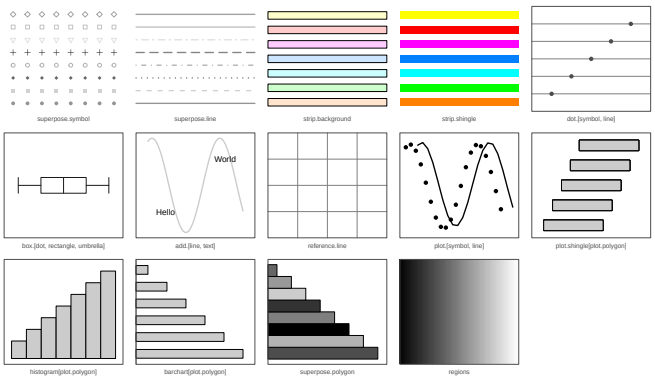
Los ajustes que son usados repetidamente pueden ser reunidos en un tema. El paquete mosaic proporciona un tema llamado `theme.mosaic()`. La función `show.settimgs()`

despliega los ajustes del tema que está activo.

```
trellis.par.set(col.whitebg())
show.settings()
```



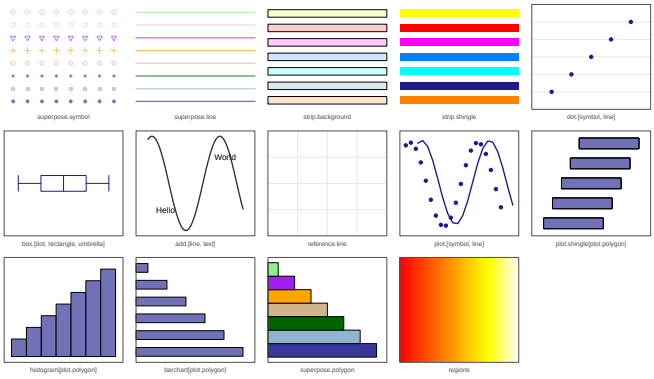

```
trellis.par.set(theme.mosaic(bw=TRUE))
show.settings()
```




```
trellis.par.set(theme.mosaic())
show.settings()
```

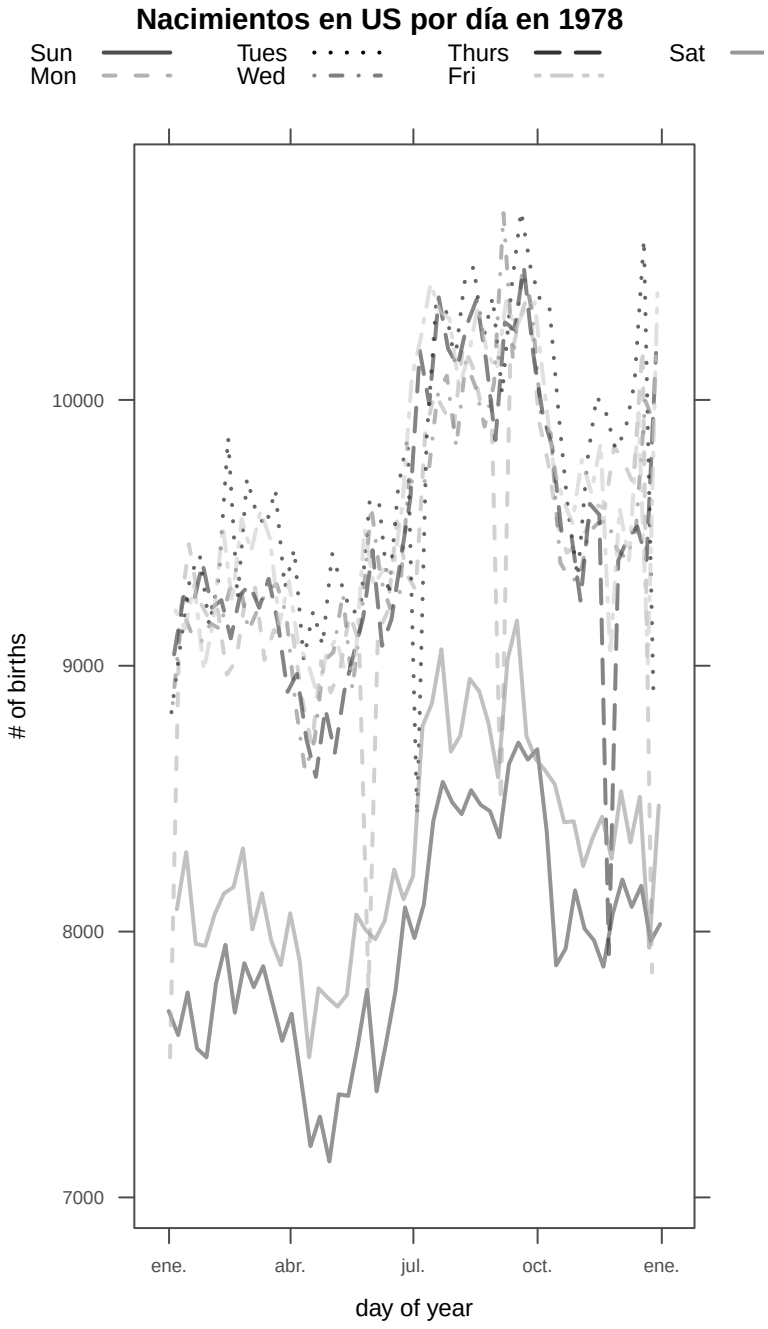
Más Info

En la versión impresa de este libro, los tres ejemplos aparecen en blanco y negro y fueron procesadas con theme.mosaic(bw=TRUE). En la versión online, el primer y tercer ejemplo aparecen en color



Los temas también puede ser asignados a los `par.settings` si queremos afectar únicamente un gráfico

```
xyplot(births ~ date, data=Births78,
  groups=wday, type='l',
  main="Nacimientos en US por día en 1978",
  auto.key=list(columns=4, lines=TRUE, points=FALSE),
  par.settings=theme.mosaic(bw=TRUE),
  xlab="day of year",
  ylab="# of births"
)
```


Nacimientos en US por día en 1978
Sun
Mon
Tues
Wed
Thurs
Fri
Sat
# of births
10000
9000
8000
7000
ene.
abr.
jul.
oct.
ene.
day of year

Some Examples

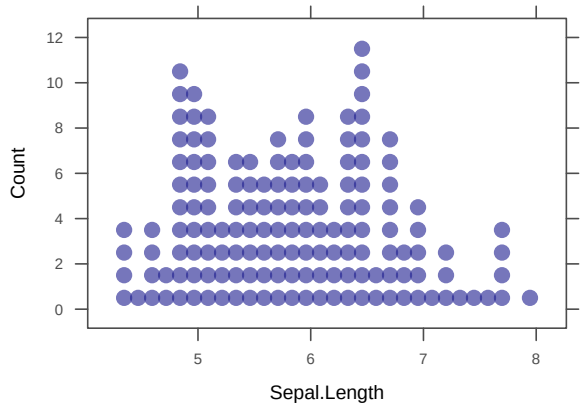
5.7 *Algunos ejemplos adicionales*

5.7.1 *Gráficos de puntos*

Los gráficos de puntos no son usualmente vistos en las lecturas estadísticas como se ven en la educación estadística, donde pueden tomar un rol importante en ayudar a los estudiantes a aprender e interpretar histogramas (y polígonos de frecuencias y gráficos de densidad). Un gráfico de puntos representa cada valor de una variable cuantitativa con un punto. Los valores son redondeados un poco para que los puntos logren alinearse de la mejor manera, y los puntos son apilados en pequeñas torres cuando los valores de los datos en el cluster están muy cerca uno del otro. Los gráficos de puntos son primariamente utilizados cuando se tienen conjuntos de datos de tamaño modesto y que puede ser utilizado como puente o conexión para otros gráficos, cuando no es una conexión entre el componente de un gráfico y la observación individual.

Aquí hay un ejemplo de la utilización de las medidas de sépalo guardadas en el conjunto de datos `iris`

```
dotPlot(~ Sepal.Length, data=iris,
                n=30,      # approx. 30 bins/columns
                alpha=.6)  # partially transparent
```

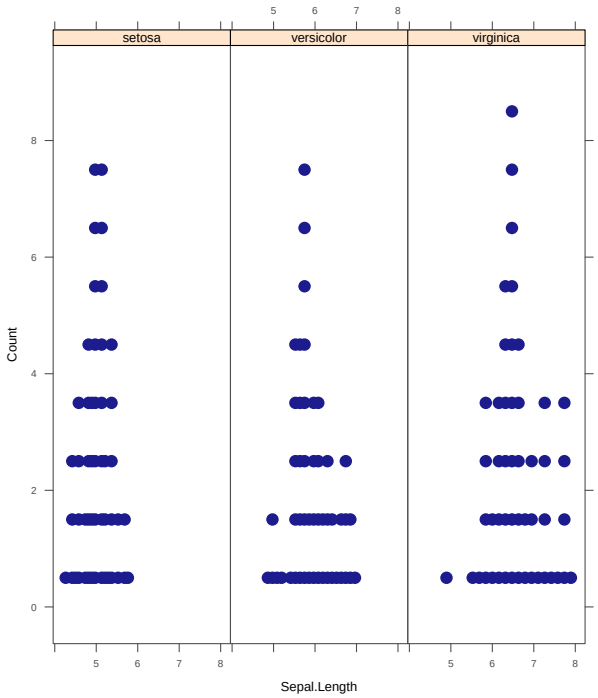


Podemos usar una variable condicional para separar los

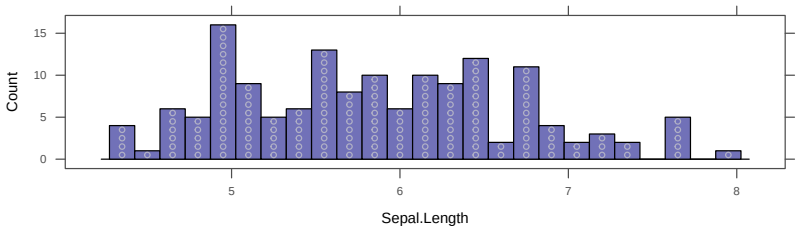
Consejo de Enseñanza
Los gráficos de puntos son útiles para desplegar distribuciones muestrales y distribuciones de bootstrap, especialmente si el número total de puntos es elegido para que sea algo simple, como 1000. En ese caso las probabilidades pueden ser estimadas contando puntos.

gráficos de puntos para cada tres especies en este conjunto de datos.

```
dotPlot(~ Sepal.Length | Species, data=iris, n=20,
        layout=c(3,1)) # 3 columnas (x) y 1 fila (y)
```



La conexión entre los histogramas y los gráficos de puntos puede ser visualizada poniendo uno encima del



otro

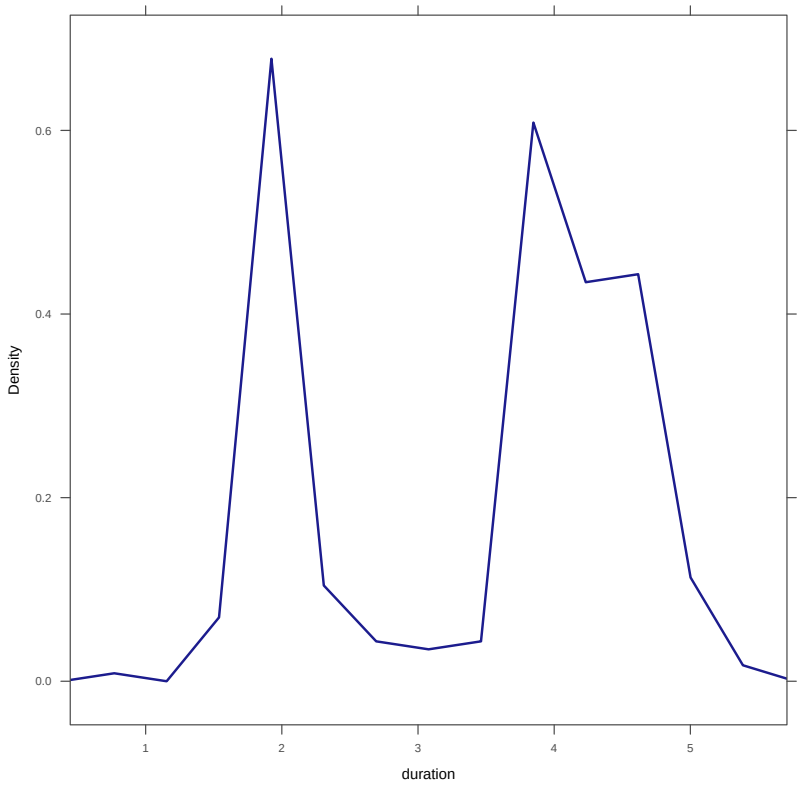
5.7.2 *Polígonos de frecuencia:* `freqpolygon()`

Los **polígonos de frecuencia** y **los gráficos de densidad proveen una alternativa**, diferente a los histogramas que hace más fácil sobreponer representaciones a distintos grupos de datos. Un polígono de frecuencia es creado

del mismo resumen de datos (conteos y clases) que un histograma, pero en lugar de representar cada clase con una barra, es representada con un punto (en el centro donde está lo más alto de la barra del histograma).

Estos puntos son conectados con una línea. Aquí se muestra un ejemplo que muestra la distribución de los tiempos de erupción de una secuencia de observaciones del conjunto de datos faithful.

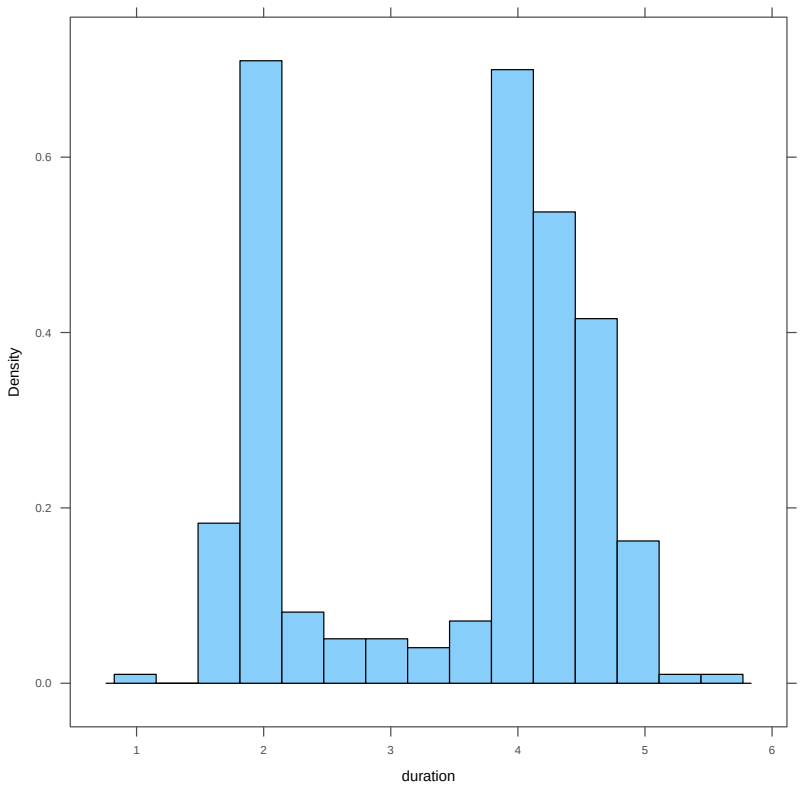
```
require(MASS)
freqpolygon( ~ duration, data=geyser, n=15)
```

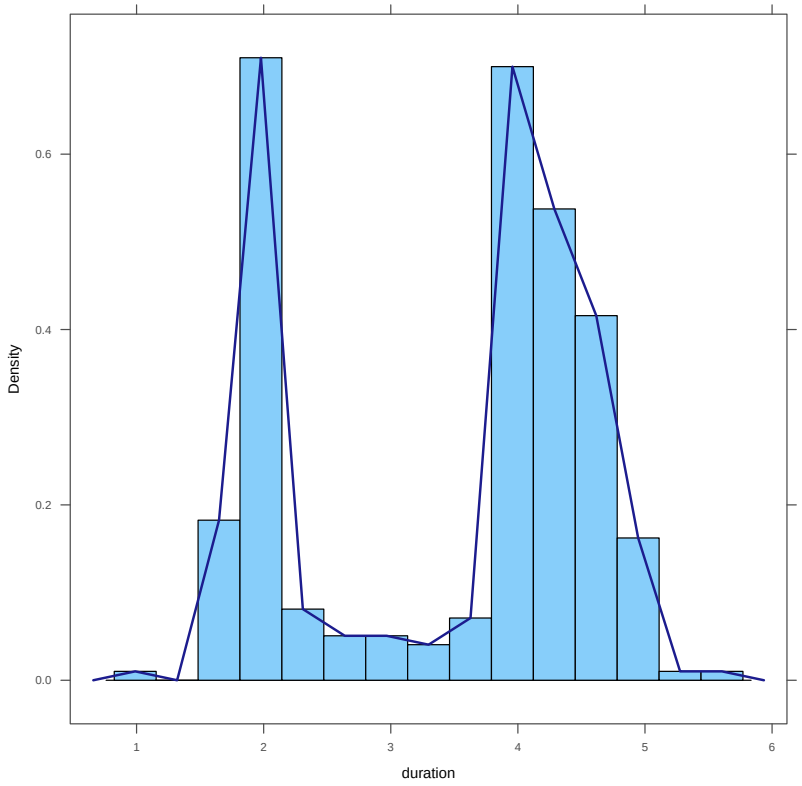


Numéricamente, los datos están siendo resumidos y representados en exactamente la misma forma que con los histogramas, pero visualmente las líneas horizontales y verticales del histograma son reemplazadas por líneas con cierta pendiente.

PRECAUCIÓN!
El conjunto de datos `faithful` contiene datos similares, pero los nombres de la variable están escogidos extrañamente. El set de datos `geyser` en el paquete `MASS` tiene mejores nombres y más datos.

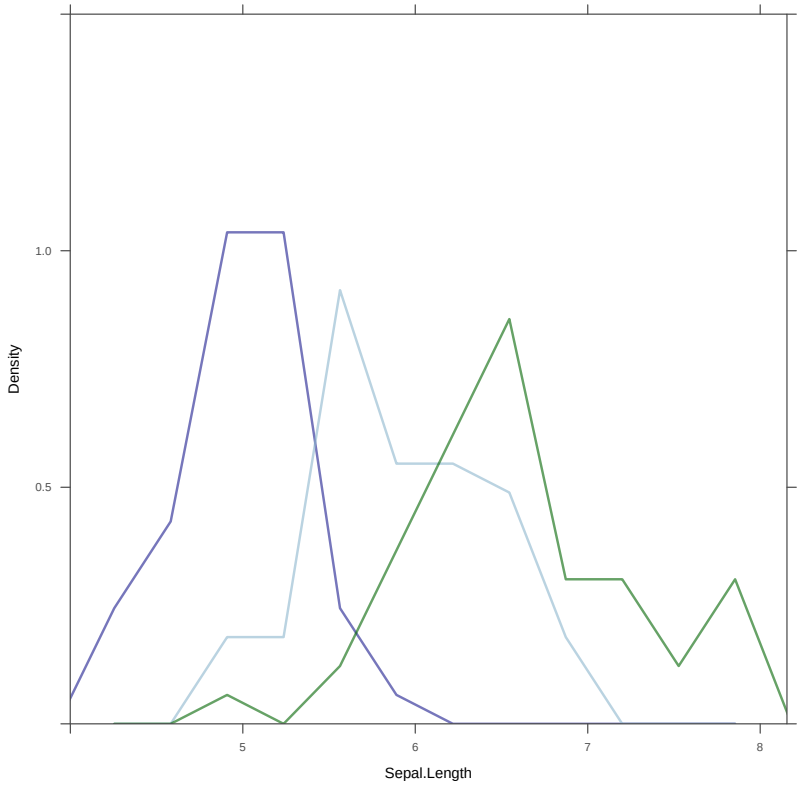
CONSEJO DE ENSEÑANZA
Haga ver que hay una característica interesante de esta distribución, una clara bimodalidad. En particular, la media y la mediana en este caso no son buenas medidas para la duración "típica"de una erupción, puesto que casi ninguna de las duraciones de la erupción está cerca dela media y la mediana.


Density
0.0
0.2
0.4
0.6
1
2
3
4
5
6
duration



Esto puede dar una representación visual más precisa en algunas situaciones (puesto que la distribución puede ser un mejor çoncluyente"). Más importante, hace muchísimo más fácil sobreponer múltiples distribuciones.

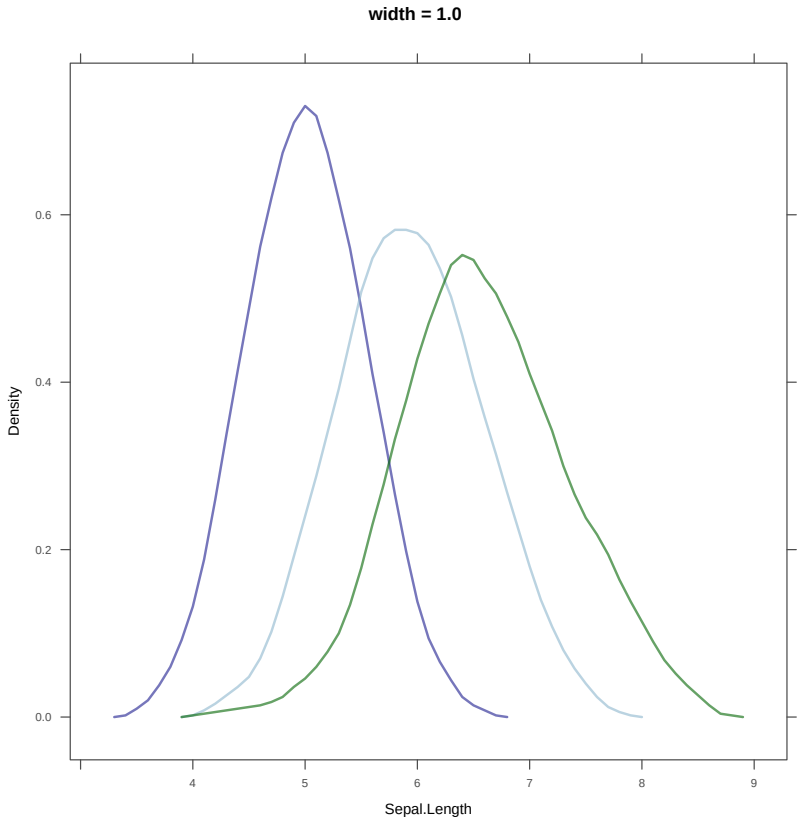
```
freqpolygon( ~ Sepal.Length, data=iris,
                        groups=Species,
                        ylim=c(0,1.5)  # Ajustar el eje y manualmente
)
```



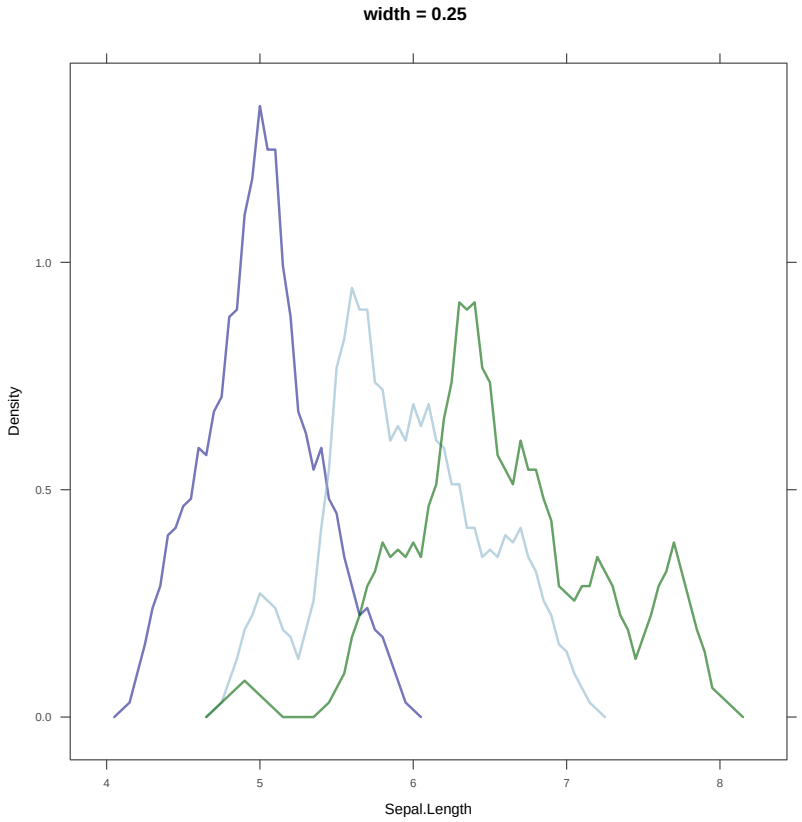
5.7.3 *Gráficos ASH: histogramas de promedio desplazado*

Los histogramas son sensibles a la elección del ancho de las clases y los bordes (o centros). Una forma de reducir esta dependencia son los gráficos ASH o histogramas de promedio desplazado. La altura de un gráfico ASH es la altura promedio de todos los histogramas con anchos de clase cambiados. Si está familiarizado con los gráficos de densidad (discutidos en la siguiente sección), un gráfico ASH se los recordará, pero son más fáciles de explicar a principiantes.

```
ashplot( ~ Sepal.Length, data=iris, groups=Species,
         width = 1.0, main = "width = 1.0")
```



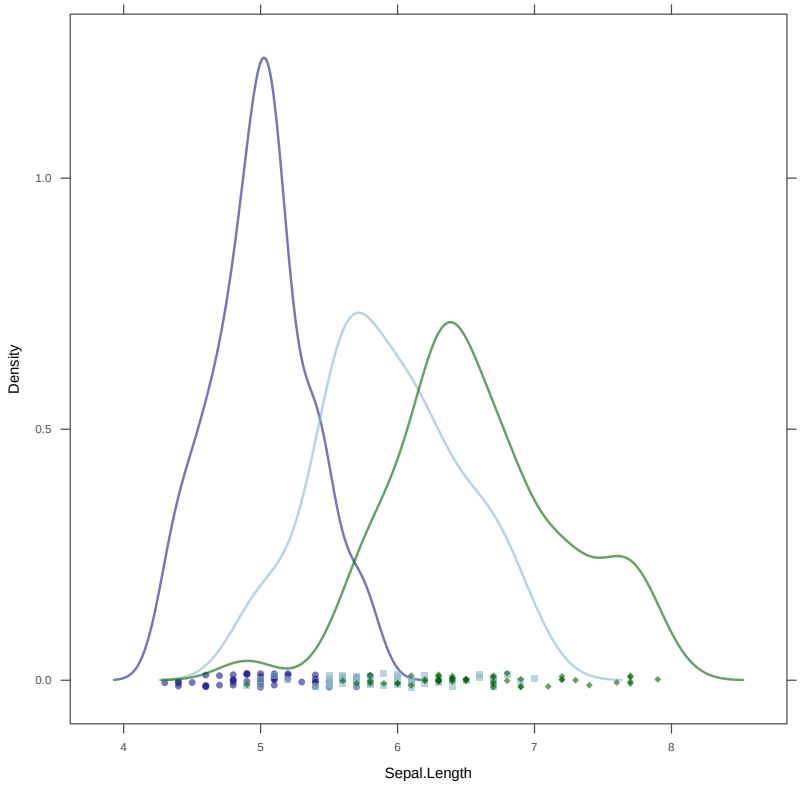

```
ashplot( ~ Sepal.Length, data=iris, groups=Species,
         width = 0.25, main = "width = 0.25")
```



5.7.4 *Gráficos de densidad:* `densityplot()`

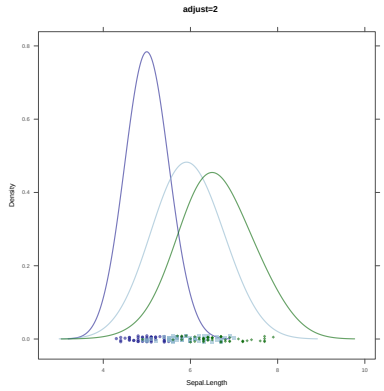
Los gráficos de densidad son similares a los polígonos de frecuencia, pero la representación lineal pieza por pieza es reemplazada por una curva suavizada.

```
densityplot( ~ Sepal.Length, data=iris, groups=Species)
```

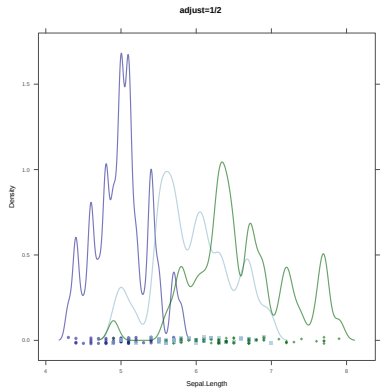


Los principiantes no necesitan saber muchos detalles sobre como una curva suavizada es generada, pero debe ser introducido el argumento de `adjust` que controla el grado de suavizamiento. Es difícilmente equivalente elegir entre clases con intervalos más largos o más anchos para un histograma o un polígono de frecuencia. El valor predeterminado es 1. Los valores mayores suavizan más drásticamente; los valores menores, menos.

```
densityplot( ~ Sepal.Length, data=iris, groups=Species,
             adjust=3, main="adjust=2")
```




```
densityplot( ~ Sepal.Length, data=iris, groups=Species,
             adjust=1/3, main="adjust=1/2")
```



5.7.5 *La escala de densidad*

Hay tres escalas que pueden ser usadas para los gráficos en la sección precedente: `conteo`, `porcentaje`, y `densidad`. Los estudiantes iniciando van a estar más relacionados con la escala de `conteos` y posiblemente también la de `porcentaje`. La escala de `densidad` captura los aspectos más importantes de estos gráficos:

El área es equivalente a la proporción.

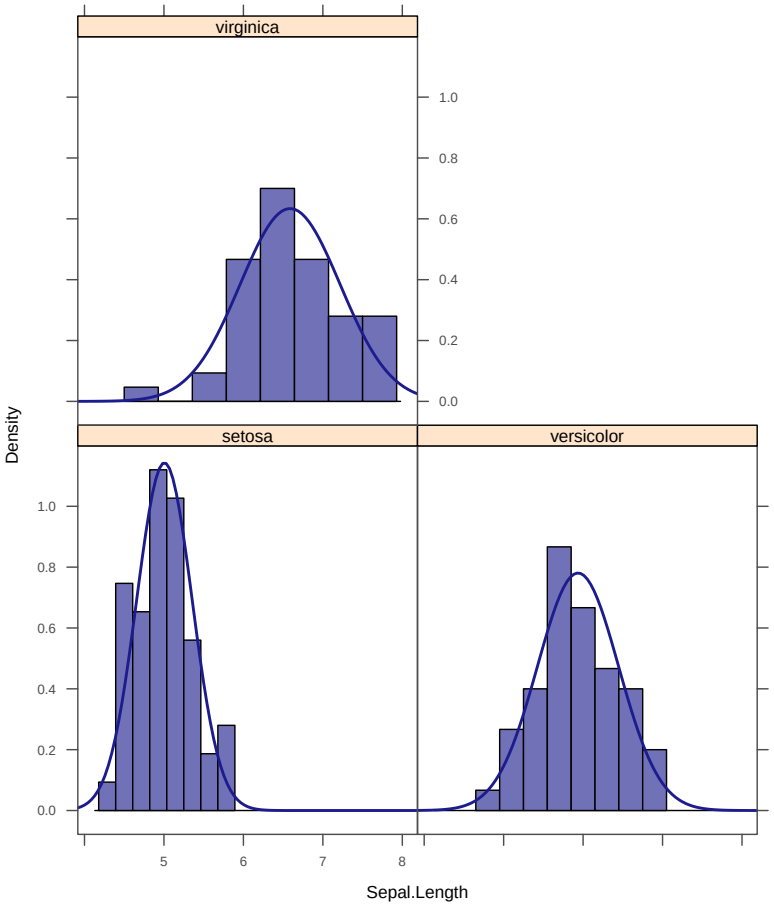
La escala de densidad es escogida, de tal forma que la proporcionalidad es 1, en este caso tenemos que

El área equivale a la proporción.

Esta es la única escala disponible para `densityplot()` y es la escala más adecuada si se está primariamente interesado en la *forma* de la distribución. La escala vertical es poco afectada por la elección del ancho de clase o los multiplicadores de `adjust`. Es también la escala apropiada para utilizar cuando se sobrepone una función de densidad en un histograma, algo que el paquete mosaic hace fácil.

Consejo de Enseñanza
Haga algunos histogramas o polígonos de frecuencia con una escala de densidad y vea si sus estudiantes pueden determinar la escala. Elegir anchos de clase convenientes (pero no 1) y comparar gráficos con diferentes anchos de clase y diferentes tipos de escala puede ayudarlos a conjeturar acerca de la escala de densidad.

```
histogram( ~ Sepal.Length | Species, data=iris, fit="normal")
```



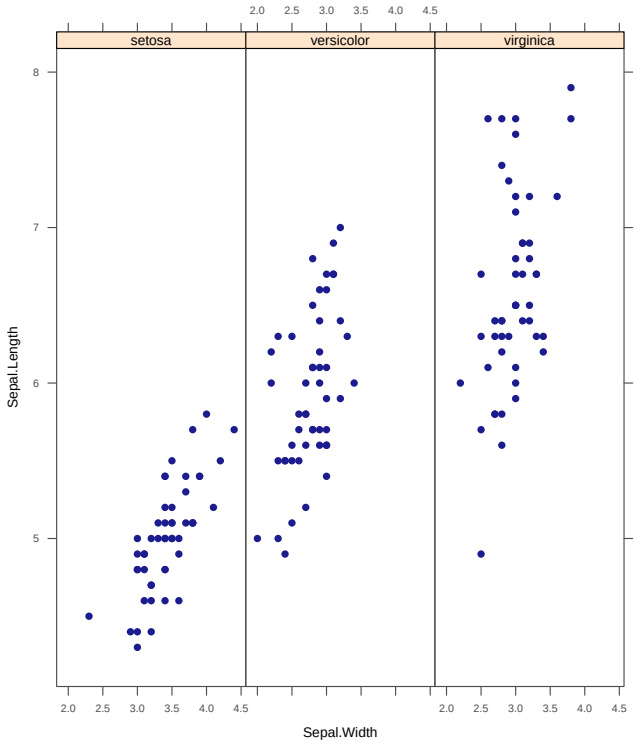
Las otras escalas son de uso primariamente cuando uno quiere poder leer los conteos de clase o porcentajes

del gráfico

5.7.6 ¿Grupos o paneles?

El siguiente ejemplo, que utiliza el conjunto de datos `iris`, nos muestra una comparación con el uso de grupos o paneles para separar subconjuntos de datos. Primero ponemos tres especies en tres paneles diferentes.

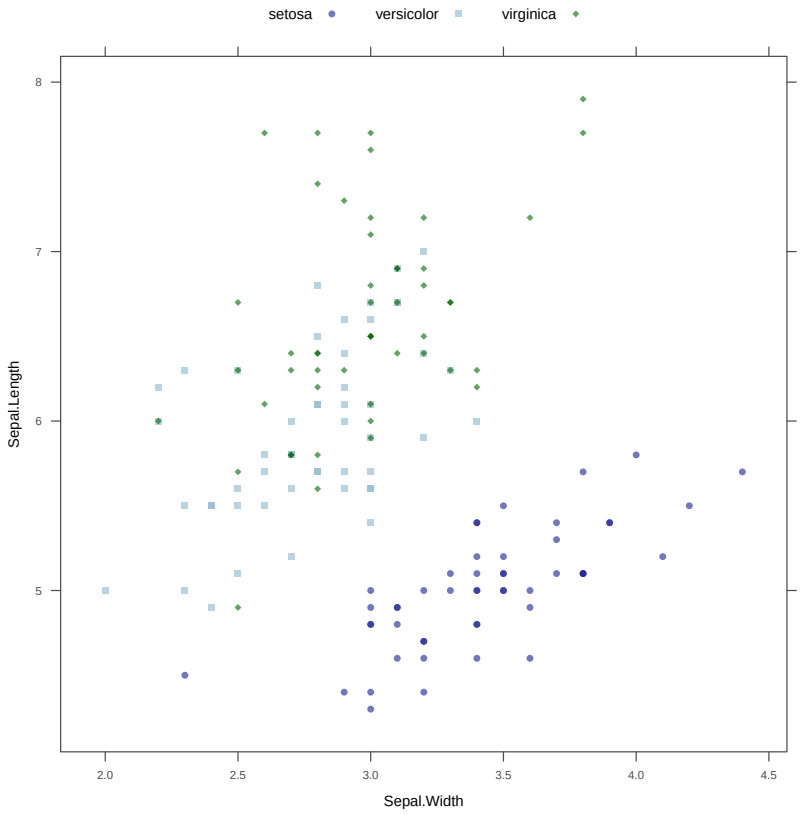
```
xyplot(Sepal.Length ~ Sepal.Width | Species, data=iris,
  layout=c(3,1)) # Los controles de diseño
```




```
#(layout) controlan el número de columnas y filas
```

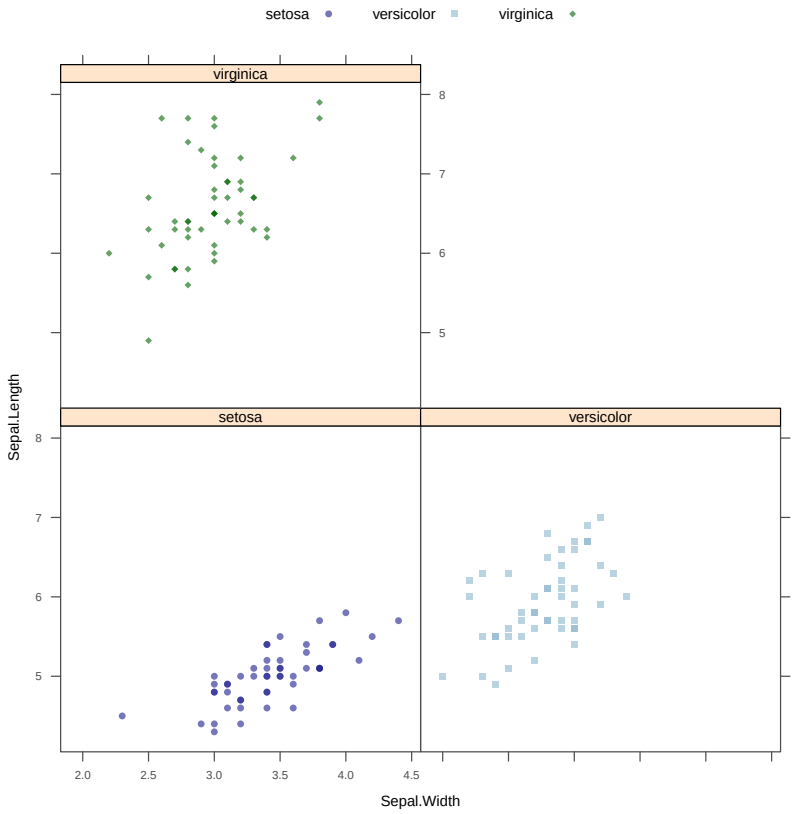
Alternativamente, podemos usar el argumento de grupos para indicar las diferentes especies, usando diferentes símbolos en el mismo panel.

```
xyplot(Sepal.Length ~ Sepal.Width, groups=Species,
  auto.key=list(columns=3), data=iris)
```



En ocasiones es de gran ayudar usar ambos, los paneles y los grupos por símbolo.

```
xyplot(Sepal.Length ~ Sepal.Width | Species, groups=Species,
  auto.key=list(columns=3), data=iris)
```



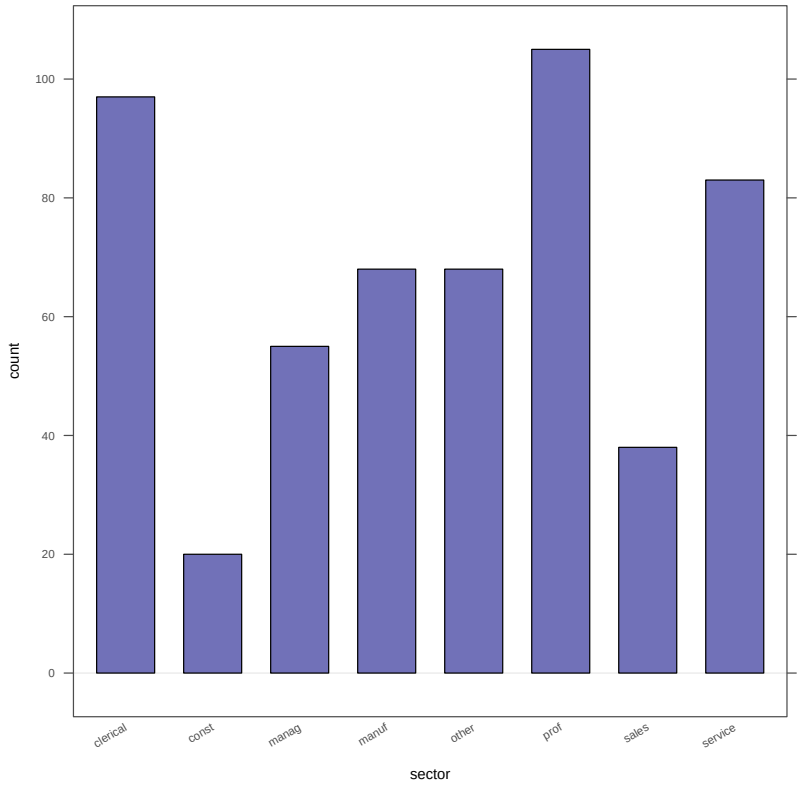
5.7.7 *Lidiando con etiquetas largas*

Suponga que queremos desplegar la siguiente tabla (basada en los datos de la encuesta de población de Estados Unidos de 1985) usando gráficos de barras.

```
tally( ~ sector, data=CPS85)

## sector
## clerical    const    manag    manuf    other     prof
##       97       20       55       68       68      105
##    sales  service
##       38       83
```

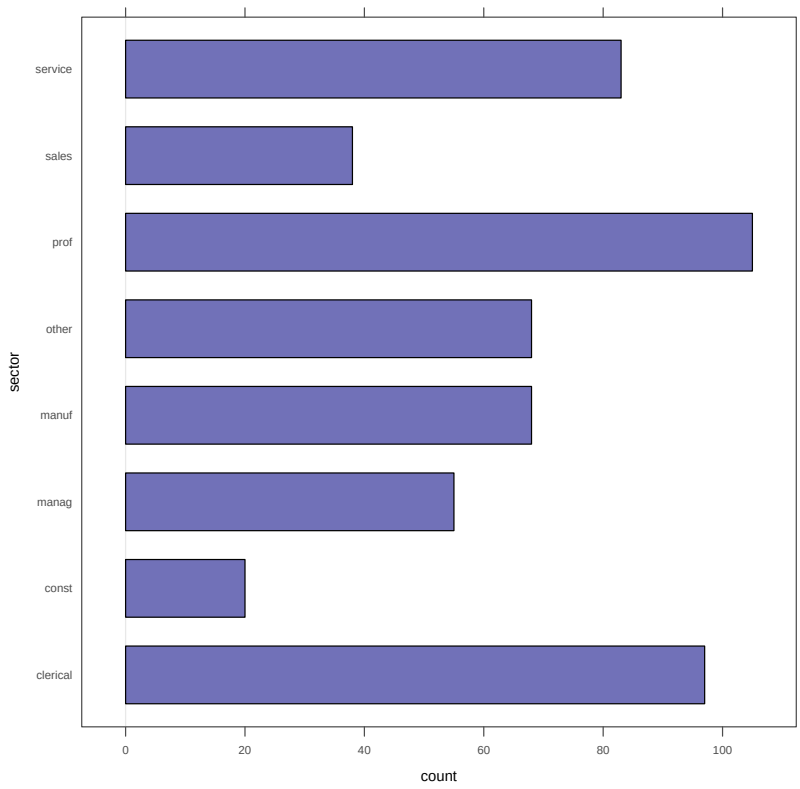
La función de `mosaic` `bargraph()` puede mostrar estas tablas como gráficos de barras, pero no hay suficiente espacio para las etiquetas

```
bargraph(~ sector, data=CPS85)
```



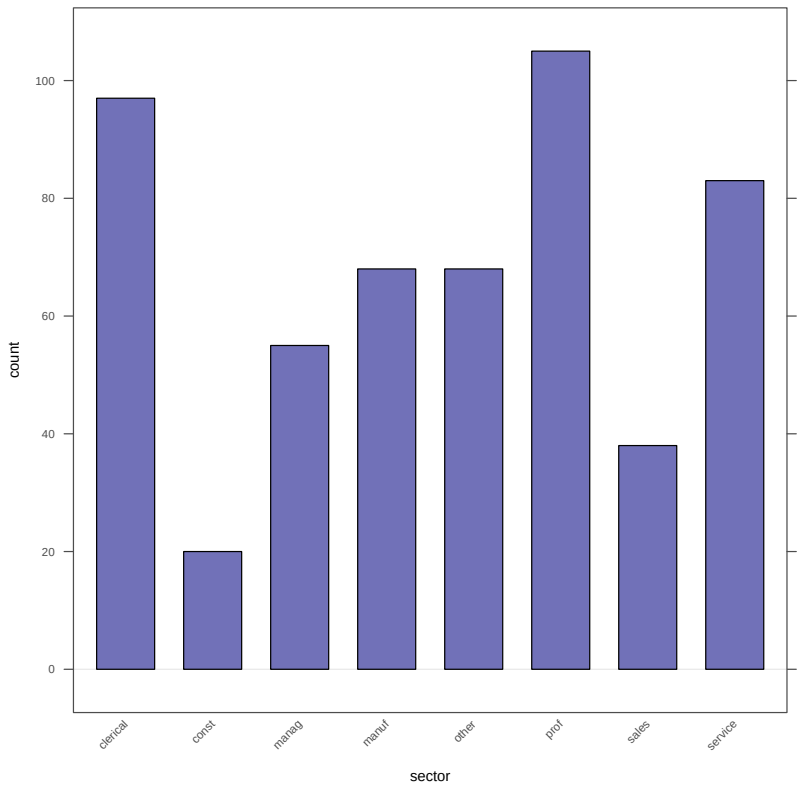
Una solución a esto puede ser utilizar barras horizontales

```
# barras horizontales
bargraph(~ sector, data=CPS85, horizontal=TRUE)
```



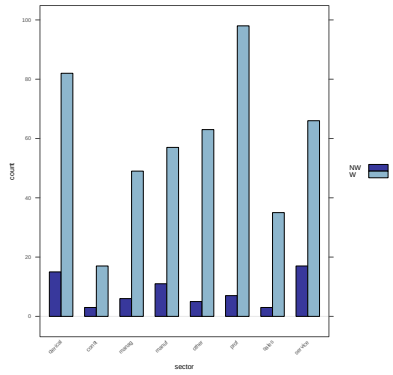
Otra puede ser rotar las etiquetas.

```
bargraph(~ sector, data=CPS85,
                   scales=list(x=list(rot=45)))
```

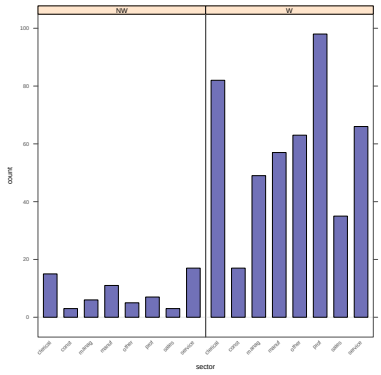


Y como con otros gráficos `lattice`, podemos agregar condiciones de agrupamiento o condicionar nuestro gráfico.

```
bargraph(~ sector, data=CPS85, groups=race,
                   auto.key=list(space="right"),
                   scales=list(x=list(rot=45)))
```



```
bargraph(~ sector | race, data=CPS85,
                          scales=list(x=list(rot=45)))
```



5.8 *Salvando sus gráficos*

Hay unas cuantas formas de guardar gráficos en RStudio, pero la forma más sencilla es probablemente la siguiente:

1. En la viñeta de plots, haga click en el botón .Export".
2. Copie la imagen al clipboard usando el click derecho..
3. Vaya al documento (e.g. Microsoft Word) y pegue la imagen.
4. Arregle el temaño y la posición como sea necesario

Se puede guardar todas estas exportaciones y copiados y pegados si usa `RMarkdown` , o `knitr`/LaTeXpara preparar sus documentos

Otra forma de que un gráfico pueda ser exportado a un documento.

R brinda una función como `pdf()` y `png()` que pueden ser utilizadas para guardar gráficos en una variedad de formatos. Vea la documentación de estas funciones para detalles y enlaces para funciones que pueden ser utilizados para guardar gráficos en otros formatos.

5.9 *`mplot()`*

La función `mplot()` hace una variedad cosas, dependiendo con la información que le demos. Cuando `mplot()` se le da un conjunto de datos en RStudio, abre un gráfico

Más Info

`mplot()` es una función *genérica*. R incluye muchas funciones genéricas (como `print()` y `plot()` y `summary()`). Estas funciones inspeccionan los objetos pasados como argumentos (al menos el primero) y deciden qué hacer con base en la clase de estos argumentos.

interactivo con controles que permiten al usuario seleccionar variables y crear gráficos de diferentes tipos.

5.10 *Repaso de comandos de* R

Aquí hay un pequeño resumen de comandos introducidos en este capítulo.

```
require(mosaic)                          # Carga el paquete mosaic
require(mosaicData)                      # carga los datos del paquete

tally( ~ sector, data=CPS85)             # Tabla de frecuencias
tally( ~ sector + race, data=CPS85)      # Tabulación cruzada del
#sector por etnia
mean( ~ age, data = HELPrct)             # media de la edad de los participantes
mean( ~ age | sex, data = HELPrct)       # Media de la edad en los participante
#separada por sexo
mean(age ~ sex, data = HELPrct)          # Media de la edad
#de los sujetos hombres y mujeres
median(x); var(x); sd(x);                # Más resumenes numéricos
quantile(x); sum(x); cumsum(x)           # Aún más resumenes numéricos
favstats( ~ Sepal.Length, data=iris)     # Cálcula los resúmenes
#numéricos usuales.

histogram( ~ Sepal.Length | Species, data=iris)   # histogramas
#(con caracterísitas extra)
dotPlot( ~ Sepal.Length | Species, data=iris)     # Gráficos de puntos
#para cada especie
freqpolygon( ~ Sepal.Length, groups = Species, data=iris) # Polígonos
#de frecuencia sobrepuestos
densityplot( ~ Sepal.Length, groups = Species, data=iris) # gráficos
#de densidad sobrepuestos
qqmath( ~ age | sex, data=CPS85)                          # Gráfico
#cuantil-cuantil
bwplot(Sepal.Length ~ Species, data = iris)               # Gráficos
#de caja uno al lado del otro
xyplot(Sepal.Length ~ Sepal.Width | Species, data=iris)   # Gráficos
#de dispersión uno al lado del otro
bargraph( ~ sector, data=CPS85)                           #Gráfico de
#barras
```

```
mplot(HELPrct)                          # Gráfico interactivo
```

5.11 Ejercicios

5.1. El conjunto de datos `Utilities2` en el paquete mosaic contiene información sobre las cuentas de varias utilidades en una residencia en Minnesota, recolectadas alrededor del tiempo. Puesto que el número de días en el ciclo de facturación varía de mes a mes, variables como `gasbillpay` (`elecbillpday`, etc.) contiene la cuenta del gas (cuenta de electricidad, etc.) dividido por el número de días del ciclo de facturación.

a) Use la documentación para determinar que contiene la variable `kwH`.

b) Haga un gráfico de dispersión de `gasbillpday` y `monthsSinceY2K`

```
xyplot(gasbillpday ~ monthsSinceY2K, data=Utilities2,
                     type='l')        # la letra l
```

c) ¿Qué hace `type='l'`? Haga su gráfico sin esto. ¿Cuál hace más fácil leer la situación?

d) d) ¿Qué pasa si reemplazamos `type='l'` con `type='b'`?

e) Haga un gráfico de dispersión de `gasbillpday` por mes(`month`). ¿Qué observó?

f) Haga diagramas de caja uno al lado del otro de `gasbillpday` por mes usando el conjunto de datos `Utilities2`. ¿Qué observa?

La primera vez tal vez no le devuelva lo que espera. La razón es que el mes está codificado usando números, entonces R lo trata como un dato numérico. Queremos que lo trate como datos categóricos. Para hacer esto en R use `factor(month)` en lugar de `month`. R llama los datos categóricos **factor**.

g) Haga cualquier gráfico que desee usando estos datos. Incluya una copia de su gráfico y una discusión sobre qué se puede aprender del mismo.

6
Inferencia basada en la simulación

El abordaje del remuestreo se ha vuelto cada vez más importante en la educación estadística[1,2].El paquete `mosaic` proporciona una herramienta simplificada para respaldar la enseñanza de la inferencia basada en las pruebas de aleatorización y los métodos de bootstrap. Nuestra meta es enfocar la atención en las partes importantes de estas técnicas (e.g, donde la aleatoriedad entra y nos dice cómo usar la distribución resultante) mientras esconde algunos detalles involucrados en crear círculos y acumular valores

[1] N. Tintle, B. Chance, G. Cobb, S. Roy, T. Swanson, and J. VanderStoep. Combating anti-statistical thinking using simulation-based methods throughout the undergraduate curriculum. *The American Statistician*, 69(4), 2015

[2] Tim C. Hesterberg. What teachers should know about the bootstrap: Resampling in the undergraduate statistics curriculum. *The American Statistician*, 2015

6.1 *Empezando tempranamente*

Una de las ventajas de la inferencia basada en la simulación es que se puede empezar a enseñar la inferencia tempranamente en el curso. La sección 3.1 describe un ejemplo (basado en la dama que probaba té de Fisher) de que la usamos en ocasiones desde el primer día de clases. Los libros de texto que usan abordajes basados en simulaciones también empiezan la discusión del proceso de inferencia inmediatamente, usando otros ejemplos.[3,4]. Incluso cuando se enseña un curso más tradicional, la simulación de la dama que prueba té o algún otro ejemplo pueden ser introducidos tempranamente en el curso para ayudar a los estudiantes a entender ideas claves que involucran la prueba de hipótesis y la estimación.

[3] Robin H Lock, Patti Frazer Lock, and Kari Lock Morgan. *Statistics: Unlocking the Power of Data*. Wiley Global Education, 2012

[4] Nathan Tintle, Beth Chance, George Cobb, Allan Rossman, Soma Roy, Todd Swanson, and Jill VanderStoep. *Introduction to Statistical Investigations*. Wiley Global Education, 2015

6.2 *Prueba de hipotesis*

La prueba de hipótesis puede ser pensada como un proceso de 4 pasos:

1. Establecer una hipótesis nula y alternativa
2. Calcular un estadístico de prueba
3. Determinar un valor p.
4. Esbozar una conclusión

En los cursos tradicionales de estadística introductoria, una vez que el flujo de trabajo ha sido dominado, el principal trabajo para los estudiantes es aplicar la fórmula correcta para calcular la prueba estadística estándar en el paso 2 y usar una tabla o la computadora para determinar el valor p basado en la distribución teórica conocida (usualmente aproximada) para el estadístico de prueba sometido a la hipótesis nula.

En un abordaje basado en las simulaciones, los pasos 2 y 3 cambian. En el paso 2 no es requerido que el estadístico de prueba sea normalizado de acuerdo a una distribución conocida. En lugar de esto, el estadístico de prueba es más natural, como la diferencia entre dos medias

$$\overline{y}_1 - \overline{y}_2$$

puede ser usado en lugar de la prueba t para dos muestras

$$\frac{\overline{y}_1 - \overline{y}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}.$$

En el paso 3, usamos la aleatorización para aproximar la distribución muestral de nuestro estadístico de prueba. Nuestro ejemplo de la dama que cata té muestra como esto puede ser hecho desde los principios de la clase, tan temprano como en el primer día.[5] . Este ejemplo es un poco inusual, después de todo. Como las distribuciones muestrales son tan sencillas, la simulación requiere crear una distribución de aleatorizaciones que es completamente específicada sin datos de referencia: Es una distribución binomial con parámetros determinados por el

[5] Ver sección 3.1

tamaño de muestra y la hipótesis nula, podemos simular con `rflip()`.

De manera más típica, vamos a usar la aleatorización para crear nuevos conjuntos de datos simulados, que son como nuestros datos originales en cierta forma, pero hacen la hipótesis nula verdadera. Para cada conjunto de datos simulado, calculamos el estadístico, como lo hicimos con la muestra original. Juntos, esta colección de estadísticos de prueba calculados de muestras simuladas constituye nuestra distribución de aleatorizaciones.

Cuando creamos una distribución de aleatorizaciones, vamos a intentar satisfacer 3 principios

1. Ser consistentes con la hipótesis nula

   Necesitamos simular un mundo en el cual la hipótesis nula es real. Si no hacemos esto, no estaremos poniendo a prueba nuestra hipótesis nula.

2. Use los datos en la muestra original

   Los datos originales deberían aclarar algunos aspectos de la distribución que no están determinados por la hipótesis nula. Por ejemplo, la hipótesis nula sobre una media no nos dice nada sobre la forma de la distribución de la población, pero los datos nos pueden dar una pista.

3. Refleje la forma en la que los datos originales fueron recolectados

6.2.1 Pruebas de permutaciones usando `shuffle()`.

El paquete `mosaic` proporciona `shuffle()` como un sinónimo de `sample()`. Cuando se usa sin argumentos adicionales, permuta el primer argumento.

```
shuffle(1:10)

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

shuffle(1:10)

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

Aplicar `shuffle()` a una variable explicatoria nos permite probar la hipótesis nula de que la variable explicatoria, en efecto, no tiene poder explicatorio. Esta idea se puede usar para probar

- la equivalencia de dos o más proporciones,
- la equivalencia de dos o más medias,
- si el parámetro de regresión es 0.

Por ejemplo, probemos si un hombre joven y una mujer joven tienen el mismo promedio de temperatura corporal, usando el conjunto de datos que contiene temperaturas corporales de 50 estudiantes universitarios, 25 hombres y 25 mujeres.

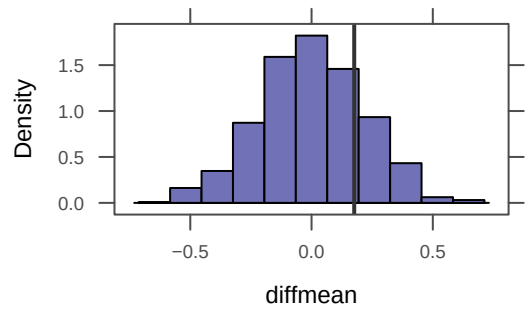
```
require(Lock5withR)
inspect(BodyTemp50)

##
## categorical variables:
##   name  class levels  n missing
## 1  Sex factor      2 50       0
##                                    distribution
## 1 Female (50%), Male (50%)
##
## quantitative variables:
##       name   class  min   Q1 median   Q3 max mean    sd  n
## 1 BodyTemp numeric 96.4 97.8   98.2 98.8 101 98.3 0.765 50
## 2    Pulse integer 57.0 70.2   75.0 78.0  89 74.4 6.440 50
## 3   Gender integer  0.0  0.0    0.5  1.0   1  0.5 0.505 50
##   missing
## 1       0
## 2       0
## 3       0
```

1. Enuncie la hipotesis nula y alternativa.
   - H_0: La media de temperatura del cuerpo es la misma para hombres y mujeres
   - H_a: La media de temperatura difiere entre hombres y mujeres
2. Calcule un estadístico de prueba

```
favstats( BodyTemp ~ Sex, data = BodyTemp50)

##      Sex  min   Q1 median   Q3   max mean    sd  n missing
## 1 Female 96.9 97.6   98.1 98.7  99.5 98.2 0.675 25       0
## 2   Male 96.4 98.0   98.3 98.8 100.8 98.3 0.851 25       0

T <- diffmean( BodyTemp ~ Sex, data = BodyTemp50); T

## diffmean
##    0.176
```

3. Use la aleatorización para calcular el valor p.

```
Temp2.Null <-
  do(1000) * diffmean( BodyTemp ~ shuffle(Sex), data = BodyTemp50)
histogram( ~ diffmean, data = Temp2.Null, center = 0, v = 0.176)
```




```
tally( ~ (diffmean >= T), data = Temp2.Null)

## (diffmean >= T)
##  TRUE FALSE
##   214   786

prop( ~ (diffmean >= T), data = Temp2.Null)

##  TRUE
## 0.214
```

4. Esboce una conclusión.

   El valor p es grande, entonces estos datos no dan una razón para rechazar la hipótesis de que los estudiantes universitarios, masculinos y femeninos, tienen la misma temperatura corporal.

6.2.2 *Calculando valores p*

En el ejemplo anterior, no era estrictamente necesario porque el histograma claramente muestra que el estadístico observado (0.176) no es inusual, si la hipótesis nula fuera verdad, entonces estos datos no nos ofrecen ninguna razón para rechazar la hipótesis nula de que los estudiantes universitarios masculinos y femeninos tienen la misma media de temperatura.

Aun así, hay dos asuntos relacionados al cálculo del valor p que queremos agregar en este ejemplo: incluir el estadístico de prueba observado en la distribución nula, y calcular valores p de 2 colas.

¡Precaución!
Si está utilizando un libro de texto que cubre el tema de pruebas de aleatorización, revise si incluye el estadístico de prueba calculado de los datos originales en la distribución nula o no.

Si la hipótesis nula es verdadera, entonces no solo los datos generados aleatoriamente, sino también los datos originales fueron generados en un mundo en el cual la hipótesis nula es verdadera. Entonces tiene sentido agregar el estadístico de prueba a la distribución de aleatorizaciones antes de calcular el valor p. Esto tiene dos ventajas. Primero, asegura que nuestro rango de error tipo I no es más grande que el rango nominal. Segundo, evita reportar un valor p de 0, puesto que siempre va habrá al menos un estadístico de prueba tan extremo como el que se generó de los datos originales.

¡Precaución!
Aunque usar 999 o 9999 réplicas puede resultar en valores p que son "número redondeados", hay algún riesgo de que estudiantes usen la distinción del 999 vs 1000 como su forma primaria de decir si está creando una distribución de aleatorizaciones o una de bootstrap.

Para simplificar este cálculo, podemos elegir usar 999 o 9999 replicas en lugar de 1000 o 10000. El paquete mosaic incluye la función prop1() que agrega un conteo adicional al numerador y el denominador, con el propósito de automatizar este tipo de cálculos de valores p. Esto resulta en un valor p (de una cola) un poco más grande

```
prop1( ~ (diffmean >= T), data = Temp2.Null)

##  TRUE
## 0.215
```

El único reto para el instructor es decidir cuándo introducir el cambio y si introducir este pequeño cambio va afectar el cálculo del valor p.

Pero necesitamos un valor p de dos colas, dada nuestra hipótesis alternativa. La forma preferida de calcular valores p de dos colas es también la más sencilla: Use el doble del valor p de una cola

```
2 * prop1( ~ (diffmean >= T), data = Temp2.Null)

## TRUE
## 0.43
```

Un abordaje alternativo en ocasiones visto, agregaría la proporción de la distribución de aleatorizaciones que está debajo de -T = -0.176. Para una distribución de aleatorizaciones simétrica, esto debería dar un resultado similar, pero esto no se lleva a cabo cuando la distribución de aleatorizaciones está muy distorsionada, es un poco más difícil de calcular, y no es una transformación invariante, entonces las pruebas que son equivalentes como pruebas de una cola, pueden no resultar equivalentes en pruebas de dos colas. Parece que no hay razón para introducir este método a los estudiantes.

Consejo de Enseñanza
Esta alternativa puede ser cubierta en el libro de texto que esté utilizando, entonces puede ser que sus estudiantes la usen, aunque no la haya enseñado

6.2.3 *Algunos ejemplos adicionales*

La técnica del "shufflingçon una variable explicatoria puede ser aplicado a un amplio rango de situaciones. Las siguientes plantillas ilustran la similitud de estos

```
Two.Proportions <- do(999) * diffprop(y ~ shuffle(x),     data = Data)
Two.Means       <- do(999) * diffmean(y ~ shuffle(x),     data = Data)
Linear.model    <- do(999) *       lm(y ~ shuffle(x) + a, data = Data)
Two.Way.Table   <- do(999) *    chisq(y ~ shuffle(x),     data = Data)
```

Como ejemplo, consideremos la proporción de sujetos en el Health Evaluation and Linkage to Primary Care que fueron admitidos para el programa de abuso de sustancias: alcohol, cocaína y heroína. Nos gustaría saber si hay evidencia de que estas proporciones difieren para hombres y mujeres. En nuestro conjunto de datos, observamos diferencias modestas

Nota
La función `chisq()` calcula un estadístico ji-cuadrado tanto de la fórmula como del conjunto de datos, de una tabla producida por `tally()`, o de un objeto producido por `chisq.test()`.

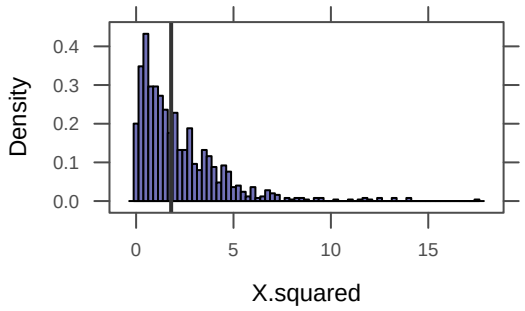
```
tally( substance ~ sex, data = HELPrct,
       format="prop", margins = TRUE)

##           sex
## substance female  male
##   alcohol  0.336 0.408
##   cocaine  0.383 0.321
```

```
##   heroin   0.280 0.272
##   Total    1.000 1.000
```

¿Pueden esas diferencias ser atribuidas a la suerte? ¿O los resultados nos dan evidencia confiable que la droga de escogencia varía (un poco) entre hombres y mujeres?

Podemos simular en un mundo en el cual las proporciones varían solamente por la variabilidad del muestreo aleatorio usando `shuffle()` para permutar la etiqueta del sexo (o equivalentemente la de la sustancia)

```
T <- chisq(substance ~ shuffle(sex), data = HELPrct); T  # test statistic

## X.squared
##       1.8

Substance.Null <-
  do(999) * chisq(substance ~ shuffle(sex), data = HELPrct)
histogram( ~ X.squared, data = Substance.Null, v = T, width = 0.25)
```




```
prop1( ~(X.squared >= T), data = Substance.Null)

##  TRUE
## 0.453
```

El histograma y el valor p sugieren que las diferencias entre hombres y mujeres no son estadísticamente significativos.

6.2.4 *Probando una sola media*

Un problema con nuestro sistema es la prueba para una sola media. Ilustremos un una prueba con $H_0 : \mu =$

NOTA
De alguna forma, esta es la hipótesis que fue más difícil de manejar con nuestro sistema. Vea más adelante, por una razón no nos molestó mucho.

98,6 usando una muestra de 50 temperaturas corporales. Probar la hipótesis nula de la forma

- H_0: $\mu = \mu_0$

presenta un caso especial. Contrario a los ejemplos anteriores a este, no hay variables explicatorias qué mezclar. Contrario a la prueba para una única proporción, la hipótesis nula no viene completamente a especificar la distribución de muestreo.

At least there is an obvious candidate for a test statistic: the sample mean, $\overline{y}$.

NOTA
Muchos libros usan $\overline{X}$ en lugar de $\overline{Y}$.

```
mean( ~ BodyTemp, data = BodyTemp50)

## [1] 98.3
```

Este estadístico de prueba es fácilmente aplicado a cualquier conjunto de datos, solo necesitamos una forma de generar conjuntos de datos aleatorios en los cuales la hipótesis nula es verdadera. Como se menciona anteriormente, no hay una variable explicatoria que se pueda mezclar. Si mezclamos `BodyTemp` (o todo el conjunto de datos), vamos a obtener la misma media siempre, puesto que la media no depende del orden.

En lugar de esto, muestreamos con reemplazo. La función `resample()` nos ayuda en esto.

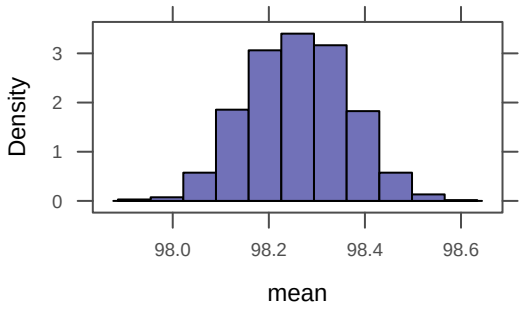
```
resample(1:10)   # notice the duplicates

##  [1] 10  5  5  6  1  3  8  4  6  2
```

Podemos re-muestrear variables individuales o toda la tabla de datos. (Puesto que sólo hay una variable en este análisis, los resultados van a ser los mismos de todas formas.)

```
# Esto no funciona:
Temp0.Null <-
  do(999) * mean( ~ BodyTemp, data = resample(BodyTemp50))
```

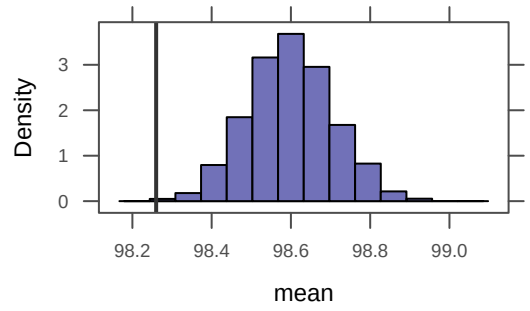
Desafortunadamente, `Temp0.Null` no es una distribución de aleatorizaciones. Inspeccionando el histograma se puede ver que la distribución no está centrada en 98.6, entonces no estamos simulando un universo donde la hipótesis nula es verdadera.

```
histogram( ~mean, data = Temp0.Null)
```



En cambio, está centrada en la media de nuestra muestra original, 98.26. Podemos cambiar la distribución con 98.6 - 98.26 = 0.34. Esto puede darnos de resultado una distribución que tiene la misma forma que la de nuestros datos con una media de 98.6, como la hipótesis nula exige.

```
Temp1.Null <- do(9999) *
  mean( ~ BodyTemp + (98.6 - 98.26), data = resample(BodyTemp50))
histogram( ~ mean, data = Temp1.Null, v = 98.26, center = 98.6)
```



Como antes, podemos estimar el valor p tabulando que tan seguido vemos un valor como 98.26.

```
2 * prop1( ~ (mean <= 98.26), data = Temp1.Null)

##   TRUE
## 0.0016
```

En este caso, el valor p es bastante pequeño – pareciera que 98.6 no es la media de la temperatura corporal.

NOTA
Usamos más replicas en este ejemplo para darnos un mejor estimado de este valor p pequeño.

De todas las distribuciones de aleatorizaciones, las distribuciones de aleatorizaciones usadas para probar hipótesis sobre un media son las más complicadas de crear, por el cambio que es necesario para centrar la distribución y el uso de `resample()` (que puede causar confusión con las distribuciones de bootstrap). Afortunadamente, crear un intervalo de confianza de una distribución de bootstrap en estas situaciones es muy directo, y generalmente preferimos intervalos de confianza que valores p en esta situación.

6.3 El Bootstrap

El método de bootstrap es utilizado (primariamente) para crear intervalos de confianza. La idea fundamental es bastante simple, además ayuda a reforzar ideas importante sobre lo que es un intervalo de confianza.

6.3.1 La idea detrás del Bootstrap

Suponga que queremos estimar la media de la temperatura usando el conjunto de datos de `BodyTemp50`. Es suficientemente simple calcular la media de nuestros datos.

```
mean( ~ BodyTemp, data = BodyTemp50)

## [1] 98.3
```

Lo que falta, es sentido de que tan precisa es la estimación. La forma más común de presentar esta información es un intervalo de confianza.

Si tuviéramos acceso a toda la población, podríamos generar muchas muestras aleatorias y observar cuanta variabilidad hay estimada de muestra a muestra (ver sección 5.8). En la práctica, nunca vamos a tener acceso a una población total (de lo contrario no necesitaríamos hacer estimaciones). La idea clave del bootstrap es tratar nuestras muestras como representaciones aproximadas de la población y generar una distribución muestral aproximada por medio del muestreo con reemplazo de nuestra muestra. La forma de la distribución de bootstrap indica qué tan precisa es *nuestra estimación*.

¡Precaución!
Hay métodos más complicados para calcular intervalos de confianza de bootstrap que tienen un mejor desempeño. Nosotros aquí introducimos dos métodos sencillos aquí. En ocasiones vamos a volver a hablar en el curso del libro sobre los intervalos-t de bootstrap.

Nota
Podemos usar métodos de bootstrap para estimar el error en nuestra estimación también

La forma de nuestra distribución de bootstrap dice que tan precisa es nuestra estimación.

Antes de proceder, hay algunas cosas importantes que hay que resaltar del proceso.

1. El re-muestreo no da una mejor estimación

   El re-muestreo es solamente usado para estimar una *variabilidad* de muestra a muestra, no es un intento de mejorar el estimado. Si intentáramos mejorar la estimación usando las muestras de bootstrap, estaríamos simplemente haciéndola peor produciendo estimados de nuestro estimado y esencialmente doblando el error en la estimación.

2. Re-muestrear funciona mejor con muestras grandes que con pequeñas.

   Las muestras pequeñas representan la población de forma poco verosímil. Si bien los métodos de re-muestreo nos pueden brindar métodos que funcionan tan bien como los tradicionales en situaciones estándar, y además pueden ser aplicados en un amplio rango de situaciones sin desempeño menor; esto no cambia el principio fundamental de tener tamaños de muestra suficiente.

3. Los dos métodos de bootstrap que presentaremos fueron elegidos por la simpleza, no por su desempeño.

   El valor que se encontró al introducir el bootstrap en cursos introductorios es pedagógico, no científica. El intervalo de percentil y error estándar son introducidos posteriormente y son accesibles a los estudiantes, además de que pueden ser aplicados a una amplia gama de situaciones. Pero estas no van a la vanguardia. En la sección 5.7 vamos a discutir un poco el intervalo t de bootstrap, un método de bootstrap más preciso. Otros métodos, como el BCa (error corregido y acelerado) o ABC (confianza aproximada de bootstrap) también mejoran el método del percentil y del error estándar, pero están fuera del alcance de la mayoría de cursos introductorios.

Paquetes como `resample` y `boot` proporcionan funciones para calcular intervalos usando métodos más sofisticados.

6.3.2 *Intervalos de confianza de bootstrap para una media*

Crear una distribución de aleatorizaciones para probar una hipótesis para una media tenía algunos retos. Afortunadamente, un intervalo de confianza es usualmente preferido en estas situaciones, y crear distribuciones de bootstrap para una media es muy directo: Simplemente calculamos la media de la temperatura corporal de muchas versiones remuestreadas de nuestros datos originales

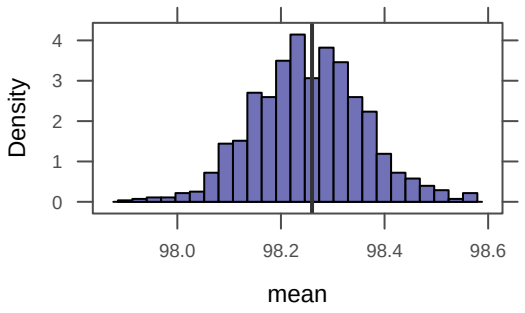
```
Temp.Boot <-
  do(1000) * mean( ~BodyTemp, data = resample(BodyTemp50))
```

Cuando es aplicado a todo el conjunto de datos, la función `resample()` muestrea con reemplazo para producir nuevos conjuntos de datos, con el mismo número de filas del original pero con algunas duplicadas y otras que no aparecen.

Idealmente, la distribución de bootstrap tiene que ser unimodal, aproximadamente simétrica, y centrada en el estimado original.

¡Precaución!
En situaciones para nada deseables, quizás tengamos que ajustar el error o usar métodos más sofisticados.Es bueno para los estudiantes tener el hábito de revisar estas características de la distribución de bootstrap antes de usar los métodos que presentamos en esta sección.

```
mean( ~ BodyTemp, data = BodyTemp50)

## [1] 98.3

mean( ~ mean, data = Temp.Boot)

## [1] 98.3

histogram( ~ mean, data = Temp.Boot, nint = 25,
           v = mean( ~ BodyTemp, data = BodyTemp50),
           c = mean( ~ BodyTemp, data = BodyTemp50)
           )
```



Para calcular un intervalo de confianza del percentil al 95 %, podemos determinar el rango central del 95 % de la distribución de bootstrap. La función `cdata()` automatiza este cálculo.

```
cdata( ~ mean, data = Temp.Boot, p = 0.95)

##       low        hi central.p
##     98.06     98.48      0.95
```

Alternativamente, `qdata()` puede ser utilizado para obtener los extremos izquierdos y derechos separadamente (o para un intervalo de confianza de solamente una cola).

```
qdata( ~ mean, data = Temp.Boot, p = 0.025)

##        p quantile
##    0.025   98.058

qdata( ~ mean, data = Temp.Boot, p = 0.975)

##        p quantile
##    0.975   98.476
```

Un segundo método simple para calcular un intervalo de confianza de una distribución de bootstrap incluye utilizar la distribución de bootstrap para estimar el error estándar.

```
SE <- sd( ~ mean, data = Temp.Boot); SE

## [1] 0.106

estimate <- mean( ~ BodyTemp, data = BodyTemp50)
estimate

## [1] 98.3

estimate + c(-1,1) * 2 * SE

## [1] 98.0 98.5
```

Este método no se desempeña tan bien como el método del percentil, pero puede servir como un buen puente para los intervalos de confianza basados en la fórmula,

incluidos usualmente en cursos que se enfocan en métodos basados en simulación. Como reemplazar la constante 2 con un valor apropiado crea intervalos de confianza más precisos o permite diferentes niveles es cosa de un poco de sutileza. El método más simple es usar los cuantiles de una distribución normal. Reemplazar la distribución con una distribución t apropiada va ensanchar los intervalos y puede mejorar la cobertura, pero la distribución t es la correcta en pocos casos - como cuando estimamos la media de una población normal - y puede cubrir mal cuando la población es distorsionada. [6]

[6] Tim C. Hesterberg. What teachers should know about the bootstrap: Resampling in the undergraduate statistics curriculum. *The American Statistician*, 2015

Como cada uno de estos métodos produce un intervalo de confianza que depende únicamente de la distribución de los estimados calculados de re-muestras, se pueden implementar fácilmente en una amplia variedad de situaciones. Calcular cualquiera de estos intervalos de confianza simples de distribuciones bootstrap puede ser automatizado usando una extensión de `confint()`

```
confint(Temp.Boot, method = c("percentile", "stderr"))

##   name lower upper level     method estimate
## 1 mean  98.1  98.5  0.95 percentile     98.3
## 2 mean  98.0  98.5  0.95     stderr     98.3
##   margin.of.error df
## 1              NA NA
## 2           0.213 49
```

Todos lo que se muestra, es la generación de la distribución del bootstrap.

6.3.3 *Intervalos de confianza de bootstrap para la diferencia de medias*

Si estamos interesados en un intervalo de confianza para la diferencia en medias de grupos, podemos usar `resample()` y `do()` para generar distribuciones de bootstrap en una de las dos formas.

```
Temp.Boot2a <-
  do(1000) * diffmean(age ~ sex, data = resample(HELPrct))
Temp.Boot2b <-
  do(1000) * diffmean(age ~ sex, data = resample(HELPrct, groups = sex))
```

En el segundo ejemplo, el re-muestreo sucede entre los grupos de sexo, de tal forma que los conteos marginales para cada sexo se mantengan fijados. Esto puede ser especialmente importante si uno de los grupos es pequeño, puesto que de otra forma los re-muestreos podrían no incluir ninguna observación en el grupo

```
favstats(age ~ sex, data = HELPrct)

##      sex min Q1 median   Q3 max mean   sd   n missing
## 1 female  21 31     35 40.5  58 36.3 7.58 107       0
## 2   male  19 30     35 40.0  60 35.5 7.75 346       0

D <- diffmean( age ~ sex, data = HELPrct); D

## diffmean
##   -0.784

favstats(age ~ sex, data = resample(HELPrct))

##      sex min Q1 median Q3 max mean   sd   n missing
## 1 female  21 30     34 39  58 35.3 7.80  98       0
## 2   male  19 30     35 39  58 35.3 7.59 355       0

favstats(age ~ sex, data = resample(HELPrct, groups = sex))

##      sex min Q1 median Q3 max mean   sd   n missing
## 1 female  21 31     37 40  58 37.2 7.93 107       0
## 2   male  19 30     35 41  60 35.7 7.95 346       0
```

NOTA

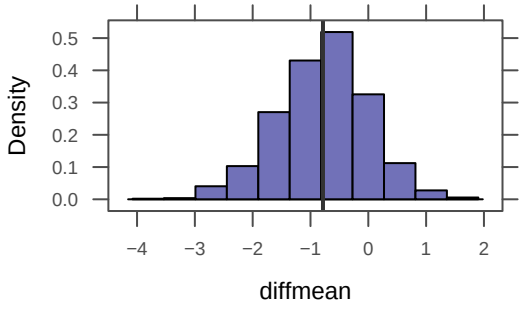
Es útil adoptar una convención en cuanto al nombramiento de la aleatorización y las distribuciones de bootstrap. Los nombres deben reflejar los datos que están siendo utilizados y si la distribución es de bootstrap o de aleatorizaciones. Usualmente usamos `.Random` o `.Null` para indicar la distribución de aleatorizaciones y `.Boot` para indicar las de bootstrap.

De aquí, el cálculo de los intervalos de confianza procede como antes.

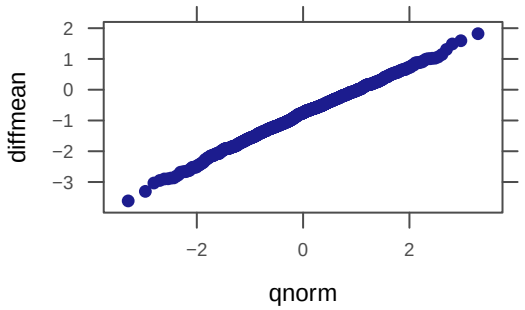
```
histogram( ~ diffmean, data = Temp.Boot2b, v = D)
```

NOTA

Inspeccionar visualmente la distribución de bootstrap por irregularidades y errores es un paso importante para asegurarse que el intervalo del percentil no está siendo aplicado en una situación en la cual se puede desempeñar pobremente.




```
qqmath( ~ diffmean, data = Temp.Boot2b)
```




```
cdata( ~ diffmean, p = 0.95, data = Temp.Boot2b)

##       low        hi central.p
##    -2.444     0.702     0.950
```

Alternativamente, podemos calcular un intervalo de confianza basado en un estimado del bootstrap del error estándar.

```
SE <- sd( ~ diffmean, data = Temp.Boot2b); SE

## [1] 0.789

D + c(-1,1) * 2 * SE

## [1] -2.362  0.794
```

Ambos intervalos pueden ser calculados usando `confint()`, si preferimos.

```
confint(Temp.Boot2b, method = c("percentile", "stderr"))

##       name lower upper level     method estimate
## 1 diffmean -2.44 0.702  0.95 percentile   -0.784
## 2 diffmean -2.34 0.758  0.95     stderr   -0.784
##   margin.of.error  df
## 1              NA  NA
## 2            1.55 452
```

6.3.4 *Comparación de distribuciones bootstrap*

Para ilustrar la similitud entre comandos usados para crear distribuciones bootstrap, aquí presentamos cinco ejemplos que pueden aparecer en cursos introductorios.

```
One.Proportion  <- do(1000) *     prop(   ~ x, data = resample(Data))
Two.Proportions <- do(1000) * diffprop( y ~ x, data = resample(Data, groups = x))
One.Mean        <- do(1000) *     mean(   ~ x, data = resample(Data))
Two.Means       <- do(1000) * diffmean( y ~ x, data = resample(Data, groups = x))
Correlation     <- do(1000) *      cor( y ~ x, data = resample(Data))
```

6.4 *Remuestreo para una regresión*

Hay al menos dos formas que podemos considerar para crear una distribución de bootstrap para un modelo lineal. Podemos facilmente ajustar un modelo lineal de un conjunto de datos re-muestreado. Pero en algunas situaciones esto puede tener efectos indeseables. Por ejemplo, observaciones de influencia pueden aparecer duplicadas en algunas re-muestras y desaparecer en otras.

Otra opción es usar el re-muestreo residual". En el re-muestreo residual, el nuevo conjunto de datos tiene todos los valores predichos de los datos originales y una nueva respuesta se crea agregando al modelo ajustado los residuales re-muestreados.

Los dos métodos son simples de implementar; podemos de igual forma re-muestrear los datos o el modelo por sí mismo.

```
mod <- lm( length ~ width + sex, data = KidsFeet)          # modelo original
do(1) * lm(length ~ width + sex, data=KidsFeet)           # Ver como do() lo trata

##   Intercept width   sexG sigma r.squared    F numdf dendf
## 1      10.4   1.6 -0.133  1.04     0.413 12.7     2    36
##   .row .index
## 1    1      1

do(2) * lm( length ~ width + sex, data = resample(KidsFeet))  # datos remuestreados

##   Intercept width   sexG sigma r.squared     F numdf dendf
## 1     11.39  1.48 -0.132 0.981     0.338  9.17     2    36
## 2      7.46  1.95 -0.338 1.005     0.485 16.92     2    36
##   .row .index
## 1    1      1
## 2    1      2

do(2) * lm( length ~ width + sex, data = resample(mod))       # residuales remuestrados

##   Intercept width    sexG sigma r.squared    F numdf dendf
## 1      5.84  2.13 -0.1113 0.849     0.642 32.3     2    36
## 2      9.20  1.74 -0.0222 0.979     0.465 15.7     2    36
##   .row .index
## 1    1      1
## 2    1      2

do(2) * relm(mod)                 # remuestreo residual abreviado

##   Intercept width    sexG sigma r.squared     F numdf dendf
## 1      13.3  1.26 -0.0696 0.913     0.355  9.92     2    36
## 2       4.5  2.24  0.0278 1.093     0.534 20.59     2    36
##   .row .index
## 1    1      1
## 2    1      2
```

De aquí, esto va dirigido a crear intervalos de confianza para la pendiente (o intercepto, o cualquier coeficiente) en un modelo lineal.

```
Kids.Boot <- do(1000) * relm(mod)
cdata( ~ width, data = Kids.Boot, p = 0.95)

##       low        hi central.p
##     0.971     2.214     0.950

confint( Kids.Boot, parm = "width")

##    name lower upper level     method estimate
```

```
## 1 width 0.971  2.21  0.95 percentile     1.44
```

6.5 ¿Qué va primero: valores p o intervalos?

TEsto es tema de discusión entre los instructores y autores de libros. Los dos libros de estadística introductoria dan respuestas diferentes. Uno[7] introduce primero la prueba de hipotesis, el otro[8] empieza con los intervalos de confianza de bootstrap. TEstos libros difieren también de algunos otros también. Permanece pendiente si la mejor práctica emerge de ciertos problemas o seguirá siendo de preferencia personal. Esto no es diferente al viejo debate de si se debe empezar por los datos cuantitativos o categóricos - otra forma en la que estos dos libros basados en simulación divergen.

[7] Nathan Tintle, Beth Chance, George Cobb, Allan Rossman, Soma Roy, Todd Swanson, and Jill VanderStoep. *Introduction to Statistical Investigations*. Wiley Global Education, 2015

[8] Robin H Lock, Patti Frazer Lock, and Kari Lock Morgan. *Statistics: Unlocking the Power of Data*. Wiley Global Education, 2012

6.6 Lidiando con la variabilidad de Monte Carlo

Puesto que la aleatorización y las distribuciones de bootstrap envuelven componentes aleatorios, valores p e intervalos de confianza calculados de los mismos datos van a variar. Para los estudiantes (y graduados), esto puede ser desconcertante porque no hay una respuesta çorrecta".

La cantidad de variabilidad de Monte Carlo depende del número de réplicas usadas para crear las distribuciones de aleatorizaciones o bootstrap. Y los estudiantes necesitan alguna guía en cuantas replicas usar. Es importante que no usen muy pocas, esto introduciría mucho ruido al valor p y el cálculo del intervalo de confianza. Pero cada replica toma tiempo, y la ganancia marginal para cada replica adicional decrece conforme el número de réplicas incrementa. No tiene mucho sentido usar millones de réplicas (a menos que la meta sea estimar valores p muy pequeños). Generalmente usamos aproximadamente 1000 por rutina o por trabajo preliminar, incrementamos esto a 10000 cuando queremos reducir los

efectos de la variabilidad de Monte Carlo..

En condiciones de laboratorio, puede ser instructivo tener que comparar los valores p o los intervalos de confianza usando 1000 y 10000 réplicas. De forma alternativa, el instructor puede generar varios valores p o intervalos de confianza para ilustrar el mismo principio.

6.7 *Mejores intervalos de confianza*

Los intervalos de confianza del percentil y del .error estándar de t con bootstrap"pueden ser mejorados en un amplio número de maneras. En un primer curso, generalmente hacemos un poco más que mencionar este hecho, y alentamos a los estudiantes a inspeccionar la forma de la distribución de bootstrap para observar indicadores de posibles problemas con el método del percentil.

Una mejoría que puede ser explicada a los estudiantes en un curso que combina los abordajes tanto de fórmula como de simulación es el intervalo de t de bootstrap. En lugar de intentar determinar la mejor cantidad de grados de libertad para la distribución t de student, la t de bootstrap aproxima la verdadera distribución de

$$t = \frac{\hat{\theta} - \theta}{SE}$$

usando la distribución de bootstrap de

$$t^* = \frac{\hat{\theta}^* - \hat{\theta}}{SE^*},$$

Donde $\hat{\theta}^*$ y SE^* son el estimado y el error estándar estimado calculado de cada distribución de bootstrap. Implementar el intervalo t de bootstrap requiere o un nivel extra de esbozo conceptual o muchos más cálculos para determinar los valores de SE^*. Si la fórmula de error estándar existe (e.g., $SE = s/\sqrt{n}$), esto puede ser aplicado a cada muestra de bootstrap simultáneo al estimador. Una alternativa es iterar el proceso de bootstrap (re-muestreando de cada re-muestreo) para estimar el SE. Puesto que el error estándar es más fácil de estimar que el intervalo de confianza, una menor cantidad de remuestreos son requeridos (por re-muestreo) en un segun-

do nivel; sin embargo, el trabajo computacional adicional es significativo.

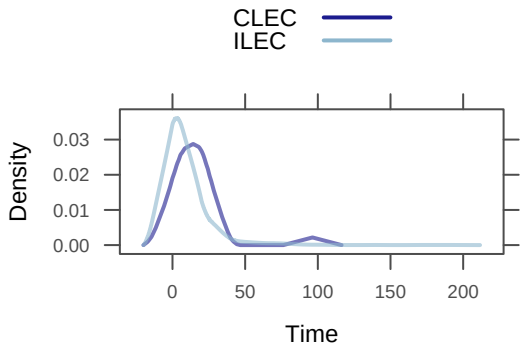
El paquete `mosaic` no intenta brindar una estructura general para la t de bootstrap u otros métodos de bootstrap Paquetes como `resample`[9] y `boot`[10] son más apropiados para situaciones en las cuales la velocidad y la precisión son de máxima importancia. Sin embargo, el intervalo de confianza de la t de bootstrap puede ser calculado usando `confint()`, `do()` y `favstats()` en caso de estimar una sola media o la diferencia entre estas medias.

En el ejemplo anterior, analizamos un conjunto de datos del paquete `resample`. El conjunto de datos `Verizon` contiene tiempos de reparación para clientes de los servicios en CLEC(competitiva) e ILEC(obligatorio).

[9] Tim Hesterberg. *resample: Resampling Functions*, 2015. R package version 0.4

[10] Angelo Canty and Brian Ripley. *boot: Bootstrap R (S-Plus) Functions*, 2015. R package version 1.3-17; and A. C. Davison and D. V. Hinkley. *Bootstrap Methods and Their Applications*. Cambridge University Press, 1997

```
# El paquete resample tiene problemas de nombres con mosaic
#Entonces sólo cargamos los datos, no el paquete

data(Verizon, package = "resample")
ILEC <- Verizon %>% filter(Group == "ILEC")
favstats( ~ Time, groups = Group, data = Verizon)

##   Group min   Q1 median    Q3   max  mean   sd    n missing
## 1  CLEC   0 5.43  14.33 20.71  96.3 16.51 19.5   23       0
## 2  ILEC   0 0.73   3.59  7.08 191.6  8.41 14.7 1664       0

 ashplot( ~ Time, groups = Group, data = Verizon,
          auto.key = TRUE, width = 20)
```

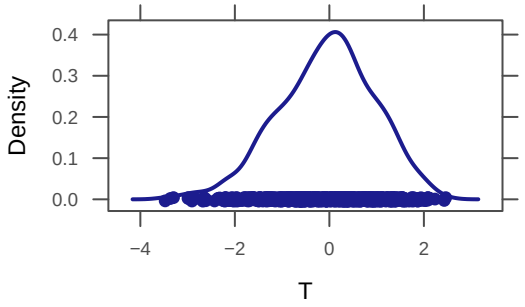


La irregularidad de las distribuciones en el tiempo de reparación y los tamaños de muestra no equivalentes resalta las diferencias entre la t de bootstrap y otros métodos más simples.

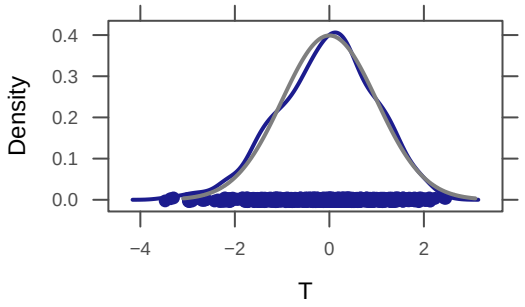
```
BootT1 <-
   do(1000) * favstats(~ Time, data = resample(ILEC))
confint(BootT1, method = "boot")

##   name lower upper level      method estimate
## 1 mean  7.76  9.19  0.95 bootstrap-t     8.41

BootT2 <-
  do(1000) * favstats( ~ Time, groups = Group,
                       data = resample(Verizon, groups = Group))
confint(BootT2, method = "boot")

##       name lower upper level      method estimate
## 1 diffmean -22.5 -2.57  0.95 bootstrap-t     -8.1
```

Esto también se puede lograr manualmente, aunque los cálculos son un poco más interesantes para el caso de 2 muestras. Aquí están los cálculos manuales para el caso de 1 muestra:

```
estimate <- mean( ~ Time, data = ILEC)
estimate

## [1] 8.41

SE <- sd( ~ mean, data = BootT1); SE

## [1] 0.361

BootT1a <-
  BootT1 %>%
  mutate( T = (mean - mean(mean)) / (sd/sqrt(n)))
q <- quantile(~ T, p = c(0.975, 0.025), data = BootT1a)
q

## 97.5%  2.5%
##  1.81 -2.15

estimate - q * SE

## 97.5%  2.5%
##  7.76  9.19

densityplot( ~ T, data = BootT1a)
```




```
plotDist("norm", add = TRUE, col="gray50")
```



Para una comparación, aquí hay intervalos producidos por el métodos de `t.test` y el método del percentil.

```
confint(t.test( ~ Time, data = ILEC))

## ~Time
##   mean of x lower upper level
## 1      8.41  7.71  9.12  0.95

BootT1b <-
  do(1000) * mean( ~ Time, data = resample(ILEC))
confint(BootT1b, method = "perc")

##   name lower upper level     method estimate
## 1 mean   7.7   9.1  0.95 percentile     8.41
```

```
confint(t.test(Time ~ Group, data = Verizon))

## Time ~ Group
##   mean in group CLEC mean in group ILEC  lower upper level
## 1               16.5               8.41 -0.362  16.6  0.95

BootT2b <-
  do(1000) * diffmean(Time ~ Group, data = resample(Verizon, groups = Group))
confint(BootT2b, method = "perc")

##       name lower upper level     method estimate
## 1 diffmean -16.6 -1.69  0.95 percentile     -8.1
```

En una situación como esta, los intervalos producidos por `t.test()` son más angostos, haga lo mínimo para compensar por las irregularidades, encubra y evite más seguido una dirección que la otra.

Aunque estos métodos no son presentados a los estudiantes, es bueno para los instructores estar por lo menos de cierta forma familiarizados con los asuntos envueltos en esto y algunos de los métodos que han sido desarrollados para manejarlos.[11]

[11] Tim C. Hesterberg. What teachers should know about the bootstrap: Resampling in the undergraduate statistics curriculum. *The American Statistician*, 2015

6.8 Simulando distribuciones muestrales

Concluimos de este capítulo que con un uso más de `sample()`. Si usamos el los datos que tenemos como una población, `sample()` puede ser usado para generar muestras aleatorias de un tamaño específico para ilustrar la idea de una distribución de muestreo. Podemos usar esto para ilustrar las distribuciones de muestreo de una media muestral, por ejemplo.

Para el ejemplo vamos a usar los datos de `NHANES`. Estos datos han sido ajustados para reflejar las ponderaciones muestrales usadas en las encuestas del American National Health and Nutrition Examination y es una buena aproximación de una muestra aleatoria simple de tamaño 10000 de la población de Estados Unidos. Para propósitos del ejemplo, vamos a tratar esto como una población total y considerar las muestras generadas de él, enfocándonos (en el momento) en la variable Age.

```
require(NHANES)
mean( ~ Age, data = NHANES)           # population mean

## [1] 36.7
```

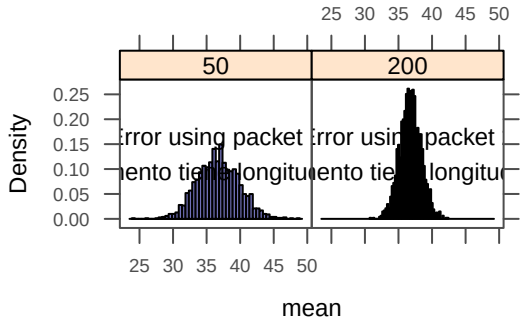
Vamos a considerar muestras de tamaños 50 y 200. Esto puede ser usado para demostrar el rol del tamaño de muestra en las distribuciones muestrales.

```
mean( ~ Age, data = sample(NHANES, 50))  # media de una muestra

## [1] 34.1

mean( ~ Age, data = sample(NHANES, 50))  # mean of another sample

## [1] 38.4
```

```
# Usamos bind_rows() para combinar dos distribuciones muestrales
# (con diferentes tamaños de muestra) en un único conjunto de datos para
# hacer los resúmenes gráficos y numéricos más sencillos.
SamplingDist <-
  bind_rows(
    do(2000) * c(mean = mean( ~ Age, data = sample(NHANES, 50)), n= 50),
    do(2000) * c(mean = mean( ~ Age, data = sample(NHANES, 200)), n= 200)
  )
```

```
mean( mean ~ n, data = SamplingDist)    # Media de la distribución muestral

##   50  200
## 36.7 36.8

sd( mean ~ n, data = SamplingDist)      # Desviación estándar de la

##   50  200
## 3.14 1.60

#distribución de muestreo
```

```
sd( ~ Age, data = NHANES) / c("50" = sqrt(50), "200" = sqrt(200))

##   50  200
## 3.17 1.58

#SE de la fórmula
histogram( ~ mean | factor(n), data = SamplingDist,
           nint = 50, density = TRUE)
```



Un abordaje similar puede ser usado para crear distribuciones de muestreo en otro tipo de situaciones

7
Lo que los estudiantes necesitan saber de R y cómo enseñarlo

En el capítulo 2, damos una pequeña orientación en el IDE RStudioy vemos qué pasa en sus viñetas y paneles . En el capítulo 4, mostramos como hacer uso de una notación común de resúmenes gráficos, resúmenes numéricos, y modelaje. En este capítulo cubrimos algunos detalles adicionales que son importantes para que los estudiantes sepan sobre el lenguaje de R.

Más Info

Asegúrese de ojear *Una guía de estudiantes para R*, también. Este pequeño libro contiene un resumen corto de todos los comandos necesarios para el análisis estadístico típicamente visto en los dos primeros cursos de estadística.

7.1 Dos preguntas

Cuando introdujimos la notación de fórmula en el capítulo 4, presentamos dos preguntas importantes que se deben hacer antes de construir un comando de R. Estas preguntas son útiles en contextos más allá de la notación o plantilla de fórmula, y de hecho para otros sistemas computacionales más allá de R, entonces los repetimos aquí.

1. ¿Qué quiere que haga R?

   Esto usualmente va determinar cuál función de R usar.

2. ¿Qué tiene que saber R para hacerlo?

   Esto va determinar las entradas de la función.

Cuando sus estudiantes formulen sus preguntas sobre R en lo que quieren que R haga y qué necesita saber R para hacerlo, entonces sabrá que estas preguntas fueron interiorizadas.

Consejo de Enseñanza

Cuando los estudiantes tienen dificultad logrando una tarea en R, asegúrese que pueden responder estas preguntas antes de mostrarles qué hacer. Si no pueden contestarlas, entonces el problema no es primariamente con R. Si hace esto consistentemente, eventualmente, va encontrar a sus estudiantes presentado sus preguntas de Rcontestando estas dos preguntas y después preguntando "¿Entonces, cómo logro que R haga eso?"Probablemente después de contestar esas dos preguntas, ya van a saber cómo hacer que Rhaga lo que quieren - a menos que estén preguntando la funcionalidad de algo que aún no haya presentado.

7.2 *Cuatro cosas que saber sobre* R

Como en casi todos los lenguajes computacionales, R debe usarse en sus términos. R no aprende la personalidad y el estilo de sus usuarios. Llevarse con R es mucho más sencillo si usted tiene en mente (y le recuerda a sus estudiantes) algunas características claves del lenguaje R.

CONSEJO DE ENSEÑANZA
A algunos estudiantes se les dificulta entender la importancia de las mayúsculas. Así que tal vez tenga que recordarlo unas cuantas veces más.

1. R es sensible a casos

   Si usted pone una mayúscula de más en algo en R no va hacer lo que usted deseaba. Desafortunadamente no hay una convención sobre como las mayúsculas deben ser usadas, entonces debe ponerle atención cuando se encuentre con nuevas funciones y conjuntos de datos

2. Las funciones de R usan la siguiente sintaxis:

   ```
   functionname( argument1, argument2, ... )
   ```

   - Los argumentos están siempre rodeados por *paréntesis (redondos)* y *separados por comas*.
     Algunas funciones (como `data()`) no requieren argumentos, pero igual necesitan paréntesis.
   - Si escribe el nombre de una función sin paréntesis, va ver el *código* para esa función (esto generalmente no es lo que se quiere, a menos que sea curioso y quiera saber cómo es implementado).

CONSEJO DE ENSEÑANZA
Introduzca las funciones enfatizando las preguntas *¿Qué queremos que haga la computadora?* y *¿Qué información necesita la computadora para hacer esto?* La respuesta a la primera pregunta determina qué función usar. La respuesta a la segunda determina cuáles argumentos deben especificarse.

3. El relleno de línea de código y las flechas pueden mejorar la velocidad de la escritura y la precisión.

   Si empieza un comando y después le da la tecla enter, R y RStudiole van a presentar una lista de posibles formas de completar el comando. Si oprime ENTER después de abrir un paréntesis de una función, RStudiole va desplegar la lista de argumentos que espera.

   Las flechas lo pueden ayudar a encontrar comandos pasados cuando se trabaja en la consola.

4. Si ve una entrada con +, significa que Restá pidiendo más entrada o código.

   Esto puede significar que hace falta cerrar el paréntesis o que se ha hecho algún error de sintaxis. Si tuvo algún problema y solo quiere la salida normal, presione la tecla de escape para empezar el comando de nuevo.

PRECAUCIÓN!
Sus estudiantes en ocasiones van a encontrarse en un hueco sintáctico del que no pueden salir. Enséñeles a presiones la tecla ESC a tiempo.

7.3 Instalando y usando paquetes

R es un software libre. Su desarrollo es respaldado por un grupo base de desarrolladores y una gran comunidad de usuarios. Una forma mediante la cual los usuarios apoyan R es compartiendo paquetes que contienen datos y funciones para una amplia variedad de tareas. Como instructor, va querer elegir una serie de paquetes que apoyen la forma en la que da su curso.

Si necesita instalar un paquete, es muy probable que esté en CRAN, the Comprehensive R Archive Network. Antes de usar un paquete, lo debe **instalar** (una vez por computadora o cuenta de usuario) y **cargar** (una vez por sesión de R). El proceso de instalación descarga el paquete y prepara para compilar (si es necesario) y pone los componentes en la dirección adecuada para uso futuro. Cargar hace los paquetes previamente instalados disponibles para usar en la sesión de R.

Por ejemplo, para usar el paquete mosaic, primero debemos instalarlo:

Consejo de Enseñanza

Si ajusta RStudiocon el servidor, puede instalar todos los paquetes que quiera usar. Puede incluso configurar el servidor para auto cargar paquetes que usa frecuentemente. Los estudiantes que usen R en sus propias computadoras van a necesitar saber cómo instalar y cargar estos paquetes.

```
install.packages("mosaic")
# Obtener el paquete del CRAN
```

Una vez que el paquete ha sido instalado, debe ser *cargado* para que esté disponible en la sesión o creación del documento:

```
library(mosaic)
# Una alternativa de cargarlo
library(mosaicData)
# Carga los conjuntos de datos
```

or

```
require(mosaic)
# Otra alternativa para cargarlo
require(mosaicData)
# Carga los conjuntos de datos
```

Consejo de Enseñanza

Usar `library()` es más común en estas situaciones, pero nos dimos cuenta que los estudiantes recuerdan mejor la palabra `require()`. Para su propósito, ambos hacen esencialmente lo mismo. La mayor diferencia es como responde cuando un paquete no puede cargarse (usualmente porque este no está instalado). require() genera un mensaje de alerta y devuelve un valor lógico que se puede usar al programar. library() genera un error cuando el paquete no puede ser cargado.

La viñeta de paquetes en RStudio vuelve instalar y cargar paquetes particularmente fácil y evita la necesidad de usar `install.packages()` para paquetes en CRAN,

Más Info

Aunque el comando se llama `library()`(librería), lo que se carga es un paquete no una librería.

Precaución!

Recuerde que en los archivos Rnw y RMarkdown, cualquier paquete que use tiene que ser cargado *dentro del archivo*.

además hace cargar paquetes en la consola tan fácil como seleccionar de un menú. Las funciones `require()` y `library()` son necesarias incluso en `RMarkdown`, `knitr`/LaTeXy archivos de script.

Si está corriendo en una máquina en la cual no tiene los privilegios para escribir la dirección predeterminada de librería, puede instalar una copia personal del paquete. Si la dirección de su librería personal está primero en `R_Libs`, esto probablemente pase automáticamente. De lo contrario, puede especificar la dirección manualmente:

```
install.packages("mosaic", lib="~/R/library")
```

CRAN no es el único repositorio de paquetes de R. Bioconductor es otro gran y popular repositorio, especialmente para aplicaciones biológicas y ha incrementado la cantidad de autores que hacen los paquetes disponibles via github. Por ejemplo, puede también instalar el paquete mosaic usando

```
# Si no ha instalado devtools
install.packages("devtools")
require(devtools)
install_github("ProjectMOSAIC/mosaic")
```

Ocasionalmente se puede encontrar un paquete de interés que no está disponible en algún repositorio, puede encontrar instrucciones de cómo instalarlo. Si no, puede usualmente instalarlo directamente de un archivo zip.

```
# repos = NULL indica que utilice el archivo, no el repositorio
install.packages('some-package.tar.gz', repos=NULL)
```

> De aquí en adelante vamos a asumir que el paquete `mosaic` ha sido instalado y cargado

7.4 Consiguiendo ayuda

Si algo no sale bien, o si no puede recordar algo, es bueno saber a dónde buscar por ayuda. Además de preguntar a sus amigos y vecinos, puede usar el sistema de ayuda de R.

7.4.1 ?

Para conseguir ayuda de una función específica o de un conjunto de datos, simplemente escriba ? antes de su nombre:

```
?log      # Ayuda para la función log
```

```
?HELPrct  # Ayuda para un conjunto de datos en el paquete mosaic.
```

Esto le va mostrar la documentación del objeto en el cual está interesado.

7.4.2 apropos()

Si no sabe exactamente el nombre de una función, puede dar parte del nombre y R va encontrar funciones que calcen. Las comillas son necesarias para esto.

```
apropos('tally')

## [1] "add_tally"  "add_tally_" "statTally"  "tally"
## [5] "tally"      "tally_"

# Debe incluir comillas solas o dobles
```

Más Info

Note que `tally` aparece dos veces. Esto es porque hay 2 funciones `tally()`, una en el paquete `mosaic` y otra en el paquete `dplyr`. La función `find()` puede ser utilizada para determinar a qué paquetes pertenece una función. En ese caso, el paquete mosaic se encarga de navegar entre las dos versiones de `tally()`. En otros casos, tal vez necesite especificar la función del paquete que quiere.

7.4.3 ?? y help.search()

Si lo anterior falla, puede darle un uso más amplio a ?? o `help.search()`, que va encontrar parejas no solo entre los nombres de funciones y conjuntos de datos, sino también en la documentación para estos. Las comillas son opcionales aquí.

7.4.4 Ejemplos y demos

Muchas funciones y conjuntos de datos incluyen código de ejemplos demostrando su uso típico. Por ejemplo,

No todos los autores de paquetes son igual de habilidosos creando ejemplos. Algunos ejemplos no son existentes o inútiles, otros son excelentes

```
example(histogram)
```

va generar unos cuantos gráficos de ejemplos (y proporcionar los comandos utilizados para crearlos). Ejemplos como este tienen la intención de ayudar qué tan específico funcionan las funciones de R. Estos ejemplos también aparecen al final de la documentación para funciones o conjuntos de datos.

El paquete `mosaic` (y otros paquetes también) también incluye demos. Los demos son pedazos de código de R que puede ser ejecutado usando el comando `demo()` con el nombre del demo. Para ver cómo funcionan los demos, intente esto:

```
demo(lattice)
```

Los demos tienen la inteción de ilustrar un concepto o un método y son independientes de cualquier función particular o conjunto de datos.

Puede también conseguir una lista de demos disponibles usando

```
demo()                    # todos los demos
demo(package='mosaic')
# solo los demos del paquete mosaic
```

7.5 Data

7.5.1 Data Frames

Los conjuntos de datos usualmente están almacenados en una estructura llamada **data frame**.

Consejo de Enseñanza
Los estudiantes que recolectan sus propios datos, especialmente si los almacenan en Excel, es poco probable que pongan los datos en un formato correcto a menos que se les enseñe explícitamente a hacerlo.

Los data frames son estructuras de bi-dimensionales.

- Filas, que corresponden a **unidades de observación** (personas, animales, plantas u otros objetos de los cuales estamos recolectando datos).
- Columnas que corresponden a **variables** (medidas recolectadas en cada unidad de observación).

El data frame de `Births78` contiene cuatro variables medidas para cada día en 1978. Hay varias formas en las que podemos ver cómo es el data frame de Births78.

CONSEJO DE ENSEÑANZA
Para ayudar a los estudiantes a mantener las variables y los data frames data frames claros, adoptamos una convención de que los data frames en el paquete `mosaicData` están en mayúscula y las variables (usualmente) no lo están. Esta convención ha funcionado bien, y quizás lo quiera adoptar en sus conjuntos de datos también.

```
head(Births78)            #Muestra las primeras filas

##         date births dayofyear  wday
## 1 1978-01-01   7701         1   Sun
## 2 1978-01-02   7527         2   Mon
## 3 1978-01-03   8825         3  Tues
## 4 1978-01-04   8859         4   Wed
## 5 1978-01-05   9043         5 Thurs
## 6 1978-01-06   9208         6   Fri
```

```
sample(Births78, 4)       #Muestra 4 filas seleccionadas aleatoriamente

##           date births dayofyear  wday orig.id
## 105 1978-04-15   7527       105   Sat     105
## 287 1978-10-14   8554       287   Sat     287
## 149 1978-05-29   7780       149   Mon     149
## 320 1978-11-16   9568       320 Thurs     320
```

```
summary(Births78)

##       date                births        dayofyear
##  Min.   :1978-01-01   Min.   : 7135   Min.   :  1
##  1st Qu.:1978-04-02   1st Qu.: 8554   1st Qu.: 92
##  Median :1978-07-02   Median : 9218   Median :183
##  Mean   :1978-07-02   Mean   : 9132   Mean   :183
##  3rd Qu.:1978-10-01   3rd Qu.: 9705   3rd Qu.:274
##  Max.   :1978-12-31   Max.   :10711   Max.   :365
##
##     wday
##  Sun  :53
##  Mon  :52
##  Tues :52
##  Wed  :52
##  Thurs:52
##  Fri  :52
##  Sat  :52

#Proporciona resumenes de información sobre cada variable
```

```
inspect(Births78)     #Brinda un resumen de la información de cada variable

##
## categorical variables:
##   name   class levels   n missing                                   distribution
## 1 wday ordered      7 365       0 Sun (14.5%), Mon (14.2%), Tues (14.2%) ...
##
```

```
## quantitative variables:
##        name   class  min   Q1 median   Q3   max mean  sd   n missing
## 1    births integer 7135 8554   9218 9705 10711 9132 818 365       0
## 2 dayofyear integer    1   92    183  274   365  183 106 365       0
##
## time variables:
##   name   class      first       last min_diff max_diff   n missing
## 1 date POSIXct 1978-01-01 1978-12-31   1 days   1 days 365       0
```

```
str(Births78)             #Muestra la estructura de cualquier objeto R

## 'data.frame': 365 obs. of  4 variables:
##  $ date     : POSIXct, format: "1978-01-01" ...
##  $ births   : int  7701 7527 8825 8859 9043 9208 8084 7611 9172 9089 ...
##  $ dayofyear: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ wday     : Ord.factor w/ 7 levels "Sun"<"Mon"<"Tues"<..: 1 2 3 4 5 6 7 1 2 3 ...
```

La salida de `str()` está también disponible en la viñeta del entorno.

De forma interactiva, también puede intentar

```
?Births78
```

para acceder la documentación del conjunto de datos. Esto también está disponible en la viñeta de Help. Finalmente la viñeta de Environment proporciona una lista de datos en el entorno global. Presionar un conjunto de datos saca un visor como

```
View(Births78)
```

Podemos ganar acceso a una única variable en el conjunto de datos usando el operador `$`, o alternativamente, usando `with()`

```
dataframe$variable
with(dataframe, variable)
```

Por ejemplo,

```
Births78$births
with(Births78, births)
```

va mostrar contenidos de la variable `births` en el conjunto de datos `Births78`.

Como vamos a ver, hay relativamente pocas instancias en las cuales se necesita el operador `$`.

Listar todo el conjunto de valores para una variable particular no tiene mucho uso en conjuntos de datos grandes. Preferimos calcular resúmenes gráficos o numéricos. Pronto lo haremos.

7.5.2 *Los peligros de `attach()`.*

La función `attach()` en R puede ser usada para hacer que objetos en data frames sean accesibles en R sin tanta escritura, pero nosotros desaprobamos su uso, usualmente guía a conflictos de nombres y otras complicaciones. A Google R Style Guide[1] lo afirma

> *Las posibilidades de crear errores al usar `attach()` son numerosas. Evítelas.*

¡Precaución!
Evite el uso de `attach()`.

[1] `http://google-styleguide.googlecode.com/svn/trunk/google-r-style.html`

7.5.3 *Los datos en los paquetes*

Los conjuntos de datos en los paquetes de R son los más sencillos de trabajar. Los conjuntos de datos en los paquetes de R son los más sencillos de trabajar. En la sección 6.5.4, vamos a describir como cargar sus propios datos en R y RStudio pero le recomendamos empezar con datos de los paquetes, y eso es lo que vamos a hacer aquí, también.

Consejo de Enseñanza
Empiece usando datos en paquetes y después enseñándole a los estudiantes como importar sus propios datos una vez que hayan entendido como trabajar con datos.

Muchos paquetes contienen conjuntos de datos. Usted puede ver una lista de todos los conjuntos de datos en los paquetes cargados usando

```
data()
```

Puede opcionalmente elegir restringir a la lista de un sólo paquete:

```
data(package="mosaic")
```

Típicamente se pueden usar conjuntos de datos simplemente escribiendo sus nombres. Pero si ya ha usado ese nombre para algo o necesita refrescar los datos después de algunos cambios que ya no necesita, puede explícitamente cargarlos usando la función `data()`, con el nombre del conjunto de datos que quiere utilizar.

Más Info
Esto depende del paquete. La mayoría de autores ajustan sus paquetes con un "lazy loading"de datos. Si no, entonces tiene que usar `data()` explícitamente.

```
data(Births78)
```

No hay efecto visible de este comando, pero `Births78` ahora acaba de ser re-cargado del paquete `mosaicData` y está listo para utilizarse. Cualquier cosa que haya hecho anteriormente en algo con un nombre del conjunto de datos es reemplazada por la versión del paquete `mosaicData`.

¡Precaución!
Si dos paquetes incluyen conjuntos de datos con el mismo nombre, hay que especificar cual paquete es del que quiere los datos `data(Births78, package=mosaicData)`.

7.5.4 Usando sus propios datos

Eventualmente los estudiantes van a querer dejar de usar datos de ejemplos en paquetes de R y usar datos que han encontrado y recolectado ellos. Cuando esto pase, depende del tipo de estudiantes que tenga y el tipo de curso que esté enseñando.

Consejo de Enseñanza
Empiece usando datos de paquetes y enfóquese en lo que R puede hacer con los datos. Después, una vez que los estudiantes estén familiarizados con R y entiendan el formato requerido de los datos, enséñeles a importar sus propios datos.

R proporciona las funciones `read.csv()` (Para archivos separados por coma), read.table() (para archivos delimitados por un espacio en blanco) y load() (para cargar datos en el formato de R). El paquete mosaic incluye una función llamada read.file() que usa unos ajustes predeterminados un poco distintos e infiere si se debe usar read.csv(), read.table() o load() basado en el nombre del archivo.

Puesto que la mayoría de softwares pueden exportar de formato csv, esto se ha vuelto una *lingua franca* para moverse en diferentes softwares. Los datos en excel, por ejemplo, pueden ser exportados como archivos csv y subsecuentemente ser leídos en R. Hay un peligro en hacer esto, sin embargo, algunos tipos de datos no se exportan de Excel cómo se espera. Una forma más segura de leer archivos excel es usar `read_excel()` una función del paquete `readxl`. El paquete `haven` incluye funciones y utilidades para leer datos en otros tipos de formato que son exportados de paquetes estadísticos como SAS o Stata.

Algunas de estas funciones que ingieren datos aceptan una URL como nombre de archivo, lo cual brinda una forma fácil de distribuir datos via Internet:

```
births <-
  read.table('http://www.calvin.edu/~rpruim/data/births.txt', header=TRUE)
head(births) #Nacidos vivos en el año 78 en USA.

##     date births datenum dayofyear
## 1 1/1/78   7701    6575         1
## 2 1/2/78   7527    6576         2
## 3 1/3/78   8825    6577         3
## 4 1/4/78   8859    6578         4
## 5 1/5/78   9043    6579         5
## 6 1/6/78   9208    6580         6
```

Podemos omitir el `header=TRUE` si usamos `read.file()`.

```
births <-
  read.file('http://www.calvin.edu/~rpruim/data/births.txt')

## Reading data with read.table()
```

7.5.5 *Importando datos de* RStudio

La interface de RStudio proporciona algunas herramientas GUI para cargar datos. Si está usando el servidor de RStudio primero necesita cargar los datos al servidor (en la viñeta de Files), y después importar los datos en la sesión de R (en la viñeta de Environment)

Si está utilizando la versión de escritorio, subir el archivo no es necesario.

CONSEJO DE ENSEÑANZA
Recuérdele a los estudiantes que el proceso de 2 pasos (subir e importar) funciona como las imágenes en Facebook. Primero la sube a Facebook, y después están ahí y se pueden incluir en publicaciones, etc.

7.5.6 *Trabajando con datos pretabulados*

Puesto que los datos categóricos son tan sencillos de resumir en tablas, usualmente las tablas de frecuencia y contingencia se nos dan en lugar de todos los datos. Se pueden ingresar estas tablas manualmente usando una combinación de `c()`, `rbind()` y `cbind()`.

Incluso si usa la GUI DE RStudio para el trabajo interactivo, va necesitar saber cómo usar funciones como `read.csv()` para trabajar en archivos de tipo RMarkdown, o `knitr`/LaTeX.

```
myrace <- c( NW=67, W=467 )
#c es de combinar o concatenar
myrace

##  NW   W
##  67 467
```

CONSEJO DE ENSEÑANZA
Esto es una técnica importante si usa un libro de texto que presenta datos categóricos pretabulados

```
mycrosstable <- rbind(
  NW = c(clerical=15, const=3, manag=6,  manuf=11,
                 other=5,  prof=7, sales=3, service=17),
  W  = c(82,17,49,57,63,98,35,66)
                          )
mycrosstable
```

```
##    clerical const manag manuf other prof sales service
## NW       15     3     6    11     5    7     3      17
## W        82    17    49    57    63   98    35      66
```

Reemplazar `rbind()` con `cbind()` le va permitir dar los datos en forma de columna

Este arreglo de datos es suficiente para aplicar prueba de ji-cuadrado, pero no lo es para aplicar un formato de gráfico `lattice`. Nuestra tabla cruzada aún le falta información- los nombres de las variables que están siendo guardadas. Podemos agregar esta información si la convertimos en una tabla.

Consejo de Enseñanza

Si graficar datos categóricos pre-tabulados es importante, probablemente sólo quiera entregarles a los estudiantes una función que cubra y simplifique todo esto. Generalmente evitamos esta situación, entregando los datos en crudo o presentando un análisis de datos en tabla usando resúmenes gráficos.

```
class(mycrosstable)
```

```
## [1] "matrix"
```

```
mycrosstable <- as.table(mycrosstable)
```

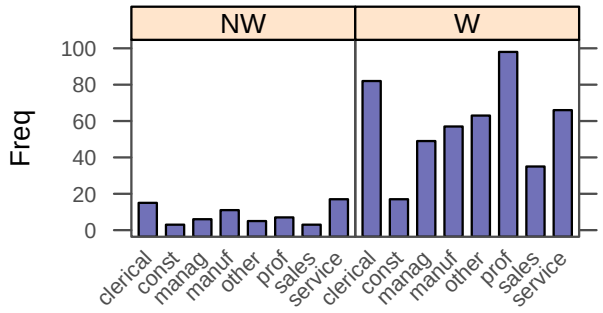
```
#mycrosstable ahora tiene dimnames, pero no tienen nombre
dimnames(mycrosstable)

## [[1]]
## [1] "NW" "W"
##
## [[2]]
## [1] "clerical" "const"    "manag"    "manuf"    "other"
## [6] "prof"     "sales"    "service"
```

```
# Agreguemos dimnames con sentido
names(dimnames(mycrosstable)) <- c('race', 'sector')
mycrosstable

##     sector
## race clerical const manag manuf other prof sales service
##   NW       15     3     6    11     5    7     3      17
##   W        82    17    49    57    63   98    35      66
```

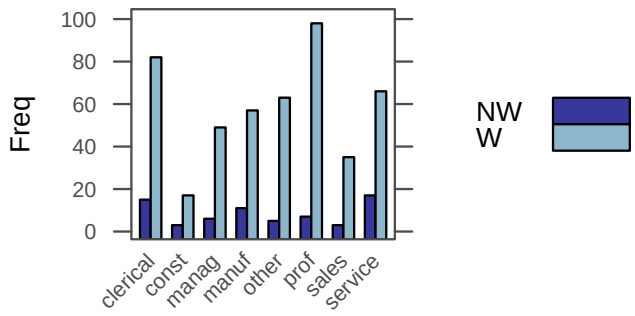
Podemos usar `barchart()` en lugar de `bargraph()` para graficar datos ya tabulados, pero primer necesitamos una transformación más.

```
head(as.data.frame(mycrosstable))

##   race   sector Freq
## 1   NW clerical   15
## 2    W clerical   82
## 3   NW    const    3
## 4    W    const   17
## 5   NW    manag    6
## 6    W    manag   49
```

```
barchart( Freq ~ sector | race,
          data=as.data.frame(mycrosstable),
          auto.key=list(space='right'),
          scales=list(x=list(rot=45))
                 )
```




```
barchart( Freq ~ sector, groups=race,
          data=as.data.frame(mycrosstable),
          auto.key=list(space='right'),
          scales=list(x=list(rot=45))
                 )
```



7.5.7 Desarrollando buenos habitos con los datos

Aunque usted enseñe a los estudiantes como recolectar e importar sus datos, los estudiantes tienen que estar entrenados para seguir buenas prácticas de organización de datos:

1. Elegir buenos nombres para las variables
2. Poner los nombres de las variables en la primera fila
3. Usar una fila subsecuente para cada unidad observacional.
4. Dar al conjunto de datos resultante un buen nombre

Algunos científicos pueden sentirse decepcionados de que los conjuntos de datos R no mantienen rastros de información adicional, como las unidades en las que las observaciones fueron guardadas. Este tipo de información debería ser guardada, junto con la descripción de los protocolos usados para recolectar estos datos, las observaciones hechas durante el proceso de guardado, etc. Esta información debe ser mantenida en un libro de códigos o una libreta de laboratorio.

7.6 Repaso de comandos de R

Aquí hay un pequeño repaso de los comandos introducidos en este capítulo

```
require(mosaic)                  # Carga el paquete mosaic
require(mosaicData)
# carga los conjuntos de datos de mosaic
answer <- 42
# Guarda el número 42 en el objeto answer
log(123); log10(123); sqrt(123)
# Algunas funciones numéricas estándar
x <- c(1,2,3)
# Generar un vector que contiene 1, 2 y 3

data(iris)
# carga de nuevo el conjunto de datos iris
names(iris)
# ver los nombres de iris
head(iris)
# las primeras filas del conjunto iris
sample(iris, 3)
# 3 filas aleatorias de iris
inspect(iris)
# resume cada variable del conjunto de datos iris
summary(iris)
# resume cada variable del conjunto de datos
str(iris)
# Muestra la estructura del conjunto de datos

mydata <- read.table("file.txt")
# Lee los datos de un archivo de texto
mydata <- read.csv("file.csv")
# lee los datos de un csv
mydata <- read.file("file.txt")
# lee los datos de un csv o un archivo de texto
require(readxl)
mydata <- read_excel("file.xlsx")
# lee los datos de un archvivo excel
```

7.7 *Ejercicios*

7.1. La tabla presentada a continuación es de un estudio de la luz nocturna durante la infancia y la visión (más tarde en su vida).

	sin miopía	miopía	miopía alta
oscuridad	155	15	2
luz nocturna	153	72	7
luz total	34	36	3

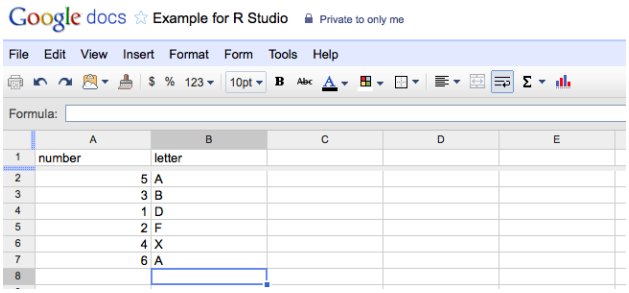
a) Recree la tabla en R.

b) ¿Qué porcentaje de los sujetos durmieron con luz de noche en su infancia?

Hay varias formas de hacer esto. Puede usar R como una calculadora y hacerlo aritméticamente. Puede evitar esto usando la función `tally()`. Vea `?tally` para documentación

c) Haga una representación gráfica de estos datos. ¿Qué revela?

7.2. Ingrese un pequeño conjunto de datos de una hoja de cálculo de Excel o Google e impórtelo a RStudio.



8
Lo que los instructores necesitan saber de R

Recomendamos mantener la cantidad de R que los estudiantes deben saber al mínimo, y elegir funciones que utilicen y apoyen la notación de fórmula cuando sea posible, para mantener las funciones con sintaxis similares. Sin embargo, hay algunas cosas adicionales que los instructores (y tal vez algunos estudiantes) deben saber de R. Resaltamos algunas en este capítulo.

Es posible que piense que algunas de estas cosas son útiles para que las conozcan sus estudiantes. Eso va depender de las metas de su curso y las habilidades de los estudiantes. En los cursos de mayor nivel, mucho del material en este capítulo es apropiado para estudiantes..

8.1 Algunas recomendaciones del ciclo de trabajo

Nuestro consejo de flujo de trabajo puede ser resumido en una pequeña oración:

Piense como un programador.

No requiere habilidades de programación muy sofisticadas para ser bueno usando R. De hecho, la mayoría de los usos de R para enseñar estadística pueden hacerse un paso a la vez, donde cada línea de código hace una útil tarea completa. Después de inspeccionar la salida (y guardarlo para posibles cálculos o computaciones después), uno puede proceder a la próxima operación

No pensamos que nuestra clase use R para programación puesto que usamos R mayormente de forma declarativa, no algorítmica

Sin embargo, podemos pedir prestado del conocimiento colectivo de la comunidad programadora y adoptar algunas prácticas que van a hacer nuestra experiencia más placentera, más eficiente y con menor probabilidad de errores.

- Guarde su código en un documento.

Puede ser tentador hacer todo en la consola. Pero esta es efímera. Es mejor tener el hábito de guardar archivos con el código. Haga el hábito (y haga que sus estudiantes tengan ese hábito) de trabajar con scripts de R y especialmente archivos de RMarkdown..

Puede correr todo el código de R usando

```
source("file.R")
```

RStudio tiene opciones adicionales para ejecutar todas las líneas en un documento. Vea los botones en la viñeta para cualquier archivo R script, RMarkdown o Rnw. (Puede crear un archivo en el menú principal del menú File)

Si usted trabaja en la entrada de la consola interactiva y ha estado poniendo comandos en un archivo, puede guardarlos con:

```
savehistory("someRCommandsIalmostLost.R")
```

RStudio Puede selectivamente copiar porciones de su historia en un archivo de script (o la consola) usando la viñeta de History.

Más Info

R puede ser utilizado para crear scripts ejecutables. Opciones para análisis y manejo es apoyado en el paquete `optparse`.

- Use nombres con significado.

  Raramente los objetos deberían ser llamados con una sola letra.

  Adopte una convención personal respecto a esto en caso de letras. Esto significa que tiene una cosa menos que recordar cuando intente llamar un objeto. Por ejemplo, en el paquete `mosaicData`, todos los conjuntos de datos empiezan con mayúscula. La mayoría de las variables empiezan con una no mayúscula (hay algunas excepciones para variables con nombres que se sabe por qué están en mayúscula).

- Adopte modismos reutilizables

  Los programadores se refieren a pequeños patrones que recurren alrededor de su código como modismos. Por ejemplo, aquí está el çompute, guarde, despliegueï-diom.

```
# compute, guarde y despliegue
footModel <- lm(length ~ width, data=KidsFeet); footModel

##
## Call:
## lm(formula = length ~ width, data = KidsFeet)
```

```
##
## Coefficients:
## (Intercept)        width
##        9.82         1.66
```

```
# Alternativa que refleja el orden de las operaciones
lm(length ~ width, data=KidsFeet) -> footModel; footModel

##
## Call:
## lm(formula = length ~ width, data = KidsFeet)
##
## Coefficients:
## (Intercept)        width
##        9.82         1.66
```

Usualmente hay múltiples maneras de hacer lo mismo en R, pero si adopta buenos modismos de programación, va ser más claro para usted y sus estdiantes qué estaba haciendo

- Use funciones reutilizables

  Aprender a escribir sus propias funciones (ver sección 7.7) va incrementar su eficiencia y también lo va a ayudar a entender mejor como R funciona. Esto, de regreso, le va a ayudar a solucionar los mensajes de error de su estudiantes.(Más mensajes de error en 7.10). También hace posible simplificar tareas que quiere que los estudiantes hagan en R. Así es como el paquete mosaic es originado – como una colección de herramientas que hemos ensamblados durante el tiempo para hacer la enseñanza y el aprendizaje más sencillo.

- Comente su código.

  Es increíble lo que uno puede olvidar. El signo de comentario en R es #. Si está trabajando en RMarkdown o Rnw, puede incluir texto en un buen formato para describir lo qué está haciendo y por qué.

8.2 *Estructuras de datos de* R *primarias*

Todo en R es un objeto de una clase particular y entendiendo los tipos de objetos que R usa le quita el misterio a bastantes mensajes que R produce, además de comportamientos inesperados de comandos que no funcionan de la manera que se esperaba. No vamos a intentar darle una descripción exhaustiva de la taxonomía de objetos de R, pero en su lugar, nos vamos a enfocar en unas cuantas características importantes y ejemplos.

8.2.1 *Objetos y clases*

En R, los datos son guardados en **objetos**. Cada objeto tiene un *nombre, contenido y una clase*. La clase de un objeto nos dice qué tipo de cosa es. La clase de un objeto puede ser pedida usando `class()`

Más Info
Muchos objetos también tienen *atributos* que contienen información adicional del objeto, pero a menos que esté programando con estos objetos, no se tiene que preocupar mucho por estos.

```
class(KidsFeet)

## [1] "data.frame"

class(KidsFeet$birthmonth)

## [1] "integer"

class(KidsFeet$length)

## [1] "numeric"

class(KidsFeet$sex)

## [1] "factor"

str(KidsFeet)

## 'data.frame': 39 obs. of  8 variables:
##  $ name      : Factor w/ 36 levels "Abby","Alisha",..: 10 24 36 20 23 34 13 4 14 8 ...
##  $ birthmonth: int  5 10 12 1 2 3 2 6 5 9 ...
##  $ birthyear : int  88 87 87 88 88 88 88 88 88 88 ...
##  $ length    : num  24.4 25.4 24.5 25.2 25.1 25.7 26.1 23 23.6 22.9 ...
##  $ width     : num  8.4 8.8 9.7 9.8 8.9 9.7 9.6 8.8 9.3 8.8 ...
##  $ sex       : Factor w/ 2 levels "B","G": 1 1 1 1 1 1 1 2 2 1 ...
##  $ biggerfoot: Factor w/ 2 levels "L","R": 1 1 2 1 1 2 1 1 2 2 ...
##  $ domhand   : Factor w/ 2 levels "L","R": 2 1 2 2 2 2 2 2 2 1 ...

#Muestra la clase para cada variable
```

De aquí, vemos que KidsFeet es un conjunto de datos con variables de diferentes tipos (enteros, numéricas y factores). Estos tipos de variables es posible que las encuentre, aunque puede ver variables lógicas(TRUE o FALSE) o de tipo character(texto) también.

Los factores son la forma más común de datos categóricos que se puede almacenar en R, pero a veces la clase de texto (character) puede ser mejor. La clase de un objeto determina qué cosas pueden hacerse y cómo aparece cuando es impreso, graficado o desplegado en la consola..

Más Info
Una diferencia entre un factor y una variable de texto es que el factor sabe los posibles valores, incluso si algunos de ellos no ocurren. A veces esto es una ventaja (para tabular celdas vacías en una tabla) y a veces una desventaja (cuando los factores son utilizados como identificadores únicos)

8.2.2 *Containers*

La situación es actualmente un poco más complicada. La variable `birthmonth` en `KidsFeet` no es un único entero, es una colección de enteros Hay más de un tipo de contenedor en R. Los contenedores usados para variables en el conjunto de datos se llaman **vectores**. Estas cosas dentro de un vector están ordenadas (empezando con 1) y deben ser del mismo tipo.

Los vectores pueden ser creados usando la función `c()`:

Más Info
Incluso cuando tenemos solamente un entero, R lo va tratar como un contenedor de enteros con sólo un entero en él.

```
c(2, 3, 5, 7)

## [1] 2 3 5 7

c("Abe", "Betty", "Chan")

## [1] "Abe"   "Betty" "Chan"

c(1.2, 3.2, 4.5)

## [1] 1.2 3.2 4.5
```

Si usted intenta poner diferentes tipos de objetos en un mismo vector, R va intentar convertirlos todos en el mismo tipo de objeto. Si no lo logra, va generar un error.

Precaución!
Cuando se lee los datos creados en otros software (como Excel) o guardados en un archivo CSV, es importante saber cómo los datos faltantes fueron indicados, de otra forma, el código para los datos faltantes puede ser interpretado como un character, causando todos los otros artículos o cosas en esa columna sean convertidos en valores character también, perdiendo una parte importante de la información.

```
x <- c(1, 1.1, 1.2); x

## [1] 1.0 1.1 1.2

#Convierte el entero en numérico
class(x)
```

```
## [1] "numeric"

y <- c(TRUE, FALSE, 0, 1, 2); y

## [1] 1 0 0 1 2

#Convierte los lógicos a numéricos
class(y)

## [1] "numeric"

z <- c(1, TRUE, 1.2, "vector"); z

## [1] "1"      "TRUE"   "1.2"    "vector"

#Todos convertidos a character(texto)
class(z)

## [1] "character"
```

Los factores pueden ser creados envolviendo un vector con `factor()`:

```
w <- factor(x); w

## [1] 1   1.1 1.2
## Levels: 1 1.1 1.2

class(w)

## [1] "factor"
```

Note como los factores despliegan sus **niveles** (posibles valores) como también los valores por sí mismos. Cuando los datos categóricos son codificados como enteros, es importante recordar convertirlos a factores para ciertos procedimientos estadísticos y gráficos.

Enteros con patrones o vectores numéricos pueden ser creados usando el operador : o la función `seq()`.

CAVANDO HONDO

Un factor puede ser ordenado o no ordenado (que puede afectar como los estadísticos de prueba se desempeñan, de otra forma no importa mucho). El predeterminado para los factores son no ordenados. Si los factores son ordenados o desordenados, los niveles van a aparecer en un orden fijado – alfabético. La distinción entre los factores ordenados y desordenados está relacionado con si el orden es importante o es arbitrario

```
1:10

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

seq(1, 10, by=0.5)

##  [1]  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0
## [12]  6.5  7.0  7.5  8.0  8.5  9.0  9.5 10.0
```

Los artículos individuales en un vector pueden ser accedidos o asignados usando el operador de paréntesis cuadrado:

```
w[1]

## [1] 1
## Levels: 1 1.1 1.2

x[2]

## [1] 1.1

y[3]

## [1] 0
```

Los valores faltantes están codificados como `NA` (no disponibles). Preguntar por una entrada fuera de los alcances de un vector devuelve un `NA`. Asignar un valor fuera del alcance del vector resulta en un vector que se hace del tamaño de tal forma que el nuevo valor pueda ser guardado en la posición apropiada.

```
z[5]    #Esto no es un error, pero devuelve un NA

## [1] NA

q <- 1:5
q

## [1] 1 2 3 4 5

q[10] <- 10
#Los elementos de 6 a 9 se rellenan con un NA
q

##  [1]  1  2  3  4  5 NA NA NA NA 10
```

R también proporciona algunas característica más inusuales (pero muy útiles) para acceder a los elementos de un vector.

```
letters                       # alfabeto

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

Más Info

`letters` is a built-in character vector containing the lower case letters. `LETTERS` contains capitals.

```
x <- letters[1:10]; x         # Primeras 10 letras

##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"

x[2:4]

## [1] "b" "c" "d"

# seleccione lo contenido de 2 a 4
x[2:4] <- c("X","Y","Z"); x

##  [1] "a" "X" "Y" "Z" "e" "f" "g" "h" "i" "j"

# Cambiar lo contenido de 2 a 4
y <- (1:10)^2; y

##  [1]   1   4   9  16  25  36  49  64  81 100

# Primeros 10 cuadrados
y [ y > 20 ]

## [1]  25  36  49  64  81 100

# selecciona aquellos mayores a 20
```

El último elemento merece un poco de comentarios. La expresión adentro de los paréntesis cuadrados evalúa a un vector de valores lógicos.

```
y > 20

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

Los valores lógicos después son usados para seleccionar (true) o no seleccionar (false) los elementos en el vector, produciendo un nuevo (y potencialmente más pequeño) vector. Si el número de valores lógicos propor-

cionados es menor que la cantidad de valores del vector, los valores están siendo **reciclados** (repetidos).

```
y[ c(TRUE,FALSE) ]          # cada uno

## [1]  1  9 25 49 81

y[ c(TRUE,FALSE,FALSE) ]    # Cada tercero

## [1]   1  16  49 100
```

Una **matriz** es una tabla de valores bi-dimensional, o sea de valores que tienen el mismo tipo. Como con los vectores, todo adentro de ellos debe ser del mismo tipo. Pero las matrices son bi-dimensionales – cada dato dentro de ellas está localizado en una fila y columna. Un **array** es una versión multi-dimensional de una matriz. Las matrices y los arrays son contenedores importantes para el trabajo estadístico, pero poco probable que sean utilizados por principiantes.

```
M <- matrix(1:15, nrow=3); M    # a 3 x 5 matrix

##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    4    7   10   13
## [2,]    2    5    8   11   14
## [3,]    3    6    9   12   15
```

Las dimensiones de un array, matriz o data frame pueden ser obtenidas usando `dim()`, `nrow()` y `ncol()`

```
dim(M)

## [1] 3 5

dim(KidsFeet)

## [1] 39  8

nrow(KidsFeet)

## [1] 39

ncol(KidsFeet)

## [1] 8
```

Más Info

En el habla oficial de R, la distinción que hacemos entre vectores y listas es realmente una distinción entre vectores *atómicos* y *listas* (que también se llaman vectores *genéricos*). De hecho, ellos deben ser del mismo tipo *atómico*. Los vectores atómicos son los cimientos

Otro contenedor comúnmente utilizado en R es la lista. Ya hemos visto algunos ejemplos de listas, como argumentos de funciones de gráficos de `lattice`. Las listas son ordenadas, pero los datos en una lista pueden ser objetos de cualquier tipo, no necesitan ser todos del mismo tipo. Sin maquillaje, un data frame es una lista de vectores con la restricción de que cada vector debe tener el mismo tamaño (contener el mismo número de datos).

Las listas pueden crearse usando la función `list()`.

```
l <- list(1, "two", 3.2, list(1, 2)); l

## [[1]]
## [1] 1
##
## [[2]]
## [1] "two"
##
## [[3]]
## [1] 3.2
##
## [[4]]
## [[4]][[1]]
## [1] 1
##
## [[4]][[2]]
## [1] 2

length(l)       #Note que l tiene 4 elementos, no 5

## [1] 4
```

Se puede acceder a los atos en la lista con un doble paréntesis cuadrado (`[[ ]]`)

```
l[[1]]

## [1] 1
```

Usando un único paréntesis cuadrado(`[ ]`) en su lugar, devuelve una sublista en lugar de un elemento. Entonces `l[[1]]` es un vector, pero `l[1]` es una lista que contiene un vector.

```
l[1]

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

Ambos vectores y listas pueden tener nombres. Los nombres pueden ser creados cuando el vector o la lista es creada o pueden ser puestos después. Se puede acceder a los elementos del vector y la lista por nombre o por posición.

```
x <- c(one=1, two=2, three=3); x

##   one   two three
##     1     2     3

y <- list(a=1, b=2, c=3); y

## $a
## [1] 1
##
## $b
## [1] 2
##
## $c
## [1] 3

x["one"]

## one
##   1

y[["a"]]   # retrieve items from a list with [[ ]]

## [1] 1

names(x)

## [1] "one"   "two"   "three"

names(x) <- c("A", "B", "C"); x

## A B C
## 1 2 3
```

Los operadores de acceso - `[ ]` y `[[ ]]` para las listas – son en realidad funciones en R. Esto tiene consecuencias importantes:

- Acceder a elementos en un vector es más lento en lenguajes como C/C++ donde el acceso es hecho por un puntero aritmético.

- Estas funciones pueden tener argumentos con nombre, entonces puede ver el código así

```
M

##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    4    7   10   13
## [2,]    2    5    8   11   14
## [3,]    3    6    9   12   15

M[5]

## [1] 5

M[,2]                   #Esto es 1-d (un vector)

## [1] 4 5 6

M[,2, drop=FALSE]       #Esto es 2-d (todavía una matriz)

##      [,1]
## [1,]    4
## [2,]    5
## [3,]    6
```

Los data frames pueden ser construidos suministrando `data.frame()` a las variables(como vectores):

```
ddd <- data.frame(number=1:5, letter=letters[1:5])
```

8.2.3 Funciones vectorizadas

Los vectores son tan importantes para R que merecen alguna discusión adicional. Muchas funciones de R y operaciones son "vectorizadasz pueden ser aplicadas no sólo a un valor individual, sino a un vector completo, que en este caso se les aplica a todas las componentes y muestra un vector de valores transformados. Muchas de las funciones más utilizadas de las matemáticas funcionan de esta forma.

```
x <- 1:5; y <- seq(10, 60, by=10)
x
```

```
## [1] 1 2 3 4 5

y

## [1] 10 20 30 40 50 60

y + 1                    #Agrega un 1 a cada elemento

## [1] 11 21 31 41 51 61

x * 10                   #Multiplica por 10 cada elemento

## [1] 10 20 30 40 50

x < 3                    #Verifica cual es menor a 3

## [1]  TRUE  TRUE FALSE FALSE FALSE

x^2                      #El cuadrado de cada elemento

## [1]  1  4  9 16 25

sqrt(x)                  #Raíz cuadrada de cada elemento

## [1] 1.00 1.41 1.73 2.00 2.24

log(x)                   #Log natural

## [1] 0.000 0.693 1.099 1.386 1.609

log10(x)                 #Log base 10

## [1] 0.000 0.301 0.477 0.602 0.699
```

Los vectores puede combinarse en una matriz usando `rbind()` o `cbind()`. Esto puede facilitar comparaciones poniéndolas una al lado de la otra This can facilitate side-by-side comparisons.

```
#compare round() y signif() juntando por fila en una matriz
z <- rnorm(5); z

## [1] -0.5605 -0.2302  1.5587  0.0705  0.1293

rbind(round(z, digits=3), signif(z, digits=3))

##       [,1]  [,2] [,3]   [,4]  [,5]
## [1,] -0.56 -0.23 1.56 0.0710 0.129
## [2,] -0.56 -0.23 1.56 0.0705 0.129
```

8.2.4 *Funciones que actúan sobre vectores como vectores*

Otras funciones, incluyendo muchas funciones estadísticas, son diseñadas para calcular un único número (técnicamente un vector de tamaño 1) de todo un vector.

```
z <- rnorm(100)
#Funciones estadísticas básicas; note el uso de los nombres
c(mean=mean(z), sd=sd(z), var=var(z), median=median(z))

##    mean      sd     var  median
##  0.0607  0.9089  0.8260 -0.0114

range(z)

## [1] -2.31  2.19

x <- 1:10
c(sum=sum(x), prod=prod(x))   #sumatorias y productos

##     sum    prod
##      55 3628800
```

Incluso, otras funciones devuelven vectores que son derivados del vector original, pero no como una transformación de los componentes.

```
z <- rnorm(5); z

## [1] -0.045 -0.785 -1.668 -0.380  0.919
```

```
sort(z); rank(z); order(z)

## [1] -1.668 -0.785 -0.380 -0.045  0.919
## [1] 4 2 1 3 5
## [1] 3 2 4 1 5
```

```
x <- 1:10
rev(x)           # reverso x

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

diff(x)          #diferencias de pares

## [1] 1 1 1 1 1 1 1 1 1

ediff(x)         #Diferencias sin cambiar el tamaño

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

cumsum(x)        #Suma acumulativa

##  [1]  1  3  6 10 15 21 28 36 45 55

cumprod(x)       #Producto acumulado

##  [1]       1       2       6      24     120     720    5040
##  [8]   40320  362880 3628800
```

Que una función esté vectorizada o trato el vector como una unidad depende de su implementación. Usualmente, las cosas son implementadas de la manera que usted lo espera. Ocasionalmente puede descubrir que una función que usted quería vectorizada no lo es. Cuando escribe sus propias funciones, piense un poco si debería ser vectorizada, y pruebela con vectores de tamaño mayor a 1 para asegurarse que consiga el comportamiento que quería.

MÁS INFO
La función `Vectorize()` es una herramienta útil para convertir una función no vectorizada en una función vectorizada.

Las operaciones abajo pueden ser de ayuda cuando escribe sus propias funciones.

`cumsum()`
`cumprod()`
`cummin()`
`cummax()`
	Devuelve un vector de acumulativos de sumatorias, restas, productorias máximos y mínimos.

`pmin(x,y,...)` `pmax(x,y,...)`	Devuelve un vector de paralelos de mínimo o máximo donde el elemento *i*ésimo es max o min de `x[i]`, `y[i]`, ...
`which(x)`	Devuelve un vector de indices de elementos de `x` que son verdaderos Uso típico: `which(y > 5)` devuelve indices de donde los elementos de `y` son mayores a 5.
`any(x)`	Devuelve un `logical` indicando si algún elemento de `x` es verdadero Uso típico: `if (any(y > 5)) { ...}`.
`na.omit(x)`	Devuelve un vector sin valores perdidos
`unique(x)`	Devuelve un vector sin valores repetidos
`table(x)`	Devuelve una tabla que cuenta el número de ocurrencias de cada valor en `x`. La tabla es similar a un vector con nombres indicando los valores, pero no es un vector.
`paste(x,y,...,` `sep=" ")`	Copia `x` y `y` juntos en un componente (como strings) con `sep` entre los elementos. Reciclando aplicaciones

8.3 Trabajando con datos

En la sección 6.5 discutimos usar datos de paquetes de R, y en la sección 6.5.4 discutimos traer nuestros propios datos a R. En ambos escenarios, hemos asumido que los datos fueron introducidos y limpiados en otro software y todo se enfocó primariamente en importar los datos. En esta sección discutimos formas de crear y manipular datos con R. Pero primero, vamos a disctuir algunos detalles más sobre la importación de datos.

Incluso si usted primariamente usa la interface de RStudio para importar sus datos, es bueno saber sobre los métodos de la línea de comando, puesto que estos son requeridos para importar datos en documentos script, RMarkdown y `knitr`/LaTeX.

8.3.1 Control más fino de la importación de datos

Los argumentos `na.strings` pueden ser utilizados para especificar códigos de missing values. Ajuste los

`na.strings` como en la lectura siguiente de los archivos csv que pueden ser producidos por sistemas como SAS.

```
someData <- read.csv('file.csv',
  na.strings=c('NA',",'.','-','na'))
```

SAS usa un punto (`.`) para codificar los datos faltantes y algunos exportadores de csv usan '-'. Si no se hace la definición anterior o alguna parecida de los na.string, R va tratar los marcados de datos faltantes como datos normales, en lugar de `NA`. Esto fuerza a toda la variable a ser de tipo character (texto) aunque puede ser puramente numérica. De forma predeterminada, R va recodificar los datos de tipo character (texto) como un factor. Si prefiere dejar esas variables en formato character(texto), puede usar

Más Info
La función `read.file()` en el paquete `mosaic` usa esto como predeterminado de los `na.strings`.

```
someData <- read.file('file.csv',
  stringsAsFactors=FALSE)

## Reading data with read.csv()
```

Más Info
Esto funciona como `read.csv()` y `read.table()` también

Un control incluso más fino puede ser obtenido manualmente ajustando la clase (tipo) usado para cada columna en el documento. Además, esto acelera la lectura del documento. Para un archivo .csv con cuatro columnas, podemos declararlas para ser de clase entera, numérica, character (texto), y factor con el siguiente comando.

```
someData <- read.file('file.csv',
  na.strings=c('NA',",'.','-','na'),
  colClasses=c('integer','numeric','character','factor'))

## Reading data with read.csv()
```

8.3.2 *Ingresando los datos manualmente*

Ya hemos visto que la función `c()` puede ser utilizada para combinar elementos en un sólo vector..

```
x <- c(1, 1, 2, 3, 5, 8, 13); x

## [1]  1  1  2  3  5  8 13
```

La función `scan()` puede acelerar la entrada de datos en la consola permitiéndole borrar las comas. Valores individuales son separados por un espacio en blanco o nuevas líneas. Una línea en blanco es usada como señal del fin de los datos. Como predeterminado, `scan()` espera datos numéricos, pero es posible decirle a `scan()` que espere algo más, como datos en texto. Hay otras opciones para los tipos de datos, pero los numéricos y de texto son los más importantes de manejar. Vea `?scan` para más información y ejemplos.

¡Precaución!
Cuando use `scan()` asegúrese de recordar guardar los datos en algún lado. De otra forma, va tener que escribirlos de nuevo.

8.3.3 Simulando distribuciones muestrales

R tiene funciones que hacen simple muestrear de un amplio rango de distribuciones. Cada una de estas funciones empieza con la letra 'r'(de 'random') seguido del nombre de la distribución (usualmente abreviada). Los argumentos de la función son para especificar el tamaño de muestra deseada y cualquier valor paramétrico requerido para la distribución. Por ejemplo, para simular elegir una muestra de tamaño 12 de una población normal con media 100 y desviación estándar 10, usamos:

```
rnorm(12, mean=100, sd=10)

##  [1]  94.2 106.1  83.8  99.4 105.2 103.0 101.1  93.6  91.5
## [10]  89.8 101.2  90.5
```

Las funciones para muestrear de otras distribuciones incluyen: `rbinom()`, `rchisq()`, `rt()`, `rf()`, `rhyper()`, etc.

Es también hacer hacer una muestra (con o sin reemplazo) de datos existentes usando `sample()` y `resample()`.

```
x <- 1:10
#muestra aleatoria de tamaño 5 de x (sin reemplazo)
sample(x, size=5)

## [1]  4  7 10  9  6
```

```
#muestra aleatoria de tamaño 5 de x(con reemplazo)
resample(x, size=5)

## [1] 8 3 3 6 3
```

Usar `resample()` hace más fácil simular distribuciones pequeñas discretas. Por ejemplo tirar 20 dados. Podemos usar

```
resample(1:6, size=20)

##  [1] 4 5 2 3 3 6 6 6 5 6 4 4 3 3 1 4 6 1 1 1
```

Para trabajar con cartas, el paquete `mosaicData` proporciona un vector llamado `Cards` y `deal()` como una alternativa de `sample()`.

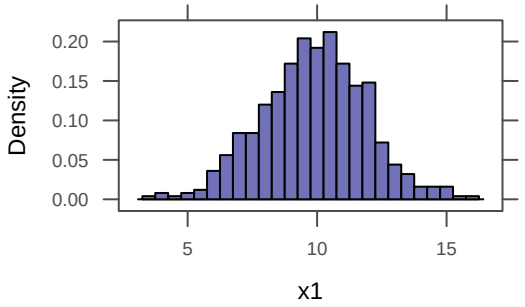
```
deal(Cards, 5)    #Mano de poker

## [1] "3S" "KH" "JS" "JD" "5C"

deal(Cards, 13)   #Un bridge? alguien?

##  [1] "9H" "AH" "8C" "8D" "QC" "4C" "7D" "9S" "JC" "7S" "4H"
## [12] "AC" "6C"
```

Si quiere repartir las manos bien, puede crear un factor de `Cards` primero:

```
hand <- deal(factor(Cards, levels=Cards), 13)
sort(hand)       # sorted by suit, then by denomination

##  [1] 2C  5C  6C  AD  5H  8H  10H JH  KH  3S  7S  9S  10S
## 52 Levels: 2C 3C 4C 5C 6C 7C 8C 9C 10C JC QC KC AC ... AS
```

Example 8.1. Para propósitos de enseñanza, en ocasiones es bueno crear un histograma que tiene la forma apróximada de una distribución. Una forma de hacer esto es muestrear aleatoriamente la muestra de una distribución deseada y hacer un histograma del resultado

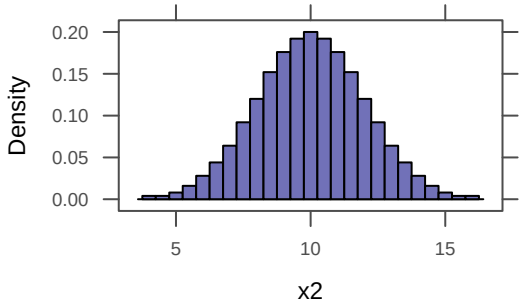
```
x1 <- rnorm(500, mean=10, sd=2)
histogram( ~ x1, width=.5)
```



Esto funciona, pero el gráfico resultante tiene una cantidad alta de ruido.

La función `ppoints()` devuelve probabilidades igualmente espaciadas que nos permiten obtener cuantiles teóricos de la distribución normal. El gráfico que resulta de esto, ilustra la muestra ideal de una distribución normal.

```
x2 <- qnorm(ppoints(500), mean=10, sd=2)
histogram( ~ x2, width=.5)
```



Esto no es como los datos reales se van a ver (incluso si vienen de una población normal), pero puede ser mejor para propósitos ilustrativos remover el ruido. ◇

8.3.4 Guárdando los datos

`write.table()` y `write.csv()` pueden ser usados para guardar datos de R en archivos planos.

```
ddd <- data.frame(number=1:5, letter=letters[1:5])
write.table(ddd, "ddd.txt")
write.csv(ddd, "ddd.csv")
```

Para más de una importación o exportación de datos, especialmente de otros formatos, vea el manual de *R Data Import/Export* (Importanción/Exportación de datos en R) disponible en.
@

Más Info
Si quiere salvar un objeto en R pero no su nombre, puede usar la función `I()`f you want to save an R object but not its name, you can use `saveRDS()` and choose its name when you read it with `readRDS()`.

8.4 *Manipulating Data Frames with* `dplyr`

Hay algunas formas de manipular data frames en R. El abordaje que ilustramos aquí depende mucho de las funciones en el paquete `dplyr`. Este paquete es cargado cuando el paquete `mosaic` es cargado. El paquete `dplyr` define cinco operaciones primarias en un data frame.

1. `mutate()` – agregar o cambiar variables
2. `select()` – elegir un subconjunto de columnas
3. `filter()` – elegir un subconjunto de filas
4. `summarise()` – reducir todos los datos en el conjunto de datos a una fila de resumen
5. `arrange()` – reordenar las filas

Esto se vuelve especialmente poderoso cuando se combina con el sexto comando, `group_by()`.

6. `group_by()` –divide el data frame en diferentes subconjuntos

Adicionalmente las funciones `inner_join()` y `left_join()` pueden ser utilizadas para combinar datos de múltiples data frames

8.4.1 *Agregando nuevas variables a un data frame.*

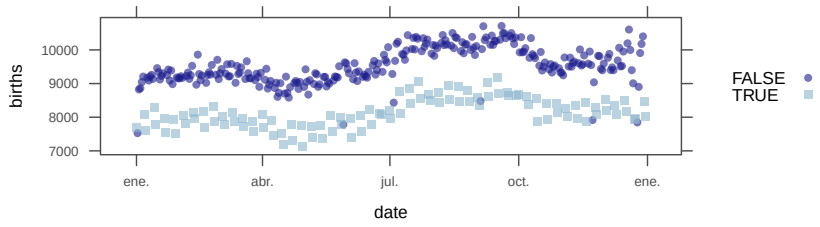
La función `mutate()` puede ser usada para agregar o modificar variables en un data frame.

Aquí mostramos cómo modificar el data frame de `Births78` para que contenga una nueva variable, `weekend`(fin de semana), que distingue entre los días entre semana y los fines de semana.

Nota
`mutate()` ingresa otras variables en el data frame, incluyendo cualquiera creada anteriormente en el mismo comando `mutate()`.

```
data(Births78)
weekdays <- c("Sun", "Mon", "Tue", "Wed", "Thr", "Fri", "Sat")
Births <-
  Births78 %>%
  mutate(weekend = wday %in% c("Sat", "Sun"))

head(Births, 3)

##         date births dayofyear wday weekend
## 1 1978-01-01   7701         1  Sun    TRUE
## 2 1978-01-02   7527         2  Mon   FALSE
## 3 1978-01-03   8825         3 Tues   FALSE
```

```
xyplot(births ~ date, Births, groups=weekend, auto.key=list(space='right'))
```



Número de nacimientos en Estados Unidos en 1978 coloreado por día de la semana.

El conjunto de datos `CPS85` contiene datos de la Encuesta Actual de Población (actual en 1985, es eso). Dos de las variables en este data frame son `age` y `educ`. Podemos estimar el número de años que un a trabajador ha estado en la fuerza de trabajo si asumimos que ha estado en la fuerza de trabajo desde que terminó la su educación, y su edad de graduación es 6 más el número de años de educación obtenida.

```
CPS85 <- mutate(CPS85, workforce.years = age - 6 - educ)
favstats( ~ workforce.years, data=CPS85)

##  min Q1 median Q3 max mean   sd   n missing
##   -4  8     15 26  55 17.8 12.4 534       0
```

```
tally( ~ (exper - workforce.years), data=CPS85)

## (exper - workforce.years)
##   0   4
## 533   1
```

Con una variable categórica, en ocasiones queremos modificar el esquema de código

```
HELP2 <- mutate(HELPrct,
  newsex = factor(female, labels=c('M','F')))
```

Es una buena idea hacer algún tipo de control de sanidad para asegurarse que la recodificación funcionó de acuerdo a su intención.

```
tally( ~ newsex + female, data=HELP2)

##       female
## newsex   0   1
##      M 346   0
##      F   0 107
```

La función `derivedFactor()` puede simplificar crear factores basado en algunas pruebas lógicas.

```
HELP3 <- mutate(HELPrct,
  risklevel = derivedFactor(
    low = sexrisk < 5,
  medium = sexrisk < 10,
        high = sexrisk >=10,
        .method = "first"      # use first rule that applies
        )
)
```

```
head(HELP3, 4)

##   age anysubstatus anysub cesd d1 daysanysub dayslink drugrisk e2b female    sex g1b
## 1  37            1    yes   49  3        177      225        0  NA      0   male yes
## 2  37            1    yes   30 22          2       NA        0  NA      0   male yes
## 3  26            1    yes   39  0          3      365       20  NA      0   male  no
## 4  39            1    yes   15  2        189      343        0   1      1 female  no
##   homeless i1 i2 id indtot linkstatus link   mcs  pcs pss_fr racegrp satreat sexrisk
## 1   housed 13 26  1     39          1  yes 25.11 58.4      0   black      no       4
## 2 homeless 56 62  2     43         NA <NA> 26.67 36.0      1   white      no       7
## 3   housed  0  0  3     41          0   no  6.76 74.8     13   black      no       2
## 4   housed  5  5  4     28          0   no 43.97 61.9     11   white     yes       4
##   substance treat risklevel
## 1   cocaine   yes       low
## 2   alcohol   yes    medium
## 3    heroin    no       low
## 4    heroin    no       low
```

8.4.2 *Desechando variables*

Puesto que ya tenemos `educ`, no tenemos razones para dejarnos nuestra nueva variable `workforce.years`. Desechemosla. Note el uso inteligente del signo de menos.

```
CPS1 <- select(CPS85, -workforce.years)
head(CPS1, 1)

##   wage educ race sex hispanic south married exper union age
## 1    9   10    W   M       NH    NS Married    27   Not  43
##   sector
## 1  const
```

Cualquier número de variables pueden ser desechadas o se pueden mantener de esta forma supliendo un vector de nombres de variables.

```
CPS1 <- select(CPS85, c(workforce.years,exper))
```

Las columnas pueden especificarse por número, así como por nombre (esto puede ser peligroso si está equivocado sobre dónde están las columnas).

CAVANDO HONDO

Los programadores profesionales en R, así como Hadley Wickham, el autor del paquete `dplyr`, toman como ventaja una característica especial del lenguaje que permite que la notación del menos signifique .exclusión".

```
CPSsmall <- select(CPS85, select=1:4)
head(CPSsmall,2)

##   select1 select2 select3 select4
## 1     9.0      10       W       M
## 2     5.5      12       W       M
```

Las funciones `matches()`, `contains()`, `starts_with()`, `ends_with()`, y `number_range()` son funciones especiales que sólo funcionan en el contexto de `select()` pero pueden ser útiles para describir conjuntos de variables o para mantenerlas o desecharlas.

```
head(select(HELPrct, contains("risk")), 2)

##   drugrisk sexrisk
## 1        0       4
## 2        0       7
```

La función anidada en el comando previo hace el código un poco más difícil de leer, y podría ser peor si estuviéramos utilizando algunas funciones más. El paquete magritt (que carga cada vez que dplyr carga, ergo, carga cada vez que mosaic carga) nos proporciona una alternativa para la sintaxis

```
HELPrct %>% select(contains("risk")) %>% head(2)

##   drugrisk sexrisk
## 1        0       4
## 2        0       7
```

El operador `%>%` usa las salidas de la parte izquierda como la primera entrada para la función que está a la derecha. Esto hace sencillo encadenar varios comandos de manipulación de datos en el orden que se aplican a los datos sin tener que anidar con mucho cuidado los paréntesis, y sin tener que pasar salidas de una función como un argumento para la próxima.

Aquí hay unos cuantos ejemplos más:

```
HELPrct %>% select(ends_with("e"))   %>% head(2)

##   age female substance
## 1  37      0   cocaine
## 2  37      0   alcohol
```

```
HELPrct %>% select(starts_with("h")) %>% head(2)

##   homeless
## 1   housed
## 2 homeless

HELPrct %>% select(matches("i[12]")) %>% head(2)  # regex matching

##   i1 i2
## 1 13 26
## 2 56 62
```

8.4.3 *Renombrando las variables*

Ambos, los nombres de la columna (variables) y los nombres de la fila de un conjunto de datos pueden ser cambiadas por un simple asignamiento, usando `names()` o `row.names()`.

```
ddd         # Datos pequeños que definimos antes

##   number letter
## 1      1      a
## 2      2      b
## 3      3      c
## 4      4      d
## 5      5      e

#Cambiar el nombre de las filas afecta como se imprime el data frame
row.names(ddd) <- c("Abe","Betty","Claire","Don","Ethel")
ddd

##        number letter
## Abe         1      a
## Betty       2      b
## Claire      3      c
## Don         4      d
## Ethel       5      e
```

Es también posible ajustar un único nombre con la siguiente sintaxis.

```
#El nombre está mal escrito, arreglémoslo
row.names(ddd)[2] <- "Bette"
row.names(ddd)
```

```
## [1] "Abe"    "Bette"  "Claire" "Don"    "Ethel"
```

El conjunto de datos `faithful` (en el paquete `datasets`, que está siempre disponible) tiene unos nombres desafortunados.

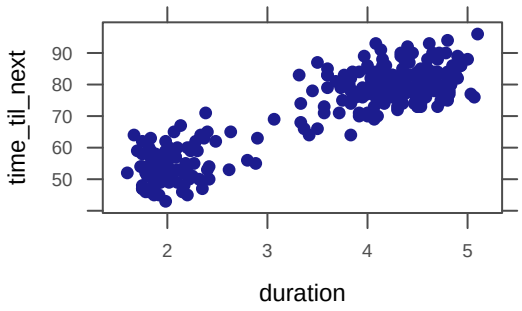
```
names(faithful)

## [1] "eruptions" "waiting"
```

Las medidas son la duración de una erupción y el tiempo a la erupción posterior, entonces mejor se les da mejores nombres.

Consejo de Enseñanza
Una solución alternativa es usar el conjunto de datos `geyser` en el paquete `MASS`. El data frame geyser tiene mejores nombres y más datos. Pero queremos ilustrar como reparar el daño en faithful.

```
names(faithful) <- c('duration', 'time_til_next')
head(faithful, 3)

##   duration time_til_next
## 1     3.60            79
## 2     1.80            54
## 3     3.33            74
```

```
xyplot(time_til_next ~ duration, faithful)
```



Si la variable que contiene un data frame es modificada o utilizada para guardar un objeto diferente, los datos originales del paquete se pueden recuperar usando `data()`.

Podemos también renombrar una única variable usando `names()`. Por ejemplo, tal vez queremos renombrar `educ`(la segunda columna) y ponerle educación.

```
names(CPS85)[2] <- 'educacion'
CPS85[1,1:4]

##   wage educacion race sex
## 1    9        10    W   M
```

Si no sabemos el número de columna (o generalmente para hacer nuestro código más claro), un poco

```
names(CPS85)[names(CPS85) == 'education'] <- 'educ'
CPS85[1,1:4]

##   wage educacion race sex
## 1    9        10    W   M
```

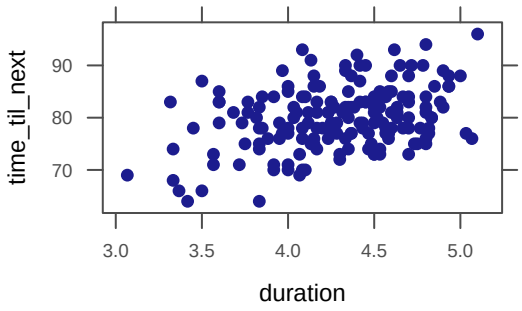
La función `select()` también se puede usar para renombrar variables.

```
data(faithful)    #restaura la versión original
faithful2 <- faithful %>%
  select(duration=eruptions, time_til_next = waiting)
head(faithful2, 2)

##   duration time_til_next
## 1      3.6            79
## 2      1.8            54
```

8.4.4 Creando subconjuntos

Podemos usar `filter()` para elegir solamente algunas filas de nuestro conjunto de datos.

```
#Cualquier estructura lógica puede ser utilizada para crear subconjuntos
faithful2 %>% filter(duration > 3) -> faithfulLong
xyplot(time_til_next ~ duration, faithfulLong)
```



Si lo único que queremos es producir un gráfico y no necesitamos guardar el subconjunto, el gráfico mostrado anteriormente también puede ser construido de cualquiera de las siguientes formas

```
xyplot(time_til_next ~ duration,
       data = faithful2 %>% filter(duration > 3))
xyplot(time_til_next ~ duration, data = faithful2,
       subset=duration > 3)
```

8.4.5 Resumiendo un conjunto de datos.

La función `summarise()` (o `summarize()`) resume un conjunto de datos en una única fila.

```
HELPrct %>% summarise(x.bar = mean(age), s=sd(age))

##   x.bar    s
## 1  35.7 7.71
```

Esto es especialmente útil en conjunto con `group_by()`, que divide los conjuntos de datos en subconjuntos. El siguiente comando va calcular la media y desviación estándar para cada sub-grupo definido por una diferente combinación de sexo y sustancia.

```
HELPrct %>% group_by(sex, substance) %>%
  summarise(x.bar = mean(age), s=sd(age))

## # A tibble: 6 x 4
## # Groups:   sex [?]
##      sex substance x.bar     s
##   <fctr>    <fctr> <dbl> <dbl>
## 1 female   alcohol  39.2  7.98
## 2 female   cocaine  34.9  6.20
## 3 female    heroin  34.7  8.04
## 4   male   alcohol  38.0  7.58
## 5   male   cocaine  34.4  6.89
## 6   male    heroin  33.1  7.97
```

Este resumen numérico basado en funciones que proporciona el paquete `mosaic` es probablemente más fácil para esta tarea particular, pero usar `dplyr` es más general.

```
favstats(age ~ sex + substance, data=HELPrct, .format="table")
```

```
##    sex.substance min Q1 median Q3 max mean   sd   n missing
## 1 female.alcohol  23 33   37.0 45  58 39.2 7.98  36       0
## 2   male.alcohol  20 32   38.0 42  58 38.0 7.58 141       0
## 3 female.cocaine  24 31   34.0 38  49 34.9 6.20  41       0
## 4   male.cocaine  23 30   33.0 37  60 34.4 6.89 111       0
## 5  female.heroin  21 29   34.0 39  55 34.7 8.04  30       0
## 6    male.heroin  19 27   32.5 39  53 33.1 7.97  94       0
```

```
mean(age ~ sex + substance, data=HELPrct, .format="table")
```

```
##            group mean
## 1 female.alcohol 39.2
## 2   male.alcohol 38.0
## 3 female.cocaine 34.9
## 4   male.cocaine 34.4
## 5  female.heroin 34.7
## 6    male.heroin 33.1
```

```
sd(age ~ sex + substance, data=HELPrct, .format="table")
```

```
##            group   sd
## 1 female.alcohol 7.98
## 2   male.alcohol 7.58
## 3 female.cocaine 6.20
## 4   male.cocaine 6.89
## 5  female.heroin 8.04
## 6    male.heroin 7.97
```

8.4.6 *Organizando el data frame.*

En ocasiones es conveniente re-ordenar el conjunto de datos. Podemos hacer esto con la función `arrange()` especificándole en qué variables estamos haciendo la clasificación.

```
HELPrct %>%
  group_by(sex, substance) %>%
  summarise(x.bar = mean(age), s=sd(age)) %>%
  arrange(x.bar)

## # A tibble: 6 x 4
## # Groups:   sex [2]
##      sex substance x.bar     s
##   <fctr>    <fctr> <dbl> <dbl>
## 1   male    heroin  33.1  7.97
## 2   male   cocaine  34.4  6.89
## 3 female    heroin  34.7  8.04
## 4 female   cocaine  34.9  6.20
## 5   male   alcohol  38.0  7.58
## 6 female   alcohol  39.2  7.98
```

8.4.7 *Uniendo conjuntos de datos*

El conjunto de datos `fusion1` en el paquete `fastR` contiene información del genotipo para un SNP (Polimorfismo de nucleotido solo) en el gen TCF7L2. El conjunto de datos `pheno` contiene fenotipos (incluyendo el status de caso/control de diabetes tipo 2) para un conjunto cruzado de individuos. Podemos unir estos dos conjuntos para explorar la asociación entre genotipos y fenotipos usando una de las funciones para juntar en `dplyr` o podemos usar la función `merge()`.

```
require(fastR)
fusion1 %>% head(3)

##      id     marker markerID allele1 allele2 genotype Adose Cdose Gdose Tdose
## 1  9735 RS12255372        1       3       3       GG     0     0     2     0
## 2 10158 RS12255372        1       3       3       GG     0     0     2     0
## 3  9380 RS12255372        1       3       4       GT     0     0     1     1

pheno %>% head(3)

##     id     t2d  bmi sex  age smoker chol waist weight height   whr sbp dbp
## 1 1002    case 32.9   F 70.8 former 4.57 112.0   85.6    161 0.987 135  77
## 2 1009    case 27.4   F 53.9  never 7.32  93.5   77.4    168 0.940 158  88
## 3 1012 control 30.5   M 53.9 former 5.02 104.0   94.6    176 0.933 143  89
```

```
#Una unión entre fusion1 y pheno, manteniendo solo sus ids, que están en ambos
fusion1m <- merge(fusion1, pheno, by.x='id', by.y='id',
                  all.x=FALSE, all.y=FALSE)
fusion1m %>% head(3)

##     id     marker markerID allele1 allele2 genotype Adose Cdose Gdose Tdose     t2d  bmi
## 1 1002 RS12255372        1       3       3       GG     0     0     2     0    case 32.9
## 2 1009 RS12255372        1       3       3       GG     0     0     2     0    case 27.4
## 3 1012 RS12255372        1       3       3       GG     0     0     2     0 control 30.5
##   sex  age smoker chol waist weight height   whr sbp dbp
## 1   F 70.8 former 4.57 112.0   85.6    161 0.987 135  77
## 2   F 53.9  never 7.32  93.5   77.4    168 0.940 158  88
## 3   M 53.9 former 5.02 104.0   94.6    176 0.933 143  89
```

```
pheno %>% left_join(fusion1, by="id") %>% dim()

## [1] 2333   22
```

```
pheno %>% inner_join(fusion1, by="id") %>% dim()

## [1] 2331   22
```

```
#¿Cuales ids están solo en \dataframe{pheno}?
setdiff(pheno$id, fusion1$id)

## [1] 4011 9131

pheno %>% anti_join(fusion1, by="id")

##     id     t2d  bmi sex age smoker chol waist weight height
## 1 4011    case 34.0   F  64  never 5.36   108   85.0    158
## 2 9131 control 26.7   M  73   <NA> 5.76    98   77.4    170
##     whr sbp dbp
## 1 0.861 160  82
## 2 0.940 119  72
```

La diferencia entre un inner join y un left join es que el inner join sólo incluye filas del primer conjunto de datos que emparejan con el segundo, mientras el left join incluye todas las filas incluso si no tienen pareja en el segundo. En el ejemplo anterior, hay dos sujetos en pheno que no aparecen en fusion1.

`merge()` maneja estas distinciones con los argumentos `all.x` o `all.y()`. En este caso, puesto que los valores son el mismo para cada conjunto de datos, podemos colapsar `by.x` y `by.y` y colapsar `all.x` y `all.y` a `all`. La primera de estas específica cual(es) columna(s) usa para identificar los casos emparejados. El segundo indica si los casos en un conjunto de datos que no aparecen en el otro conjunto se deben quedar (`TRUE`) o se deben desechar (llenando con `NA` si se necesita) o desechados del conjunto de datos que se unió.

Ya estamos listos para iniciar nuestro análisis

```
tally( ~ t2d + genotype + marker, data=fusion1m)

## , , marker = RS12255372
##
##          genotype
## t2d        GG  GT  TT
##   case    737 375  48
##   control 835 309  27
```

8.5 *Consiguiendo datos de bases de mySQL*

El paquete `RMySQL` permite acceder directamente a datos en bases de datos en MySQL y el paquete dplyr facilita procesar estos datos de la misma forma que datos para un data frame. Esto hace fácil trabajar con grandes conjuntos de datos almacenados en bases de datos públicas. El ejemplo a continuación consulta el buscador de genoma de la UCSC para encontrar todos los genes conocidos del cromosoma 1.

UCSC — Universidad de California, Santa Cruz

```
#Conectándose a las bases de datos de la UCSC
require(dbplyr)
library(RMySQL)
UCSCdata <- src_mysql(
  host="genome-mysql.soe.ucsc.edu",
  user="genome",
  dbname="mm9")
# Tome alguna de todas las tablas en la base de datos
KnownGene <- tbl(UCSCdata, "knownGene")

#Consiga el nombre del gen, cromosoma, empiece y termine sitios
#para genes en cromosoma 1
Chrom1 <-
  KnownGene %>%
  select(name, chrom, txStart, txEnd) %>%
  filter(chrom == "chr1")
```

Lo que resulta de `Chrom1` no es un conjunto de datos en forma de tabla de datos pero se comporta mucho como uno.

```
class(Chrom1)

## [1] "tbl_dbi"  "tbl_sql"  "tbl_lazy" "tbl"
```

```
Chrom1 %>%
  mutate(length=(txEnd - txStart)/1000) -> Chrom1l
Chrom1l

## # Source:   lazy query [?? x 5]
## # Database: mysql 5.6.26-log
## #   [genome@genome-mysql.soe.ucsc.edu:/mm9]
##          name chrom txStart   txEnd length
##         <chr> <chr>   <dbl>   <dbl>  <dbl>
##  1 uc007aet.1  chr1 3195984 3205713   9.73
##  2 uc007aeu.1  chr1 3204562 3661579 457.02
##  3 uc007aev.1  chr1 3638391 3648985  10.59
##  4 uc007aew.1  chr1 4280926 4399322 118.40
##  5 uc007aex.2  chr1 4333587 4350395  16.81
##  6 uc007aey.1  chr1 4481008 4483816   2.81
##  7 uc007aez.1  chr1 4481008 4486494   5.49
##  8 uc007afa.1  chr1 4481008 4486494   5.49
##  9 uc007afb.1  chr1 4481008 4486494   5.49
## 10 uc007afc.1  chr1 4481008 4486494   5.49
```

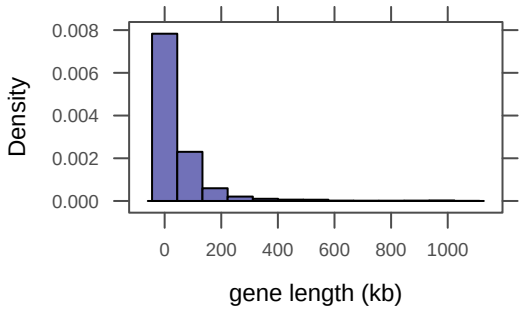
¡Precaución!
Las operaciones aritméticas en el comando `mutate()` en este caso son ejecutadas en SQL, no en R, por ende la gama de funciones que permite es más pequeña. No es posible, por ejemplo, ejecutar un logaritmo para todos usando `log()`. Para eso, primero debemos guardar la base en un data frame real.

```
## # ... with more rows
```

Para la eficiencia, todos los datos no son sacados de la base de datos hasta que sean necesarios (o hasta que se lo pidamos usando `collect()`). Esto nos permite, por ejemplo, inspeccionar las primeras filas de una potencial salida gigante de la base de datos sin en realidad haber hecho el trabajo requerido para sacar esos datos.

Pero ciertas cosas no funcionan a menos que juntemos los resultados de la base de datos en un data frame. Para hacer un gráfico de los datos usando `lattice` o `ggplot2`, por ejemplo, primero tenemos que usar `collect()` para juntarlo en un data frame.

```
Chrom1df <- collect(Chrom1l)       # collect into a data frame
histogram( ~ length, data=Chrom1df, xlab="gene length (kb)")
```



8.6 *Dándole nueva forma con `tidyr`.*

En ocasiones los datos vienen en una forma que no va de acuerdo a nuestros propósitos. El paquete `tidyr` incluye algunas funciones para organizar los datos, incluyendo `spread()` y `gather()`, que puede ser utilizado para convertir entre formatos “largosz “anchos". Podemos querer saber esto por un cambio en la perspectiva sobre lo que la unidad de observación es, por ejemplo. Por ejemplo en el conjunto de datos `traffic`, cada fila es un año y los datos, y los datos para multiples estados son proporcionados

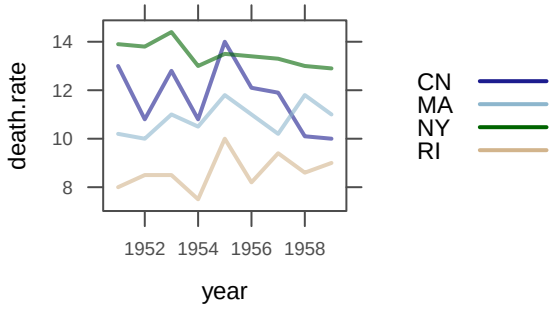
```
traffic

##   year cn.deaths   ny   cn   ma   ri
## 1 1951       265 13.9 13.0 10.2  8.0
## 2 1952       230 13.8 10.8 10.0  8.5
## 3 1953       275 14.4 12.8 11.0  8.5
## 4 1954       240 13.0 10.8 10.5  7.5
## 5 1955       325 13.5 14.0 11.8 10.0
## 6 1956       280 13.4 12.1 11.0  8.2
## 7 1957       273 13.3 11.9 10.2  9.4
## 8 1958       248 13.0 10.1 11.8  8.6
## 9 1959       245 12.9 10.0 11.0  9.0
```

Podemos re-formatear esto de tal forma que cada fila contenga la medida para un único estado en un año juntando los estados en columnas.

```
require(tidyr)

## Loading required package: tidyr

LongTraffic <-
  traffic %>%
  select(-cn.deaths) %>%
  gather(state, death.rate, ny:ri)
head(LongTraffic)

##   year state death.rate
## 1 1951    ny       13.9
## 2 1952    ny       13.8
## 3 1953    ny       14.4
## 4 1954    ny       13.0
## 5 1955    ny       13.5
## 6 1956    ny       13.4
```

Este formato largo nos permite crear gráficos como este.

```
xyplot(death.rate ~ year, data = LongTraffic, groups = toupper(state),
       type = "l",
       auto.key = list(space = "right", lines = TRUE, points = FALSE))
```

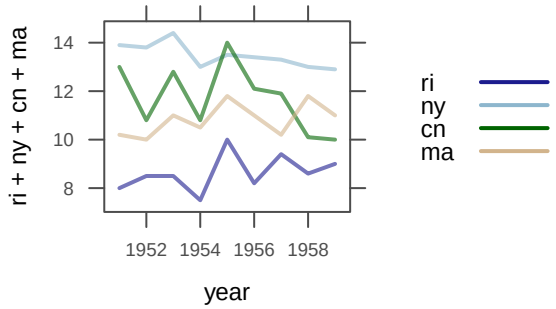


Podemos también reformatearlo de la otra forma, en esta ocasión teniendo todos los datos de un estado en una fila de la matriz de datos.

```
StateTraffic <-
  LongTraffic %>%
  spread(state, death.rate)
StateTraffic %>% head(3)

##   year   cn   ma   ny  ri
## 1 1951 13.0 10.2 13.9 8.0
## 2 1952 10.8 10.0 13.8 8.5
## 3 1953 12.8 11.0 14.4 8.5
```

Podemos crear un gráfico usando este formato de datos también, pero implica una notación de fórmula que no hemos visto antes:

```
xyplot(ri + ny + cn + ma ~ year, data=StateTraffic, type = "l",
       auto.key = list(space = "right", lines = TRUE, points = FALSE))
```



8.7 *Funciones en* R

Las funciones en R tienen algunos componentes:

- un **nombre** (como `histogram`)[1]
- Una lista ordenada de **argumentos**, que sirven como entradas de la función.

  Estas son emparejadas primero por nombre y después por el orden que se le da a los valores en la llamada de la función. Esta es la razón por la cual no siempre incluimos el nombre del argumento en los llamados de la función. Por otro lado, la disponibilidad de los nombres significa que no tenemos que recordar el orden en el cual el argumento es listado.

  Los argumentos usualmente tienen **valores predeterminados** que son utilizados si no se le da ningún valor al llamado de la función.
- Un **valor de salida**

  Es la salida de la función, puede ser asignado a una variable usando el operador de asignación (`=`, `<-`, o `->`).
- **Efectos secundarios**

  Una función puede hacer otras cosas (como hacer un gráfico o ajustar algunas preferencias) que no son necesariamente parte del valor de salida

[1] En realidad, es posible llamar funciones sin utilizar su nombre; y para funciones pequeñas que sólo se necesitan una vez, esto puede ser útil

Cuando se lee la documentación de ayuda para una función de R, va poder ver que está organizada en secciones que están relacionadas a estos componentes. La lista de argumentos aparece en la sección de Usage con otros valores predeterminados. Los detalles sobre cómo los argumentos suelen aparecen en la sección de **Arguments** (argumentos). El valor de salida en la sección de **Value** (valor). Cualquier efecto secundario usualmente es mencionado en la sección de **Details**.

Ahora intentemos escribir nuestra propia función. Suponga que frecuentemente desea calcular la media, mediana y desviación estándar de una distribución. Puede hacer una función que nos ayude a ahorrar un poco de escritura

Incluso si usted no termina escribiendo muchas funciones por su cuenta, escribir una función le va dar muchísima mejor percepción de como la información fluye en el código de R.

Ahora nombremos nuestra función `mystats()`. La función `mystats()` va tener sólo un argumento, que asumimos que va tener un vector de valores numéricos.Aquí está como la podemos definir:

```
mystats <- function(x) {
    mean(x)
    median(x)
    sd(x)
}
```

```
mystats((1:20)^2)

## [1] 128
```

Siempre hay formas de revisar la **clase** de un argumento, para ver si es un data frame, un vector, numérica, etc. Una función robusta debería revisar que los valores que se entregan a los argumentos son del tipo apropiado

La primera línea dice que estamos definiendo una función llamada `mystats()` con un argumento, llamado x. Las líneas dentro de los paréntesis de llave le dicen al código que sea ejecutado cuando la función es llamada. Entonces, nuestra función computa la media, después la mediana y después la desviación estándar.

But as you see, this doesn't do exactly what we wanted. So what's going on? The value returned by the last line of a function is (by default) returned by the function to its calling environment, where it is (by default) printed to the screen so you can see it. In our case, we computed the mean, median, and standard deviation, but only the standard deviation is being returned by the function and hence displayed. So this function is just an inefficient version of `sd()`. That isn't really what we wanted.

Podemos usar `print()` para imprimir cosas en el camino si queremos

```
mystats <- function(x) {
    print(mean(x))
    print(median(x))
    print(sd(x))
}

mystats((1:20)^2)

## [1] 144
```

```
## [1] 110
## [1] 128
```

Alternativamente, podemos usar una combinación de `cat()` and `paste()`, que nos va dar más control sobre como la salida es desplegada.

```
altmystats <- function(x) {
    cat(paste("  mean:", format(mean(x),4),"\n"))
    cat(paste(" edian:", format(median(x),4),"\n"))
    cat(paste("    sd:", format(sd(x),4),"\n"))
}
altmystats((1:20)^2)

##   mean: 144
##  edian: 110
##     sd: 128
```

Cualquiera de estos métodos va permitirnos ver tres valores, pero si intentamos guardarlos...

```
temp <- mystats((1:20)^2)

## [1] 144
## [1] 110
## [1] 128

temp

## [1] 128
```

Una función de R puede solamente tener un valor de salida, y predeterminadamente es el valor de la última línea en la función. En el ejemplo anterior sólo tenemos la desviación estándar puesto que es el último valor que se calculó.

Nos gustaría que la función nos de los 3 estadísticos de resumen. Nuestra solución va ser guarda todos los tres en un vector y que nos de ese vector.[2]

[2] Si los valores no están todos en la misma clase, podemos usar una lista.

```
mystats <- function(x) {
        c(mean(x), median(x), sd(x))
}
mystats((1:20)^2)

## [1] 144 110 128
```

Ahora el único problema es que si tenemos que recordar cual número es cual. Podemos arreglarlo dándole nombres a los espacios en nuestros vectores. Mientras estamos en eso, agreguemos un poco más de resúmenes favoritos a la lista. Además agreguemos explícitamente un `return()`.

```
mystats <- function(x) {
    result <- c(min(x), max(x), mean(x), median(x), sd(x))
    names(result) <- c("min","max","mean","median","sd")
    return(result)
}
mystats((1:20)^2)

##    min    max   mean median     sd
##      1    400    144    110    128

aggregate(Sepal.Length ~ Species, data=iris, FUN=mystats)

##      Species Sepal.Length.min Sepal.Length.max
## 1     setosa            4.300            5.800
## 2 versicolor            4.900            7.000
## 3  virginica            4.900            7.900
##   Sepal.Length.mean Sepal.Length.median Sepal.Length.sd
## 1             5.006               5.000          0.352
## 2             5.936               5.900          0.516
## 3             6.588               6.500          0.636
```

Note lo bien que esto trabaja con `aggregate()`. La función `favstats()` en el paquete mosaic incluye los cuartiles, la media, la desviación estándar, el tamaño de muestra y el número de observaciones faltantes

```
favstats(Sepal.Length ~ Species, data=iris)

##      Species min   Q1 median  Q3 max mean    sd  n missing
## 1     setosa 4.3 4.80    5.0 5.2 5.8 5.01 0.352 50       0
## 2 versicolor 4.9 5.60    5.9 6.3 7.0 5.94 0.516 50       0
## 3  virginica 4.9 6.23    6.5 6.9 7.9 6.59 0.636 50       0
```

Podemos conseguir una versión de nuestra nueva función que trabaje con la notación de fórmula como esta

```
#Primero creamos una versión que trabaje con vectores
mystats_ <- function(x, na.rm = TRUE) {
    result <- c(min(x, na.rm = na.rm), max(x, na.rm = na.rm), mean(x, na.rm = na.rm),
                median(x, na.rm = na.rm), sd(x, na.rm = na.rm))
    names(result) <- c("min","max","mean","median","sd")
    return(result)
}
#Ahora creemos una versión que conozca nuestra notación de fórmula
```

```
mystats <- aggregatingFunction1(mystats_, output.multiple = TRUE)

mystats(Sepal.Length ~ Species, data = iris)

##      Species min max mean median    sd
## 1     setosa 4.3 5.8 5.01    5.0 0.352
## 2 versicolor 4.9 7.0 5.94    5.9 0.516
## 3  virginica 4.9 7.9 6.59    6.5 0.636
```

8.8 *Compartiendo con y entre sus estudiantes*

Los instructores en varias ocasiones tienen sus propios conjuntos de datos para ilustrar puntos de interés estadístico o para hacer una conexión particular con la clase. En ocasiones se puede querer que la clase construya un conjunto de datos, llenando un cuestionario o contribuyendo con pequeñas partes para una colección de datos de la clase. Los estudiantes pueden estar trabajando en proyectos de grupos pequeños; es agradable tener herramientas que apoyen este trabajo de tal forma que los miembros de grupos tengan acceso los datos y puedan contribuir a los reportes escritos.

Hay ahora muchas tecnologías que apoyan esta forma de compartir. Por el bien de la simplicidad, vamos a enfatizar tres que hemos encontrado particularmente útiles, tanto en la enseñanza de la estadística como en nuestro trabajo profesional colaborativo. Estos son:

- Un sitio web, como Dropbox
- El servicio de Google Docs. .
- Un servidor basado en la web de RStudio.

Los primeros dos ahora son muy utilizados en el ambiente universitario y son accesibles simplemente mediante el ajuste de una cuenta. Ajustar un servidor de RStudio requiere algún soporte de IT, pero las habilidades de IT necesarios están en el rango de habilidades necesarias se pueden encontrar en algunos sujetos en las oficinas de IT de la facultad..

8.8.1 *Usando el servidor de* RStudio *para compartir documentos*

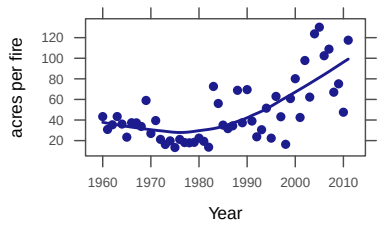
El servidor de RStudio se ejecuta desde una máquina con Linux. Los usuarios de RStudio tienen cuentas bajo el sistema de archivos de Linux y es posible ajustar directorios compartidos con permisos para que múltiples usuarios puedan leer y/o escribir archivos y los guarden ahí. Esto tiene que estar hecho afuera de RStudio, pero si está familiarizado con el sistema operativo Linux o tiene un administrador de sistema dispuestos a ayudarlo, no es difícil de hacer.

CONSEJO DE ENSEÑANZA
Cuando las cuentas ya están ajustadas en un servidor de RStudio para una nueva clase en Calvin, a cada usuario se le entrega una dirección simbólica con acceso a un directorio donde el instructor puede escribir documentos y los estudiantes únicamente pueden leer los documentos. Esto proporciona una forma sencilla de hacer datos, código de R, o historiales disponibles para estudiantes desde RStudio.

8.8.2 *Su propio sitio web*

Puede ser que ya tengan un sitio web. Tenga en mente un lugar donde puede poner archivos y que se pueda acceder a estos directamente de internet. Para compartir datos, es mejor que este sitio sea público, esto significa, que no requiera un inicio de sesión para que otros accedan a los documentos que usted sube. En este caso, `read.file()` puede leer los datos de R directamente de la siguiente URL:

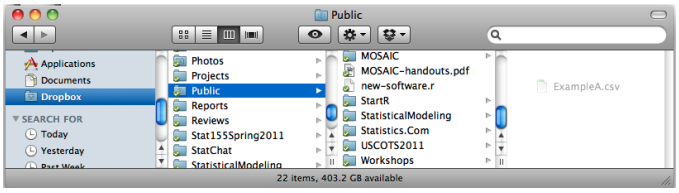
```
Fires <- read.csv("http://www.calvin.edu/~rpruim/data/Fires.csv")
head(Fires)

##   Year Fires   Acres
## 1 2011 74126 8711367
## 2 2010 71971 3422724
## 3 2009 78792 5921786
## 4 2008 78979 5292468
## 5 2007 85705 9328045
## 6 2006 96385 9873745

xyplot(Acres/Fires ~ Year, data=Fires, ylab="acres per fire",
       type=c("p","smooth"))
```



Desafortunadamente, la mayoría de nuestros sistemas de .ªpoyo de cursosçomo Moodle o Blackboard no proporcionan una forma tan sencilla de acceder a datos. El sistema de Dropbox para guardar datos en una "nube"brinda una forma muy conveniente de distribuir archivos por la web. (Diríjase a `dropbox.com` para información y para inscribirse con una cuenta gratuita.) Dropbox es rutinariamente utilizado para proporcionar respaldo y acceso a archivos coordinado en múltiples computadoras. Sin embargo, el servicio de Dropbox también ofrece un directorio público. Se puede acceder a cualquier archivo que usted suba directamente desde la URL

Para ilustrarlo, suponga que usted desea compartir algunos conjuntos de datos con sus estudiantes. Ya ha construido este en una hoja de Excel y la ha guardado en un archivo .csv, llamémoslo `ejemplo-A.csv`. Mueva este documento al directorio `Público` en Dropbox - en la mayoría de computadoras Dropbox ordena las cosas de tal forma que los directorios aparezcan exactamente como los directorios ordinarios y usted puede usar las técnicas familiares de manejo de datos, como mover los datos seleccionándolos y poniéndolos en otro lugar

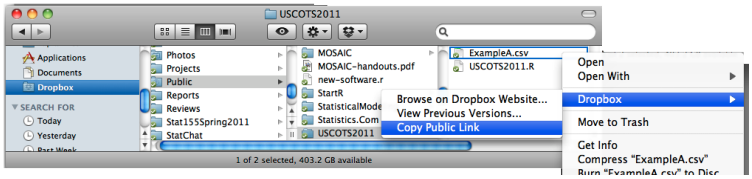
Nuestra discusión de Dropbox es primariamente para aquellos que no saben hacer esto de otra forma.



Dragging a csv file to a Dropbox Public directory

Dropbox también hace sencillo construir un URL identificador localizado en la web, para cualquier archivo, mediante el uso del mouse con comandos de menú para que pueda ponerse en una URL en el clipboard, en donde puede ser copiada en el software de apoyo del curso o cualquier otro lugar para la distribución a los estudiantes.

```
a <- read.file("http://dl.dropbox.com/u/5098197/USCOTS2011/ExampleA.csv")
```



Getting the URL of a file in a Dropbox Public directory

Esta técnica hace sencillo distribuir datos con muy poca preparación. Es lo suficientemente rápido como para hacerlo en media clase: el archivo csv está disponible para sus estudiantes (después de un poco de atraso mientras Dropbox se sincroniza). Puede incluso ser editado por usted (pero no por sus estudiantes).

La misma técnica puede ser aplicada a cualquier tipo de documento como entornos de R o scripts de R (archivos que contienen código). Por supuesto, sus estudiantes necesitan usar adecuadamente el comando de R: `load()` para un entorno y `source()` para un script.

El ejemplo anterior le consigue un archivo que imprime un mensaje de bienvenida para usted.

```
source('http://mosaic-web.org/go/R/hello.R')

## Hello there.  You just sourced a file over the web!
```

Pero usted puede poner cualquier código que quiera en los archivos que va hacer que sus estudiantes consigan. Puede instalar y cargar paquetes, recuperar o modificar conjuntos de datos, definir nuevas funciones o cualquier otra cosa que R permita.

Muchos instructores encuentran útil crear un archivo con sus scripts de R para el curso, agregando cosas y modificándolo conforme el curso progresa. Esto le permite distribuir todo tipo de función para propósitos especiales, permitiendo distribuir material de R nuevo a sus estudiantes. Esa idea brillante que tuvo a las 2 AM puede ser programada y puesta para que los estudiantes lo usen en la clase de la mañana siguiente. Luego, conforme identifica errores y refina la programación, puede hacer el software actualizado inmediatamente disponible para los estudiantes.

Si la privacidad es una preocupación, puede que quiera datos sólo para sus estudiantes, puede efectivamente lograr esto dándole a los archivos nombres que sólo conozcan sus estudiantes, e.g., `Example-A78r423.csv`.

PRECAUCIÓN!
La seguridad de este tipo no va satisfacer las regulaciones de protección de datos institucionales ni los requerimientos éticos profesionales, entonces nada verdaderamente sencillo o confidencial debe ser "protegido"de esta forma.

8.8.3 GoogleDocs

La técnica de Dropbox (o cualquier otro sistema de publicación en el internet) es una excelente idea para la difusión: Tomar documentos que creó y distribuirlos en una forma de solo-lectura para sus estudiantes. Pero cuando se quiere compartir un documento en dos vías o más, otras técnicas son necesarias, como la que proporciona el servicio GoogleDocs.

GoogleDocs permite a los estudiantes e instructores crear varias formas de documentos, incluyendo reportes, presentaciones y hojas de cálculo. (Además de crear documentos *de novo*, Google también convierte los documentos en una variedad de formatos.)

Una vez en, el sistema de GoogleDocs, los documentos pueden ser editados *simultáneamente* por múltiples usuarios en diferentes localizaciones. Pueden ser compartidos con individuos o grupos y publicado para vista sin restricción e incluso edición.

Para la enseñanza tiene una variedad de usos:

- Los estudiantes trabajando en proyectos de grupos pueden todos simultáneamente tener acceso al reporte conforme se escribe y a los datos que se están generando por el grupo.
- Toda la clase se le puede dar acceso al conjunto de datos, tanto para lectura como para escritura.
- El sistema de Google Forms puede ser utilizado para construir cuestionarios, cuyas respuestas puede poblar una hoja de cálculo que puede leer en RStudio por los creadores del cuestionario.
- Los estudiantes pueden .[en]tregarreportes y conjuntos de datos copiando el link en un sistema de apoyo como Moodle o Blackboard, o enviando el link.
- El instructor puede ingresar comentarios y/o correcciones directamente en un documento.

Una técnica efectiva para organizar es trabajo de estudiantes y asegurar al instructor (como a otros evaluadores) a tener acceso a estos, es crear un directorio de Google para cada estudiante en su clase (Dropbox se puede

usar de esta manera). Ajuste los permisos en el directorio para compartirla con cada estudiante. Cualquier cosa que él o ella ponga en el directorio estará inmediatamente disponible para el instructor. El estudiante también puede compartírselo a otro estudiante específico (erg., miembros de un proyecto de grupo).

Los datos pueden ser leídos directamente de las hojas de google usando el paquete `googlesheets`. Esto funcional mucho como `read_excel()` del paquete `readxl`.

8.9 Notas adicionales del sintaxis de R

8.9.1 El texto y las comillas

La mayoría de las veces, el texto en R debe ser contenido en comillas singulares o dobles. Usualmente no importa cual use, a menos que quiera una forma de comillas o la otra *adentro* de su texto. Después debe usar el otro tipo de comillas desde el principio hasta el final

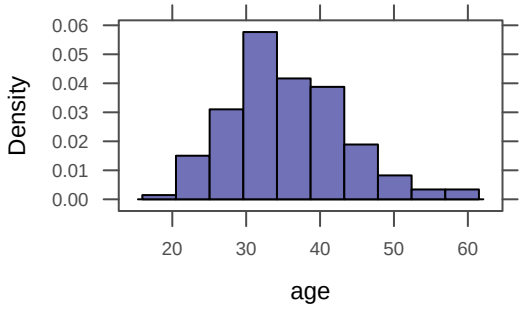
```
# La apostrofe dentro requiere comillas dobles alrededor del texto
text1 <- "Mary didn't come"
#Aquí cambiamos las cosas
text2 <- 'Do you use "scare quotes"?'
```

8.10 Mensajes de error comunes y qué los causa

8.10.1 Error: Object not found

R reporta cuando un objeto no es encontrado y cuando no puede localizar un objeto con el nombre que ha usado. Comúnmente la razón de esto es un error de escritura. Esto es fácilmente corregido escribiendo de nuevo el nombre de la forma correcta.

```
histogram( ~ age, data=HELPrct)
```



Otra razón para un error de objeto no encontrado es usar textos sin comillas cuando estas son necesarias

```
text3 <- hello
```

En este caso, R está buscando por algún objeto llamado `hello`, pero lo que queremos decir es que guarde los caracteres.

```
text3 <- "hello"
```

8.10.2 Error: unexpected . . .

Si mientras R está procesando una declaración, se encuentra algo que no tiene sentido, lo reporta como ünexpected"(inesperado). Usualmente esto es resultado de un error de escritura — como omitir una coma.

```
c(1,2 3)
falta una coma
## Error in c(): unexpected numeric constant in "c(1,2
3"
```

8.10.3 Error: object of type 'closureís not subsettable

Lo siguiente produce un error si time no ha sido definido.

```
time[3]

## Error in time[3]: objeto de tipo 'closureño es subconjunto
```

Hay una función llamada `time()` en R, entonces si usted no ha definido un vector por ese nombre, R va tratar de subdividir la función `time()`, lo cual no tiene mucho sentido.

Typically when you see this error, you have a function in a place you don't mean to have a function. The message can be cryptic to new users because of the reference to a closure.

8.10.4 Otros Errores

Si encuentra otros errores que no puede descifrar, en ocasiones pegar el mensaje de error en una búsqueda de google va encontrar discusión sobre el error en un contexto en el cual estancó a alguien más.

8.11 Repaso de comandos de R

Aquí hay un pequeño resumen de comandos introducidos este capítulo.

```
source("file.R")
#Ejecuta los comandos de un archivo

x <- 1:10
# crea un vector con los números de 1 a 10
M <- matrix(1:12, nrow=3)
# crea una matriz 3 x 4
data.frame(number = 1:26, letter=letters[1:26])
# crea un data frame
```

```r
mode(x)
#Devuelve la moda de un objeto de R
length(x)
# Devuelve el tamaño de un objeto
dim(HELPrct)
# dimension de la matriz, el arreglo o data frame
nrow(HELPrct)
# número de filas
ncol(HELPrct)
# número de columnas
names(HELPrct)
# los nombres de las variables en el dataframe
row.names(HELPrct)
# los nombres de las filas en el dataframe
attributes(x)
#devuelve los atributos de x
```

```r
toupper(x)
#pone mayúsculas
as.character(x)
#convierte a un vector de tipo caracter
as.logical(x)
# convierte a valores lógicos
as.numeric(x)
# convierte a números
as.integer(x)
#convierte a enteros
factor(x)
# convierte a factor [datos categóricos]
class(x)
#devuelve la clase del objeto x
```

```
smallPrimes <- c(2,3,5,7,11)
# crea un vector numérico
rep(1, 10)
# diez 1's
seq(2, 10, by=2)
#Números pares menores o iguales a 10
rank(x)
#Rankea lo contenido en el objeto x
sort(x)
#Devuelve los elementos de x ordenados
order(x)
# x[order(x)] de forma ordenada con x
rev(x)
#Devuelve los elementos de x en orden inverso
diff(x)
# Devuelve las diferencias entre elementos consecutivos
paste("Group", 1:3, sep="")
# lo mismo que c("Group1", "Group2", "Group3")
```

```
write.table(HELPrct, file="myHELP.txt")
# Escribir datos en un archivo
write.csv(HELPrct, file="myHELP.csv")
#Escribir datos en una archivo csv
save(HELPrct, file="myHELP.Rda")
#Guardar objetos en formato de R

modData <- HELPrct %>% mutate(old = age > 50)
#Agregar una nueva variable al conjunto de datos
women <- HELPrct %>% filter(sex=='female')
#Seleccione solamente las columnas especificadas
favs <- HELPrct %>% select(age, sex, substance)
#Mantenga solo 3 columnas

trellis.par.set(theme=col.mosaic())
#Elije el tema para los gráficos lattice
show.settings()
# inspecciona el tema de lattice
```

8.12 Ejercicios

8.1. Usando el conjunto de datos `faithful`, haga un grá-

fico de dispersión de la duración de la erupción contra el tiempo desde la última erupción..

8.2. El conjunto de datos `fusion2` en el paquete `fastR` contiene genotipos de otro SNP. Mezcle `fusion1`, `fusion2` y `pheno` en un único conjunto de datos.

Note que `fusion1` y `fusion 2` tienen las mismas columnas

```
names(fusion1)

##  [1] "id"       "marker"   "markerID" "allele1"  "allele2"
##  [6] "genotype" "Adose"    "Cdose"    "Gdose"    "Tdose"

names(fusion2)

##  [1] "id"       "marker"   "markerID" "allele1"  "allele2"
##  [6] "genotype" "Adose"    "Cdose"    "Gdose"    "Tdose"
```

Usted puede querer usar el argumento `suffixes` en `merge()` o renombrar las variables después de terminar de mezclar para hacer más fácil navegar entre ellas.

Organice su data frame sacando columnas que son redundantes o que no quiere tener en el conjunto de datos final.

9
Siendo interactivo: `manipulate` y `shiny`.

Una característica muy atractiva de RStudio es la función `manipulate()` (en el paquete `manipulate` sólo disponible en RStudio). Esta función hace más fácil crear una serie de controles (así como curvas, casillas de verificación, selecciones, etc.) que pueden ser utilizadas para cambiar dinámicamente valores dentro de una expresión. Cuando un valor es cambiado usando estos controles, la expresión es automáticamente re-ejecutada y los gráficos creados son un resultado del re-dibujo. Esto puede ser utilizado para rápidamente crear un prototipo de algunas actividades y demos que son parte de lecciones de estadística.

`shiny` es un sistema de desarrollo web nuevo para R, el cual es diseñado por el equipo de RStudio. shiny usa un modelo reactivo de programación para hacer relativamente fácil para un programador en R crear aplicaciones web interactivas, bien diseñadas y usando R sin necesidad de saber mucho acerca de programación web. Programar en `shiny` es más enredado que usar manipulate, pero ofrece al diseñador más flexibilidad. Una de las metas al crear shiny fue apoyar ambientes corporativos, donde un pequeño número de estadísticos y programadores pueden crear aplicaciones web que

puede ser utilizadas por otros en la compañía sin necesidad de saber R. Esta misma idea también ofrece muchas posibilidades para propósitos educativos. Incluso algunos han aconsejado implementar una interfaces GUI bien extensas a funcionalidades comunes de R usando shiny.

9.1 *Empezando con `manipulate`.*

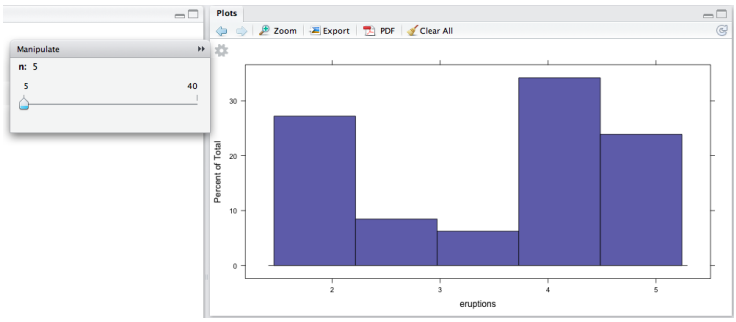
La función `manipulate()` y las varias funciones de control que son utilizadas en esta función se pueden utilizar solamente después de cargar el paquete manipulate, que está solo disponible en RStudio

```
require(manipulate)
```

9.1.1 *Deslizadores*

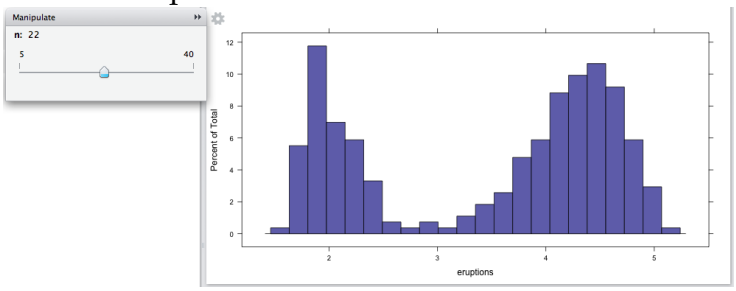
```
manipulate(
  histogram( ~ eruptions, data=faithful, n=N),
  N = slider(5,40)
)
```

Esto genera un gráfico junto con un deslizador que tiene un rango de 5 a 40 para el número de clases (barras)



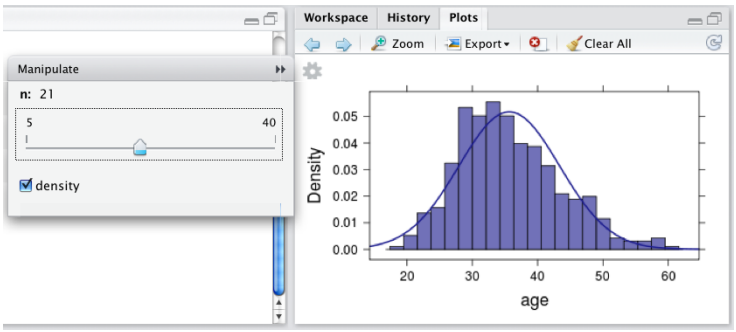
Encontramos útil ponerle mayúsculas a las entradas con las que la expresión manipulada se va enganchar en los controles de `manipulate`. Esto ayuda a evitar choques con los nombres y las señales de cómo la expresión principal manipulada está siendo usada.

Cuando movemos el deslizador, tenemos una mejor vista de las erupciones en `Faithful`.



9.1.2 2 Revisar las cajas (poner una función de densidad

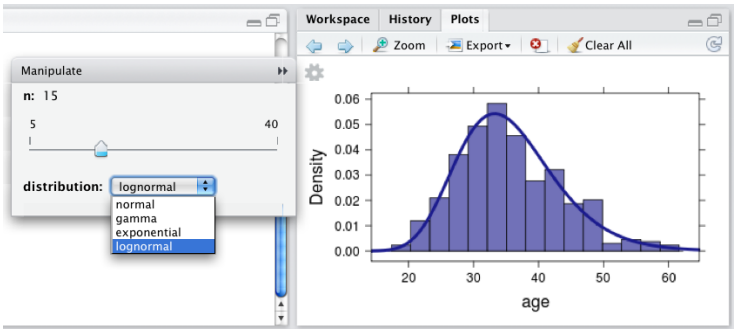
```
manipulate(
  histogram( ~ age, data=HELPrct, n=N, density=DENSITY),
  N = slider(5,40),
  DENSITY = checkbox()
)
```



9.1.3 Menús

Los menús pueden ser agregados usando la función `picker()`.

```
manipulate(
    histogram( ~ age, data=HELPrct, n=N,
                      fit=DISTRIBUTION, dlwd=4),
    N = slider(5,40),
    DISTRIBUTION =
        picker('normal', 'gamma', 'exponential', 'lognormal',
               label="distribution")
)
```

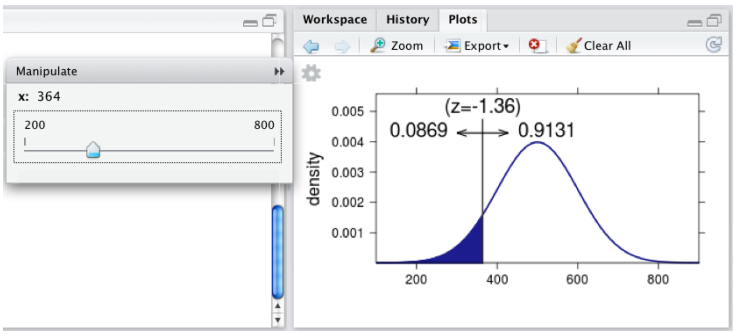


9.1.4 *Visualizando la distribución normal*

En esta sección gradualmente vamos a construir un pequeño ejemplo de `manipulate` que muestra la flexibilidad que viene de escribir una función que nos da de vuelta un objeto `manipulate`. Estas funciones pueden ser distribuidas a los estudiantes para permitirles explorar interactivamente de una forma más flexible.

Empezamos por crear una ilustración de una cola de probabilidades en la distribución normal.

```
manipulate(
    xpnorm( X, 500, 100, verbose=FALSE, invisible=TRUE ),
    X = slider(200,800) )
```



La versión anterior, puede ser utilizada también para investigar probabilidades centrales y de la cola.

```
manipulate(
    xpnorm( c(-X,X), 500, 100, verbose=FALSE, invisible=TRUE ),
    X = slider(200,800) )
```

Estos ejemplos funcionan con una distribución fijada. Aquí hay una forma más fina en la cual la función devuelve un objeto de manipulate. Esto permite fácilmente crear ilustraciones como las anteriores para cualquier distribución normal.

```
mNorm <- function( mean=0, sd=1 ) {
  lo <- mean - 5*sd
  hi <- mean + 5*sd
  manipulate(
    xpnorm( c(A,B), mean, sd, verbose=FALSE, invisible=TRUE ),
    A = slider(lo, hi, initial=mean-sd),
    B = slider(lo, hi, initial=mean+sd)
  )
```

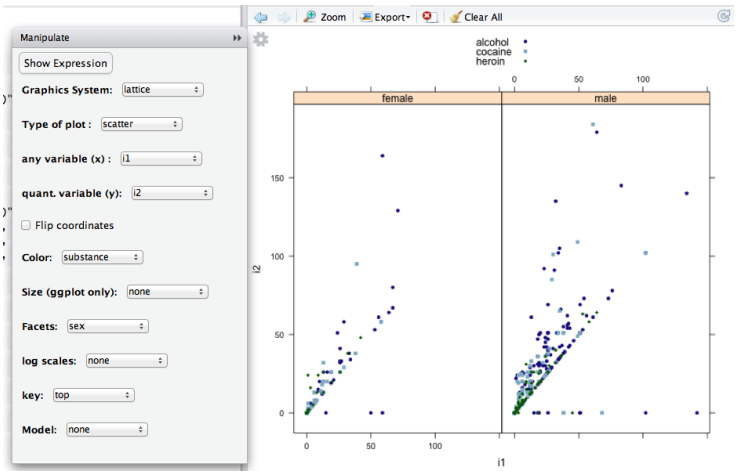
```
}
mNorm( mean=100, sd=10 )
```

9.2 mPlot()

El paquete `mosaic` proporciona la función `mPlot()` que permite a los usuarios crear una amplia variedad de gráficos usando lattice o ggplot2. Después, el código utilizado para generar este gráfico puede ser desplegado si se le pide. Esto facilita aprender estos comandos, permite a los usuarios hacer modificaciones más allá, las cuales no son posibles con la interfaz de manipulate y proporciona un mecanismo de copiar-y-pegar para poner estos gráficos en otros documentos.

Los gráficos disponibles vienen en 3 opciones; si son de una variable, de dos o un mapa.

```
# Estos son esencialmente gráficos de 2 variables
mPlot( HELPrct, "scatter" )
# Empieza con un gráfico de dispersión
mPlot( HELPrct, "boxplot" )
# Empieza con gráficos de barras
mPlot( HELPrct, "violin" )
# Empiea con gráficos de violin
# Estos son esencialmente de una variable
mPlot( HELPrct, "histogram" )
#Empieza con un histograma
mPlot( HELPrct, "density" )
#Empieza con un gráfico de densidad
mPlot( HELPrct, "frequency polygon" )
# Empieza con un polígono de frecuencia
```



9.3 *Shiny*

`shiny` es un paquete creado por el equipo de RStudio, para en sus palabras,

> [hacer] increíblemente fácil construir una aplicación web con R. Las ataduras reactivas.ªutomáticas entre las entradas y las salidas, y amplios widgets pre-fabricados hacen posible construir agradables, receptivas y poderosas aplicaciones con un esfuerzo mínimo.

Estas aplicaciones web pueden, obviamente, correr código de R para hacer los calculos y producir gráficos que aparecen en la página web.

El nivel de habilidades de código requeridas para crear esto se encuentra fuera del alcance de este libro, pero aquellos con un bagaje más amplio en programación pueden facilmente aprender las herramientos necesarias para hacer páginas web interactivas agradables a la vista. Más información sobre shiny y algunas aplicaciones de ejemplo están disponibles en `http://www.r.studio.com/shiny/`.

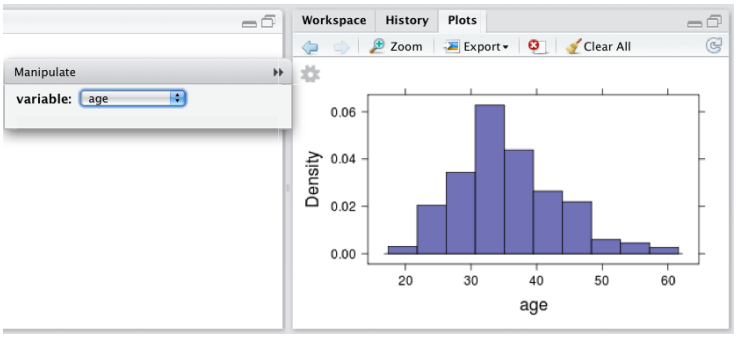
Ejercicios

9.1. El siguiente código hace un gráfico de dispersión con símbolos separados para cada sexo.

```
xyplot(cesd ~ age, data=HELPrct, groups=sex)
```

Construya un ejemplo de `manipulate` que le permita habilitar y deshabilitar el agrupamiento con una checkbox.

9.2. Construya un ejemplo de `manipulate` que use un seleccionador (picker()) para seleccionar de las variables que se tienen para hacer un gráfico. Aquí hay un ejemplo con un histograma



9.3. Diseñe su propia idea de demostración interactiva e implementela usando las herramientas de `manipulate` en RStudio

Bibliografía

[CR15] Angelo Canty and Brian Ripley. *boot: Bootstrap R (S-Plus) Functions*, 2015. R package version 1.3-17.

[DH97] A. C. Davison and D. V. Hinkley. *Bootstrap Methods and Their Applications*. Cambridge University Press, 1997.

[Fis25] R. A. Fisher. *Statistical Methods for Research Workers*. Oliver & Boyd, 1925.

[Fis70] R. A. Fisher. *Statistical Methods for Research Workers*. Oliver & Boyd, 14th edition, 1970.

[Hes15a] Tim Hesterberg. *resample: Resampling Functions*, 2015. R package version 0.4.

[Hes15b] Tim C. Hesterberg. What teachers should know about the bootstrap: Resampling in the undergraduate statistics curriculum. *The American Statistician*, 2015.

[LLM12] Robin H Lock, Patti Frazer Lock, and Kari Lock Morgan. *Statistics: Unlocking the Power of Data*. Wiley Global Education, 2012.

[NT10] D. Nolan and D. Temple Lang. Computing in the statistics curriculum. *The American Statistician*, 64(2):97–107, 2010.

[Salo1] D. Salsburg. *The Lady Tasting Tea: How statistics revolutionized science in the twentieth century*. W.H. Freeman, New York, 2001.

[TCC+15a] N. Tintle, B. Chance, G. Cobb, S. Roy, T. Swanson, and J. VanderStoep. Combating anti-statistical thinking using simulation-

based methods throughout the undergraduate curriculum. *The American Statistician*, 69(4), 2015.

[TCC+15b] Nathan Tintle, Beth Chance, George Cobb, Allan Rossman, Soma Roy, Todd Swanson, and Jill VanderStoep. *Introduction to Statistical Investigations*. Wiley Global Education, 2015.

[Wor14] ASA Undergraduate Guidelines Workgroup. 2014 curriculum guidelines for undergraduate programs in statistical science. Technical report, American Statistical Association, November 2014. `http://www.amstat.org/education/curriculumguidelines.cfm`.

Índice alfabético